# Homework #1

ECE 462/662: Principles and Engineering Applications of AI
Prof. Ozan Tonguz

Due: Feb 7, 2026

Notes

- Late submissions will not be accepted nor graded.
- Please show your work for all problems and explain reasoning.
- Please note that answers generated by large-scale language models such as ChatGPT are prohibited as submissions.

# Part 1: Conceptual Questions

For the questions below, you may find it useful to read chapter 1 and 2 from *Stuart Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach", 3rd Edition, Prentice Hall.*

## Problem 1 - True or False

**For each of the following assertions, say whether it is true or false and support your answer with examples or counterexamples where appropriate.**

A. An agent that senses only partial information about the state cannot be perfectly rational.
B. There exist task environments in which no pure reflex agent can behave rationally.
C. There exists a task environment in which every agent is rational.
D. The input to an agent program is the same as the input to the agent function.
E. Every agent function is implementable by some program/machine combination.
F. Suppose an agent selects its action uniformly at random from the set of possible actions.
G. There exists a deterministic task environment in which this agent is rational.
H. It is possible for a given agent to be perfectly rational in two distinct task environments.
I. Every agent is rational in an unobservable environment.
J. A perfectly rational poker-playing agent never loses.

## Problem 2 - Define the Performance, Environment, Actuators, and Sensors (PEAS)

**For each of the following activities, give a PEAS description of the task environment and characterize its properties. [Fully vs Partially Observable, Single agent vs Multiagent, Deterministic vs Stochastic, Episodic vs Sequential, Static vs Dynamic, Discrete vs Continuous, Known vs Unknown]**

A. Playing soccer.
B. Exploring the subsurface oceans of Titan.
C. Shopping for used AI books on the Internet.
D. Playing a tennis match.
E. Practicing tennis against a wall.
F. Performing a high jump.
G. Knitting a sweater.
H. Bidding on an item at an auction.

## Problem 3 - Define the terms

In your own words, define the following terms:

A. Intelligence
B. Artificial Intelligence
C. Agent
D. Agent function
E. Agent program
F. Reflex agent
G. Model-based agent
H. Goal-based agent
I. Utility agent
J. Learning agent

## Problem 4 - Pseudocode for agents

Write pseudocode agent programs for the following agents:
(*Hint: you might find it useful to review the diagrams in the lecture notes*)

**A.** ***Simple reflex agent with only five conditional rules, each corresponds to a different action***
**B.** ***Model-based agent with only five conditional rules, each corresponds to a different action***
**C.** ***Goal-based agent***
**D.** ***Utility-based agent***
**E.** ***Learning agent***

# Part 2: Coding Questions

## Problem 1: Reflex Agent Implementation

Let's assume that you work for a company that designs self-driving cars for different kinds of applications. Your latest project is to develop a simple reflex agent (or reflex agent) to do a monitoring task in order to secure a large neighborhood that consists of several apartment buildings. The self-driving car is equipped with different perceptrons that detect any suspicious behavior (using cameras or notifications from the control room) or fires (using smoke sensors). It needs to take appropriate actions based on the specific situation.

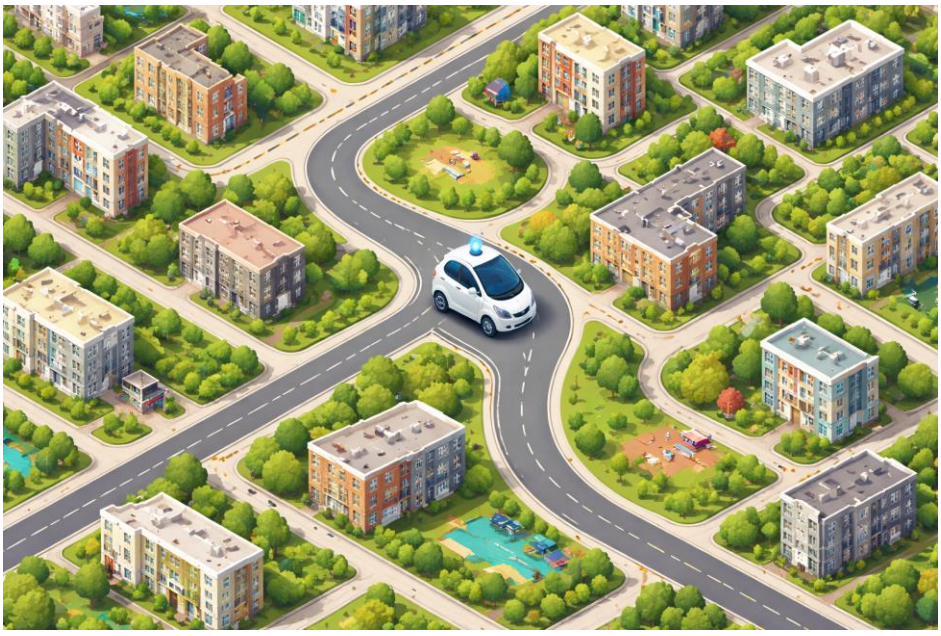

Figure 1. A demonstration of a self-driving car that does patrolling for security purposes.

The self-driving car is able to move around buildings with a speed of 10 m/s. In certain situations, it needs to increase its speed to reach the location of the emergency and take appropriate action. The set of rules for this reflex agent are (in order of priority):

1. If a suspicious behavior (using cameras) or an intruder is detected (and notified by the command center), it should emit an alarm and move to the specific location to secure the area.

2. If a fire is detected, the agent should call the fire department if a certain limit in the sensor is reached.
3. If multiple emergencies are detected, the agent should prioritize the emergencies as follows: intruders, suspicious behavior, and fires, in that order.
4. If no emergency is detected, the self-driving car should continue patrolling the neighborhood.

Write a Python function **self_driving_car** that returns a string that describes the action taken by the agent for the occurring situation as per the rules stated above (in order of priority)
As a sanity check for your implementation, find a sample output of how your design should give to the user (yours does not have to exactly the same as long as it works correctly):

```
self_driving_car()

**********************************************************************************************************

Continue patrolling the neighborhood ...
Sensors inputs ('shutdown' to turn off the agent): bjgnfd
**********************************************************************************************************

No emergency detected or noisy input to sensor
Continue patrolling the neighborhood ...

**********************************************************************************************************

Continue patrolling the neighborhood ...
Sensors inputs ('shutdown' to turn off the agent): fire
smoke sensor limit reached? yes/no:  no
**********************************************************************************************************
limit in sensor is not reached, no need to call fire department
**********************************************************************************************************

Continue patrolling the neighborhood ...
Sensors inputs ('shutdown' to turn off the agent): fire suspicious_behaviour intruder
smoke sensor limit reached? yes/no:  yes
**********************************************************************************************************
Emit alarm and move to location of intruder
**********************************************************************************************************

Continue patrolling the neighborhood ...
Sensors inputs ('shutdown' to turn off the agent): shutdown
**********************************************************************************************************

the agent will be off now

**********************************************************************************************************
```

After completing this question, answer the followings:

A. Is it a good design to formulate self-driving cars as simple reflex agents? Why?
B. In your words, what are the potential points of failures of this design? How can we improve it?
C. What other agent design would you prefer instead of this one? Why?

# Problem 2: Utility-Based Content Recommendation Agent

Design a utility-based agent that recommends video content (or articles/courses) to users based on their viewing history, interests, and engagement patterns. The goal is to calculate relevance scores for content items based on specific factors and recommend the top k items that maximize the user's engagement likelihood. See Skeleton code: hw1_content.py on canvas

**What we should see in your submission:**

**1. Implement a function recommend_content(user_profile, content_items, k):**

• user_profile: A dictionary containing user interests, viewing history, and engagement metrics.

```
{
   "history": {"video1": 0.85, "video2": 0.60},
   "interests": {"AI": 8, "programming": 5, "cooking": 3},
   "avg_watch_time": 15.0
}
```

Note: "history" maps content IDs to completion rates (0.0-1.0), "interests" maps topics to engagement counts, and "avg_watch_time" is in minutes.

• content_items: A list of dictionaries, where each dictionary represents a content item with attributes like topics, views, likes, duration, and recency.

```
[
   {"id": "video1", "topics": ["AI", "programming"], "views": 50000,
    "likes": 3500, "duration": 18, "days_since_upload": 5},
   {"id": "video2", "topics": ["cooking", "recipes"], "views": 120000,
    "likes": 8000, "duration": 12, "days_since_upload": 30}
]
```

Note: "duration" is in minutes, "topics" is a list of topic tags.

• k: The number of recommendations to return.

**2. Compute relevance scores for each content item based on alignment with user interests, engagement history, content popularity, and duration compatibility. Justify your choices in your submission.**

**3. Sort the content items by their relevance scores and return the top k recommendations.**

**4. Provide a clear explanation of:**

- The logic behind your relevance score calculation.
- Why you chose specific factors and their weights.
- Examples or test cases showing the input, the calculated relevance scores, and the final recommendations.

## Skeleton Code (hw1_content.py)

```
from typing import Dict, List, Tuple



def recommend_content(user_profile: Dict[str, any], content_items: List[Dict], k: int) -> List[Tuple[str, float]]:
    """
    Recommend top k content items based on user engagement history and interests.

    Args:
        user_profile (dict): User's viewing history, interests, and engagement metrics.
        content_items (list): List of content item dictionaries.
        k (int): Number of recommendations to return.

    Returns:
        list: Top k recommended content items with relevance scores.
    """
    # Extract user interests and engagement history
    # TODO: Extract required attributes and calculate the relevance scores.
    pass



# Step 9: Define a helper function `calculate_relevance`
def calculate_relevance(user_history: Dict[str, float], user_interests: Dict[str, int],
                        avg_watch_time: float, item: Dict) -> float:
    """
    Calculate relevance score for a content item based on user engagement patterns.

    Args:
        user_history (dict): User's past viewing completion rates (item_id -> completion_rate).
        user_interests (dict): User's topic interests with engagement counts.
        avg_watch_time (float): User's average watch time in minutes.
        item (dict): Content item details.
```

```
    Returns:
        float: Relevance score for the content item.
    """
    # TODO: Implement your relevance scoring logic here

    pass



#Test the function
# Use sample user_profile and content_items to verify the output.
if __name__ == "__main__":
    user_profile = {
        "history": #TODO,
        "interests": #TODO,
        "avg_watch_time": #TODO  # minutes
    }

    test_content_items = [
        {"id": "video1", "topics": ["cooking", "recipes"], "views": 50000, "likes": 3000,
         "duration": 20, "days_since_upload": 5},
        {"id": "video2", "topics": ["AI", "machine_learning"], "views": 120000, "likes": 8000,
         "duration": 18, "days_since_upload": 30},
        {"id": "video3", "topics": ["programming", "python"], "views": 80000, "likes": 5500,
         "duration": 12, "days_since_upload": 10},
        {"id": "video4", "topics": ["travel", "vlog"], "views": 200000, "likes": 15000,
         "duration": 25, "days_since_upload": 60},
        {"id": "video5", "topics": ["AI", "data_science"], "views": 95000, "likes": 7200,
         "duration": 14, "days_since_upload": 3}
    ]

    # Get top k recommendations
    recommendations = #TODO
    print("Recommended Content:", recommendations)
```

# Problem 3: Model-Based Agent (Exploration of the Martian Lava Tubes)

**Objective:**

The goal of this assignment is to move beyond simple reflex agents to a Model-Based Agent. You will demonstrate how an agent can maintain an internal "mental map" to navigate a partially observable environment safely.

**The Scenario:**

Your rover is exploring a Martian lava tube. Unlike previous exercises, the environment is Partially Observable: the rover can only see the (7 x 7) grid area immediately in front of its sensors. If the rover enters a "Lava" cell, the mission fails and the environment resets.

Step 1: Setup

1. Install the required libraries: pip install minigrid gymnasium.
2. Initialize the environment: MiniGrid-LavaGapS7-v0.

Step 2: Task Requirements

1. PEAS Description:

Write a formal PEAS description (Performance, Environment, Actuators, Sensors) for this specific task.

2. The Internal Model:

Define a class ModelBasedRover. This class must store a 2D NumPy array (the "Model") that persists across environment resets.

- Value 0: Unexplored.
- Value 1: Safe (Floor).
- Value -1: Hazard (Lava).

3. Coordinate Mapping:

The environment provides a relative observation (what the rover sees). You must write a function to translate these relative observations into Global Coordinates on your 2D map based on the rover's current position and heading.

4. Exploration Strategy: The agent should move toward "Unexplored" (0) cells.

- If "Lava" is detected in the field of view, mark that global coordinate as **-1**.
- Critical: Upon mission reset, the agent must use its stored map to navigate. It should never step into a coordinate previously marked as **-1**.

(Starter code link: Starter codes - HW1 (Martian).ipynb)

**Deliverables:**

Code: Your implementation of the mapping and navigation logic.

Analysis: A short paragraph explaining how your agent handles the transition from "Ego-centric" (relative) view to "Global" coordinates.

Result: A visualization or plot of your rover's internal map after it successfully reaches the goal for the third time.

# Problem 4: Utility-Based Agent (Autonomous Science Prioritization)

**Objective:**

This problem requires you to implement a Utility-Based Agent that makes "rational" decisions by weighing competing objectives: the scientific value of a sample versus the energy cost required to retrieve it.

**The Scenario:**

The rover is in a 10x10 fully observable grid. There are 5 science targets (T1…T5) available. However, the terrain is uneven. Your rover has a limited energy budget and must choose the most "valuable" path, which is not always the shortest path.

Step 1: Data Definitions

- Target Value (V): Each target has a priority score from 1 (Low) to 10 (High).
- Terrain Cost (C):

  Bedrock (Grey cells): 1 energy unit per step.

  Soft Sand (Yellow cells): 5 energy units per step.

Step 2: Task Requirements

1. Utility Function Design:

Create a mathematical function U(target) to score each target.

Example: U = Value/PathCost or U = Value - (weight x PathCost)

You must state your chosen formula and explain why it represents a "rational" trade-off.

2. Cost-Sensitive Pathfinding:

Implement Dijkstra's Algorithm to find the path to each target.

Important: Your algorithm must minimize the Sum of Terrain Costs, not the number of steps.

3. The Decision Loop:

- Evaluate the Utility (U) of all 5 targets from your current position (0, 0).
- Move to the target with the highest Utility (U).
- From that new position, re-calculate the utility for the remaining 4 targets and repeat until 3 samples are collected.

(Starter code link: Starter codes - HW1 (Martian).ipynb)

**Deliverables:**

Implementation: Python code for the UtilityFunction and the CostSensitiveDijkstra

Execution Trace:

1. A log showing:
2. The sequence of targets chosen.
3. The total energy spent.

Justification: Find one instance in your results where the rover ignored a high-value target because it was "behind" a patch of Soft Sand. Explain why this was the correct decision based on your utility function.