02465 Project: Part 2

Tue Herlau
tuhe@dtu.dk

December 4, 2024

Formalities

- The deadline for this report is **Thursday 3rd April, 2025** before 23:59.
- Submission of reports happen on DTU learn
- You can work in groups of 1, 2 or 3 students (but not 4)
- You can work in groups of 1, 2 or 3 students (but not 4)
- Collaboration policy: It is not allowed to collaborate with other groups on this project, except for discussing the text of the project with teachers and students enrolled on the course this semester. It is not allowed to communicate (or make available) solutions or parts of solutions to the project to other people. It is not allowed to use solutions from previous years, or solutions found on the internet or elsewhere.
- You can use code from the *exercises* when you solve the project, for instance the dynamical programming algorithm. The exercises may be solved with help from teachers or fellow students. However, you are not allowed to copy or share exercise code directly between groups or make solutions publicly available. This is to ensure there is no accidental copying of projects.
- Your overall evaluation will be based on your written answers and your UNITGRADE score. They will be weighted based on an assessment of the required work.

Preparing the hand-in:

Hand in these three files (please do not hand in a `.zip` file as this confuses DTU learn):

A `.tex` file with your written answers: Prepared this by modifying the template in `irlc/project2/Latex/02465project2_handin.tex` . Simply write your answers where it says **YOUR SOLUTION HERE**. I recommend keeping the layout as it is.

A `.pdf` file corresponding to this `.tex` file

A `.token` file containing your python-solutions: Generate this file by running the script `irlc/project2/project2_grade.py` . It is very important you do not modify this file.

Contribution table

DTUs exam rules require that each students contribution to the report is clearly specified. Therefore, for each element in the report, specify which student was responsible for it in the table in the template. **A report must contain this documentation to be accepted**. The responsibility assignment must be individualized. This means:

- For reports made by 3 students: Each section must have a student who is 40% or more responsible.
- For reports made by 2 students: Each section must have a student who is 60% or more responsible.

This is an external requirement. Ask me if you have any questions.

Code hand-in:

- Please keep the structure of the `irlc`-folder. All of your code which is specific to this report should be in the `irlc/project2/` directory. Solutions which use code outside the `irlc` folder cannot be verified and therefore cannot be evaluated. You can (of course) call, re-use or re-purpose any exercise code, including my solutions.
- If you wish to use additional third-party libraries please discuss them with me first to ensure you are on the right track.
- Breaking or tampering with the UNITGRADE framework, for instance by reporting a false number of points or making your solution unverifiable, is potentially cheating. Code build by reverse engineering specific tests and simply returning the values which makes them pass will not get credit and may also need to be treated as a cheating-attempt. Code which is obfuscated to the point of being unreadable cannot be evaluated.
- That asides, this is not a programming course: Strange, long, undocumented, or downright disturbing solutions will be evaluated simply based on whether they work or not.

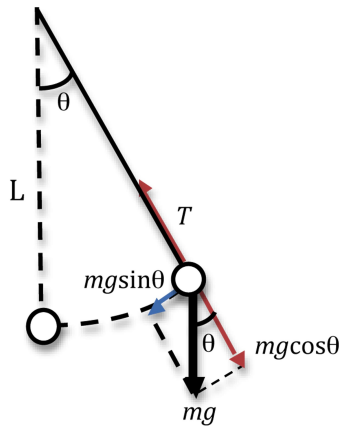


Figure 1: Master Yodas pendulum

1 Master Yodas pendulum (`yoda.py`)

Yodas pendulum hang on a string of length L, at an angle to vertical of θ and with mass m. As part of Padawan training, Yoda will start the pendulum in position $\theta = 1$, release it, and the Padewans will then meditate on its movement when it is affected by THE FORCE (see fig. 1). Using Newtons laws we can derive the equations of motions to be[1]

$$mL^2 \frac{d^2\theta(t)}{dt^2} = u - mg\sin(\theta(t))L \tag{1}$$

If we use that $\sin(\theta) \approx \theta$ when θ is small we can write this as

$$\ddot{\theta} = -\frac{g}{L}\theta + \frac{1}{mL^2}u \tag{2}$$

This equation will be our starting point and can from now on you can consider eq. (2) as the true equation of motion for θ.

Problem 1 *Formulate Yodas pendulum as a linear problem*

In the absence of a cost-function, Yodas pendulum can be written as a linear system of the form

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} \tag{3}$$

Where $\boldsymbol{x} = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$.

- Define A and B below in terms of g, L and m in the LaTeX document.
- Implement the function `get_A_B(g, L, m)` which return the two matrices A and B as numpy `ndarray`

[1] See `https://www.acs.psu.edu/drussell/Demos/Pendulum/Pendulum.html`

Info:

- The function accepts a value of g, L and m as inputs.
- The two answers should be the same.

Answer:

$$A = \begin{bmatrix} \cdots \end{bmatrix} \tag{4}$$

$$B = \begin{bmatrix} \cdots \end{bmatrix} \tag{5}$$

YOUR SOLUTION HERE

1.1 Yodas pendulum and discretization

We will now investigate what happens with the pendulum, described in eq. (3), when no external force is applied to it ($B = \boldsymbol{u} = 0$) and different types of discretization schemes, Euler and Exponential Integration (EI), are applied. The point of the exercise will be to show mathematically that one discretization scheme is numerically stable and the other is not.

Problem 2 *State at a later time*

Assume the system has been discretized using a time step of Δ to give states $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.

1. Prove that when Euler discretization or exponential discretization is used to discretize the system the final state can be written as:

$$\text{(Euler discretization):} \quad \boldsymbol{x}_N = \tilde{A}_0^N \boldsymbol{x}_0 \tag{6}$$

$$\text{(Exponential discretization):} \quad \boldsymbol{x}_N = A_0^N \boldsymbol{x}_0 \tag{7}$$

2. Provide an (analytical) expression for the two matrices $\tilde{A}_0$ and A_0 (i.e., express them in terms of known quantities such as L, g and Δ)
3. Implement the functions `A_euler` and `A_ei` to compute both matrices in python.

Info:

- As to the first part, start with $N = 1$ and $N = 2$ to get the general idea.
- As to the form of A_0 and $\tilde{A}_0$ use the lecture notes. For the exponential integration, you need to use the matrix exponential. Code which computes it is imported and described at the top of the file.

Answer:

To solve the first part, we can write $\boldsymbol{x}_N = \begin{bmatrix} \cdots \end{bmatrix}$

As for the second part we get:

$$\tilde{A}_0 = \begin{bmatrix} \cdots \end{bmatrix}, \quad A_0 = \begin{bmatrix} \cdots \end{bmatrix} \tag{8}$$

YOUR SOLUTION HERE

Problem 3 *State at a later time II*

According to the previous problem, we can in fact write

$$\text{(Euler discretization):} \quad \boldsymbol{x}_N = \tilde{M}\boldsymbol{x}_0 \tag{9}$$

$$\text{(Exponential discretization):} \quad \boldsymbol{x}_N = M\boldsymbol{x}_0 \tag{10}$$

for two matrices M and $\tilde{M}$. Implement the functions to compute both matrices in python as `M_euler` and `M_ei`

Info:

- The intention with this problem is that once you have an explicit numpy function to compute M and $\tilde{M}$, then when you are later asked to compute e.g. Eigenvalues analytically you can check your result by numerically computing the Eigenvalues of M and $\tilde{M}$. In my experience this can speed up derivations a great deal because it will quickly catch mistakes.
- The imports at the top of the file contain the potentially useful numpy functions.

In the next question, we will just focus on the Euler discretization matrices $\tilde{A}_0$ and $\tilde{M}$ and the relationship between the four Eigenvalues of these two matrices. To do so, we will use the following special case of the Eigendecomposition: Let A be a 2×2 matrix with two linearly independent unit-length Eigenvectors satisfying $A\boldsymbol{v}_1 = \lambda_1\boldsymbol{v}_1$ and $A\boldsymbol{v}_2 = \lambda_2\boldsymbol{v}_2$. Assume we form the 2×2 matrices $Q = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$. It then holds that

$$A = Q\Sigma Q^{-1}. \tag{11}$$

The reverse is also true: **If a Matrix has this representation, the Eigenvalues are the diagonal of** Σ.

Problem 4 *Eigenvalues and powers*

Use the Eigendecomposition to show there exists a simple relationship between the eigenvalues of $\tilde{A}_0$ and the eigenvalues of $\tilde{M}$ involving N.

Info:

- If stuck, always start with $N = 1$ and $N = 2$.
- Use that the Eigenvectors are orthonormal.
- Test your analytical answer numerically.
- The result turns out to be very simple.

Answer:
Assume λ_1, λ_2 are the eigenvalues ... then the Eigenvalues of M is ... similarly for $\tilde{M}$ ... **YOUR SOLUTION HERE**

Problem 5 *Analytical expression of Eigenvalues using Euler discretization*

Derive an analytical expression of the Eigenvalues λ_1 and λ_2 of $\tilde{A}_0$.

Info:

- Don't be scared if the result looks **complex**.
- This is really just a Mat1 problem.

Answer:
... we get a characteristic polynomial of ... and therefore it follows from Mat1 that the two Eigenvalues are ... **YOUR SOLUTION HERE**

Background: Let A be a matrix. The matrix norm is defined as $\|A\| = \max_{\|\boldsymbol{x}\|=1} \|A\boldsymbol{x}\|$. It holds that if λ_1, λ_2 are the two Eigenvalues of A that

$$\|A\| = \max\{|\lambda_1|, |\lambda_2|\}. \tag{12}$$

where $|\lambda|$ is the absolute value and λ may be a complex number.

Problem 6 *Bound using Euler discretization*

Use the Matrix norm, and the previous problem to derive a tight upper bound for $\|\boldsymbol{x}_N\|$ assuming $\|\boldsymbol{x}_0\| = 1$ and the problem has been discretized using Euler integration. Implement the bound as the function `xN_bound_euler(g, L, Delta, N)`.

Info:

- Upper-bound means you have to find a function F such that $\|\boldsymbol{x}_N\| \leq F(g, \Delta, L, N)$ for all $\boldsymbol{x}_0$ so that $\|\boldsymbol{x}_0\| = 1$. That the bound is tight means you cannot find a function with smaller values.
- You should use the property of the Matrix norm listed above in conjunction with your answers to the previous questions
- The solution can be written as a single line; start with the given expression and try to use the things you have seen so far one at a time.

Answer:
Using Euler discretization we get the upper bound:

$$\|\boldsymbol{x}_N\| \leq \cdots$$

YOUR SOLUTION HERE

Problem 7 *Matrix norm of Exponential discretization (harder)*

Repeat the same steps you did before to derive a tight upper-bound of $\|\boldsymbol{x}_N\|$ assuming $\|\boldsymbol{x}_0\| = 1$ and that we are using exponential integration. Use the hints below. When done, implement the function as `xN_bound_ei(g, L, Delta, N)` and check your result. **N.b. the tight upper bound must be simplified to a simple expression that does not involve complex numbers.** .

Answer:
Using exponential discretization we get an upper bound of:

$$\|\boldsymbol{x}_N\| \leq \cdots$$

YOUR SOLUTION HERE

Info:

- Most of the problem is similar to the previous one. The part which is different has to do with computing eigenvalues. Remember that you can compute eigenvalues numerically to get ideas and check you are on the right track.
- If you follow the same steps as before, you will see that your solution way back in eq. (7) means you will have a matrix of the form e^A to deal with, where A is *some* 2×2 matrix you *can* compute the Eigenvalues of. Compute these eigenvalues first.
- Note you can always write $A = Q\Sigma Q^{-1}$ using the Eigendecomposition; you don't need to know what Q is, but you can find Σ
- To use this to get the eigenvalues of e^A use the definition of the Matrix exponential and keep the following in mind:
- **Taylor** Swift has a **series of** top hits that caused **exponential** growth of her career
- Everything will be reduced to powers of the form A^n; you have dealt with this situation before. Be inspired by eq. (11) and re-write these.
- Use linear algebra to simplify things into an expression of the form $Q(sum - of - matrices)Q^-$ Use the Taylor-series trick on the sum to simplify the sum of matrices to something neat. This will give you the eigenvalues.
- The end-result will be simple; in fact, it will be simpler than before, and with a sufficient amount of hind-sight it might even seem kind of obvious.

Problem 8 *Stability*

What do the bounds on $\|\boldsymbol{x}_N\|$ (using Euler discretization) and $\|\boldsymbol{x}_N\|$ (using exponential discretization) tell you about the stability of Euler discretization and exponential discretization?

Answer:
YOUR SOLUTION HERE

2 R2D2 and control (`r2d2.py`)

In this problem, we will consider control of r2d2, who can be seen as an example of a primitive car model. R2D2 is characterized by an (x, y) location and the angle his

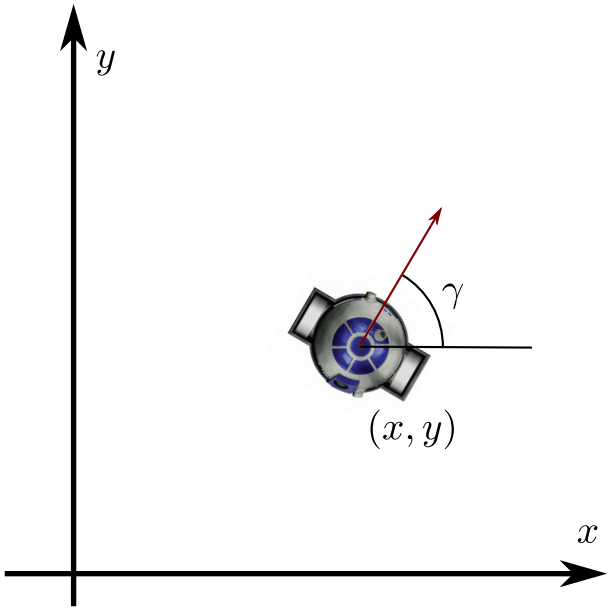


Figure 2: R2D2, as seen from above. The red arrow indicates the direction of motion.

direction of motion makes with the x-axis (this is also called the yaw) denoted by γ, see fig. 2.

R2D2 can travel forward and spin around on a dime in place. In other words, the available controls are the linear velocity v, in direction of the red arrow, as well as the angular velocity ω, which controls how fast R2D2 spins around in place (i.e., the rate at which the yaw γ changes). Taken together, we can describe R2D2 using state and control vectors

$$\boldsymbol{x}(t) = \begin{bmatrix} x(t) \\ y(t) \\ \gamma(t) \end{bmatrix}, \quad \boldsymbol{u}(t) = \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix},$$

and the equations of motion are:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) = \begin{bmatrix} v(t)\cos(\gamma(t)) \\ v(t)\sin(\gamma(t)) \\ \omega(t) \end{bmatrix} \tag{13}$$

The starting position will always be $\boldsymbol{x}(0) = \boldsymbol{x}_0 = \mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$. Your task is to help R2D2 carry out a couple of control-related tasks, all of which involve driving from the starting position $\boldsymbol{x}_0$ to a target position $\boldsymbol{x}^*$.

Problem 9 *Discretization*

We apply Euler discretization to the system with time constant Δ to get $\boldsymbol{x}_{k+1} = \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{u}_k)$. Give an explicit expression for $\boldsymbol{f}_k$ below.

Answer:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{u}_k) = \begin{bmatrix} \cdots \\ \cdots \\ \cdots \end{bmatrix}$$

YOUR SOLUTION HERE

Info:

- This is the same operation we have used to discretize all systems so far.

Problem 10 *Linearization*

The next task is to linearize the system around a particular, fixed point $\bar{\boldsymbol{x}}$, $\bar{\boldsymbol{u}}$. Give an explicit expression for the linearized system which takes the form

$$\boldsymbol{x}_{k+1} \approx A\boldsymbol{x}_k + B\boldsymbol{u}_k + \boldsymbol{d}$$

when the system is linearized around $\bar{\boldsymbol{x}} = \begin{bmatrix} 0 \\ 0 \\ \theta \end{bmatrix}$ and $\bar{\boldsymbol{u}} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

Answer:

$$\boldsymbol{x}_{k+1} \approx \begin{bmatrix} \cdots \\ \cdots \\ \cdots \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} \cdots \\ \cdots \\ \cdots \end{bmatrix} \boldsymbol{u}_k + \begin{bmatrix} \vdots \end{bmatrix}$$

YOUR SOLUTION HERE

Info:

- Remember to take Δ and $\bar{\boldsymbol{u}}$ into account.
- Linearization was discussed in lecture 7; use the general expression and compute the derivatives of $\boldsymbol{f}_k$ using what you have learned in Mat1.

Implementation: General comments You are free to use the framework or not. If you choose to use the framework, I have included a bit of boiler-plate code to set up a discrete model/environment and all you need is to complete the `ControlModel`. If you use the code, take care you understand the role of `x_target` and `Q0`! You need to set their values in the following exercises.

Problem 11 *Unitgrade self-check*

Implement the two functions to compute the Euler discretization from problem 9 (`f_euler`) and linearization matrices from problem 10 (`linearize`). The functions should return numpy arrays.

Info:

- The recommended way of doing this exercise is to specify R2D2 as a `ControlModel`, where you specify the dynamics as you have seen examples of in the exercises, and from that define a `DiscreteControlModel` and finally a `ControlEnvironment`. When you have done that, you can use function from the `DiscreteControlModel` to solve the exercise very simply, and you will get a self-check you have implemented the model correctly.
- Use unitgrade to make sure you get the result right. If you don't, check which matrices/entries in those matrices gives you problems
- We have actually worked with the linearization procedure during one of the exercises.
- Once you are done with this method, you can check (by inserting specific values and checking the result agrees) that your two previous expressions are 100% right. This is something I always try to do when I have to calculate something non-trivial.

We will now consider the problem of actually reaching the target location $\boldsymbol{x}^*$ at a minimal effort. To do so, we construct the following quadratic cost-function which depends on two parameters Q_0 and $\boldsymbol{x}^*$:

$$\int_0^{t_F} \left(\frac{1}{2} Q_0 \|\boldsymbol{x}(t) - \boldsymbol{x}^*\|^2 + \frac{1}{2} \|\boldsymbol{u}(t)\|^2 \right) dt \tag{14}$$

To simplify things, we will always consider a planning horizon of $t_F = 5$ seconds.

2.1 Optimal planning

Our first approach will be based on optimal planning using direct collocation. In this approach, we will handle the problem of reaching the destination as an equality constraint $\boldsymbol{x}(t_F) = \boldsymbol{x}^*$, and therefore set $Q_0 = 0$. The cost function is in other words simply

a quadratic cost function with $R = I$:

$$\int_0^{T_F} \frac{1}{2}\|\boldsymbol{u}(t)\|^2 dt. \tag{15}$$

Problem 12 *Optimal planning*

Implement the method `drive_to_direct(x_target, plot)`. The function should plan a path to the end-point $\boldsymbol{x}^* = \begin{bmatrix} 2 & 2 & \frac{\pi}{2} \end{bmatrix}^\top$ using direct collocation (see above). The boolean variable `plot` controls if the method plot the resulting state trajectory or not.

The method should return a $N \times 3$ `ndarray` of the state-trajectory. When you call the direct collocation method, first use a grid of size $N = 10$, then $N = 20$, and finally $N = 40$. Below, insert a plot of both the states trajectory, i.e. time along the x-axis and the coordinates of $\boldsymbol{x}$ along the y-axis, as well as a plot of R2D2s (x, y) position.

Answer:

Insert your figure here

Insert your figure here

Info:

- Easiest approach is to use the method we already implemented. I used the `DirectAgent`. Look at the examples.
- The trace plot can (here and elsewhere) be made using a call such as `plot_trajectory(traj[0],` but you are free to write your own plot code.
- The code for the second plot is already included in the script.

2.2 Simple Linearization

Although direct methods should solve the problem optimally they are brittle. Our first attempt at solving this problem will involve simple linearization using [Her24, algorithm 23]. The dynamics is discretized as before, and the cost function is discretized in the usual manner to be:

$$c_k = \Delta \left(\frac{1}{2} Q_0 \|\boldsymbol{x}_k - \boldsymbol{x}^*\|^2 + \frac{1}{2} \|\boldsymbol{u}_k\|^2 \right) \tag{16}$$

When we apply simple linearization, the system is linearized around $\bar{\boldsymbol{x}} = \mathbf{0}$ and $\bar{\boldsymbol{u}} = \mathbf{0}$. Use this to derive an LQR controller for the linearized problem (by planning on a horizon of $N = 50$ steps using the linearized dynamics and cost-matrices), and use this to plan the future actions.

Problem 13 *Control using simple linearization*

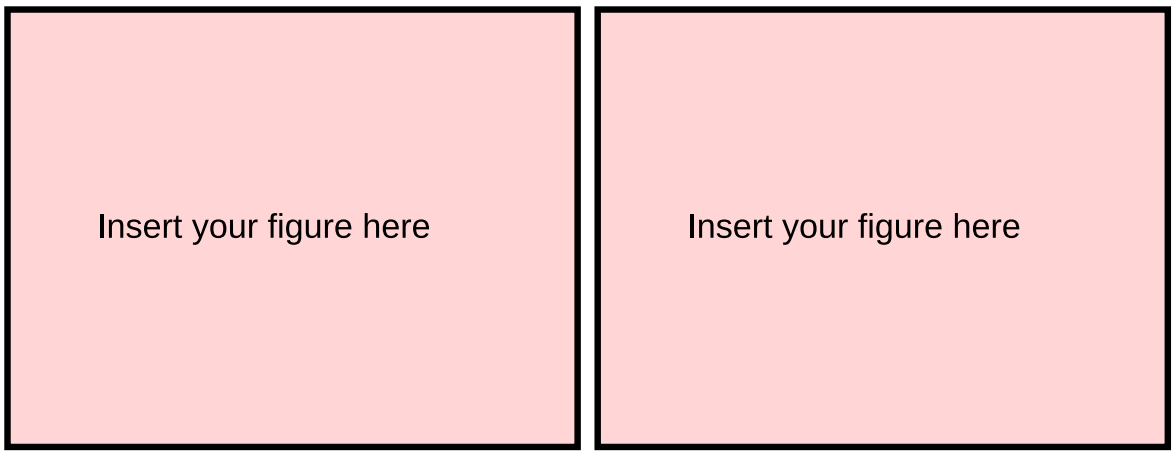
Implement the simple linaerization procedure as the function `drive_to_linearization(x_target, plot)`. The `plot`-variable should control whether we are doing plotting or not. The function should plan using the simple linarization method to reach `x_target` and return the obtained states when that plan was carried out in a simulation of the environment (i.e. using RK4 on a fine grid).

When done, test the method using first a simple problem consisting of driving forward for two meters, and then the problem we are interested in, where R2D2 drive to $(2, 2)$ and face north:

- $\boldsymbol{x}^* = \begin{bmatrix} 2 & 0 & 0 \end{bmatrix}^\top$
- $\boldsymbol{x}^* = \begin{bmatrix} 2 & 2 & \frac{\pi}{2} \end{bmatrix}^\top$

Insert figures below of the obtained state trajectories. The first case should work perfectly, the second case will fail. Explain the result.

Answer:



Intuitively, the second case fails because... **YOUR SOLUTION HERE**

Info:

- We solved a problem very similar to this in the exercises. You can re-use the solution without writing a new agent.
- You can turn on rendering of R2D2 by using the supplied rendering function in the `utils.py` file to see what happens.

2.3 Model-predictive control and Iterative linearization

Simple linearization fails to solve the problem, and full iterative LQR is difficult to implement and may fail to converge. In this problem, we will see if we can get away with something which is a lot simpler.

What we are going to do is, in each step, to apply the simple linearization procedure we tested in problem 13 (but where we expand around the current state) and then re-plan in the next step. Specifically, the method you should implement is defined in algorithm 1:

Your task is to implement this method and check the outcome.

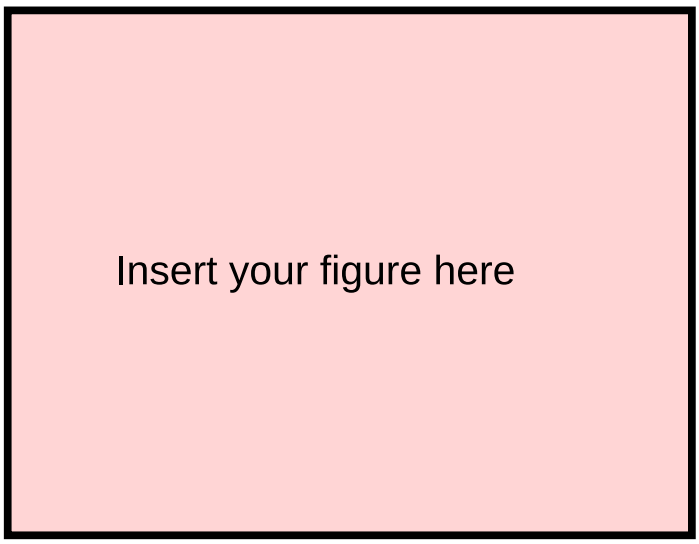
Problem 14 *MPC*

Apply iterative LQR to solve the problem by implementing the function `drive_to_iterative_linearization(x_target, plot)`. Insert the plot of the trajectory the script generates below, and comment on why you think it performs better.

Algorithm 1 MPC and iterative linearization

1: Initialize $\bar{\boldsymbol{u}} = \mathbf{0}$.
2: **for all** planning steps $k = 0, 1, 2, \ldots,$ **do**
3: Assume the agent is in state $\boldsymbol{x}_k$
4: Linearize the dynamics around $\bar{\boldsymbol{x}} = \boldsymbol{x}_k$ and $\bar{\boldsymbol{u}}$. Then apply LQR with a planning horizon of $N = 50$ to get control matrices $L_0, \boldsymbol{l}_0, L_1, \boldsymbol{l}_1, \ldots$
5: Compute and return the action $\boldsymbol{u}_k = L_0 \boldsymbol{x}_k + \boldsymbol{l}_0$
6: Set $\bar{\boldsymbol{u}} = \boldsymbol{u}_k$
7: **end for**
8: ▷ Note that line 4 and 5 are exactly what the Linarization agent, which you implemented in the exercises, does when called with $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$!

Answer:



Iterative linearization solves the problem because... **YOUR SOLUTION HERE**

Info:

- I solved this by writing a new `Agent`. Since you are re-using the iterative linearization approach, you can use this agent, but with the different choices of linearization points $\bar{\boldsymbol{x}}$, $\bar{\boldsymbol{u}}$.

References

[Her24] Tue Herlau. Sequential decision making. (Freely available online), 2024.