AE4803 - Robotic Systems and Autonomy.
Class Project - Fall 2022.

Instructor: Vincent Pacelli

Due on Friday, December 9, 2022

The goal of this project is to apply Differential Dynamic Programming to a quadrotor system provided below. While not mandatory, you may find it helpful to modify your algorithm to include numerical methods known as regularization and line search. You will also use it in conjunction with a technique called barrier states to avoid obstacles. Details on the numerical techniques barrier states are provided later in the document.

You can work on this project either individually or as a group of maximum 4 students. Submissions should include a PDF document with all the plots requested below and the Matlab code. Each team should have one submission with the members names on the first page and indicating the role of each member in a brief sentence.

1. **(50 Points) Apply DDP on the dynamics of the quadrotor**. The task for the quadrotor is to starts from initial state and reach a terminal state **with zero velocity**. The dynamics of the quadrotor are provided in a following section. The initial position states $(x(0), y(0), z(0)) = (-3, -2, -1)$ and the terminal states are $(x(t_f), y(t_f), z(t_f)) = (5, 3, 2)$ in where $t_f = 8$ seconds. Use a time discretization of with $\Delta t = 0.01$. You are free to choose the tuning parameters of the cost function and your DDP algorithm.

   Provide plots of all the states and control inputs of the quadrotor versus time as well as a 3D plot of the quadrotor's position trajectory. Make sure to add appropriate legends and titles.

2. **(30 Points) Recursive Model Predictive Control fashion**. Repeat the first part but with Recursive MPC. A refresher on Recursive MPC is provided later in the document.

3. (**20 points**) **Avoiding obstacles using (Discrete) Barrier States with DDP**. Avoid the obstacles defined by the safety functions

$$h_1(\mathbf{x}) = (x - 2.2)^2 + (q_y - 2.2)^2 + (z - 1)^2 - 1 \tag{1}$$
$$h_2(\mathbf{x}) = x^2 + (y + 0.2)^2 + z^2 - 1 \tag{2}$$
$$h_3(\mathbf{x}) = (x - 3)^2 + y^2 + (z - 0.5)^2 - 1 \tag{3}$$

4. (**Bonus 20 points**) **Implement DDP on the Robotarium (Link)**. Use a car model (the unicycle model) to derive a robotarium robot to move from an initial point to a final point. You are free to choose the initial states, final states, time horizon, descretization constant, and cost. You should perform two experiments, one using the feed-forward control only and one using the feed-forward and feedback solutions. Discuss the difference between the two

experiments and provide the robotarium videos with the planned trajectory plotted on the robotarium floor.

Quadrotor System

These are dynamics of a quadrotor system, in a "X" configuration. The body coordinate frame is aligned such that the X-direction is directed halfway between the first and second rotor, the Y-direction is halfway between the first and fourth rotor (if the X-direction is forward, the Y-direction is left), and the Z-direction up through the body (as is consistent with the right hand rule). There is no drag or motor dynamics considered in this model. Inertial coordinates are in NWU (north, west, up). The transformation from the inertial coordinates to the body coordinates is defined by an Euler 1-2-3 rotation $R(\phi, \theta, \psi)^{\mathrm{T}} = R_1(\phi)R_2(\theta)R_3(\psi)$.

The quadrotor model consists of the following variables and parameters:

- The thrust u_i due to rotor i.
- Rotation matrix from body to inertial coordinates: $R(\phi, \theta, \psi)$
- Quadrotor mass: $m = 0.5$ kg
- Quadrotor inertia matrix $I = \mathrm{diag}(I_{xx}, I_{yy}, I_{zz})$ where $I_{xx} = I_{yy} = 0.0032$ kgm^2 and $I_{zz} = 0.0055$ kgm^2
- Rotor torque constant: $k_t = 0.01691/\mathrm{m}$
- Rotor moment arm: $l = 0.17\mathrm{m}$
- Gravity constant: $g = 9.81$ m/s^2
- Body roll, pitch, and yaw rates: p, q, r
- Euler angle roll, pitch, and yaw: ϕ, θ, ψ

The system state is defined as:

$$\mathbf{x} = (x,\ y,\ z,\ \dot{x},\ \dot{y},\ \dot{z},\ \phi,\ \theta,\ \psi,\ p,\ q,\ r) \tag{4}$$

The controls are the forces of the rotors:

$$\mathbf{u} = (u_1,\ u_2,\ u_3,\ u_4) \tag{5}$$

Translational equations of motion:

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = R \begin{bmatrix} 0 \\ 0 \\ u_1 + u_2 + u_3 + u_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \tag{6}$$

$$R = \begin{bmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ \sin(\theta)\sin(\phi)\cos(\psi) - \cos(\phi)\sin(\psi) & \sin(\theta)\sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi) & \sin(\phi)\cos(\theta) \\ \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi) & \sin(\theta)\sin(\psi)\cos(\phi) - \sin(\phi)\sin(\psi) & \cos(\theta)\cos(\phi) \end{bmatrix}$$

Rotational Equations of motion:

$$I_{xx}\dot{p} = \frac{\sqrt{2}}{2}(u_1 + u_3 - u_2 - u_4)l - (I_{zz} - I_{yy})qr \tag{7}$$

$$I_{yy}\dot{q} = \frac{\sqrt{2}}{2}(u_3 + u_4 - u_1 - u_2)l + (I_{zz} - I_{xx})pr \tag{8}$$

$$I_{zz}\dot{r} = k_t(u_1 + u_4 - u_2 - u_3) \tag{9}$$

Finally note that body rates are NOT equivalent to euler rates i.e $(\dot{p} \neq \dot{\phi})$, etc. The relationship between them is as follows for our selection of attitude representation.[1]

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \tan(\theta)\sin(\phi) & \tan(\theta)\cos(\phi) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \frac{\sin(\phi)}{\cos(\theta)} & \frac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{10}$$

[1] This equation is also where the infamous singularity of Euler angle representations appears, also known as gimbal locking!

Recursive Model Predictive Control

Recursive model predictive control is a way of providing feedback control via trajectory optimization algorithms. Consider the following optimal control problem:

$$\underset{u_0,\ldots,u_{t_f-1}}{\text{minimize}} \quad \sum_{t=0}^{t_f} l_t(x_t, u_t) + l_{t_f}(x_{t_f}) \quad \text{subject to} \quad x_{t+1} = f(x_t, u_t), x_0 = x_{\text{ic}}. \tag{11}$$

Solving this optimal control problem gives a single trajectory of control inputs that drives the robot in an optimal manner from the given x_{ic}. However, when executed, the robot will not track the planned trajectory perfectly due to uncertainty about the actual initial state, imperfections in the mathematical model, etc.

The iLQR and DDP algorithms we discussed provide a manner to ameliorate this problem for small disturbances: in addition to returning a nominal trajectory of inputs, $\bar{u}_0, \ldots, \bar{u}_{t_f-1}$, they provide a linear feedback controller that tracks the corresponding state trajectory $\bar{x}_0, \ldots, \bar{x}_{t_f}$. However, the linear controller will only work well as long as the disturbances do not drive the robot far from the planned trajectory.

A better approach is available at the cost of some additional computation: at every time step, plan a new trajectory using the current state (estimate) $\hat{x}_t$ as x_{ic}. Pseudocode for this method is provided in Algorithm 1. The function $\texttt{MPC}(x_{\text{ic}}, \tau)$ solves the above optimal control problem with $t_f = \tau$ and returns the optimal control sequence $\bar{u}_0, \ldots, \bar{u}_\tau$. Note that τ does not need to match t_f for the optimal control problem and may be longer or shorter. Since we are replanning at every time step, only the control input $\bar{u}_0$ is used.

Finally, to reduce up the computation time, a technique known as "warm starting" may be used. Recall that your DDP algorithm takes in an initial guess for the feedback controller. The idea is that, for sufficiently small time steps and disturbances, the solution returned by DDP should not change much from one time step to another. Thus, you can start DDP with the previous solution, and improve it using a small number of DDP iterations for the current time step. Implementing warm starting is optional, but if your code takes a long time to run, you may use this method to speed things up.

Algorithm 1 Recursive MPC

for $t = 0, \ldots, t_f$ **do**
  Measure state $\hat{x}_t$.
  $\bar{u}_0, \ldots, \bar{u}_\tau \leftarrow \texttt{MPC}(\hat{x}_t, \tau)$
  Apply control input $\bar{u}_0$
end for

Obstacle Avoidance via Barrier States

Let $h(x)$ define the boundary of the subset of the state space $\mathbf{S} \subset \mathbf{X}$, which is *safe* (does not contain an obstacle) i.e.:

$$\mathbf{S} := \{x \in \mathbf{X} \mid h(x) \geq 0\}. \tag{12}$$

Denote the boundary and interior of $\mathbf{S}$ respectively as:

$$\text{bd } \mathbf{S} := \{x \in \mathbf{X} \mid h(x) = 0\} \qquad \text{int } \mathbf{S} := \{x \in \mathbf{X} \mid h(x) > 0\}. \tag{13}$$

Note that if there are multiple obstacle sets $\mathbf{O}_i := \{x \in \mathbf{X} \mid h_i(x) \leq 0\}$, then $h(x)$ can be constructed as $h(x) = \prod_i h_i(x)$. Next, select a barrier function $B_h(x)$ satisfying for any sequence $(x_k)_{k=0}^{\inf}$ such that $x_k \in \text{int } \mathbf{S}$:

$$\lim_{k\to\infty} x_k \in \text{bd } \mathbf{S} \implies \lim_{k\to\infty} B_h(x_k) = \infty. \tag{14}$$

Informally, this requirement states that the barrier function $B_h(x)$ goes to infinity on the boundary of the safe set. Popular choices of barrier functions for a given $h(x)$ include:

1. $B_h(x) = -\log(h(x))$
2. $B_h(x) = \frac{1}{h(x)}$
3. $B_h(x) = -\log\left(\frac{h(x)}{h(x)+1}\right)$

For a target equilibrium state, x_{eq}, define $w_t = B_h(x_t) - B_h(x_{\text{eq}})$. The value w_t is known as the *barrier state* because it evolves in time in correspondence with the system dynamics $x_{t+1} = f(x_t, u_t)$ in the following manner:

$$w_{t+1} = B_h(f(x_t, u_t)) - B_h(x_{\text{eq}}). \tag{15}$$

Thus, for a given dynamical system, safe set $\mathbf{S}$, and barrier function $B_h(x)$, we can define an augmented dynamical system:

$$\underbrace{\begin{bmatrix} x_{t+1} \\ w_{t+1} \end{bmatrix}}_{z_{t+1}} = \underbrace{\begin{bmatrix} f(x_t, u_t) \\ B_h(f(x_t, u_t)) - B_h(x_{\text{eq}}) \end{bmatrix}}_{g(z_t, u_t)}. \tag{16}$$

Then, through choice of a cost functions of the form,

$$l(z, u) = c(x, u) + d(w), \qquad l_{t_f}(z, u) = c_{t_f}(x) + d_{t_f}(w) \tag{17}$$

leads to a provably safe trajectory when optimized using, e.g., DDP. That is, DDP is used to solve the optimization problem:

$$\underset{u_0,\ldots,u_{t_f}}{\text{minimize}} \sum_{t=0}^{t_f-1} l(z_t, u_t) + l_{t_f}(z_{t_f}) \quad \text{subject to} \quad z_{t+1} = g(z_t, u_t). \tag{18}$$

A common choice for $d(w)$ or $d_{t_f}(w)$ is $\alpha\|w\|^2$ for some $\alpha > 0$.

Numerical Techniques for DDP

The quadrotor system is more complicated than the Dubin's Car system from the previous homework. As such, it may be difficult to find good cost functions for which DDP converges properly while also achieving the goal. Two modifications to DDP that can improve convergence are line search and regularization.

Line Search

Line search simply refers to searching along the line of a descent direction to find a set of decision variables (i.e., control inputs) that best reduces the objective. This prevents large gradients from causing the optimization algorithm to jump over local minima. For DDP, this takes the form of evaluating the control inputs,

$$\tilde{u}_t = \bar{u}_t + \epsilon_i k_t + K_t(\tilde{x}_t - \bar{x}_t) \tag{19}$$

for a sequence of decreasing step sizes $\epsilon_1, \dots \epsilon_M \in [0, 1]$ where $\epsilon_1 = 1$ and $\epsilon_M = 0$. For ϵ_1, the result is the normal DDP algorithm. For ϵ_M, the algorithm will not change the trajectory (i.e., DDP has converged). The IADP algorithm with line search will look like the following:

Algorithm 2 Iterative Approximate Dynamic Programming (IADP) with Line Search

function IADP($x_0, K^{(0)}_{0:t_f-1}, k^{(0)}_{0:t_f-1}, N$)
  $\bar{x}^{(0)}_{0:t_f}, \bar{u}^{(0)}_{0:t_f-1}, J^{(0)} \leftarrow$ ForwardPass($x_0; K^{(0)}_{0:t_f-1}, k^{(0)}_{0:t_f-1}$)
  for $i = 0, \dots, N$ **do**
    $K_{0:t_f-1}, k_{0:t_f-1} \leftarrow$ BackwardPass($\bar{x}^{(i)}_{0:t_f}, \bar{u}^{(i)}_{0:t_f-1}$)
    for $j = 1, \dots, M$ **do**
      $\bar{x}_{0:t_f}, \bar{u}_{0:t_f-1}, J \leftarrow$ ForwardPass($x_0; K_{0:t_f-1}, \epsilon_j k_{0:t_f-1}, \bar{x}^{(i)}, \bar{u}^{(i)}$)
      if $J < J^{(i)}$ **then**
        $\bar{x}^{(i)}_{0:t_f}, \bar{u}^{(i)}_{0:t_f-1}, J^{(i)} \leftarrow \bar{x}_{0:t_f}, \bar{u}_{0:t_f-1}, J$
        break
      end if
    end for
  end for
  return $K^{(N)}_{0:t_f-1}, k^{(N)}_{0:t_f-1}, (J^{(0)}, \dots J^{(N)})$
end function

Regularization

Recall that DDP works by taking a second-order Taylor expansion and using the FONCO to minimize the resulting quadratic optimization problem. However, the FONCO is only a necessary

condition — whether or it yields a minimum depends on whether the quadratic is the right shape! Regularization is a process where you automatically tweak the cost function inside the backward pass to ensure that the quadratic problem always has a minimum.

There are two ways you can go about this process. You are free to use either method as you see fit. In both cases, the goal is to ensure that $\partial_u^2 \tilde{H}_t$ is positive-definite, which guarantees that the Taylor expansion will have a minimum. In one dimension, you can consider a polynomial $ax^2 + bx + c$. This polynomial only has a minimum when $a > 0$. In the multidimensional case we are working with, the positive definite criteria corresponds to making sure that the smallest eigenvalue of $\partial_u^2 \tilde{H}_t$, i.e., $\lambda_{\min}(\partial_u^2 \tilde{H}_t)$ is larger than some positive value ξ. The reason that ξ is positive is that, even though theoretically we just need to check that $\lambda_{\min}(\partial_u^2 \tilde{H}_t) > 0$, matrix inversion algorithms perform poorly when eigenvalues are very small, and the resulting numerical error can destabilize your DDP algorithm.

The first method is to simply add μI to $\partial_u^2 \tilde{H}_t$ until the smallest eigenvalue is greater than ξ, i.e., at each time step in the backward pass, replace H_{uu} with

$$\partial_u^2 \tilde{H}_t = \mu I + \partial_u^2 H_t. \tag{20}$$

You should then check if $\lambda_{\min}(\partial_u^2 \tilde{H}_t) > \xi$ and, if the condition is not satisfied, increase μ. This modification is equivalent to adding the term $\mu u_t^{\mathrm{T}} u$ to the cost function.

The second method is slightly more complicated but also even more numerically stable. It penalizes deviations from the state trajectory about which the Taylor expansion occurs. The result modifies both $H_u u$ and H_{ux}.

$$[\partial_u^2 \tilde{H}_t]_{ij} = \partial_{u_i} \partial_{u_j} c + \partial_{u_i} f^{\mathrm{T}} (\partial_x^2 V_{t+1} + \mu I) \cdot \partial_{u_j} f + \partial_x V_{t+1}^{\mathrm{T}} \partial_{u_i} \partial_{u_j} f, \tag{21}$$

$$[\partial_u \partial_x \tilde{H}_t]_{ij} = \partial_{u_i} \partial_{x_j} c + \partial_{u_i} f^{\mathrm{T}} (\partial_x^2 V_{t+1} + \mu I) \cdot \partial_{x_j} f + \partial_x V_{t+1}^{\mathrm{T}} \partial_{u_i} \partial_{x_j} f. \tag{22}$$

To summarize, at each time t of the backward pass, regularization is performed by:

1. Setting $\mu = 0$.
2. Compute $\partial_u^2 \tilde{H}_t$ and, if using the second method, $\partial_u \partial_x \tilde{H}_t$.
3. Compute $\lambda_{\min}(\partial_u^2 \tilde{H}_t)$.
4. If $\lambda_{\min}(\partial_u^2 \tilde{H}_t) < \xi$, increase μ and repeat from Step 2.
5. Otherwise, proceed with computing the rest of the backward pass with $\partial_u^2 \tilde{H}_t$ instead of $\partial_u^2 H_t$ and, if modified, $\partial_u \partial_x \tilde{H}_t$ instead of $\partial_u \partial_x H_t$