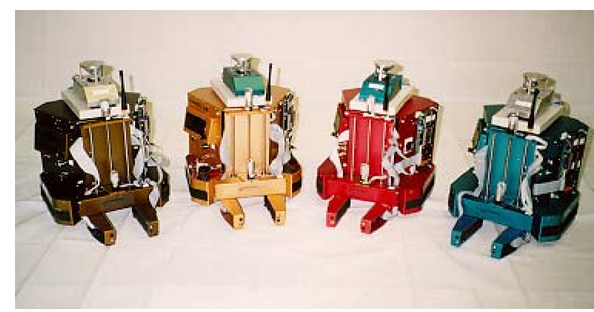
# Big Picture: Overview of Issues and Challenges in Autonomous Robotics + Impact on Practical Implementation Details

August 27, 2002

Class Meeting 2



# Announcements/Questions

- Course mailing list set up:

  cs594-sir@cs.utk.edu

  If you haven't received a "welcome" message from this mailing list, see William Duncan (TA) right away.

- Any questions about Assignment #1?

# Today's Objective: Understand big picture + challenges and realization in terms of practical implementation details

## Remember last time -- Issues we'll study this term:

- Robot control architectures
- Biological foundations
- Design of behavior-based systems
- Representation Issues
- Sensing
- Adaptation
- Multi-robot systems
- Path planning
- Navigation
- Localization
- Mapping

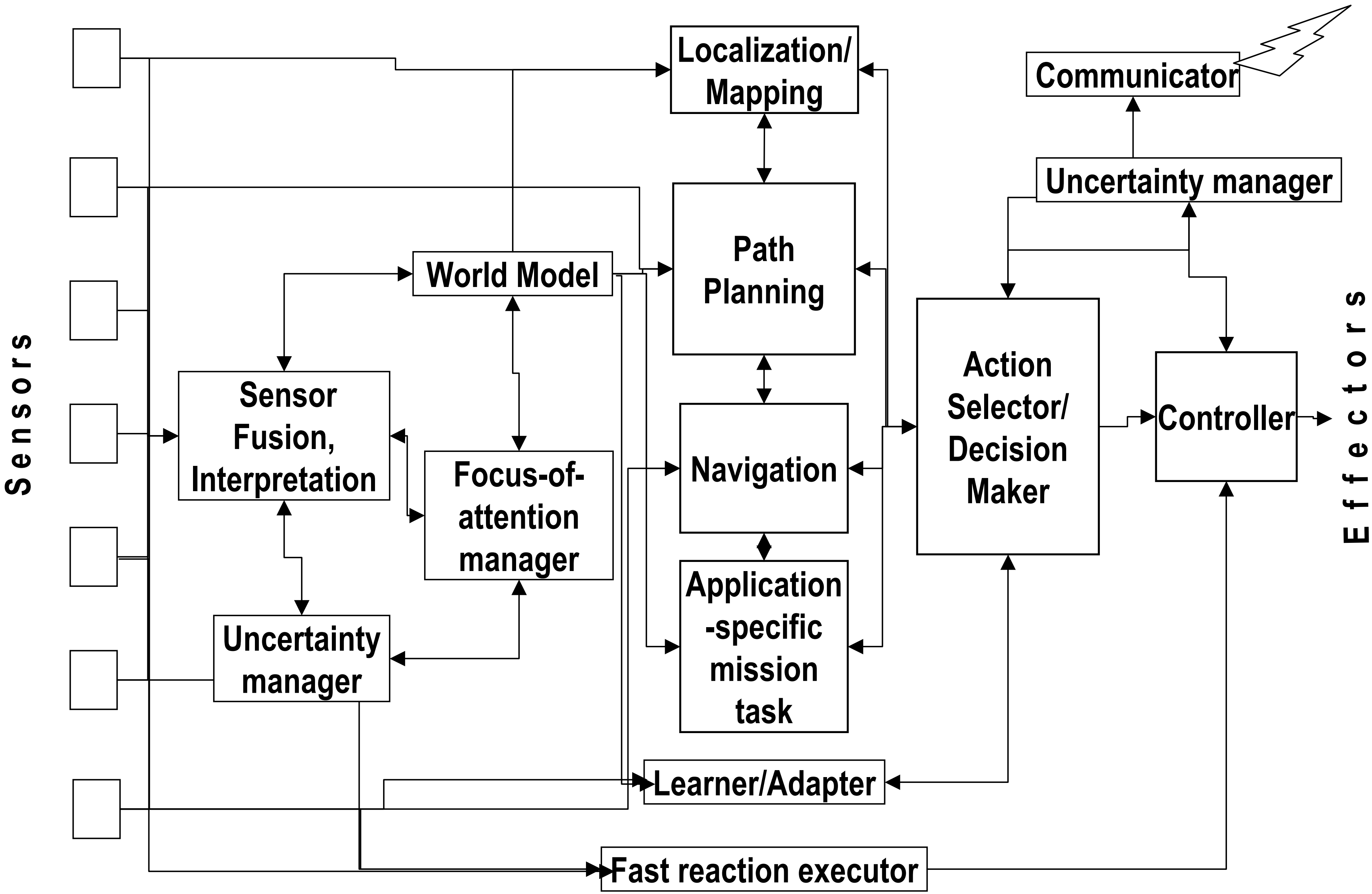
# Same issues from robot perspective

- Where am I? [localization]
- How do I interpret my sensor feedback to determine my current state and surroundings? [sensor processing/perception]
- How do I make sense of noisy sensor readings? [uncertainty management]
- How do I fuse information from multiple sensors to improve my estimate of the current situation? [sensor fusion]
- What assumptions should I make about my surroundings? [structured/unstructured environments]
- How do I know what to pay attention to? [focus-of-attention]

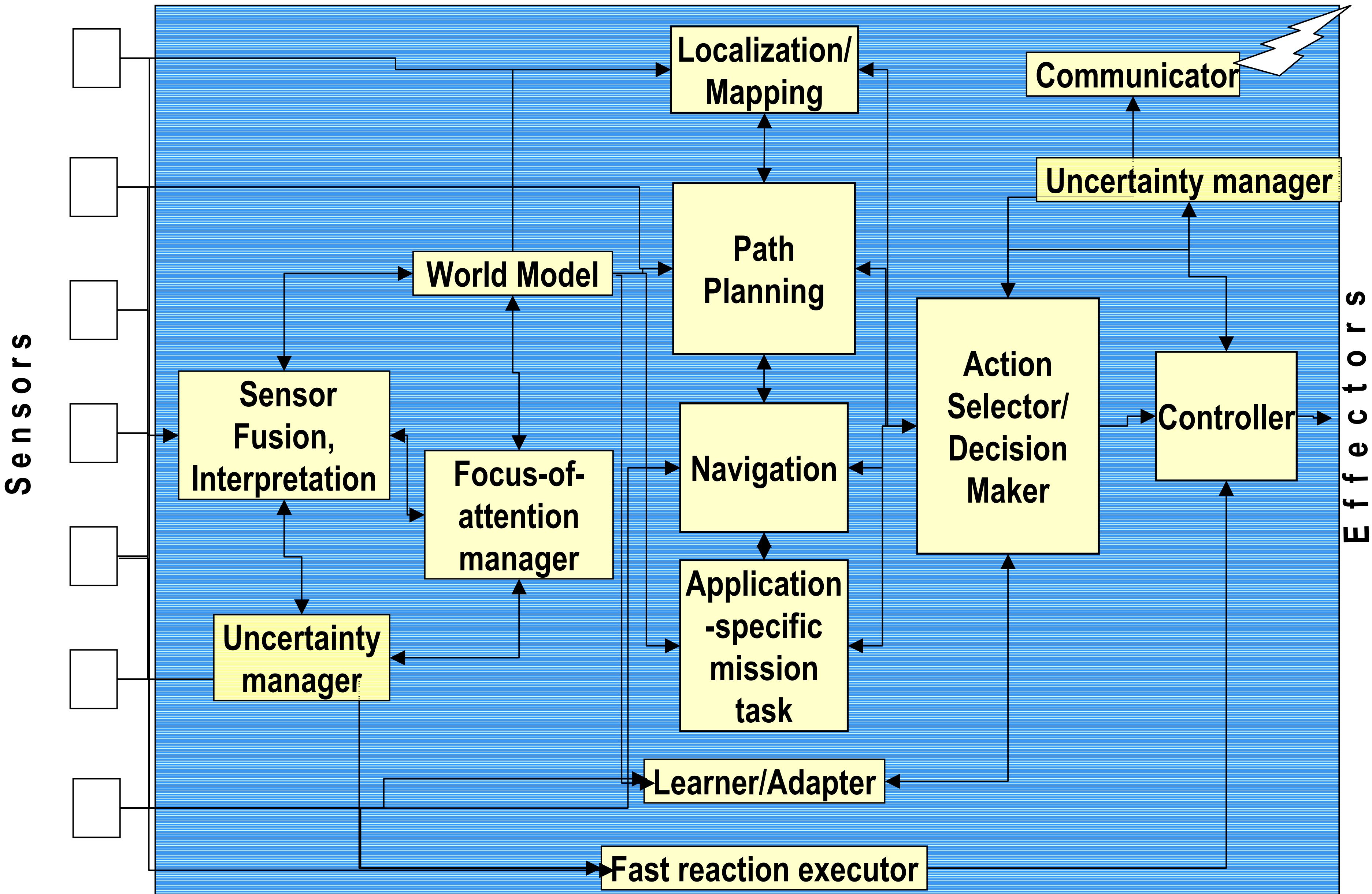
# More issues from robot perspective

- What should my control strategy be to ensure that I respond quickly enough? [control architecture]
- How should I make decisions? [reasoning, task arbitration]
- Where do I want to be, and how do I get there? [path planning, navigation]
- I have lots of choices of actions to take -- what should I do in my current situation? [action selection]
- How should I change over time to respond to a dynamic environment? [learning, adaptation]
- Why doesn’t the same action that worked in this situation before not work now? [hidden state]
- How should I work with other robots? [multi-robot cooperation, communication]

# Functional Modules of a (Hypothetical) Intelligent Mobile Robot

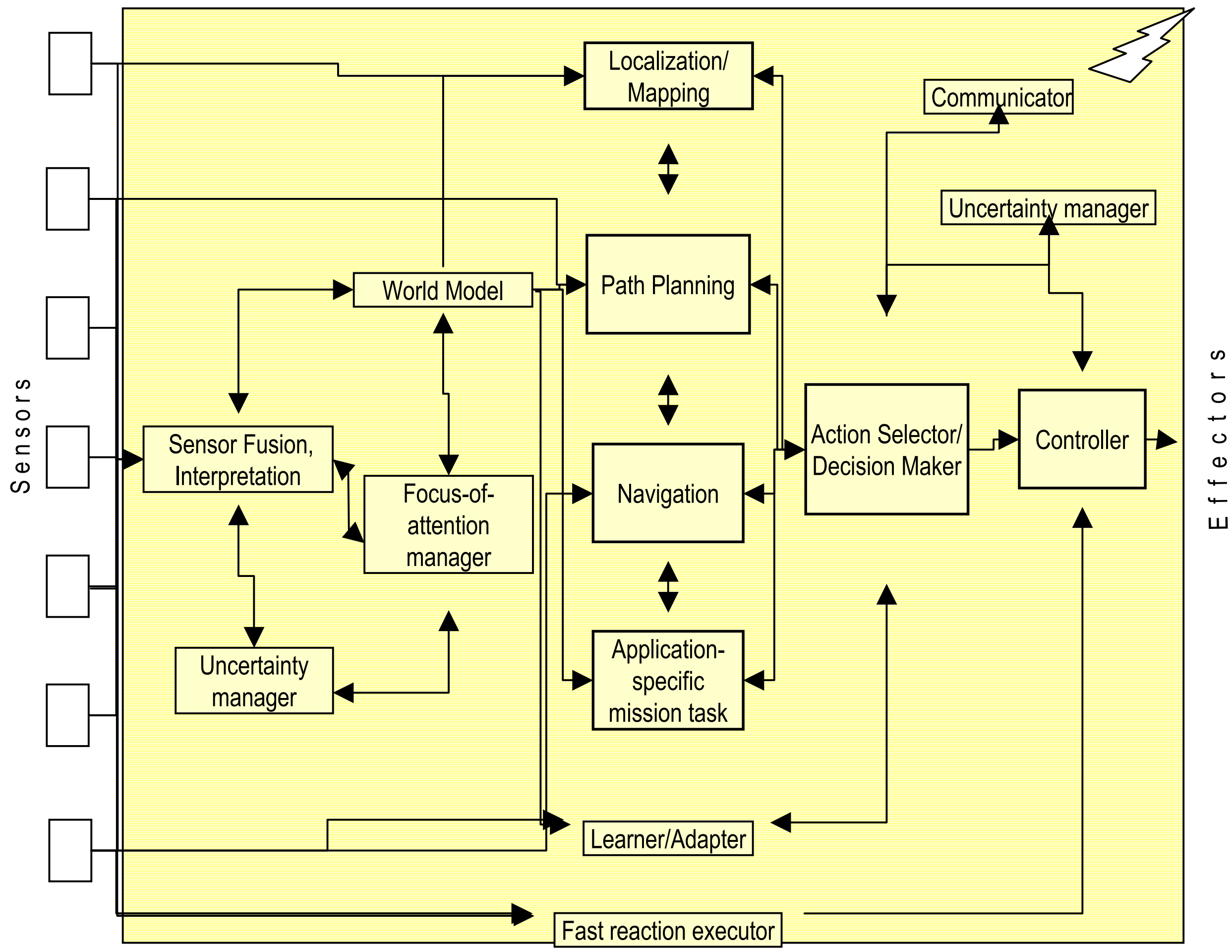


# Mobile Robot Software Challenge: Usually, all "reasoning" functions reside on single processor



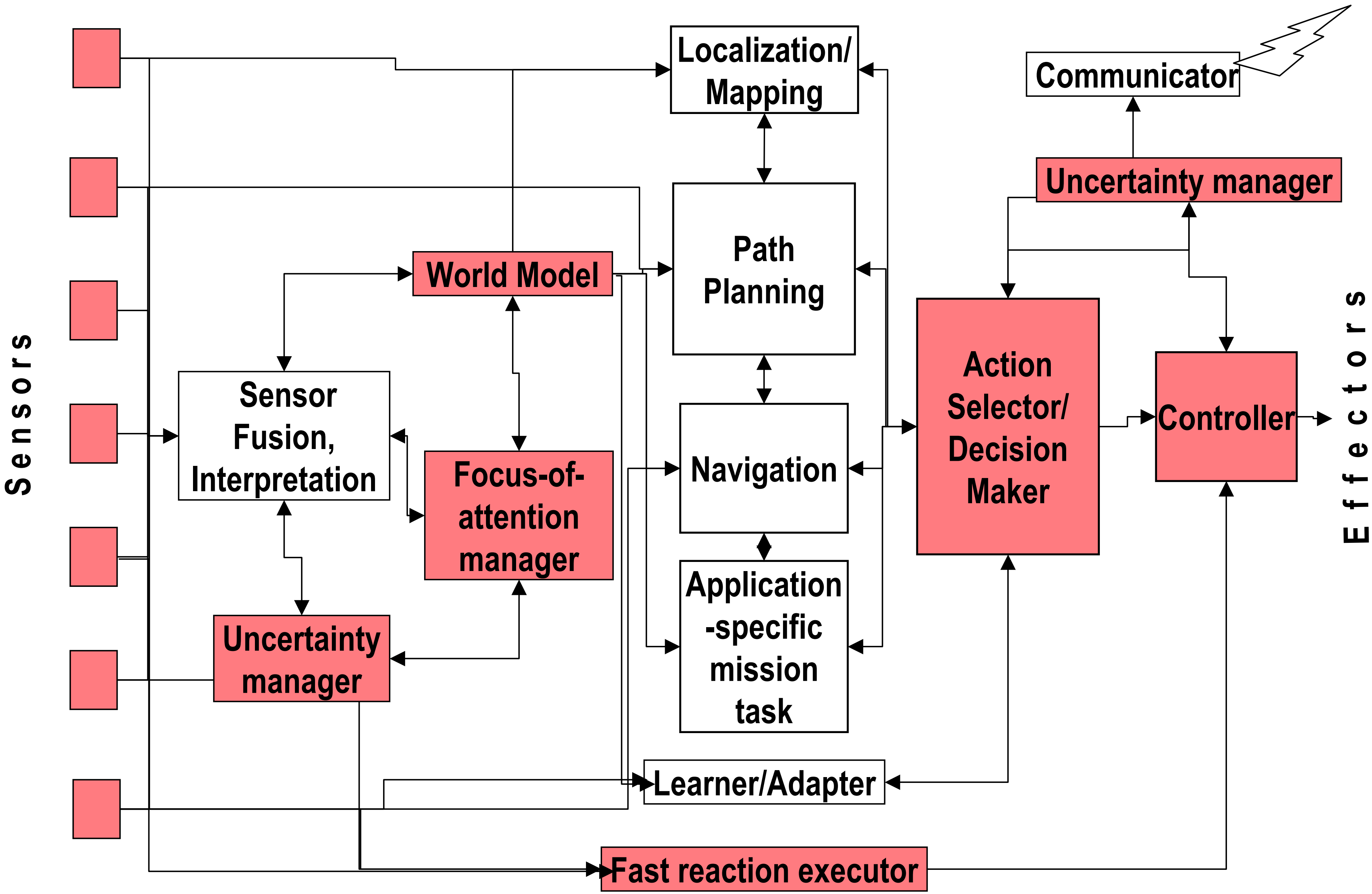
# How do we organize all this?

- Typical organizations:
  - Hierarchical
  - Behavior-based / Reactive
  - Hybrid

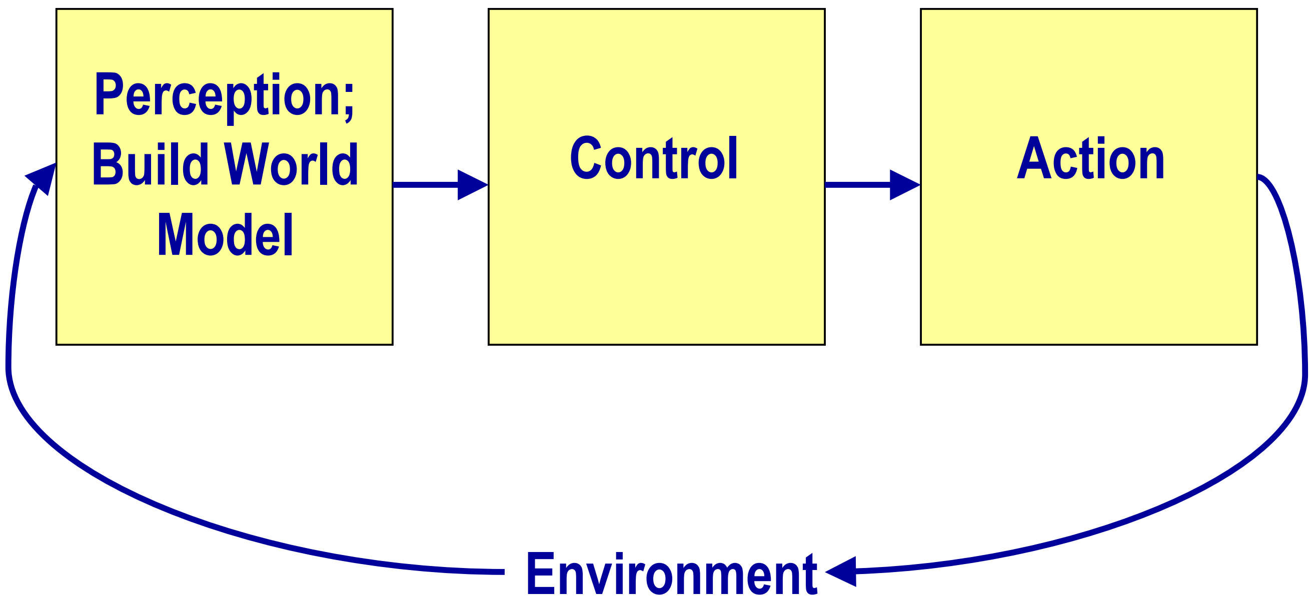



***Focus of classes 4-10***

# Functional Modules Related to Control Architecture



# Hierarchical Organization

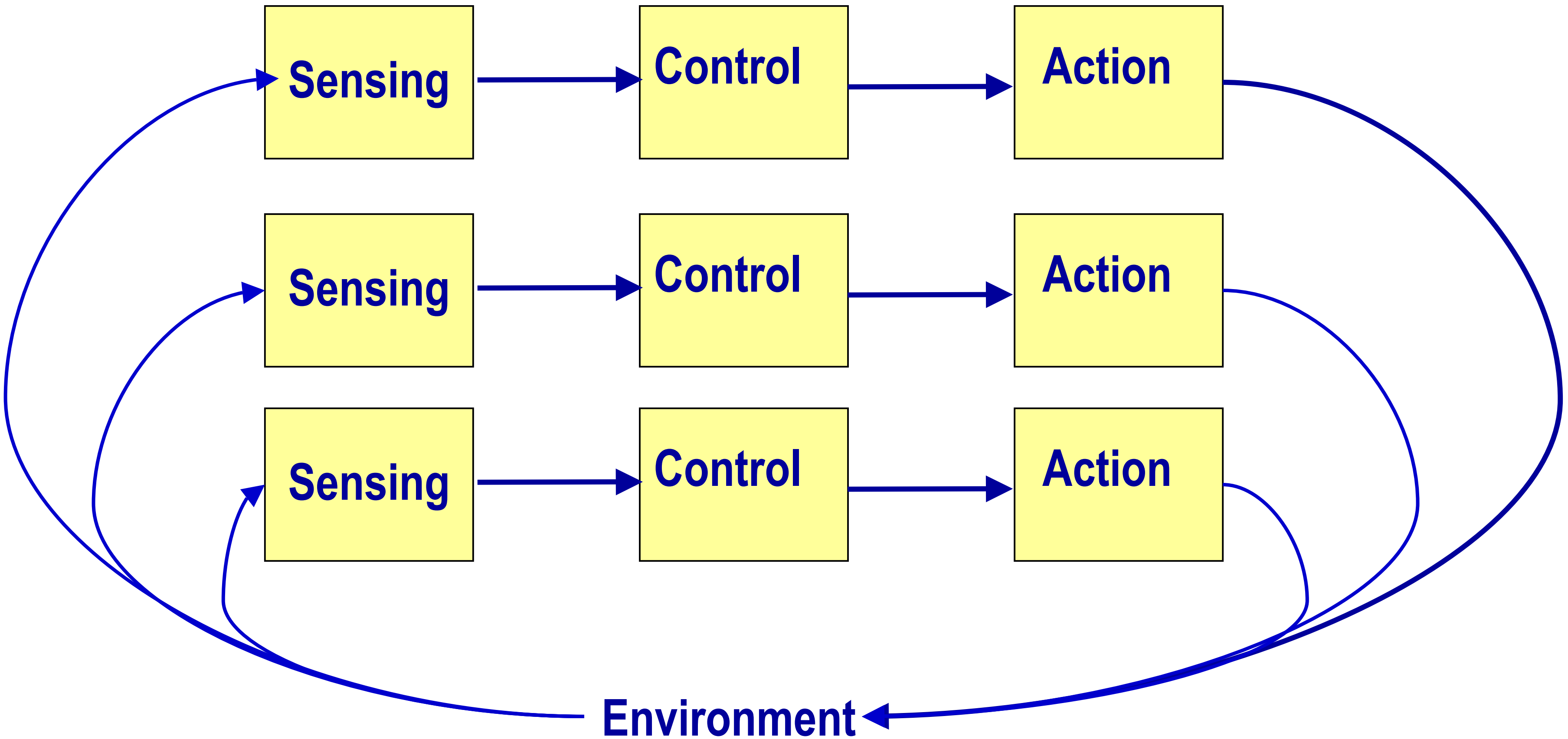



*Earliest robot control projects used this approach, with limited success.*

# Behavior-Based / Reactive: Based on Biological Paradigm

***Philosophy:***

***"World is own best model; therefore, don't try to build another world model"***

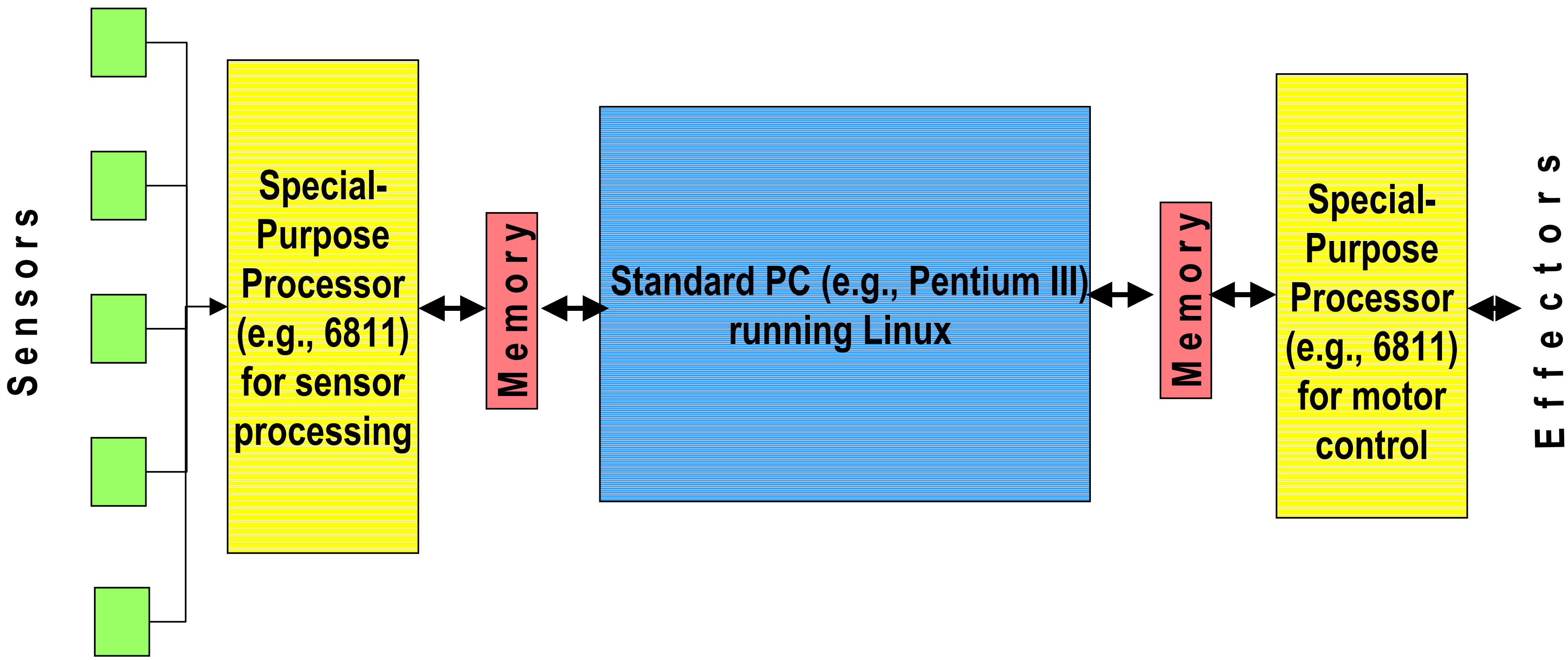



***More recent robot projects use this approach, with better success.***

***Focus of classes 5-10: Biological Parallels, Behavior-Based Paradigm and Design***

# Typical mobile robot implementation architecture

- Essentially:  PC on wheels/legs/tracks/...

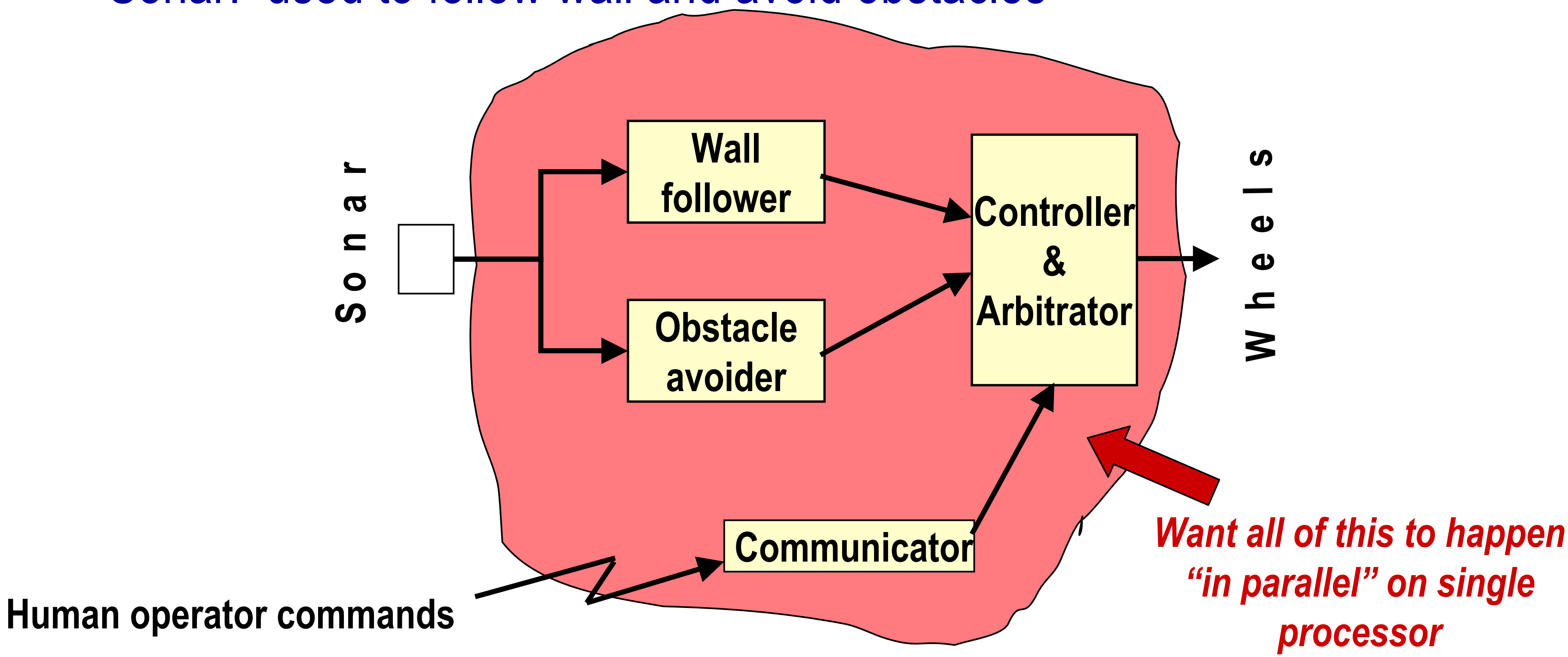


# Implication for Robot Control Code

- Two options:
  - Separate threads with appropriate interfaces, interrupts, etc.
  - Single process with “operating-system”-like time-slicing of procedures
- Usually:  combination of both
- For now:  let’s examine single process with “operating-system”-like time-slicing of procedures

# Simple program: Follow walls and obey human operator commands

- Assume we have the following functions needed:
  - Communications to operator interface -- commands such as "stop", "go", etc.
  - Sonar: used to follow wall and avoid obstacles



# Typical “single process” control approach to achieve functional parallelism

```
int wall_follower( ) {/* one time slice of work */}
int obstacle_avoider( ) {/* one time slice of work */}
int communicator( ) {/* one time slice of work */}
int controller_arbitrator ( ) {/* decides what action to take */}

main()
 {
   while (forever)
     { wall_follower();
       obstacle_avoider();
       communicator();
       controller_arbitrator();
     }
 }
```

***Note of caution: dependent upon programmer to ensure individual functions return after “time slice” completed***

# Control Commands to Nomad200 Simulator

vm(translation, wheel_rotation, turret_rotation)

Robot continues to execute the given velocity commands until another command issued.

*Thus, duration of "time slice" is important.*

# Current Trend:
# More sophisticated programming infrastructure

- Provide infrastructure to ease programming
  - Eliminates need for programmer to define procedure “time slices”
- Object-oriented infrastructure facilitiating “parallel” operation (via operating system time sharing) of various modules
- Examples:
  - Behavior language
  - Use of CORBA (Common Object Request Broker Architecture)
  - “Mobility”
  - Etc.

# In Robot Design Choices, Must Consider Real-World Challenges

Recall from last meeting: Software Challenges:

- Autonomous: robot makes majority of decisions on its own; no human-in-the-loop control (as opposed to *teleoperated*)
- Mobile: robot does not have fixed based (e.g., wheeled, as opposed to *manipulator arm*)
- Unstructured: environment has not been specially designed to make robot's job easier
- Dynamic: environment may change unexpectedly
- Partially observable: robot cannot sense entire state of the world (i.e., "hidden" states)
- Uncertain: sensor readings are noisy; effector output is noisy

*Let's look at these in more detail…*

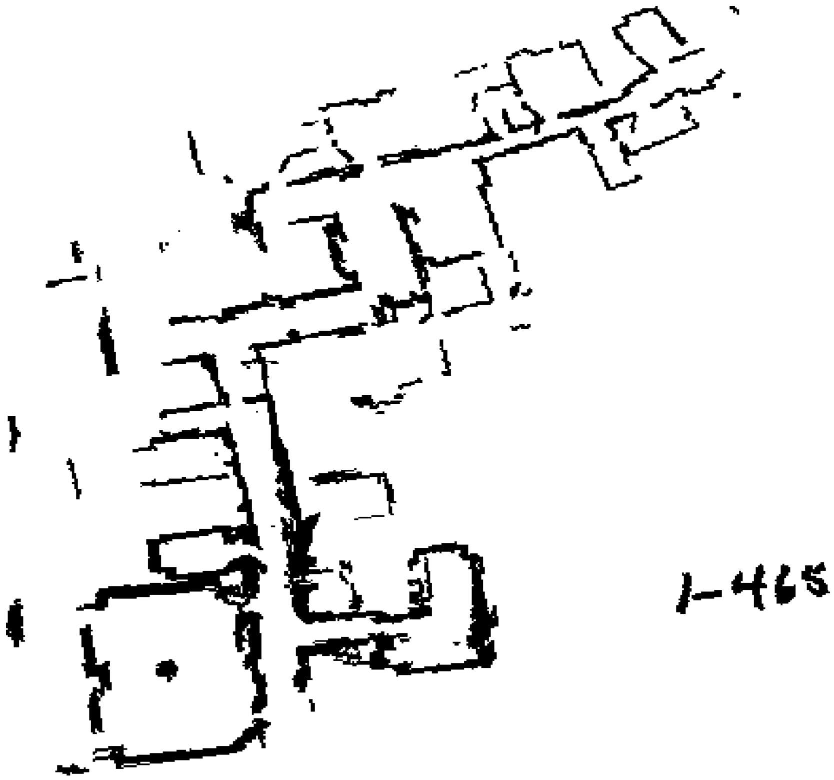
# Examples and Effect of Unstructured Environment

- Examples of unstructured environment:
  - Nearly all natural (non-man-made) environments:
    - Deserts
    - Forests
    - Fields
  - To some extent, man-made environments not specifically designed for robots

- Impact:
  - Difficult to make assumptions about sensing expectations
  - Difficult to make assumptions about environmental characteristics

# Example of Taking Advantage of Semi-Structured Environment

- If in most man-made buildings, assume perpendicular walls; allows straightening of “warped” walls caused by accumulated error.

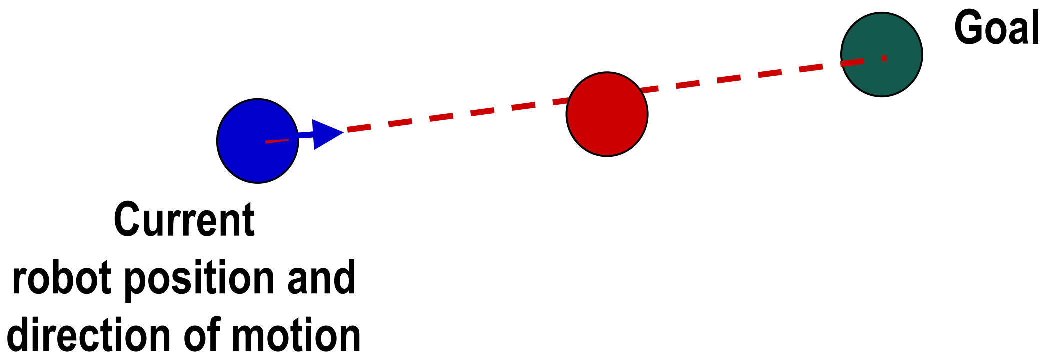


# Sources and Effect of Dynamic Environment

- Sources of dynamic environment:
  - Other robots/agents in the area
    - Teammates
    - Adversaries
    - Neutral actors
  - Natural events (e.g., rain, smoke, haze, moving sun, power outages, etc.)

- Impact:
  - Assumptions at beginning of mission may become invalid
  - Sensing/Action loop must be tight enough so that environment changes don't invalidate decisions

# Example of Effect of Dynamic Environment




Possible control code:

```
while (forever) do:
    { free = check_for_obstacle_to_goal();
      if (free)
          move_straight_to_goal();
      sleep(a_while);
    }       briefly
```

# Causes and Effect of Partially Observable Environment

- Causes of partially observable environment:
  - Limited resolution sensors
  - Reflection, occlusion, multi-pathing, absorption
- Impact:
  - Same actions in “same” state may result in different outcomes

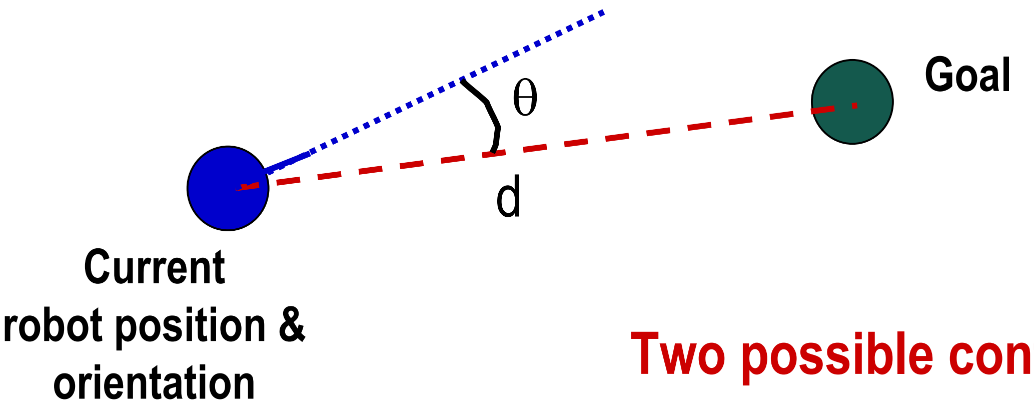
**Example:**
**Glass walls--laser sensors tricked**

# Sources and Effect of Uncertainty/Noise

- Sources of sensor noise:
  - Limited resolution sensors
  - Sensor reflection, multi-pathing, absorption
  - Poor quality sensor conditions (e.g., low lighting for cameras)

- Sources of effector noise:
  - Friction: constant or varying (e.g., carpet vs. vinyl vs. tile; clean vs. dirty floor)
  - Slippage (e.g., when turning or on dusty surface)
  - Varying battery level (drainage during mission)
- Impact:
  - Sensors difficult to interpret
  - Same action has different effects when repeated
  - Incomplete information for decision making

# Example of Effect of Noise on Robot Control Code: “Exact” Motions vs. Servo Loops




**Two possible control strategies:**

(1) “Exact” motions:

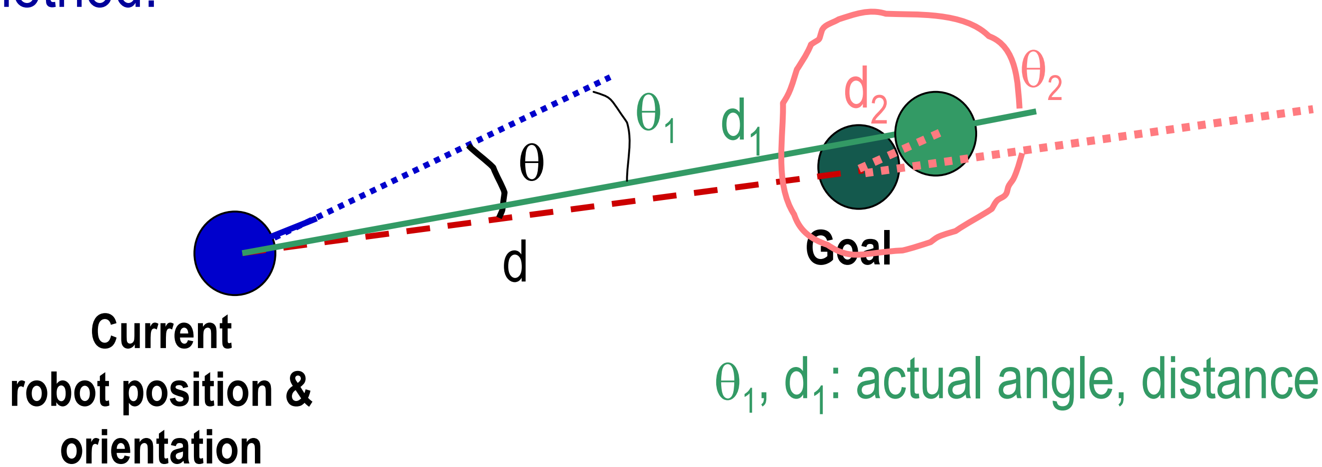
- Turn right by amount θ
- Go forward by amount d

(2) Servo loop:

- If to the left of desired trajectory, turn right.
- If to the right of desired trajectory, turn left.
- If online with desired trajectory, go straight.
- If error to desired trajectory is large, go slow.
- If error to desired trajectory is small, go fast.

# Consider effect of noise: "Exact" control method

- "Exact" method:



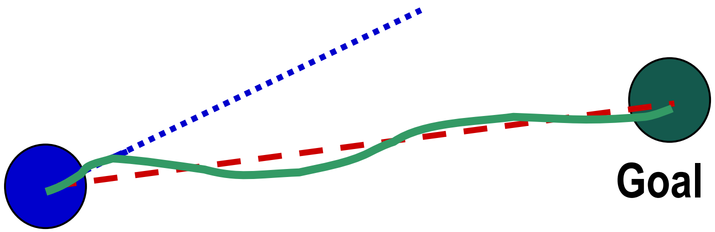

$\theta_1$, $d_1$: actual angle, distance traveled;

Noise ➔ overshoot goal; have to turn back to goal

***Doesn't give good performance***

# Consider effect of noise: Servo method

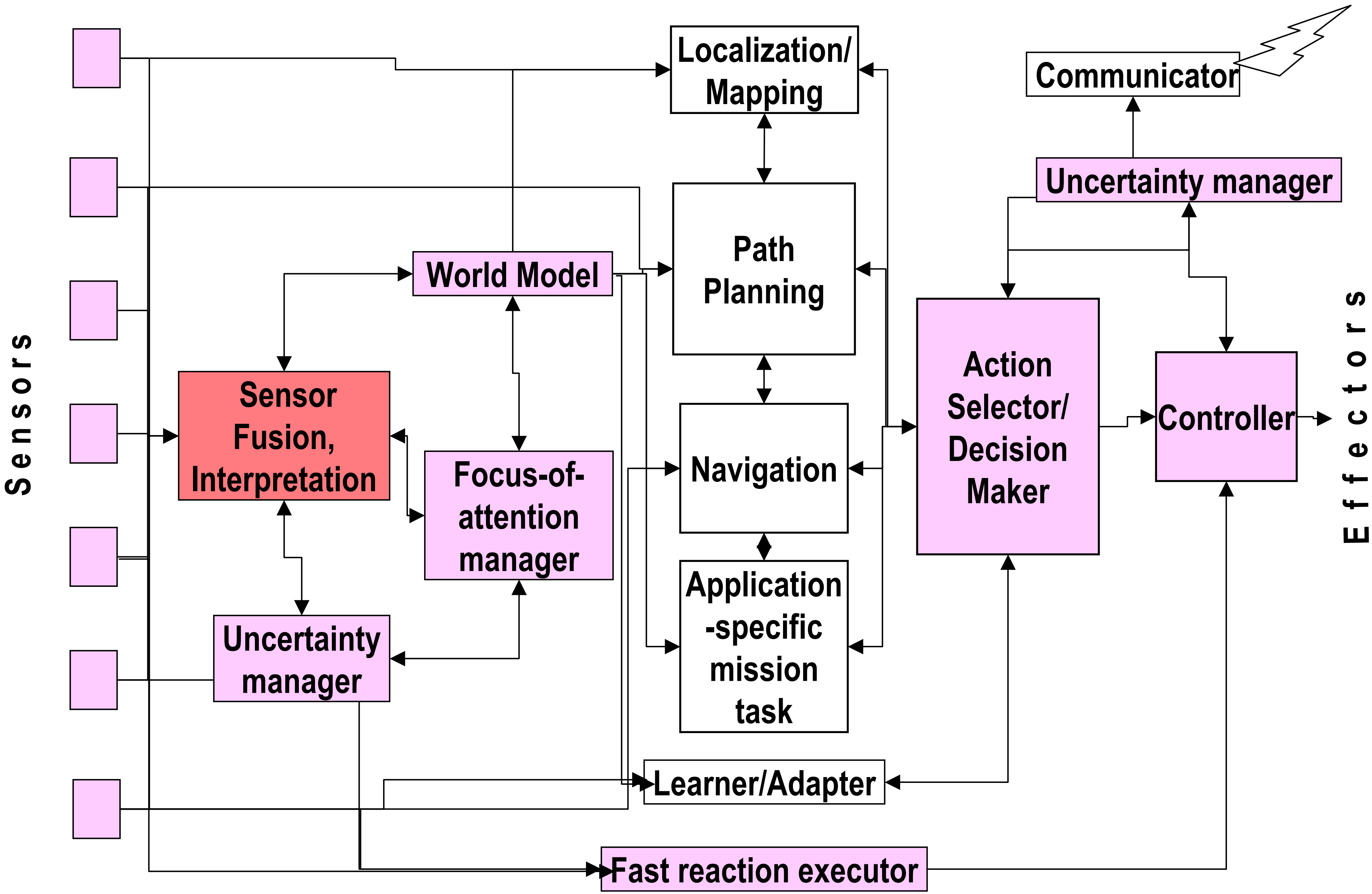
- Servo method:




***Much better performance in presence of noise***

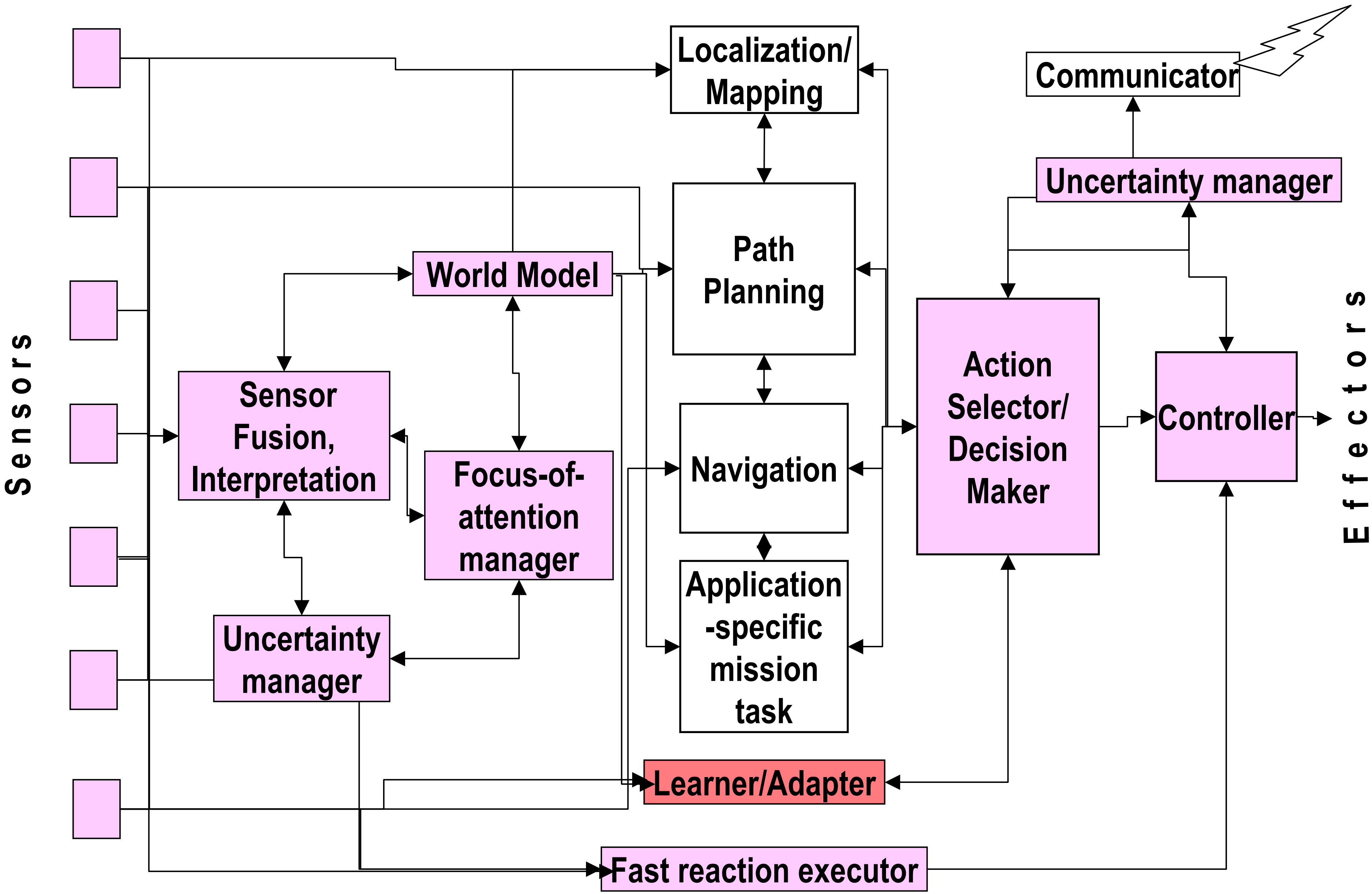
# Other Functional Modules We'll Study

*Focus of class 11-12: Sensing*



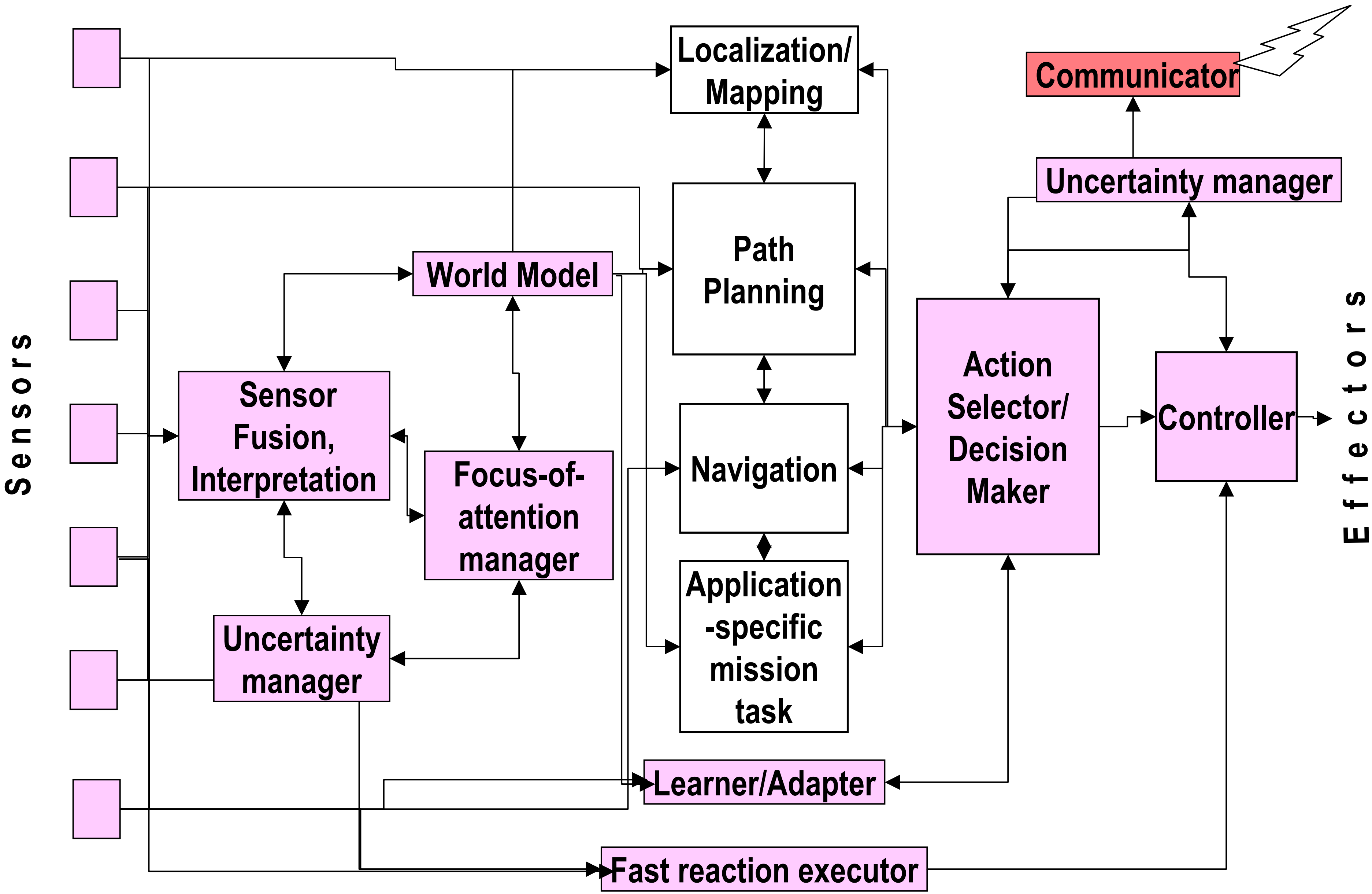
# Other Functional Modules We'll Study

*Focus of class 15-16: Adaptive Behavior*



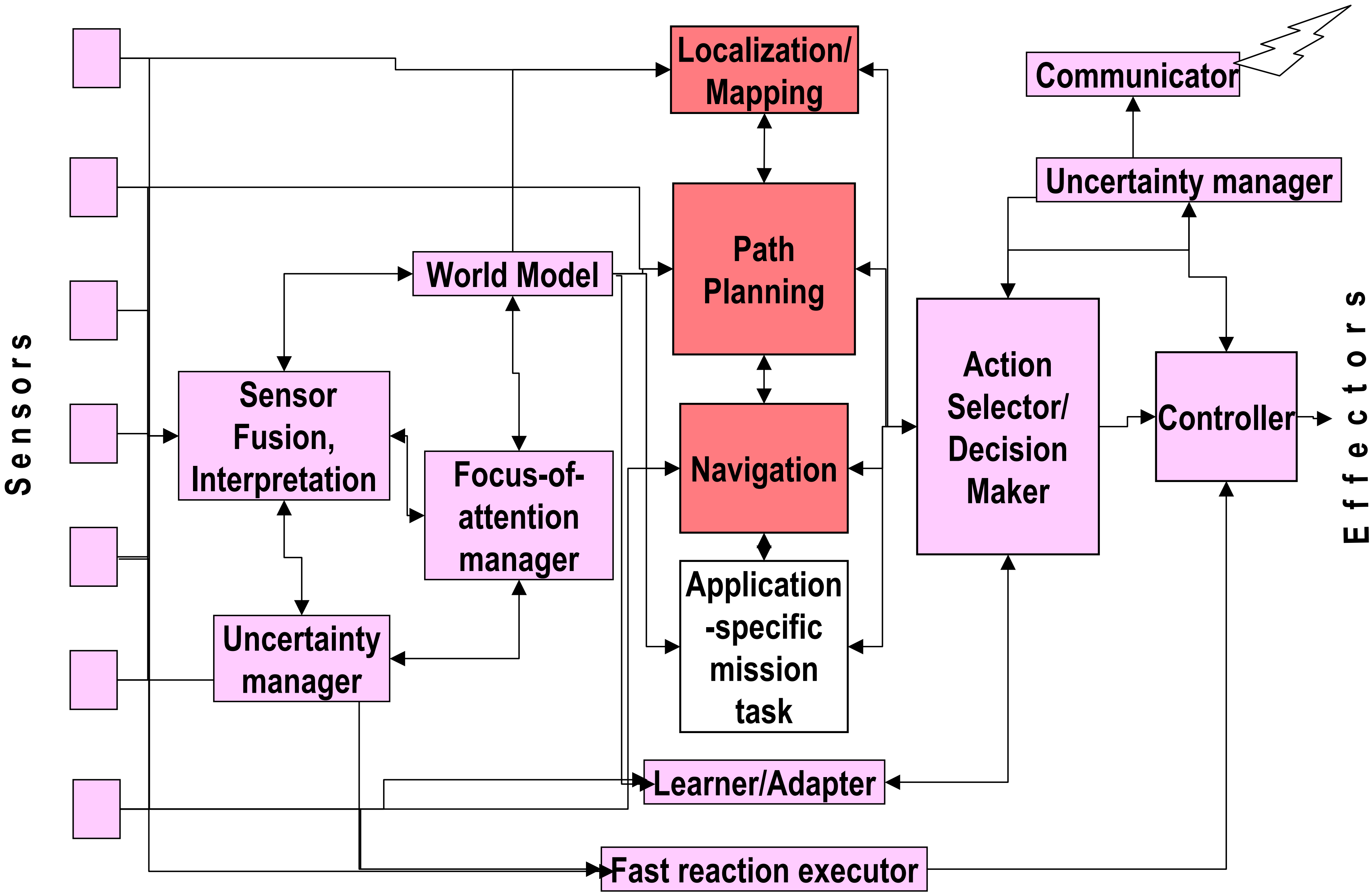
# Other Functional Modules We'll Study

*Focus of class 17-18: Multi-Robot Systems*

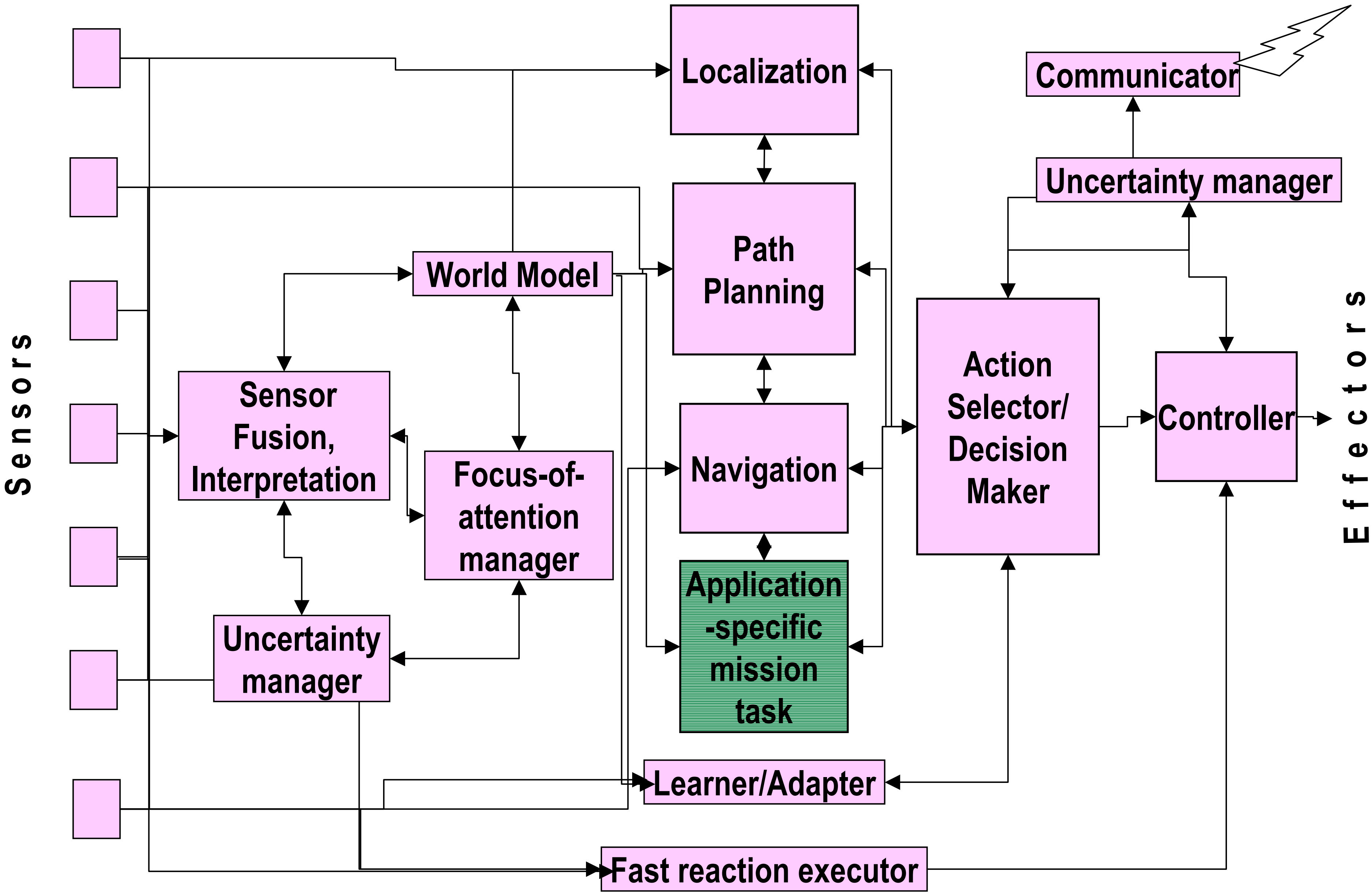


# Other Functional Modules We'll Study

*Focus of class 19-24: Navigation, Localization, Path Planning, Mapping*



# Your Final Project Will Look at Application-Specific Task



# Preview of Next Class

- History of Intelligent Robotics:
  - Earliest robots to present-day state-of-the-art
  - Evolution of control approaches from hierarchical to behavior-based