Lecture 04

1. Robotic Components Recap

- **Effectors**: Limbs/tools for movement (arms, wheels 🦾 🚗).
- **Perception**: Sensors (cameras 👁️, LiDAR, touch).
- **Control**: "Brain" layers (fast reflexes + long-term planning 🧠).
- **Power**: Energy source (batteries 🔋).
- **Communication**: Data transfer (Wi-Fi, gestures 📡).

---

2. Programming Considerations

- **Decompose Actions**: Split tasks into **behaviors** (e.g., "avoid walls" + "follow light" → Roomba 🧹).
- **Action-Oriented Perception**: Focus sensors *only* on relevant data (e.g., ignore noise when avoiding obstacles 🚧).
- **Behaviors**:
  - **Independent**: Run in parallel (e.g., "walk" + "balance" for a robot 🦿).
  - **Compete/Cooperate**: One behavior can override another (e.g., "stop" overrides "move" when obstacle detected 🛑).

---

3. Automation vs. Autonomy

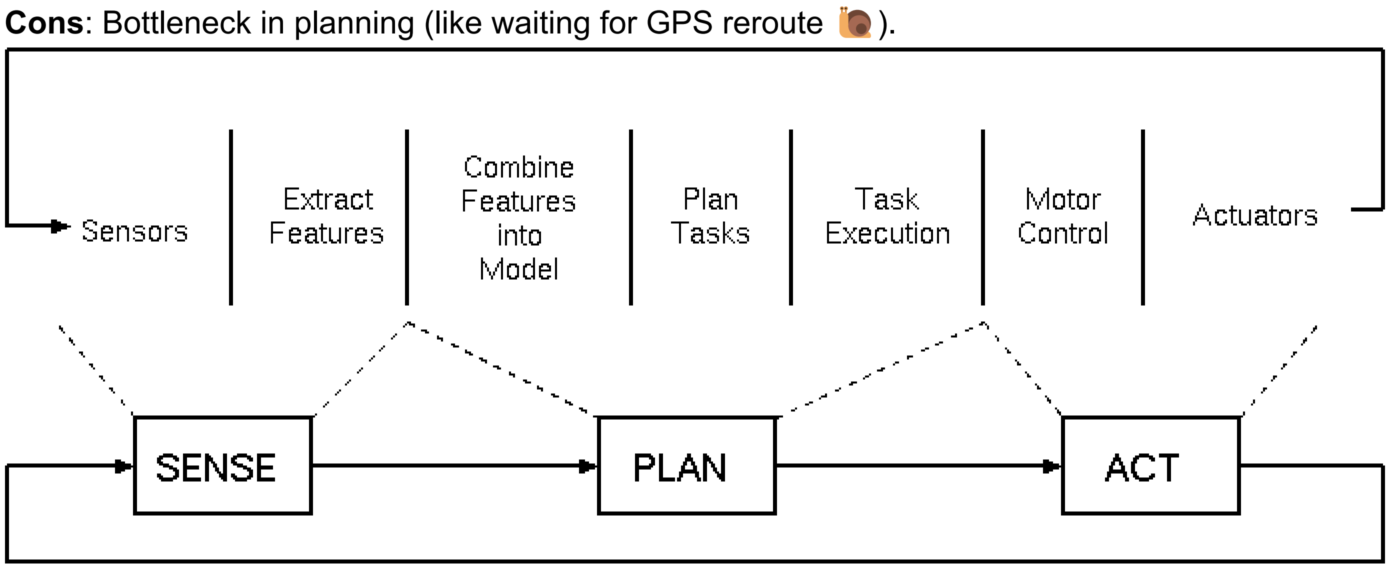
Aspect	Automation (Washing Machine 🧺)	Autonomy (Self-Driving Car 🚗)
Planning	Pre-programmed, repetitive 🔄	Adapts to new situations 🧠
World Model	Closed (everything is known 🔒)	Open (handles unknowns 🌍)
Control	Signals (fixed rules 📟)	Symbols (learned decisions 🧩)

---

4. Robot Reactive

Hierarchical (SPA: Sense → Plan → Act)

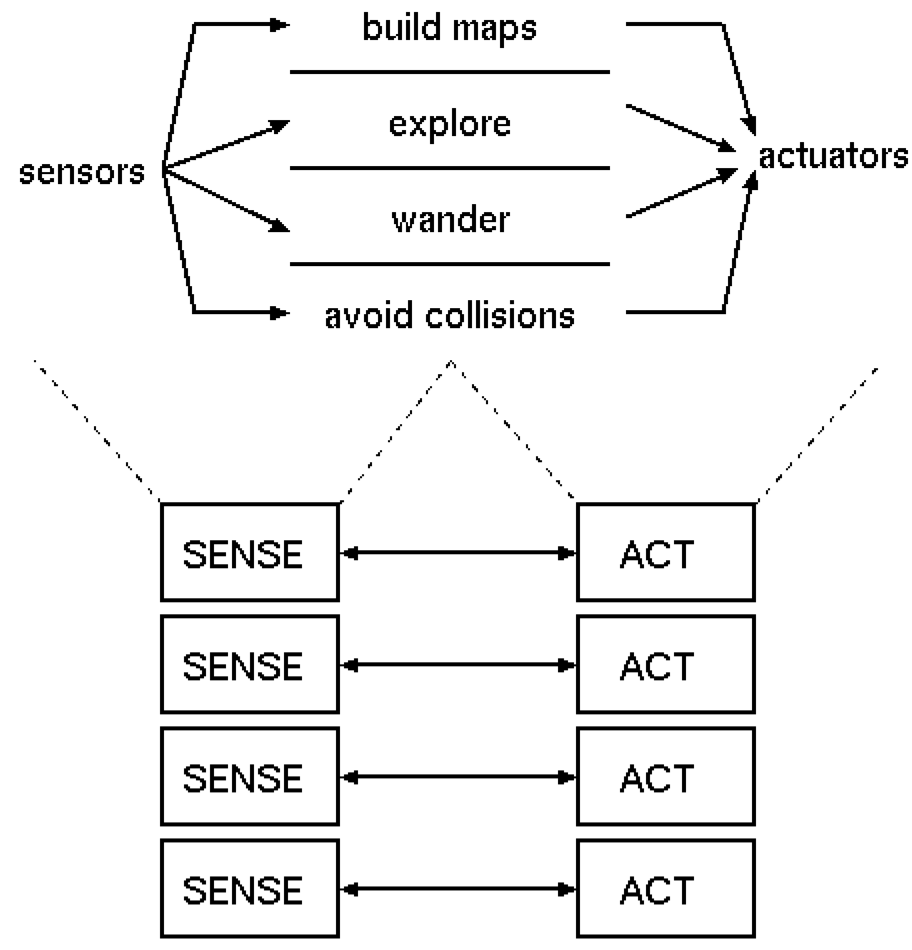
- **Structure**: Slow, global planning (e.g., factory robot arm 🏭).
- **Cons**: Bottleneck in planning (like waiting for GPS reroute 🐌).



Reactive (SA: Sense → Act)

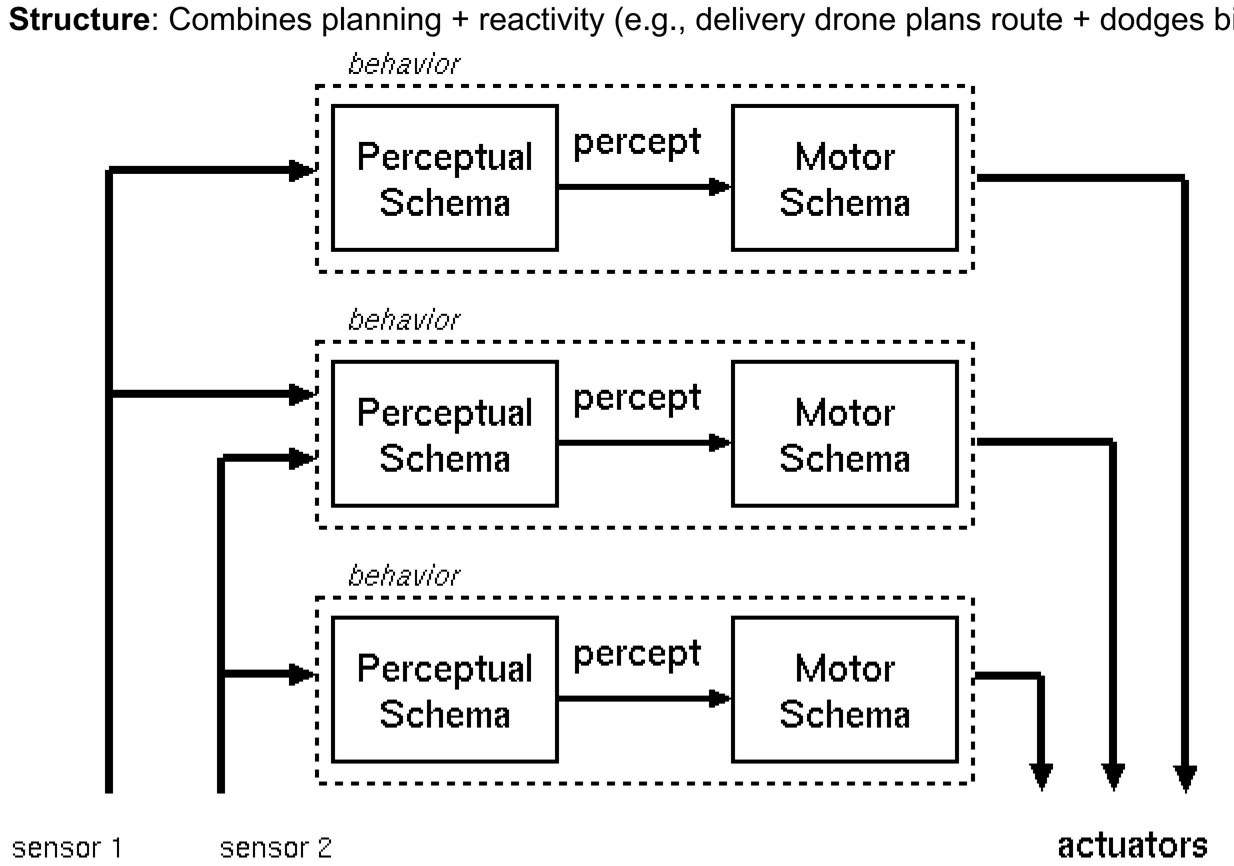
- **Structure**: Fast, no planning (e.g., Roomba avoiding furniture 🚀).
- **Pros**: Real-time responses, simple.

- **Cons**: No long-term strategy (acts on *current* sensors only).



Hybrid (Plan → Sense → Act)

- **Structure**: Combines planning + reactivity (e.g., delivery drone plans route + dodges birds 🛰️).



---

5. Reactive Paradigm Attributes

- **Behaviors**: Direct sensor→action maps (e.g., "if near wall, turn").
- **Local Sensing**: Each behavior uses *dedicated sensors* (e.g., infrared for obstacle avoidance 🔦).
- **No Memory**: Acts only on *current* sensor data (no past/future).

---

6. Key Characteristics of Reactive Systems

1. **Situated in Environment**: Robot is part of the world (e.g., vacuuming changes dust levels 🌪️).
2. **Behaviors = Building Blocks**: No central controller (e.g., ant colony behavior 🐜).
3. **Local Sensing**: Ignore irrelevant data (e.g., focus on light, not sound 💡).
4. **Good Software Design**: Modular, testable (like LEGO blocks 🧱).
5. **Animal-Inspired**: Behaviors mimic nature (e.g., bird flocking 🦅).

---

7. Advantages of Reactive Programming

- **Modularity**: Add/remove behaviors easily (e.g., add "climb stairs" to a robot 🪜).
- **Real-Time**: Fast execution (no planning delays ⚡).
- **Incremental Testing**: Test each behavior separately (e.g., test "balance" before "walk" 🦿).

Cheat Sheet

Term	Key Idea	Example
Automation	Repetitive, closed-world tasks 🔄	Dishwasher 🍻
Autonomy	Adapts to open-world, learns 🌍	Self-driving car 🚙
Reactive Paradigm	SENSE→ACT, no planning 🚀	Roomba avoiding obstacles 🧹
Behavior Modularity	Easy to test/expand 🧩	Adding "detect rain" to a drone 🌧️

إِنَّ اللَّهَ وَمَلَائِكَتَهُ يُصَلُّونَ عَلَى النَّبِيِّ ۚ يَا أَيُّهَا الَّذِينَ آمَنُوا صَلُّوا عَلَيْهِ وَسَلِّمُوا تَسْلِيمًا (56)

---

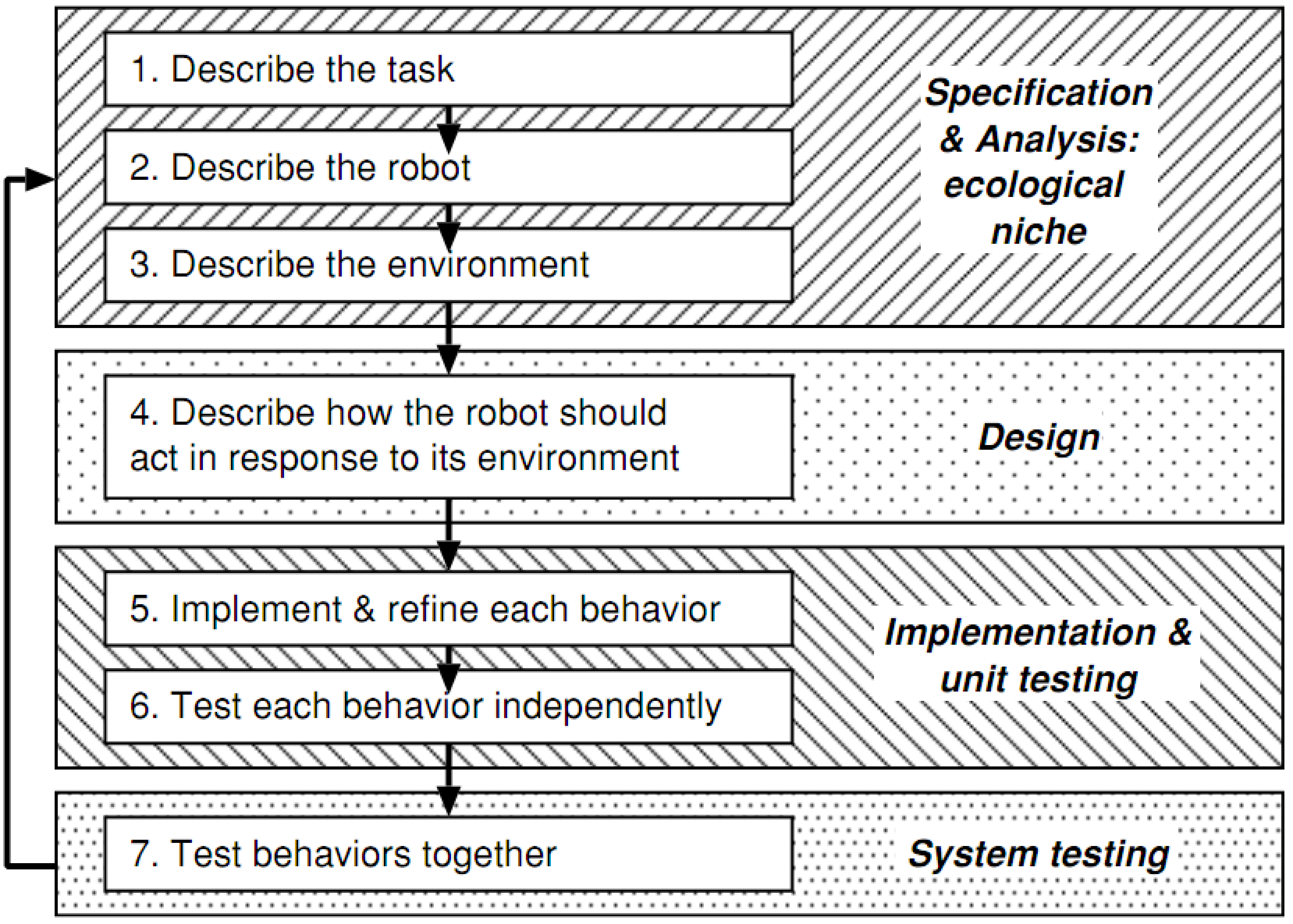
1. Representative Reactive Architectures

Goal: Combine behaviors to achieve complex tasks.
Two main approaches:

1. **Subsumption Architecture** 🏗️
   - **Layers of Competence**: Higher layers *override* lower ones.
     - *Example*:
       - **Level 0**: Avoid obstacles (sonar → turn away 🚧).
       - **Level 1**: Wander randomly (override obstacle avoidance with random direction 🌀).
       - **Level 2**: Follow corridors (override wandering to stay centered 🛤️).
   - **Mechanisms**:
     - **Inhibition** 🚫 : Block lower-layer outputs.
     - **Suppression** 🔀 : Replace lower-layer inputs.
2. **Potential Fields Methodologies** 🧲
   - **How it works**:
     - Behaviors = *vectors* (direction + magnitude ➡️).
     - Combine vectors (sum them) for final movement.
   - *Example*:
     - **Avoid obstacle** = repulsive force (←).
     - **Move to goal** = attractive force (→).
     - **Result**: Robot takes a path around obstacle to goal 🎯 .

---

2. Steps in Designing a Reactive Behavioral System



---

3. Reactive Navigation 🧭

- **No Maps Needed!**
  - **Examples**:
    - Follow a light 💡 (phototaxis).
    - Follow walls 🧱 (bug algorithm).
    - Random walk (Roomba 🧹).
  - **Pros**: Fast, simple.
  - **Cons**: Limited to simple tasks (no long-term planning).

---

4. Robot Joints & DOF 🔩

Joint Types:

Joint	Motion	DOF	Example
Revolute (R)	Rotation 🔄	1	Elbow joint 🤖
Prismatic (P)	Linear ↔	1	Drawer slide 🗄
Screw (H)	Rotate + Translate 🌀	1	Screw in a lid 🧊
Cylindrical (C)	Rotate + Translate along axis	2	Microscope focus knob 🔬
Universal (U)	Two orthogonal rotations ✖	2	Car steering 🚗
Spherical (S)	3D rotation 🌐	3	Shoulder joint 🫀

Degrees of Freedom (DOF):

- **DOF**: Independent movements a robot can make.
  - *Example*: A car 🚗 has 2 DOF (steer + accelerate).
- **Configuration Space (C-space)**: All possible robot poses (positions + orientations).

---

5. Robot Paradigms Recap 🧠

Primitive	Input	Output	Example
SENSE	Sensor data (e.g., distance)	Processed info (e.g., obstacle detected)	Camera detects wall 👁️
PLAN	Sensor/cognitive info	Strategy (e.g., path)	Compute route to goal 🗺️
ACT	Directives (from PLAN)	Motor commands	Turn wheels to avoid obstacle 🛑

---

6. Key Takeaways

- **Subsumption**: Layers override lower behaviors (e.g., wander > avoid obstacles).
- **Potential Fields**: Vector math guides movement (repel + attract).
- **Reactive Navigation**: Simple, no maps (Roomba 🧹 vs. self-driving car 🚗).
- **DOF**: Determines robot's flexibility (e.g., 6-DOF arm 🦾).

Cheat Sheet

Term	Key Idea
Subsumption	Layers override lower behaviors 🏗️
Potential Fields	Vectors sum for movement ➕ ➡️
Revolute Joint	Rotates (1 DOF) 🔄
Reactive Navigation	No maps, direct sensing → action 🧭
