EECS590 – Mini Project 2

Mini Project 2 Report

Author: **Davis Payne**

Course: **EECS590**

Date: January 30, 2026

Answers

Problem 1

The Python code is available at `Davis_Payne_MiniProject2_Problem1.py`.

Question 1

Optimal Policy and Qualitative Behavior

After convergence of value iteration, the optimal policy $\pi^*(s)$ is extracted by selecting, for each state s, the action that maximizes the expected one-step return plus the discounted value of the next state. Formally, the optimal policy is given by

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \sum_{s'} P(s' \mid s, a) \left(R(s, a, s') + \gamma V^*(s') \right).$$

This greedy action selection with respect to the optimal value function $V^*(s)$ is applied independently at each state.

Optimal Policy Extraction

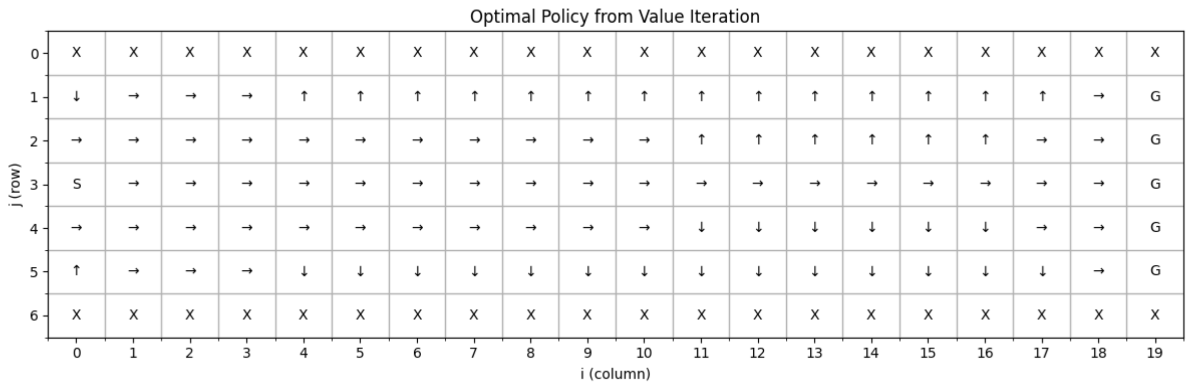

Figure 1: Optimal policy obtained from value iteration.

Inspection of the extracted policy (Figure 1) reveals that the drone strongly prefers to remain near the center line of the chasm ($j = 3$). In the central region of the grid, the optimal action is predominantly to move forward, indicating that the agent prioritizes steady progress toward the goal. When the drone deviates above or below the center line, the policy frequently selects corrective actions that steer it back toward the middle of the grid.

Near the crash boundaries ($j \leq 0$ or $j \geq 6$), the policy becomes more cautious and favors actions that reduce the likelihood of being pushed out of bounds by wind. Overall, the policy demonstrates risk-aware behavior: it advances efficiently when safe, but actively corrects its trajectory to maintain a buffer from hazardous regions. Thus, the optimal policy favors safety over risky shortcuts and does not intentionally exploit boundary regions.

Effect of Increasing Wind Strength

As the wind parameter p increases, stochastic disturbances become more frequent and more severe. Consequently, the optimal policy adapts by becoming increasingly conservative. With higher wind strength, the agent performs corrective actions more often and re-centers itself sooner after deviations. Forward progress is occasionally sacrificed in favor of maintaining a safer position, reflecting an increased emphasis on minimizing crash probability. In short, larger values of p induce a stronger preference for remaining within the safest corridor of the state space.

Effect of Reducing the Crash Penalty

When the crash reward is reduced to -1, crashing becomes no more costly than a single time step. Under this reward structure, the optimal policy becomes substantially more aggressive. The agent is less inclined to correct its position toward the center and is more willing to tolerate risky states near the boundaries. Since the expected penalty of crashing is low, the policy prioritizes faster progress over safety, even if it increases the probability of termination. As a result, the behavior shifts from conservative and risk-averse to opportunistic and risk-taking.

Question 2

Challenge (a): Fine-Grained Grid as an Approximation

We refine the original 20×7 grid into a 60×21 grid by subdividing each coarse tile into a 3×3 block. Thus, one coarse cell corresponds to 3 fine columns and 3 fine rows. Denote the fine-grid indices by $(\tilde{i}, \tilde{j})$ with

$$\tilde{i} \in \{0, \dots, 59\}, \qquad \tilde{j} \in \{0, \dots, 20\},$$

and define the fine-grid center line by $\tilde{j} = 10$ (since 21 rows).

(1) Mapping geometry and terminal sets. The start and goal regions are mapped by scaling coordinates by 3:

$$(0, 3) \mapsto (0, 9), \qquad (19, n) \mapsto (59, 3n).$$

Crash boundaries refine similarly: in the coarse problem, the episode terminates when $j \leq 0$ or $j \geq 6$. On the fine grid, we terminate when the vertical coordinate leaves the corresponding band,

$$\tilde{j} \leq 0 \quad \text{or} \quad \tilde{j} \geq 20.$$

(Equivalently, one may mark the top and bottom fine rows as absorbing crash boundaries.)

(2) Adapting the wind parameter p. In the coarse problem, a single action is followed by a wind disturbance with total probability mass controlled by p (with additional mass for a larger ± 2 push). On the fine grid, one coarse time step corresponds to 3 fine time steps. To preserve the *overall* likelihood of a wind disturbance over the distance of one coarse tile, we choose a per-fine-step wind probability $\tilde{p}$ such that the probability of *no wind* over 3 fine steps matches the coarse case:

$$(1-\tilde{p})^3 \approx (1-p) \quad \Longrightarrow \quad \tilde{p} = 1-(1-p)^{1/3}.$$

This keeps the wind "strength per unit distance" approximately invariant under refinement.

For distance-from-center scaling, replace the coarse center $j=3$ by the fine center $\tilde{j}=10$ and define

$$\tilde{E}(\tilde{j}) = \frac{1}{1+(\tilde{j}-10)^2}, \qquad \tilde{p}(\tilde{j}) = \operatorname{clip}\left(\frac{\tilde{B}}{\tilde{E}(\tilde{j})}, 0, 1\right),$$

where $\tilde{B} = \tilde{p}(10) = \tilde{p}$ is the fine-grid analogue of $B = p(3)$.

(3) Adapting wind step sizes. A coarse push of ± 1 row corresponds to ± 3 fine rows, and a coarse push of ± 2 corresponds to ± 6 fine rows. Therefore, the same wind structure can be applied on the fine grid by using jump sizes ± 3 and ± 6 in $\tilde{j}$ (with the same 50/50 split in direction).

(4) Adapting rewards. To preserve the same return scale, we keep the *per unit distance* cost unchanged. Since moving forward one coarse tile now takes 3 fine steps, we set the per-step reward to

$$\tilde{r}_{\text{step}} = -\frac{1}{3},$$

so that the total step cost over 3 fine steps equals the coarse cost -1. Terminal rewards can be kept the same,

$$\tilde{R}_{\text{goal}} = R, \qquad \tilde{r}_{\text{crash}} = r,$$

because reaching the goal/crash represents the same physical event (just with finer resolution).

Question 2b

Challenge (b): Continuous Limit on $[0,19] \times [0,6]$ and an Approximation

We consider a limiting version of the problem with a well-defined state space

$$\mathcal{S} = [0,19] \times \{0,1,\ldots,6\},$$

where the horizontal coordinate is continuous and the vertical coordinate remains discrete. Since a closed-form solution is infeasible, we approximate the continuous dimension using a fine discretization with

$$dx = 0.05, \qquad dy = 1,$$

so states are of the form (x_k, y) with $x_k = k\,dx$.

Dynamics. Actions are interpreted at the refined scale:

$$\texttt{forward}: (x,y) \mapsto (x+dx, y), \qquad \texttt{left/right}: (x,y) \mapsto (x, y \mp 1).$$

The episode terminates when $x \geq 19$ (goal) or $y \leq 0$ or $y \geq 6$ (crash). Wind is applied after the action exactly as in the discrete model, with probabilities depending on the distance from the centerline.

Scaling the wind. To preserve wind strength per unit distance, we choose a per-step probability $\tilde{p}$ such that the probability of no wind over unit distance is unchanged:

$$(1-\tilde{p})^{1/dx} \approx (1-p) \quad \Rightarrow \quad \tilde{p} = 1-(1-p)^{dx}.$$

For small dx, this yields $\tilde{p} \approx p\,dx$. If the drone reaches the goal too easily, increasing the base parameter p restores comparable difficulty.

Rewards and interpolation. To maintain the same cost per unit horizontal progress, the step reward is scaled to

$$r_{\text{step}} = -dx,$$

while terminal rewards remain unchanged. Bellman updates require evaluating $V(x, y)$ at intermediate horizontal locations; we therefore use linear interpolation in x, while keeping y discrete.

This approximation yields a consistent continuous-limit analogue of the original grid-world problem while remaining numerically tractable. Python code attached here: The Python code is available at `Davis_Payne_MiniProject2_Problem1Q2b.py`.

Question 2c

Continuous time formulation. Instead of discrete time steps, assume the drone moves in continuous time with constant horizontal velocity $v > 0$. The state is $s(t) = (x(t), y(t)) \in [0, 19] \times \{0, \ldots, 6\}$, with

$$\frac{dx}{dt} = v \quad \Rightarrow \quad x(t) = x(0) + vt,$$

so horizontal distance scales linearly with v. For numerical implementation, we discretize time with a small dt, yielding

$$x_{k+1} = x_k + v\,dt,$$

which is equivalent to a spatial step $dx = v\,dt$. Wind is applied after the deterministic motion using the same mechanism as before, with probabilities interpreted per unit distance (approximately $\tilde{p} \approx p\,dx$). To preserve cost per unit distance, the step reward is scaled as $r \approx -dx$, while terminal rewards remain unchanged.

Question 2d

The formula defines wind probability in continuous space by integrating the probability density p(x) along the drone's trajectory. The left side represents the exact calculation: the first integral accumulates wind probability during horizontal movement from j to j+tv, while the second integral handles vertical movement where wind probability remains constant at p(j). The right side provides a computationally efficient approximation using the trapezoidal rule for numerical integration. Specifically, [p(j) + p(j+vt)]/2 · t approximates the horizontal integral by averaging the wind probability at the start and end positions, while p(j)(1-vt) handles the vertical component. This linear interpolation maintains consistency with the discrete grid case while enabling smooth continuous-space transitions. The formula bridges discrete MDPs to continuous stochastic control by replacing finite sums with integrals.

Question 2e

Instead of discrete instantaneous pushes, wind now applies a continuous drift velocity over time interval $[0, 1]$. A standard wind event (probability p) applies velocity v away from the centerline $j = 3$, while a stronger p^2 event applies velocity $2v$ for the same duration. The drone's position evolves continuously as

$$j(t) = j_0 + v_{\text{drift}} \cdot t$$

during the drift interval. For overlapping events, the recommended approach is to queue them: complete the current drift over $[0, 1]$, then apply the next event starting at $t = 1$, maintaining temporal consistency with the discrete model. Alternatively, you could interrupt (stop current drift and immediately apply new velocity from current position) or superpose (add velocities together). This continuous drift formulation is more physically realistic than discrete jumps since real wind applies continuous forces rather than instantaneous impulses. The queuing approach keeps each "time step" as one complete drift interval, making the MDP formulation cleaner.

Question 2f

(f) Extending the Windy Chasm with Orientation

To extend the Windy Chasm problem, the state is expanded to include the drone's orientation. Each state is described by the horizontal position i, the vertical position j, and the orientation angle θ. The horizontal position is discretized in steps of one unit, while the vertical position is discretized more finely in steps of 0.05. The orientation is discretized into a small number of directions (for example, eight equally spaced angles) to keep the state space manageable.

Actions now control both the drone's orientation and whether thrust is applied. At each step, the drone may rotate left, rotate right, or maintain its current orientation, and may either apply thrust or remain stationary. When thrust is applied, the drone moves forward in the direction it is facing. This movement is decomposed into horizontal and vertical components based on the current orientation.

Wind is modeled as a vertical disturbance that pushes the drone away from the centerline of the chasm. If the drone is above the centerline, the wind pushes it upward; if it is below the centerline, the wind pushes it downward. As in the original Windy Chasm problem, wind events occur stochastically, with stronger wind occurring less frequently. The effect of the wind is applied during each time step before the drone's position is snapped back to the nearest grid point.

After motion and wind effects are applied, the resulting position is discretized back onto the grid. If the drone leaves the vertical bounds of the chasm, it crashes and the episode ends with a negative reward. Reaching the far side of the chasm terminates the episode with a positive reward, while all other steps incur a small negative cost.

Including orientation allows the optimal policy to exploit orientation to counteract wind effects. Rather than only moving straight across the chasm, the drone can angle into the wind to maintain stability, leading to more realistic and flexible navigation strategies under strong wind conditions.

Problem 2: Robot Motion Control via Reward Shaping

Part (a): State Representation

Proposed State Representation

The state should capture the robot's deviation from straight-line motion using lidar data. Define the state as:

$$s = (d_L, d_R, \alpha_L, \alpha_R, \Delta v)$$

where:

Distance Features:

- d_L = average distance to left wall from left-sector lidar rays
- d_R = average distance to right wall from right-sector lidar rays

Angular Features:

- α_L = average angle of incidence for left-sector rays
- α_R = average angle of incidence for right-sector rays

Asymmetry Feature:

- $\Delta v = d_R - d_L$ (lateral deviation metric)

Discretization Strategy

We can discretize the asymmetry into states:

- **Centered (C):** $|\Delta v| < \epsilon$ (symmetric distances)
- **Left deviation (L1, L2, L3):** $\Delta v < -\epsilon$ (closer to left wall)
- **Right deviation (R1, R2, R3):** $\Delta v > \epsilon$ (closer to right wall)

The grid position indicates how far off-center the robot is. The "A" represents the agent starting at center. States marked "1" are acceptable corridor positions, "W" are walls (terminal crash states).

Why This is Markovian

This state representation is approximately Markovian because:

1. **Geometric Sufficiency:** The cone geometry ensures that current lidar readings fully determine the robot's lateral position and orientation relative to the corridor axis.
2. **Deviation Capture:** The asymmetry $\Delta v = d_R - d_L$ captures the robot's lateral offset. If $\Delta v > 0$, the robot is closer to the left wall (veering left). If $\Delta v < 0$, closer to right wall (veering right).
3. **Angular Information:** The angles α_L, α_R capture rotational misalignment. If the robot is angled, the incidence angles will differ from perpendicular.
4. **Future Independence:** Given current lidar readings (distances and angles), the future state distribution depends only on the current state and action, not on the history of how the robot arrived there.

5. **Kalman Filter Integration:** The Kalman filter provides smoothed estimates of (r, θ) that incorporate both lidar and odometry, reducing noise and making the state more Markovian by filtering out measurement noise.

Approximation Caveats:

- Motor dynamics have memory (momentum, acceleration), but at low speeds these effects are minimal
- Slip history could matter, but the cone geometry provides immediate geometric feedback
- The discretization loses some information, but captures the essential deviation signal

Part (b): Reward Function Design

Proposed Reward Function

Define $R(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$ as:

$$R(s, a) = r_{\text{forward}} + r_{\text{centering}} + r_{\text{correction}} + r_{\text{crash}}$$

Component Breakdown:

$$\begin{aligned}
r_{\text{forward}} &= +1.0 \quad \text{(constant reward for forward progress)} \\
r_{\text{centering}} &= -\lambda_1 |\Delta v| \quad \text{(penalty for being off-center)} \\
r_{\text{correction}} &= -\lambda_2 |a_{\text{correction}}| \quad \text{(penalty for large corrections)} \\
r_{\text{crash}} &= \begin{cases} -100 & \text{if collision (hit wall)} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

Parameter Suggestions:

- $\lambda_1 = 0.5$ (moderate penalty for deviation)
- $\lambda_2 = 0.1$ (small penalty to encourage smooth control)

Justification

1. **Forward Progress:** Constant +1 reward encourages motion regardless of hardware specifics.
2. **Centering Penalty:** $-\lambda_1 |\Delta v|$ encourages staying near the corridor center, making the robot robust to variations. This works for any hardware because it's based on geometric feedback, not encoder readings.
3. **Smooth Corrections:** $-\lambda_2 |a_{\text{correction}}|$ discourages oscillatory control, which would arise from aggressive over-correction. This promotes smooth, efficient motor control.
4. **Crash Avoidance:** Large negative reward for hitting walls ensures safety-critical constraint satisfaction.
5. **Hardware Independence:** The reward is based on geometric state (lidar) rather than encoder accuracy, making it robust to encoder errors, motor noise, and slip.

Part (c): Policy Iteration Challenges

Application of Policy Iteration

Policy iteration alternates between:

Policy Evaluation:

$$V^\pi(s) = \sum_{s'} P(s'|s, \pi(s)) \left[R(s, \pi(s)) + \gamma V^\pi(s')\right]$$

Policy Improvement:

$$\pi'(s) = \arg\max_a \sum_{s'} P(s'|s, a) \left[R(s, a) + \gamma V^\pi(s')\right]$$

Challenges with Convergence

1. Continuous Underlying State

Challenge: The true state $(r, \theta, \text{position})$ is continuous, but we discretize into grid cells.

Impact:

- Discretization creates aliasing: multiple continuous states map to the same discrete state
- Loss of precision near bin boundaries
- Transition probabilities are approximations of continuous dynamics

Mitigation:

- Use finer discretization in critical regions (near center)
- Add interpolation between adjacent states
- Consider tile coding or radial basis functions for better continuous state representation

2. Sensor Noise

Challenge: Lidar has measurement noise $\sim \mathcal{N}(0, \sigma^2_{\text{lidar}})$, odometry has drift.

Impact:

- Stochastic observations mean the same physical state can produce different lidar readings
- Transition probabilities $P(s'|s, a)$ must account for observation noise
- State estimation uncertainty from Kalman filter

Mitigation:

- Use Kalman filter estimates (already incorporated) to reduce noise
- Model $P(s'|s, a)$ as a mixture distribution over neighboring states
- Collect more samples to estimate noisy transitions accurately
- Consider POMDP formulation if noise is severe

3. Hardware Variability

Challenge: Motor response varies due to:

- Battery voltage (affects motor torque)
- Temperature (affects friction)
- Wheel wear (affects effective radius)
- Surface conditions (slip varies)

Impact:

- Transition dynamics $P(s'|s,a)$ are non-stationary
- Policy learned on fresh batteries may fail on depleted batteries
- Difficult to estimate accurate transition model from limited data

Mitigation:

- Collect training data across diverse conditions (battery levels, surfaces)
- Use robust policy optimization (maximize worst-case performance)
- Periodically re-estimate transitions online
- Consider model-free methods (Q-learning, actor-critic) that adapt online

4. Sample Complexity

Challenge: Accurately estimating $P(s'|s,a)$ requires many samples per (s,a) pair.

Impact:

- With noisy transitions, variance is high
- Need $O(1/\epsilon^2)$ samples to estimate probabilities within ϵ accuracy
- Large state/action space exacerbates this

Mitigation:

- Use function approximation to share information across similar states
- Prioritize exploration of high-uncertainty transitions
- Use model-based methods with uncertainty quantification (Gaussian Processes)

Recommended Approach

Given these challenges, a hybrid approach works best:

1. **Initial Policy Iteration:** Use tabular policy iteration with coarse discretization to get a baseline policy
2. **Transition Estimation:** Collect extensive empirical data to estimate $P(s'|s,a)$ with confidence intervals
3. **Online Adaptation:** Deploy the policy and use online Q-learning or policy gradient methods to fine-tune based on actual hardware behavior
4. **Robust Design:** Design the reward function (part b) to be hardware-agnostic by relying on geometric lidar feedback rather than encoder precision

Problem 3: Circuit Design via Sequential Decisions

(a) State Representation

The environment is modeled as a Markov decision process where the state consists of the agent's position on the grid and the current circuit layout. Formally, a state can be written as

$$s = (p, G),$$

where $p = (i, j)$ denotes the agent's grid position and G represents the component configuration of all grid cells. Each cell in G can be empty, contain a wire, or contain a NAND gate. The input cells provide binary signals A and B, and the output cell produces signal C.

This representation is Markov because the next state depends only on the current position, the current layout, and the selected action. Once the layout is known, no additional history is required to determine future behavior.

(b) Evaluating the `Done` Action

When the agent selects the `Done` action, the constructed circuit is evaluated to determine whether it implements the XOR function,

$$C = A \oplus B.$$

The evaluation is performed by testing all four input combinations $(A, B) \in \{0, 1\}^2$. For each case, signals are propagated deterministically through the grid following the orientation rules of wires and NAND gates. The output signal observed at the sink cell is compared with the correct XOR output. A circuit is considered correct only if it produces the correct result for all four input combinations.

(c) Insulation Constraint

An insulation constraint is introduced between grid cells $(1, 1)$ and $(2, 1)$. This constraint removes any direct connection between these cells during signal propagation, ensuring that no signal can pass across this boundary. The purpose of this constraint is to prevent unintended signal paths and to enforce more structured circuit designs.

(d) State-Space Growth and Computational Feasibility

Let n denote the number of grid cells and let K be the number of possible component types per cell. The number of possible circuit layouts grows on the order of K^n. Including the agent's position yields approximately nK^n possible states.

For a 4×4 grid ($n = 16$) with $K = 3$ component types, the number of states becomes extremely large, making exact tabular value iteration computationally infeasible. As a result, practical solutions require restricting the action space or using approximate or search-based methods.

Illustrative Scenario

Consider a scenario in which the agent has placed two NAND gates and several wires forming a partial circuit. For the input $(A, B) = (1, 0)$, the signal propagates correctly to produce $C = 1$, but for $(A, B) = (1, 1)$ the output remains high due to an unintended wire path. In this case, selecting `Done` yields a low reward because the circuit fails to satisfy the XOR truth table for all inputs.

The agent must therefore continue modifying the layout, for example by removing the unintended wire or inserting an additional NAND gate to block the undesired signal. This scenario illustrates how the reward signal depends on the global behavior of the circuit rather than on individual local placements, encouraging the agent to reason about long-term consequences of its actions.