Lab 03: Designing and Implementing Deliberative Intelligent Agents

---

Learning Objectives

After completing this lab, students will be able to:

- Understand the concept of deliberative intelligent agents and how they differ from reactive agents.
- Explain the working principles of model-based reflex, goal-based, and utility-based agents.
- Design agent-environment interactions using percepts, internal state, goals, and utility functions.
- Implement simple deliberative agents using structured procedures.
- Analyze the strengths and limitations of different deliberative agent architectures.

---

1 Theoretical Background

1.1 What are Deliberative Agents?

Deliberative agents are intelligent agents that **reason about the world before acting**. Unlike simple reflex agents, they do not rely only on the current percept. Instead, they:

- Maintain an **internal model** of the environment
- Consider **future consequences** of actions
- Make decisions based on **goals or utility values**

Deliberative agents are suitable for dynamic, partially observable, and complex environments.

1.2 Types of Deliberative Agents

Agent Type	Key Feature	Decision Basis
Model-Based Reflex Agent	Internal state (world model)	Condition-action rules
Goal-Based Agent	Explicit goals	Goal achievement
Utility-Based Agent	Utility function	Maximum expected utility

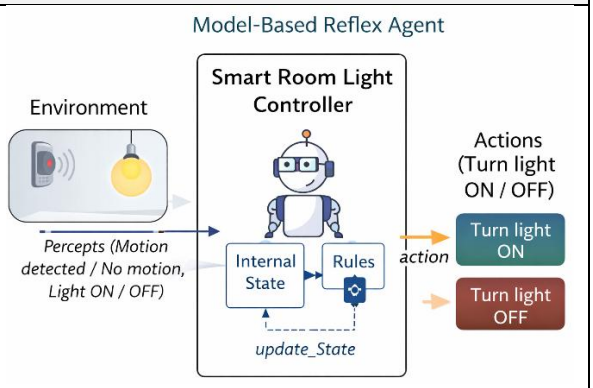
1.3 Model-Based Reflex Agent

A Model-Based Reflex Agent extends the simple reflex agent by maintaining an **internal state** that represents aspects of the environment not directly observable. Key characteristics include:

- Maintains a world model
- Updates internal state using percept history
- Uses condition–action rules
- Does not explicitly plan ahead

```
Agent Program (Conceptual):
```

```
state ← update_state(state, percept)
rule ← rule_match(state, rules)
action ← rule.action
```

Example: Smart Room Light Controller	
Environment Description: • A room with a motion sensor and a light • Motion may not always be detected correctly **Percepts:** • Motion detected / No motion • Light ON / OFF **Internal State:** • Last known motion status • Time since last motion **Actions:** • Turn light ON • Turn light OFF	 **Expected Outcome:** The light remains ON if recent motion was detected, even if the sensor temporarily fails.
Procedure to Implement	
Step 1: Define percepts and actions **Step 2:** Create an internal state variable **Step 3:** Write a state update function	**Step 4:** Define condition–action rules **Step 5:** Select action based on updated state

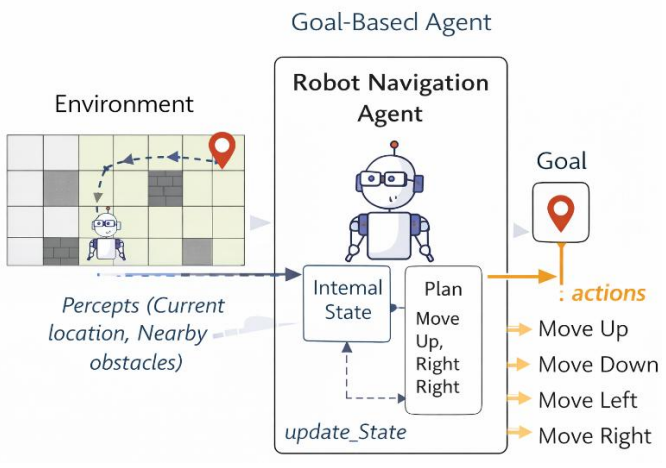
1.4 Goal-Based Agent

A Goal-Based Agent makes decisions by considering **future states** and selecting actions that help achieve a predefined goal. Key characteristics include:

- Explicit representation of goals
- Requires search or planning
- More flexible than reflex agents

```
Agent Program (Conceptual):

state ← update_state(state, percept)
plan ← search(state, goal)
action ← first_action(plan)
```

Example: Robot Navigation Agent	
Environment Description: • A robot moves in a grid environment • Some cells are blocked by obstacles **Goal:** • Reach the destination cell **Percepts:** • Current location • Nearby obstacles	

Actions: Move Up, Down, Left, Right	**Expected Outcome:** The robot reaches the goal following a valid path.

Procedure to Implement

Step 1: Represent the environment as a grid **Step 2:** Define initial state and goal state **Step 3:** Implement a simple search algorithm	**Step 4:** Generate a path to the goal **Step 5:** Execute actions step by step

1.5 Utility-Based Agent

A Utility-Based Agent selects actions based on a **utility function**, which measures how desirable a state is. Key characteristics include:

- Handles trade-offs between multiple goals
- Chooses optimal actions under uncertainty
- Uses utility values instead of binary goals

Agent Program (Conceptual):

```
For each possible action:
    compute expected utility
Choose action with maximum utility
```

Example: Autonomous Taxi Agent

Environment Description:

- A taxi operates in a city grid

Factors Affecting Utility:

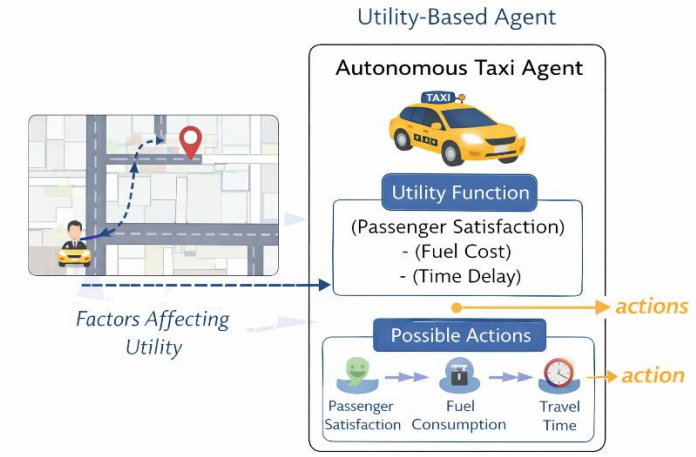
- Passenger satisfaction
- Travel time
- Fuel consumption
- Safety

Actions:

- Pick up passenger
- Choose route
- Drop passenger

Utility Function Example:

```
Utility = (Passenger Satisfaction)
          – (Fuel Cost)
          – (Time Delay)
```



Expected Outcome: The agent chooses the most efficient and beneficial route, not necessarily the shortest.

Procedure to Implement

Step 1: Identify decision factors **Step 2:** Assign utility values to outcomes **Step 3:** Define a utility function	**Step 4:** Evaluate possible actions **Step 5:** Select action with highest utility

2 Agent-Environment Implementation

Below we provide the details for our 8x8 maze solver grid environment, where the agent starts at some cell and tries to reach the goal cell. Whie the agent moves it finds walls as obstacles and tries to avoid them.

Grid Maze Solver 8x8

Maze Structure

- Grid size: 8×8
- Cells indexed (r, c) where 1-8
- Each cell can have walls on N, E, S, W

Start and Goal

- Start: (1,1)
- Goal: (7,6)

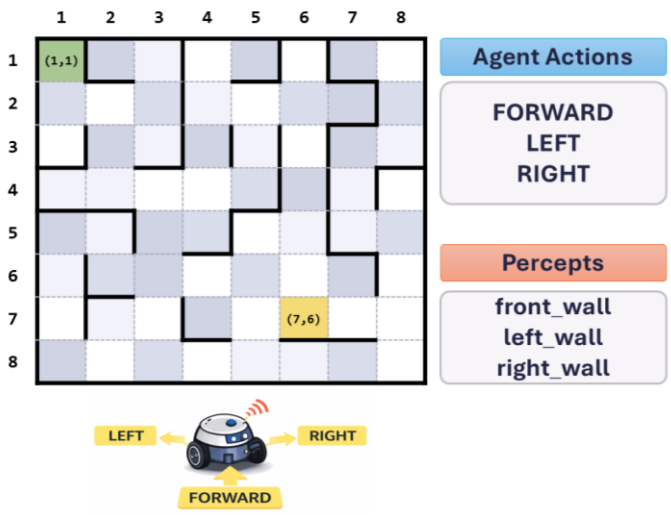
Data Structures

Wall representation
- wall[r][c] = {N:bool, E:bool, S:bool, W:bool}

Robot State + Movement
- Position: (r, c)
- Direction: dir ∈ {N,E,S,W}

Actions

- FORWARD
- TURN_LEFT
- TURN_RIGHT



Percepts

Each step the robot senses walls relative to its heading:
- front_wall ∈ {true,false}
- left_wall ∈ {true,false}
- right_wall ∈ {true,false}

2.1 Maze Grid Solver Model-Based Reflex Agent

Even though the maze is known, a model-based reflex agent is characterized by: (1) internal state (memory), (2) condition–action rules, and (3) no explicit planning/search.

Model-Based Reflex Agent for Grid Maze Solver 8x8

Internal State
- visited[r][c]
- last_action or last_direction

Rule Set
- If next to goal → move into goal
- If there is an unvisited neighbor → move to it (fixed priority e.g., Up, Right, Down, Left)
- Else → backtrack: move to a neighbor with the smallest visited count

Implementation procedure
- Initialize visited to False
- Loop until goal:
- mark current cell visited
- get legal neighbors from known wall map
- apply rule priorities to pick next move
- execute move

2.2 Maze Grid Solver Goal-Based Agent

Goal-Based Agent for Grid Maze Solver 8x8
Goal • Reach (7,7) **Planning Method** • Use BFS (or A*) to compute a shortest path from start to goal. • You can compute the full plan once: ○ plan = shortest_path(start, goal) ○ then execute actions sequentially **Loop per step** 1. Sense and update known_walls 2. Plan shortest path to goal with BFS/A* 3. Take first move of the plan 4. Repeat **Implementation procedure** 1. Build graph: for each cell, connect to neighbor cells that are not blocked by walls 2. Run BFS / A* 3. Get path as list of cells: [(0,0), (0,1), ... , (N-1,N-1)] 4. Convert path into actions 5. Execute actions in order

2.3 Maze Grid Solver Utility-Based Agent

A utility-based agent: (1) uses a utility function to compare outcomes, (2) chooses actions that maximize (expected) utility, and (3) may or may not explicitly plan. In known deterministic mazes, expected utility simplifies to just "utility of resulting state."

Utility-Based Agent for Grid Maze Solver 8x8
Utility function Example features for a candidate next cell s': • d_goal(s') = shortest-path distance from s' to goal (precompute once with BFS from goal) • visited_penalty if revisiting **Planning Method** • Run BFS (or A*) on the current known_walls $$U(s') = -d\_goal(s') \; - \; \lambda \cdot \mathbf{1}[\text{visited}(s')] \; - \; \tau \cdot \text{turn\_cost}$$ Where: • λ discourages revisits • τ discourages too many turns (useful in micromouse) **Implementation procedure** 1. Precompute dist_to_goal[r][c] using BFS from goal 2. Loop until goal: ○ for each legal action: ▪ compute next cell ▪ compute U(next) ○ pick max utility action ○ execute

Exercise

1. For the maze grid agent, identify: Performance Measure, Environment, Actuators, and Sensors. Write your answer in PEAS table format.
2. Classify the maze environment with respect to:

    - Fully observable / Partially observable
    - Deterministic / Stochastic
    - Episodic / Sequential
    - Static / Dynamic
    - Discrete / Continuous

3. Given the following behaviors, identify the agent type:

    - Agent follows fixed rules without planning.
    - Agent computes shortest path to goal before moving.
    - Agent selects actions based on numerical scores.

4. Consider the model-based reflex agent and 8×8 maze grid. Change the start cell to (8,1) and the goal cell to (3,8) and identify the following:

    - Sequence of cells visited by the agent.
    - Percepts at each step.
    - Actions taken by agent at each step.

5. With in 8×8 maze grid example consider the utility-based agent and do the followings:
    - Introduce a cost of movement:
        - Forward move = 1
        - Turning left or right = 2
    - Modify the agent's decision logic to include movement cost.
    - Observe how the agent's path changes.
    - At each step now show the action taken by the agent and the total cost.
6. Remove one action (LEFT) from the agent's action set in the same example with goal-based agent and do the following:

    - Update the action list.
    - Run the agent and analyze if it can still reach the goal.