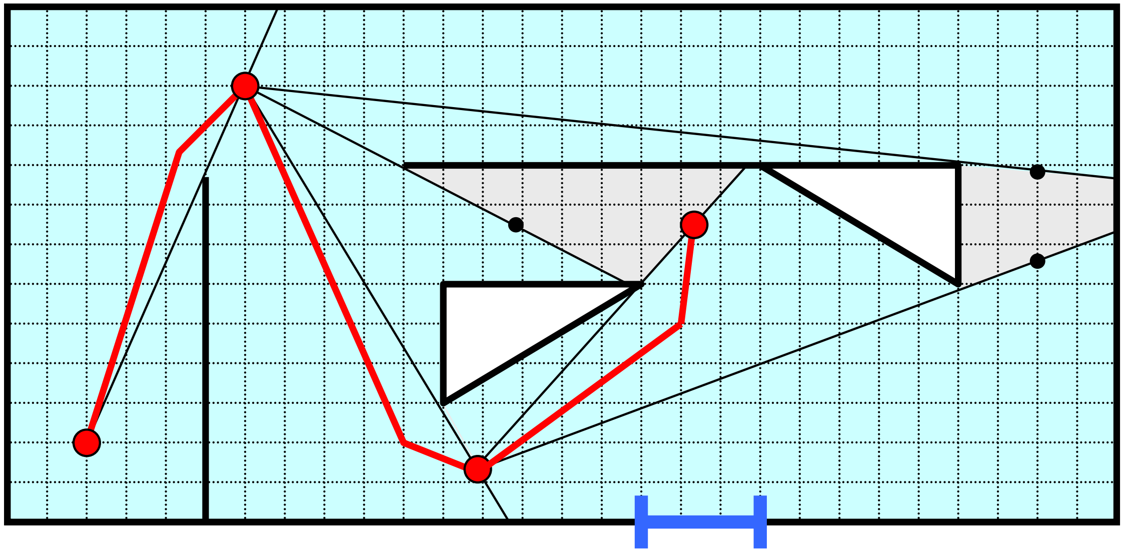
Control for Mobile Robots

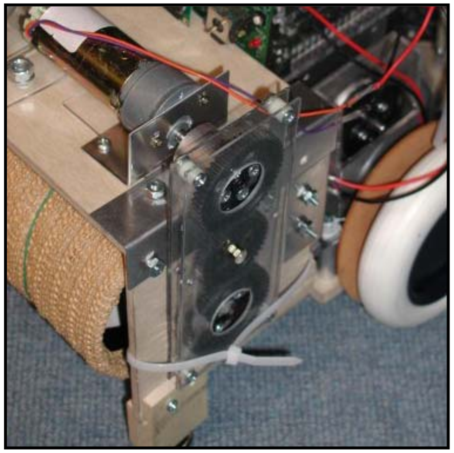


Christopher Batten

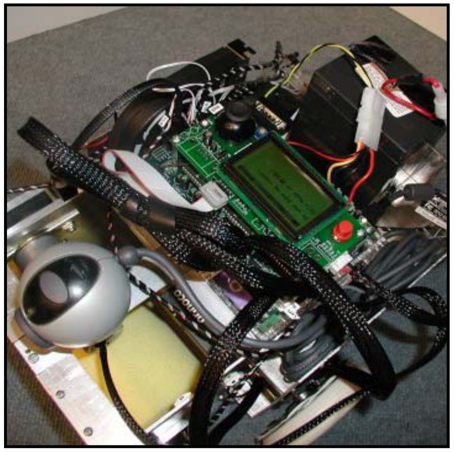
Maslab IAP Robotics Course
January 11, 2006

Building a control system for a mobile robot can be very challenging

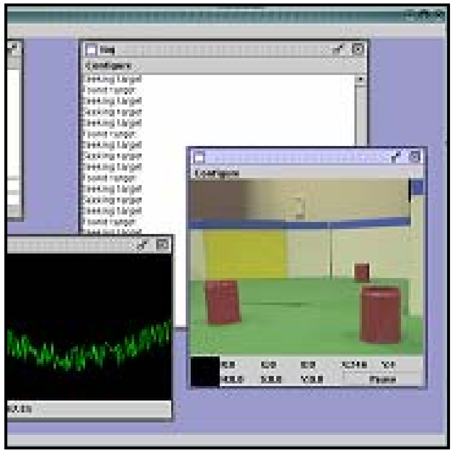
Mobile robots are very complex and involve many interacting components



Mechanical



Electrical



Software

Your control system must integrate these components so that your robot can achieve the desired goal

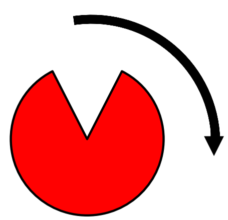
Building a control system for a mobile robot can be very challenging

Just as you must carefully **design** your robot chassis you must carefully **design** your robot control system

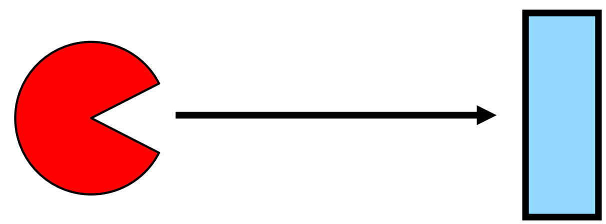
- How will you debug and test your robot?
- What are the performance requirements?
- Can you easily improve aspects of your robot?
- Can you easily integrate new functionality?

Basic primitive of a control system is a behavior

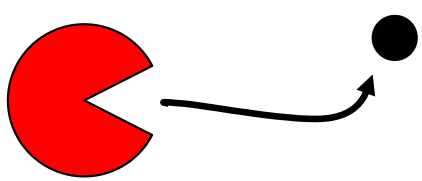
Behaviors should be well-defined, self-contained, and independently testable



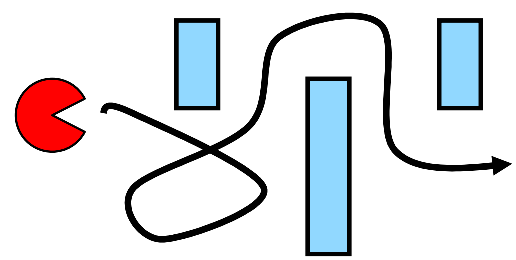
Turn right 90°



Go forward until reach obstacle

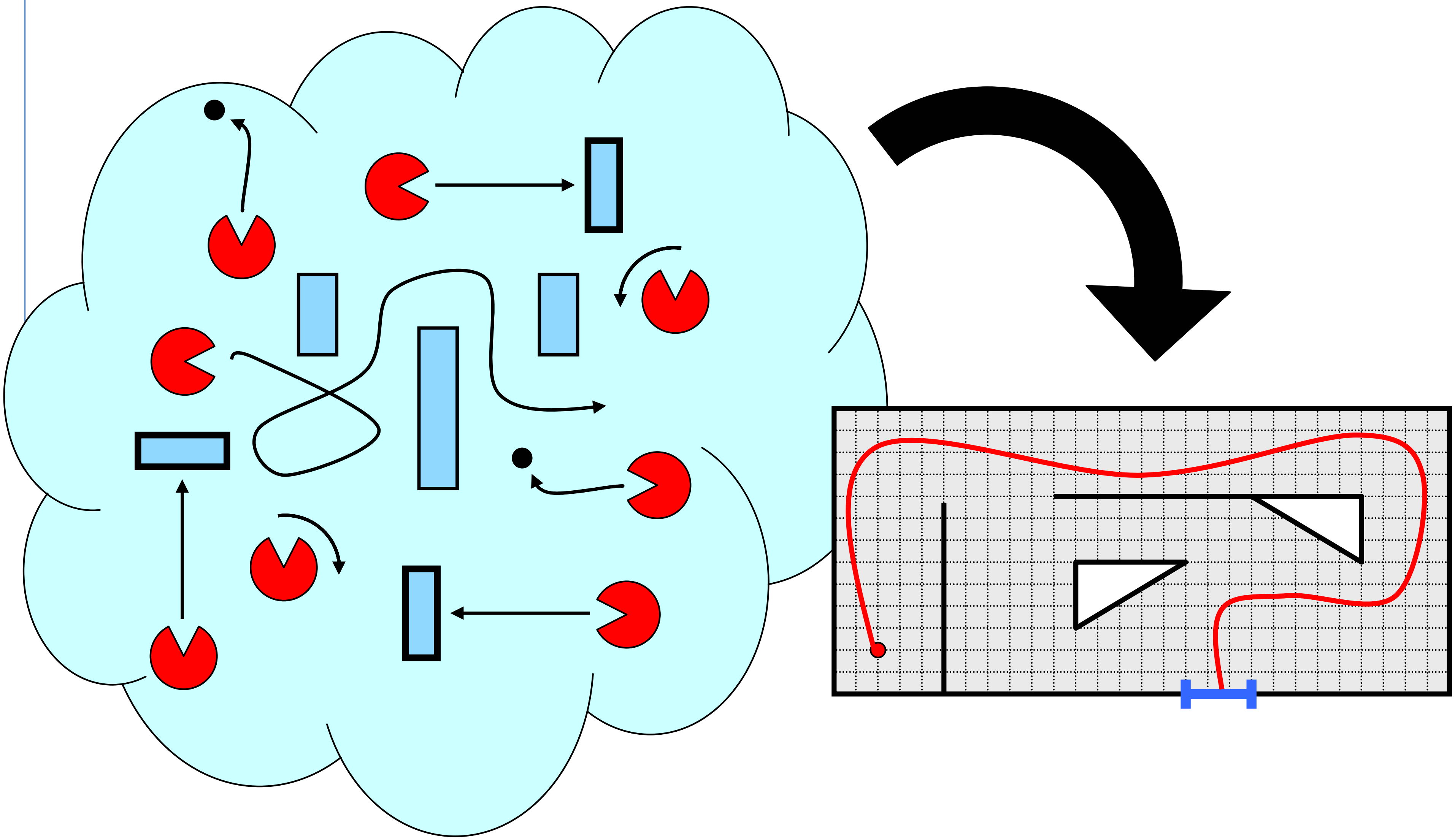


Capture a ball



Explore playing field

Key objective is to compose behaviors so as to achieve the desired goal

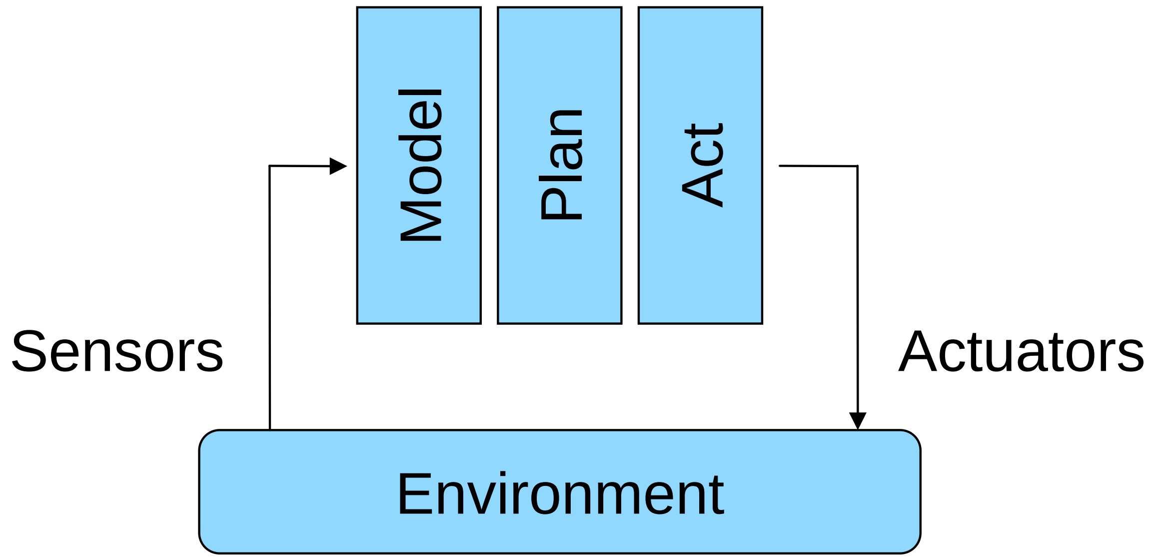


Outline



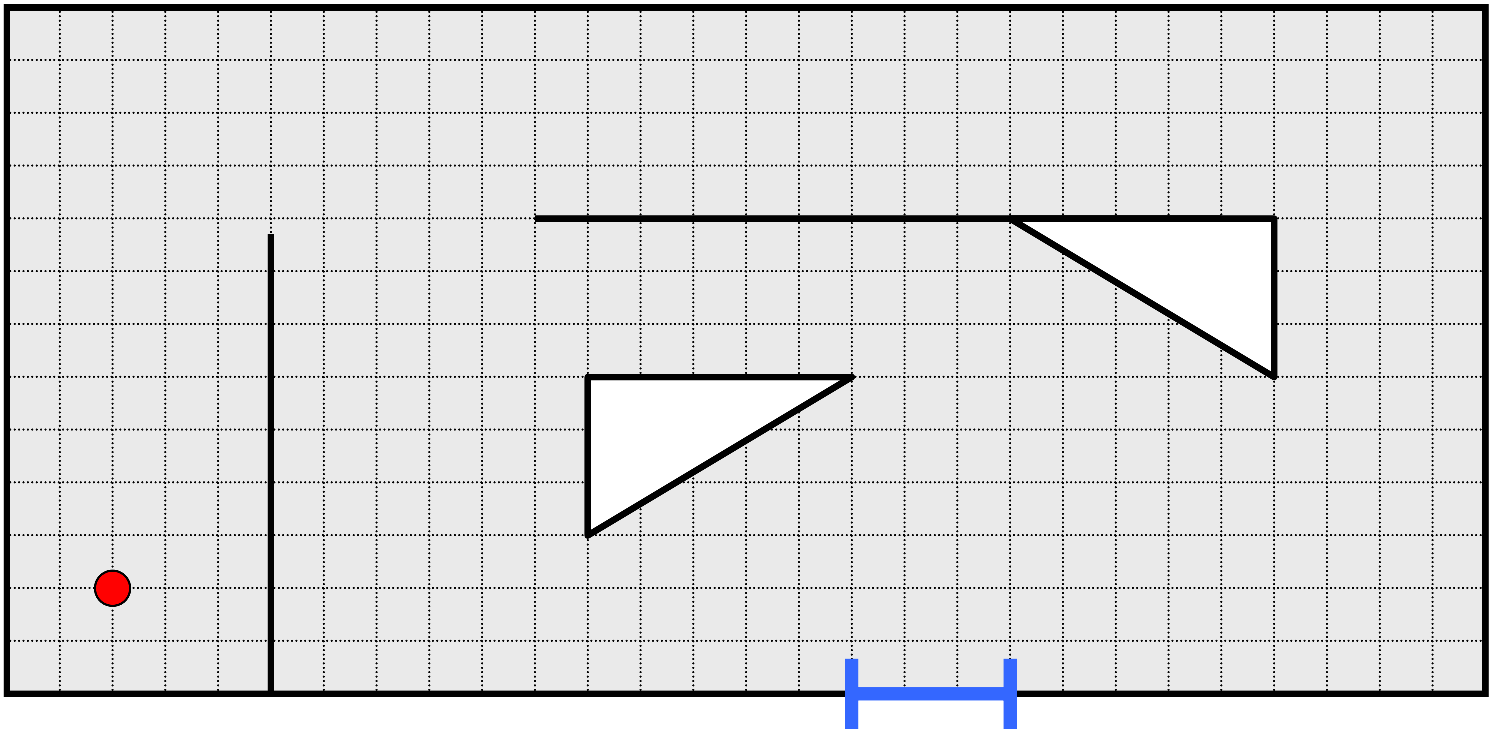
- High-level control system paradigms
  - Model-Plan-Act Approach
  - Behavioral Approach
  - Finite State Machine Approach
- Low-level control loops
  - PID controllers for motor velocity
  - PID controllers for robot drive system
- Examples from past years

Model-Plan-Act Approach



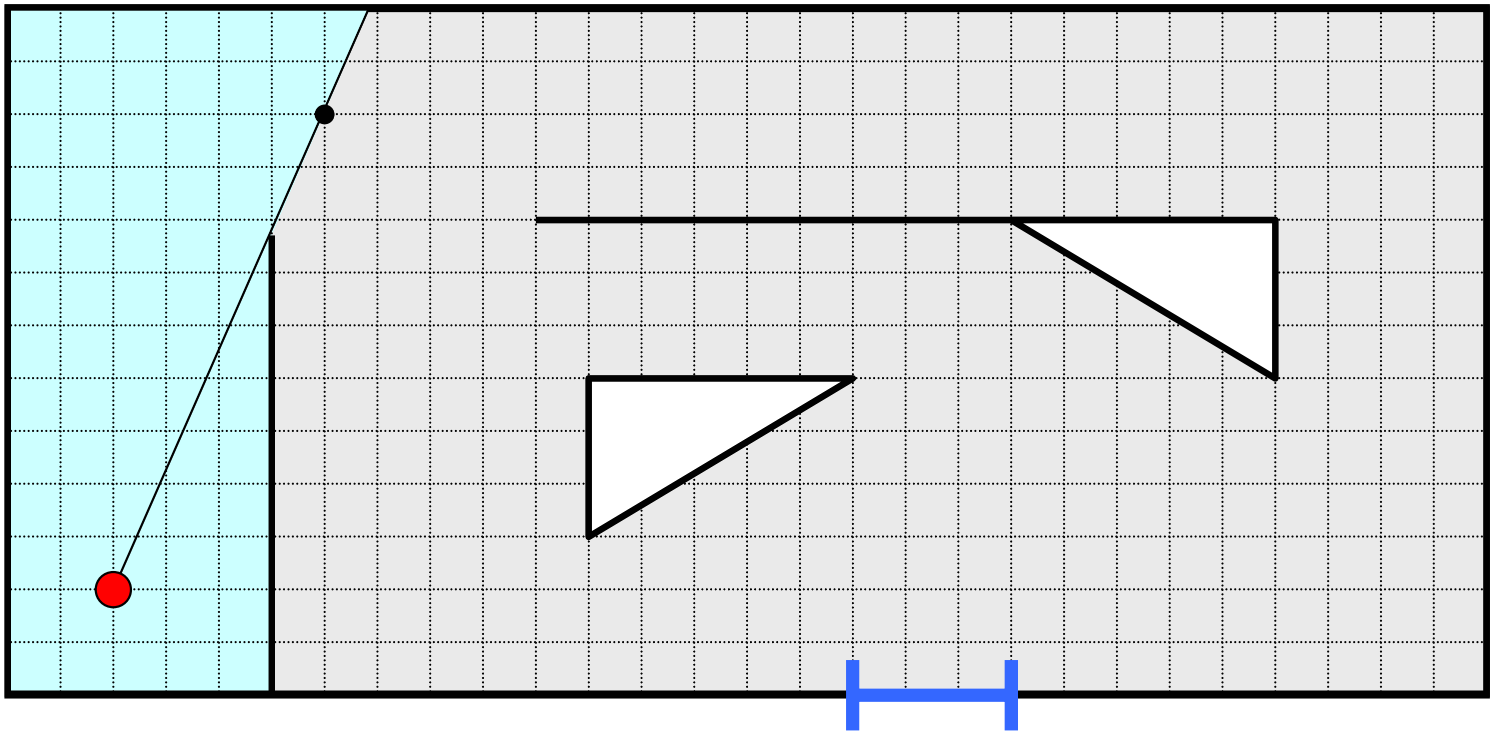
1. Use sensor data to create model of the world
2. Use model to form a sequence of behaviors which will achieve the desired goal
3. Execute the plan (compose behaviors)

Exploring the playing field using model-plan-act approach



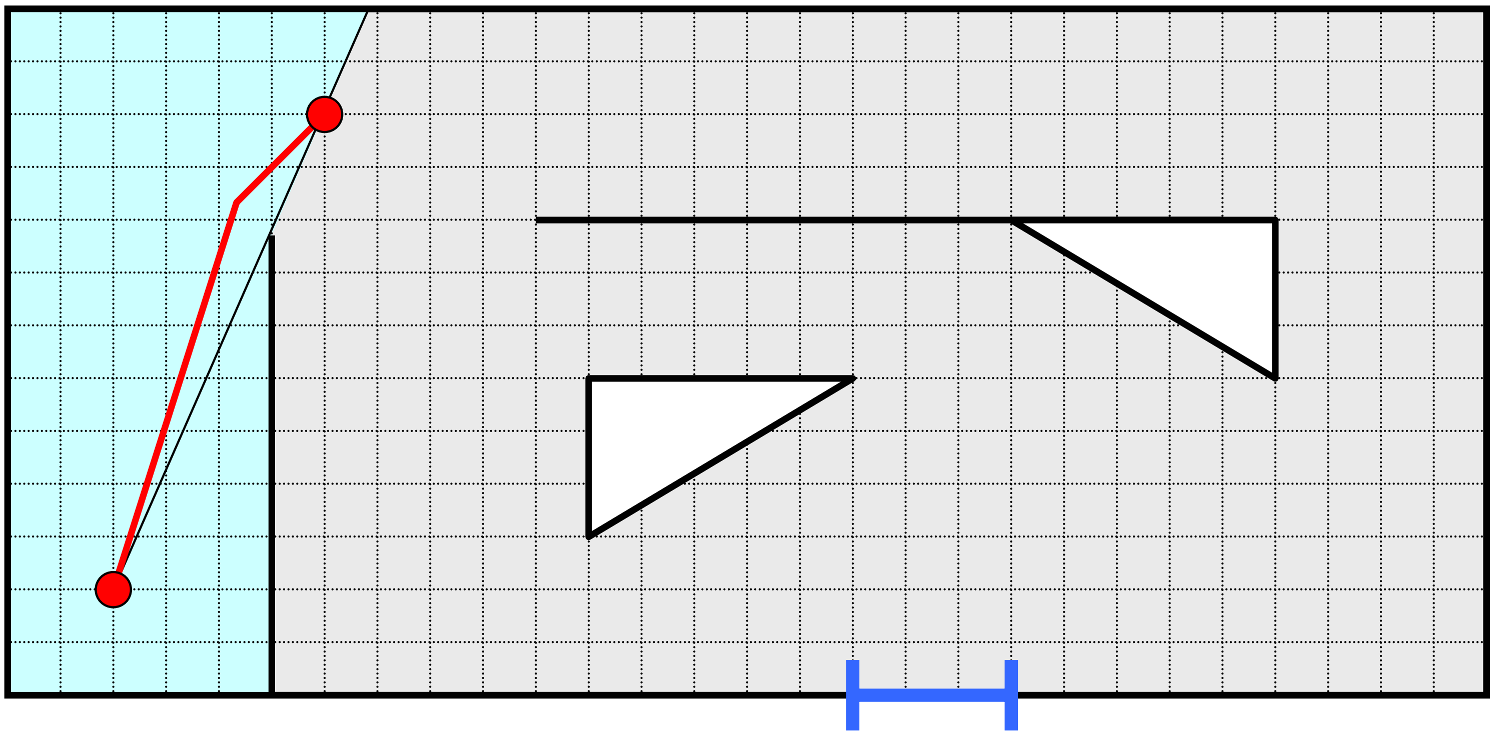
Red dot is the mobile robot while the blue line is the mousehole

Exploring the playing field using model-plan-act approach



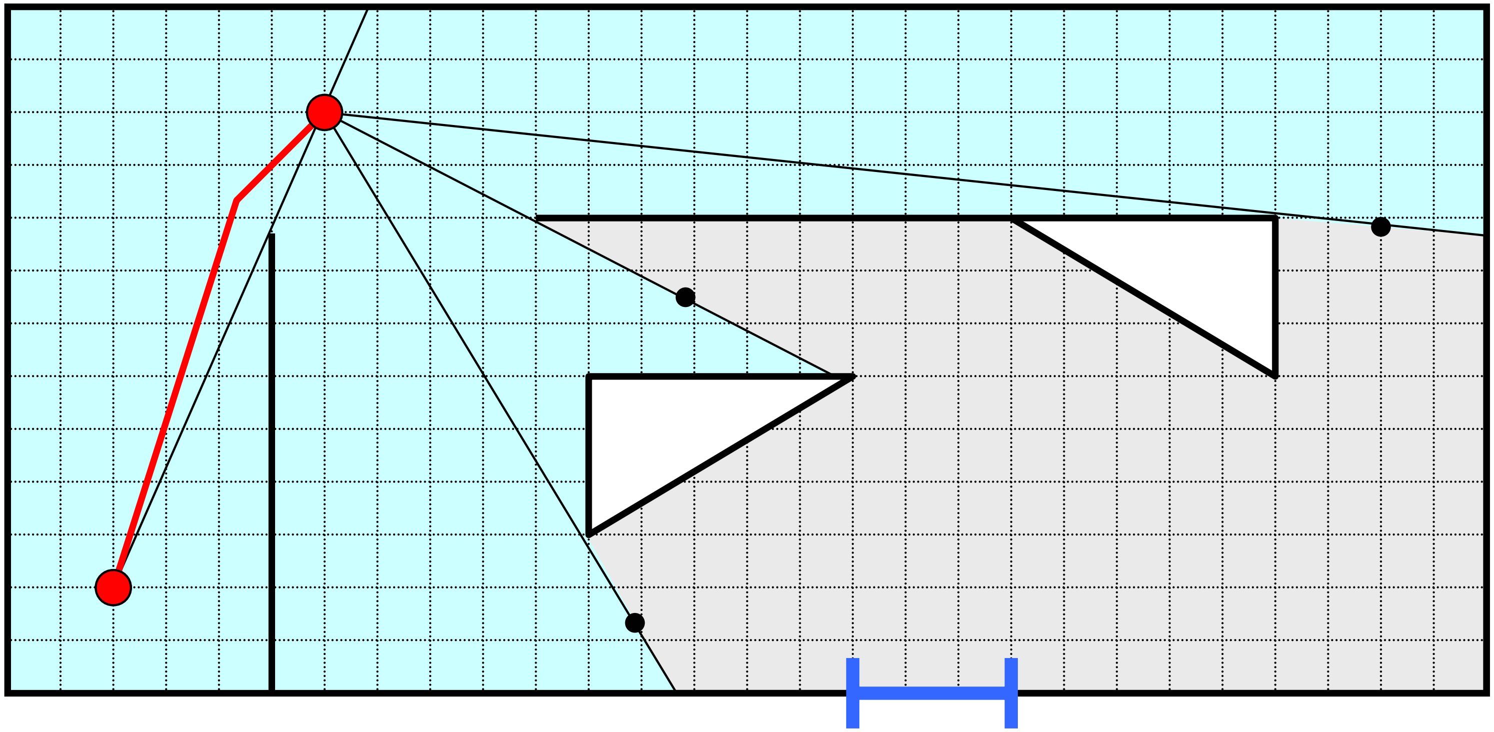
Robot uses sensors to create local map of the world and identify unexplored areas

Exploring the playing field using model-plan-act approach



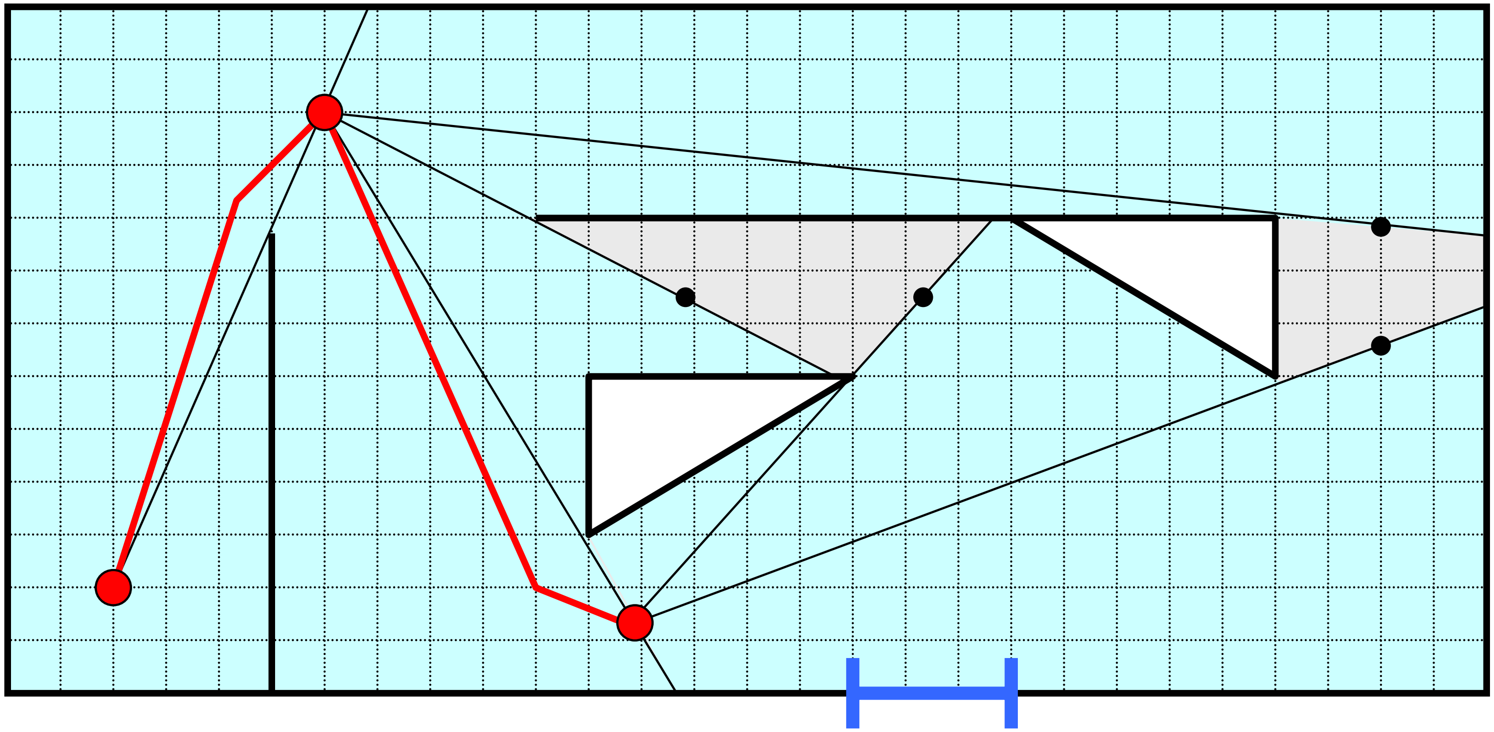
Robot moves to midpoint of unexplored boundary

Exploring the playing field using model-plan-act approach



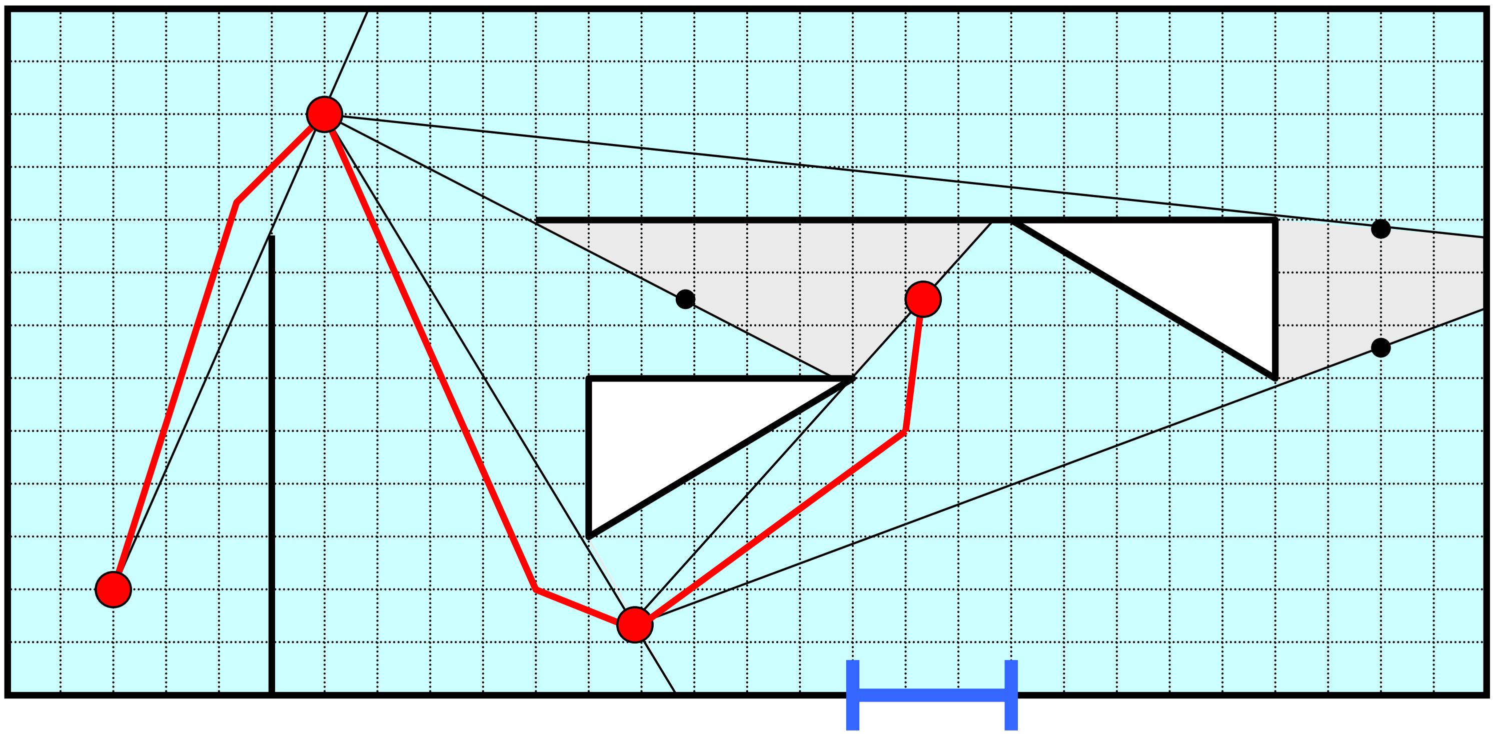
Robot performs a second sensor scan and must align the new data with the global map

Exploring the playing field using model-plan-act approach



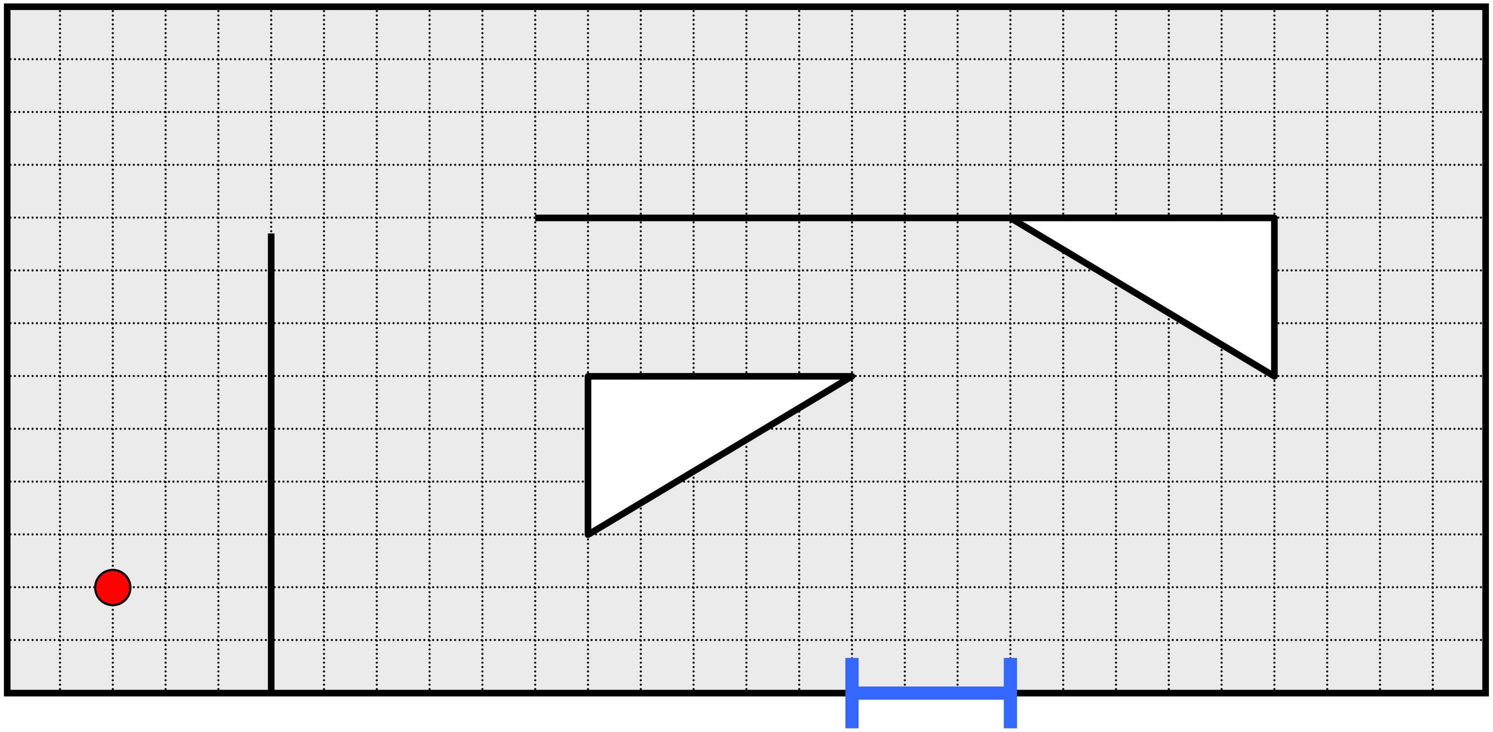
Robot continues to explore the playing field

Exploring the playing field using model-plan-act approach



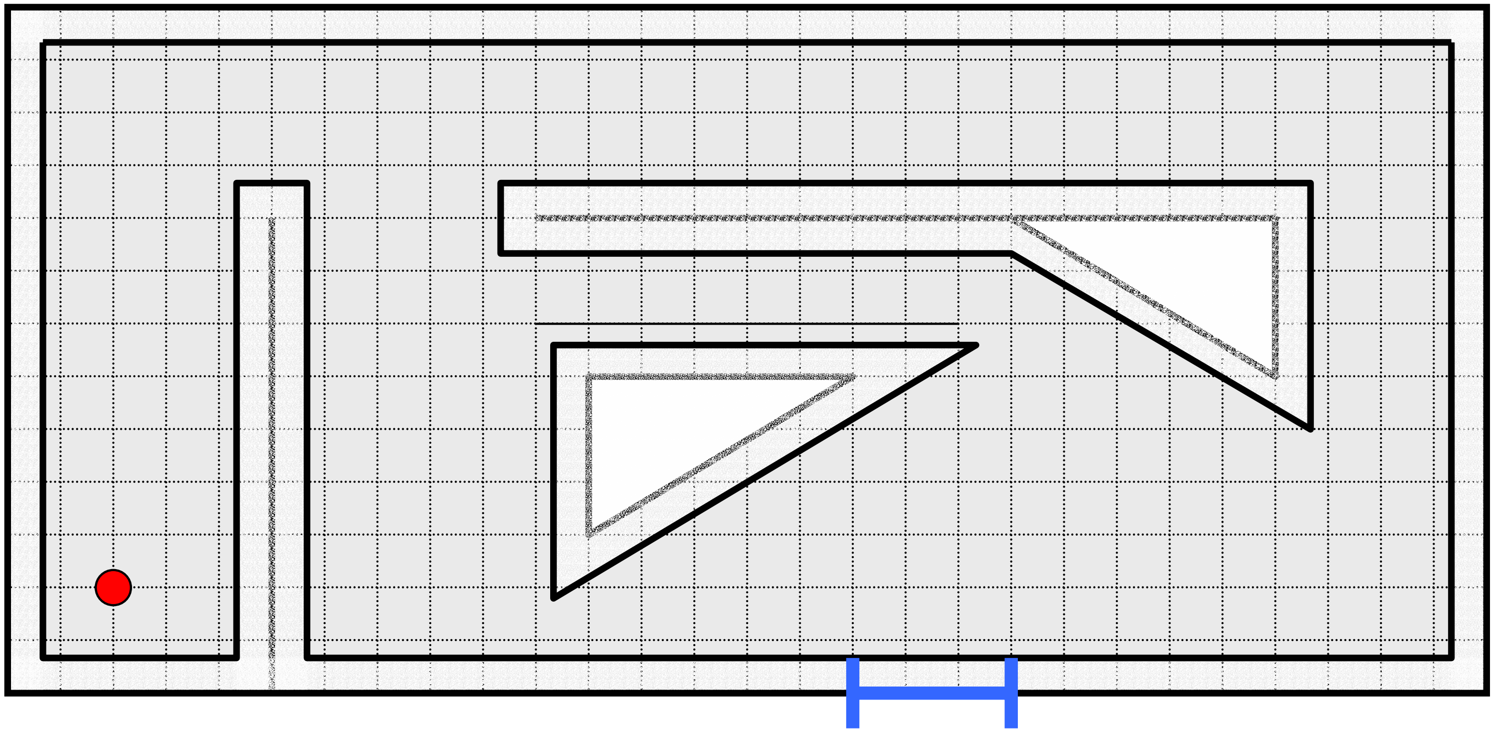
Robot must recognize when it starts to see areas which it has already explored

Finding a mousehole using model-plan-act approach



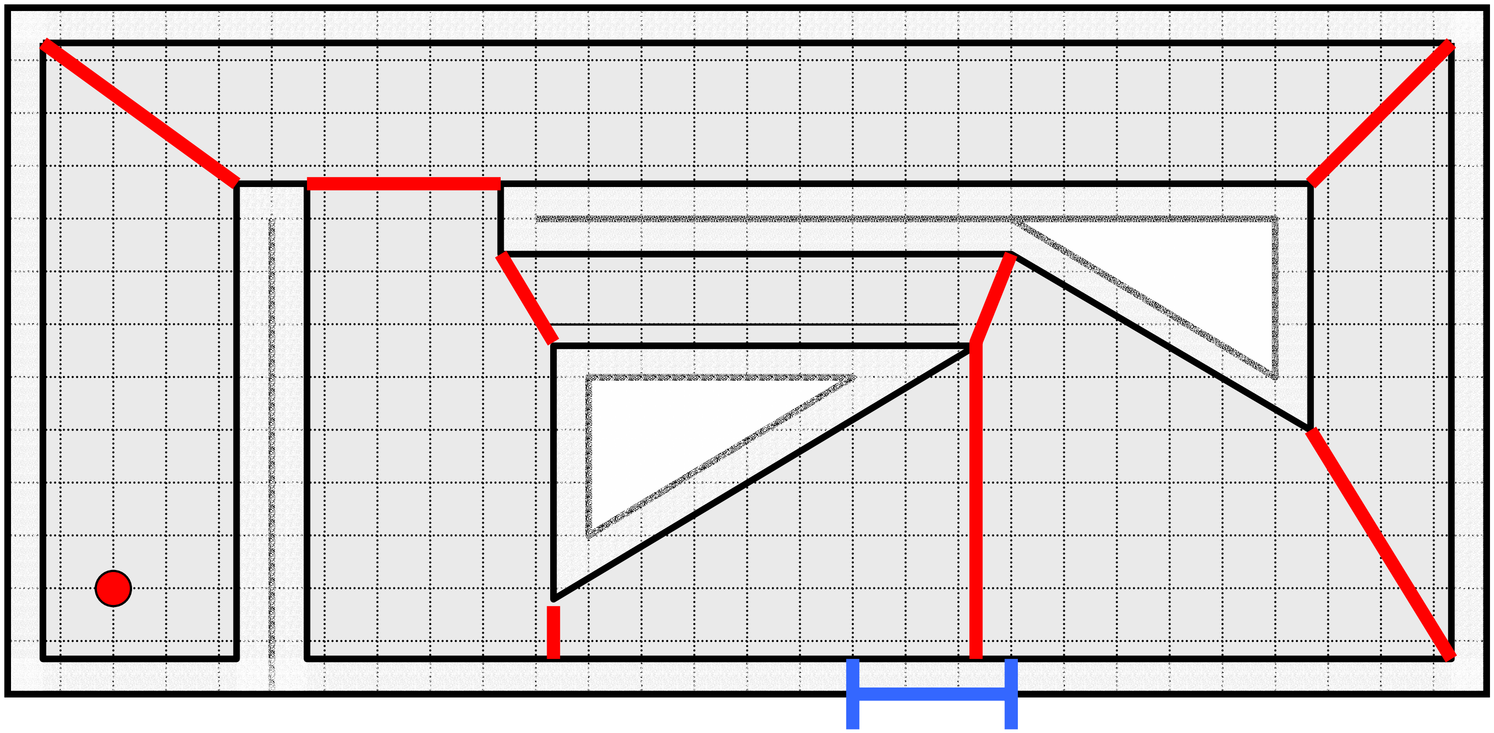
Given the global map,
the goal is to find the mousehole

Finding a mousehole using model-plan-act approach



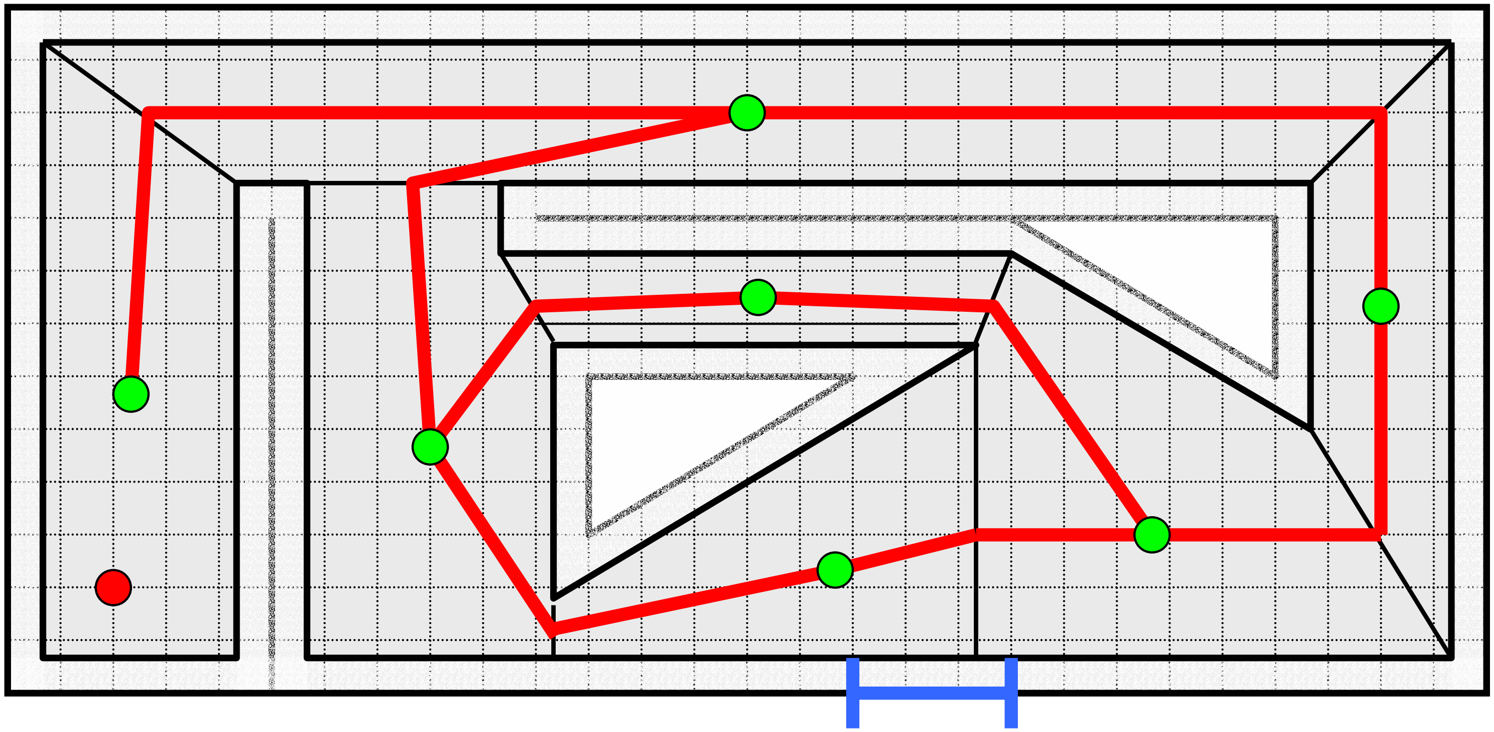
Transform world into configuration space by convolving robot with all obstacles

Finding a mousehole using model-plan-act approach



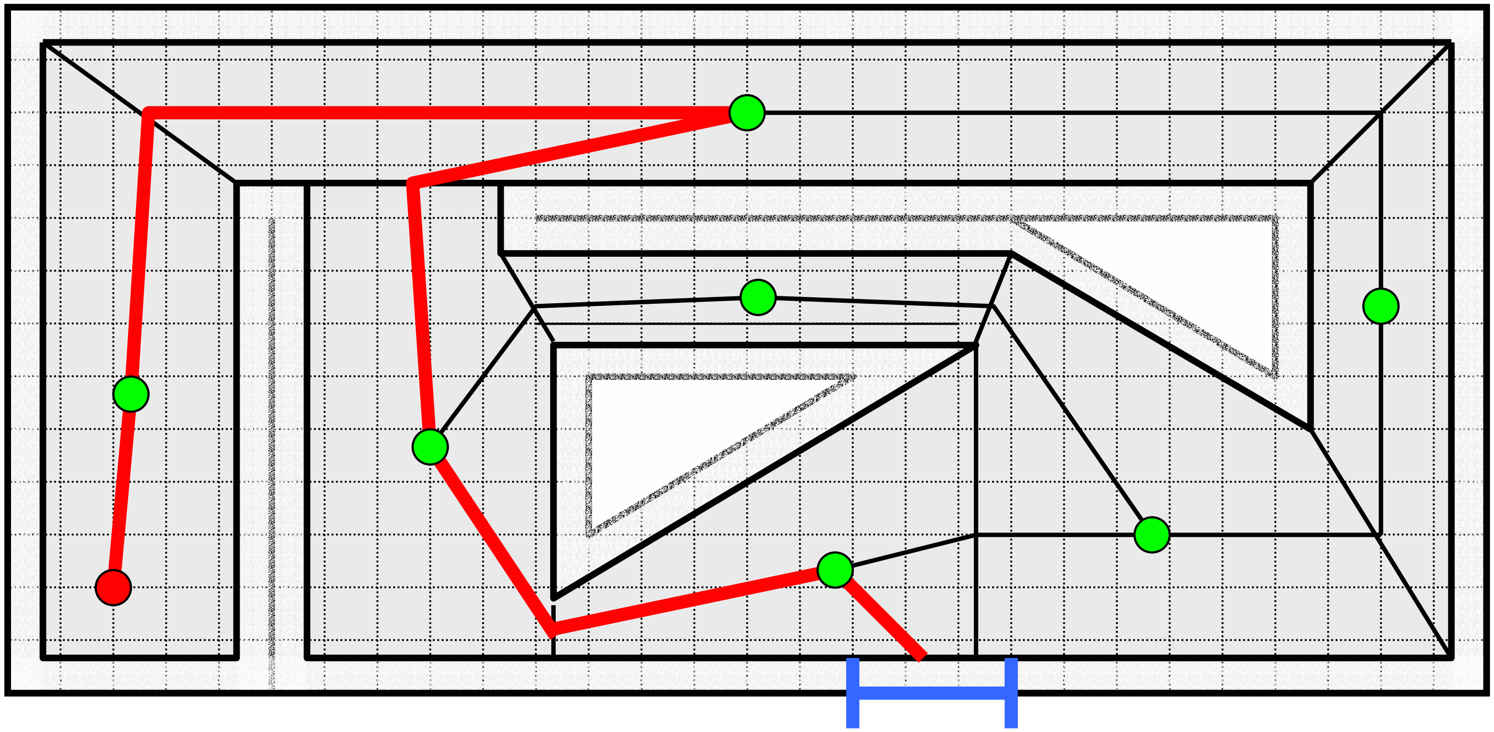
Decompose world into convex cells
Trajectory within any cell is free of obstacles

Finding a mousehole using model-plan-act approach



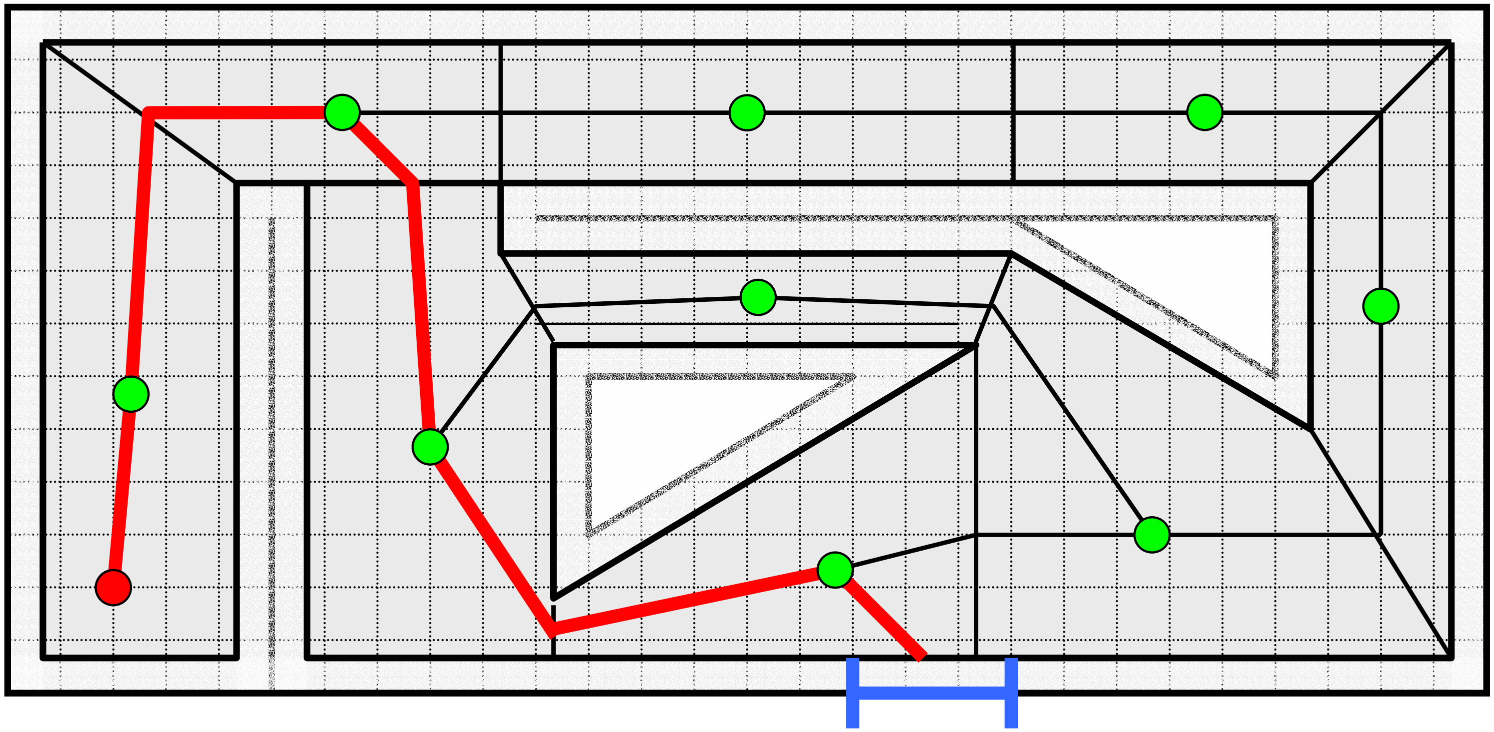
Connect cell edge midpoints and centroids to get graph of all possible paths

Finding a mousehole using model-plan-act approach



Use an algorithm (such as the A* algorithm) to find shortest path to goal

Finding a mousehole using model-plan-act approach



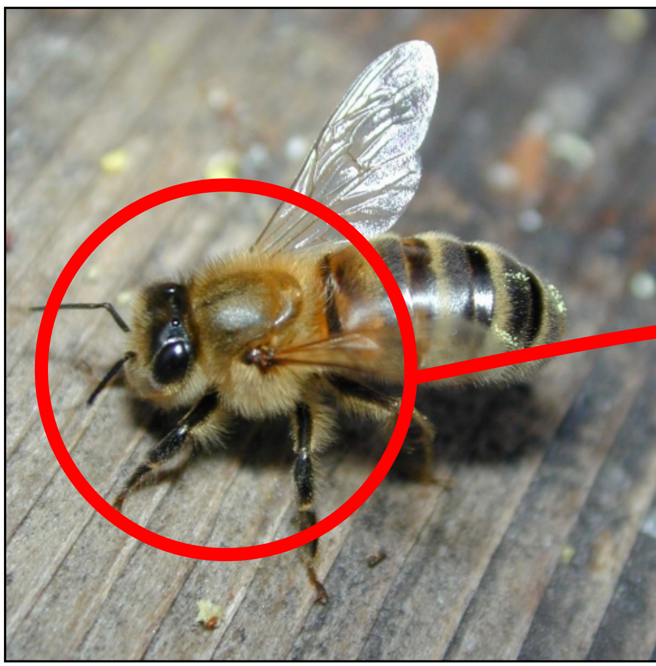
The choice of cell decomposition can greatly influence results

Advantages and disadvantages of the model-plan-act approach

- Advantages
  - Global knowledge in the model enables optimization
  - Can make provable guarantees about the plan
- Disadvantages
  - Must implement all functional units before any testing
  - Computationally intensive
  - Requires very good sensor data for accurate models
  - Models are inherently an approximation
  - Works poorly in dynamic environments

Emergent Approach

Living creatures like honey bees are able to explore their surroundings and locate a target (honey)



Used with permission, © William Connolley
http://wnconnolley.ork.uk

Is this bee using the model-plan-act approach?

Emergent Approach

Living creatures like honey bees are able to explore their surroundings and locate a target (honey)

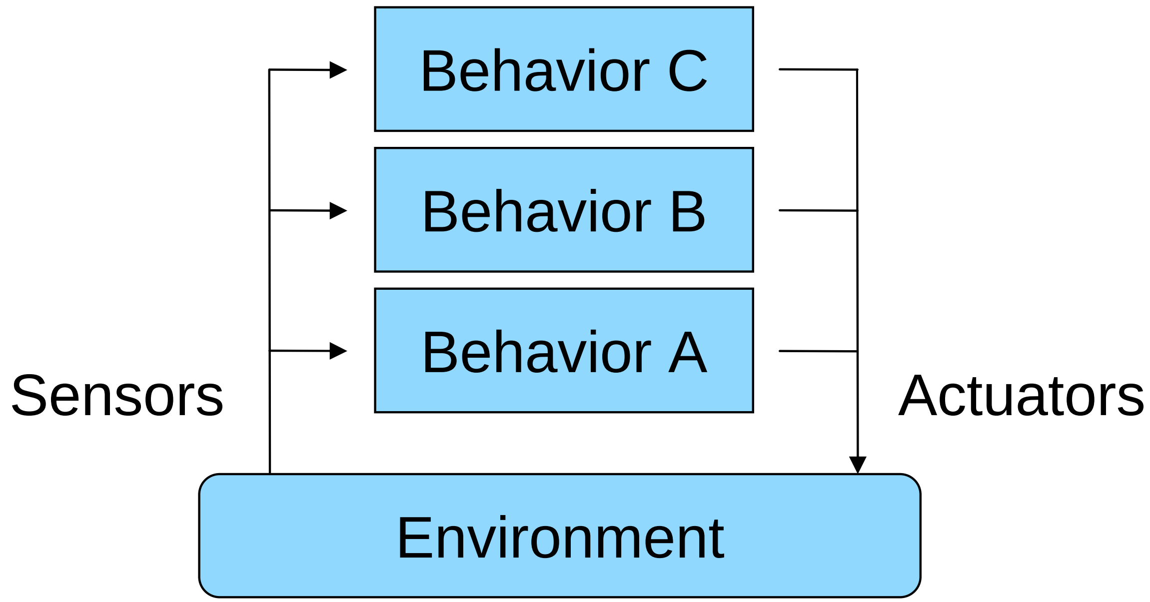
Used with permission, © William Connolley
http://wnconnolley.ork.uk

Probably not! Most likely bees layer simple reactive behaviors to create a complex emergent behavior

Emergent Approach

Should we design our robots so they act less like robots and more like honey bees?

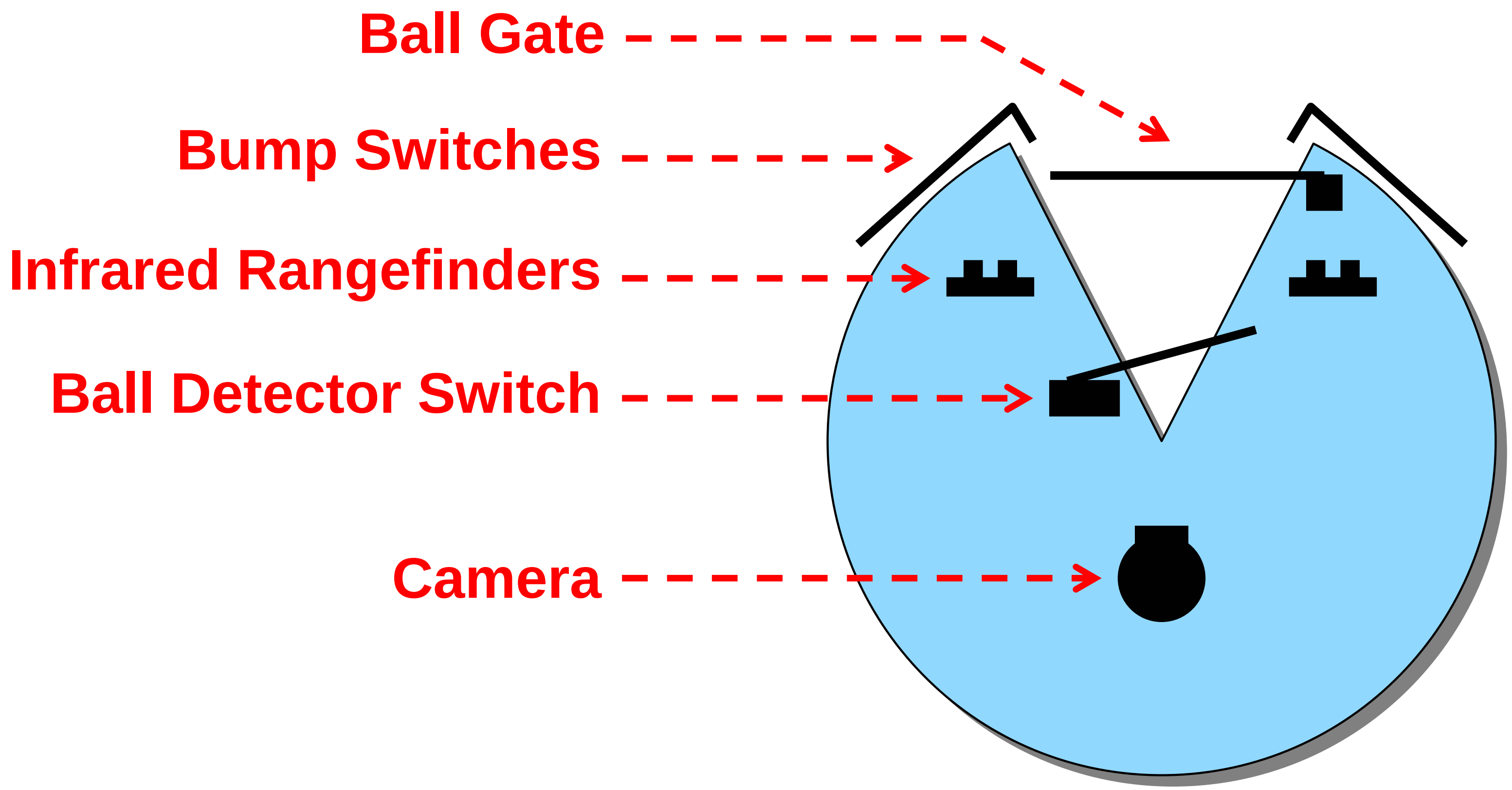
Emergent Approach



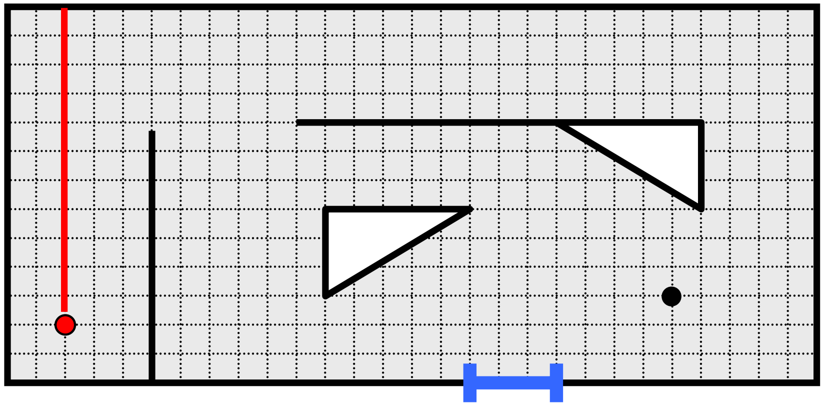
As in biological systems, the emergent approach uses simple behaviors to directly couple sensors and actuators

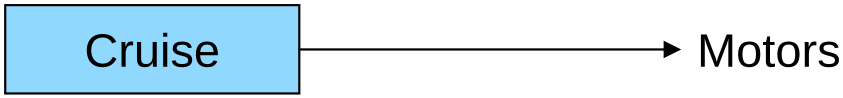
Higher level behaviors are layered on top of lower level behaviors

To illustrate the emergent approach we will consider a simple mobile robot



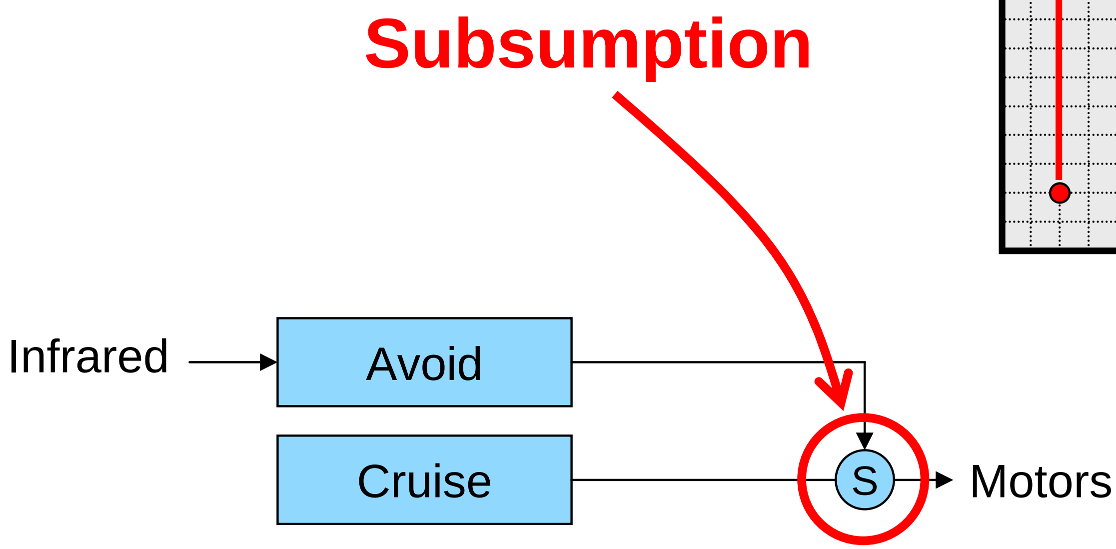
Layering simple behaviors can create much more complex emergent behavior

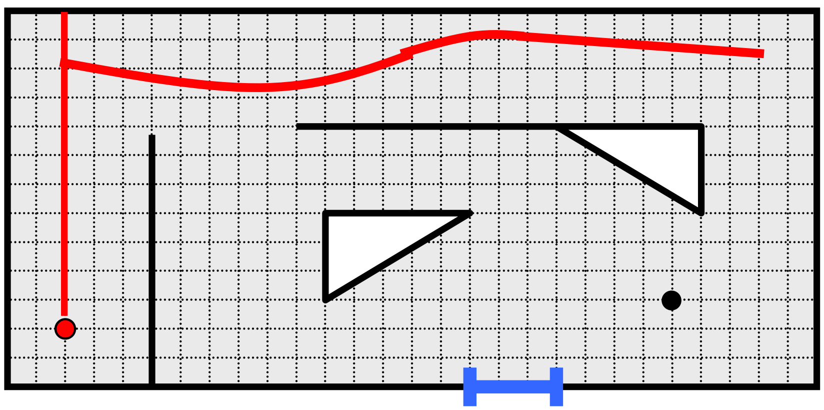




Cruise behavior simply moves robot forward

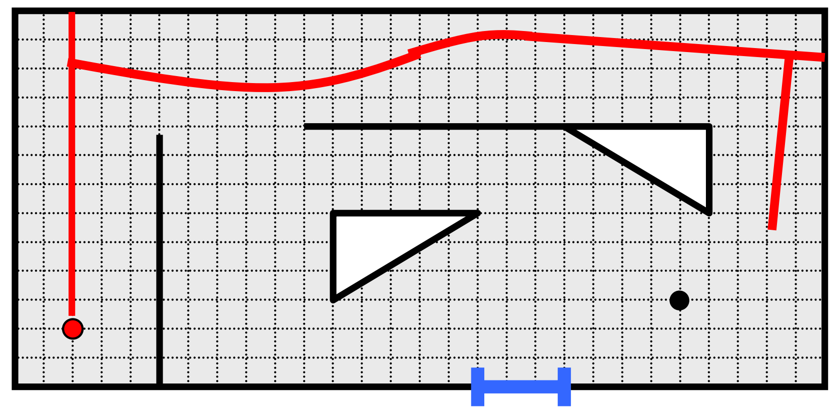
Layering simple behaviors can create much more complex emergent behavior

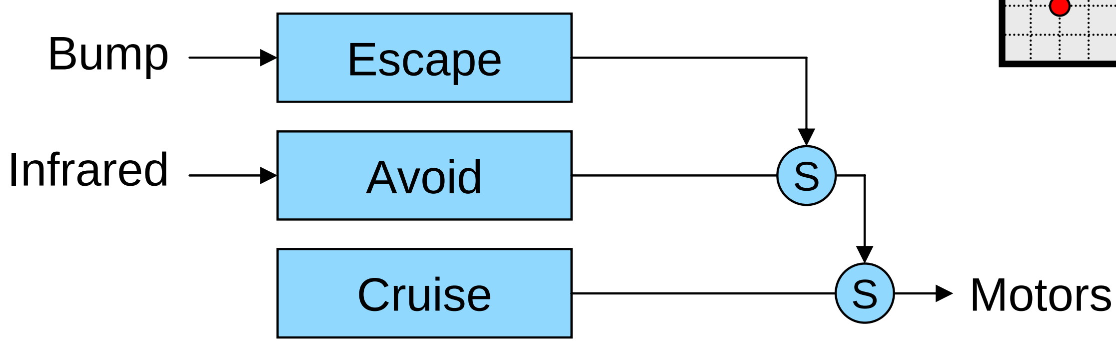




Left motor speed inversely proportional to left IR range
Right motor speed inversely proportional to right IR range
If both IR < threshold stop and turn right 120 degrees

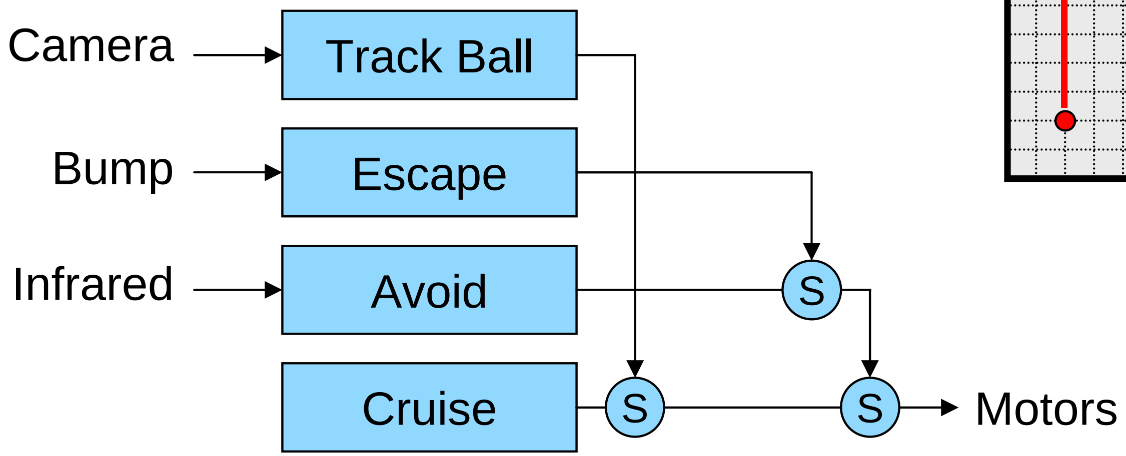
Layering simple behaviors can create much more complex emergent behavior

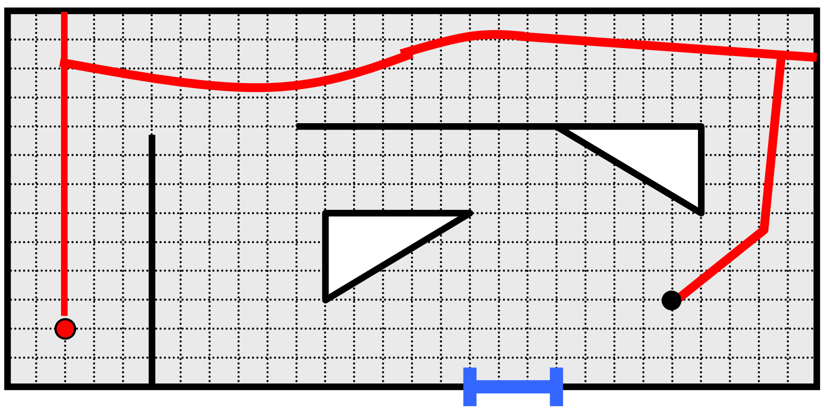




Escape behavior stops motors,
backs up a few inches, and turns right 90 degrees

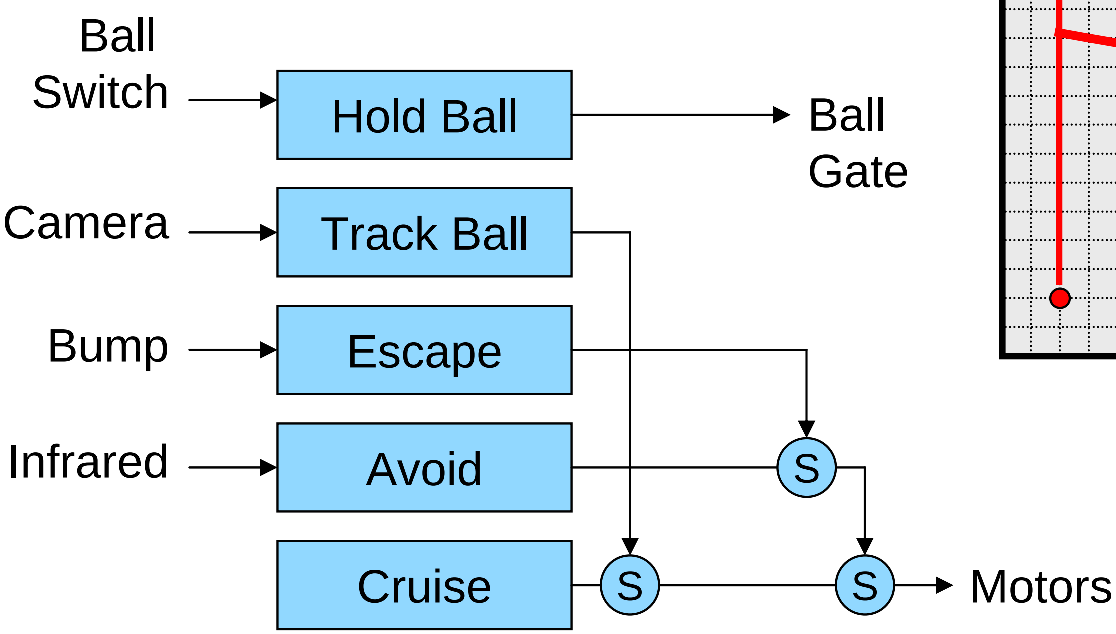
Layering simple behaviors can create much more complex emergent behavior

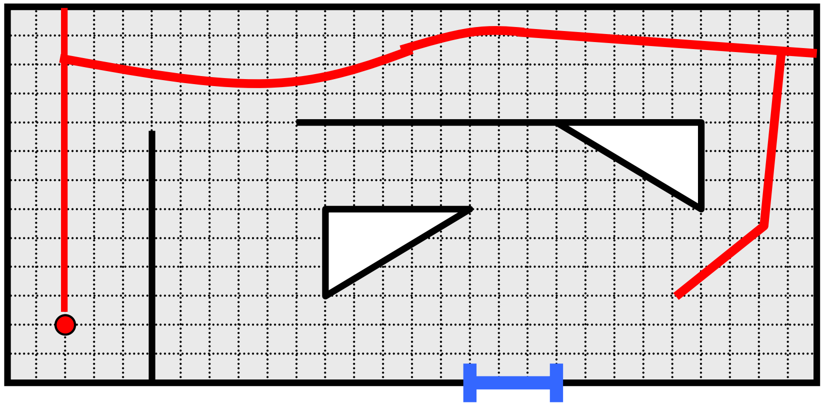




The track ball behavior adjusts the
motor differential to steer the robot towards the ball

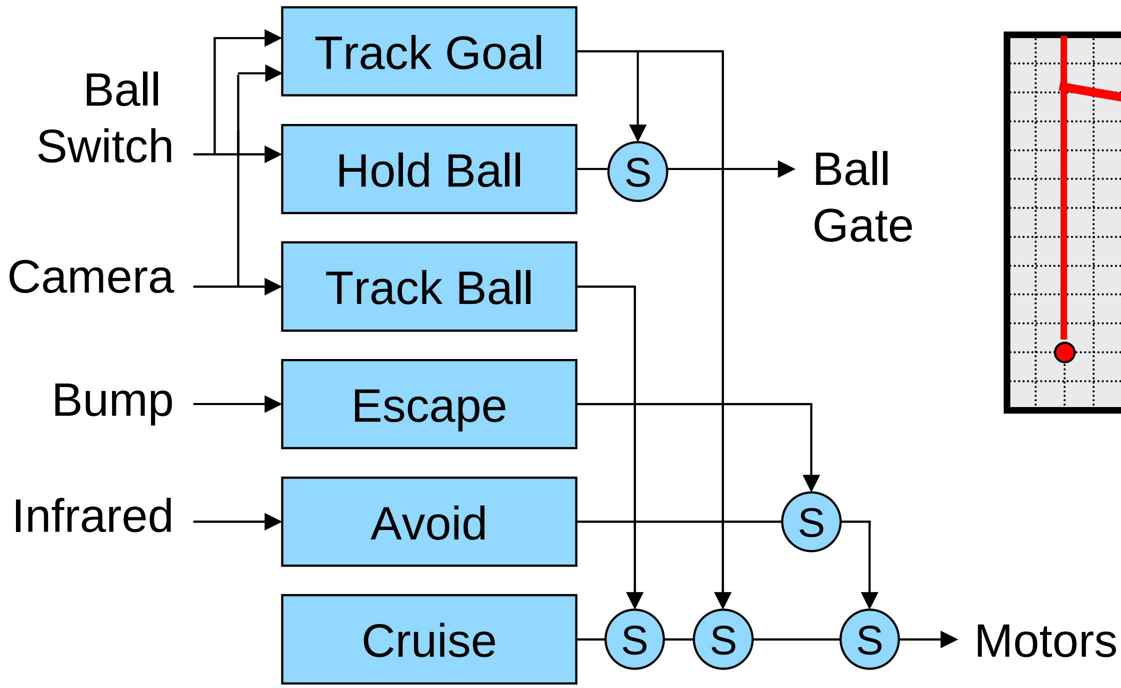
Layering simple behaviors can create much more complex emergent behavior

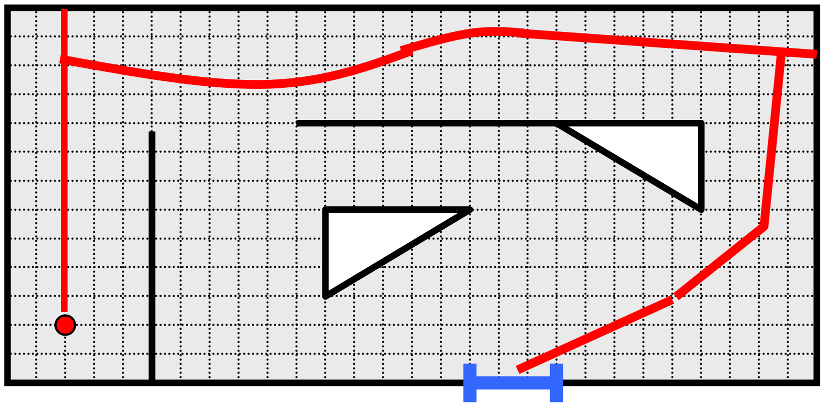




Hold ball behavior simply closes ball gate when ball switch is depressed

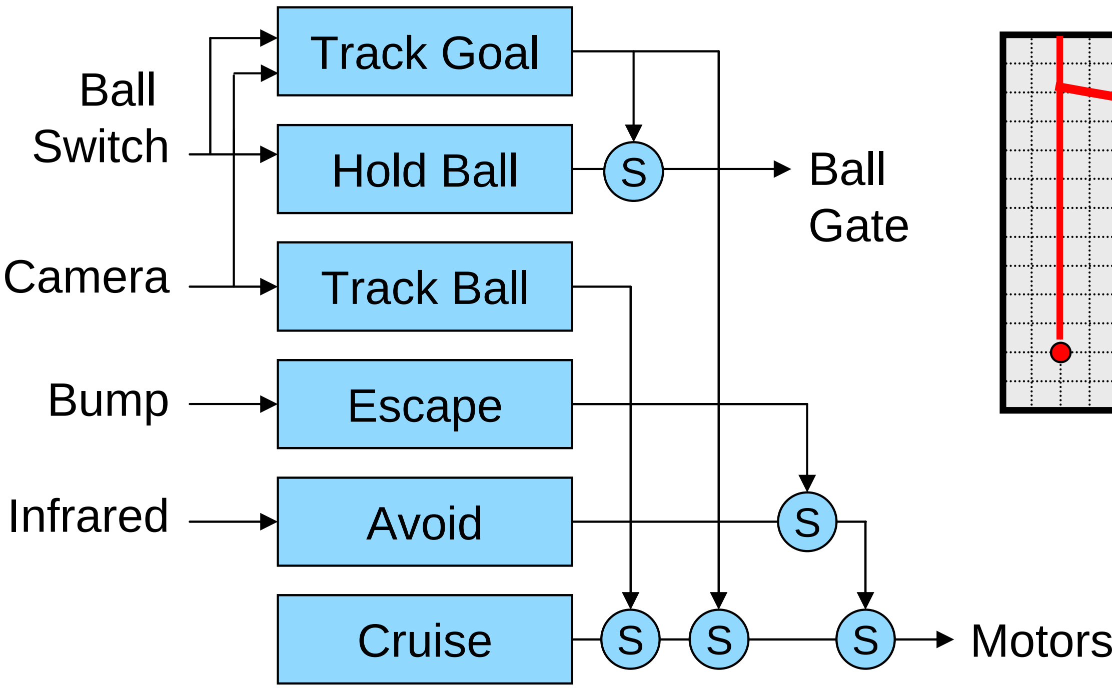
Layering simple behaviors can create much more complex emergent behavior

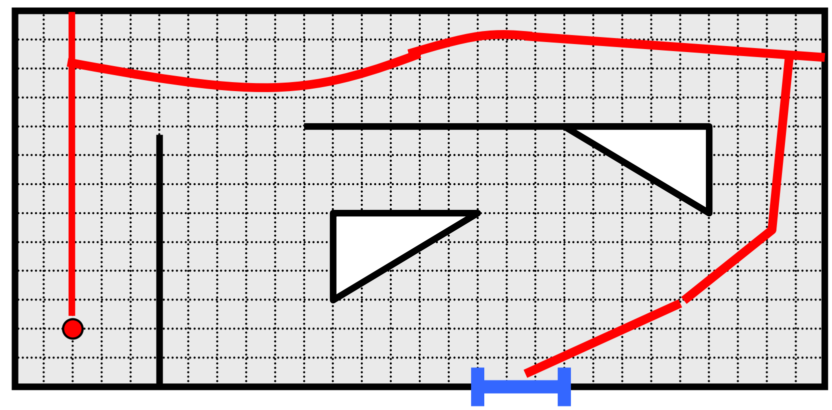




The track goal behavior opens the ball gate and adjusts the motor differential to steer the robot towards the goal

Layering simple behaviors can create much more complex emergent behavior



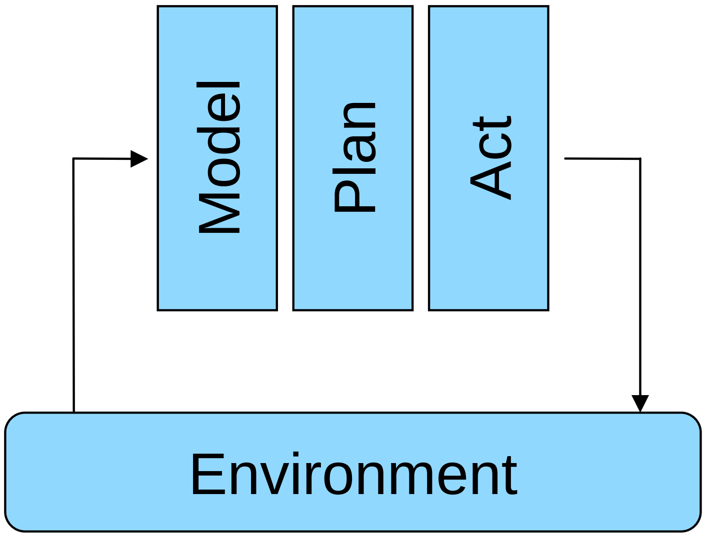


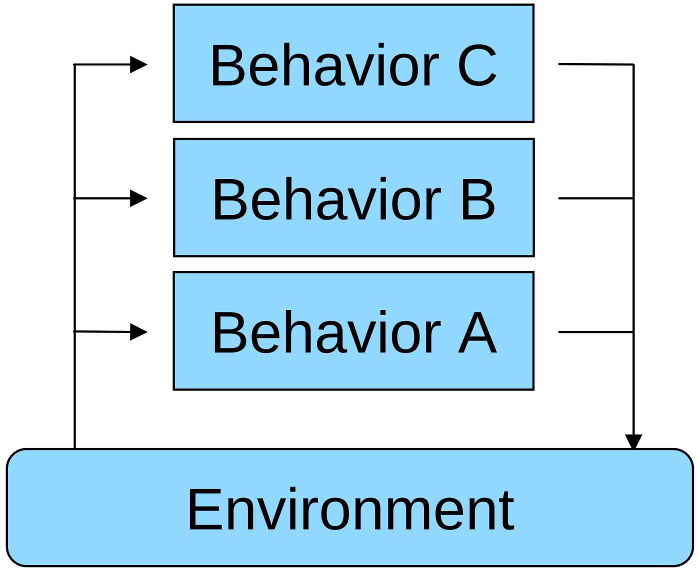
All behaviors are always running in parallel and an arbiter is responsible for picking which behavior can access the actuators

Advantages and disadvantages of the behavioral approach

- Advantages
  - Incremental development is very natural
  - Modularity makes experimentation easier
  - Cleanly handles dynamic environments
- Disadvantages
  - Difficult to judge what robot will actually do
  - No performance or completeness guarantees
  - Debugging can be very difficult

Model-plan-act fuses sensor data, while emergent fuses behaviors

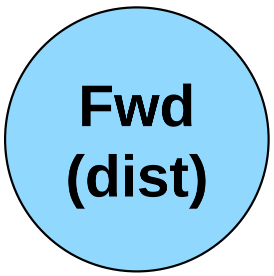




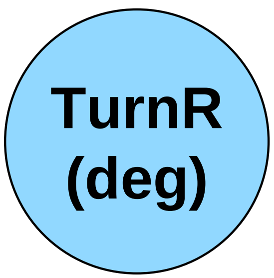
Model-Plan-Act	Emergent
Lots of internal state	Very little internal state
Lots of preliminary planning	No preliminary planning
Fixed plan of behaviors	Layered behaviors

Finite State Machines offer another alternative for combining behaviors

FSMs have some preliminary planning and some state. Some transitions between behaviors are decided statically while others are decided dynamically.

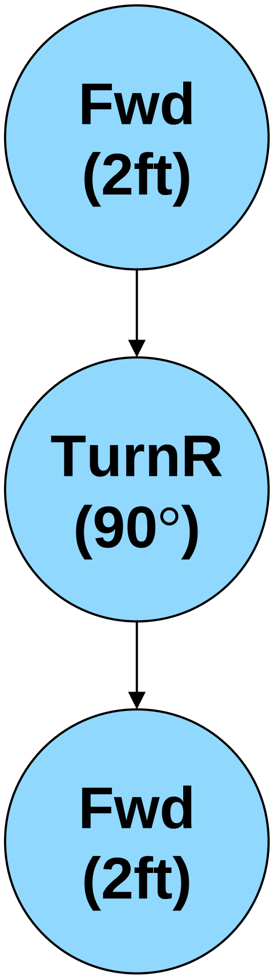


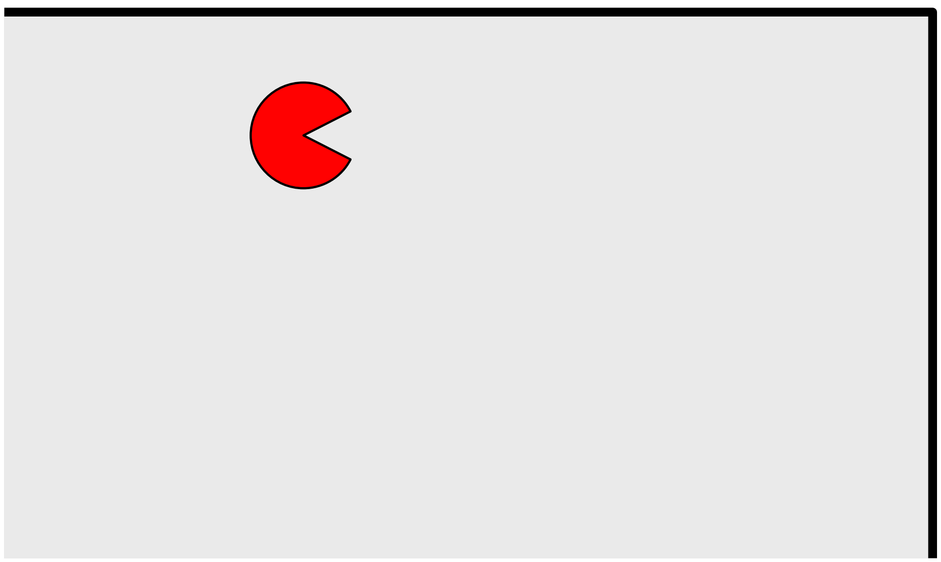
Fwd behavior moves robot straight forward a given distance



TurnR behavior turns robot to the right a given number of degrees

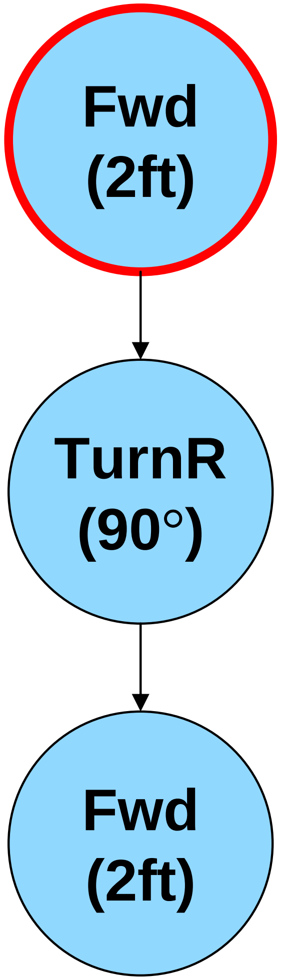
Finite State Machines offer another alternative for combining behaviors

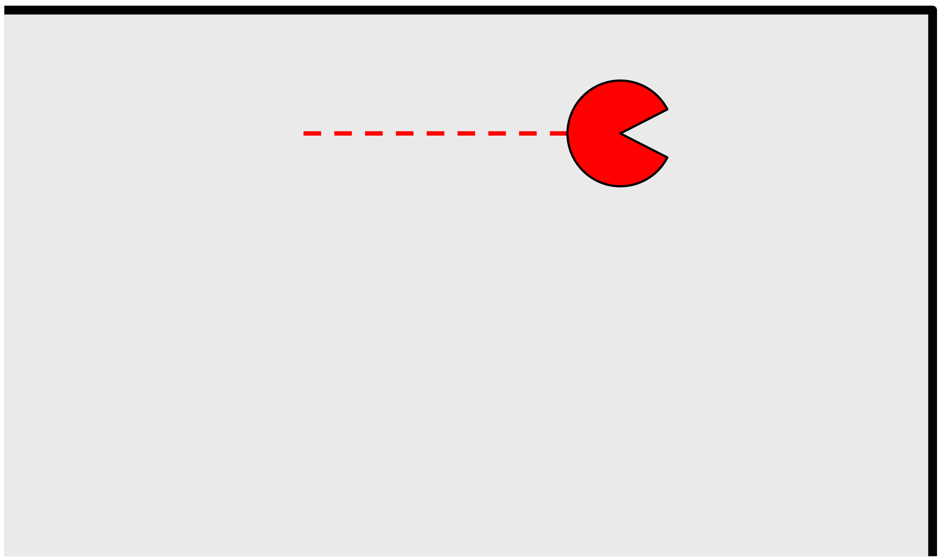




Each state is just a specific behavior instance - link them together to create an open loop control system

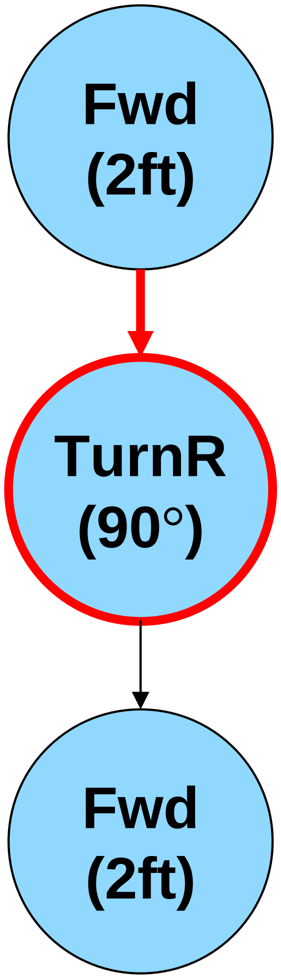
Finite State Machines offer another alternative for combining behaviors

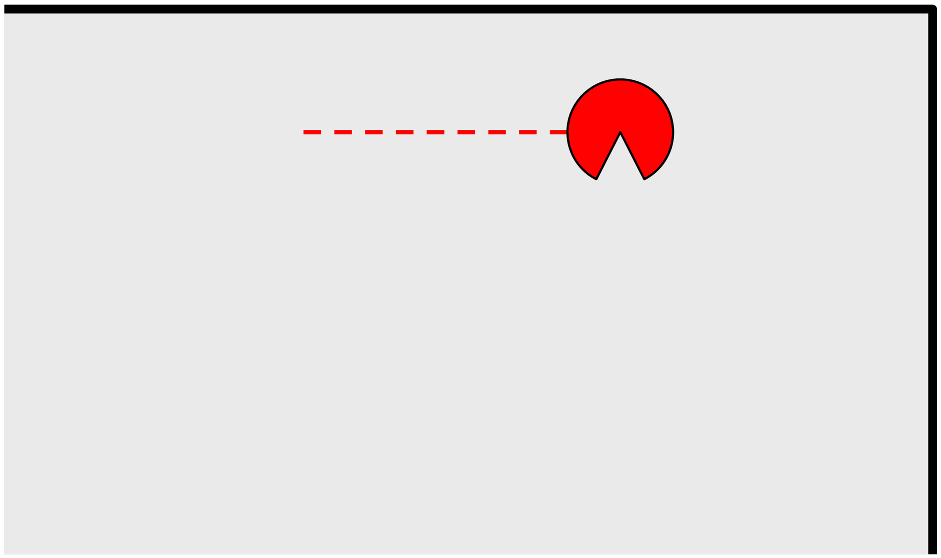




Each state is just a specific behavior instance - link them together to create an open loop control system

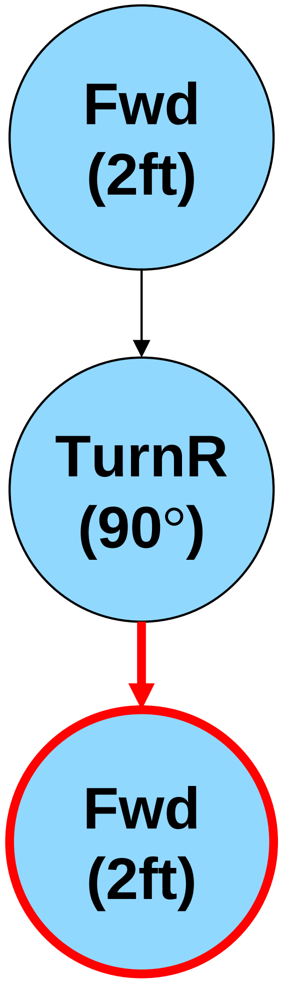
Finite State Machines offer another alternative for combining behaviors

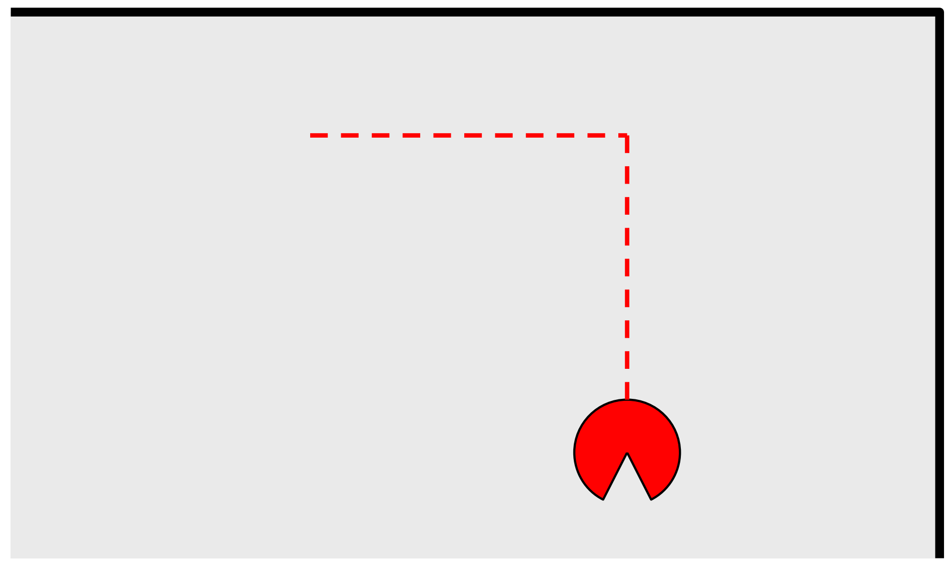




Each state is just a specific behavior instance - link them together to create an open loop control system

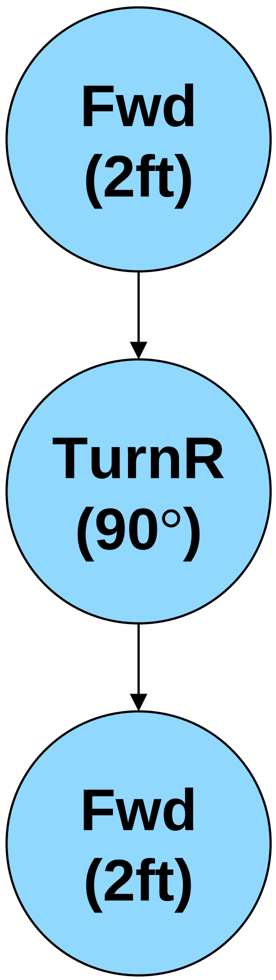
Finite State Machines offer another alternative for combining behaviors

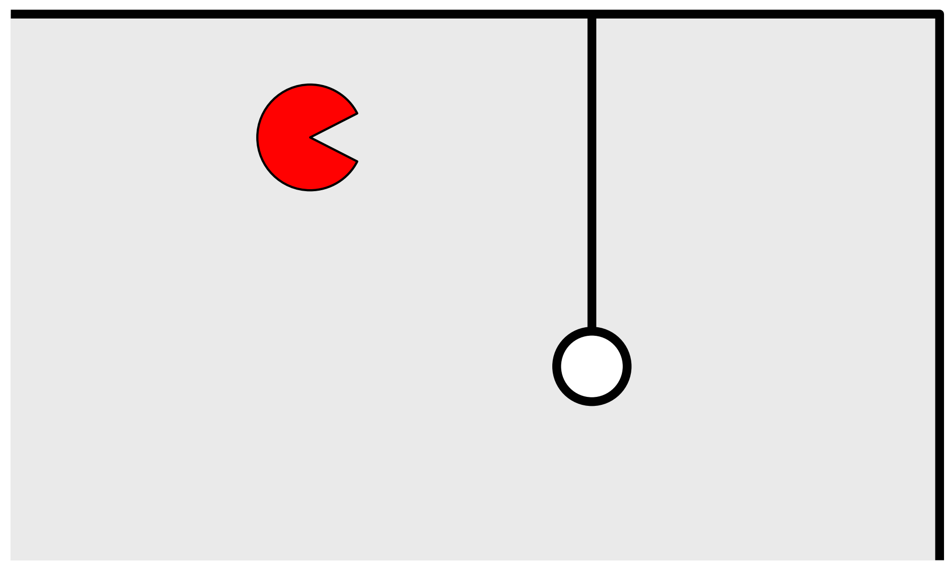




Each state is just a specific behavior instance - link them together to create an open loop control system

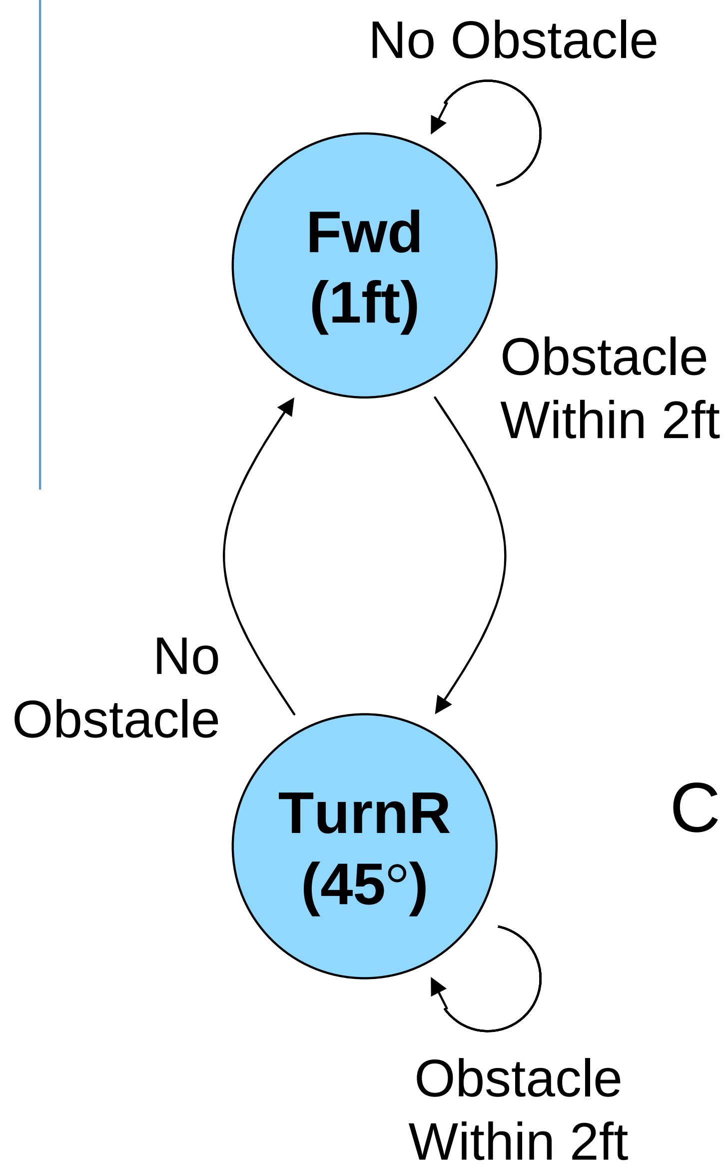
Finite State Machines offer another alternative for combining behaviors

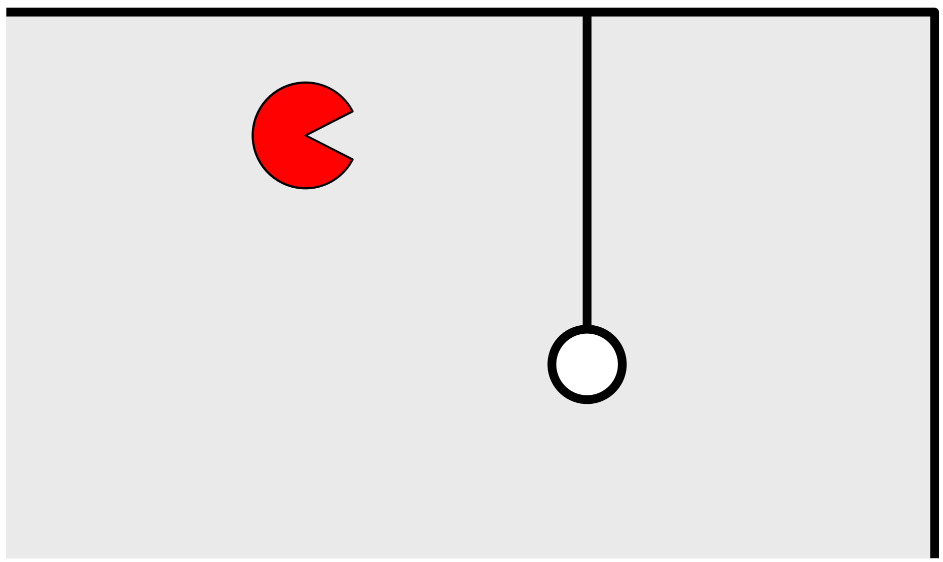




Since the Maslab playing field is unknown, open loop control systems have no hope of success!

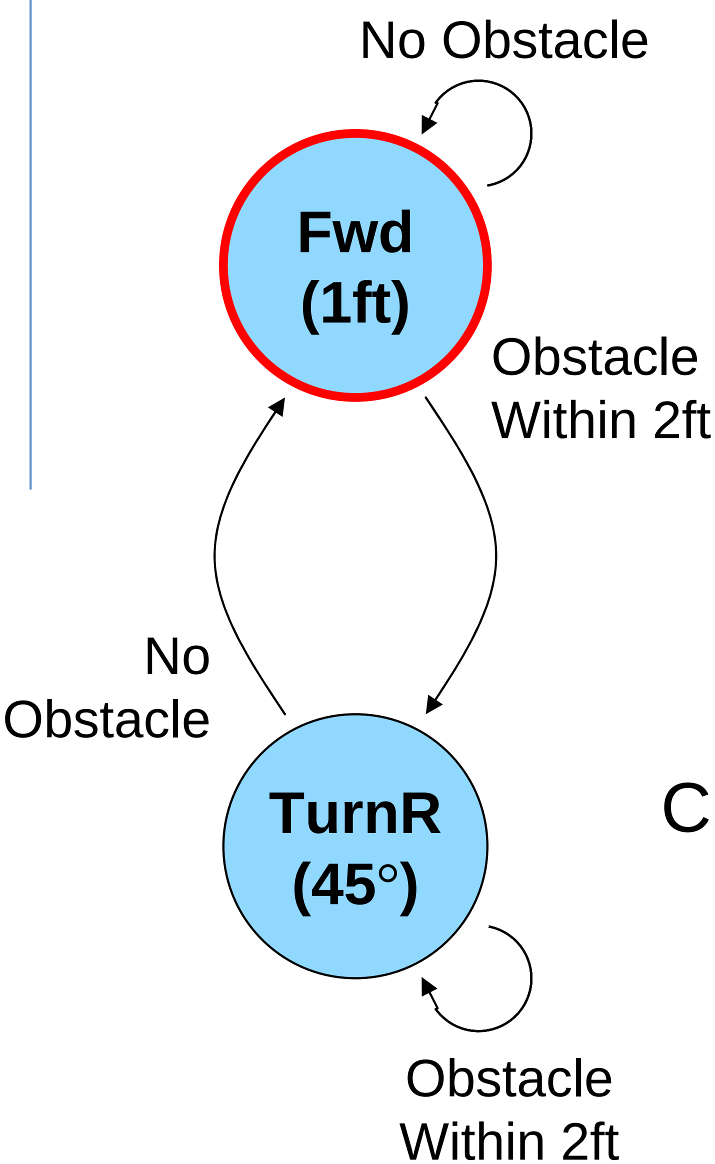
Finite State Machines offer another alternative for combining behaviors

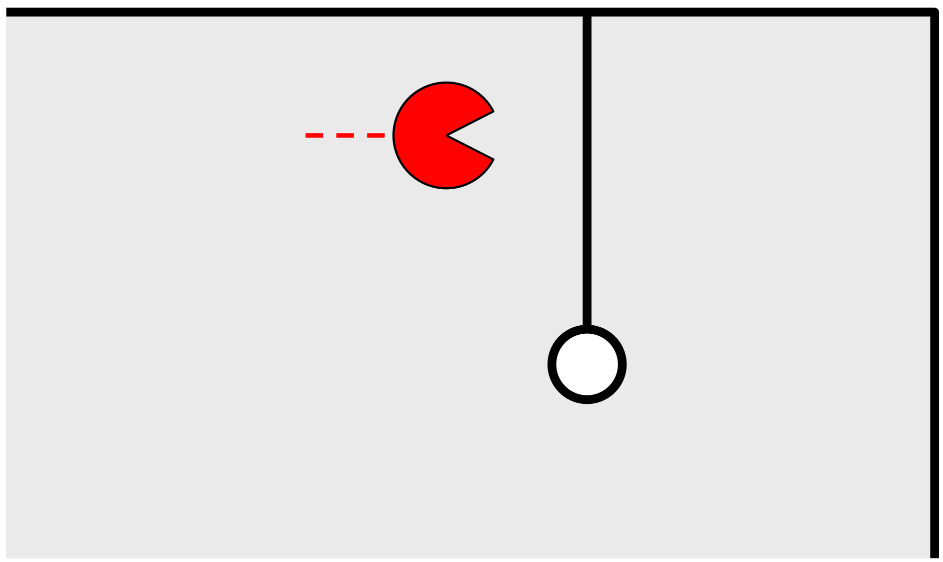




Closed loop finite state machines use sensor data as feedback to make state transitions

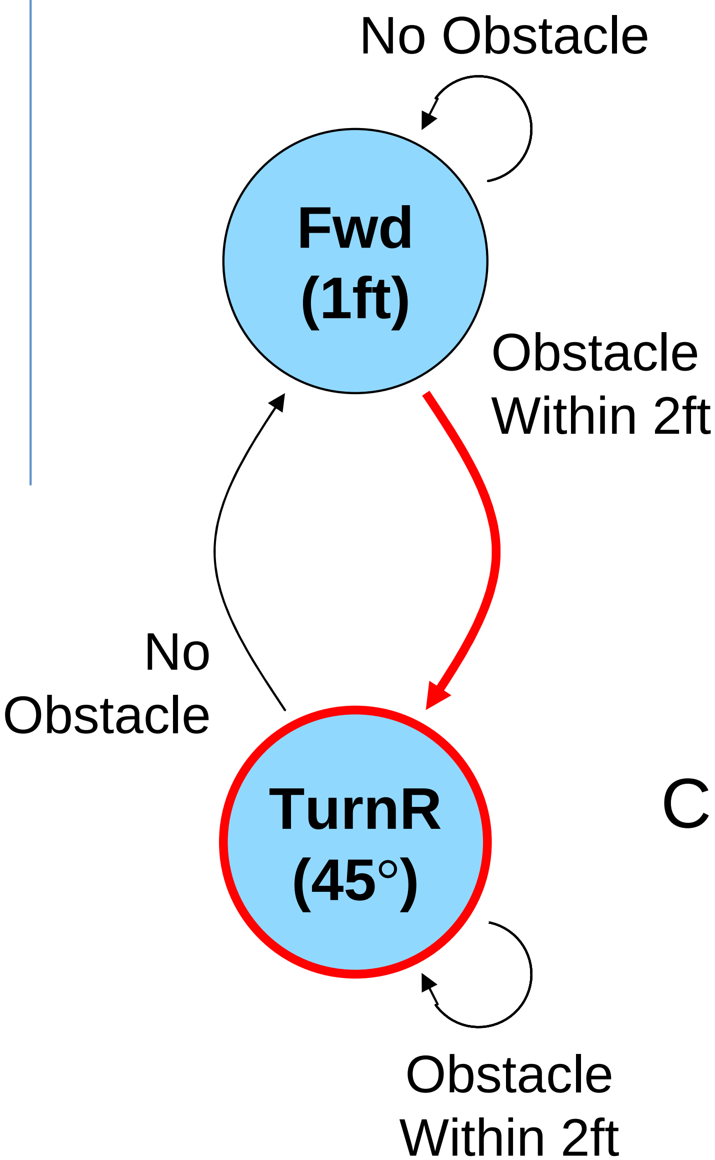
Finite State Machines offer another alternative for combining behaviors

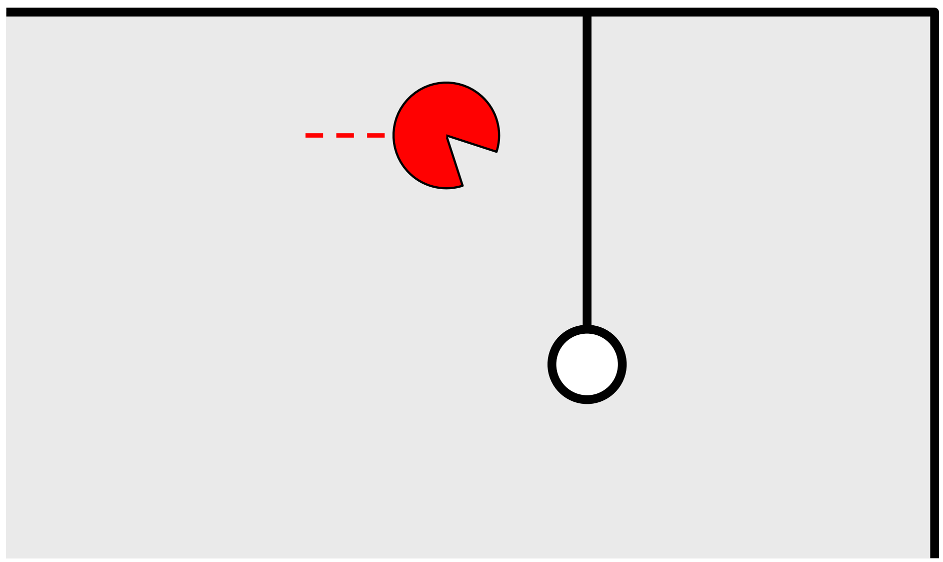




Closed loop finite state machines use sensor data as feedback to make state transitions

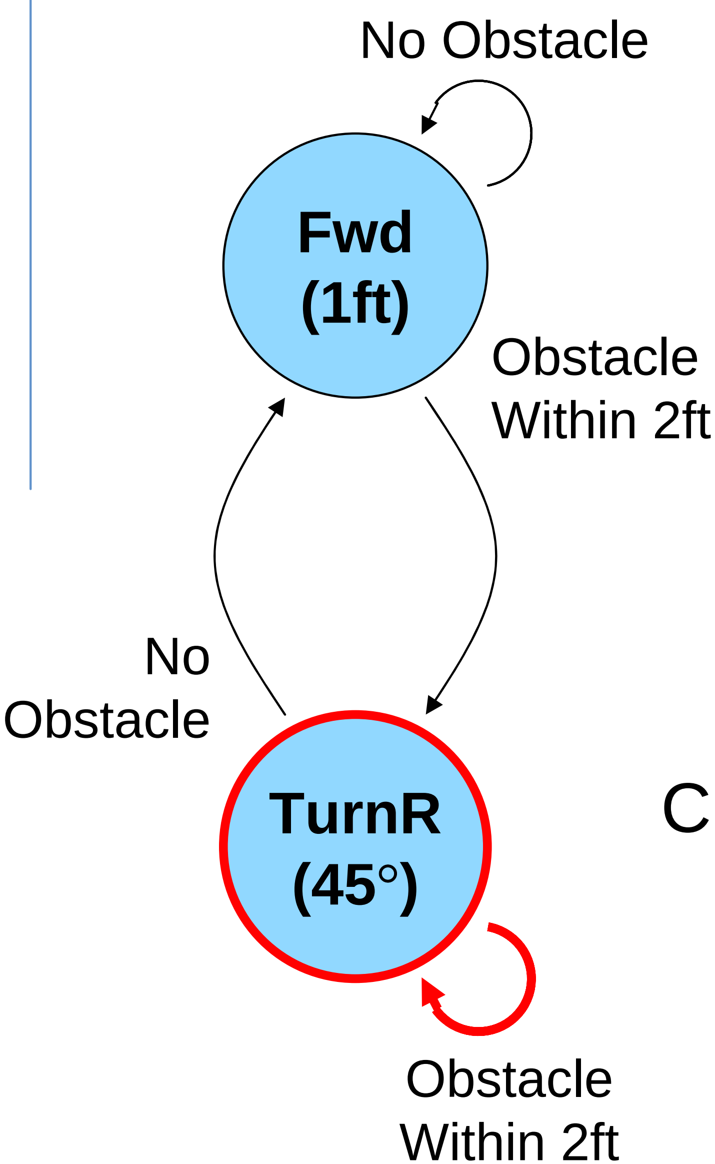
Finite State Machines offer another alternative for combining behaviors

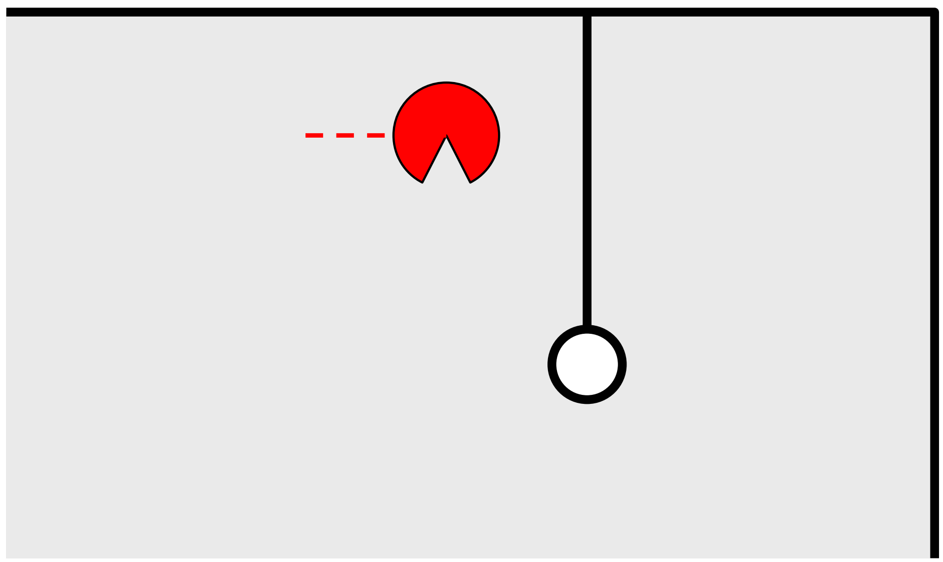




Closed loop finite state machines use sensor data as feedback to make state transitions

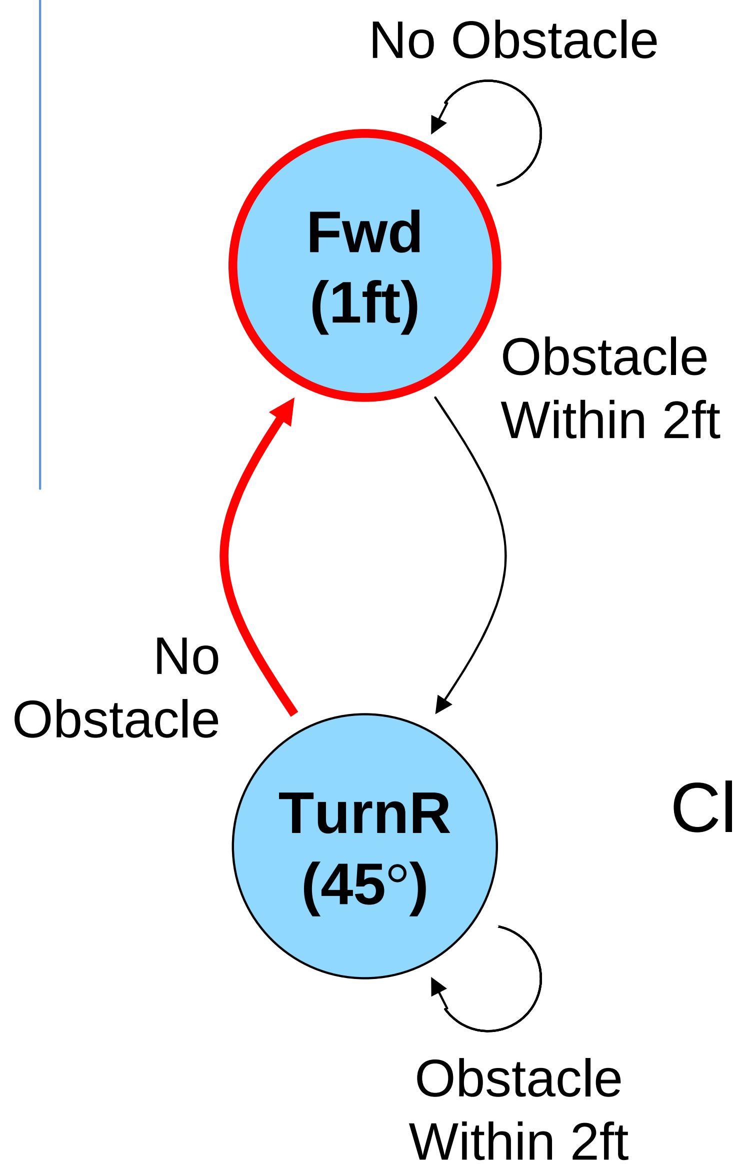
Finite State Machines offer another alternative for combining behaviors

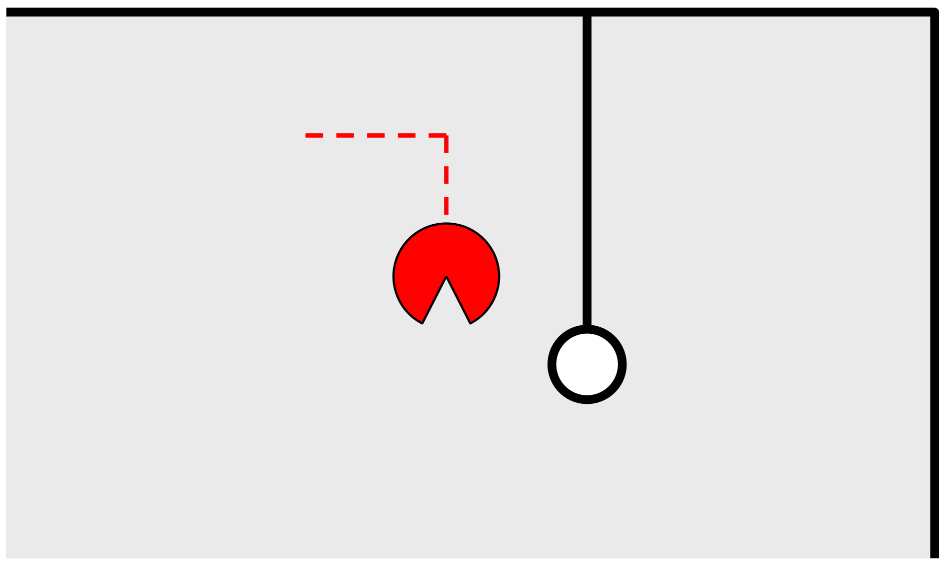




Closed loop finite state machines use sensor data as feedback to make state transitions

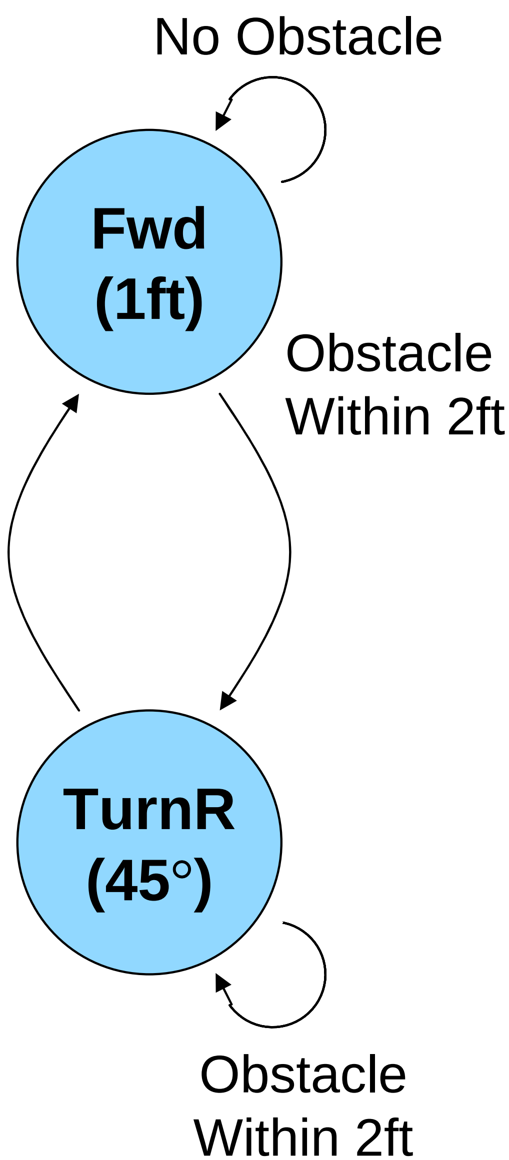
Finite State Machines offer another alternative for combining behaviors





Closed loop finite state machines use sensor data as feedback to make state transitions

Implementing a Finite State Machine in Java



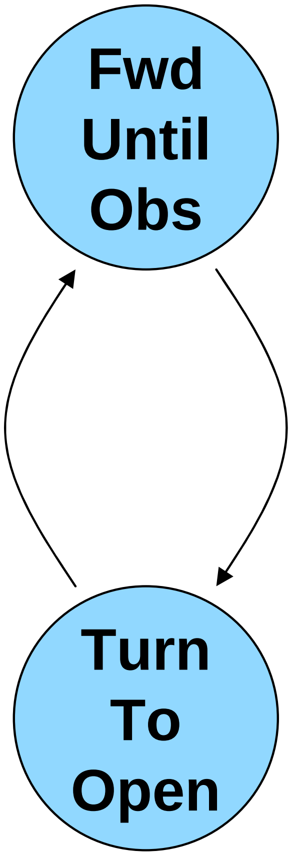

```
switch ( state ) {

  case States.Fwd_1 :
    moveFoward(1);
    if ( distanceToObstacle() < 2 )
      state = TurnR_45;
    break;

  case States.TurnR_45 :
    turnRight(45);
    if ( distanceToObstacle() >= 2 )
      state = Fwd_1;
    break;

}
```

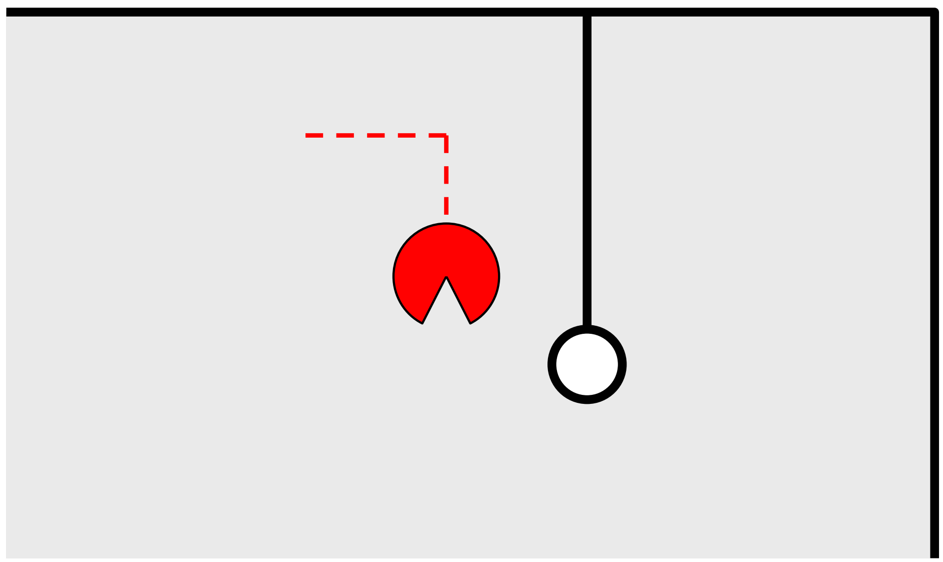
Implementing a FSM in Java

- Implement behaviors as parameterized functions
- Each case statement includes behavior instance and state transition
- Use enums for state variables

```
switch ( state ) {

  case States.Fwd_1 :
    moveFoward(1);
    if ( distanceToObstacle() < 2 )
      state = TurnR_45;
    break;

  case States.TurnR_45 :
    turnRight(45);
    if ( distanceToObstacle() >= 2 )
      state = Fwd_1;
    break;

}
```

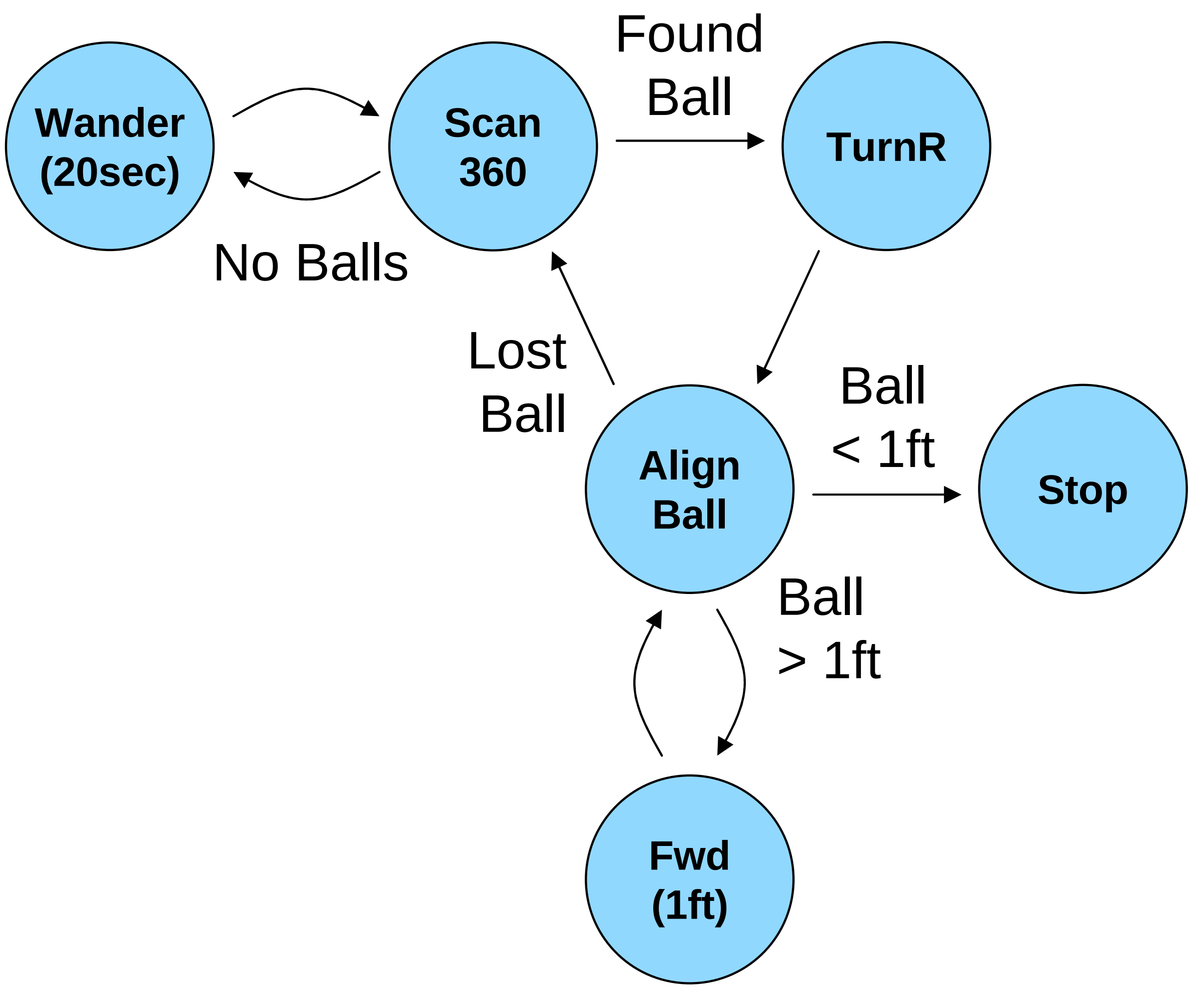
Finite State Machines offer another alternative for combining behaviors



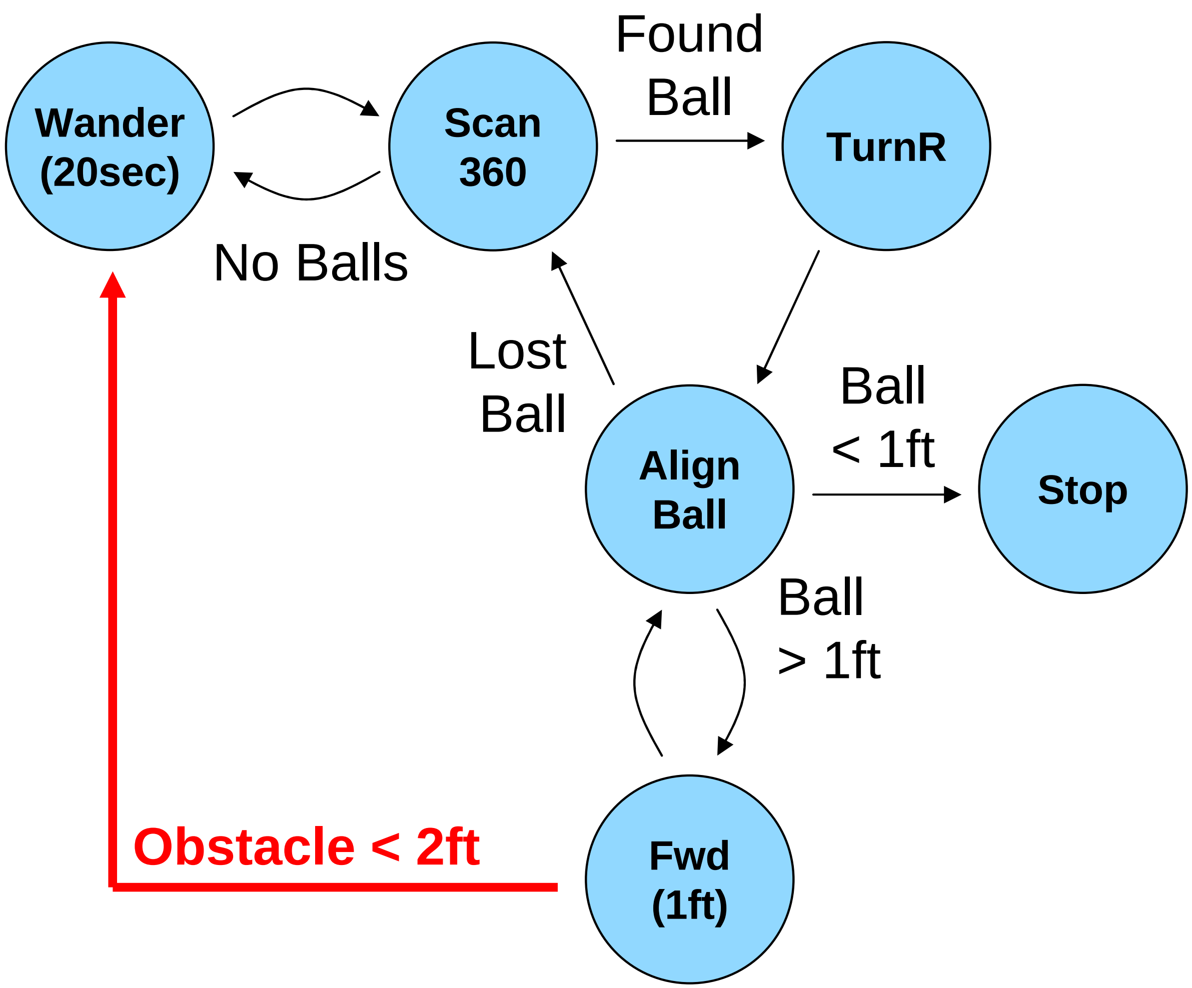


Can also fold closed loop feedback into the behaviors themselves

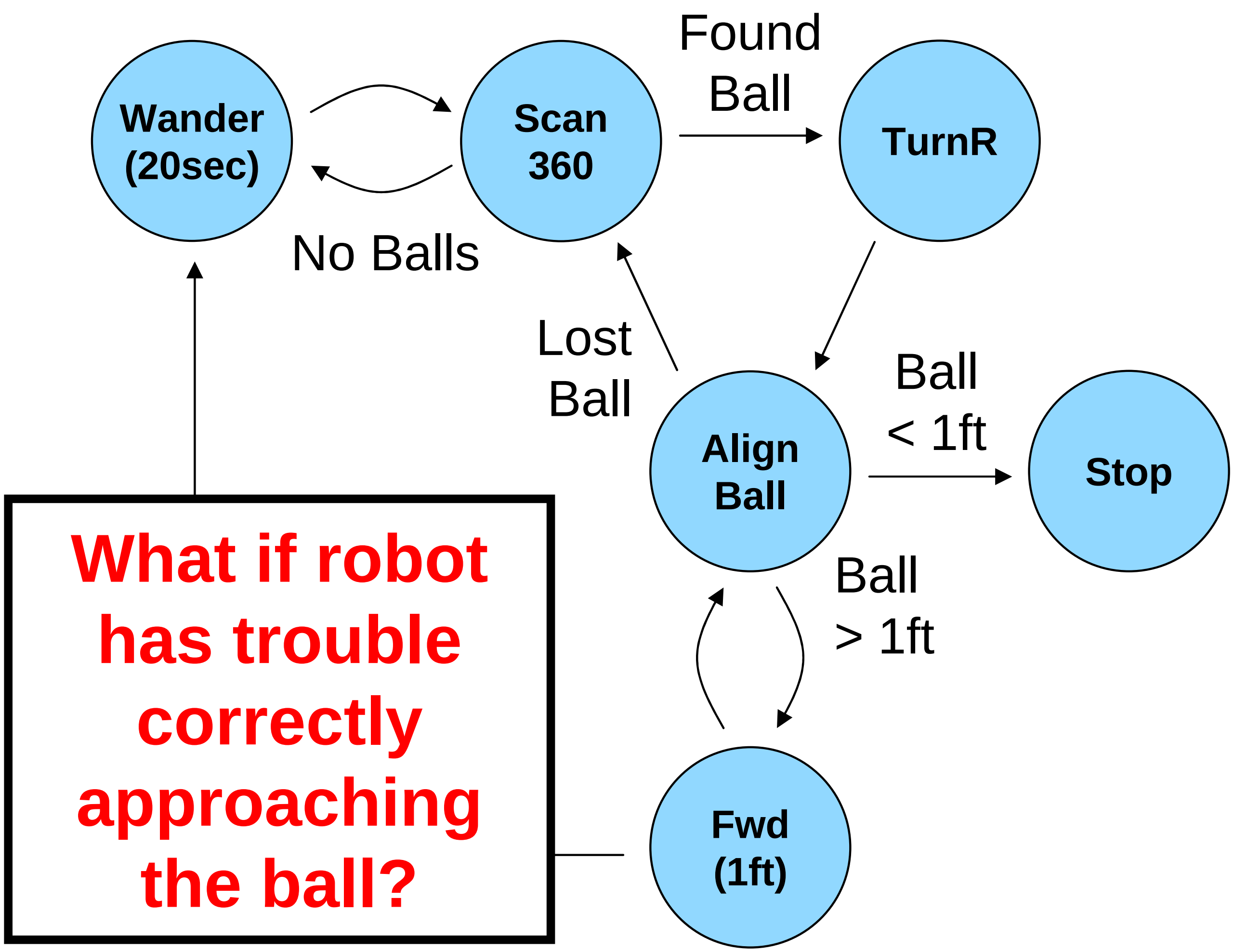
Simple finite state machine to locate red balls



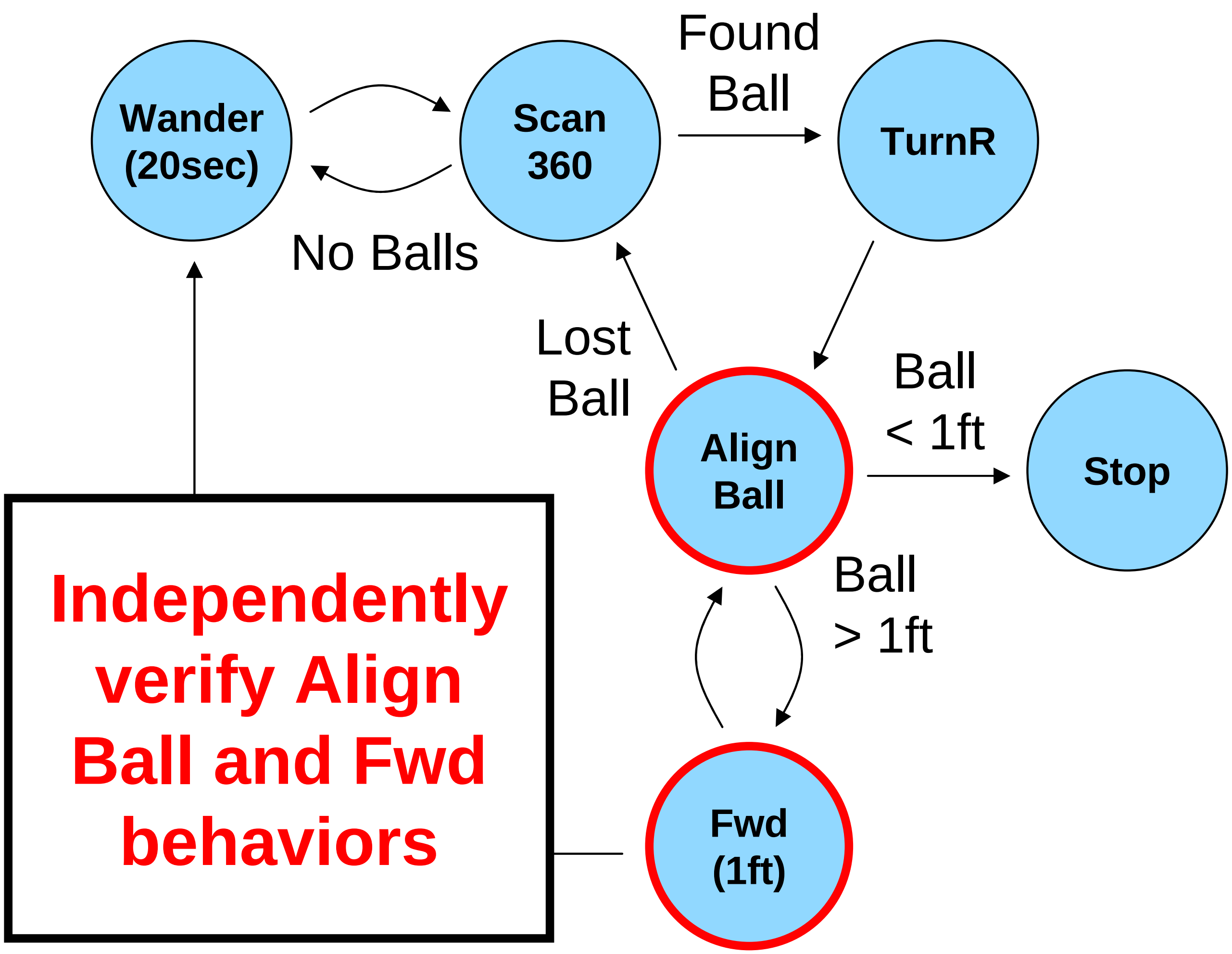
Simple finite state machine to locate red balls



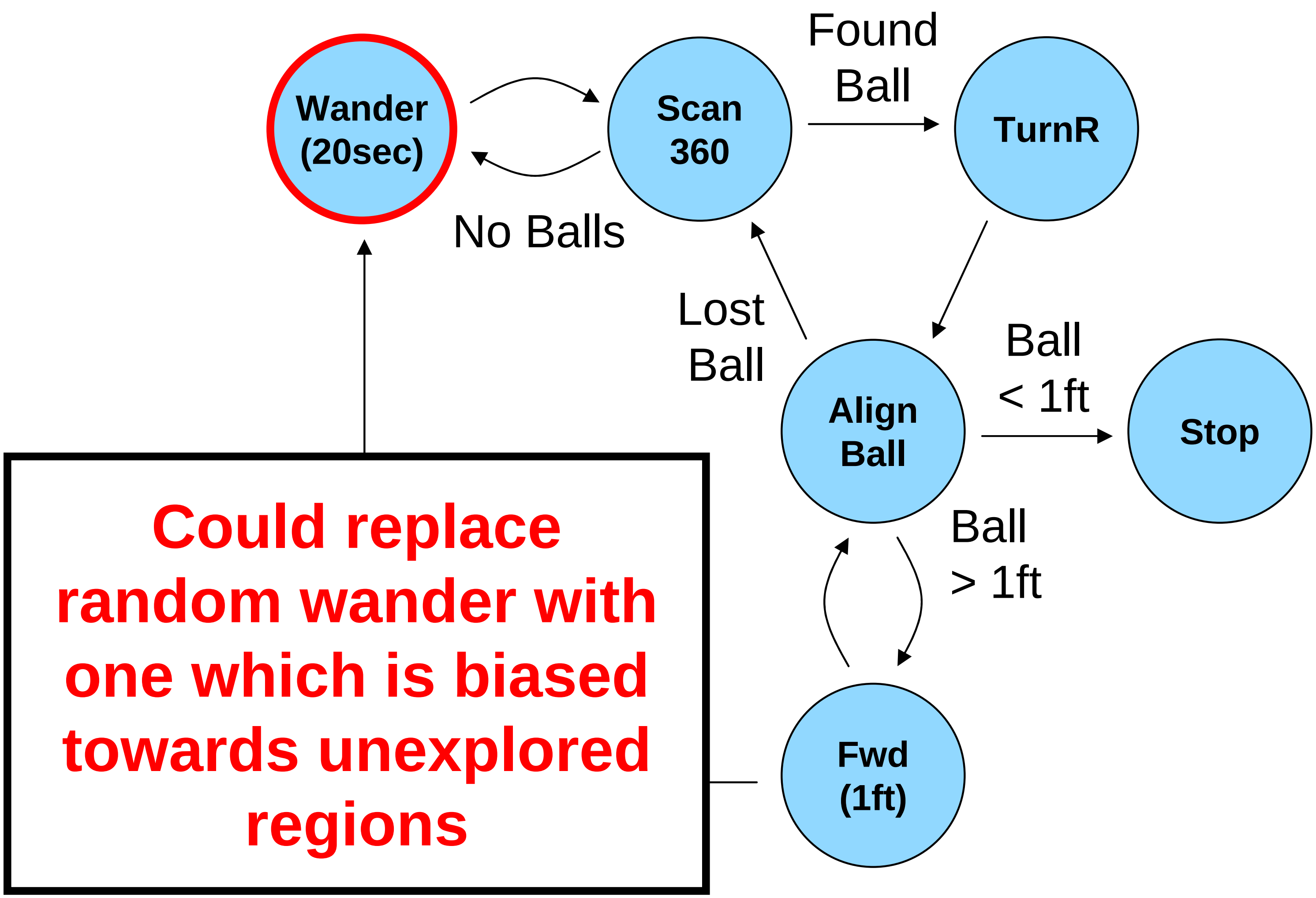
To debug a FSM control system verify behaviors and state transitions



To debug a FSM control system verify behaviors and state transitions



Improve FSM control system by replacing a state with a better implementation



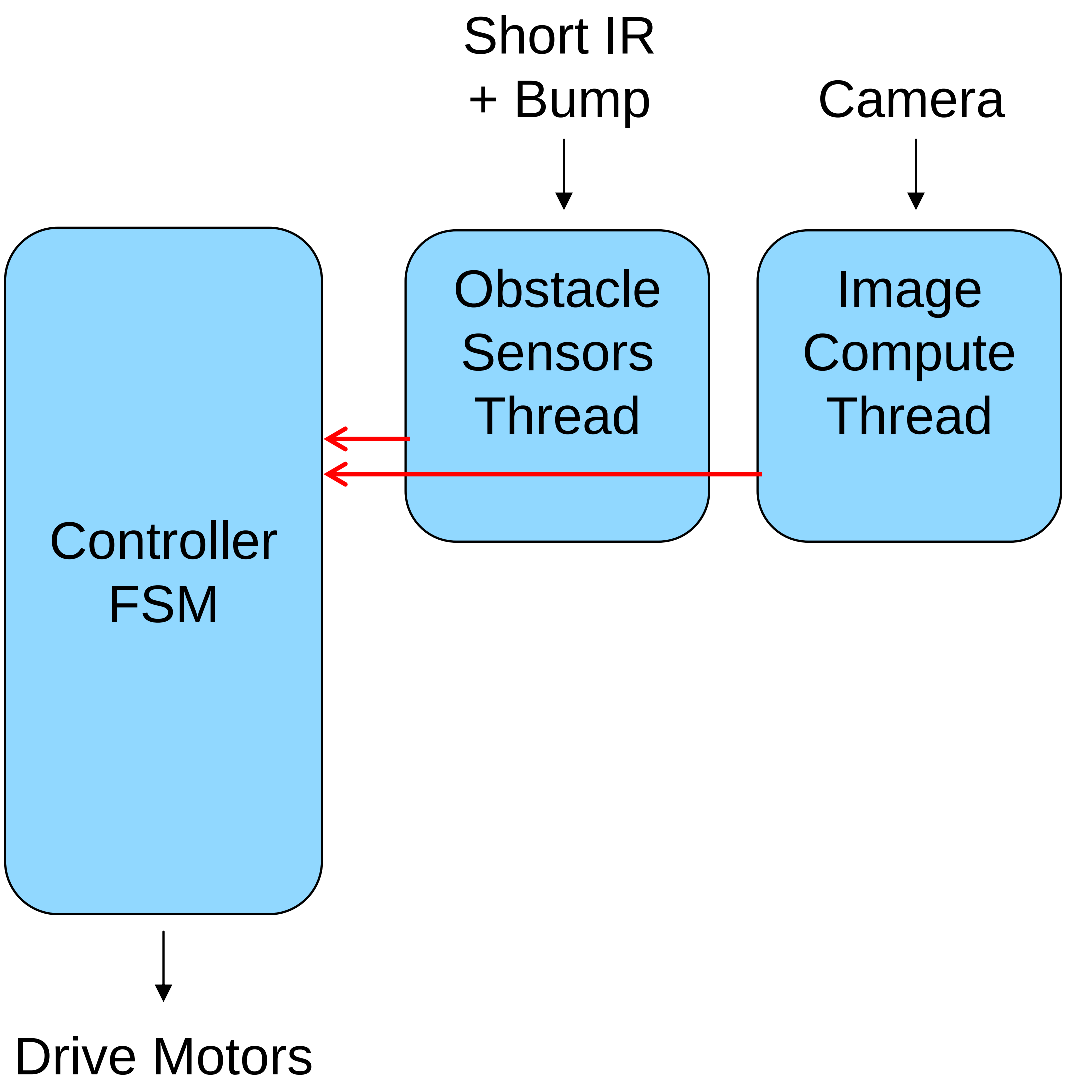
Improve FSM control system by replacing a state with a better implementation

What about integrating camera code into wander behavior so robot is always looking for red balls?

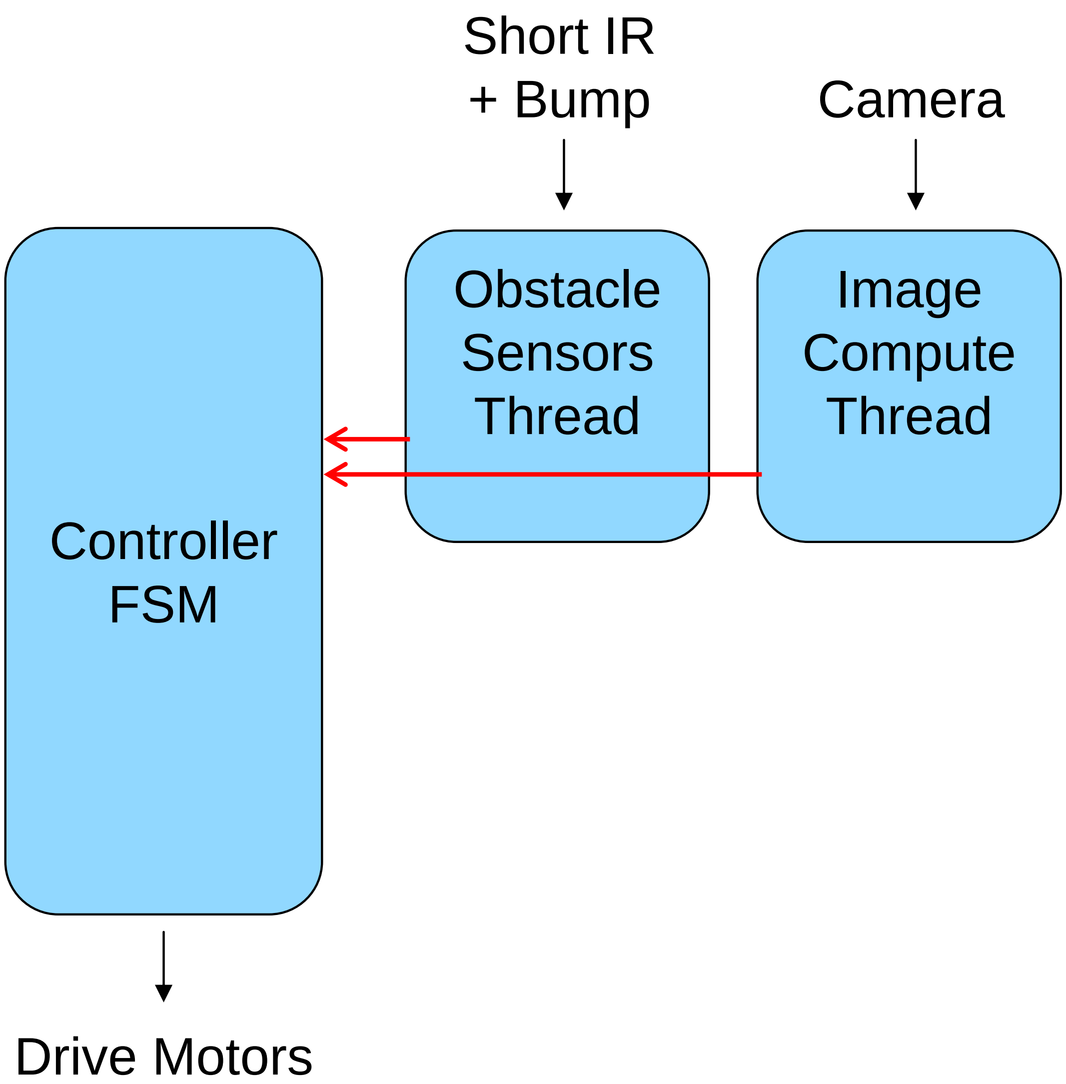
- Image processing is time consuming so might not check for obstacles until too late
- Not checking camera when rotating
- Wander behavior begins to become monolithic

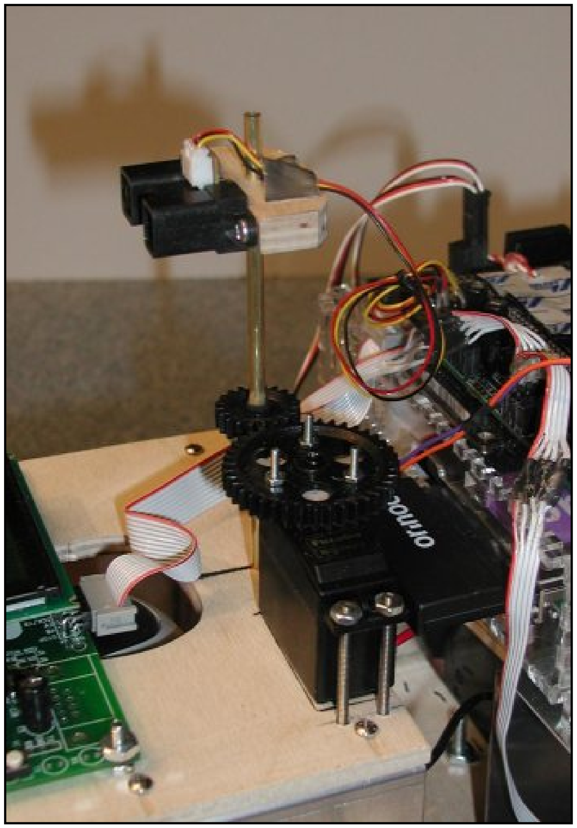
```
ball = false
turn both motors on
while ( !timeout and !ball )
  capture and process image
  if ( red ball ) ball = true

  read IR sensor
  if ( IR < thresh )
    stop motors
    rotate 90 degrees
    turn both motors on
  endif

endwhile
```

Multi-threaded finite state machine control systems

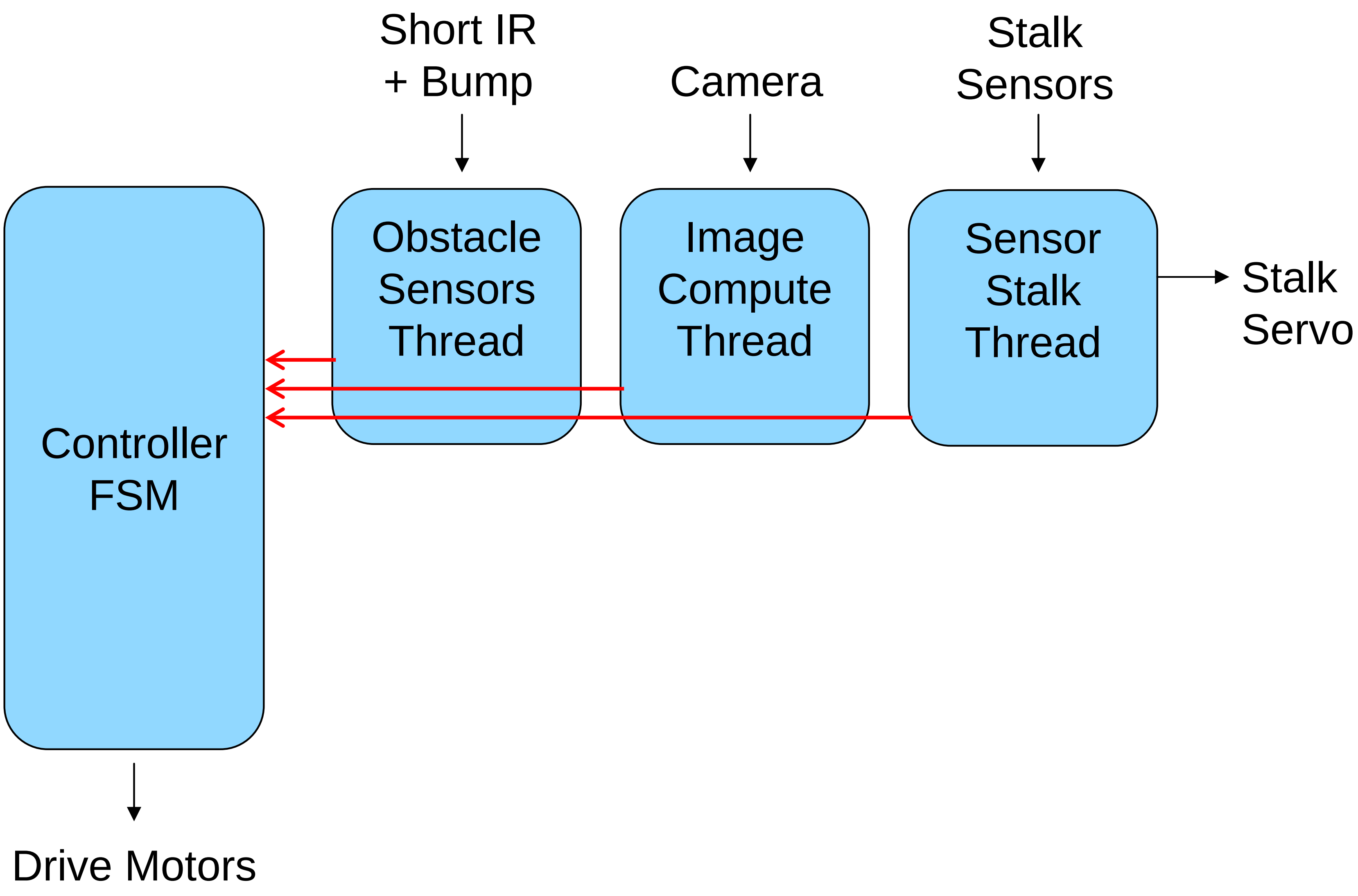


Multi-threaded finite state machine control systems

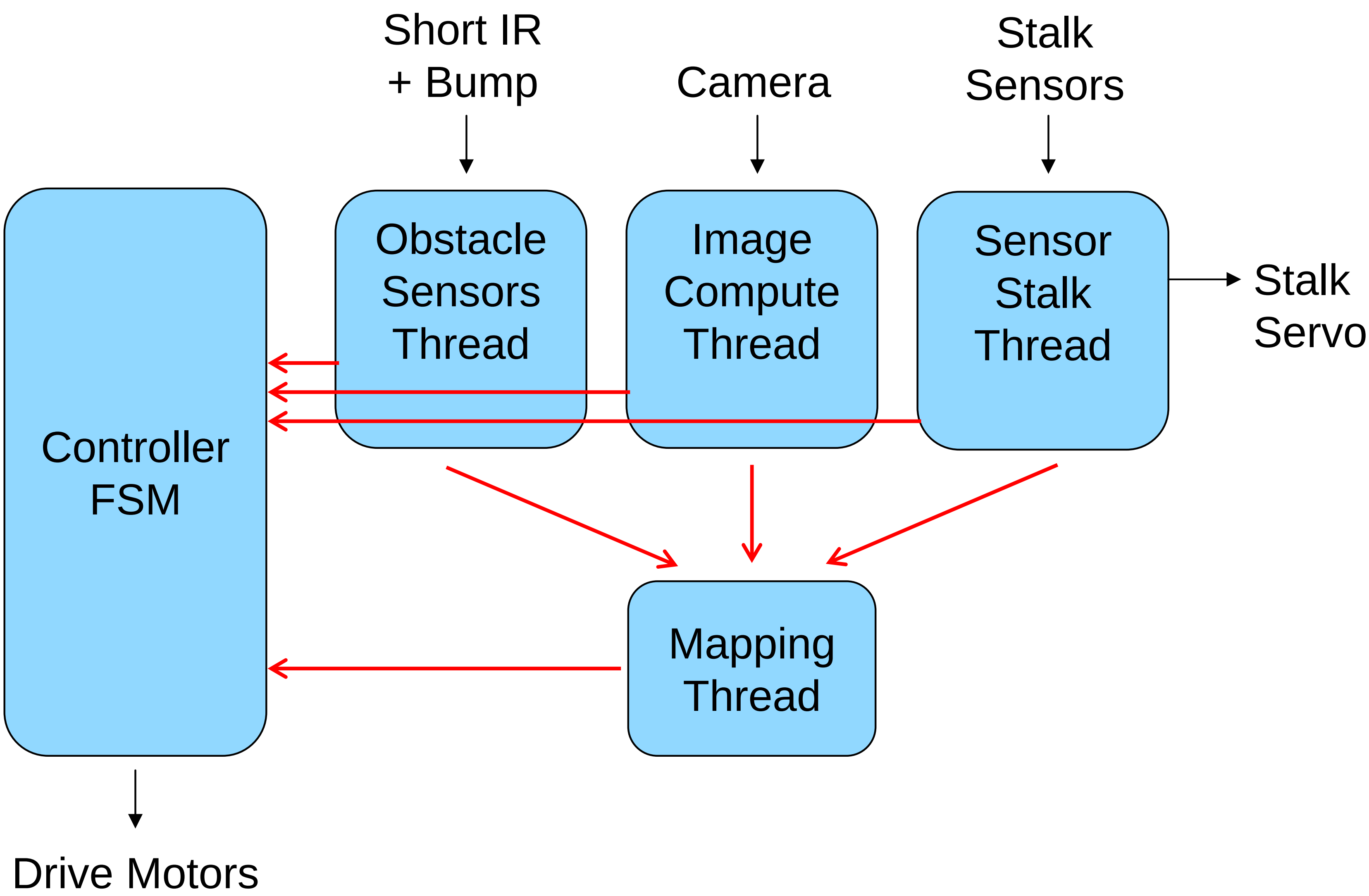




Multi-threaded finite state machine control systems



Multi-threaded finite state machine control systems



FSMs in Maslab

Finite state machines can combine the model-plan-act and emergent approaches and are a good starting point for your Maslab robotic control system

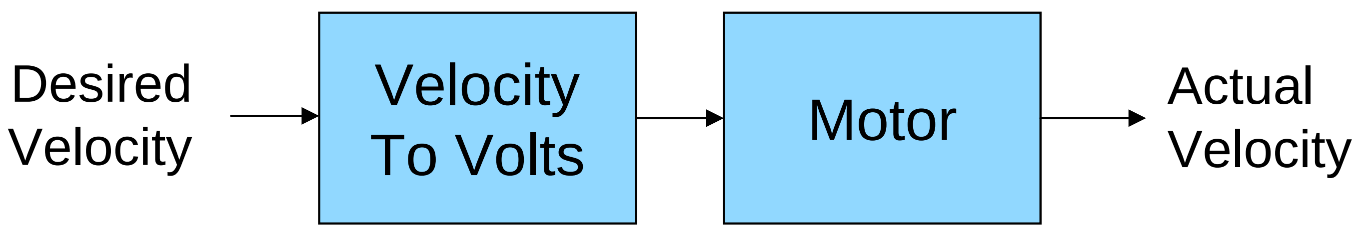
Outline

- High-level control system paradigms
  - Model-Plan-Act Approach
  - Behavioral Approach
  - Finite State Machine Approach
- **Low-level control loops**
  - **PID controller for motor velocity**
  - **PID controller for robot drive system**
- Examples from past years

Problem: How do we set a motor to a given velocity?

Open Loop Controller

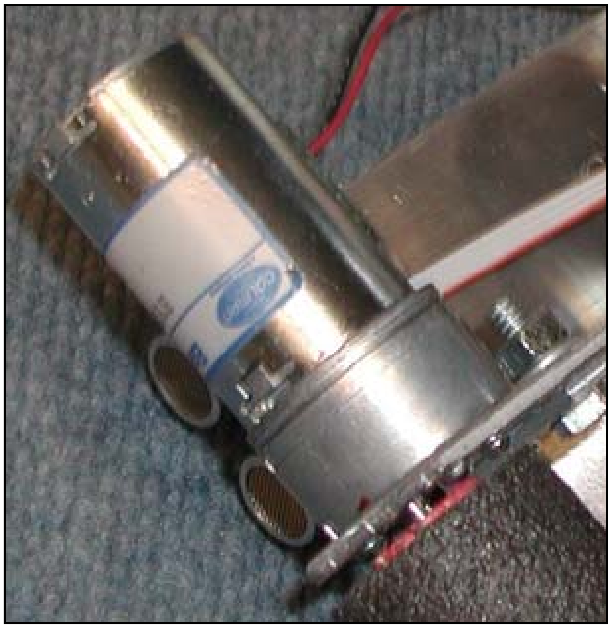
- Use trial and error to create some kind of relationship between velocity and voltage
- Changing supply voltage or drive surface could result in incorrect velocity

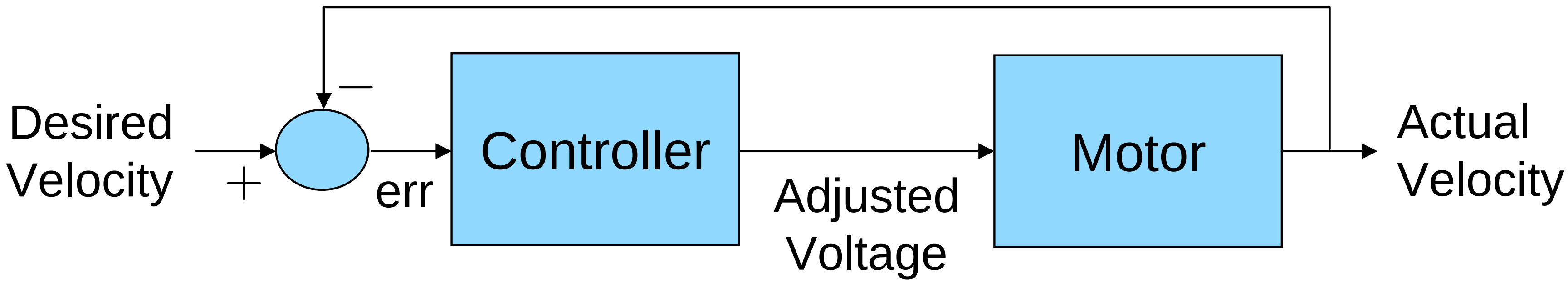


Problem: How do we set a motor to a given velocity?

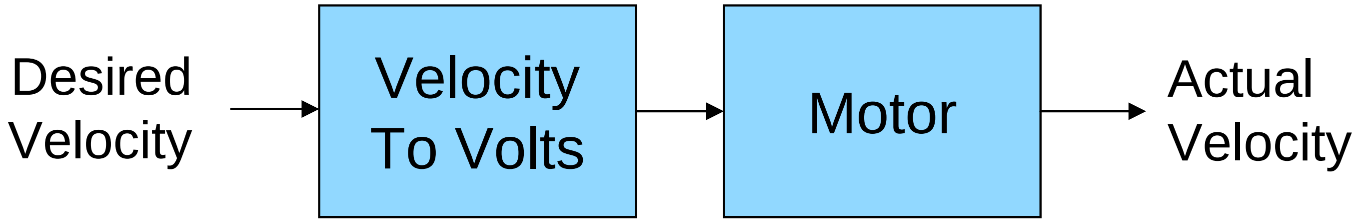
Closed Loop Controller

- Feedback is used to adjust the voltage sent to the motor so that the actual velocity equals the desired velocity
- Can use an optical encoder to measure actual velocity



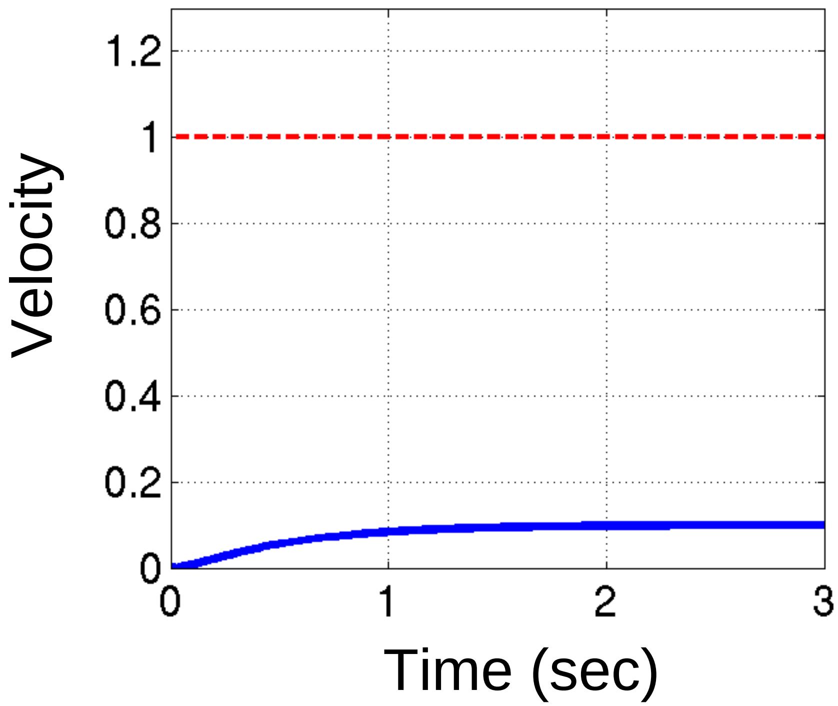


Step response with no controller

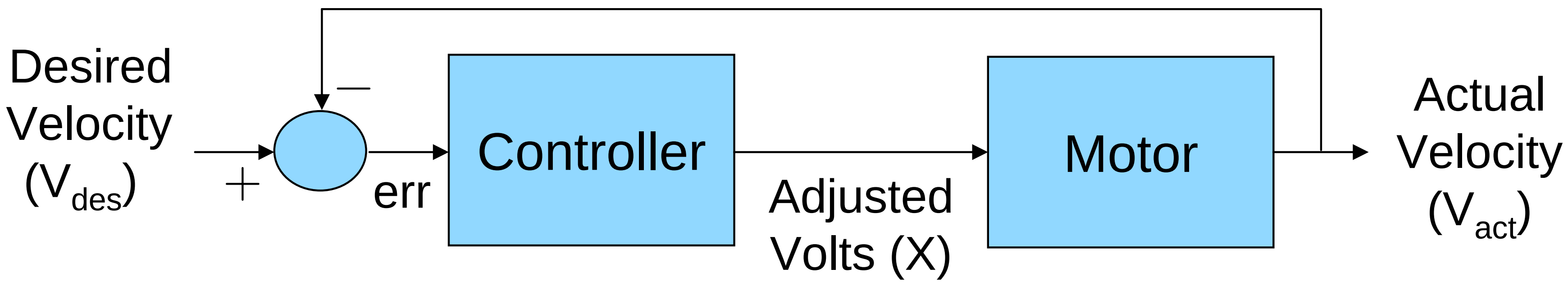


- Naive velocity to volts
- Model motor with several differential equations

- Slow rise time
- Stead-state offset

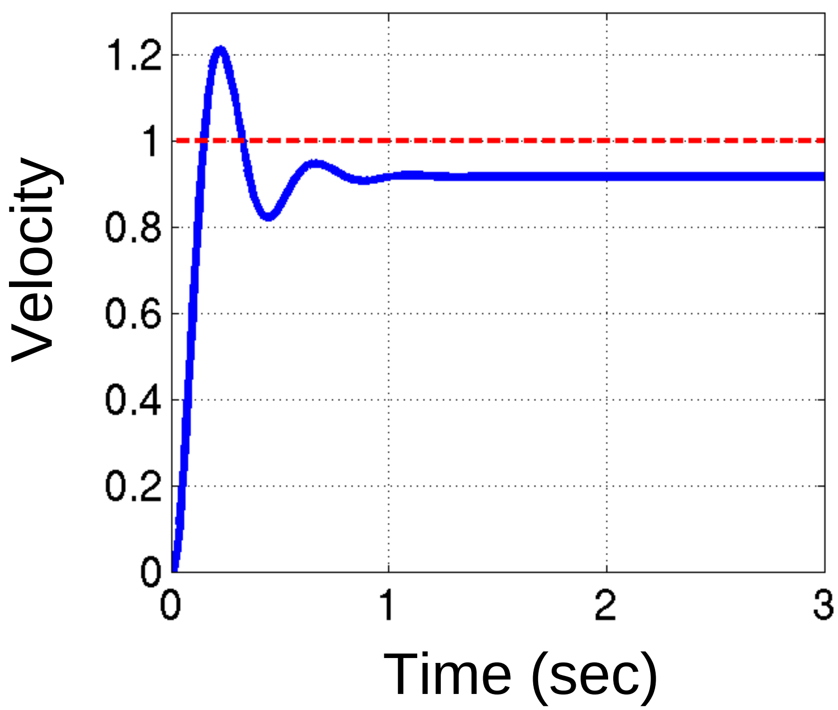


Step response with proportional controller

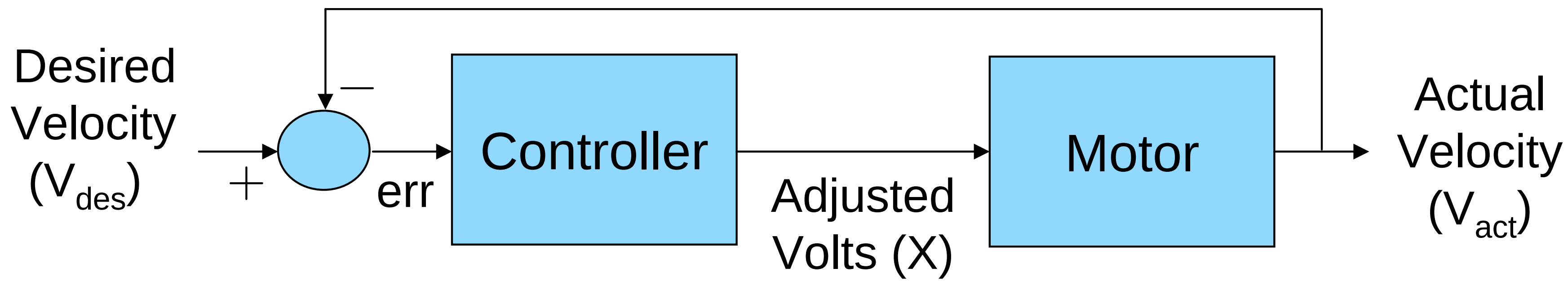


$$X = V_{des} + K_P \cdot (V_{des} - V_{act})$$

- Big error big = big adj
- Faster rise time
- Overshoot
- Stead-state offset (there is still an error but it is not changing!)

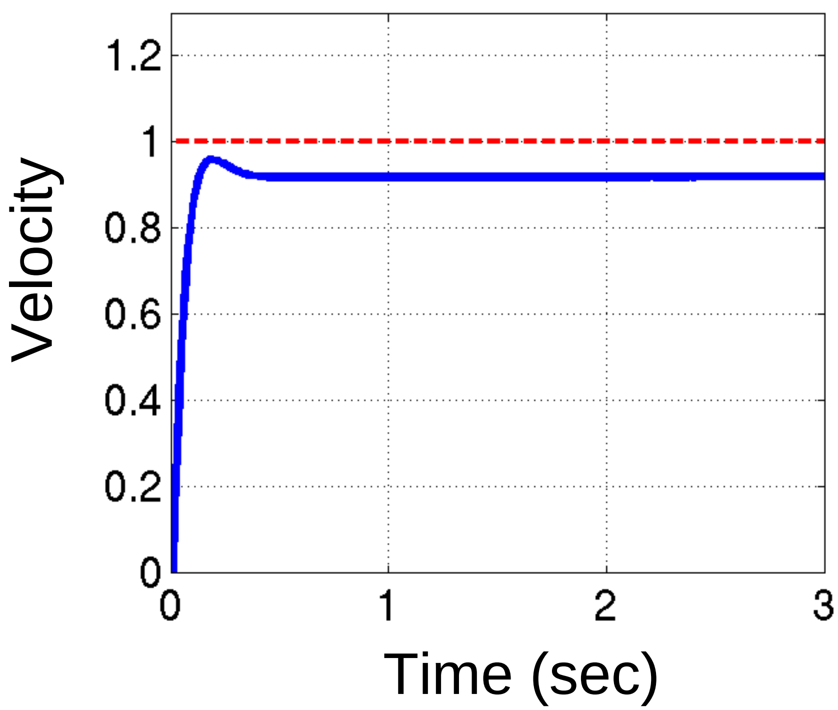


Step response with proportional-derivative controller

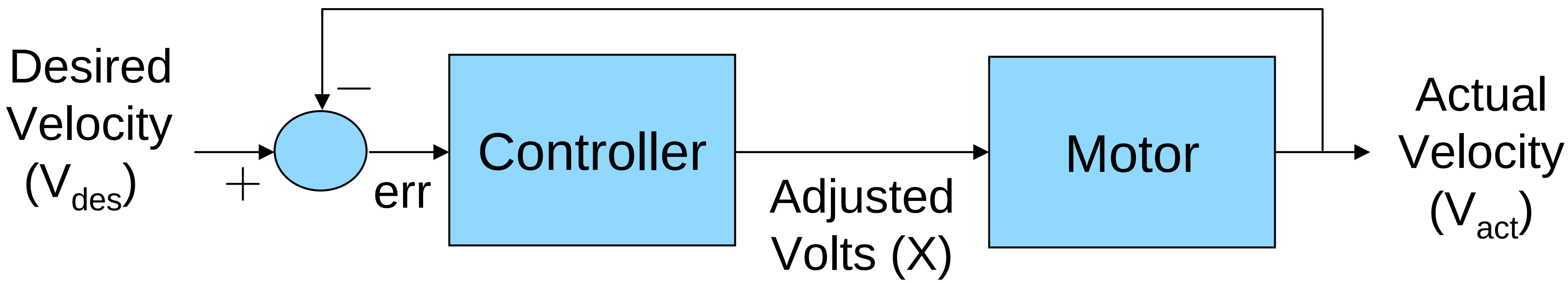


$$X = V_{des} + K_P e(t) - K_D \frac{de(t)}{dt}$$

- When approaching desired velocity quickly, de/dt term counteracts proportional term slowing adjustment
- Faster rise time
- Reduces overshoot

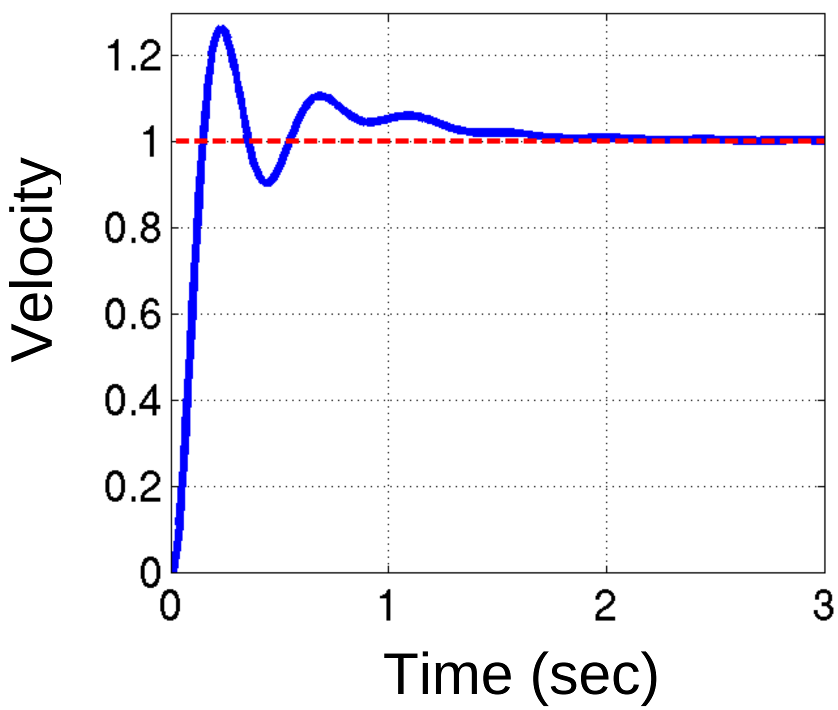


Step response with proportional-integral controller

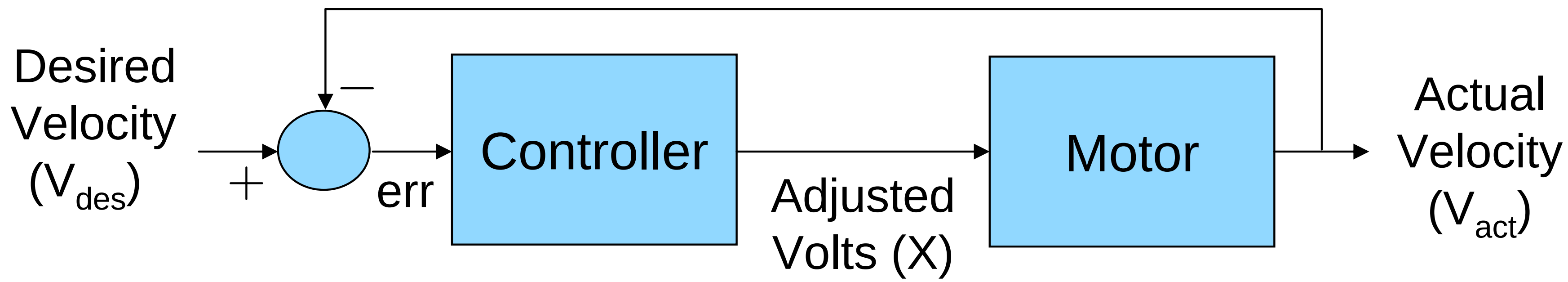


$$X = V_{des} + K_P e(t) - K_I \int e(t)\, dt$$

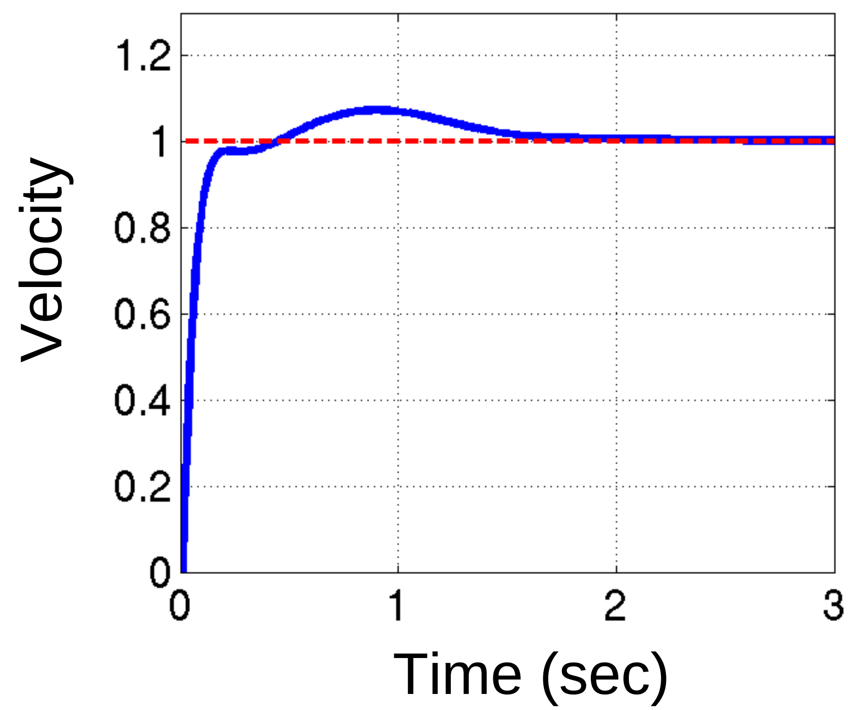
- Integral term eliminates accumulated error
- Increases overshoot



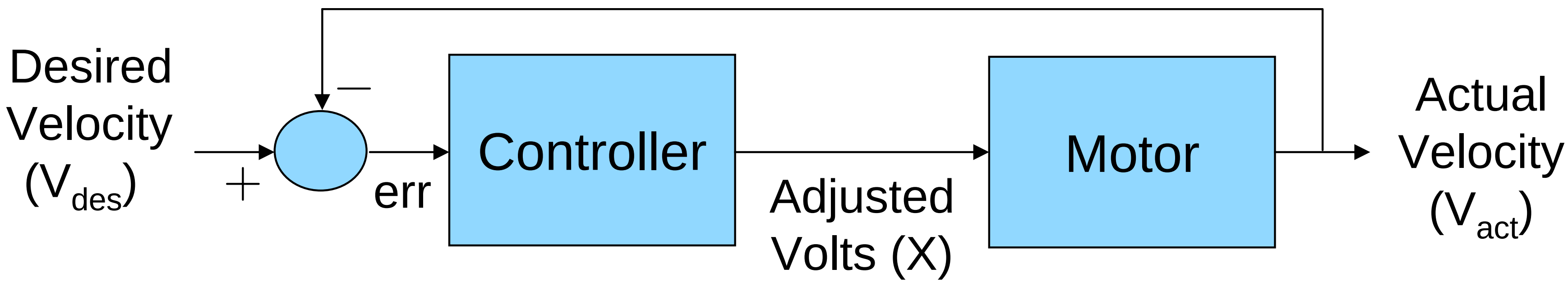
Step response with PID controller



$$X = V_{des} + K_P e(t) + K_I \int e(t)\, dt - K_D \frac{de(t)}{dt}$$

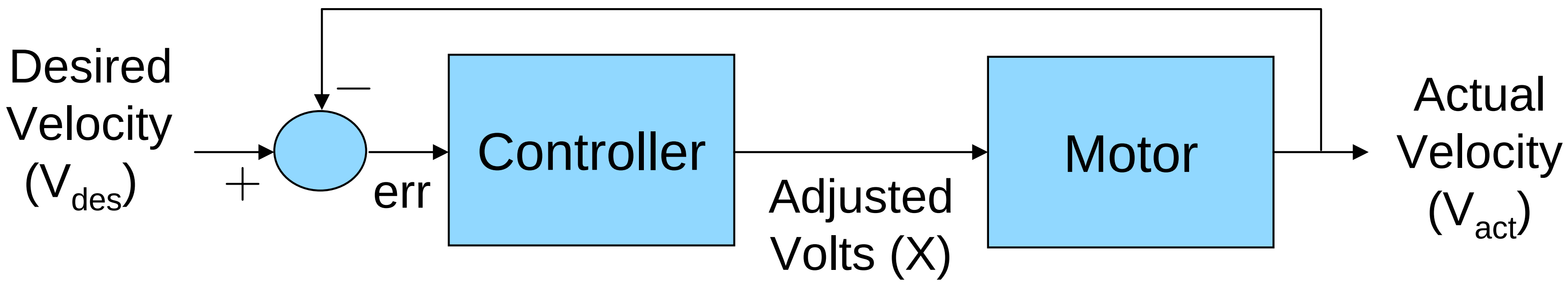


Choosing and tuning a controller



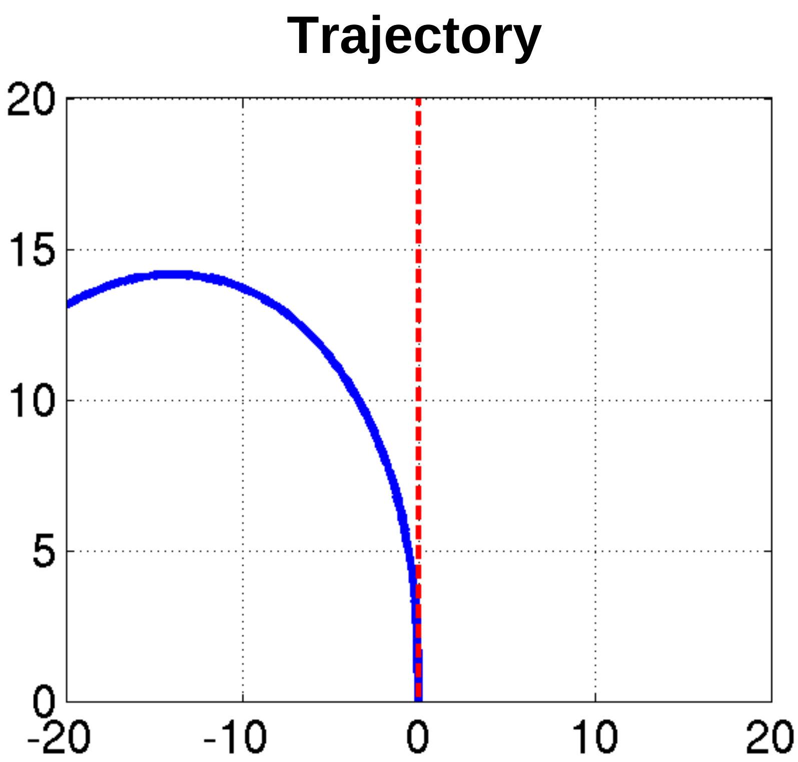
	Rise Time	Overshoot	SS Error
Proportional	Decrease	Increase	Decrease
Integral	Decrease	Increase	Eliminate
Derivative	~	Decrease	~

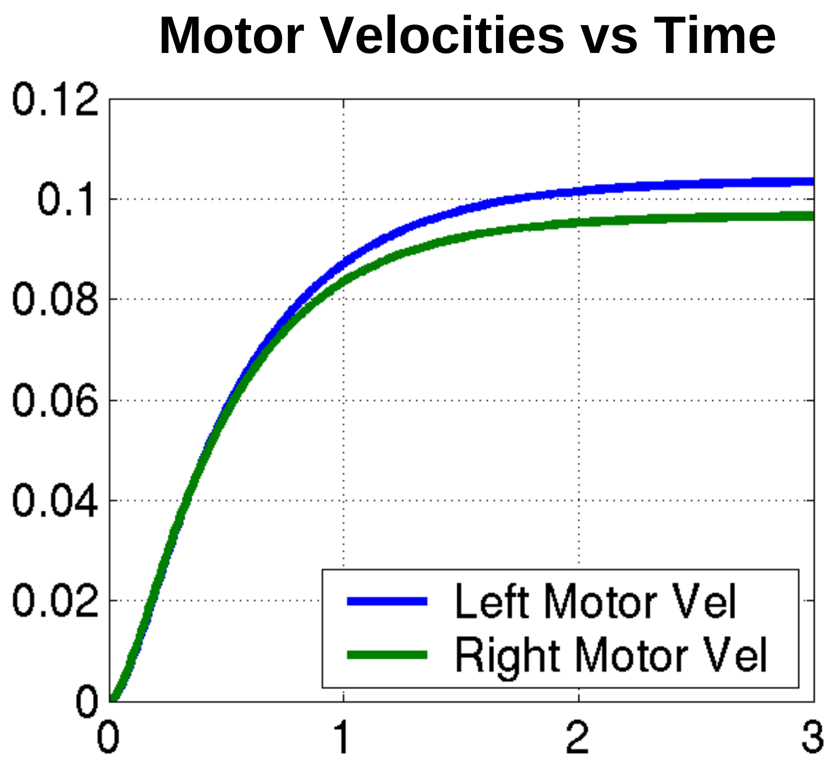
Choosing and tuning a controller



- Use the simplest controller which achieves the desired result
- Tuning PID constants is very tricky, especially for integral constants
- Consult the literature for more controller tips and techniques

Problem: How do we make our robots go in a nice straight line?

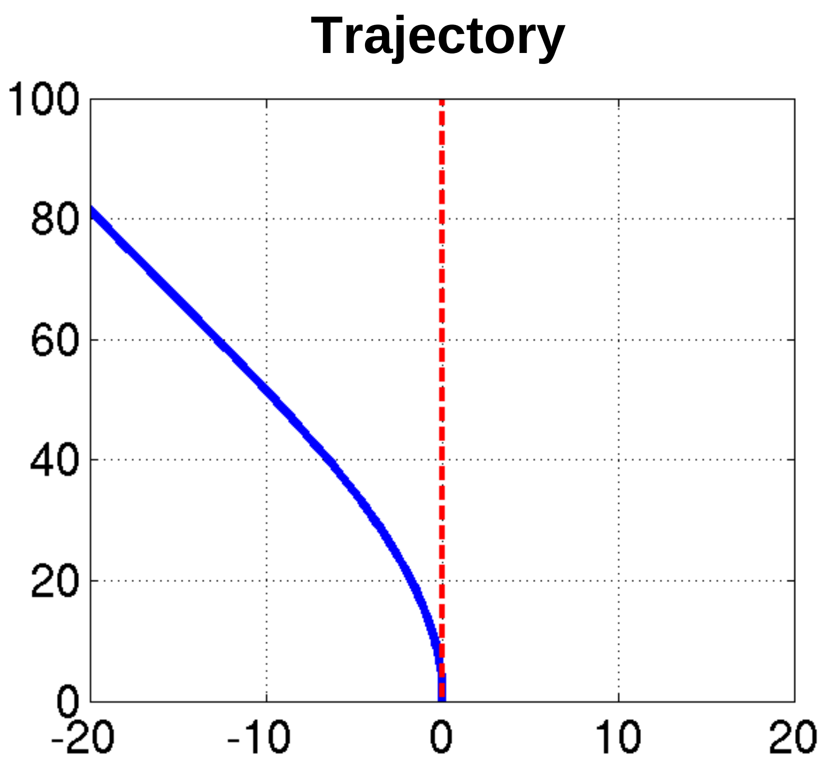


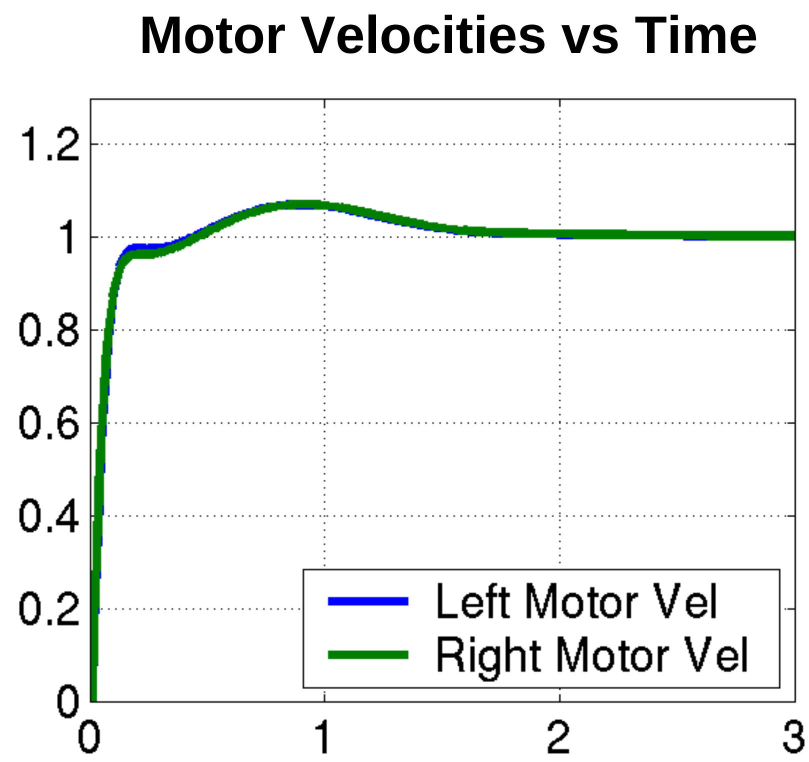


Model differential drive with slight motor mismatch

With an open loop controller, setting motors to same velocity results in a less than straight trajectory

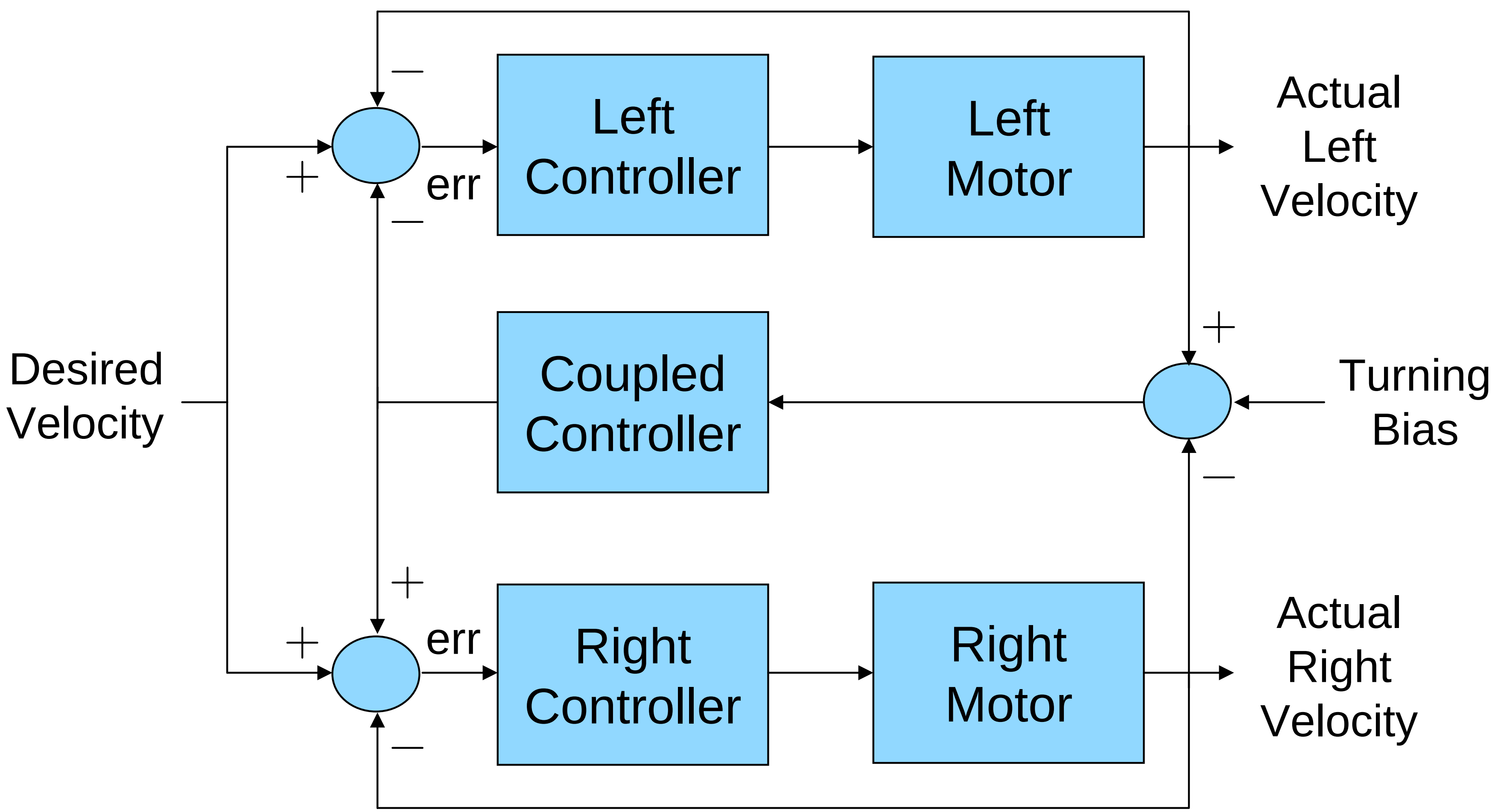
Problem: How do we make our robots go in a nice straight line?





With an independent PID controller for each motor, setting motors to same velocity results in a straight trajectory but not necessarily straight ahead!

We can synchronize the motors with a third PID controller



Inspired from “Mobile Robots”, Jones, Flynn, and Seiger, 1999

We can synchronize the motors with a third PID controller

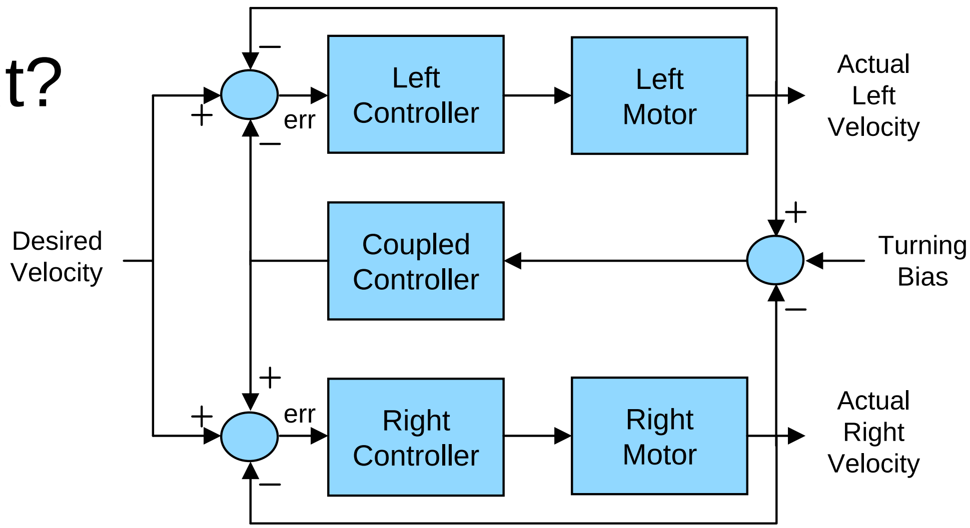
What should the coupled controller use as its error input?

Velocity Differential

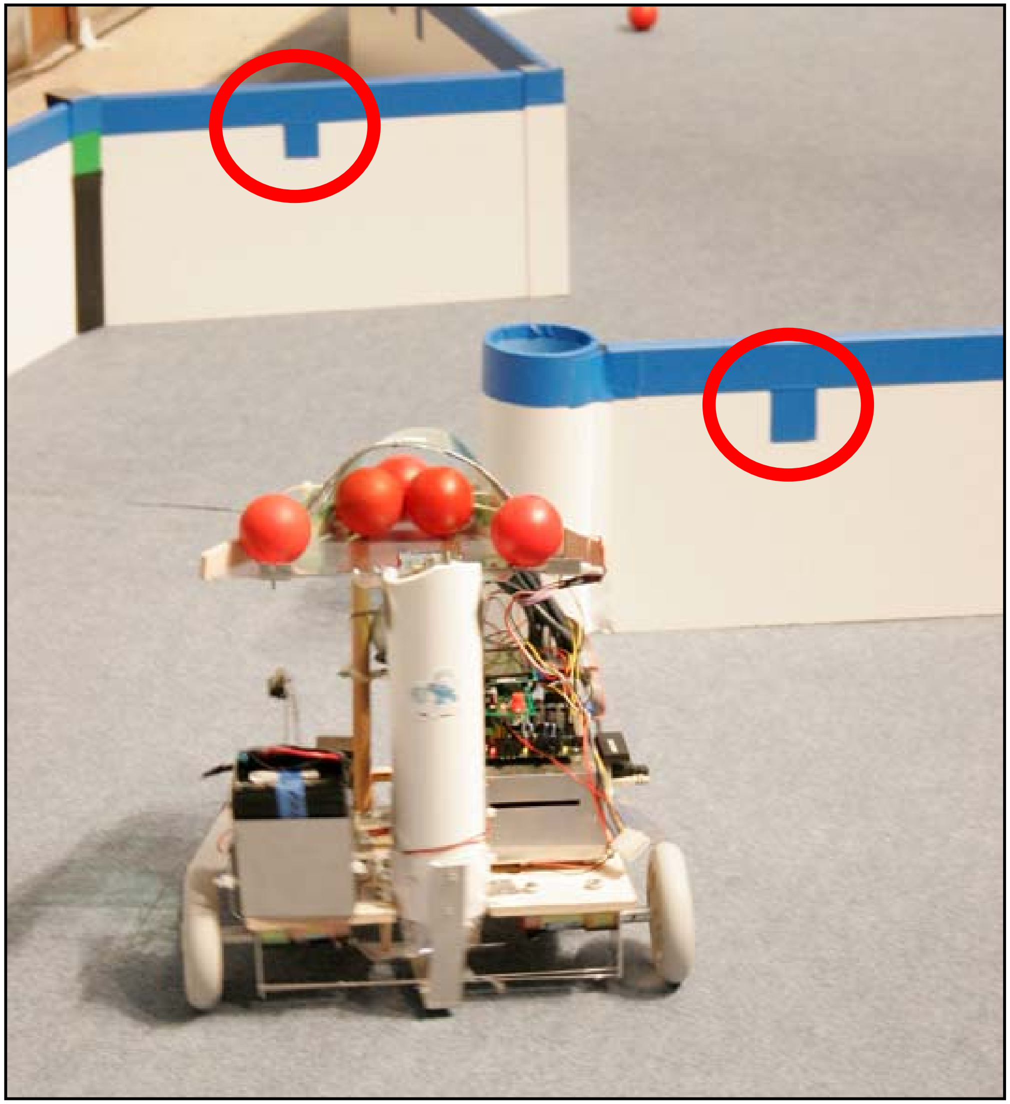
- Will simply help the robot go straight but not necessarily straight ahead

Cumulative Centerline Offset

- Calculate by integrating motor velocities and assuming differential steering model for the robot
- Will help the robot go straight ahead

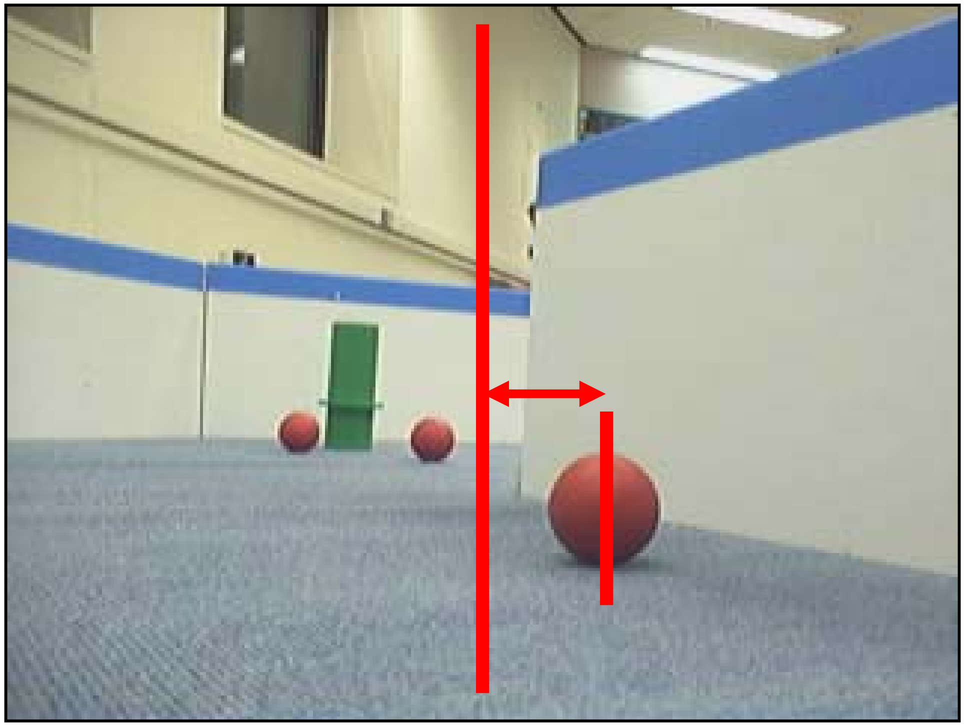


The digital camera is a powerful sensor for estimating error in our control loops



- Track wall ticks to see how they move through the image
- Use analytical model of projection to determine an error between where they are and where they should be if robot is going straight
- Push error through PID controller

The digital camera is a powerful sensor for estimating error in our control loops



- Track how far ball center is from center of image
- Use analytical model of projection to determine an orientation error
- Push error through PID controller

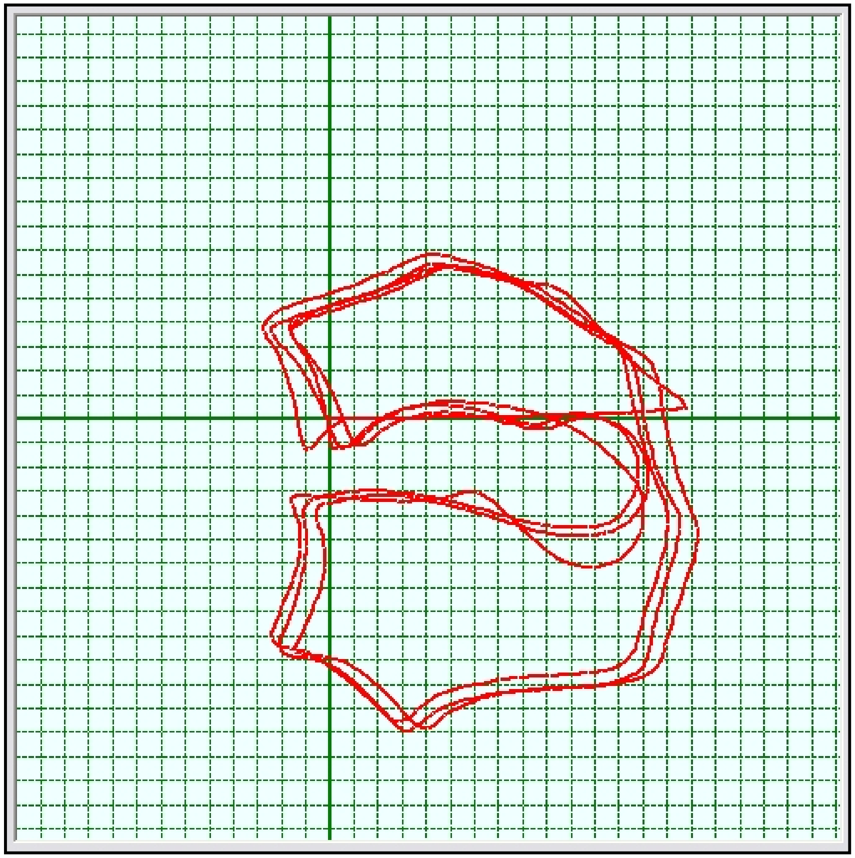
What if we just used a simple proportional controller? Could lead to steady-state error if motors are not perfectly matched!

Outline

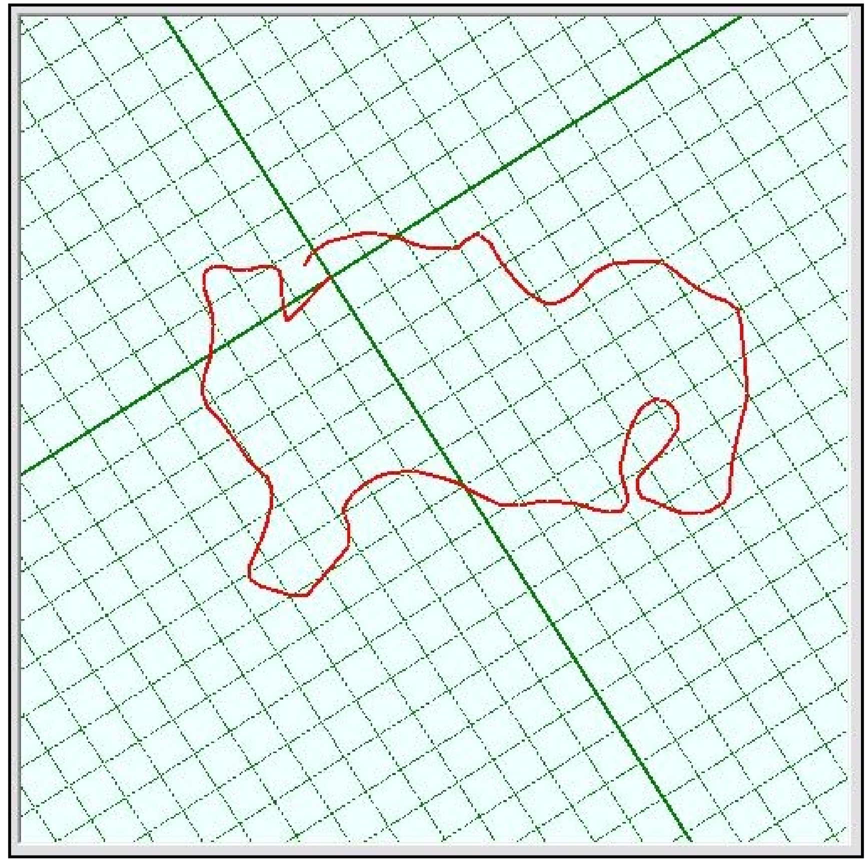


- High-level control system paradigms
  - Model-Plan-Act Approach
  - Behavioral Approach
  - Finite State Machine Approach
- Low-level control loops
  - PID controller for motor velocity
  - PID controller for robot drive system
- **Examples from past years**

Team 15 in 2005 used a map-plan-act approach (well at least in spirit)

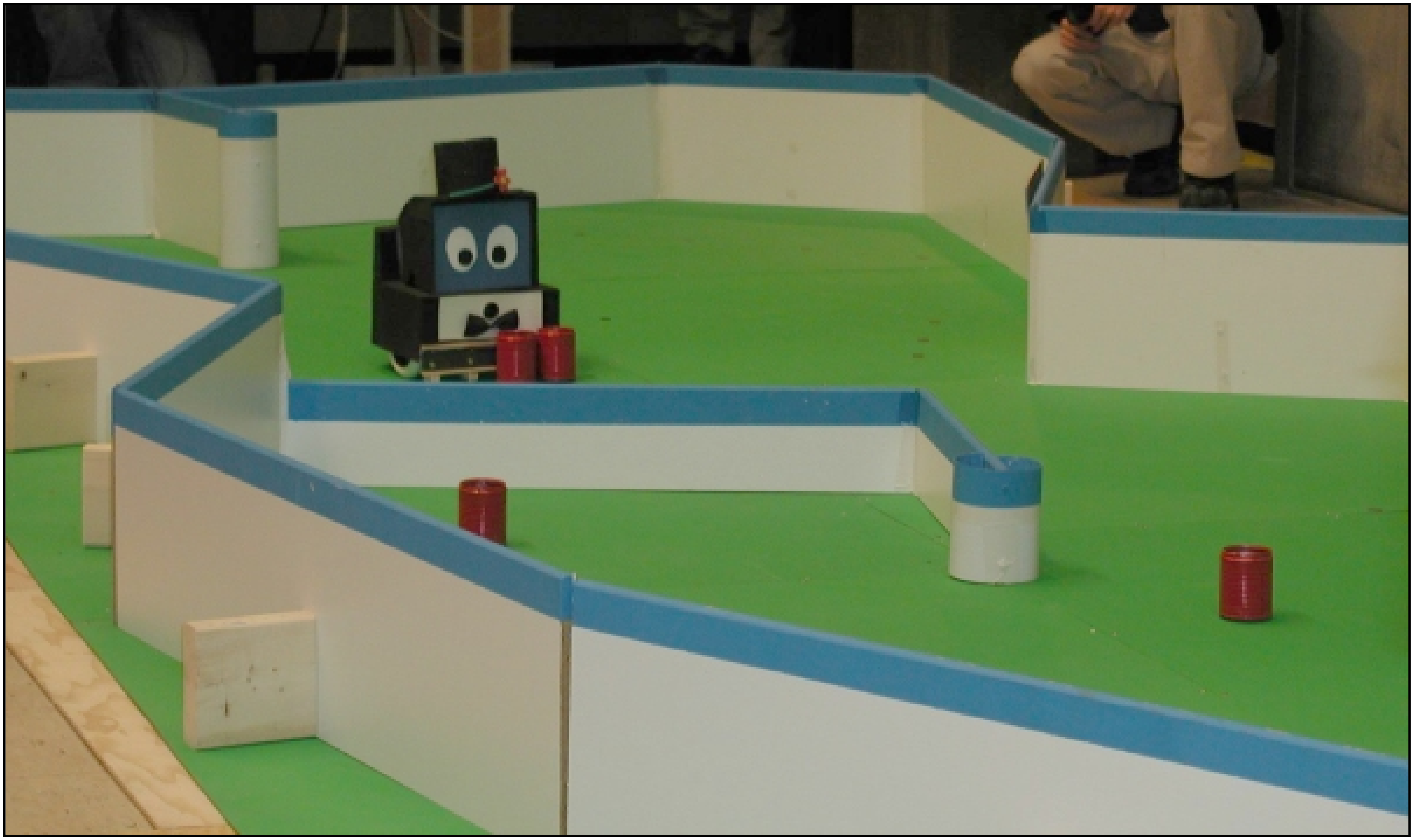


Multiple runs around
a mini-playing field

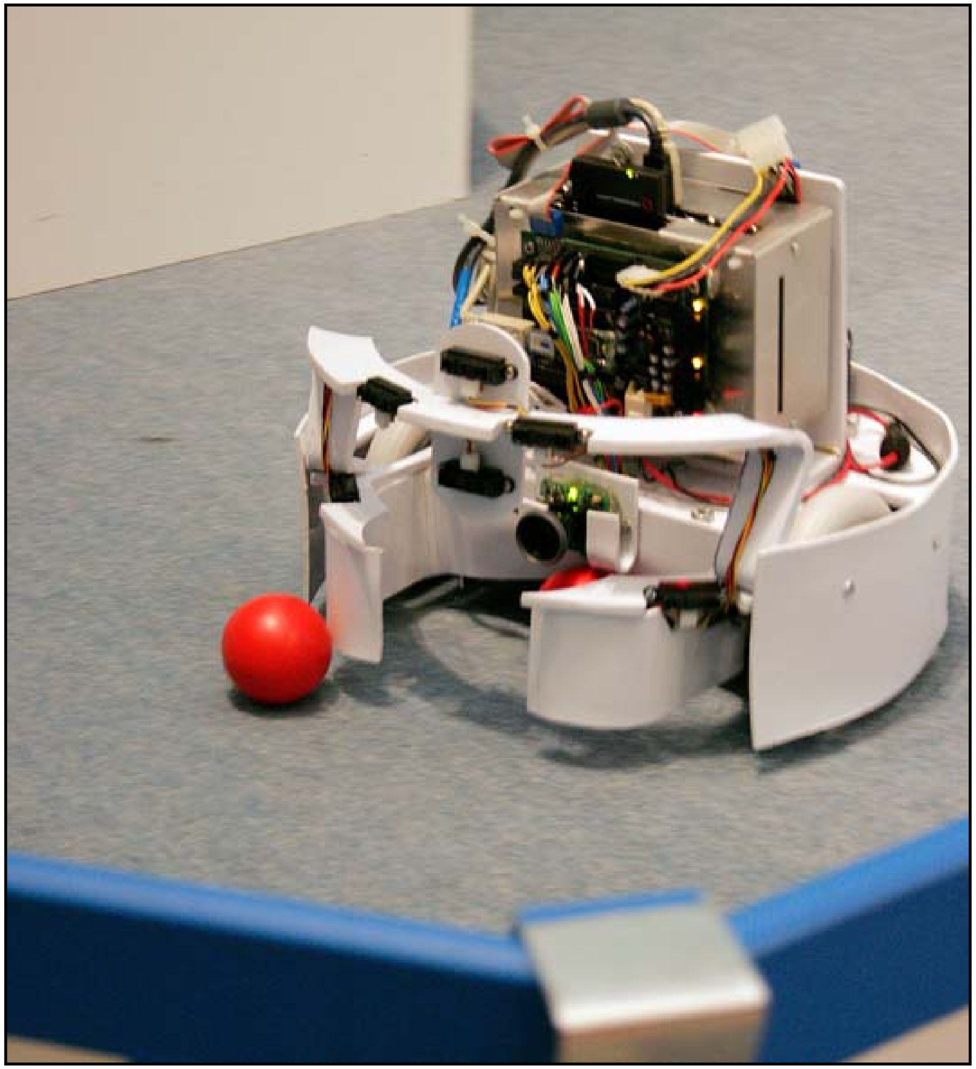


Odometry data from
exploration round of contest

Team 10 in 2003 used odometry so Bob could retrace his steps and return home



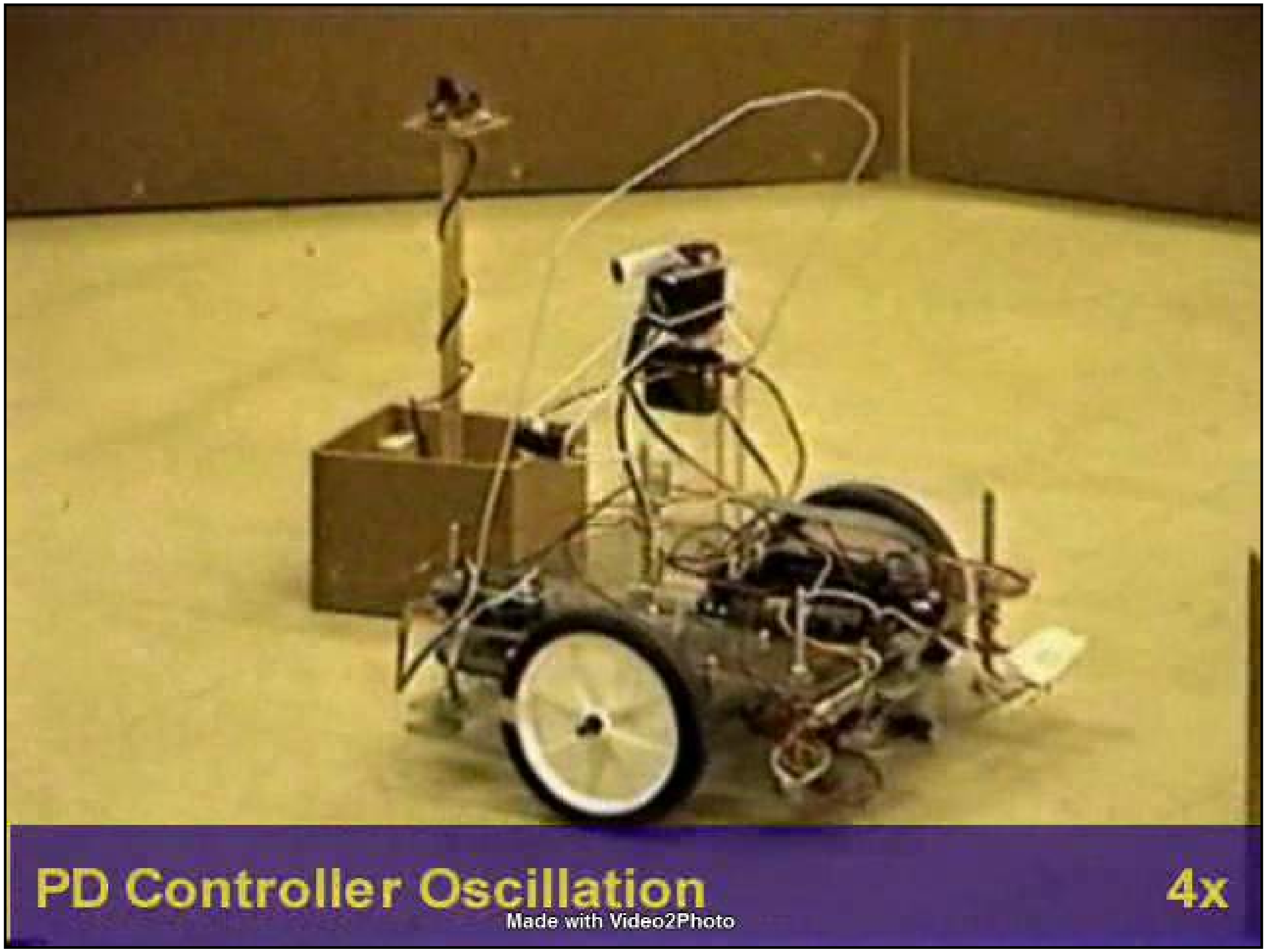
Team 4 in 2005 used an emergent approach with four layered behaviors



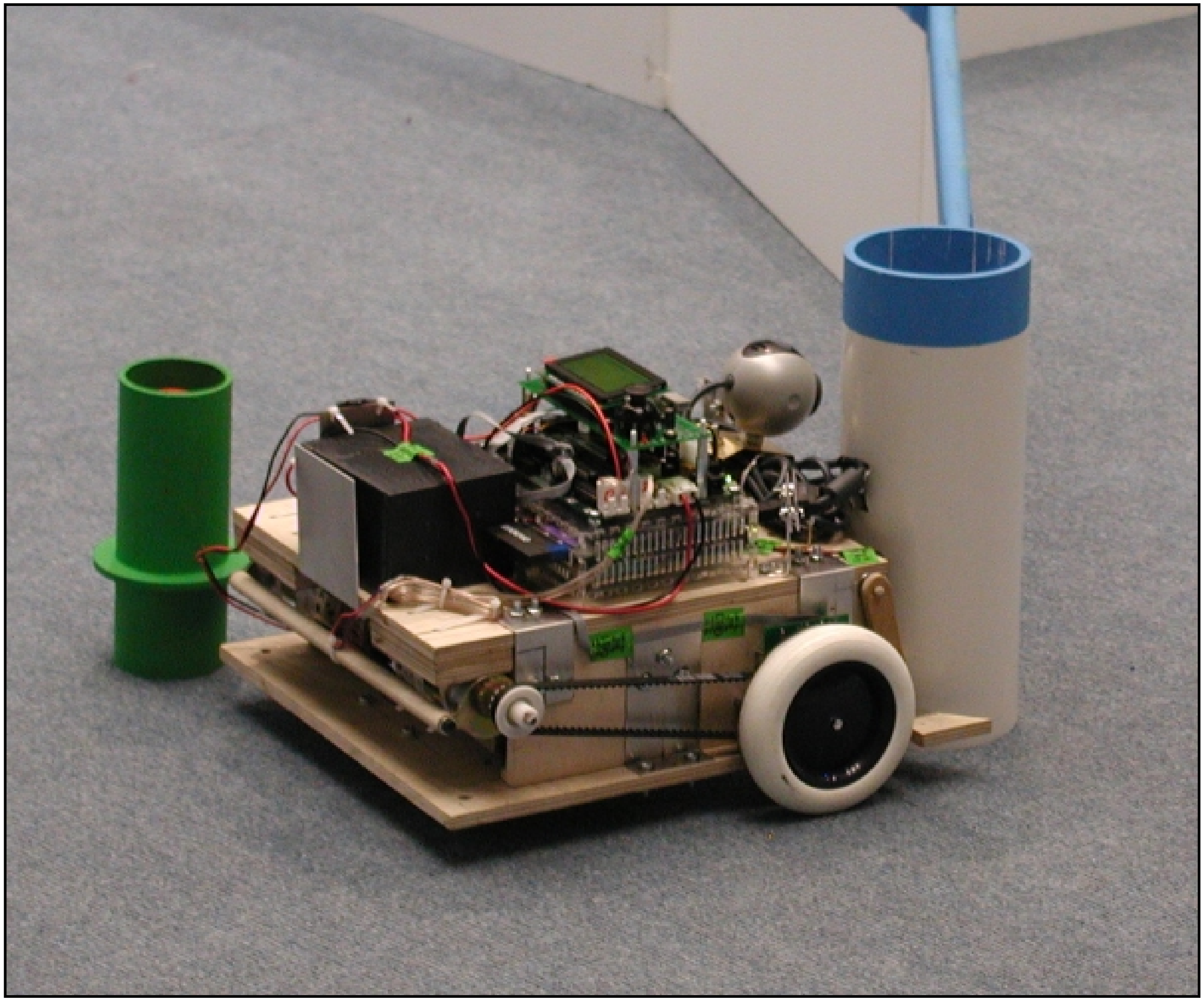
- **Boredom:** If image doesn’t change then move randomly
- **ScoreGoals:** If image contains a goal the drive straight for it
- **ChaseBalls:** If image contains a ball then drive towards ball
- **Wander:** Turn away from walls or move to large open areas

Team 16 from 2004 used their gyro and a closed loop controller to turn exactly 180º

Poorly tuned PID controllers can cause your robot to oscillate "randomly"



Team 12 in 2004 learned the hard way how important testing is



Take Away Points

- Integrating **feedback** into your control system "closes the loop" and is essential for creating robust robots
- Simple **finite state machines** make a solid starting point for your Maslab control systems
- Spend time this week **designing behaviors** and deciding how you will **integrate** these behaviors to create your control system