Stanford University

AA222/CS361: Engineering Design Optimization

Spring 2021

Prof. Mykel J. Kochenderfer • Remote • email: *mykel@stanford.edu*

PROJECT 3 **Due date: May 21, 2021 (5pm Pacific)**

In this project, we want to optimize the trajectories of four vehicles over 30 time steps in two-dimensional Euclidean space.

Dynamics. The four vehicles are initialized on the vertices of a square with a side length of 100 m. We treat the vehicles as points. The velocities and accelerations of all four vehicles are initially zero. The x-position of vehicle i at time t is given by $x_t^{(i)}$. The notation for the y-position is similar. We can change the accelerations of the four vehicles once every second. The x-components of the velocity and acceleration for agent i at time t are $\dot{x}_t^{(i)}$ and $\ddot{x}_t^{(i)}$, respectively. The dynamics are as follows:

$$\dot{x}_t^{(i)} = \dot{x}_{t-1}^{(i)} + \ddot{x}_{t-1}^{(i)} \tag{1}$$

$$x_t^{(i)} = x_{t-1}^{(i)} + \dot{x}_{t-1}^{(i)} \tag{2}$$

for $t \in \{2, \ldots, 30\}$. The update for the y-components is similar. The initial conditions start with $t = 1$, and the dynamics allow us to propagate the positions, velocities, and accelerations to $t = 30$. We have the constraint that both the x and y components of acceleration fall within $[-1, 1]$.

Objective and constraints. Each vehicle must reach the opposing corner of its starting location at $t = 30$. There are no requirements about the final velocity. We want to minimize the sum of the squared acceleration magnitudes from $t = 1$ to $t = 30$. Let v_1 be the vehicle that starts at $(0, 0)$, v_2 at $(100, 0)$, v_3 at $(0, 100)$, and v_4 at $(100, 100)$. In order to avoid collisions between vehicles, each vehicle must be at the following locations at $t = 15$:

v_1 at $(50, 35)$ v_2 at $(65, 50)$ v_3 at $(35, 50)$ v_4 at $(50, 65)$.

Implementation. You may use any software packages you want to solve this problem. They should be capable of solving this problem to optimality. In Julia, you may consider using JuMP.jl with the Ipopt solver. Alternatively, you may use Convex.jl with ECOS or SCS solvers. Similar capabilities exist in Python with cvxpy and in Matlab with CVX.

What to submit. Submit your code and a plot of the resulting lateral positions over time all as a single PDF file on Gradescope. You must also include the objective value of your solution. You may brainstorm ideas with other students and compare outputs, but your code must be implemented by yourself.