Introduction to Game Programming and Robotics

Unit # 7

Three Laws of Robotics

- The Three Laws of Robotics (often shortened to The Three Laws or Three Laws) are a set of rules devised by the science fiction author Isaac Asimov.
- The rules are introduced in his 1942 short story "Runaround". The Three Laws are:
  - A robot may not injure a human being or, through inaction, allow a human being to come to harm.
  - A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
  - A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Behavior

- Behaviors are the way to implement functionality when using the behavior framework. A behavior represents an action or reaction to events (triggers) and resulting request to control where the robot moves.
- A behavior may have zero or many triggers connected to its input.
- An example of a behavior with no triggers is CRUISE. It processes no input from the robot. It requests to drive the robot given its two internal constants; speed and steering. Every time the behavior runs, the output is the same.

Behavior (Cont'd)

- An example of a behavior implementing a trigger is BUMP ESCAPE.
- The behavior does nothing until the bumpers are depressed. The robot requests to back up for a time, spin right or left, move forward when a collision occurs and relinquishes control when the escape is complete.
- Other examples of a behavior trigger are: an amount of time passing, a random number generator, a sensor value, battery level, or a combination of inputs.
- There are two types of behaviors:
  - Servo
  - Ballistic

Servo Behavior

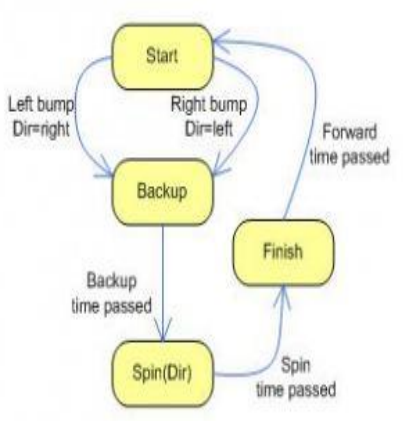
- Typically, a servo behavior employs a feedback control loop as its control component.
- These behaviors may or may not use external input, but provide instant feedback as an output.
- CRUISE is the simplest form of servo behavior. It takes no input and outputs a constant speed and steering.
- These behaviors all process input (or lack thereof) and determine motor commands directly from that input in a stateless feedback/control loop.

Ballistic Behavior

- A ballistic behavior, like a shell fired from a cannon, once triggered, follows a predictable trajectory through to completion.
- In ballistic control, the position trajectory and velocity profile is computed once, then the actuator carries it out.
- There are no “in-flight” corrections, just like a ballistic missile doesn’t make any course corrections.
- In order to accomplish a precise task with ballistic control, everything about the device and how it works has to be modeled and figured into the computation.
- The ballistic control is open-loop control.

Ballistic Behavior (Cont'd)

- Another term for a ballistic behavior is a finite state machine. Ballistic behaviors maintain state between successive calls in order to complete a series of steps before relinquishing control of the robot.
- BUMP ESCAPE is a great example of the ballistic behavior. When the robot collides with an object, BUMP ESCAPE is triggered. The robot backs up for a time, spins, positions forward, and returns to its start state for the next collision.



Ballistic Behavior (Cont'd)

- Complexity in implementing ballistic behaviors stems from maintaining state between loops while still keeping the robot on its mission. Here are a few examples to consider:
  - During the backup state above, a higher order behavior is triggered. It takes control of the robot and then releases control. If the BUMP ESCAPE is triggered again what state should it start in?
  - What if another behavior is triggered in the middle of an escape? How will the robot respond so that it does not crash again or worse, oscillate between a collision and another behavior's action.

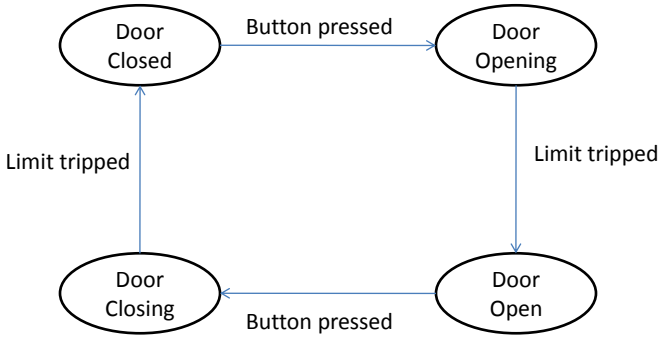
Finite State Machine

- Pure servo behaviors live in the moment.
- Every time such a behavior is called, it computes what to do right now.
- The behavior pays no attention to what it did the last time it was called and it makes no preparation for what it will do next time.
- We say that such behaviors have no state.
- Finite State Machine (FSM) is a type of system that has a limited number of states and has well defined rules specifying how the system is allowed to transition from one state to another.

Garage System

- If the door is closed and I press the open/close button, the door begins moving up.
- When it is all the way open, the door trips a limit switch and motion stops.
- If I push the button when the door is open, it begins moving down.
- As the door reaches the bottom of its travel, it trips another limit switch such that motion stops just when the door is fully closed.

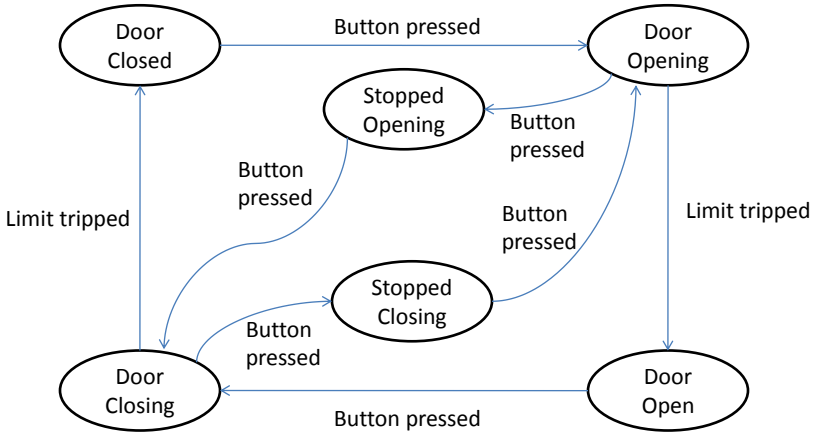
Garage System (Cont'd)



Garage System (Cont'd)

- I don't have to wait until the door reaches its fully up or down position before I press the button again.
- If I push the open/close button while the door is moving, the door halts immediately.
- Pressing the button again from this state makes the door move in the direction opposite to the way it had been going.

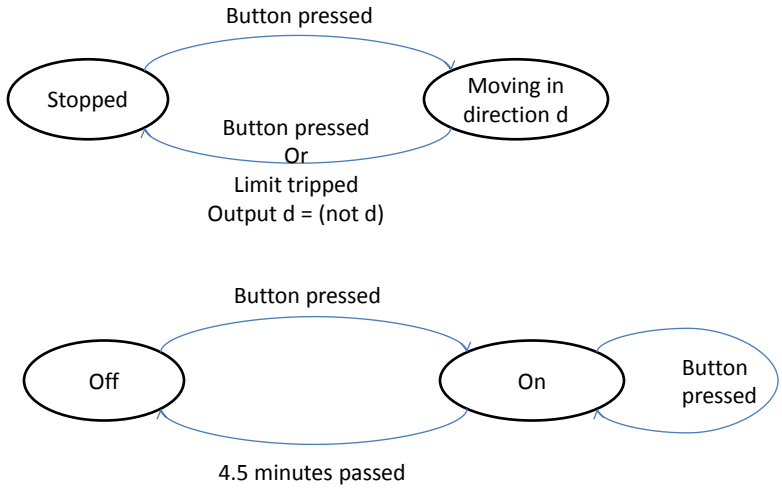
Garage System (Cont'd)



Garage System (Cont'd)

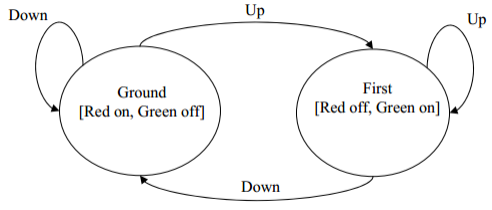
- The garage door opener includes a courtesy light that comes on automatically as soon as I press the open/close button.
- The manufacturer's intent is to give the user time to get into or out of the car at night when, without the courtesy light, the garage would be dark.
- After pressing the open/close button, the light goes on and remains on until 4.5 minutes have passed – then a relay clicks and the light blinks off.
- Further modifications in the FSM would compromise on its readibility.

Simplified FSM

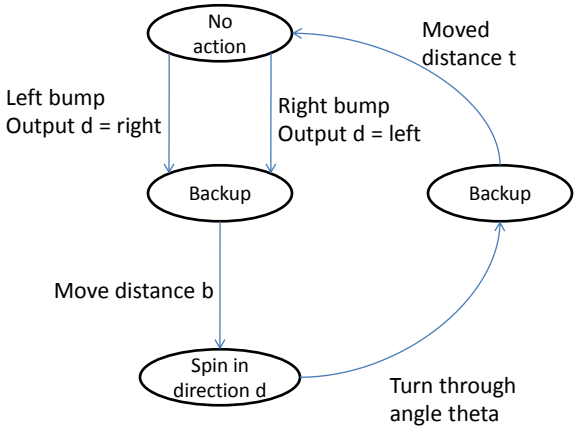


Elevator

- In this example, we'll be designing a controller for an elevator. The elevator can be at one of two floors: Ground or First.
- There is one button that controls the elevator, and it has two values: Up or Down.
- Also, there are two lights in the elevator that indicate the current floor: Red for Ground, and Green for First.
- At each time step, the controller checks the current floor and current input, changes floors and lights in the obvious way.



FSM Example: Escape (Sumo)



FSM for Sumo Wrestling

SURVIVE	The robot enters this state when it detects the dohyo (ring) edge. Its goal is to survive by not going off the ring. The robot rotates away from the sensor that sensed the edge to face back towards the center of the ring.	1. Rotate away from the line sensor that sensed the edge. 2. Switch to HUNT state when rotation is complete.
HUNT	The robot is not at the edge of the ring but hasn't sensed the opponent. The robot moves around in an arcing pattern so its range sensors will sweep across the ring in hopes of sensing the opponent	1. Switch to SURVIVE state if ring edge detected. 2. Switch to TARGET state if range sensors indicate an object ahead. 3. Otherwise, drive in an arc by applying more power to one wheel than the other

FSM for Sumo Wrestling (Cont'd)

TARGET	The opponent has been sensed ahead. Aim the robot to face the opponent directly.	1. Switch to SURVIVE state if ring edge detected. 2. If still sensing opponent, but opponent is not directly ahead turn slightly to aim at opponent. 3. If opponent is directly ahead and close switch to ATTACK state. 4. If opponent is ahead but not close, move straight forward. 5. Otherwise, switch to HUNT state.
ATTACK	The opponent has been found and aiming is complete. Drive straight ahead at full power to push the opponent off the ring.	1. Switch to SURVIVE state if ring edge detected. 2. Otherwise, drive straight forward at full power.

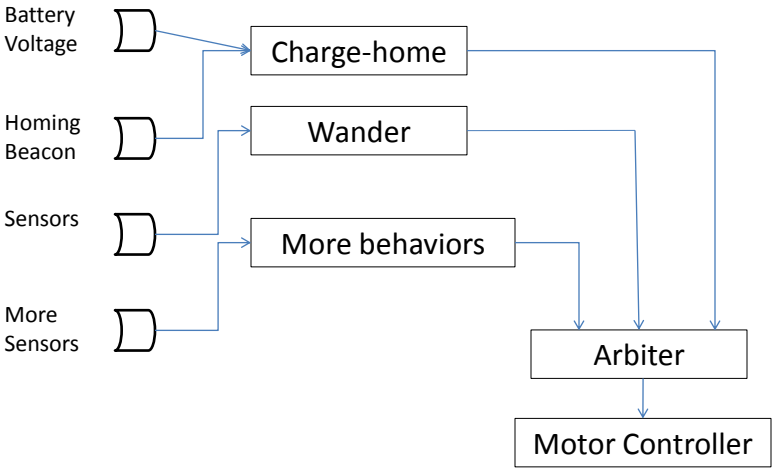
Exercises

- FSM diagram of a flush toilet.
- FSM diagram of a washing machine.

Arbitration

- A primitive behavior consists of a control system that makes the robot act in a certain way and a trigger to decide when it is appropriate to take such actions.
- As long as only one behavior triggers at a time, things work smoothly.
- But what ensues when two or more behaviors happened to trigger at the same time, each one wanting the robot to take a different action?

Arbiter



Fixed-Priority Arbiter

- The most commonly used arbiter is fixed-priority arbiter.
- Fixed-priority means that the programmer has decided in advance which behavior ought to win any time a conflict occurs.
- Each connection from a behavior to the arbiter is given a unique priority.

Graceful Degradation

- Things never go smoothly for robots operating in the real world. In particular
  - A command intended to direct the robot to move in a particular way instead, because of uncontrollable environmental effects, causes the robot to move in a different way.
  - The robot's program makes an assumption about the world that turns out not to be true.
  - The robot's sensors fail outright or produce false negative or false positive results.

Graceful Degradation (Cont'd)

- Note:
  - A false negative means that the sensor did not react when it should have.
  - A false positive occurs when a sensor reports a condition that does not exist.
- Regardless of all this, we want our robots to soldier on.
- The ability of a system to continue to perform at a reduced level in the presence of subsystem errors and failures is known as *graceful degradation*.

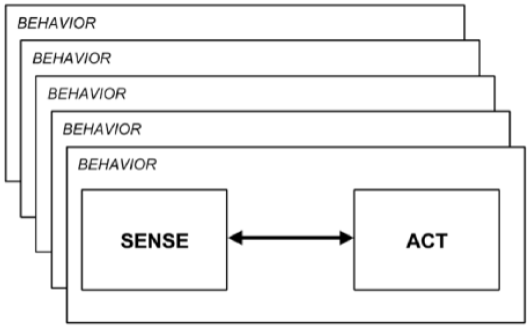
Reactive Paradigm

- The fundamental attribute of the reactive paradigm is that all actions are accomplished through behaviors.
- As in ethological systems, behaviors are a direct mapping of sensory inputs to a pattern of motor actions that are then used to achieve a task.
- From a mathematical perspective, behaviors are simply a transfer function, transforming sensory inputs into actuator commands.

Reactive Paradigm (Cont'd)

- Sensing in the Reactive Paradigm is local to each behavior, or behavior-specific.
- Each behavior has its own dedicated sensing.
- But in other cases, more than one behavior can take the same output from a sensor and process it differently (via the behavior's perceptual schema).
- One behavior literally does not know what another behavior is doing or perceiving.

Reactive Paradigm (Cont'd)



Hybrid Deliberative/Reactive Architecture

- The cost of reactivity is a system that eliminates planning or any functions which involves remembering or reasoning about the global state of the robot relative to its environment.
- This means that a robot can not plan optimal trajectories (path planning), make maps, monitor its own performance, or even select the best behaviors to use to accomplish a task (general planning).

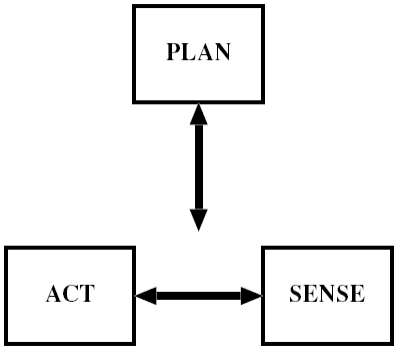
Hybrid Deliberative/Reactive Architecture

- Not all of these functions involve planning per se; map making involves handling uncertainty, while performance monitoring (and the implied objective of what to do about degraded performance) involves problem solving and learning.
- In order to differentiate these more cognitively oriented functions from path planning, the term ***deliberative*** was coined.

Hybrid Deliberative/Reactive Architecture (Cont'd)

- The Reactive Paradigm also suffered somewhat because most people found that designing behaviors so that the desired overall behavior would emerge was an art, not a science.
- Architectures which use reactive behaviors, but also incorporate planning, are now referred to as being part of the Hybrid Deliberative/Reactive Paradigm.

Hybrid Deliberative/Reactive Architecture (Cont'd)



Multi-Agents

- Collections of two or more mobile robots working together are often referred to as teams or *societies of multiple mobile robots, or more concisely multi-agents.*
- Multi-agent teams are desirable for many reasons. In the case of planetary explorers or removing land mines, more robots should be able to cover more area.
- Like ants and other insects, many cheap robots working together could replace a single expensive robot, making multi-agents more cost effective.

Swarm Robots

- Indeed, the term *swarm robots is becoming popular to refer to large numbers* of robots working on a single task.
- Another motivation of multiple robots is redundancy: if one robot fails or is destroyed, the other robots can continue and complete the job, though perhaps not as quickly or as efficiently.

Swarm Robots (Cont'd)

- http://www.youtube.com/watch?v=CJOubyiITsE

RoboCup

- Multi-agent teams are becoming quite popular in robot competitions, especially in RoboCup
- In RoboCup, teams of real or simulated robots play soccer against other teams.
- The soccer task explicitly requires multiple robots that must cooperate with each other, yet react as individuals.

Problems with teams of multiple agents

- *Designing teams is hard.* How does a designer recognize the characteristics of a problem that make it suitable for multi-agents? How does the designer (or the agents themselves) divide up the task?
- Are there any tools to predict and verify the social behavior?
- Having more robots working on a task or in a team increases the possibility that individual robots will unintentionally interfere with each other, lowering the overall productivity.
- It is hard for a team to recognize when it, or members, are unproductive.

Communication

- It is not clear when communication is needed between agents, and what to say.
- Many animals operate in flocks, maintaining formation without explicit communication (e.g., songs in birds, signals like a deer raising its tail, speaking).
- Formation control is often done simply by perceiving the proximity to or actions of other agents; for example, schooling fish try to remain equally close to fish on either side.
- But robots and modern telecommunications technology make it possible for all agents in a team to literally know whatever is in the mind of the other robots, though at a computational and hardware cost. How can this unparalleled ability be exploited?

Reliability

- What happens if the telecommunications link goes bad? Cell phones aren't 100% reliable, even though there is tremendous consumer pressure on cell phones, so it is safe to assume that robot communications will be less reliable.
- Is there a language for multi-agents that can abstract the important information and minimize explicit communication?

Autonomy and Individuality

- The "right" level of individuality and autonomy is usually not obvious in a problem domain.
- Agents with a high degree of individual autonomy may create more interference with the group goals, even to the point of seeming "autistic."
- But agents with more autonomy may be better able to deal with the open world.

Characteristics of Multi-Agents

- heterogeneity
- control
- cooperation
- goals

Heterogeneity

- Heterogeneity refers to the degree of similarity between individual robots that are within a collection.
- Collections of robots are characterized as being either heterogeneous or homogeneous.
- Heterogeneous teams have at least two members with different hardware or software capabilities, while in homogeneous teams the members are all identical.
- To make matter more confusing, members can be homogeneous for one portion of a task by running identical behaviors, then become heterogeneous if the team members change the behavioral mix or tasks.

Homogeneous Swarms

- Most multi-agent teams are homogeneous swarms.
- Each robot is identical, which simplifies both the manufacturing cost and the programming.
- The biological model for these teams are often ants or other insects which have large numbers of identical members.
- As such, swarms favor a purely reactive approach, where each robot operates under the Reactive Paradigm.

Heterogeneous Teams

- A new trend in multi-agents is heterogeneous teams.
- A common heterogeneous team arrangement is to have one team member with more expensive computer processing.
- That robot serves as the team leader and can direct the other, less intelligent robots, or it can be used for special situations.
- The danger is that the specialist robot will fail or be destroyed, preventing the team mission from being accomplished.

Marsupial Robots

- A special case of a cooperative, heterogeneous team of robots has been dubbed marsupial robots.
- The motivation for marsupial robots stemmed from concerns about deploying micro-rovers for applications such as Urban Search and Rescue.
- Micro-rovers often have limited battery power, which they can't afford to spend just traveling to a site.
- Likewise, micro-rovers may not be able carry much on-board processing power and need to have another, more computationally powerful workstation do proxy (remote) processing.
- A marsupial team consists of a large robot which carries one or more smaller robots to the task site, much like a kangaroo mother carries a joey in her pouch.

Marsupial Robots (Cont'd)

- Like a joey, the daughter robot is better protected in the pouch and can conserve energy or be recharged during transport.
- The mother can protect a delicate mechanism or sensor from collisions while it navigates through an irregular void.
- The mother can also carry a payload of batteries to recharge (feed) the daughter. It can serve as a proxy workstation, moving to maintain communications.
- The mother is likely to be a larger robot, while the daughter might be a micro-rover with sensors very close to the ground.
- The mother will have a better viewpoint and sensors, so in some circumstances it can communicate advice to the smaller daughter to help it cope with a "mouse's eye" view of the world.

Marsupial Robots (Cont'd)

- http://www.youtube.com/watch?v=BV7XseULLok

Mars Pathfinder

- The Mars Pathfinder mission is similar to a marsupial robot in that a micro-rover was transported to a mission site and the transport vehicle served as a support mechanism.
- However, our definition of marsupial assumes the mother is a fully mobile agent and can recover and retask the micro-rover.

Social Entropy

- One rough measure of the degree of heterogeneity is the *social entropy metric.*
- The point of social entropy is to assign a numerical value for rating diversity (or disorder) in a team.
- The number should be 0 if all team members are the same (homogeneous).
- The number should have the maximum value if all the team members are different.
- The number of team members which are different should make the overall number higher.

Social Entropy (Cont'd)

- The formula for the social entropy is

$$Het(\mathcal{R}) \quad = \quad -\sum_{i=1}^{c} p_i \log_2(p_i)$$

- There are two types of robots in the team, call castes or c: the mother and the daughters. Therefore c=2. The team pi is the decimal percent of robots belonging to cast ci. If i=1 for mother and i=2 for daughters:
- Suppose p1 = ¼ and p2 = ¾ .

Social Entropy (Cont'd)

- The social entropy is

$$\begin{aligned} Het(\mathcal{R}) &= -\sum_{i=1}^{c} p_i \log_2(p_i) \\ &= -(0.25 \log_2 0.25 + 0.75 \log_2 0.75) \\ &= -((-0.50) + (-0.31)) \\ &= 0.81 \end{aligned}$$

Control

- Control of multi-agents can fall in a spectrum bounded by centralized control and distributed control regimes.
- In centralized control, the robots communicate with a central computer. The central computer distributes assignments, goals, etc., to the remote robots.
- The robots are essentially semi-autonomous, with the centralized computer playing the role of a teleoperator in a teleoperated system.
- In distributed control, each robot makes its own decisions and acts independently. Of course, there is a range of regimes between fully centralized and fully distributed; the robots can interact with a central controller to receive new goals, then operate for the duration of the mission in a distributed manner.

Centralized Control

- Examples of full and partial centralized control can be found in Small Size league of RoboCup Soccer. Teams of 5 robots are controlled remotely by a central computer.
- Each robot had a unique pattern of bright colors to make it visible from the overhead cameras, and the overhead camera is connected to a central processor.
- The robots communicate with the central processor over a radio link.
- The central processor can give either explicit directions or just locations of other robots and the ball, letting the robot's onboard behaviors generate the (one hopes) correct response.

Centralized Control (Cont'd)

- The robots must have a set of basic tactical behaviors but may either receive strategic commands from the central computer or have on-board strategic behaviors.
- This type of control is conceptually equivalent to the Hybrid Reactive-Deliberative Paradigm, where the reactive layer physically resides on the robot and the deliberative layer resides on the central workstation.

Distributed Control

- Distributed control is more natural for soccer playing than centralized control, because each player reacts independently to the situation.
- An example of distributed control in robot soccer playing is the mid-sized league or Simulation 3D league in RoboCup.
- Notice that in robot soccer the robots are inherently heterogeneous.
- Although they may be physically the same, each robot is programmed with a different role, most especially Goalie, Striker, and Defender.

Cooperation

- Cooperation refers to how the robots interact with each other in pursuing a goal.
- Robots can show active cooperation by acknowledging one another and working together.
- Note that this does not necessarily mean the robots communicate with each other.
- For example, in robot soccer, one robot can pass the ball to another robot as part of an offensive play.
- The cooperation does not require communication—if a robot has the ball, can't see goal and can see team mate, then it passes to team mate, but this does require being aware of the teammates.

Cooperation (Cont'd)

- It is easy to think of cooperation in terms of robots working together on a task.
- Another aspect of cooperation is physical cooperation, where the robots physically aid each other or interact in similar ways.
- Marsupial robots are certainly a type of physical cooperation, especially during deployment and docking.

Working on a Goal

- The final dimension for characterizing a collection of multi-agents is how the robot works on a goal.
- If all the robots in the collection work on attaining the same explicit goal, then they are said to share a single goal, versus having individual goals.

Summary of MAS

- In summary, many tasks favor the use of many cheap robots rather than a single expensive one.
- These collections of multiple robots are often referred to as multi-agents.
- Individual robots in a multi-agent team are generally programmed with behaviors, most often as purely reactive systems, but occasionally with a hybrid architecture.

Summary of MAS (Cont'd)

- Heterogeneity refers to whether the member robots are identical in software and hardware.
- Cooperation may be either active or non-active.
- Control may fall in the spectrum from fully distributed to fully centralized.
- A robot society may have a single, explicitly shared goal or each robot may have its own goal.