Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Dynamic Systems and Control
IDSC
Institut für Dynamische Systeme und Regelungstechnik

151-0563-01 **Dynamic Programming and Optimal Control** (Fall 2024)

**Programming Exercise** Topic: Infinite Horizon Problems

Issued: Oct 30, 2024 Due: Dec 15, 2024

# The Way of Water

As part of a water-based drone light show in lake Zürich, we need to safely navigate the main drone to its designated position. All the other drones are already positioned at their assigned locations and they are equipped with GPS to ensure that they remain stationary throughout the process.

The main drone, equipped with bright LEDs, has attracted the attention of a curious swan. The swan may attempt to catch the drone, and if it succeeds, we will need to deploy a new drone from the starting point. Similarly, if the main drone collides with one of the stationary drones, we will have to send out a new drone. Note that the stationary drones remain unaffected by any collisions, whereas the main drone sinks (and the TAs of DPOC will have to go pick them up).

## Problem Setup

### State Space

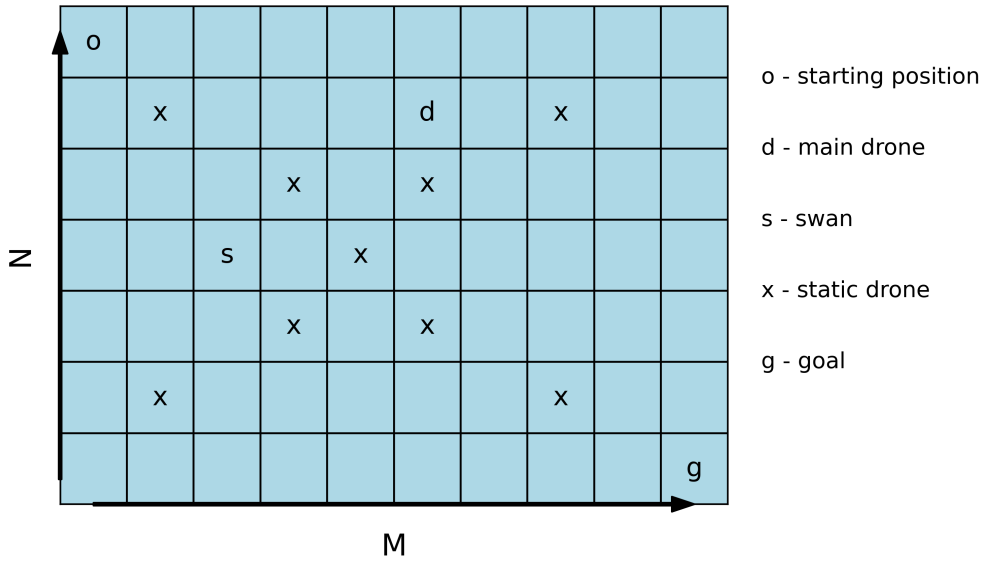



*Figure 1: Depiction of the state* $x_k = (5, 5, 2, 3)$

- We discretize the area of the lake where our light show takes place into a $M \times N$ grid as can be seen in Figure 1, where $M$ (`Constants.M`) is the number of columns, indexed from west to east, and $N$ (`Constants.N`) is the number of rows, indexed from south to north. Additionally, let $\mathcal{G} = \{0, 1, \ldots, M-1\} \times \{0, 1, \ldots, N-1\}$ be the set of all possible positions in the grid.
- The location of the starting position is provided in `Constants.START_POS` as a `numpy.array`:

  $$\texttt{[}x_{\text{start}},\ y_{\text{start}}\texttt{]} \in \mathcal{G}.$$

- The goal position is provided in `Constants.GOAL_POS` as a `numpy.array`:

  $$\texttt{[}x_{\text{goal}},\ y_{\text{goal}}\texttt{]} \in \mathcal{G}.$$

- The locations of the static drones are provided in `Constants.DRONE_POS` as `numpy.array`:

  $$\texttt{[[}x_{\text{static\_drone\_1}},\ y_{\text{static\_drone\_1}}\texttt{]}, \ldots, \texttt{[}x_{\text{static\_drone\_}d},\ y_{\text{static\_drone\_}d}\texttt{]]}.$$

  In order to get the $x$- and $y$-coordinate of static drone $i$,

  $$\texttt{[}x_{\text{static\_drone\_}i},\ y_{\text{static\_drone\_}i}\texttt{]} \in \mathcal{G},$$

  you can use `Constants.DRONE_POS[i]`.

- Our drone has the necessary sensors to locate itself and the swan. We include the position of the swan in the state, so that the state is as follows:

  $$\begin{aligned} S &= (\{0, 1, \ldots, M-1\} \times \{0, 1, \ldots, N-1\})^2 \\ \texttt{state} &= \texttt{[}x_{\text{drone}}, y_{\text{drone}}, x_{\text{swan}}, y_{\text{swan}}\texttt{]} \in S \end{aligned} \tag{1}$$

  where $x_{\text{drone}}, y_{\text{drone}}$ are the coordinates of our drone and $x_{\text{swan}}, y_{\text{swan}}$ are the coordinates of the swan. When we discuss the dynamics aside from the code, we write

  $$x = (x_{\text{drone}}, y_{\text{drone}}, x_{\text{swan}}, y_{\text{swan}}) \in \mathcal{S}.$$

- The processor on the drone has full knowledge of its current state and you can implement a feedback policy.

  We will use the following indexing scheme. `Constants.STATE_SPACE[state_index]` returns the state as listed below. Your solution will be marked as incorrect if you do not adhere to this convention!

| **Index** | **State** |
|---|---|
| 0 | (0,0,0,0) |
| 1 | (1,0,0,0) |
| 2 | (2,0,0,0) |
| ⋮ | ⋮ |
| $M$ | (0,1,0,0) |
| $M+1$ | (1,1,0,0) |
| ⋮ | ⋮ |
| $M \cdot N$ | (0,0,1,0) |
| ⋮ | ⋮ |
| $M \cdot N \cdot M \cdot N - 1$ | (M-1, N-1, M-1, N-1) |

*Table 1: Index to state mapping.*

- We provide the two functions `idx2state(idx)` and `state2idx(state)`, which can be used to convert

$$\texttt{idx} \in \{0, \ldots, M \cdot N \cdot M \cdot N - 1\} \quad \leftrightarrow \quad \texttt{state} = [x_{\text{drone}}, y_{\text{drone}}, x_{\text{swan}}, y_{\text{swan}}].$$

## Input Space

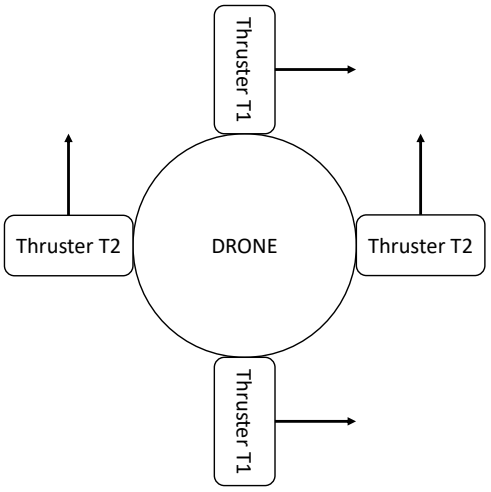



*Figure 2: Drone thrusters configuration.*

- The drone is equipped with four low-cost thrusters as shown in Figure 2. These thrusters can only be in one of three configurations: `forward` ($u = 1$), `backward` ($u = -1$), `still` ($u = 0$). We pair them as in Figure 2, so that the thrusters can only be activated in pairs T1 and T2. As indicated in Figure 2, the forward direction for the pair T1 points east, whereas for T2 it points north.

- Combining the possible pairs of configurations, we can obtain the following motion:

| | $T1 = -1$ | $T1 = 0$ | $T1 = 1$ |
|---|---|---|---|
| $T2 = -1$ | South-West | South | South-East |
| $T2 = 0$ | West | Stay | East |
| $T2 = 1$ | North-West | North | North-East |

*Table 2: Available input combinations. By activating forward or backwards T1 and T2, we can move the drone in 8 directions. By keeping the motors still we keep the drone still.*

- The input space is therefore

$$\begin{aligned} \mathcal{U} &= \{-1, 0, 1\}^2 \\ u &= (u_{\mathrm{T1}}, u_{\mathrm{T2}}) \in \mathcal{U} \end{aligned} \tag{2}$$

  where $u_{\mathrm{T1}}, u_{\mathrm{T2}}$ are the inputs applied to T1 and T2.

- We will use the following indexing scheme. `Constants.INPUT_SPACE[input_index]` returns the input as a `numpy.array` as listed below. Your solution will be marked as incorrect if you do not adhere to this convention!

| **Index** | **Input** |
|---|---|
| 0 | (-1,-1) |
| 1 | (0, -1) |
| 2 | (1, -1) |
| 3 | (-1, 0) |
| 4 | (0, 0) |
| 5 | (1, 0) |
| 6 | (-1, 1) |
| 7 | (0, 1) |
| 8 | (1, 1) |

*Table 3: Index to input mapping.*

## Disturbances

### Currents

The lake has strong currents that affect the movement of the drone. In this setting, the disturbance can be both unwanted and helpful! We have collected data on the currents and know the following about them:

- With probability `Constants.CURRENT_PROB[x,y]`[1] the drone at the coordinates $(x, y) \in \mathcal{G}$ will be affected by the current. With probability $1 -$ `Constants.CURRENT_PROB[x,y]` the drone is unaffected.

- If the drone is affected by the current, a displacement $w_k^{\text{current}} \in \{-2, -1, 0, 1, 2\}^2$ takes place. The displacement is modeled by a `numpy.array`, which is available via[2]

  `Constants.FLOW_FIELD[x,y]` -> $[w_{k,x}^{\text{current}}, w_{k,y}^{\text{current}}]$.

[1] Note that `numpy` uses (row, column) indexing, which is different from what we depicted in Figure 1, but this is purely a convention. Please use the indexing as provided here. That is, `Constants.CURRENT_PROB` is a `numpy.array` with shape `(M,N)`.

[2] That is, `Constants.FLOW_FIELD` is a `numpy.array` with shape `(M,N,2)`.

Hence, $w_k^{\text{current}}$ has the following probability distribution (conditioned on $x_k$ only; the word `Constants` in `Constants.FLOW_FIELD` and `Constants.CURRENT_PROB` is omitted for brevity):

$$p_{w_k^{\text{current}}|x_k}(w_k|x_k) = \begin{cases} \texttt{CURRENT\_PROB}[x_{\texttt{drone}}, y_{\texttt{drone}}] \\ \qquad\qquad \text{if } w_k^{\text{current}} = \texttt{FLOW\_FIELD}[x_{\texttt{drone}}, y_{\texttt{drone}}], \\ 1 - \texttt{CURRENT\_PROB}[x_{\texttt{drone}}, y_{\texttt{drone}}] \\ \qquad\qquad \text{if } w_k^{\text{current}} = 0, \\ 0 \qquad\qquad \text{otherwise.} \end{cases} \tag{3}$$

**Swan**

As mentioned in the beginning, the drone is hunted by a swan. Since its movement is random, it is modeled as a disturbance:

- With probability `Constants.SWAN_PROB` the Swan moves by one cell in the direction of the drone $w_k^{\text{swan}} \in \{-1, 0, 1\}^2$. With probability 1-`Constants.SWAN_PROB` the Swan stays in its position, $w_k^{\text{swan}} = (0, 0)$.
- In order to determine the closest cell to the drone for the swan, we use the angle $\theta = \arctan2(y_{\text{drone}} - y_{\text{swan}}, x_{\text{drone}} - x_{\text{swan}})$. By splitting the circle into 8 equal parts we can map $\theta$ to one of the 8 surrounding cells as follows:
  - **East (E)**: $-\frac{\pi}{8} \leq \theta < \frac{\pi}{8}$
  - **North-East (NE)**: $\frac{\pi}{8} \leq \theta < \frac{3\pi}{8}$
  - **North (N)**: $\frac{3\pi}{8} \leq \theta < \frac{5\pi}{8}$
  - **North-West (NW)**: $\frac{5\pi}{8} \leq \theta < \frac{7\pi}{8}$
  - **West (W)**: $\theta \geq \frac{7\pi}{8}$ or $\theta < -\frac{7\pi}{8}$
  - **South-West (SW)**: $-\frac{7\pi}{8} \leq \theta < -\frac{5\pi}{8}$
  - **South(S)**: $-\frac{5\pi}{8} \leq \theta < -\frac{3\pi}{8}$
  - **South-East (SE)**: $-\frac{3\pi}{8} \leq \theta < -\frac{\pi}{8}$

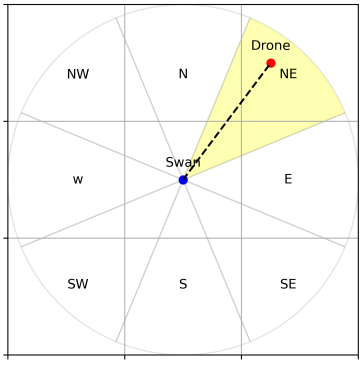



*Figure 3: Depiction of the swan moving towards the main drone and of the divisions in angular regions. Since the swan heads NE to the drone, if it moves, will move to the NE cell.*

- The swan can move through every cell even if there is a stationary drone inside.

Now, let $\text{dir}(x_k) \in \{-1, 0, 1\}^2$ be the direction from swan to main drone at time step $k$ according to the list above. Then $w_k^{\text{swan}}$ has the following probability distribution:

$$p_{w_k^{\text{swan}}|x_k}(w_k^{\text{swan}}|x_k) = \begin{cases} \texttt{Constants.SWAN\_PROB} & \text{if } w_k^{\text{swan}} = \text{dir}(x_k), \\ 1 - \texttt{Constants.SWAN\_PROB} & \text{if } w_k^{\text{swan}} = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

**Spawn**
When the main drone sinks, either because it hits a static drone or because it is caught by the swan, it moves to a random grid cell except the start position $\text{s} = (x_{\text{start}}, y_{\text{start}})$ (because you would hush it away to deploy a new drone). For this case we introduce the disturbance variable $w_k^{\text{spawn}} \in \{0, 1, \ldots, M-1\} \times \{0, 1, \ldots, N-1\}$, with the following probability distribution:

$$p_{w_k^{\text{spawn}}|x_k,u_k}(w_k^{\text{spawn}}|x_k, u_k) = \begin{cases} \frac{1}{M \cdot N - 1} & \text{if } w_k^{\text{spawn}} \neq (x_{\text{start}}, y_{\text{start}}), \\ 0 & \text{else.} \end{cases} \tag{5}$$

Note that the probability distribution of $w_k^{\text{spawn}}$ does not depend on the current state or the current input, so we will account for when it has an effect on the swan position in the dynamics of our system model.

The disturbances $w_k^{\text{current}}, w_k^{\text{swan}}$ and $w_k^{\text{spawn}}$ are all mutually independent. At time step $k$ we denote the entire disturbance vector as $w_k = (w_k^{\text{current}}, w_k^{\text{swan}}, w_k^{\text{spawn}})$.

## Dynamics

In this section we are going to incrementally build up the dynamics of the given problem. Let

$$x_k = \begin{pmatrix} x_{\text{drone},k} \\ y_{\text{drone},k} \\ x_{\text{swan},k} \\ y_{\text{swan},k} \end{pmatrix}, \quad u_k = \begin{pmatrix} u_{\text{T1},k} \\ u_{\text{T2},k} \end{pmatrix}, \quad w_k^{\text{current}} = \begin{pmatrix} w_{x,k}^{\text{current}} \\ w_{y,k}^{\text{current}} \end{pmatrix},$$
$$w_k^{\text{swan}} = \begin{pmatrix} w_{x,k}^{\text{swan}} \\ w_{y,k}^{\text{swan}} \end{pmatrix}, \quad w_k^{\text{spawn}} = \begin{pmatrix} w_{x,k}^{\text{spawn}} \\ w_{y,k}^{\text{spawn}} \end{pmatrix}.$$

We notice that $u_k$ only influences the evolution of the first two elements of the state (i.e. the position of the drone and <u>not</u> the position of the swan), $w_k^{\text{current}}$ only influences $x_{\text{drone}}$ and $y_{\text{drone}}$, $w_k^{\text{swan}}$ only influences $x_{\text{swan}}$ and $y_{\text{swan}}$, and $w_k^{\text{spawn}}$ only influences $x_{\text{swan}}$ and $y_{\text{swan}}$.

As an example, consider

$$x_k = \begin{pmatrix} 2 \\ 3 \\ 0 \\ 0 \end{pmatrix}, \; u_k = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \; w_k^{\text{current}} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}, \; w_k^{\text{swan}} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \; w_k^{\text{spawn}} = \begin{pmatrix} 4 \\ 5 \end{pmatrix},$$

and suppose there is no static drone on the way from $(2, 3)$ to $(4, 2)$ in the grid. Then

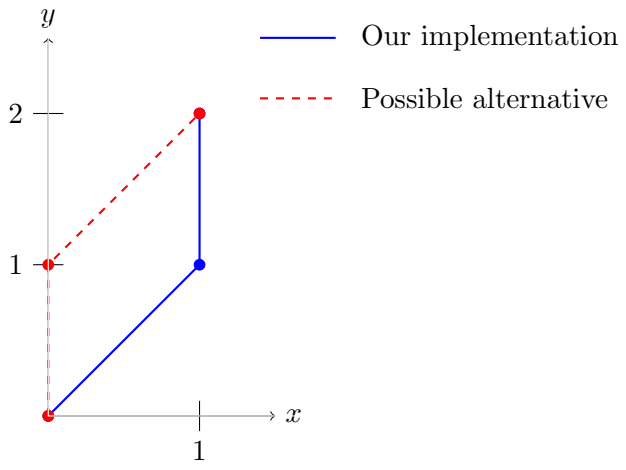
$$x_{k+1} = \begin{pmatrix} x_{\text{drone},k} & +u_{\text{T1},k} & +w_{x,k}^{\text{current}} & \\ y_{\text{drone},k} & +u_{\text{T2},k} & +w_{y,k}^{\text{current}} & \\ x_{\text{swan},k} & & & +w_{x,k}^{\text{swan}} \\ y_{\text{swan},k} & & & +w_{y,k}^{\text{swan}} \end{pmatrix} = \begin{pmatrix} 2 & +1 & +1 & \\ 3 & +1 & -2 & \\ 0 & & & +1 \\ 0 & & & +1 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \\ 1 \\ 1 \end{pmatrix}.$$

We now focus on including the case of the drone moving outside the grid or onto a cell with the swan or a static drone. For this, let $\mathrm{s} = (x_{\text{start}}, y_{\text{start}})$ and denote

$$c_k = \begin{pmatrix} x_{\text{drone},k} \\ y_{\text{drone},k} \end{pmatrix}, \quad d_k = \begin{pmatrix} x_{\text{drone},k} + u_{\text{T1},k} + w_{x,k}^{\text{current}} \\ y_{\text{drone},k} + u_{\text{T2},k} + w_{y,k}^{\text{current}} \end{pmatrix} \quad \text{and} \quad e_k = \begin{pmatrix} x_{\text{swan},k} + w_{x,k}^{\text{swan}} \\ y_{\text{swan},k} + w_{y,k}^{\text{swan}} \end{pmatrix}. \tag{6}$$

We make the following considerations and model the dynamics:

1. If the drone ends up in the cell of any other stationary drone on its way to the next state, it crashes. We provide the function `bresenham(start, end)` that returns the coordinates that the drone traverses on a line between `start` and `end`. As the traversed cells might be ambiguous (e.g. `start=[0,0], end=[1,2]` in the figure below), make sure to use the provided function.




2. If the main drone sinks, the swan moves with equal probability to any grid cell except the starting position. The drone sinks if $d_k = e_k$ or $d_k \notin \mathcal{G}$ or when the main drone crashes with a static drone on the way from its current position ($c_k$) to its next position ($d_k$), i.e. $\texttt{bresenham}(c_k, d_k) \cap \texttt{Constants.DRONE\_POS} \neq \emptyset$[3]. When the main drone sinks, you position a new main drone in the starting position:

$$x_{k+1} = \begin{cases} \begin{pmatrix} \mathrm{s} \\ w_k^{\text{spawn}} \end{pmatrix} & \text{if } d_k = e_k \text{ or } d_k \notin \mathcal{G} \text{ or } \texttt{bresenham}(c_k, d_k) \cap \texttt{Constants.DRONE\_POS} \neq \emptyset, \\ \begin{pmatrix} x_{\text{drone},k} & +u_{\text{T1},k} & +w_{x,k}^{\text{current}} & \\ y_{\text{drone},k} & +u_{\text{T2},k} & +w_{y,k}^{\text{current}} & \\ x_{\text{swan},k} & & & +w_{x,k}^{\text{swan}} \\ y_{\text{swan},k} & & & +w_{y,k}^{\text{swan}} \end{pmatrix} & \text{otherwise.} \end{cases} \tag{7}$$

[3]Here we are making a slight abuse of notation. In the code, $\texttt{bresenham}(c_k, d_k)$ and `Constants.DRONE_POS` are lists and not sets, but the idea is that there is a collision with a static drone if they share at least one element.

3. If the swan is on the goal position at the same time the drone reaches it, you need to send a new drone.

4. As soon as the drone reaches the goal position and there is no swan there, the show can start: Congrats! Your job is done.

5. Please adhere to the convention that the transition probabilities at the undefined states (where the main drone is at the same position as a static drone or as the swan) are zeros only. Your solution will be marked as incorrect otherwise!

### Cost Function

- We want to reach the starting position of the main drone for the show (i.e. the goal) as soon as possible using the least amount of energy (thus, minimizing the control inputs applied) and deploying the least number of drones possible. We therefore decide to minimize the cost function defined by the following stage cost $g_k$:

$$\begin{aligned} g_k(x_k, u_k, w_k) = &\texttt{Constants.TIME\_COST} \\ &+ \texttt{Constants.THRUSTER\_COST} \cdot (|u_{\mathrm{T1},k}| + |u_{\mathrm{T2},k}|) \\ &+ \texttt{Constants.DRONE\_COST} \cdot h(x_k, u_k, w_k) \end{aligned} \tag{8}$$

  where $h(x_k, u_k, w_k)$ equals 1 if we need to send a new drone at time step $k+1$ and 0 otherwise.

- Please adhere to the convention that the expected stage costs at the undefined states (where the main drone is at the same position as a static drone or as the swan) are zeros. Your solution will be marked as incorrect otherwise!

### Tasks

Your tasks are the following (all are required to obtain a score, but it helps in making progress to complete them separately and in order):

(a) Implement the function

`compute_transition_probabilities` in `ComputeTransitionProbabilites.py`

to return the transition probabilities `P` $\in \mathbb{R}^{K \times K \times L}$, where $K$ is the number of possible states and $L$ is the number of possible inputs. To compute `P`, adhere to the ordering convention in Section **State Space** and Section **Input Space**:

`P[i,j,k]` is the probability of transitioning from state `state_space[i]` to state `state_space[j]` when applying input `input_space[k]`.

(b) Implement the function

`compute_expected_stage_cost` in `ComputeExpectedStageCosts.py`

to return the expected stage costs $Q \in \mathbb{R}^{K \times L}$. To compute `Q`, adhere to the ordering convention in Section **State Space** and Section **Input Space**:

`Q[i,k]` is the expected cost of applying input `input_space[k]` when at state `state_space[i]`.

(c) Implement the function

`solution` in `Solver.py`

to compute the optimal cost `J` $\in \mathbb{R}^K$ and the optimal policy `policy` $\in \mathbb{N}^K$, using an algorithm of your choice.

`J[i]` is the expected cost incurred when starting at state `i` using policy `policy`.

If `input_space[k]` is the optimal input for the state `state_space[i]`, you can write

```
policy[i] = input_space[k]
```

Your solution will be marked as incorrect if you do not adhere to the described conventions!

## Evaluation and Scoring

1. You are disqualified if you are found to plagiarize the solution of another team.
2. If you submit a solution that is not correct, it will not be considered for ranking. A solution is considered not correct if it fails on at least one test case. When floating points comparisons are required to check correctness, we use `numpy.allclose(a, b, rtol=1e-04, atol=1e-07)`, where `a` and `b` are the values to compare.
3. We rank the submitted solutions based on the average run time over 10 different instances of the problem.
4. For the top three submissions according to the ranking, we will issue prizes including a tour of Verity with Prof. D'Andrea, signed certificates and gift vouchers of 100 CHF.

The judgement of the TA is final.

## Python Files Provided

A set of Python files is provided on the class website. Use them to solve the above problem. Follow the structure strictly as the grading is automated for fairness reasons.

| | |
|---|---|
| `Constants.py` | An instance of the Python class `Constants` will contain the constants used in the problem. |
| `ComputeTransitionProbabilities.py` | Contains `compute_transition_probabilities`, the Python function that has to be implemented to calculate the transition probabilities `P`. |
| `ComputeExpectedStageCosts.py` | Contains `compute_expected_stage_cost`, the Python function that has to be implemented to calculate the expected stage costs `Q`. |
| `Solver.py` | Contains `solution`, the Python function that has to be implemented to solve the problem. |
| `utils.py` | Contains python functions that are shared between files, e.g. `bresenham`. Feel free to add functions here. |
| `main.py` | Python script that loads a test case (described by `Constants.py`), executes the implemented algorithms and generates the results to be visualized at a later time. |
| `test.py` | We provide three test cases that you can use to check your transition probability tensor, stage cost matrix and optimal cost by running `test.py`. You will not be evaluated on these test cases. |
| `visualization.py` | We provide a visualization script that can help you debugging your solution. After running your solver, you can visualize the computed cost and policy with `python3 visualization.py`. Click any cell in the plot to fix the position of the swan. |

## Deliverables

A maximum of two students can work as one team. Hand in one submission per team by e-mail to aterpin@ethz.ch by the due date ($15^{\text{th}}$ of December 2024, everywhere on Earth) with the subject `[programming exercise submission 2024]`, containing your Python implementation of the following files (only submit these files!):

- `ComputeTransitionProbabilites.py`
- `ComputeStageCosts.py`
- `Solver.py`
- `utils.py`

Note that you are only allowed to use packages that are part of the standard library or the `environment.yml` file.

Include all files in one `zip`-archive, named `DPOCEx_Name1_Number1(_Name2_Number2).zip`, where `Name` is the full name of the student who worked on the solution and `Number` is the student identity number (e.g `DPOCEx_JaneDoe_12345678_JohnDoe_87654321.zip`).

We will test your implementation in a conda environment created with the following instructions:

```
conda env create -f environment.yml
conda activate dpoc_pe
```

You can check your current Python version with `python3 --version`. Make sure not to use packages that are not installed with the above list of commands. If you are new to conda, we recommend that you check out this guide to get started.

We will send you a confirmation upon receiving your e-mail, but we will not check that your code is working and respects the required format. We will discard submissions that do not run in the above mentioned environment or do not fulfill the format required.

You are ultimately responsible that we receive your working solution in time.

**Submission Checklist**

- ☐ Your code runs with the original `main.py` script in a conda environment created with the commands above.
- ☐ You did not modify the function signatures (name and arguments) in the submission files
  - `ComputeTransitionProbabilites.py`
  - `ComputeExpectedStageCosts.py`
  - `Solver.py`
- ☐ You only submit the following files, which contain all your code
  - `ComputeTransitionProbabilites.py`
  - `ComputeExpectedStageCosts.py`
  - `Solver.py`
  - `utils.py`