HW #2 CS159

Due on 15 April 2021 at 9PM

Instructions

Please LaTeX your solutions using the attached template. Fill in each section with your answers and please do not change the order of sections and subsections. You need to submit both a PDF and your code. We provided a code template which can be found here:

https://github.com/1five9/1five9.github.io/raw/master/hw/hw_2.zip

Mac Users:
In the `hw_2.zip` folder, we provided an anaconda environment that contains all packages that you need for this homework. Anaconda is a distribution that aims to simplify package management and deployment. More info and details about installation can be found here. After installation, you can create a conda environment from the attached `py3_CS159.yaml` file the using the following command:

```
conda env create -f py3_CS159.yaml --name py3_CS159
```

Then, you can activate such environment using the following command:

```
conda activate py3_CS159
```

Windows and Linux Users:
Some Windows and Linux users were not able to install the conda environment on their machines. If you are having troubles setting up the conda environment you could install the following packages instead:

- casadi
- ipopt
- scipy
- numpy
- osqp
- pdp
- cvxopt
- dataclasses
- matplotlib

Note: If you just updated your Mac OS to Big Sur and you are having issues running conda from the terminal here the fix: https://github.com/conda/conda/issues/10361#issuecomment-744019125. After following the instruction, please update your xcode environment installation.

Note: Some users have encountered difficulties in installing the osqp solver. You can check if the osqp package is installed correctly by running the following file:

$$https://github.com/1five9/1five9.github.io/blob/master/hw/osqpTest.py$$

1 Problem 1

In this problem, you will design a nonlinear model predictive control for the following Euler discretized Dubins car model

$$x_{k+1} = f(x_k, u_k) = \begin{bmatrix} \bar{x}_k + v_k \cos(\theta_k) dt \\ \bar{y}_k + v_k \sin(\theta_k) dt \\ v_k + a_k dt \\ \theta_k + \delta_k dt \end{bmatrix}, \tag{1}$$

where dt is the discretization time and at time k the tuple $(\bar{x}_k, \bar{y}_k)$ represents the position of the car, v_k the velocity and θ_k the heading angle. The control inputs are the acceleration a_k and the yaw rate δ_k. In particular, you will design a nonlinear MPC to drive the system from a starting state $x_0 = [0, 0, 0, 0]^\top$ to a goal state $x_G = [10, 10, 0, \pi/2]^\top$.

In the attached folder `code/problem_1` you have the following files:

- `main.py`: This file loads the problem parameters and plots the closed-loop trajectories.
- `ftocp.py`: This files contains the FTOCP definition. You will modify this file to implement an SQP algorithm.
- `nlp.py`: This files contains the NLP definition. You should not modify this file. You will use the NLP object to benchmark the quality of your SQP approach.
- `utils.py`: This files contains the system definition. You should not modify this file.

1.1 Nonlinear MPC (1 points)

In this subsection, you will use a solver for Non-Linear Programs (NLP) to design a model predictive controller. We provided the NLP object which solves the following nonlinear finite time optimal control problem:

$$\begin{aligned} \min_{u_{t|t},\dots,u_{t+N-1|t}} \quad & \sum_{k=t}^{t+N-1} \left((x_{k|t} - x_G)^\top Q (x_{k|t} - x_G) + u_{k|t}^\top R u_{k|t} \right) + x_{t+N|t}^\top Q_f x_{t+N|t} \\ \text{subject to} \quad & x_{k+1|t} = f(x_{k|t}, u_{k|t}), \forall k \in \{t, \dots, t+N-1\}. \\ & x_{k|t} \in \mathcal{X}, u_{k|t} \in \mathcal{U}, \forall k \in \{t, \dots, t+N-1\}. \end{aligned} \tag{2}$$

where the state vector $x_{k|t} = [\bar{x}_{k|t}, \bar{y}_{k|t}, v_{k|t}, \theta_{k|t}]^\top$ and the input vector $u_{k|t} = [a_{k|t}, \delta_{k|t}]^\top$ are defined as in (1). In the above problem the state and input constraint sets are

$$\mathcal{X} = \{x \in \mathbb{R}^4 \mid ||x||_\infty \leq 15\} \text{ and } \mathcal{U} = \{u = [a, \delta]^\top \in \mathbb{R}^2 \mid |a| \leq 10, |\delta| \leq 0.5\}. \tag{3}$$

Use the `nlp` object to run a closed-loop simulation. In particular, in line #39 of the `main.py` file use the `solve` method of the `nlp` object to compute the control input at time t and store the predicted trajectory at time t.

Please turn in two figures showing the predicted trajectory on the $x-y$ plane at time $t=0$ and time $t=10$. You need to turn in also a figure that shows the predicted trajectories at all time steps $t \in \{0,\dots,\text{maxTime}\}$ and the closed-loop trajectory. Does the closed-loop trajectory overlap with the predicted trajectories? Is the result expected? Please comment.

1.2 Sequential Quadratic Programming (3 points)

In this subsection, you will approximate the solution to Problem (2) using a Sequential Quadratic Programming strategy.

- Rewrite the cost function from Problem (2) in the following form

$$[X_t^\top, U_t^\top]^\top H \begin{bmatrix} X_t \\ U_t \end{bmatrix} + q^\top \begin{bmatrix} X_t \\ U_t \end{bmatrix}$$

  where $X_t = [x_{t+1|t}^\top, \dots, x_{t+N|t}^\top]^\top \in \mathbb{R}^{nN}$, $U_t = [u_{t|t}^\top, \dots, u_{t+N-1|t}^\top]^\top \in \mathbb{R}^{dN}$ and the vector q is constructed using the goal vector $x_g = [10, 10, 0, \pi/2]^\top$.

- Modify the method `buildCost` in the file `ftocp.py`. In particular, you need to modify the linear cost vector `q`.

- Let

$$x_0 = x(t)$$
$$x_{k+1} = A(x_k^j, u_k^j)x_k + B(x_k^j, u_k^j)u_k^j + C(x_k^j, u_k^j), \quad \forall k \in \{0,\dots,N-1\}$$

  be the linearization of the discrete time system (1) around the input and state sequences $U_j^0 = [u_0^j, \dots, u_{N-1}^j]$ and $X_0^j = [x_0^j, \dots, x_N^j]$, where the matrices $A(x_k^j, u_k^j), B(x_k^j, u_k^j)$ and $C(x_k^j, u_k^j)$ are defined as in Lecture #2. Rewrite the above linearized system dynamics in the following form

$$G_x X_t + G_u U_t = Ex(t) + C$$

- Modify the method `buildEqConstr` in the file `ftocp.py`. In particular, you need to fill in the entry of the matrices `Gx` and `Gu`.

- Uncomment lines #69-97 of the `main.py` and run this file to approximate the solution to Problem (2) using one iteration of SQP and to plot the results.

- Uncomment lines #99-116 of the `main.py` file to approximate the solution to Problem (2) using two iteration of SQP and to plot the results. Notice that we used the predicted sequence of inputs to update the variable `uGuess`, which is used to compute the linearization points for the SQP iteration.

Please turn in a snippet of code for the methods `buildCost` and `buildEqConstr` from the file `ftocp.py`. Please also turn in two figures that compare the trajectories computed by the SQP algorithms and the solution to the NLP. Does the SQP algorithm well-approximate the solution from the NLP? How about the computational time? Which method is faster (consider both the linearization time and the solver time for the SQP method)?

1.3 Nonlinear MPC using an SQP Approach (2 points)

In this subsection, you will use an SQP approach to design a nonlinear MPC. Uncomment lines 120–168 and modify line #124 in the file `main.py` to compute the control action `ut` (*Hint: Use the* `ftocpSQP` *object*).

Next, we will compare two approaches to compute an initial guess used to run one iteration of the SQP algorithm.

Approach 1:
Use a vector of zeros as warm-start. In particular, modify the method `uGuessUpdate` in the file `ftocp.py`. The list `uGuess` contains the input sequence $U_t^0 = [u_t^0, \dots, u_{t+N-1}^0]$ which is used to compute the sequence of states $X_t^0 = [x_t^0, \dots, x_{t+N}^0]$. Set each $u_t^0 = [0, 0]^\top$ and run a closed-loop simulation.

Approach 2:
Use the optimal solution predicted at the previous time step as warm-start. In particular, modify the method `uGuessUpdate` in the file `ftocp.py`. The list `uGuess` which contains the input sequence $U_t^0 = [u_t^0, \dots, u_{t+N-1}^0]$ which is used to compute the sequence of states $X_t^0 = [x_t^0, \dots, x_{t+N}^0]$. Set each $u_k^0 = u_{k|t-1}^*$ for all $k \in \{t, \dots, t+N-2\}$ and $u_{t+N-1}^0 = u_{t+N-2|t-1}^*$. Notice that the optimal inputs computed at the previous time step $u_{k|t-1}^*$ are stored in the matrix `uPred`. Finally, run a closed-loop simulation.

Please turn in the plots generated for the two approaches. What do you notice? How does the closed-loop trajectories resulting from the two approaches compare with the one computed using the NLP to solve the nonlinear MPC? Compare the computation time associated with the SQP approach and NLP solver, which one is faster?

2 Problem 2

In this problem, you will design a model predictive controller to regulate a linear time invariant system to the origin. This problem is based on the section "MPC Closed-loop Properties" of the slides set used in Lecture #4. Please review the slides before starting this problem.

We consider the following discrete time system

$$x(k+1) = \begin{bmatrix} 1.2 & 1 \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k).$$

subject to the following state and input constraints

$$\mathcal{U} = \{u \in \mathbb{R}^d \mid ||u||_\infty \leq 1\} \text{ and } \mathcal{X} = \{x \in \mathbb{R}^n \mid ||x||_\infty \leq 15\}.$$

2.1 MPC Design (6 points)

In this subsection, you will design an MPC following two different approaches. In both cases you should use the weights $Q = I \in \mathbb{R}^n$ and $R = 1$.

Approach 1:

1. Uncomment line #35–#70 of the `main.py` file. Set the terminal set $\mathcal{X}_f = \{x \mid F_f x \leq b_f\} = \{0\}$, and a terminal weight P so that the closed-loop system is stable. Explain why your choice of the terminal weight P guarantees stability. (*Hint: review the slides from Lecture #3. Does the choice of P matter when $\mathcal{X}_f = \{0\}$?*)
2. Set $x(0) = [2, -1]^\top$ in line #16 and run a closed-loop simulation for $N = 3$. Plot the closed-loop trajectories and the open-loop predicted trajectories.
3. Set $x(0) = [2, -1]^\top$ in line #16 and run closed-loop simulations for different values of N. Plot the closed-loop trajectories and the open-loop predicted trajectories. What do you notice? How does the mismatch between the closed-loop trajectory and the open-loop predicted trajectory change as a function on N?

Approach 2:

1. Uncomment line #72–#122 of the `main.py` file. Pick terminal weight P to be solution to the discrete-time algebraic Riccati equation (DARE), which is computed by the `dlqr` function provided in the attached code. Then, set $\mathcal{X}_f$ to be the maximal invariant set $\mathcal{O}_\infty$ defined for the closed-loop system $x(k+1) = (A - BK)x(k) = A_{\text{cl}}x(k)$. We have provided the code to compute all these components. Please explain why your choice of the terminal components guarantees stability and recursive feasibility. (*Hint: review the slides from Lecture #4*)
2. Set $x(0) = [2, -1]^\top$ in line #16 and run a closed-loop simulation for $N = 3$. Plot the closed-loop trajectories and the open-loop predicted trajectories.
3. Set $x(0) = [2, -1]^\top$ in line #16 and run closed-loop simulations for different values of N. Plot the closed-loop trajectories and the open-loop predicted trajectories. What do you notice? How does the mismatch between the closed-loop trajectory and the open-loop predicted trajectory change as a function on N?

Please turn in the plots and your answers to each subquestion. Please follow the LaTeX template that we have provided.

2.2 MPC Region of attraction (4 points)

In this subsection, you will compare the region of attraction associated with the model predictive controller designed using **Approach 1** and **Approach 2**.

- Design an MPC using the terminal components from **Approach 1**. Set $x(0) = [13, -5.5]^\top$ in line #16 of the `main.py` file and run a closed-loop simulation for $N = 3$. Is the MPC problem at $t = 0$ feasible?
- Design an MPC using the terminal components from **Approach 2**. Set $x(0) = [13, -5.5]^\top$ in line #16 of the `main.py` file and run a closed-loop simulation for $N = 3$. Is the MPC problem at $t = 0$ feasible? If so plot the closed-loop trajectory.
- In the `main.py` file uncomment lines #123-#149 to plot the region of attraction $\mathcal{K}_3(\mathcal{X}_f)$ associated with the MPC designed using the terminal constraints $\mathcal{X}_f = \{0\}$ and $\mathcal{X}_f = \mathcal{O}_\infty$. In the same figure, plot the initial condition $x(0) = [2, -1]^\top$ and $x(0) = [13, -5.5]^\top$ used in Section 2.1 and Section 2.2. What do you notice? Does this plot explain the feasiblity of the MPC problem at time $t = 0$ for the initial conditions from Section 2.1 and Section 2.2?