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 2025)

Programming Exercise Topic: Stochastic Shortest Path Problems

Issued: Oct 29, 2025 Due: Nov 26, 2025

Infinite-Horizon Control for Endless-Corridor Flight

A small flapping-wing vehicle faces an endless, side-scrolling corridor: narrow openings appear at shifting heights, walls drift past at constant speed, and one mistimed move ends the run. The goal is simple: stay in the air and slip through every gap. We frame this as a stochastic shortest path problem.

In spirit and practice, the system we build is **b**alanced, **i**ntelligent, **r**isk-aware, and **d**urable; in the pages that follow, we refer to it as BIRD.

Simulation

Launch the simulation by running `simulation.py` to play around and get a feel for the dynamics. You can observe the following behavior:

- The BIRD's horizontal coordinate and (relative) horizontal velocity are fixed; obstacles translate left at constant speed.
- The vertical motion is controlled with discrete inputs: press `W` (weak flap), `S` (strong flap), or `No key` (no flap).
- A strong flap (`S`) introduces a stochastic effect on the acceleration, whereas a weak flap (`W`) and no flap (`No key`) have a deterministic behavior.
- New obstacles spawn at the right boundary with randomized spacing and gap height.
- Note that the BIRD can fly touching the ceiling, or "walking" on the ground.
- Yeah, because of the delayed effect of the inputs, it is quite tough to play this game manually. Can we do better using dynamic programming and optimal control?

We next formalize this behavior and pose the control objective: find a policy that efficiently steers the BIRD through the corridor while avoiding collisions.

Problem Setup

State Space

We discretize the simulation screen into a grid of size $X \times Y$ (with X columns and Y rows, `Const.X` and `Const.Y` in the code); see Figure 1. We encode the BIRD's state and the descriptions of the next M obstacles in the state

$$x_k = (y_k, v_k, d_k^{(1)}, d_k^{(2)}, \dots, d_k^{(M)}, h_k^{(1)}, h_k^{(2)}, \dots, h_k^{(M)}),$$

where

- y_k, v_k are the BIRD's altitude (row index) and vertical velocity. In particular, $y_k \in \mathcal{S}_y := \{0, 1, \dots, Y-1\}$ and $v_k \in \mathcal{S}_v := \{-V_{\max}, -V_{\max}+1, \dots, 0, \dots, V_{\max}-1, V_{\max}\}$ with $V_{\max}$ given as `Const.V_max` in the code.
- $d_k^{(1)}$ is the horizontal distance (in grid cells) from the BIRD (at the leftmost column in the grid) to the nearest obstacle. In particular, $d_k^{(1)} \in \mathcal{S}_{d_1} := \{0, 1, \dots, X-1\}$.
- $d_k^{(2)}, \dots, d_k^{(M)}$ are the distances between obstacles: $d_k^{(i)}$ is the horizontal distance between obstacle $i-1$ and i. We allow $d_k^{(i)} = 0$ to encode "no obstacle at index i". Moreover, $d_k^{(2)} > 0$ when $d_k^{(1)} = 0$. When $d_k^{(i)} = 0$, then $d_k^{(j)} = 0$ for all $j > i \geq 2$. Additionally, the total horizontal span of visible obstacles must fit on screen, i.e., $\sum_{i=1}^{M} d_k^{(i)} \leq X - 1$. We have $d_k^{(i)} \in \mathcal{S}_d := \{0\} \cup \{D_{\min}, D_{\min}+1, \dots, X-1\}$, where $0 < D_{\min} \leq X-1$ (`Const.D_min` in the code).

  Food for thoughts: Given any initial state that satisfies all the conditions discussed, what is the minimum number of obstacles on screen at all times?

- $h_k^{(1)}, \ldots, h_k^{(M)}$ are the vertical indices of the obstacles' gaps (see the Dynamics section for a description of the gaps). We have $h_k^{(i)} \in \mathcal{S}_h \subseteq \{0, 1, \ldots, Y-1\}$ is a finite set of possible gap heights (in the code, we provide `Const.S_h` as a list). For $i \geq 2$, if $d_k^{(i)} = 0$, then $h_k^{(i)} = h_d$, with the default value h_d given as `Const.S_h[0]` in the code.
- M is the maximum number of obstacles visible simultaneously.

All together, the state space at every time-step k is

$$\begin{aligned}\mathcal{S}^+ = \Big\{ & (y_k, v_k, d_k^{(1)}, d_k^{(2)}, \ldots, d_k^{(M)}, h_k^{(1)}, h_k^{(2)}, \ldots, h_k^{(M)}) \in \mathcal{S}_y \times \mathcal{S}_v \times \mathcal{S}_{d_1} \times \mathcal{S}_d^{M-1} \times \mathcal{S}_h^M \mid \\ & d_k^{(2)} > 0 \text{ if } d_k^{(1)} = 0, \\ & d_k^{(j)} = 0 \text{ if } d_k^{(i)} = 0 \text{ and } j > i \text{ for } i \geq 2, \\ & h_k^{(i)} = h_d \text{ if } d_k^{(i)} = 0 \text{ and } i \geq 2, \\ & \sum_{i=1}^{M} d_k^{(i)} \leq X - 1 \Big\}.\end{aligned}$$

Note on the choice of M. To guarantee that at most M obstacles can simultaneously fit on screen given the minimum spacing $D_{\min}$, we set

$$M = \left\lceil \frac{X}{D_{\min}} \right\rceil.$$

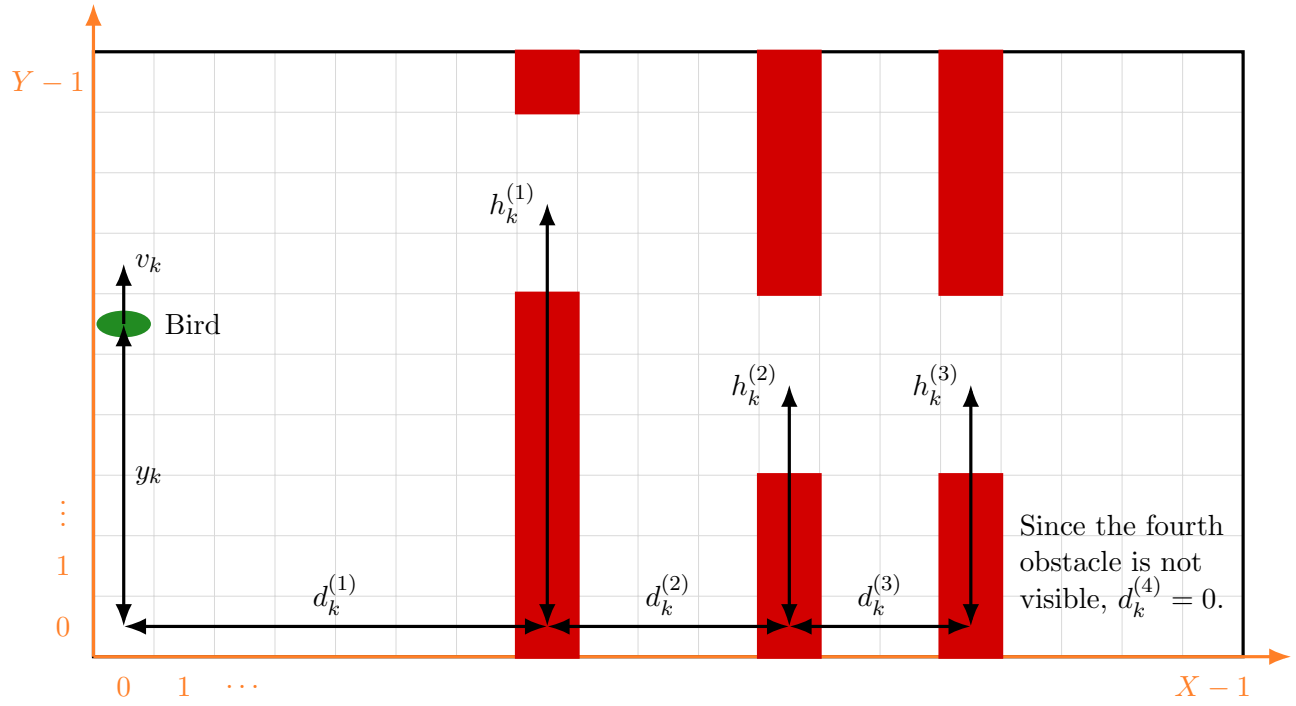

Figure 1: State-space visualization for a maximum of $M = 4$ obstacles on screen.

Input Space

In the simulation, we control the altitude of the BIRD by selecting an input u_k from a finite set at each timestep:

- $u_k = U_{\text{no\_flap}}$: no flap (no influence on current trajectory, `Const.U_no_flap` in the code).
- $u_k = U_{\text{weak}}$: weak flap (small upward impulse, `Const.U_weak` in the code).
- $u_k = U_{\text{strong}}$: strong flap (larger upward impulse, `Const.U_strong` in the code).

Overall, the input space is

$$\mathcal{U}(x_k) = \{U_{\text{no\_flap}}, U_{\text{weak}}, U_{\text{strong}}\}.$$

Ordering convention

We enumerate the discrete state space in *lexicographic order* over the tuple

$$(y_k, v_k, d_k^{(1)}, d_k^{(2)}, \ldots, d_k^{(M)}, h_k^{(1)}, h_k^{(2)}, \ldots, h_k^{(M)}).$$

Concretely, the enumeration follows the following nested loops.

```
state_space = []
for y in range(Const.Y):
  for v in range(-Const.V_max, Const.V_max + 1):
    for d1 in range(Const.X):
      for d2 in [0, *range(Const.D_min, Const.X - 1)]:
        ...
          for dM in [0, *range(Const.D_min, Const.X - 1)]:
            for h1 in Const.S_h:
              for h2 in Const.S_h:
                ...
                  for hM in Const.S_h:
                    # If all conditions are satisfied:
                    state_space.append((y,v,d1,d2,...,dM, h1,h2,...,hM))
```

Thus `state_space[i]` refers to the i-th tuple in this order. Note that certain tuples need to be excluded from the state space, as well as the terminal state! That is, we are working with $\mathcal{S}^+$. The input space is ordered as

```
input_space = [Const.U_no_flap, Const.U_weak, Const.U_strong]
```

so that indices $i = 0, 1, 2$ correspond to $u = U_{\text{no\_flap}}$, $u = U_{\text{weak}}$, and $u = U_{\text{strong}}$, respectively. We write $K = |\texttt{state\_space}|$ and $L = |\texttt{input\_space}|$.

Your solution will be marked as incorrect if you do not adhere to this convention!

Disturbances

The system is affected by the disturbances $w_k^{(\text{spawn})}, w_k^{(h)}, w_k^{(\text{flap})}$:

- *Spawn decision and gap height.* Let $s \in \mathbb{N}$ denote the available free horizontal distance to the right boundary (a precise definition of s is given later in the Dynamics section). A new obstacle appears at the right boundary ($w_k^{(\text{spawn})} = 1$) or not ($w_k^{(\text{spawn})} = 0$) with probability

$$\Pr\left(w_k^{(\text{spawn})} = 1 \mid x_k, u_k\right) = p_{\text{spawn}}(s), \qquad \Pr\left(w_k^{(\text{spawn})} = 0 \mid x_k, u_k\right) = 1 - p_{\text{spawn}}(s).$$

The probability with which this happens depends on the distance:

$$p_{\text{spawn}}(s) \;=\; \begin{cases} 0 & s \le D_{\min} - 1, \\ \dfrac{s - (D_{\min} - 1)}{X - D_{\min}} & D_{\min} \le s \le X - 1, \\ 1 & s \ge X \end{cases}$$

That is, the probability increases linearly from 0 at $s = D_{\min} - 1$ to 1 at $s = X - 1$. The new obstacle's gap height $w_k^{(h)}$ is distributed uniformly in $\mathcal{S}_h$.

- *Flapping uncertainty.* Let $\mathcal{W}_v := \{-V_{\text{dev}}, -V_{\text{dev}}{+}1, \ldots, V_{\text{dev}}{-}1, V_{\text{dev}}\}$ with $V_{\text{dev}} > 0$ (`Const.V_dev`). For the applied input $u \in \{U_{\text{no\_flap}}, U_{\text{weak}}, U_{\text{strong}}\}$, the disturbance $w_k^{(\text{flap})}$ takes values in $\mathcal{W}_v$ and has the probability distribution:

$$\Pr\left(w_k^{(\text{flap})} = \bar{w}_k^{(\text{flap})} \mid x_k, u_k\right) = \begin{cases} \dfrac{1}{2V_{\text{dev}} + 1} & \text{if } u = U_{\text{strong}}, \\ 1 & \text{if } u \in \{U_{\text{no\_flap}}, U_{\text{weak}}\} \text{ and } \bar{w}_k^{(\text{flap})} = 0, \\ 0 & \text{if } u \in \{U_{\text{no\_flap}}, U_{\text{weak}}\} \text{ and } \bar{w}_k^{(\text{flap})} \neq 0. \end{cases}$$

It is assumed that $w_k^{(*)}$ are conditionally independent with all prior variables x_p, u_p, $w_p^{(\text{spawn})}$,$w_p^{(h)}$,$w_p^{(\text{flap})}$, $p < k$, given x_k and u_k.

Dynamics

The vertical motion of the BIRD is

$$y_{k+1} = \min\{\max\{y_k + v_k, 0\}, Y - 1\},$$
$$v_{k+1} = \min\{\max\{v_k + u_k + w_k^{(\text{flap})} - g, -V_{\max}\}, V_{\max}\},$$

with gravity g (`Const.g`) and velocity bound $V_{\max}$ (`Const.V_max`).

- We say *is_collision* holds when $d_k^{(1)} = 0$ and the BIRD is outside the gap, i.e., $|y_k - h_k^{(1)}| > \frac{G-1}{2}$, where G is the gap size. On collision, our flight experiment ends. We model this with a transition to a cost-free termination state.

For the obstacle update we define the intermediate quantities $\hat{d}_{k+1}^{(i)}, \hat{h}_{k+1}^{(i)}, i = 1, \ldots, M$ distinguishing between the following cases:

- *Passing.* We say *is_passing* holds when the BIRD is horizontally aligned with the nearest obstacle, i.e., $d_k^{(1)} = 0$, and its altitude lies within the gap of that obstacle, i.e., $|y_k - h_k^{(1)}| \le \frac{G-1}{2}$, where G is the (odd) gap size (`Const.G`). Then obstacle $i = 1$ leaves the scene and indices shift:

$$\hat{d}_{k+1}^{(i)} = \begin{cases} d_k^{(2)} - 1, & \text{if } i = 1 \\ d_k^{(i+1)}, & \text{if } 2 \le i \le M - 1 \\ 0, & \text{if } i = M \end{cases}, \qquad \hat{h}_{k+1}^{(i)} = \begin{cases} h_k^{(i+1)}, & \text{for } i = 1, \ldots, M - 1, \\ h_d, & \text{for } i = M. \end{cases}$$

- *Normal drift.* When neither *is_passing* nor *is_collision* occur (i.e., $d_k^{(1)} > 0$), obstacles move one column left:

$$\hat{d}_{k+1}^{(1)} = d_k^{(1)} - 1, \qquad \hat{d}_{k+1}^{(i)} = d_k^{(i)} \text{ for } i = 2, \ldots, M, \qquad \hat{h}_{k+1}^{(i)} = h_k^{(i)} \text{ for } i = 1, \ldots, M.$$

With these quantities, we define the available free distance s as

$$s = X - 1 - \sum_{i=1}^{M} \hat{d}_{k+1}^{(i)}.$$

The quantity s determines the probability with which a new obstacle appears; see the Disturbances section. In particular, let $m_{\min} \in \{2, \dots, M\}$ be the smallest index with no assigned obstacle, i.e., $\hat{d}_{k+1}^{(m_{\min})} = 0$. If none satisfies this condition, in the following let $m_{\min} = M$ (the choice is inconsequential; can you see why, given the transitions below?). The state transition at this state component is as follows:

$$d_{k+1}^{(i)} = \begin{cases} s & \text{if } w_k^{(\text{spawn})} = 1 \text{ and } i = m_{\min} \\ \hat{d}_{k+1}^{(i)} & \text{otherwise,} \end{cases} \qquad h_{k+1}^{(m_{\min})} = \begin{cases} w_k^{(h)} & \text{if } w_k^{(\text{spawn})} = 1 \text{ and } i = m_{\min} \\ \hat{h}_{k+1}^{(i)} & \text{otherwise.} \end{cases}$$

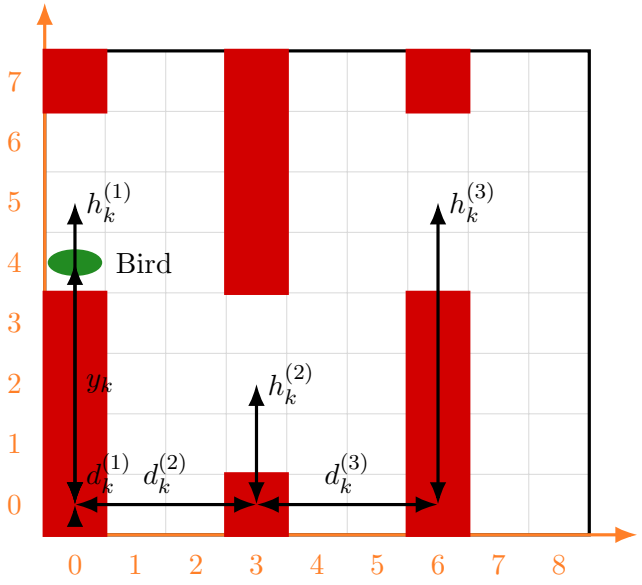

Figure 2: State-space visualization of example at time k.

Examples.

In the following, we present a few examples to illustrate the dynamics. Assume the parameters are

$$M = 3, \quad G = 3, \quad g = 1, \quad X = 9, \quad Y = 8, \quad V_{\max} = 1,$$
$$U_{\text{strong}} = 3, \quad V_{\text{dev}} = 2, \quad D_{\min} = 3, \quad S_h = \{0, 2, 5\}, \quad h_d = 0.$$

At time k, let

$$y_k = 4, \quad v_k = 0, \quad (d_k^{(1)}, d_k^{(2)}, d_k^{(3)}) = (0, 3, 3), \quad (h_k^{(1)}, h_k^{(2)}, h_k^{(3)}) = (5, 2, 5), \quad u_k = U_{\text{strong}},$$

as visualized in Figure 2. Since $u_k = U_{\text{strong}}$, the velocity deviation $w_k^{(\text{flap})} \in \{-2, \dots, 2\}$ is uniformly distributed with probability

$$\Pr\left(w_k^{(\text{flap})} = \bar{w}_k^{(\text{flap})} \mid x_k, u_k\right) = \frac{1}{2V_{\text{dev}} + 1}.$$

For this example, assume the noise realization is $w_k^{(\text{flap})} = 1$. The BIRD's vertical motion is then

$$y_{k+1} = \min\{\max\{y_k + v_k, 0\}, Y - 1\} = \min\{\max\{4 + 0, 0\}, 7\} = 4,$$
$$v_{k+1} = \min\{\max\{v_k + u_k + w_k^{(\text{flap})} - g, -V_{\max}\}, V_{\max}\} = \min\{\max\{0 + 3 + 1 - 1, -1\}, 1\} = 1.$$

Since

$$d_k^{(1)} = 0 \quad \text{and} \quad |y_k - h_k^{(1)}| = |4 - 5| = 1 \leq \tfrac{G-1}{2} = 1,$$

the condition *is_passing* holds (note that we are considering y_k to evaluate this condition, not y_{k+1}). The indices shift, and the nearest obstacle disappears. Therefore, we obtain the intermediate quantities

$$(\hat{d}_{k+1}^{(1)}, \hat{d}_{k+1}^{(2)}, \hat{d}_{k+1}^{(3)}) = (d_k^{(2)} - 1, d^{(3)}, 0) = (2, 3, 0),$$
$$(\hat{h}_{k+1}^{(1)}, \hat{h}_{k+1}^{(2)}, \hat{h}_{k+1}^{(3)}) = (h_k^{(2)}, h_k^{(3)}, h_d) = (2, 5, 0).$$

With these quantities, the available free distance s is given by

$$s = X - 1 - \sum_{i=1}^{M} \hat{d}_{k+1}^{(i)} = (9 - 1) - 2 - 3 - 0 = 3.$$

By the formula in the Disturbance section, the spawn probability is

$$p_{\text{spawn}}(s) = \frac{s - (D_{\min} - 1)}{X - D_{\min}} = \frac{3 - (3 - 1)}{9 - 3} = \frac{1}{6}.$$

We now consider two scenarios:

- *Passing, no spawning.* For this scenario, assume the spawn noise realization is $w_k^{(\text{spawn})} = 0$. Then the next obstacle states are identical to the intermediate values:

  $$(d_{k+1}^{(1)}, d_{k+1}^{(2)}, d_{k+1}^{(3)}) = (\hat{d}_{k+1}^{(1)}, \hat{d}_{k+1}^{(2)}, \hat{d}_{k+1}^{(3)}) = (2, 3, 0),$$
  $$(h_{k+1}^{(1)}, h_{k+1}^{(2)}, h_{k+1}^{(3)}) = (\hat{h}_{k+1}^{(1)}, \hat{h}_{k+1}^{(2)}, \hat{h}_{k+1}^{(3)}) = (2, 5, 0).$$

  The resulting next state for this scenario is shown in Figure 3.

- *Passing, with spawning.* For this scenario, assume the spawn noise realization is $w_k^{(\text{spawn})} = 1$ and the obstacle-height noise realization is $w_k^{(h)} = 5$. From the intermediate quantities

  $$(\hat{d}_{k+1}^{(1)}, \hat{d}_{k+1}^{(2)}, \hat{d}_{k+1}^{(3)}) = (2, 3, 0),$$

  we find that the spawning index is $m_{\min} = 3$. Therefore, the next-state values for the obstacle at index $i = m_{\min} = 3$ are

  $$d_{k+1}^{(m_{\min})} = s = 3, \qquad h_{k+1}^{(m_{\min})} = w_k^{(h)} = 5.$$

  The remaining obstacles are updated accordingly to the intermediate values. We therefore obtain the next obstacles states as

  $$(d_{k+1}^{(1)}, d_{k+1}^{(2)}, d_{k+1}^{(3)}) = (\hat{d}_{k+1}^{(1)}, \hat{d}_{k+1}^{(2)}, d_{k+1}^{(m_{\min})}) = (2, 3, 3),$$
  $$(h_{k+1}^{(1)}, h_{k+1}^{(2)}, h_{k+1}^{(3)}) = (\hat{h}_{k+1}^{(1)}, \hat{h}_{k+1}^{(2)}, h_{k+1}^{(m_{\min})}) = (2, 5, 5).$$

  The resulting next state for this scenario is shown in Figure 4.

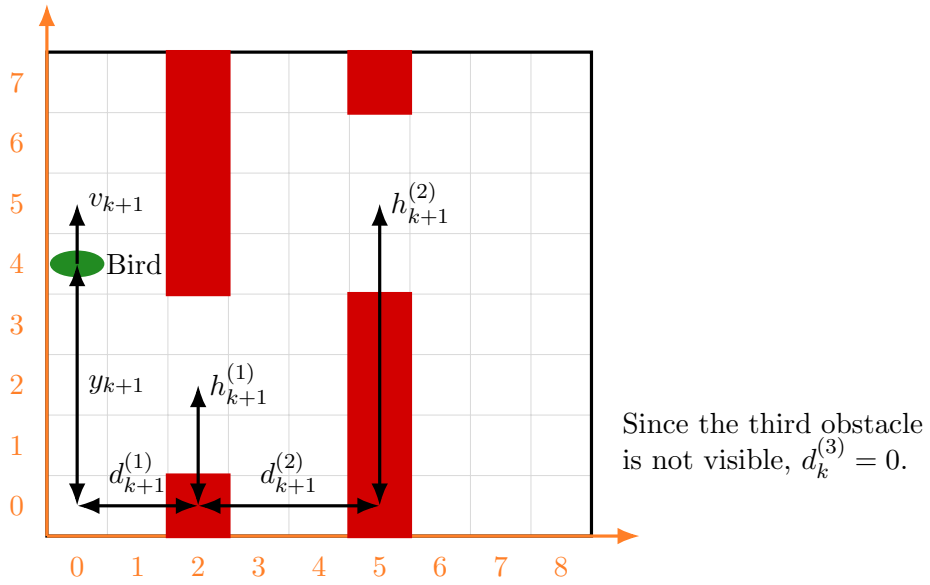

Figure 3: State-space visualization of the state at time $k+1$ for the "Passing, no spawning" *scenario.*

Cost Function

We minimize control effort while avoiding collisions. Let $h(u) \in [0, 1)$ quantify effort:

$$h(u_k) = \begin{cases} 0 & \text{if } u_k = U_{\text{no\_flap}}, \\ \lambda_{\text{weak}} & \text{if } u_k = U_{\text{weak}}, \\ \lambda_{\text{strong}} & \text{if } u_k = U_{\text{strong}}, \end{cases} \qquad \text{with} \quad 0 \le \lambda_{\text{weak}} < \lambda_{\text{strong}} < 1.$$

The $\lambda_{\text{weak}}, \lambda_{\text{strong}}$ are given in the code as `Const.lam_weak` and `Const.lam_strong`. The (expected) stage cost is

$$q(x_k, u_k) = \big(h(u_k) - 1\big).$$

Interpretation. In non-collision steps, $g(x_k, u_k) = h(u_k) - 1 < 0$: continuing yields negative cost (a "reward"), but unnecessary flaps make it *less* negative, discouraging effort. Overall, an optimal policy will always attempt to keep flying, thus avoiding collisions. There is a catch! A perfect, optimal policy would make the BIRD fly forever, incurring $-\infty$ cost. Thus, in this example, we need all policies to yield a finite cost. How do you think we achieve this? Careful when tuning parameters during your testing!

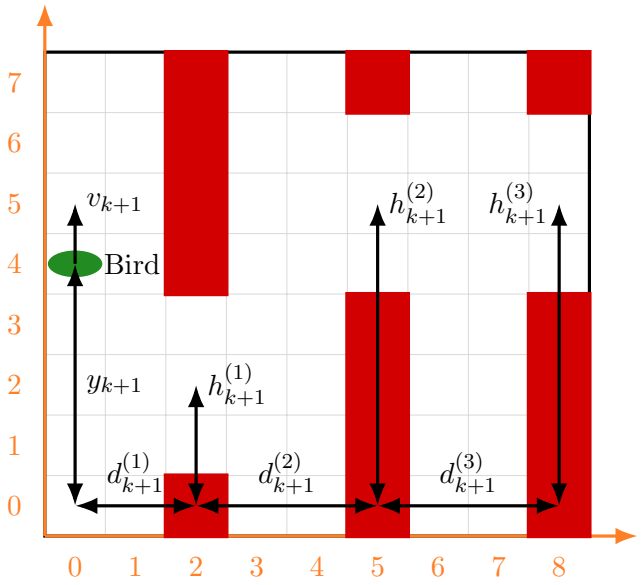

Figure 4: State-space visualization of the state at time $k+1$ for the "Passing, with spawning" *scenario.*

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 `ComputeTransitionProbabilities.py`

to return the transition probabilities `P` $\in [0,1]^{K\times K\times L}$, 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,l]` is the probability of transitioning from state `state_space[i]` to state `state_space[j]` when applying input `input_space[l]`.

(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,l]` is the expected cost of applying input `input_space[l]` 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 (you do not need to use the functions implemented in the previous steps, but they are made available).

`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.
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. We only consider the runtime of `Solver.py`, but all three templates need to be completed and will be checked for correctness.
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.

`Const.py`	An instance of the Python class `Const` will contain the numerical values 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. `is_collision`. Feel free to add functions here.
`main.py`	Python script that loads a test case (described by `Const.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.
`simulation.py`	This script contains the simulation environment. If you run this script directly you can try to control the BIRD manually.

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 26$^{\text{th}}$ of November 2025 (anywhere on Earth) with the subject `[programming exercise submission 2025]`, containing your Python implementation of the following files (only submit these files!):

- `ComputeTransitionProbabilities.py`
- `ComputeExpectedStageCosts.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 (Legi 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
  - `ComputeTransitionProbabilities.py`
  - `ComputeExpectedStageCosts.py`
  - `Solver.py`
- ☐ You only submit the following files, which contain all your code
  - `ComputeTransitionProbabilities.py`
  - `ComputeExpectedStageCosts.py`
  - `Solver.py`
  - `utils.py`