AI MODULE 2
Intelligent Agents

1] Introduction of agents

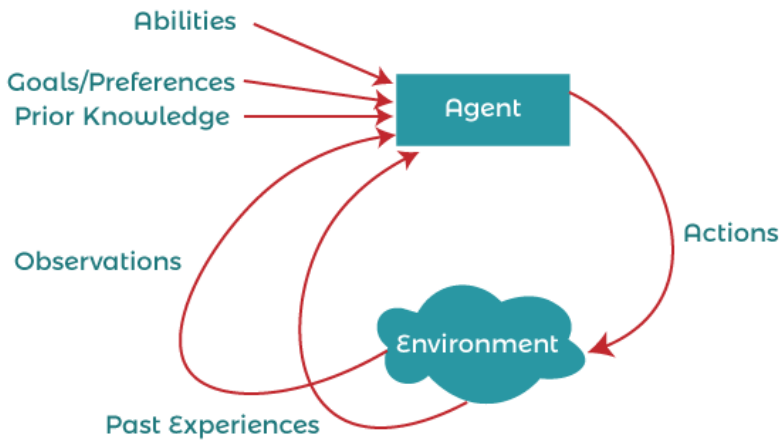
An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators

Agents operate autonomously, making decisions and taking actions based on their perceptions and internal state.

• Human agent: eyes, ears, and other organs for sensors;

hands, legs, mouth, and other body parts for actuators

• Robotic agent: cameras and infrared range finders for sensors; various motors for actuators



The structure of Agent

Agent = Architecture + Program

- **Agent**: The entity that perceives its environment through sensors and acts upon that environment through effectors.

1. **Architecture**: The hardware or platform on which the agent program runs.
    - **Function**: Provides the necessary physical infrastructure for sensing, processing, and action.
    - **Examples**: Robots, computers, mobile devices, servers, etc.
    - **Components**:
        - **Sensors**: Devices that gather information from the environment.

        - **Effectors**: Devices that perform actions in the environment.
        - **Processing Unit**: Hardware that runs the agent program.
2. **Agent Program**: The software that implements the logic of the agent.
    - **Function**: Maps percepts (inputs) to actions (outputs) based on the goals of the agent.
    - **Components**:
        - **Percept Processing**: Interprets the raw data from sensors.
        - **Decision Making**: Determines the best actions to take based on the current state and goals.
        - **Action Generation**: Issues commands to the effectors.

2] Structure of an Intelligent Agent

An intelligent agent is designed to perceive its environment and take actions to achieve specific goals. The structure of an intelligent agent typically includes the following components:

1. **Sensors**:
    - **Function**: Gather information from the agent's environment.
    - **Examples**: Cameras, microphones, touch sensors, or data inputs in software agents.
2. **Actuators**:
    - **Function**: Perform actions within the environment based on the agent's decisions.
    - **Examples**: Motors, displays, speakers, or outputs in software agents (like sending a message or executing a command).
3. **Percept Sequence**:
    - **Function**: A history of all that the agent has perceived so far. This sequence helps the agent to make informed decisions based on past perceptions.
4. **Agent Function**:
    - **Function**: Maps the current percept (or percept sequence) to an action. It determines what the agent will do next.
    - **Examples**: If the agent perceives an obstacle, it might decide to turn.
5. **Internal State**:
    - **Function**: Maintains the agent’s knowledge about the world and itself. It helps the agent keep track of past states and information that may not be immediately observable.
    - **Examples**: Memory of previous locations visited, learned knowledge, or current goals.
6. **Goals**:

- **Function**: Specific objectives that the agent aims to achieve.
- **Examples**: Finding the shortest path in a maze, maximizing rewards in a game, or completing a task.

7. **Utility Function**:
    - **Function**: Measures the agent's performance in terms of the desirability of different states. It helps in making decisions that maximize the expected utility.
    - **Examples**: Scoring systems in games, cost-benefit analysis in decision-making, or happiness levels in social robots.
8. **Learning Component**:
    - **Function**: Allows the agent to improve its performance over time based on experience.
    - **Examples**: Machine learning algorithms, reinforcement learning, and adaptation mechanisms.

Example: Structure of a Self-Driving Car (as an Intelligent Agent)

- **Sensors**: Cameras, LiDAR, GPS, accelerometers.
- **Actuators**: Steering wheel, accelerator, brakes.
- **Percept Sequence**: Continuous stream of data from sensors about the car's position, speed, and surroundings.
- **Agent Function**: Algorithm that processes sensor data to decide on acceleration, braking, and steering.
- **Internal State**: Current speed, direction, route, and memory of past obstacles or traffic conditions.
- **Goals**: Reach a destination safely and efficiently.
- **Utility Function**: Minimize travel time and energy consumption while maximizing safety.
- **Learning Component**: Use of data from driving experiences to improve navigation and control algorithms.

3] Characteristics of Intelligent Agents

Intelligent agents possess several key characteristics that enable them to operate autonomously and effectively in dynamic environments. Here are the main characteristics:

1. **Autonomy**:
    - **Definition**: The ability to operate without direct human intervention.
    - **Example**: A self-driving car can navigate roads and make driving decisions without human input.

2. **Reactivity**:
    - **Definition**: The ability to perceive the environment and respond to changes promptly.
    - **Example**: A thermostat adjusts the heating system in response to temperature changes.
3. **Adaptability**:
    - **Definition**: The ability to learn from experiences and improve performance over time.
    - **Example**: A recommendation system that personalizes suggestions based on user preferences and behavior.
4. **Goal-Oriented Behavior**:
    - **Definition**: The ability to act in a manner that is directed towards achieving specific objectives.
    - **Example**: A robot vacuum cleaner that maps out and cleans a room to ensure all areas are covered.
5. **Rationality**:
    - **Definition**: The ability to make decisions that maximize the expected outcome based on the available information.
    - **Example**: A trading algorithm that buys and sells stocks to maximize profit.
6. **Mobility (for physical agents)**:
    - **Definition**: The ability to move and operate in different physical locations.
    - **Example**: An autonomous drone that can navigate through various terrains and environments.
7. **Perception**:
    - **Definition**: The ability to gather information from the environment through sensors.
    - **Example**: A surveillance system that uses cameras and microphones to monitor an area.
8. **Action**:
    - **Definition**: The ability to perform actions that affect the environment.
    - **Example**: An industrial robot that assembles parts on a production line.

4] TYPES OF AGENTS

Simple Reflex Agent

A simple reflex agent operates by selecting actions based solely on the current percept, ignoring the rest of the percept history. It follows a set of condition-action rules, where each rule maps a condition (based

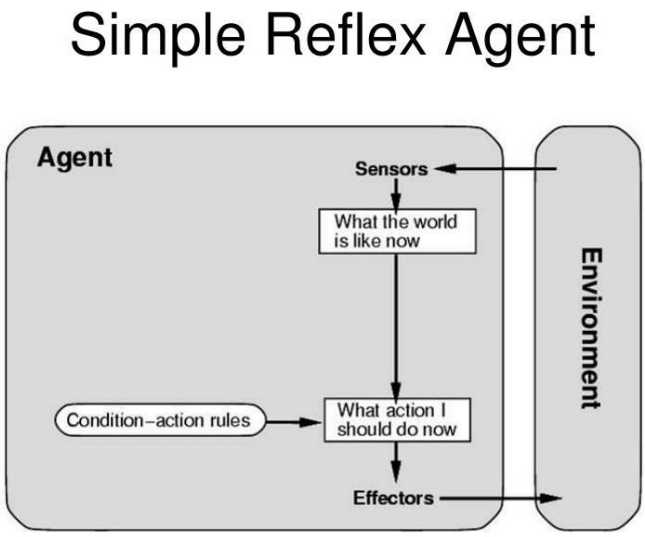
on the current percept) directly to an action. This type of agent is straightforward and does not maintain any internal state about the environment.

Structure of a Simple Reflex Agent:

1. **Sensors**: Detect the current percept from the environment.
2. **Condition-Action Rules**: A set of predefined rules that dictate what action to take based on the current percept.
3. **Actuators**: Perform the action determined by the condition-action rules.

Characteristics:

- **Reactivity**: Responds immediately to changes in the environment.
- **No Memory**: Does not store any history or internal state; each decision is made based on the current percept alone.
- **Simplicity**: Easy to design and implement, but limited in capability.



Function:

function SiMPLE_REFLEX_AGENT(Percept)returns action

def SIMPLE_REFLEX_AGENT(percept):

    state = INTERPRET_INPUT(percept)

    rule = RULE_MATCH(state, rules)

    action = RULE_ACTION(rule)

    return action

Limitations:

- **Limited Scope**: Can only handle environments that are fully observable and where the right action can always be determined from the current percept.
- **No Learning**: Cannot improve its performance based on experience.
- **Inflexibility**: Struggles with tasks that require planning, remembering past actions, or adapting to new situations.

Applications:

Simple reflex agents are best suited for straightforward tasks where the correct action can be determined directly from the current percept

Example: Vacuum Cleaner Agent

- **Environment**: A grid with dirt in some cells.
- **Percepts**: The current cell's status (clean or dirty).
- **Condition-Action Rules**:
    - If the current cell is dirty, then clean it.
    - If the current cell is clean, then move to the next cell.
- **Actuators**: The vacuum motor (to clean) and the wheels (to move).

```
def simple_reflex_agent(percept):
  if percept == "dirty":
    action = "clean"
  else:
    action = "move"
  return action
```

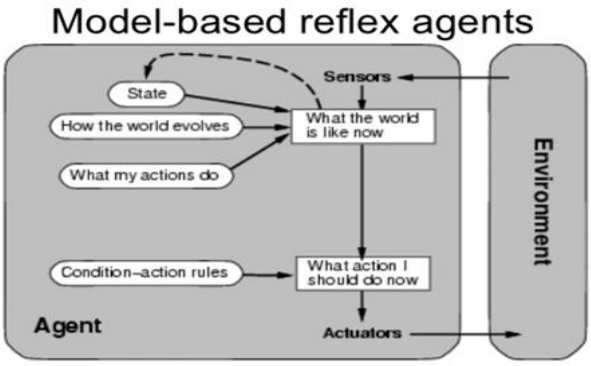
Model-Based Agents

Model-based agents are more sophisticated than simple reflex agents because they maintain an internal model of the world. This model helps the agent keep track of the unobservable aspects of the environment and predict the consequences of its actions.

Structure of a Model-Based Agent

1. **Sensors**: Gather percepts from the environment.
2. **Internal Model**: Represents the agent's knowledge about the world, including both observable and unobservable aspects.
3. **State**: The current state is determined using the internal model and the new percept.
4. **Goals**: Desired outcomes that the agent aims to achieve.

5. **Utility Function**: Measures the desirability of different states.
6. **Condition-Action Rules**: Determine actions based on the current state and goals.
7. **Actuators**: Perform actions in the environment.



Function:

Function Reflex agent with state returns an action

```
def MODEL_BASED_AGENT(percept):
    state = UPDATE_STATE(percept, internal_model)
    action = SELECT_ACTION(state, goals, utility_function)
    internal_model = UPDATE_MODEL(internal_model, action, percept)
    return action
```

Action may depend on history or unperceived aspects of the world.

− Need to maintain internal world model.

Limitations of Model-Based Agents

1. **Complexity**: Maintaining and updating an internal model can be computationally intensive and complex, especially in dynamic and unpredictable environments.
2. **Incomplete or Inaccurate Models**: The internal model may not always accurately reflect the real world, leading to suboptimal decisions.
3. **Scalability**: As the environment becomes more complex, the internal model can become too large to manage efficiently.

Example of Model-Based Agents

A smart grid is an electrical grid that uses information and communication technology to optimize the production, distribution, and consumption of electricity. Model-based agents in smart grids help manage

energy resources efficiently by predicting demand, coordinating distributed energy resources, and optimizing energy usage.

Components of a Smart Grid Energy Management System

1. **Sensors**: Measure electricity usage, production, and storage across the grid.
2. **Internal Model**: Represents the current state of the grid, including energy demand, supply, and storage levels.
3. **State**: Updated continuously with real-time data from sensors.
4. **Goals**: Ensure a reliable supply of electricity, minimize costs, and integrate renewable energy sources.
5. **Utility Function**: Measures the efficiency, cost-effectiveness, and reliability of the grid.
6. **Condition-Action Rules**: Determine actions based on the current state, goals, and utility function.
7. **Actuators**: Control devices such as switches, circuit breakers, and energy storage systems.

Goal-Based Agents

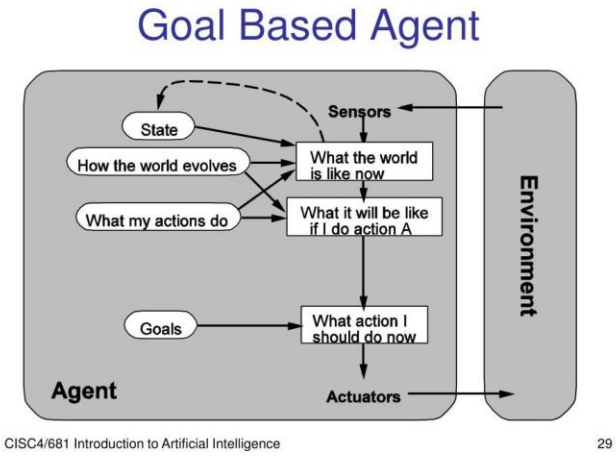
Knowing about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, right, or go straight on. The right decision depends on where the taxi is trying to get to.

Goal-based agents are designed to achieve specific goals by taking actions that lead to desired outcomes. Unlike simple reflex agents and model-based agents, goal-based agents have explicit goals that guide their decision-making process. They consider future states and determine actions that will bring them closer to achieving their goals.

Reflex agent use built in rule to map percept to action, Goal based could reason

Structure of a Goal-Based Agent

1. **Sensors**: Collect percepts from the environment.
2. **State**: Represents the current situation based on percepts.
3. **Goals**: Specific desired outcomes the agent aims to achieve.
4. **Action Selection**: Determines the best action to achieve the goals.
5. **Actuators**: Execute the chosen actions in the environment.



Function:

```
def action(agent, environment):
    state = agent.sense(environment)
    for action in agent.available_actions:
        resulting_state = agent.do(action, state)
        if resulting_state == agent.goal:
            return action
```

Example: Autonomous Delivery Drone

Scenario

An autonomous delivery drone aims to deliver packages to specific locations. The drone's goal is to deliver the package to the destination efficiently.

Components and Functions

1. **Sensors**: GPS for location, cameras for obstacle detection, and battery level sensors.
2. **State**: Current location, battery level, and detected obstacles.
3. **Goals**: Deliver the package to the destination.
4. **Action Selection**: Choose the best route and maneuvers to reach the destination while avoiding obstacles and managing battery usage.

Goal-based agents use explicit goals to guide their decision-making process, allowing them to plan and execute actions that lead to desired outcomes. They are widely used in various applications, from

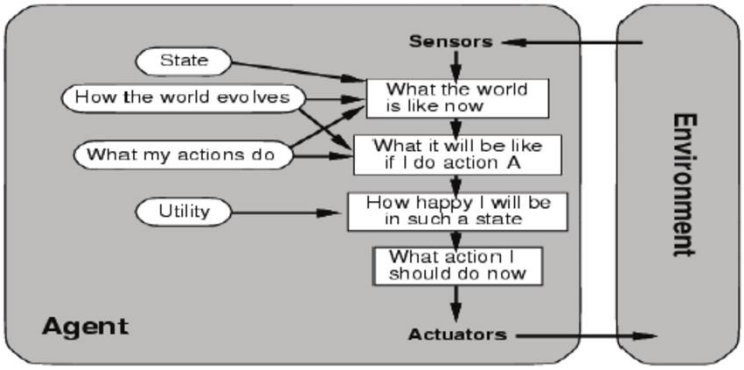
autonomous vehicles to personal assistants, but they face challenges related to goal management, computational complexity, and adaptability to dynamic environments.

Utility-Based Agents

Utility-based agents enhance goal-based agents by considering not just whether an action leads to a goal, but also how desirable the resulting state is. They use a utility function to measure the desirability of different states and select actions that maximize this utility.

Structure of a Utility-Based Agent

1. **Sensors**: Gather percepts from the environment.
2. **State**: Represents the current situation based on percepts.
3. **Goals**: Desired outcomes the agent aims to achieve.
4. **Utility Function**: Measures the desirability of states, providing a way to compare different possible outcomes.
5. **Action Selection**: Determines the best action based on maximizing utility.
6. **Actuators**: Execute the chosen actions in the environment.



Function:

```
def UTILITY_BASED_AGENT(percept):
    state = UPDATE_STATE(percept)
    actions = GET_AVAILABLE_ACTIONS(state)
    best_action = None
    max_utility = float('-inf')
    for action in actions:
```

```
        resulting_state = APPLY_ACTION(state, action)
        utility = CALCULATE_UTILITY(resulting_state)
        if utility > max_utility:
            max_utility = utility
            best_action = action
    return best_action
```

Example: Autonomous Trading Agent

Scenario

An autonomous trading agent aims to maximize profit by buying and selling stocks. The agent uses a utility function to evaluate the desirability of different trading actions based on potential profit.

Components and Functions

1. **Sensors**: Collect market data, including stock prices, trading volumes, and economic indicators.
2. **State**: Current market conditions and portfolio status.
3. **Goals**: Maximize profit.
4. **Utility Function**: Measures potential profit from trading actions.
5. **Action Selection**: Choose the trading action that maximizes utility.

Utility-based agents enhance decision-making by considering the desirability of different states using a utility function. They are applied in various fields, including finance, healthcare, autonomous vehicles, and smart home systems, to optimize actions and achieve the best possible outcomes.

5] Types of Environments in AI

The environment in which an agent operates significantly influences its design and functionality. Here are the main types of environments in AI, characterized by different attributes:

1. **Fully Observable vs. Partially Observable**
2. **Deterministic vs. Stochastic**
3. **Episodic vs. Sequential**
4. **Static vs. Dynamic**
5. **Discrete vs. Continuous**
6. **Single-Agent vs. Multi-Agent**

1. Fully Observable vs. Partially Observable

- **Fully Observable**: The agent has access to the complete state of the environment at each point in time.
    - **Example**: Chess, where the board and all pieces are fully visible to both players.
- **Partially Observable**: The agent has only partial information about the environment's state, often due to limited sensors or occlusions.
    - **Example**: Poker, where players cannot see each other's cards.

2. Deterministic vs. Stochastic

- **Deterministic**: The next state of the environment is entirely determined by the current state and the actions taken by the agent.
    - **Example**: Puzzle-solving, where each move deterministically changes the state of the puzzle.
- **Stochastic**: The next state of the environment is determined by probabilities, introducing elements of chance.
    - **Example**: Board games with dice, where outcomes depend on dice rolls.

3. Episodic vs. Sequential

- **Episodic**: The agent's experience is divided into discrete episodes, each with a clear beginning and end. Actions in one episode do not affect future episodes.
    - **Example**: Image classification tasks, where each image is classified independently of others.
- **Sequential**: Current actions affect future states and decisions, requiring the agent to consider long-term consequences.
    - **Example**: Self-driving cars, where each driving decision affects subsequent driving conditions.

4. Static vs. Dynamic

- **Static**: The environment remains unchanged while the agent is deliberating on its next action.
    - **Example**: Crosswords or Sudoku puzzles, where the puzzle doesn't change as you solve it.
- **Dynamic**: The environment can change while the agent is making decisions, requiring continuous monitoring.
    - **Example**: Real-time strategy games, where the game state continuously evolves.

5. Discrete vs. Continuous

- **Discrete**: The environment consists of a finite number of distinct states and actions.
    - **Example**: Chess, where there are a finite number of moves at any given time.

- **Continuous**: The environment has a continuous range of states and actions.
    - **Example**: Robotic arm movement, where positions and movements are not limited to discrete values.

6. Single-Agent vs. Multi-Agent

- **Single-Agent**: The environment contains only one agent operating independently.
    - **Example**: Solving a maze with one robot.
- **Multi-Agent**: The environment contains multiple agents, which may be cooperative or competitive.
    - **Example**: Soccer games, where multiple players (agents) interact and compete.

Examples and Applications

1. **Self-Driving Cars (Partially Observable, Stochastic, Sequential, Dynamic, Continuous, Multi-Agent)**:
    - Must navigate with incomplete information about other vehicles' intentions.
    - Must handle uncertain events like pedestrians suddenly crossing the street.
    - Actions taken now affect future driving conditions.
    - The environment continuously changes with moving traffic.
    - Decisions are made on a continuous spectrum (e.g., adjusting steering angles).
    - Multiple agents (other cars, pedestrians) interact in the environment.
2. **Chess (Fully Observable, Deterministic, Sequential, Static, Discrete, Single-Agent/Multi-Agent)**:
    - The entire board is visible to both players.
    - Moves are deterministic with known outcomes.
    - Each move affects future moves in the game.
    - The board state doesn't change unless a player makes a move.
    - Moves are discrete and well-defined.
    - Can be single-agent (against a computer) or multi-agent (two players).