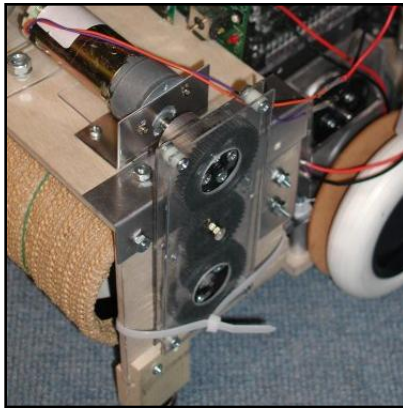


Control for Mobile Robots

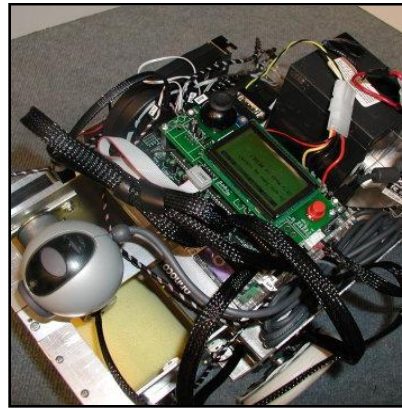
Christopher Batten

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

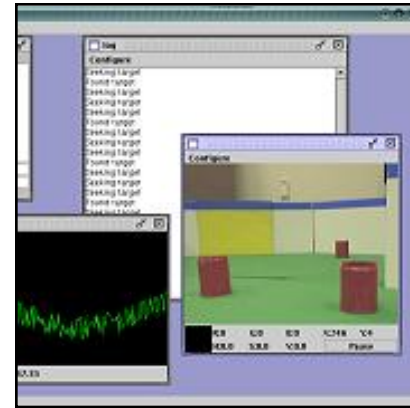
Mobile robots are very complex and involve many interacting components



Mechanical



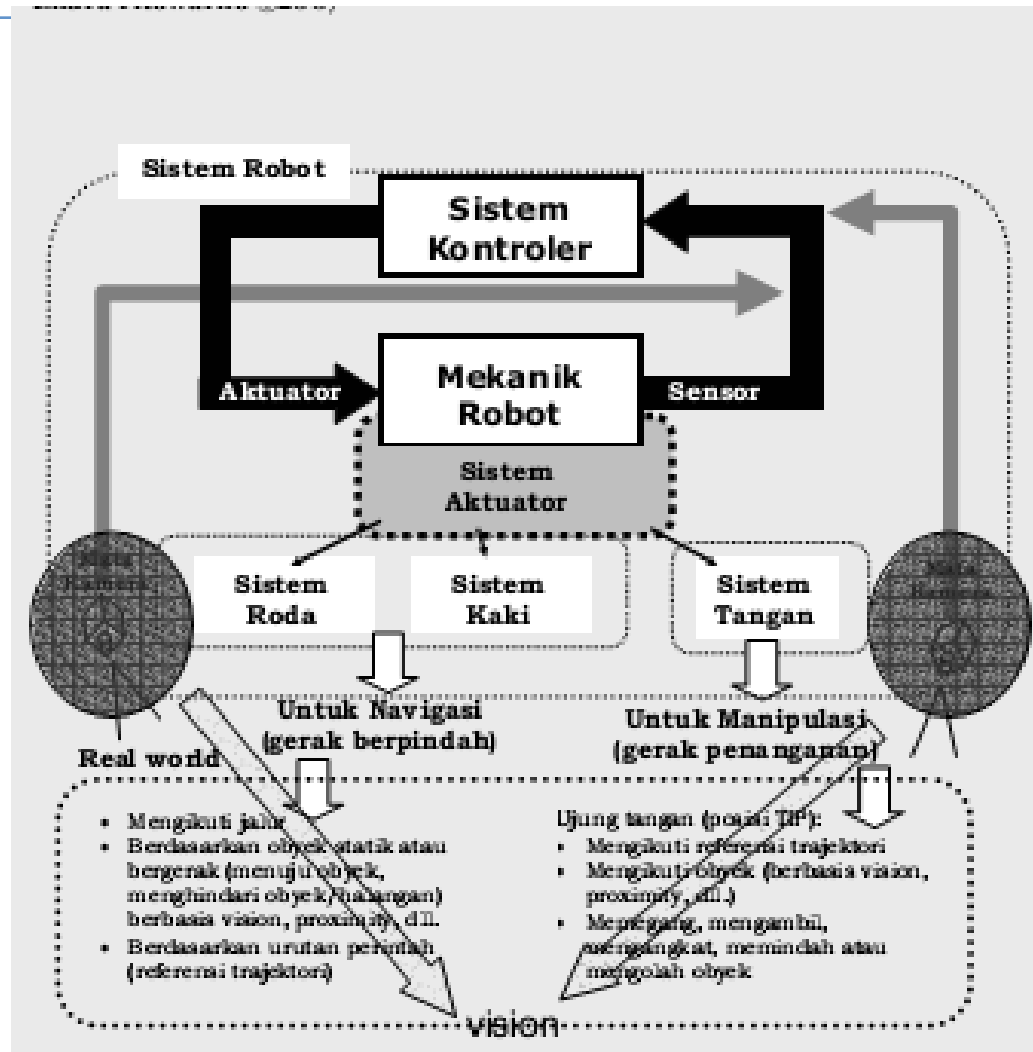
Electrical



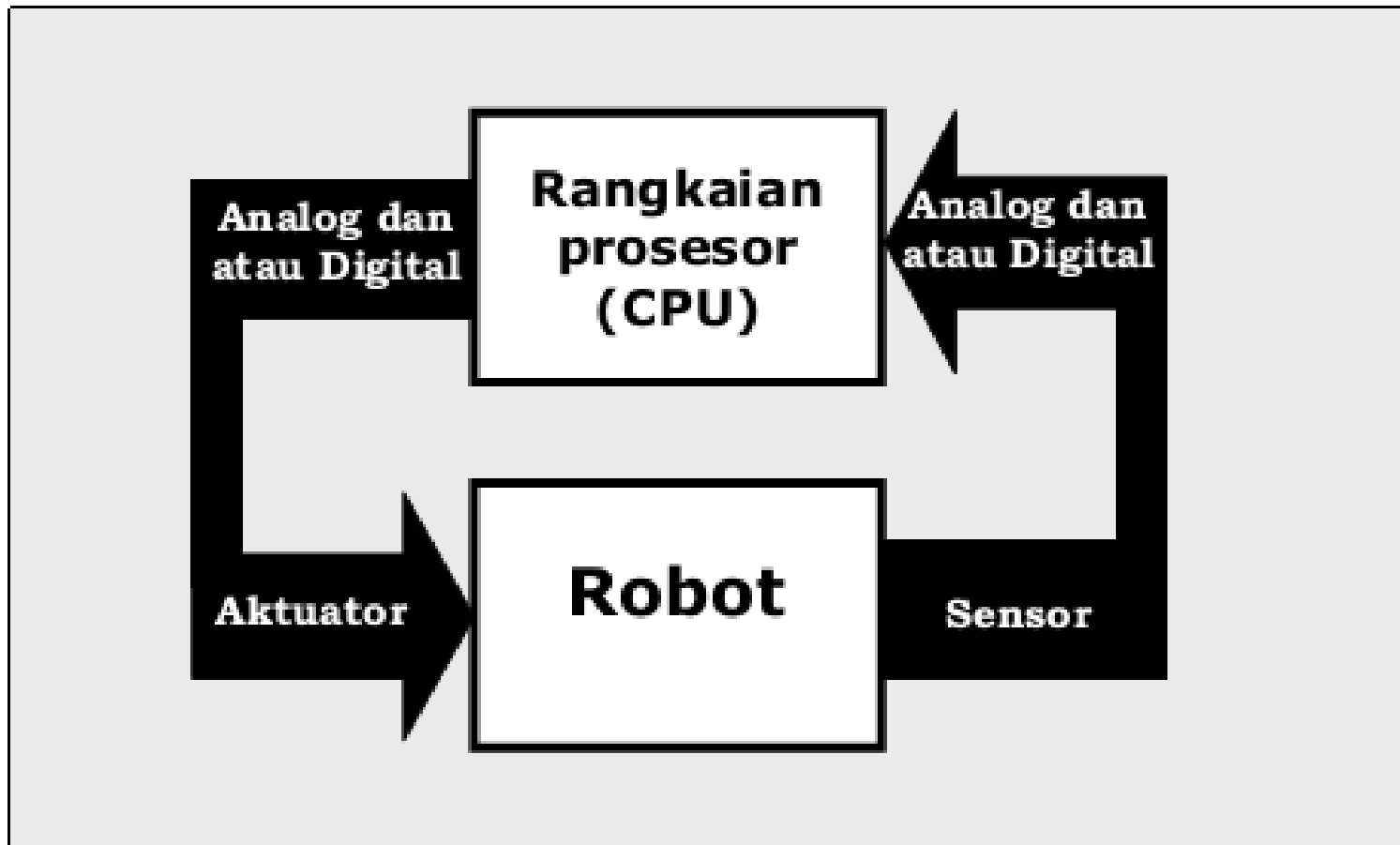
Software

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

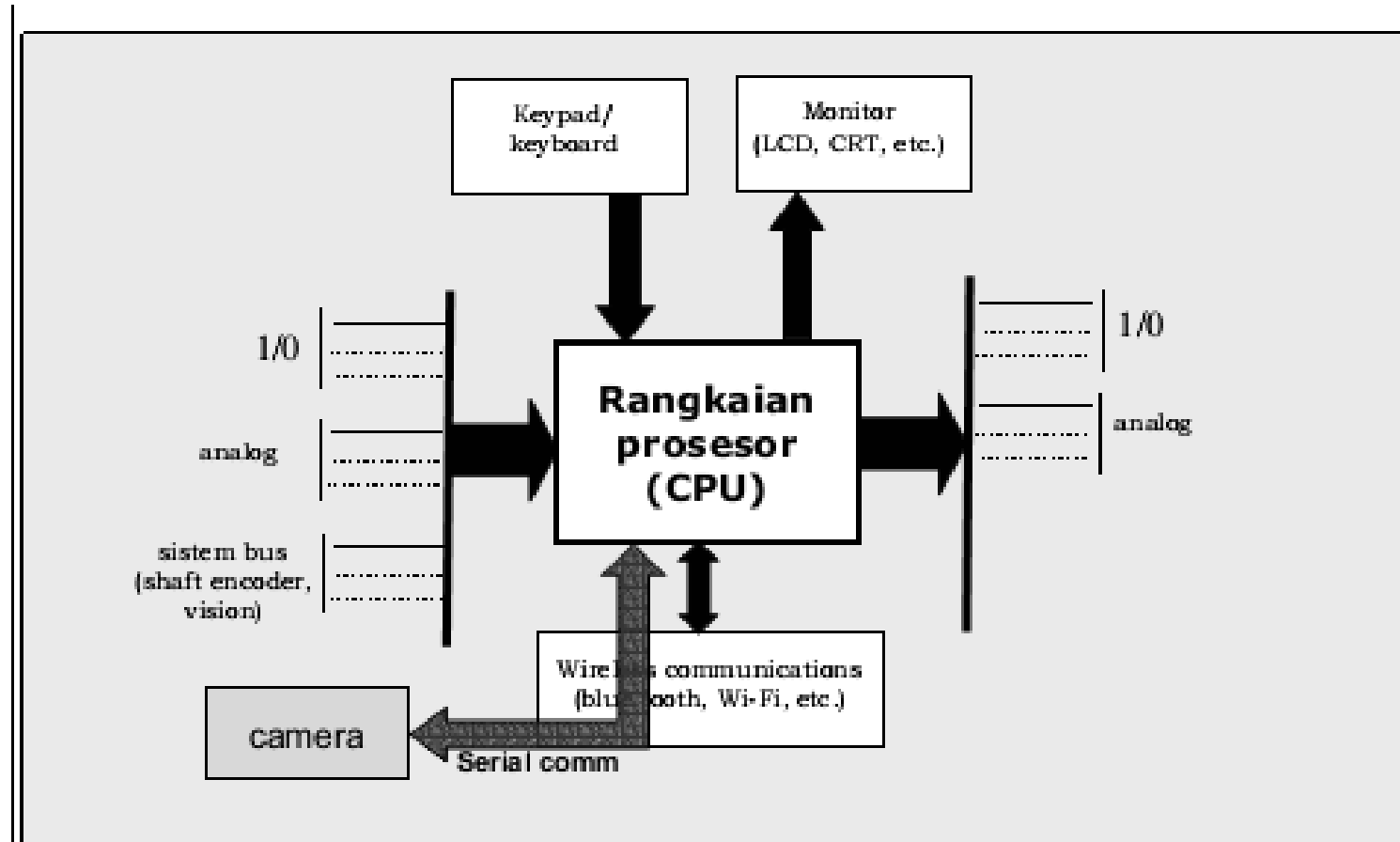
Sistem Robot dan orientasi fungsi



Sistem robot dengan kontroler prosesor



Kontroler Berbasis Processor dengan UI



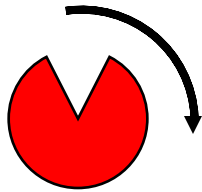
Building a control system for a mobile robot can be very challenging

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

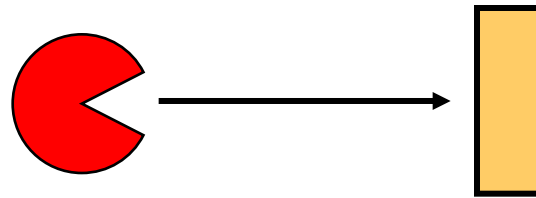
- How will you debug and test your robot?
- What are the performance requirements?
- Can you easily improve aspects of your robot?
- Can you easily integrate new functionality?

Sistem Kontrol Robot

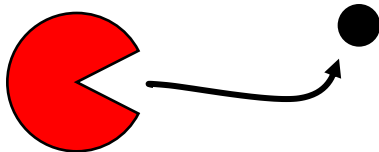
- Dasar dari sistem kontrol adalah perilaku
 - Perilaku harus didefinisikan dengan baik
 - Tiap perilaku harus dapat diuji secara independen



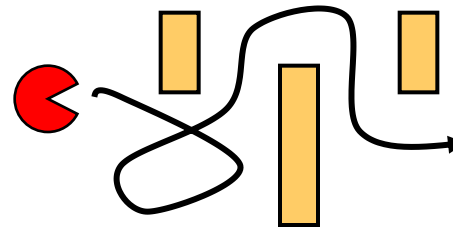
Turn right 90°



Go forward until reach obstacle

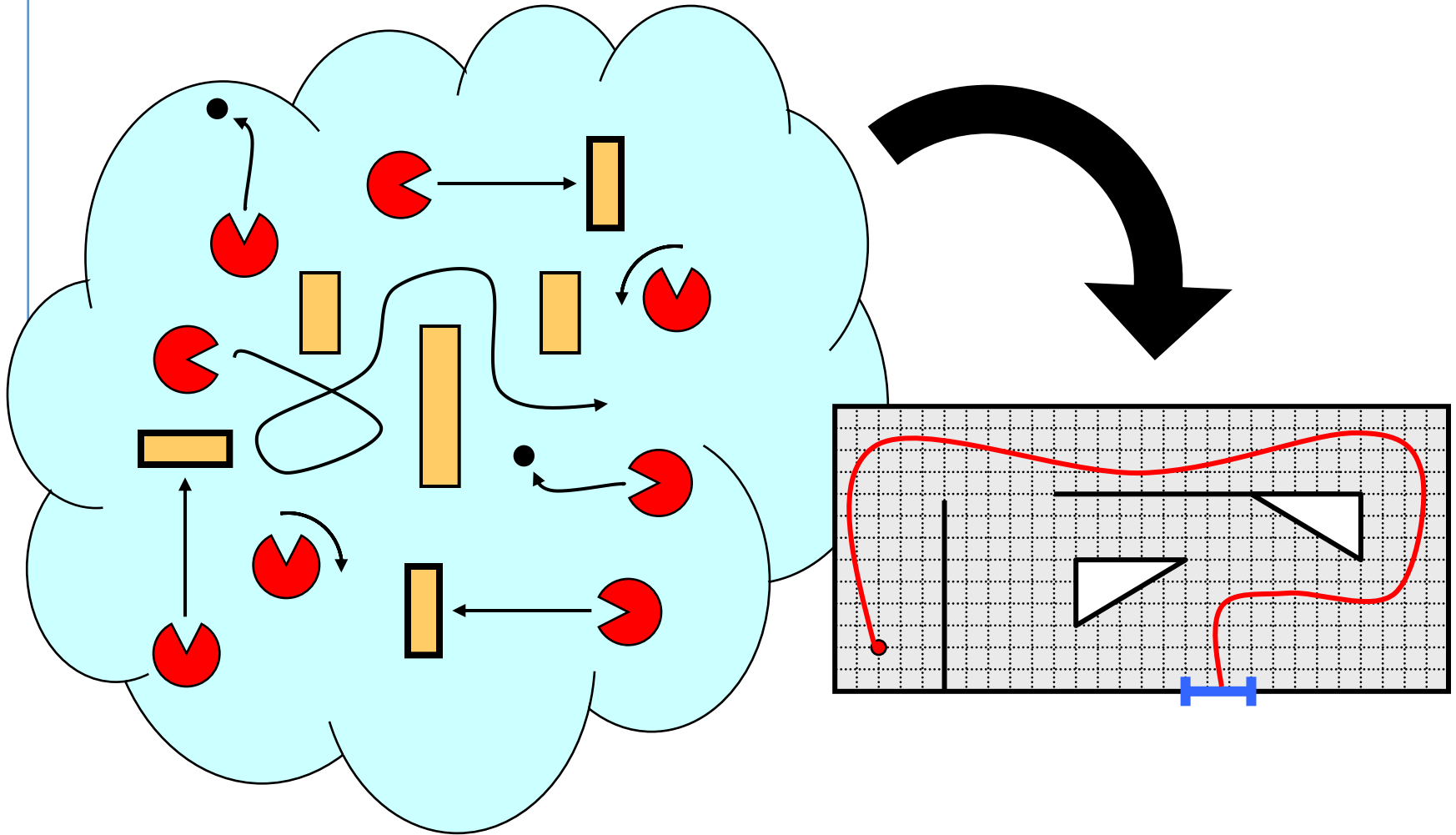


Capture a ball

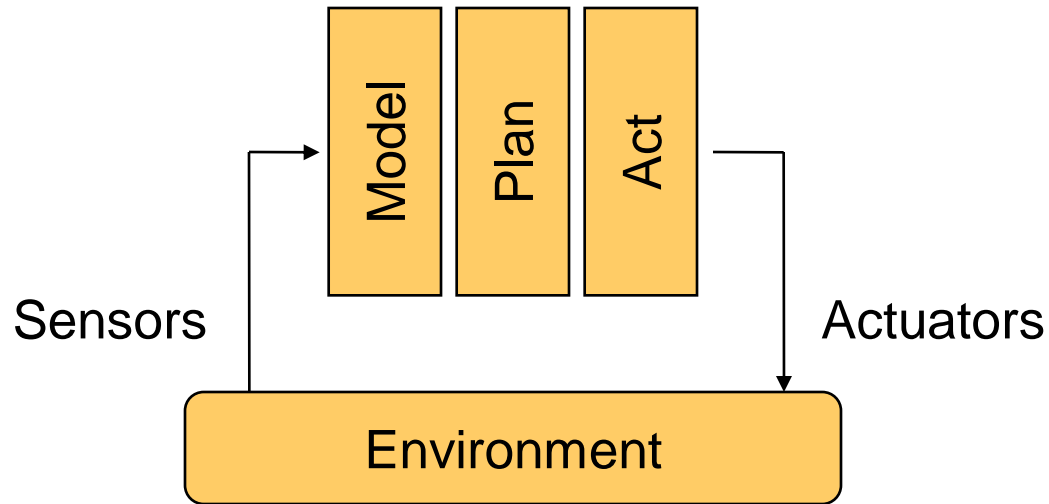


Explore playing field

Sasaran utama menyusun perilaku-perilaku shg tujuan yang diinginkan tercapai

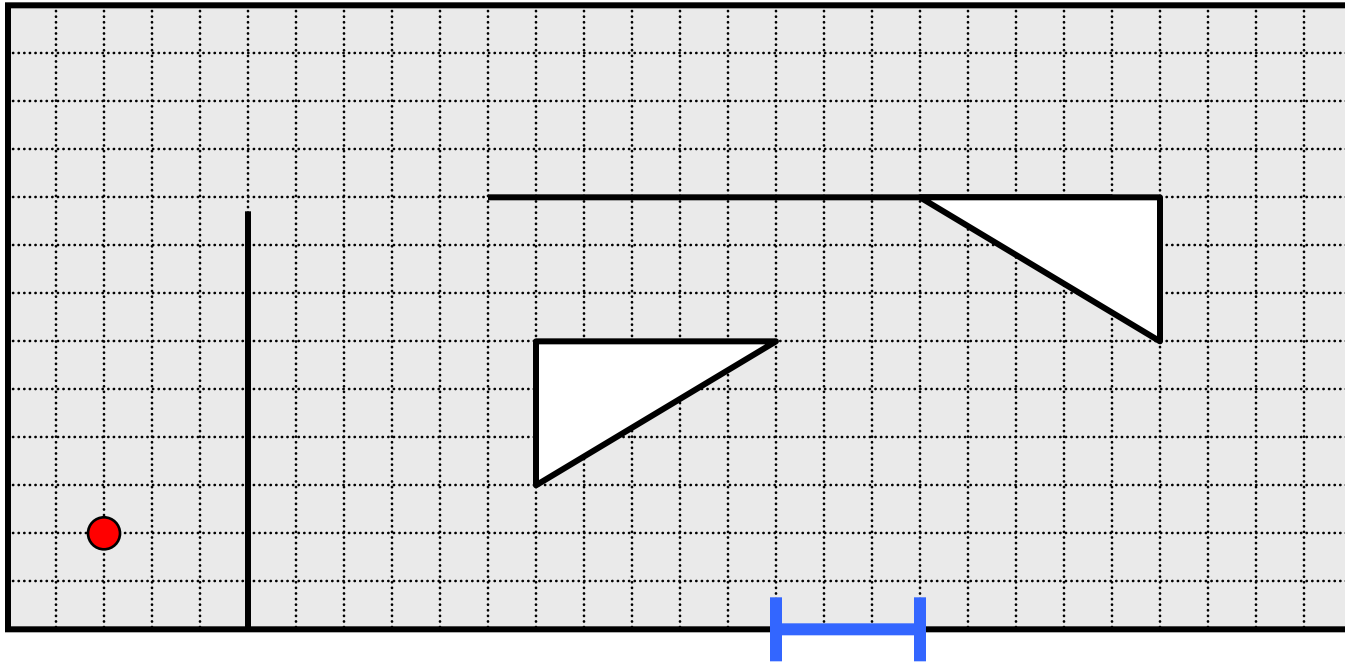


Pendekatan Model-Plan-Act



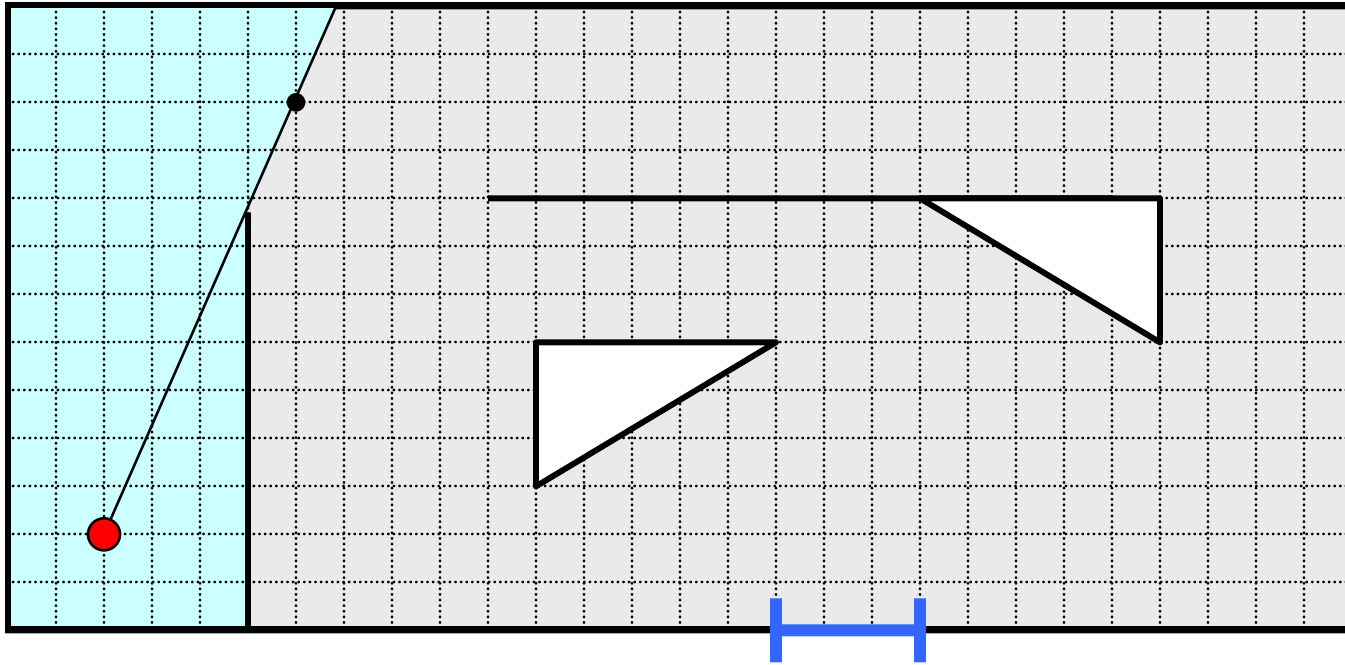
1. Gunakan data sensor untuk membuat model
2. Gunakan model membentuk urutan perilaku yang akan mencapai tujuan yang diinginkan
3. Jalankan rencana (menulis perilaku)

Exploring the playing field using model-plan-act approach



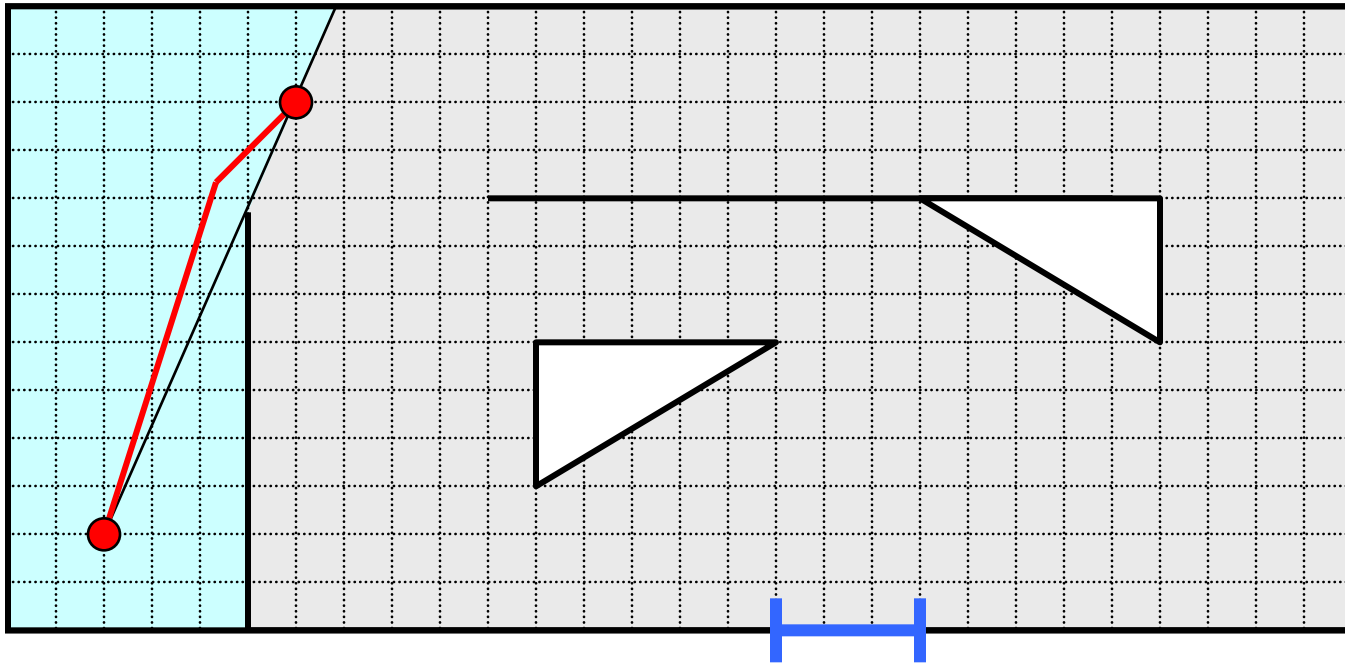
Red dot is the mobile robot
while the blue line is the mousehole

Exploring the playing field using model-plan-act approach



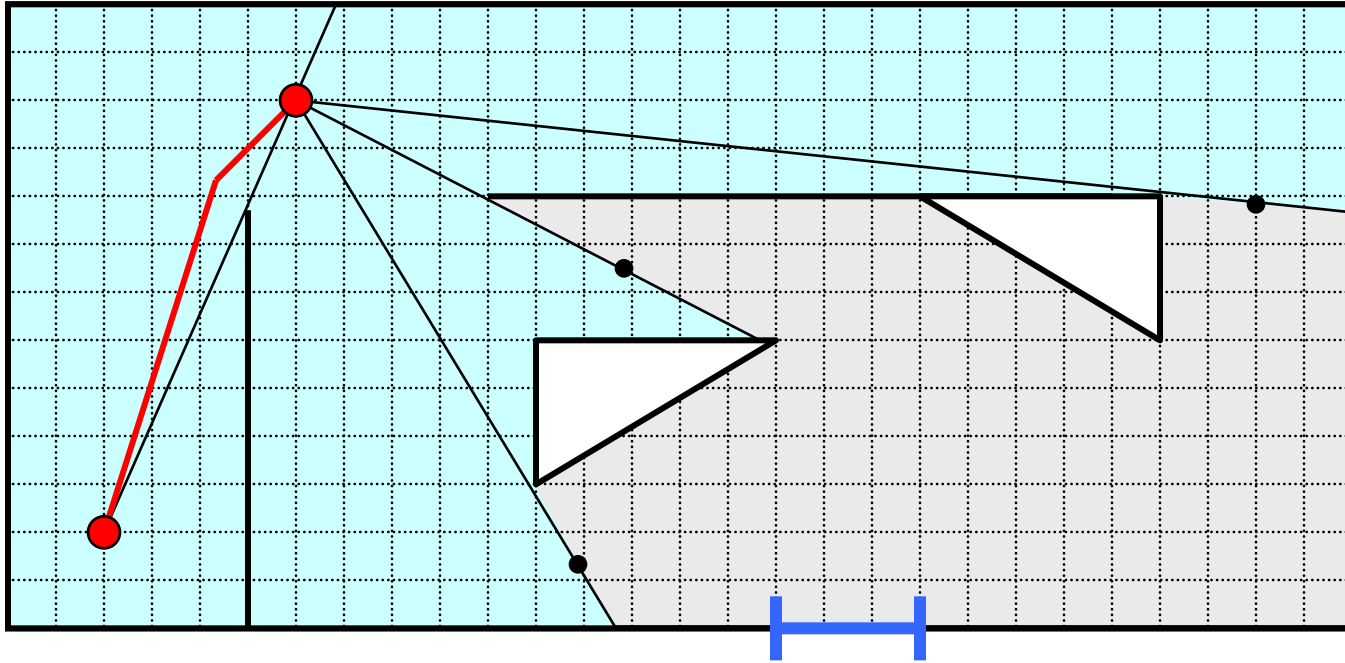
Robot uses sensors to create local map of the world and identify unexplored areas

Exploring the playing field using model-plan-act approach



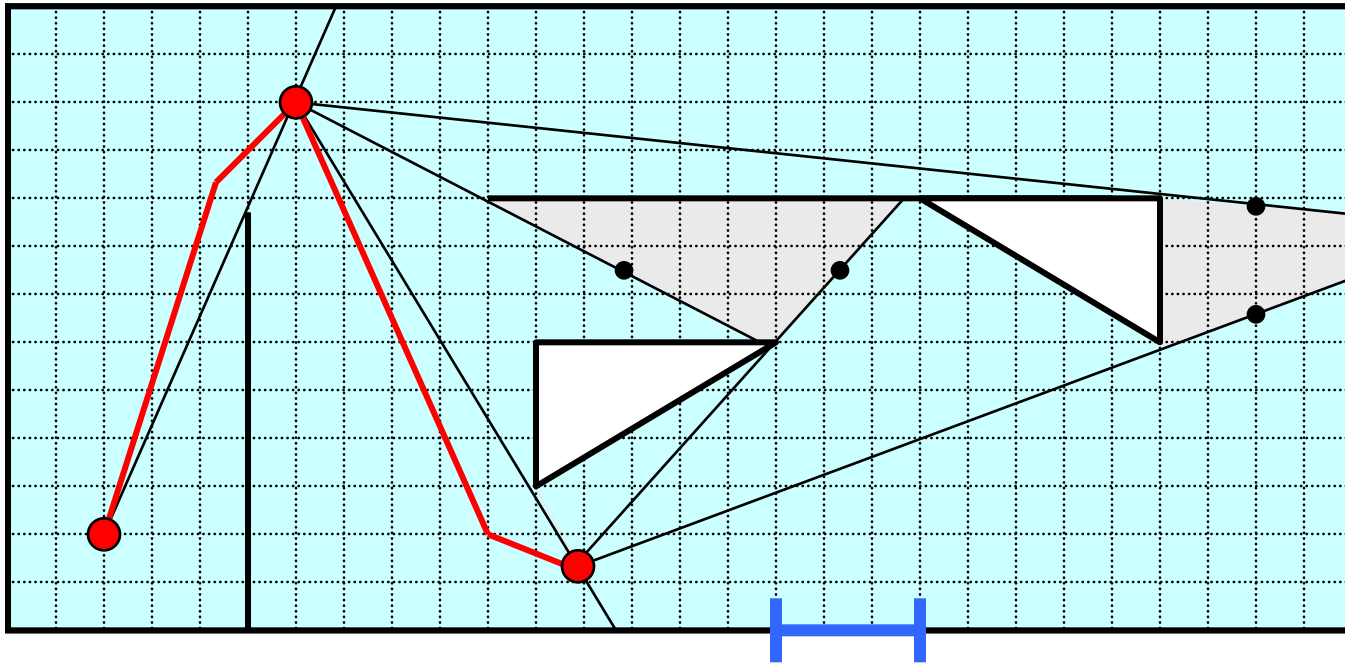
Robot moves to midpoint of
unexplored boundary

Exploring the playing field using model-plan-act approach



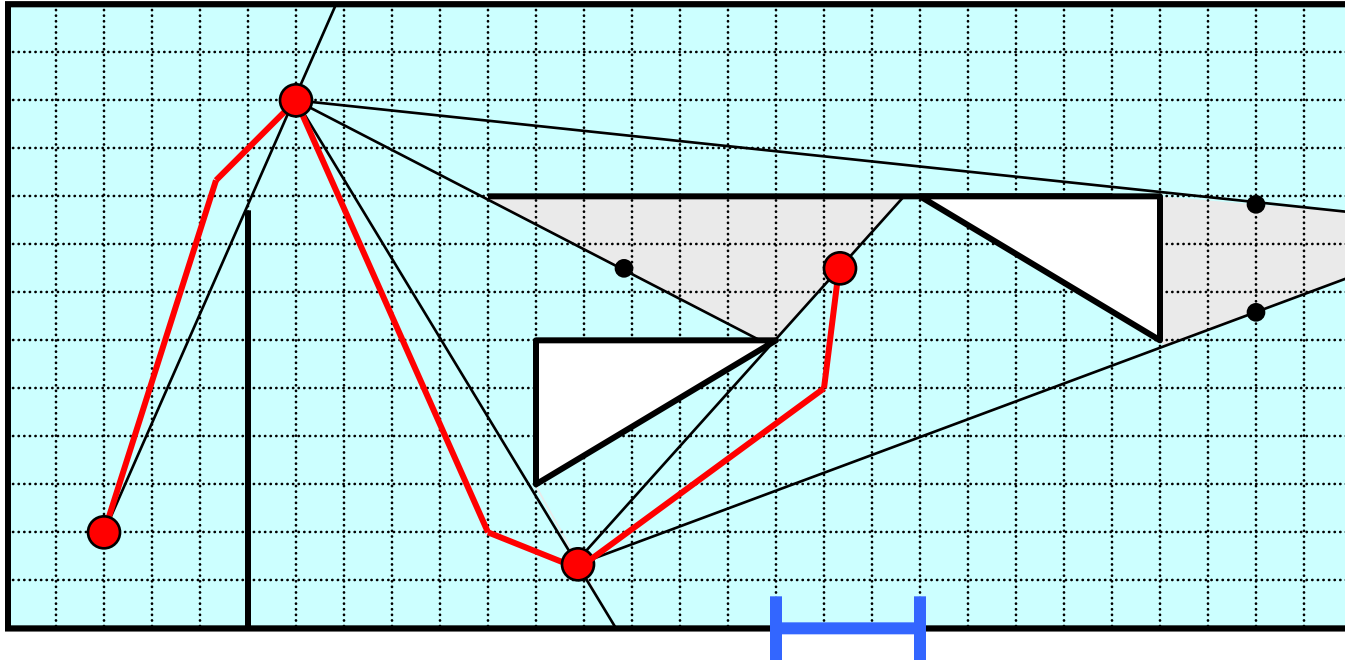
Robot performs a second sensor scan and must align the new data with the global map

Exploring the playing field using model-plan-act approach



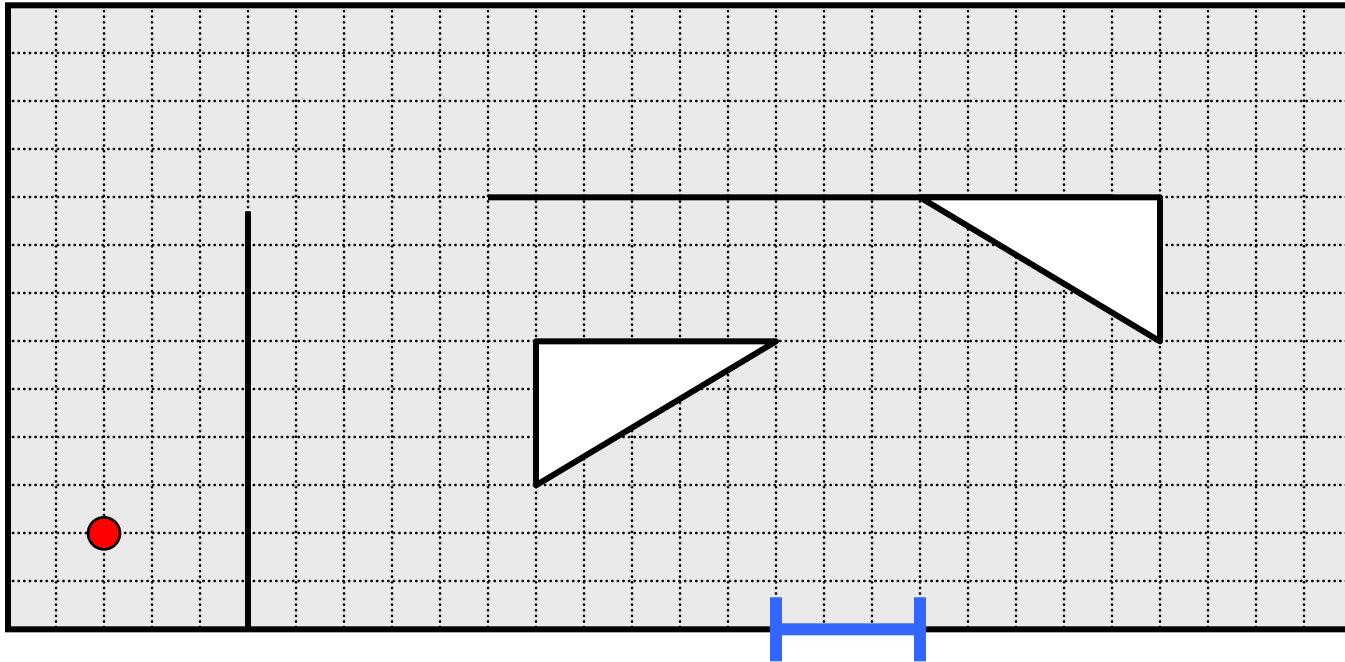
Robot continues to explore
the playing field

Exploring the playing field using model-plan-act approach



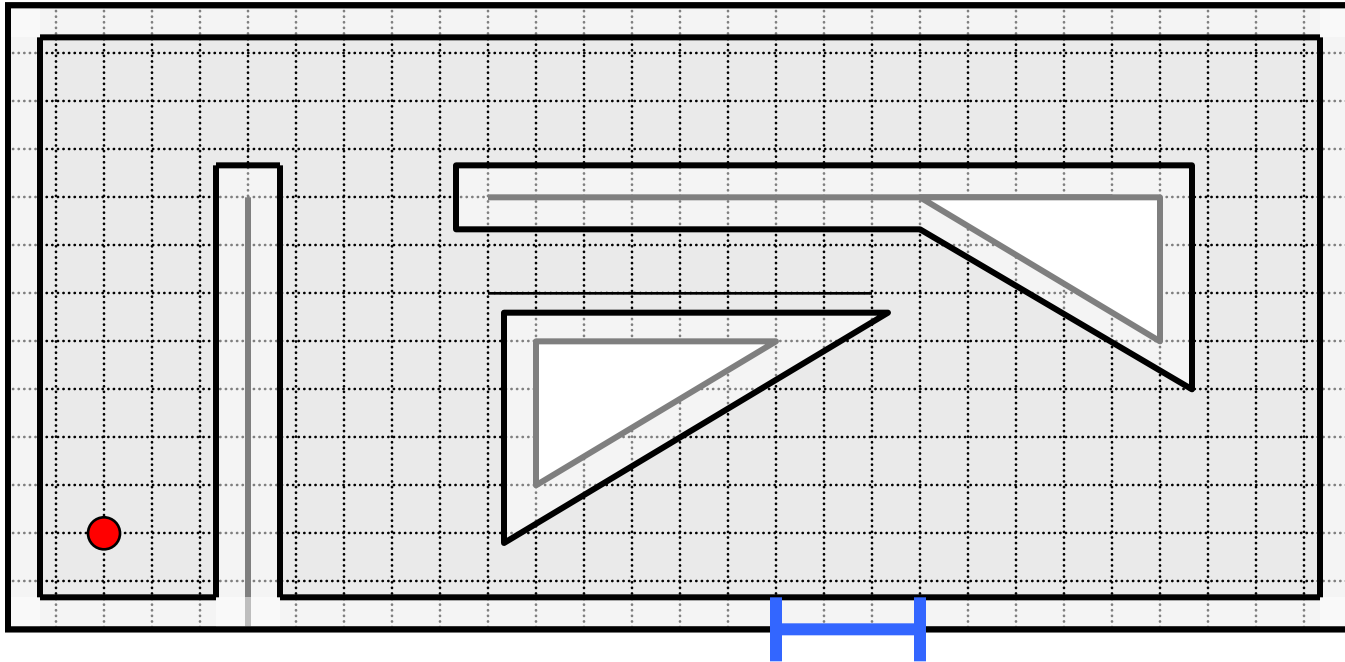
Robot must recognize when it starts to see areas which it has already explored

Finding a mousehole using model-plan-act approach



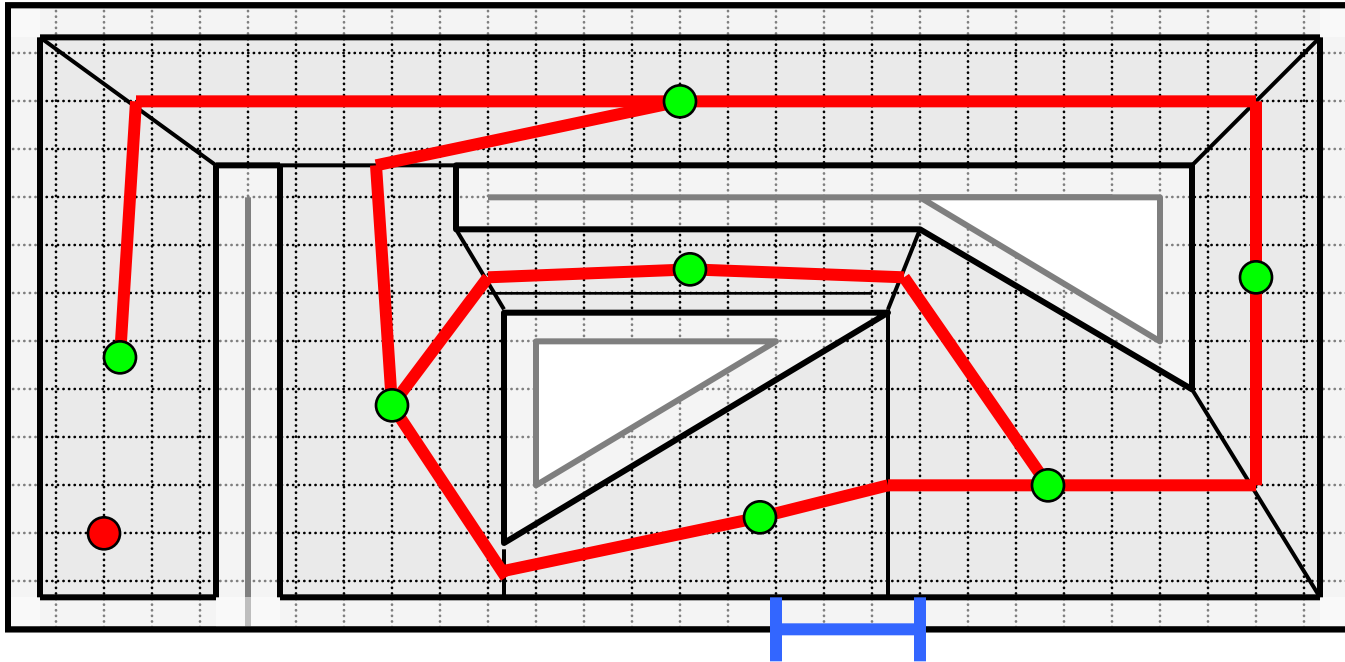
Given the global map,
the goal is to find the mousehole

Finding a mousehole using model-plan-act approach



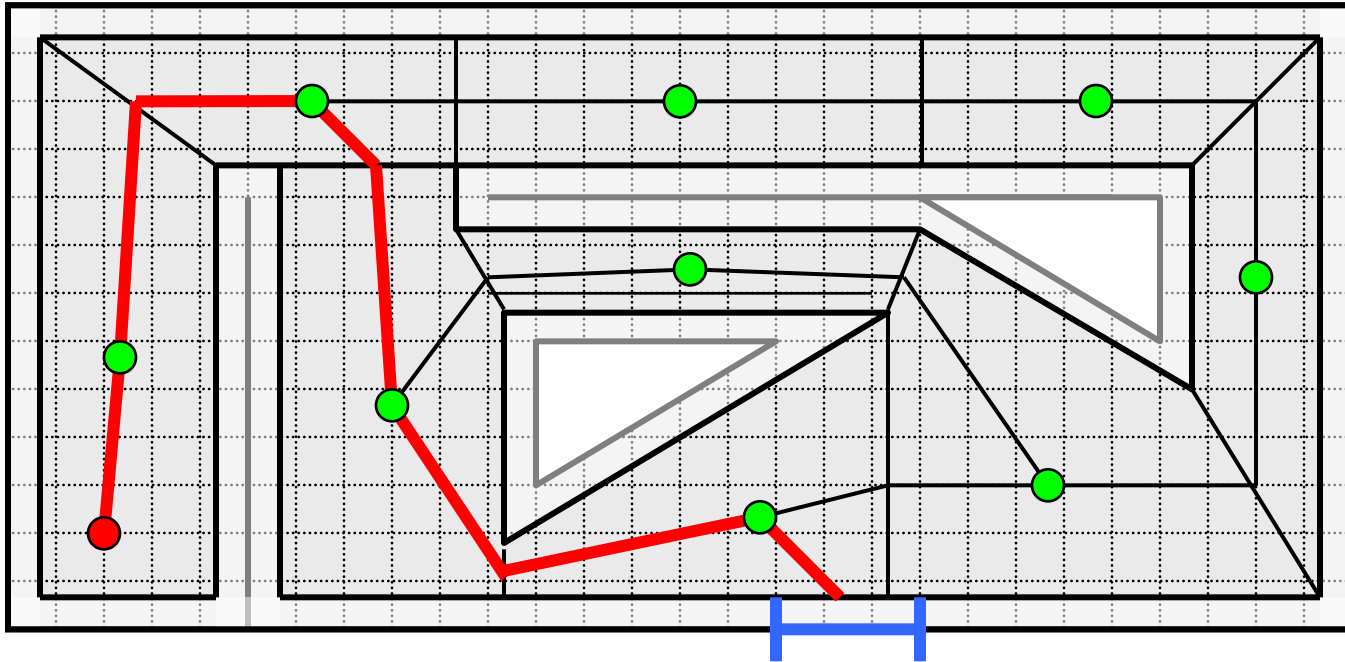
Transform world into configuration space
by convolving robot with all obstacles

Finding a mousehole using model-plan-act approach



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

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

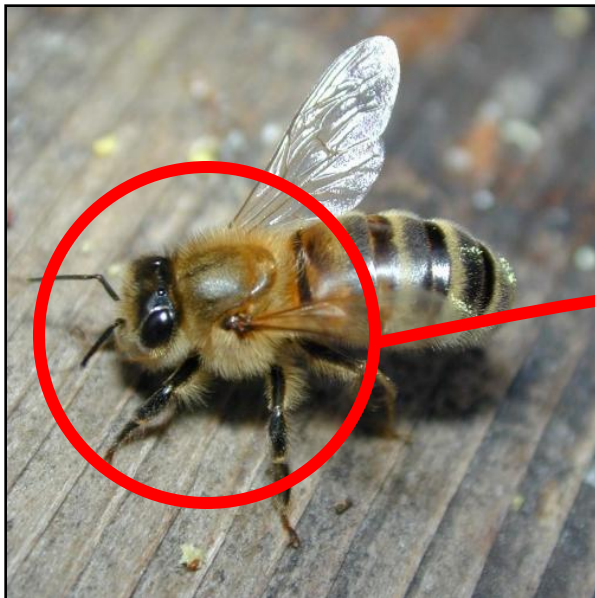
- Pengetahuan global dapat digunakan untuk menentukan optimasi
- Dapat membuat rencana

- Disadvantages

- Harus mengimplementasikan semua unit fungsional sebelum digunakan
- komputasi secara intensif
- Memerlukan data sensor yang sangat baik untuk model yang akurat
- Bekerja buruk dalam lingkungan yang dinamis

Emergent Approach

Makhluk hidup seperti lebah madu mampu mengeksplorasi lingkungan mereka dan menemukan target (madu)



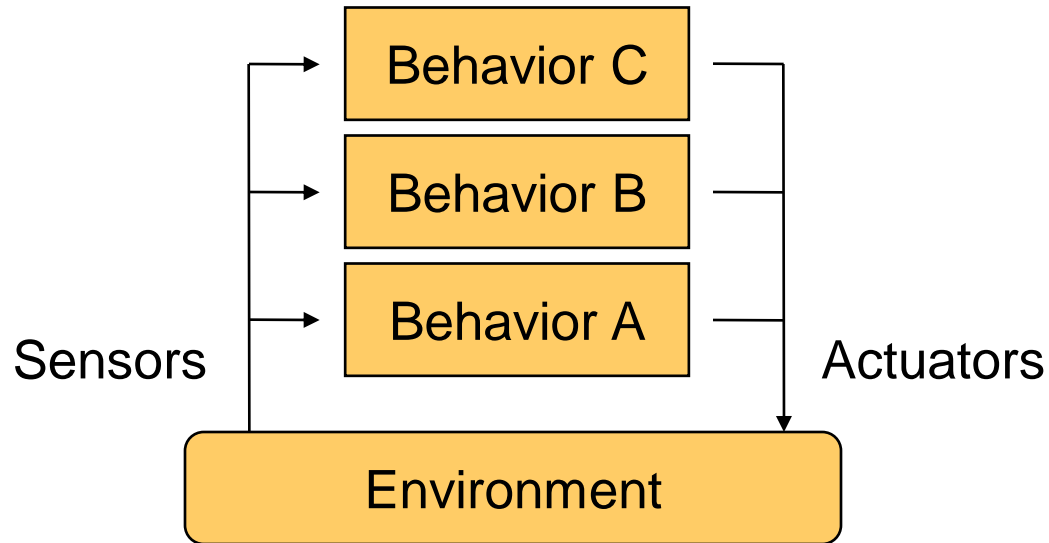
Is this bee using the model-plan-act approach?

Emergent Approach



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

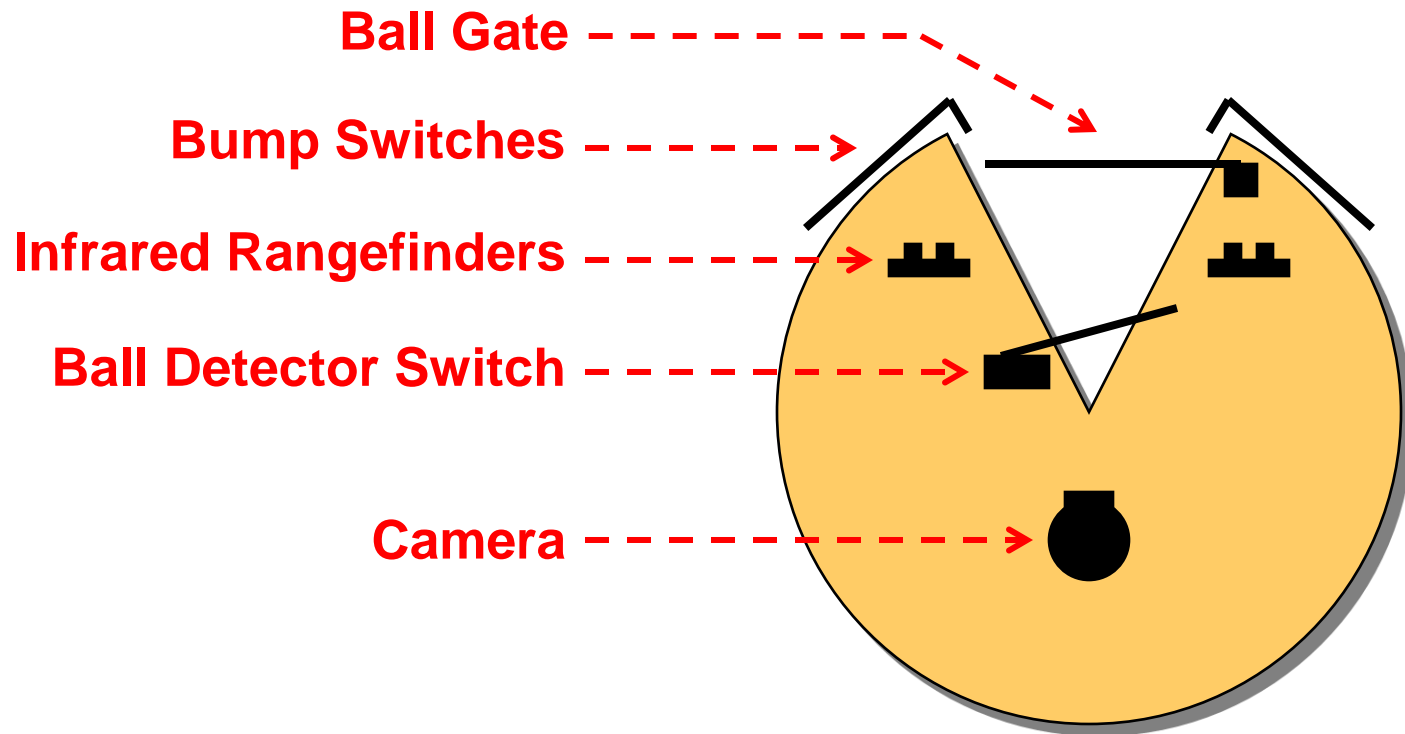
Emergent Approach



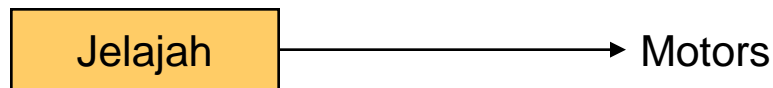
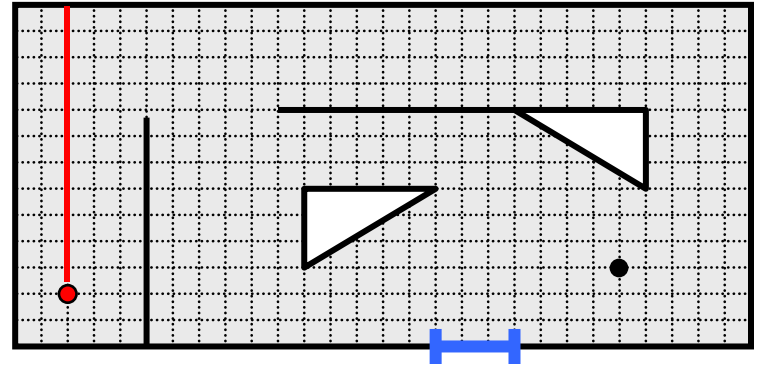
As in biological systems, the emergent approach uses simple behaviors to directly couple sensors and actuators

Higher level behaviors are layered on top of lower level behaviors

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



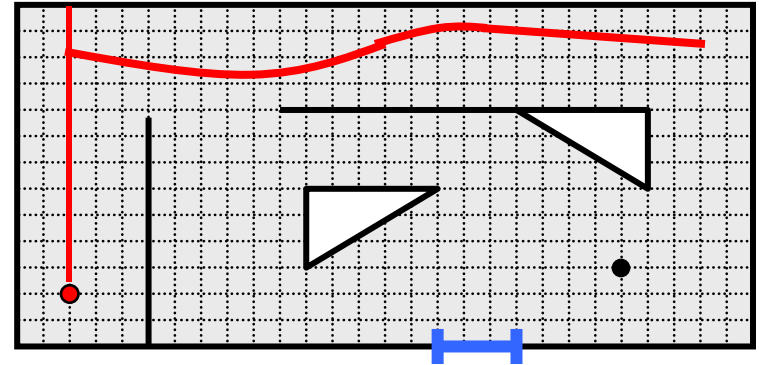
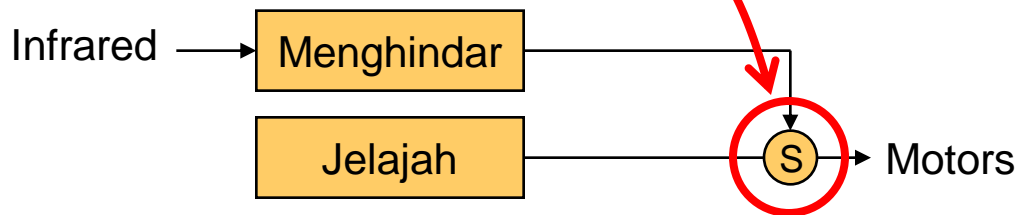
Layering simple behaviors can create much more complex emergent behavior



Cruise behavior simply moves robot forward

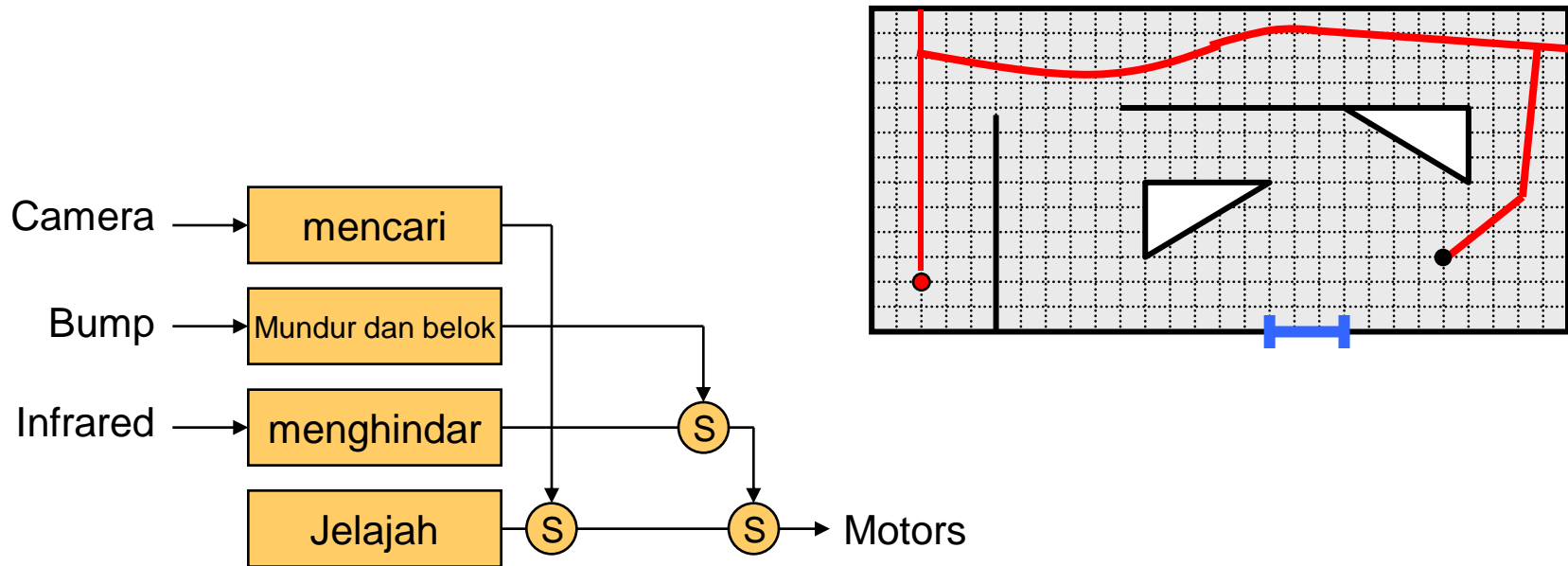
Layering simple behaviors can create much more complex emergent behavior

Subsumption



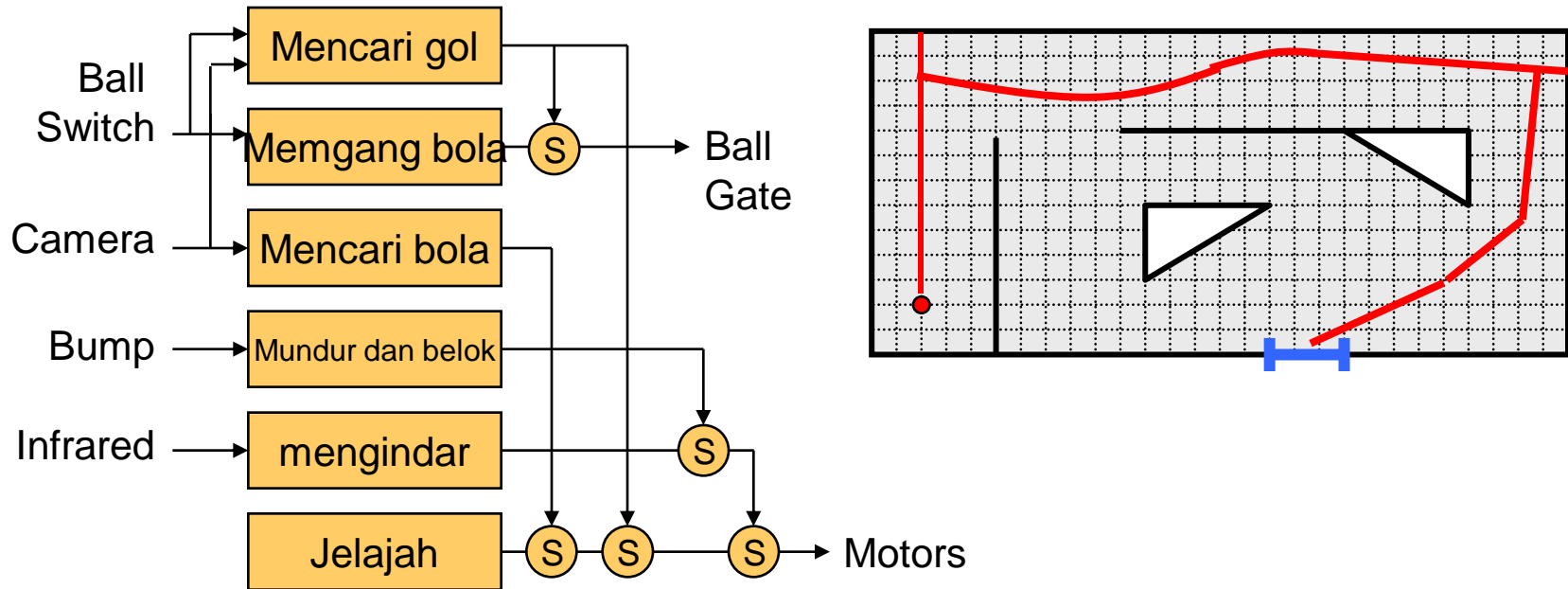
Left motor speed inversely proportional to left IR range
Right motor speed inversely proportional to right IR range
If both IR < threshold stop and turn right 120 degrees

Layering simple behaviors can create much more complex emergent behavior



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

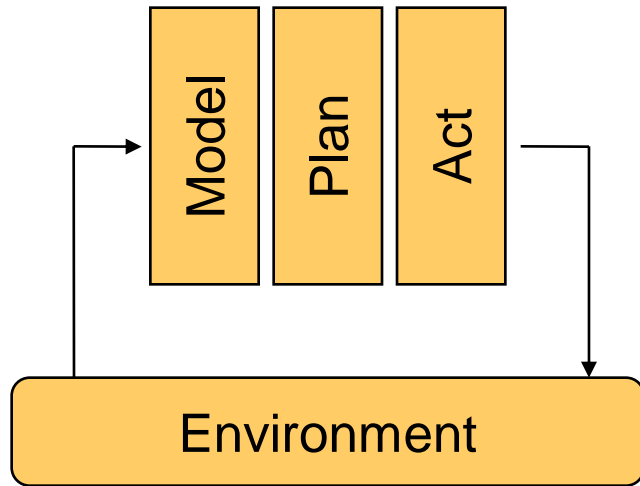


All behaviors are always running in parallel and an arbiter is responsible for picking which behavior can access the actuators

Advantages and disadvantages of the behavioral approach

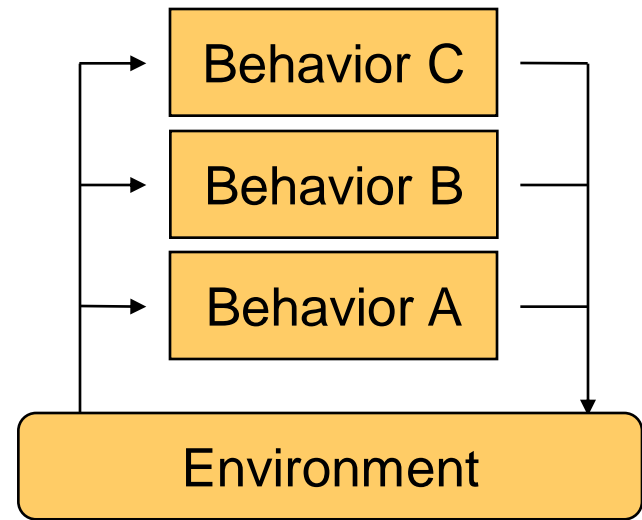
- keuntungan
 - Pengembangan bertahap sangat alami
 - Modularitas membuat eksperimen mudah
 - Dapat menangani lingkungan yang dinamis
- kekurangan
 - Sulit untuk menilai apa robot benar-benar akan melakukan
 - Tidak ada kinerja atau kelengkapan jaminan
 - Debugging bisa sangat sulit

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



Model-Plan-Act

Lots of internal state
Lots of preliminary planning
Fixed plan of behaviors



Emergent

Very little internal state
No preliminary planning
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.

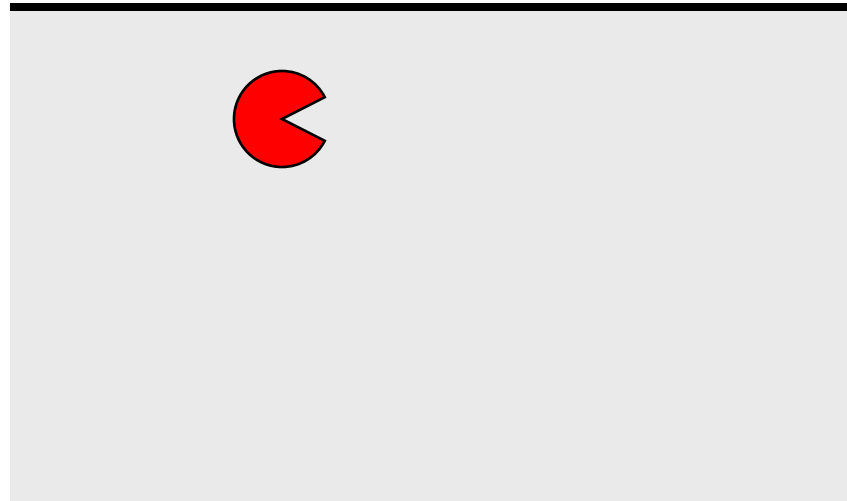
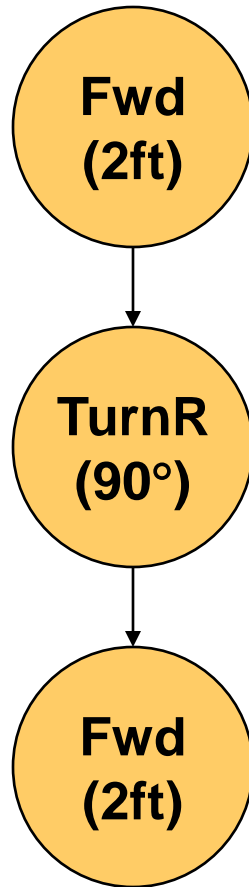
Fwd
(dist)

Fwd behavior moves robot straight forward a given distance

TurnR
(deg)

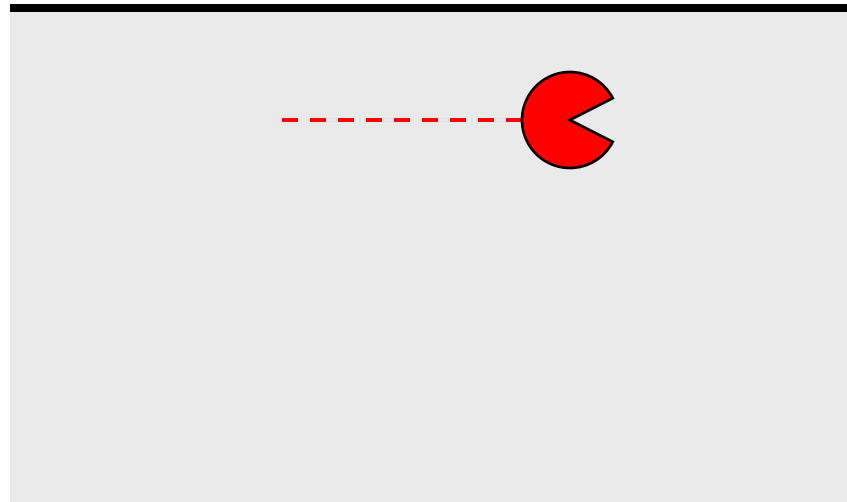
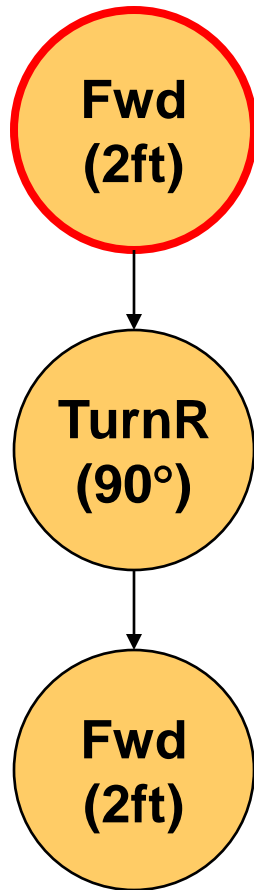
TurnR behavior turns robot to the right a given number of degrees

Finite State Machines offer another alternative for combining behaviors



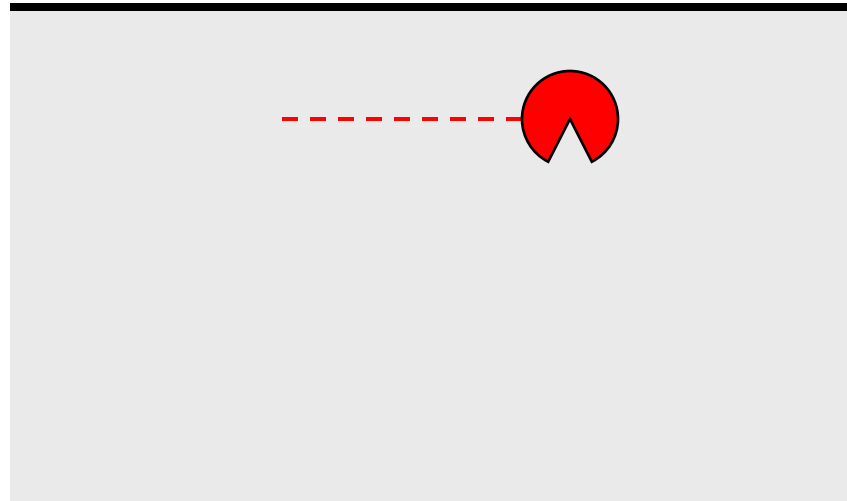
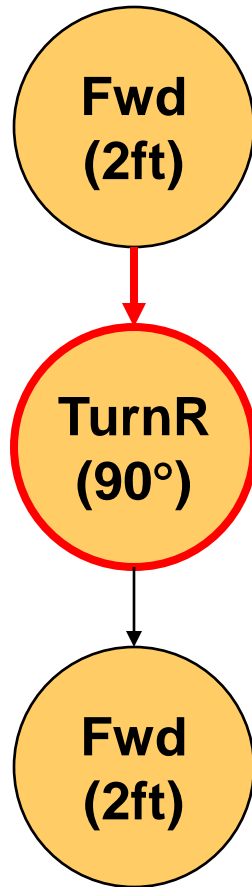
Each state is just a specific behavior instance - link them together to create an open loop control system

Finite State Machines offer another alternative for combining behaviors



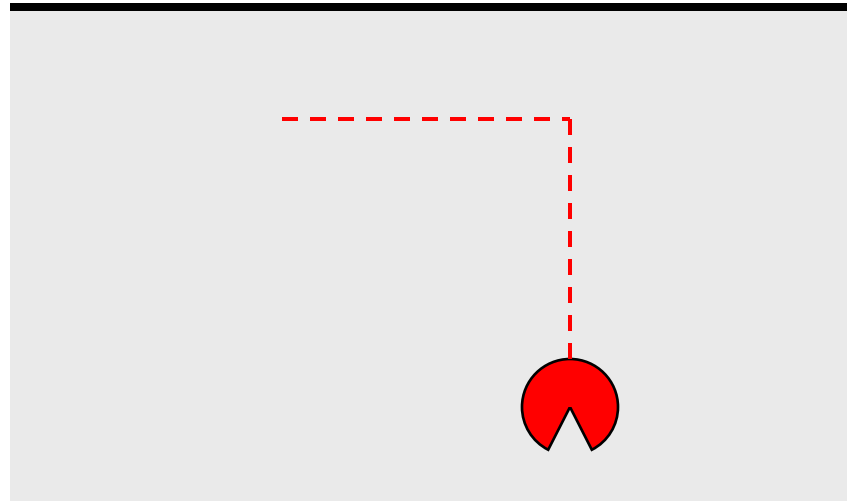
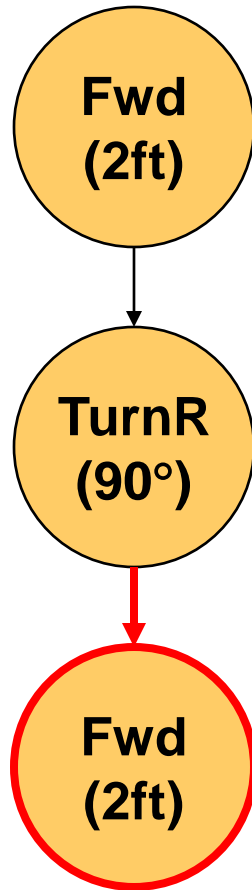
Each state is just a specific behavior instance - link them together to create an open loop control system

Finite State Machines offer another alternative for combining behaviors



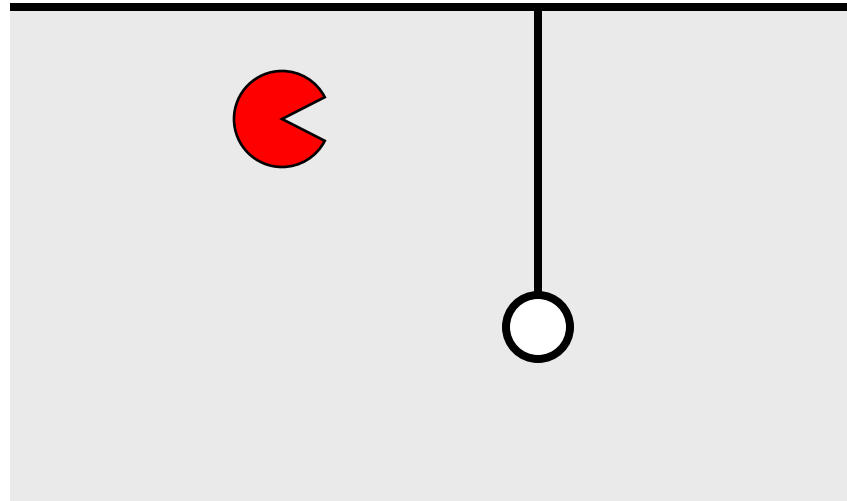
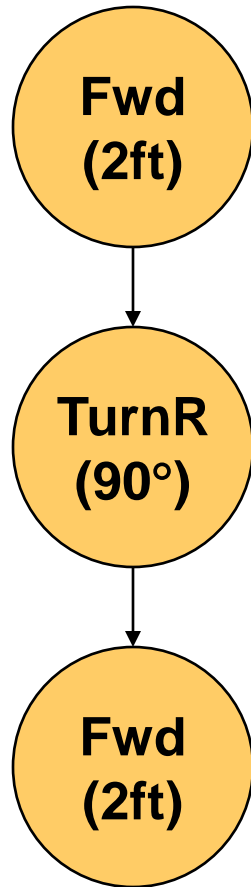
Each state is just a specific behavior instance - link them together to create an open loop control system

Finite State Machines offer another alternative for combining behaviors



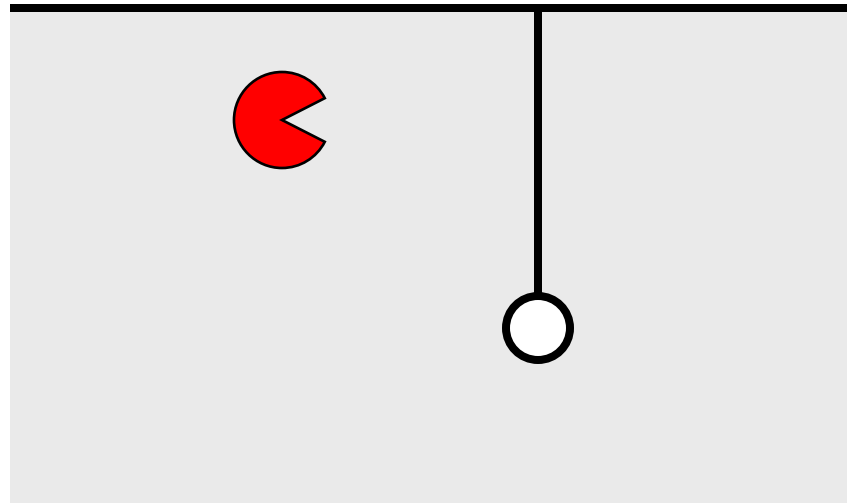
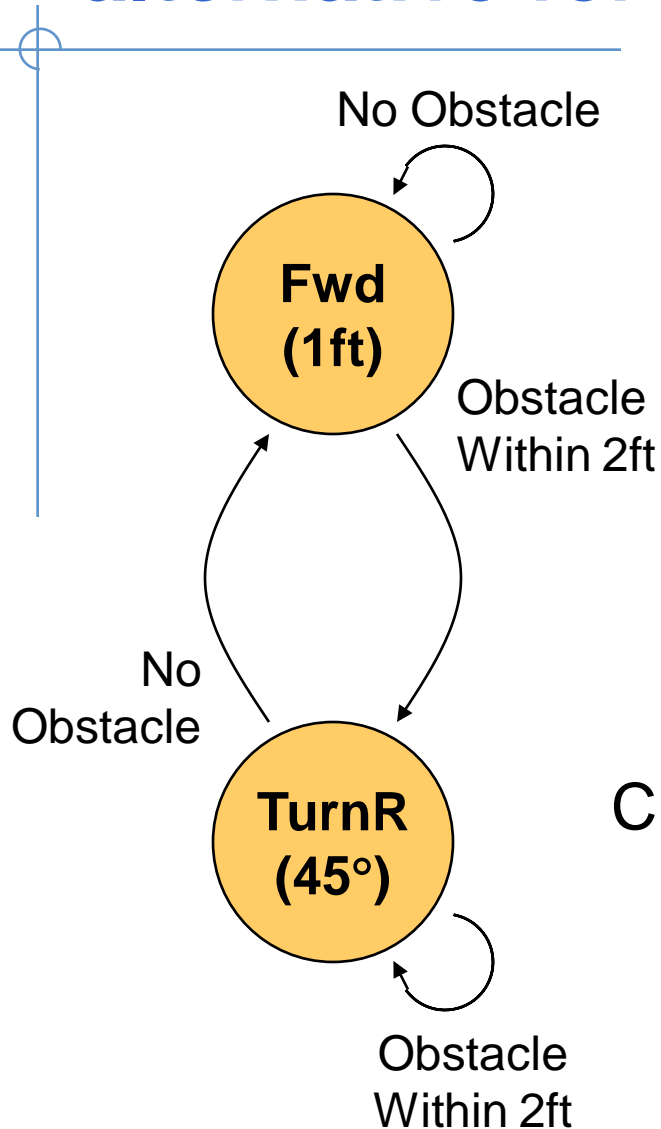
Each state is just a specific behavior instance - link them together to create an open loop control system

Finite State Machines offer another alternative for combining behaviors



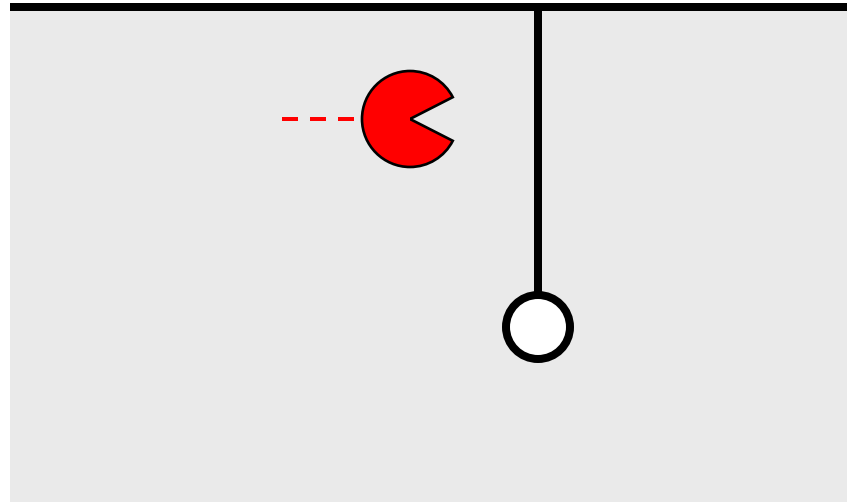
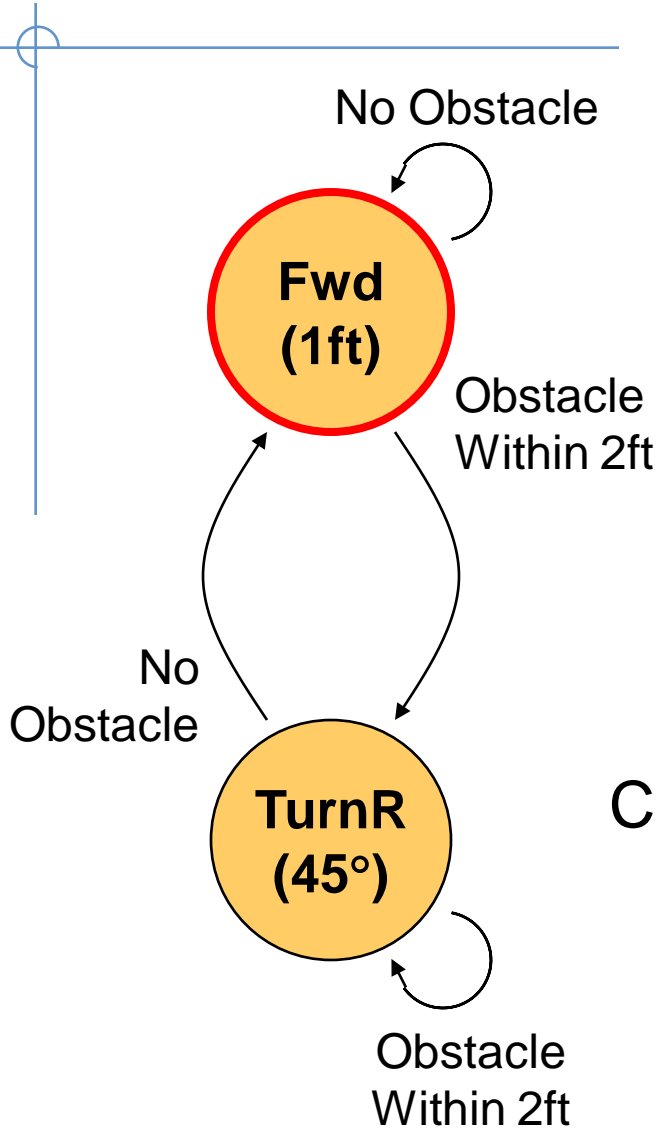
Since the Maslab playing field is unknown, open loop control systems have no hope of success!

Finite State Machines offer another alternative for combining behaviors



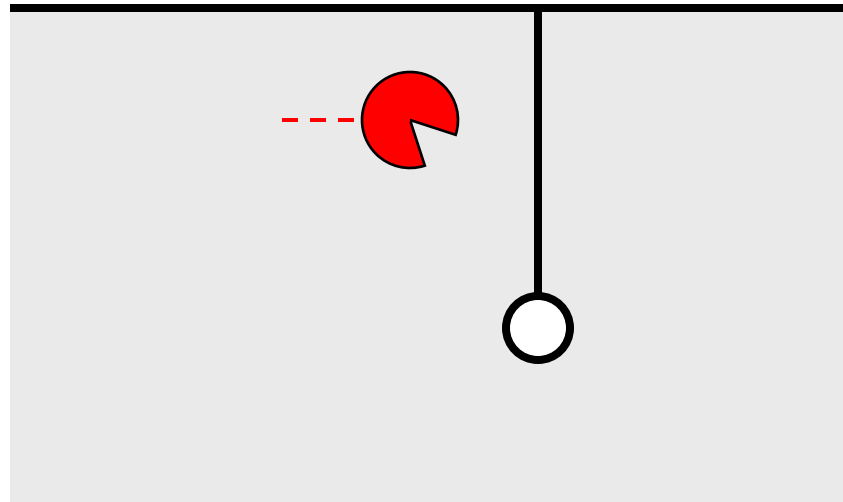
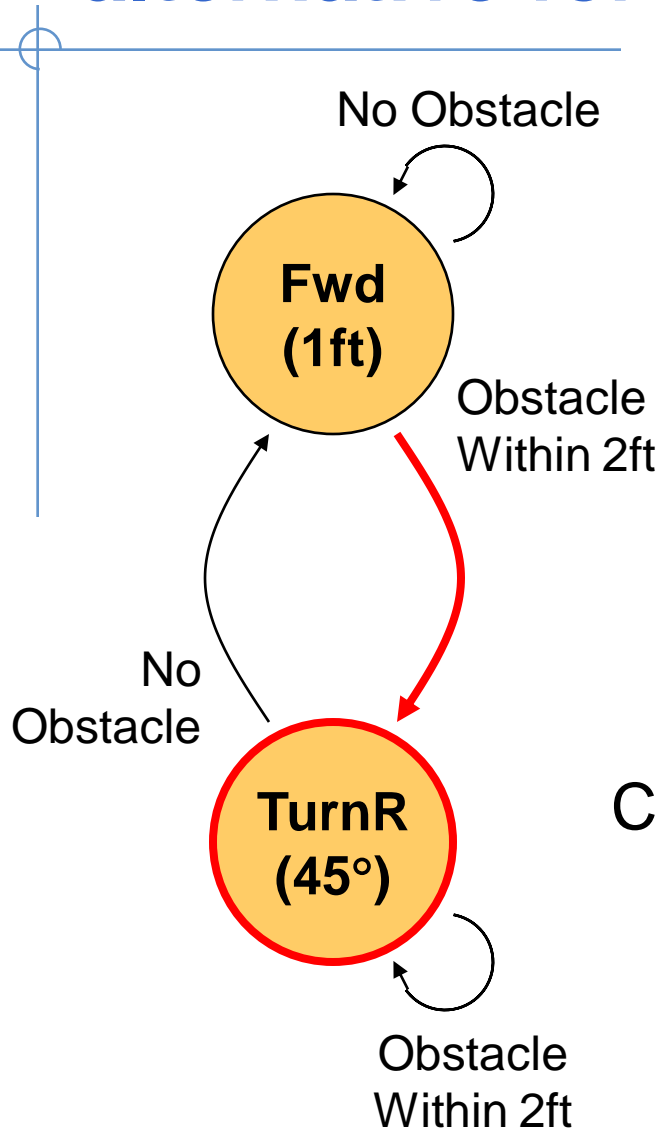
Closed loop finite state machines use sensor data as feedback to make state transitions

Finite State Machines offer another alternative for combining behaviors



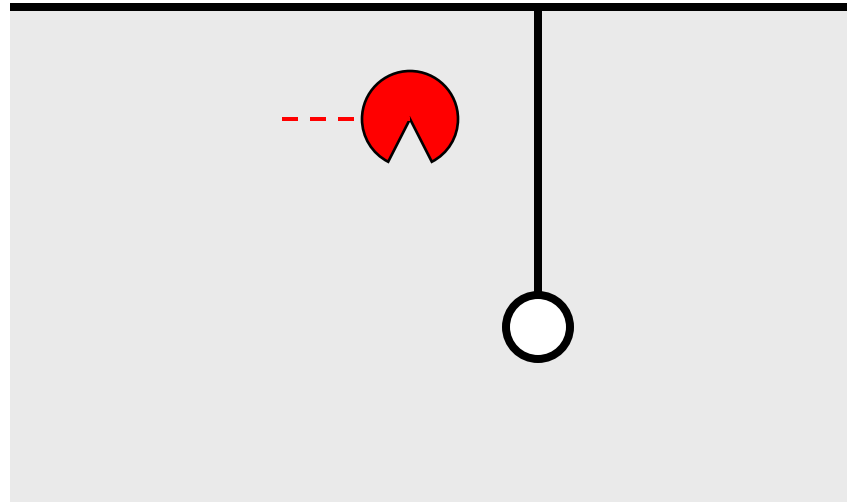
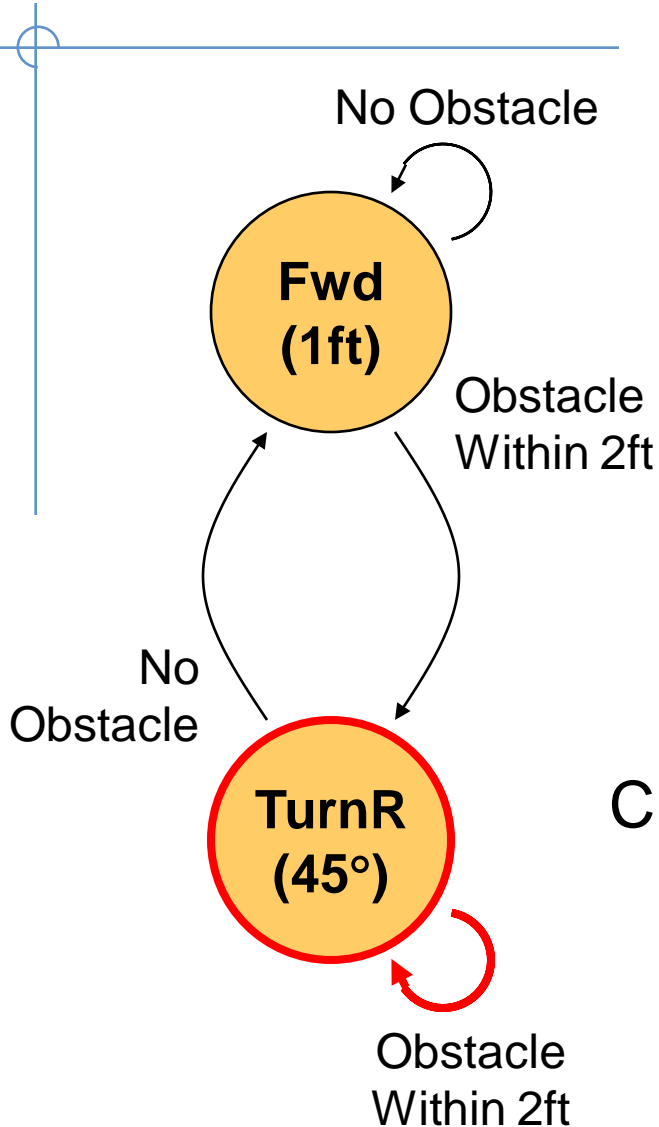
Closed loop finite state machines use sensor data as feedback to make state transitions

Finite State Machines offer another alternative for combining behaviors



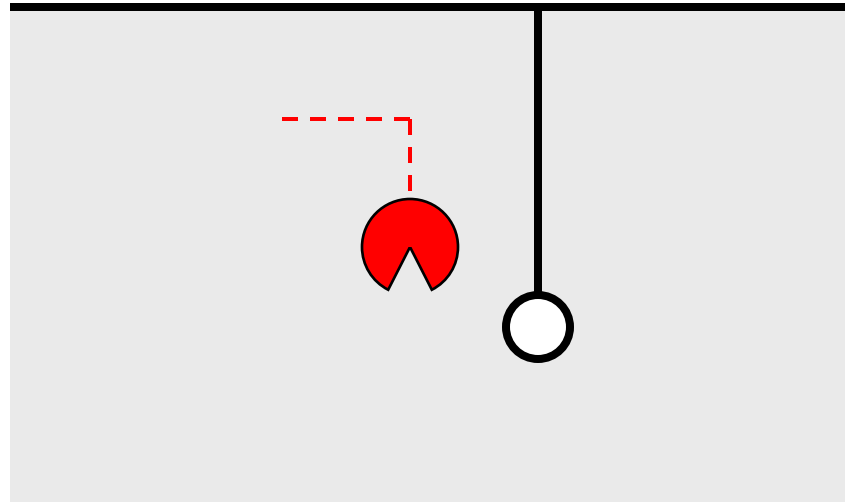
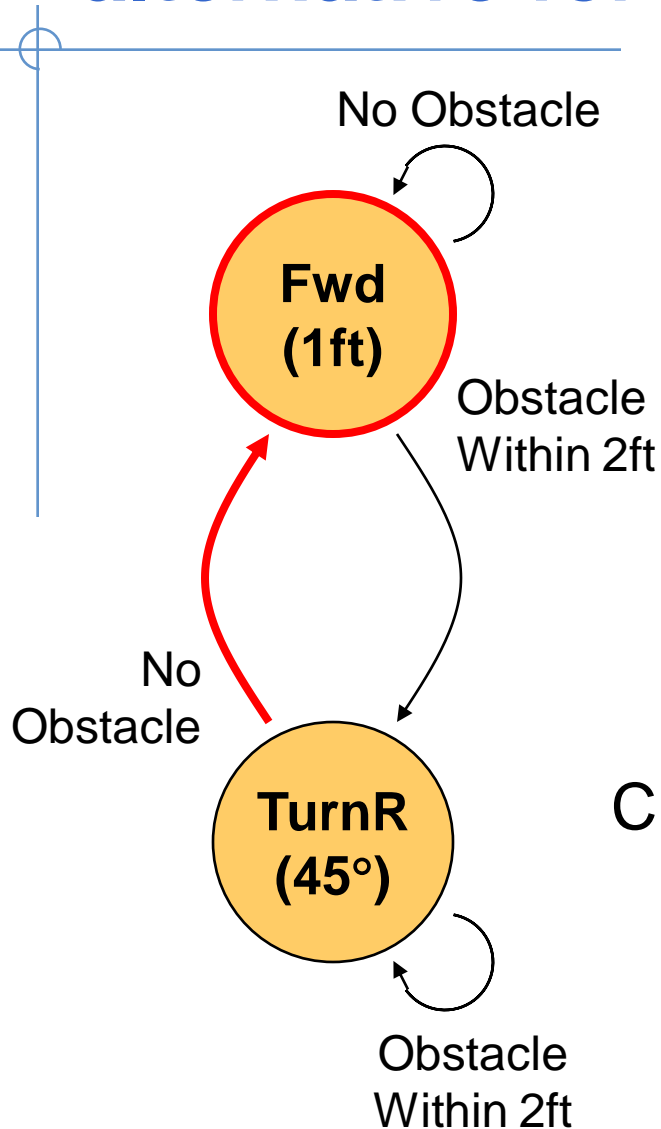
Closed loop finite state machines use sensor data as feedback to make state transitions

Finite State Machines offer another alternative for combining behaviors



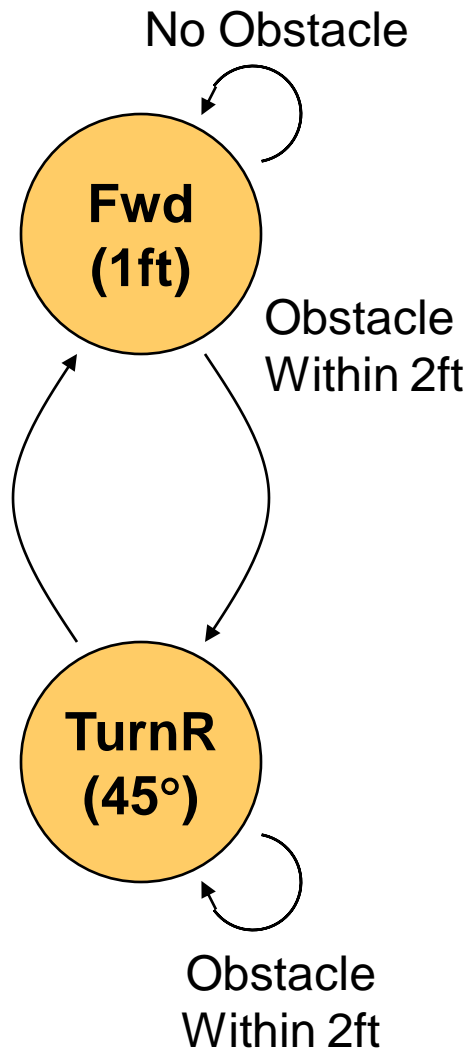
Closed loop finite state machines use sensor data as feedback to make state transitions

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



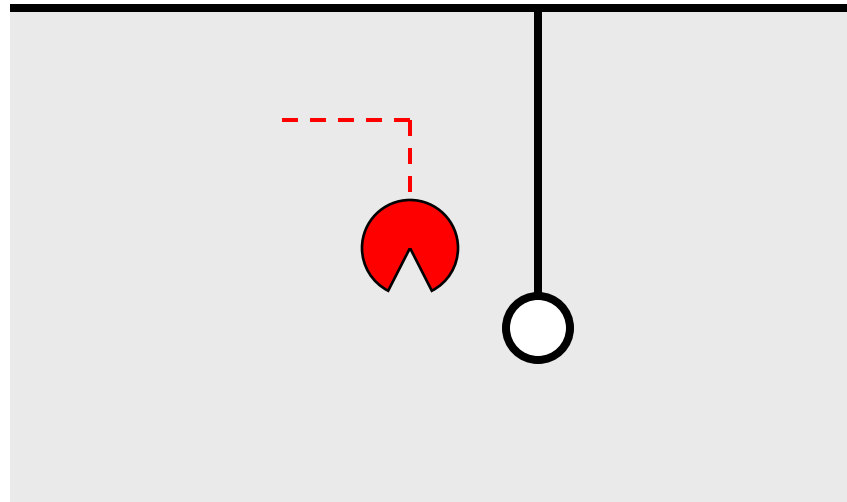
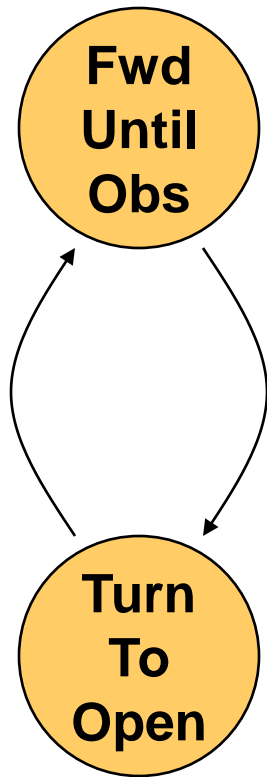
```
switch ( state ) {  
  
    case States.Fwd_1 :  
        moveFoward(1);  
        if ( distanceToObstacle() < 2 )  
            state = TurnR_45;  
        break;  
  
    case States.TurnR_45 :  
        turnRight(45);  
        if ( distanceToObstacle() >= 2 )  
            state = Fwd_1;  
        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

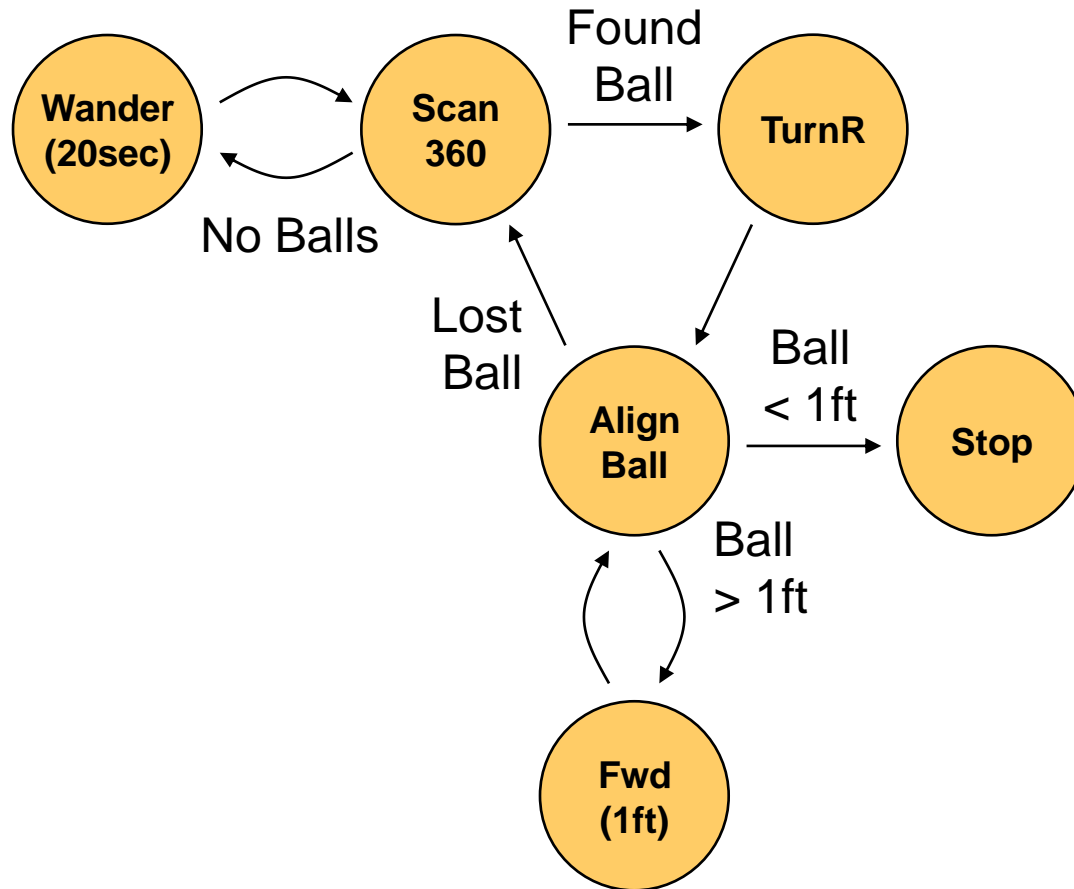
```
switch ( state ) {  
  
    case States.Fwd_1 :  
        moveFoward(1);  
        if ( distanceToObstacle() < 2 )  
            state = TurnR_45;  
        break;  
  
    case States.TurnR_45 :  
        turnRight(45);  
        if ( distanceToObstacle() >= 2 )  
            state = Fwd_1;  
        break;  
  
}
```


Finite State Machines offer another alternative for combining behaviors

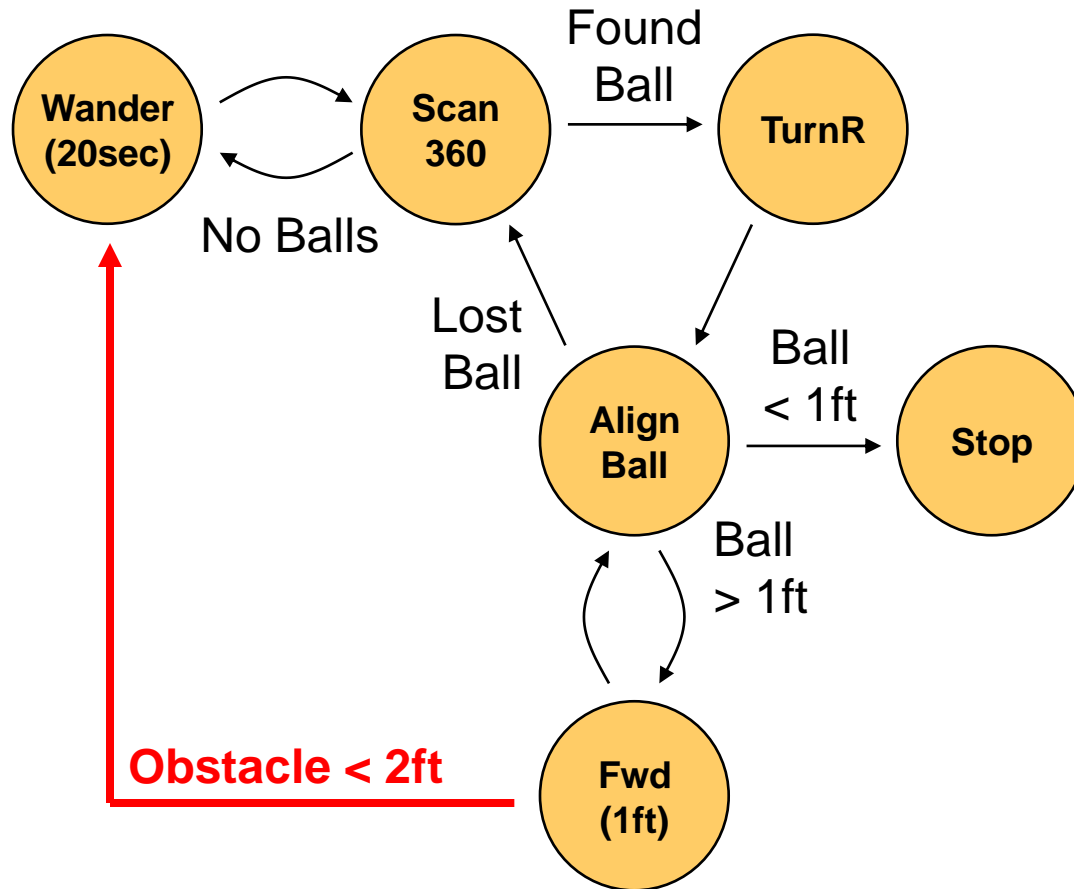


Can also fold closed loop feedback into the behaviors themselves

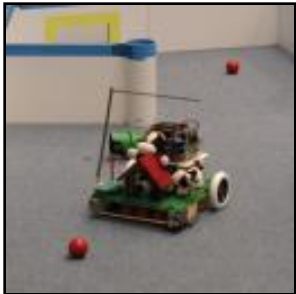
Simple finite state machine to locate red balls



Simple finite state machine to locate red balls



Outline

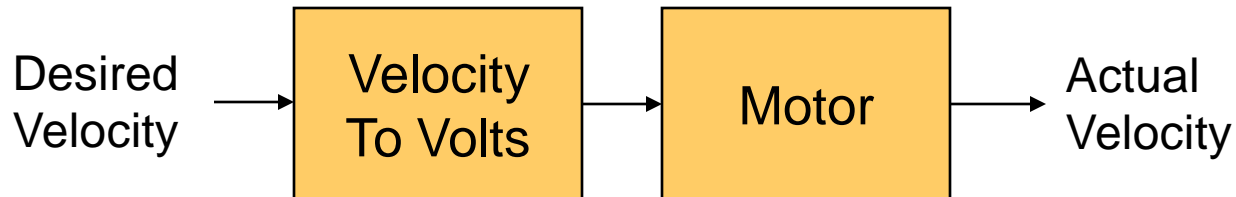
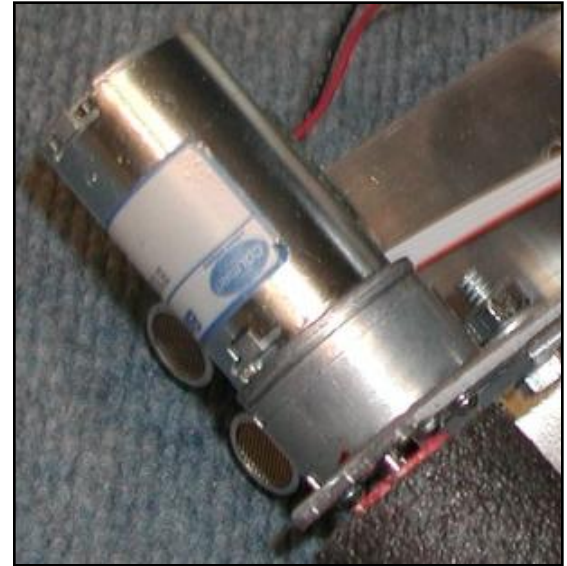


- High-level control system paradigms
 - Model-Plan-Act Approach
 - Behavioral Approach
 - Finite State Machine Approach
- **Low-level control loops**
 - **PID controller for motor velocity**
 - **PID controller for robot drive system**

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

Open Loop Controller

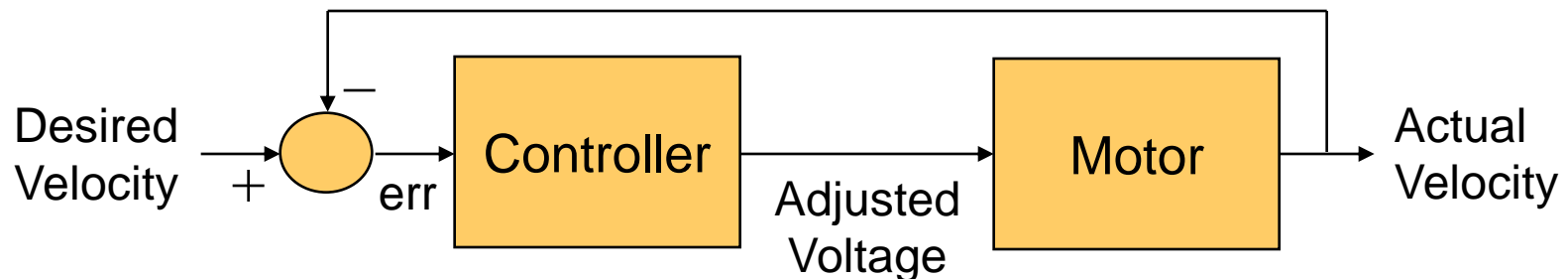
- Use trial and error to create some kind of relationship between velocity and voltage
- Changing supply voltage or drive surface could result in incorrect velocity



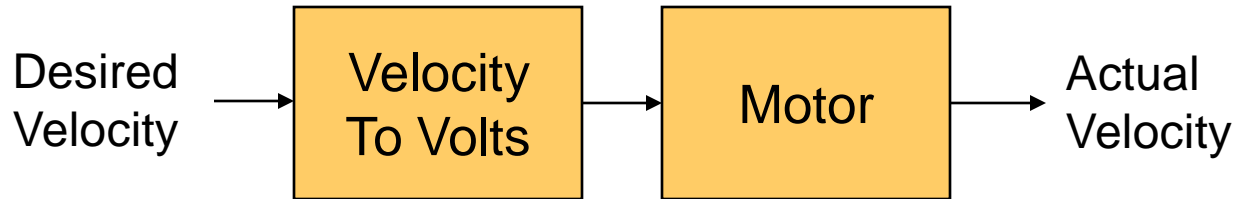
Problem: How do we set a motor to a given velocity?

Closed Loop Controller

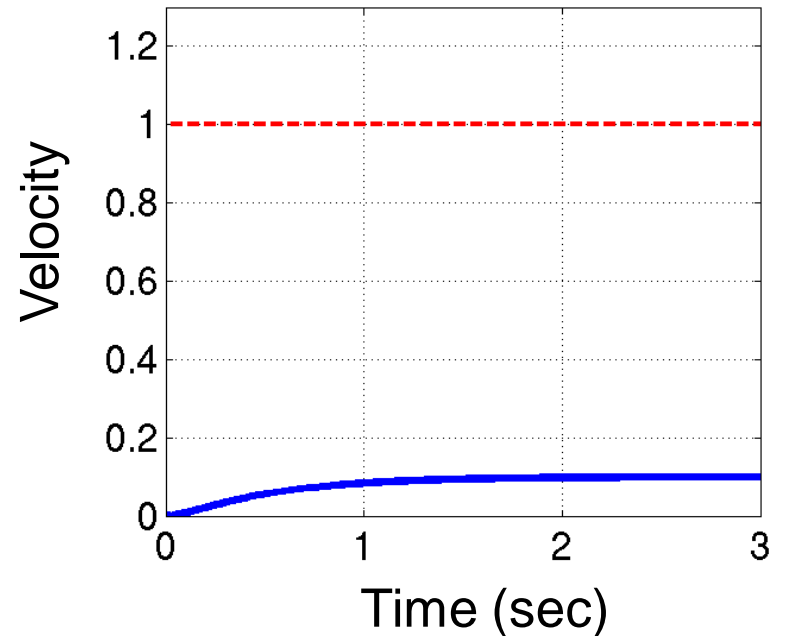
- Feedback is used to adjust the voltage sent to the motor so that the actual velocity equals the desired velocity
- Can use an optical encoder to measure actual velocity



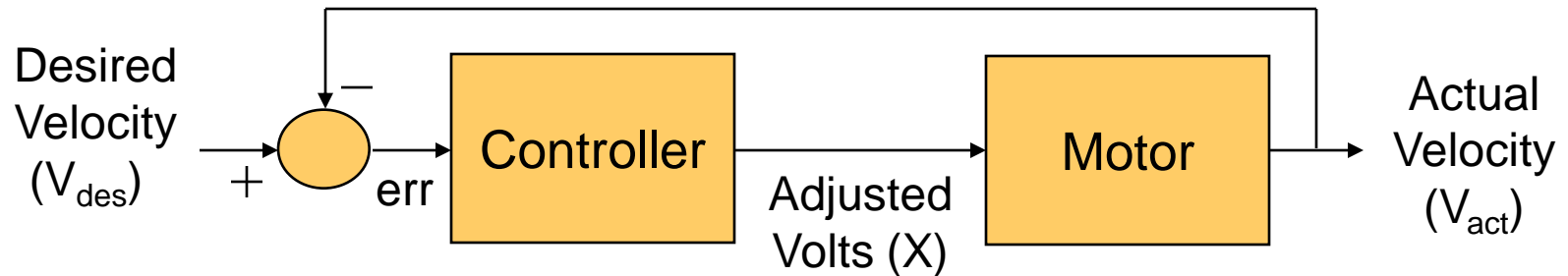
Step response with **no controller**



- Slow rise time
- Stead-state offset

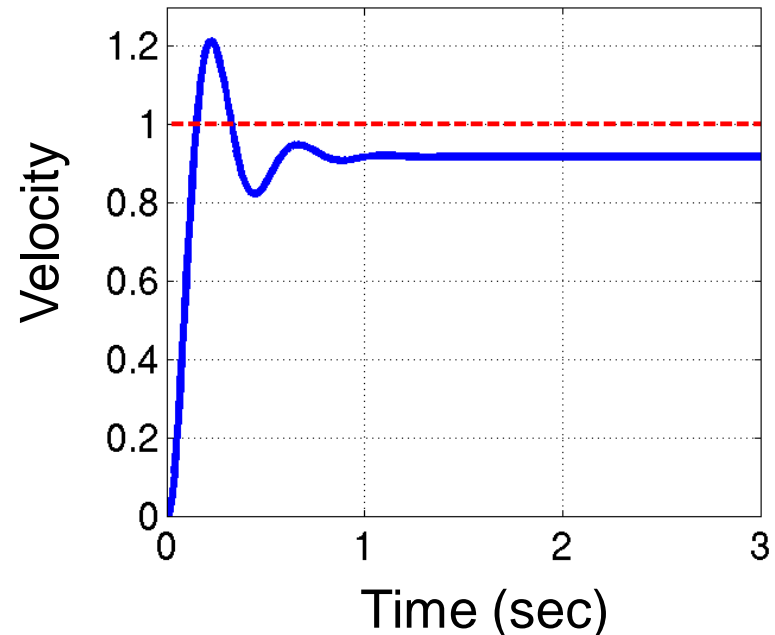


Step response with **proportional controller**

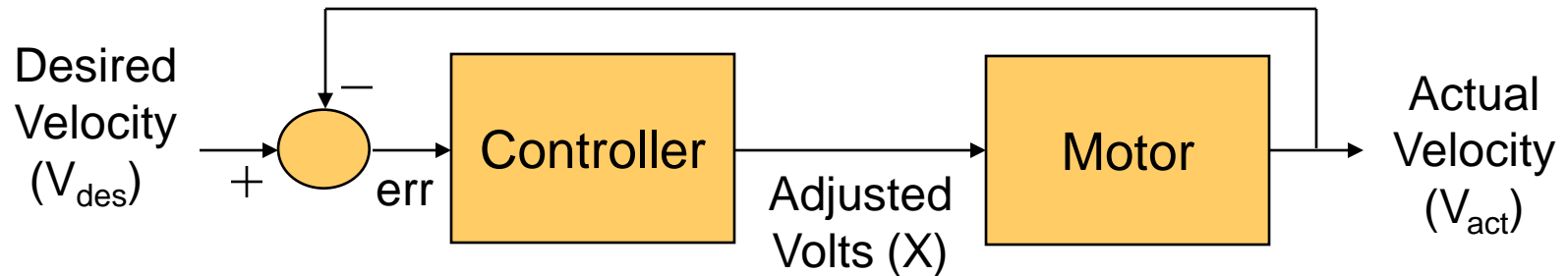


$$X = V_{des} + K_P \cdot (V_{des} - V_{act})$$

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

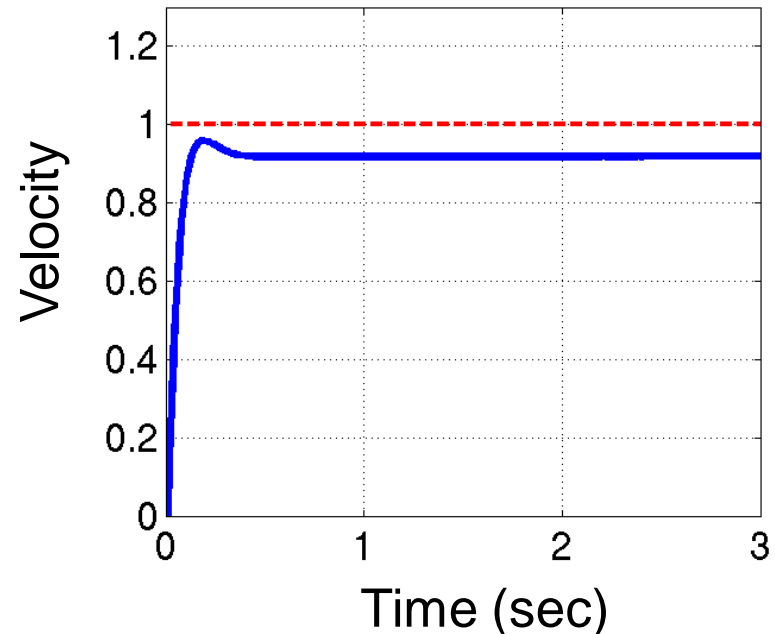


Step response with proportional-derivative controller

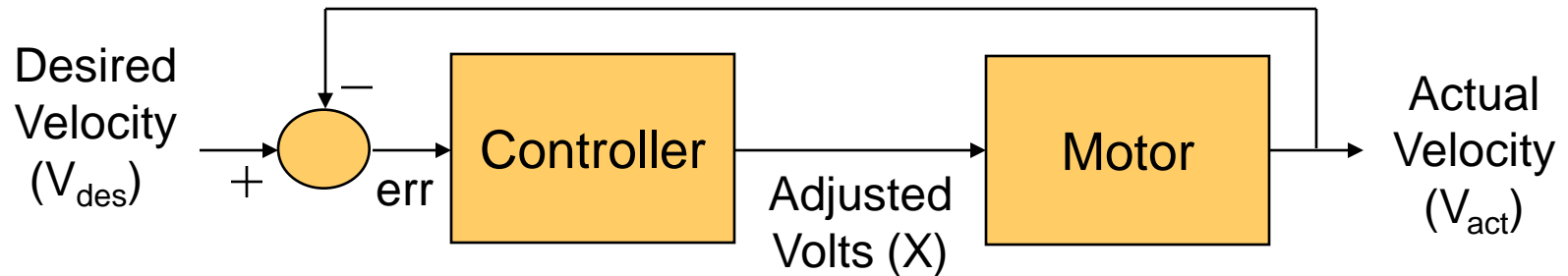


$$X = V_{des} + K_P e(t) - K_D \frac{de(t)}{dt}$$

- When approaching desired velocity quickly, de/dt term counteracts proportional term slowing adjustment
- Faster rise time (waktu naik lebih cepat)
- Reduces overshoot

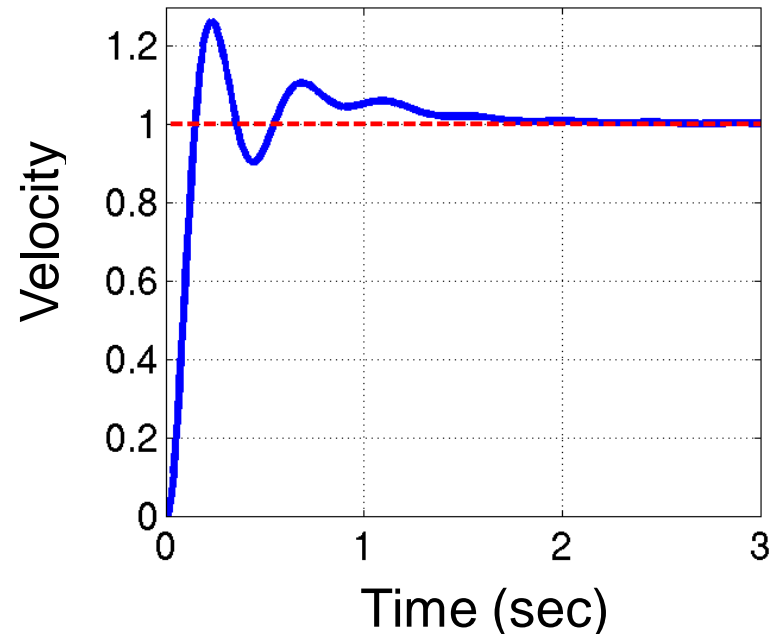


Step response with proportional-integral controller

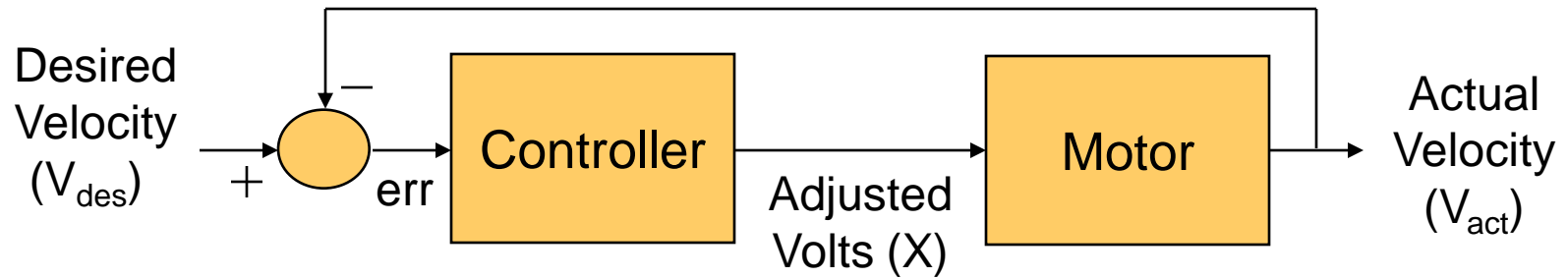


$$X = V_{des} + K_P e(t) - K_I \int e(t) dt$$

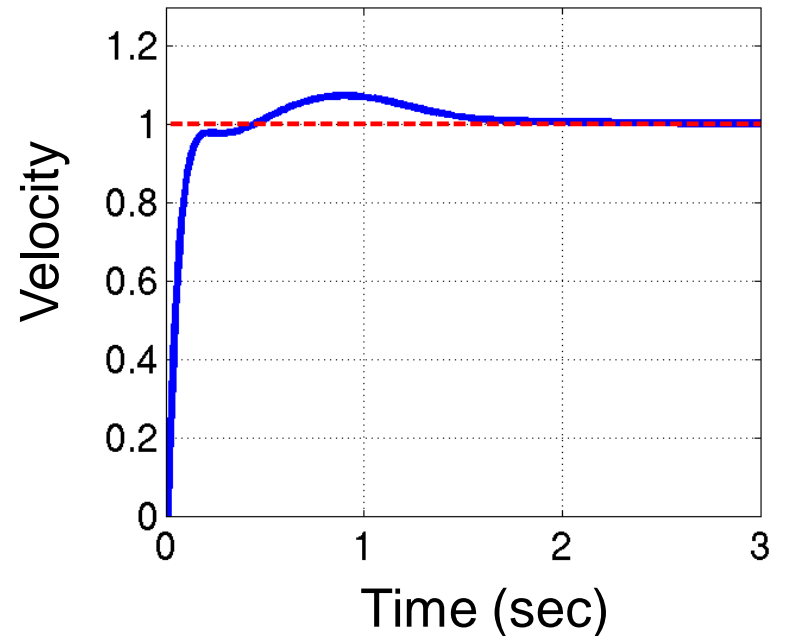
- Integral menghilangkan akumulasi error
- Meningkatkan overshoot



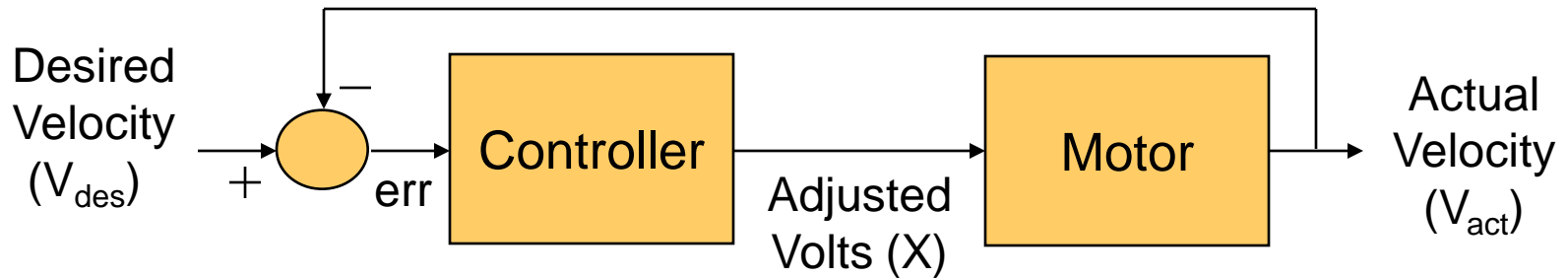
Step response with **PID** controller



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

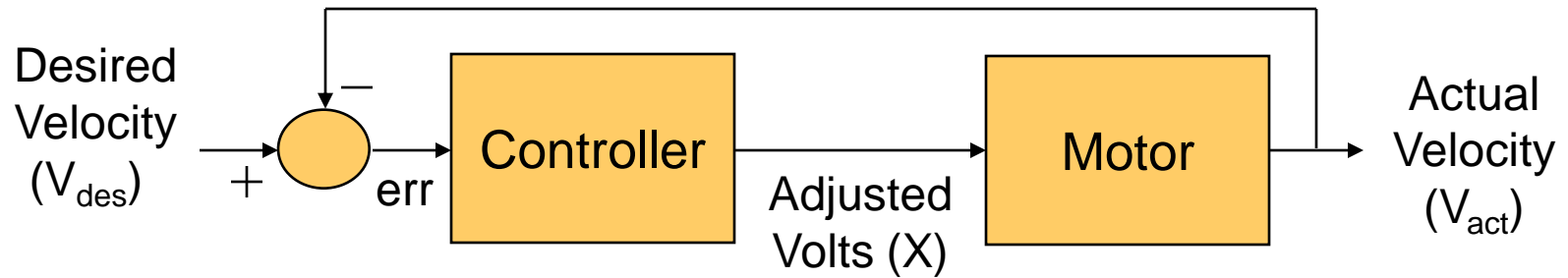


Choosing and tuning a controller



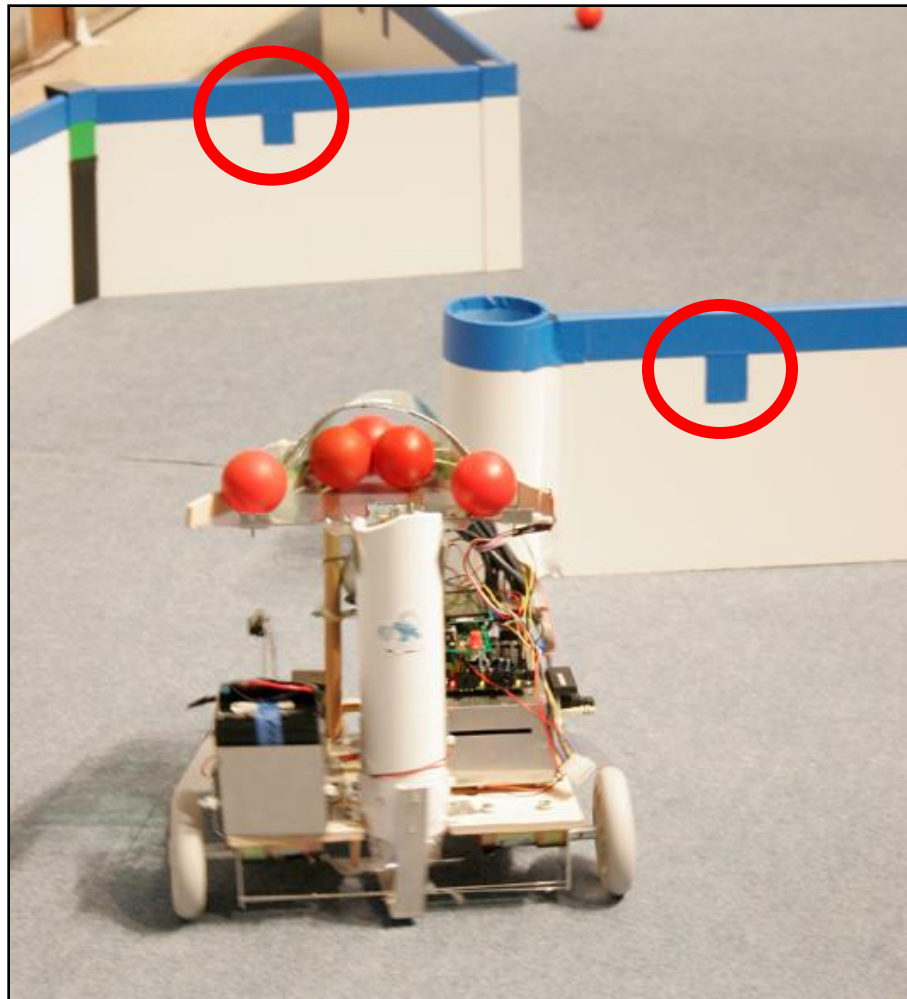
	Rise Time	Overshoot	Error
Proportional	Decrease	Increase	Decrease
Integral	Decrease	Increase	Eliminate
Derivative	~	Decrease	~

Choosing and tuning a controller



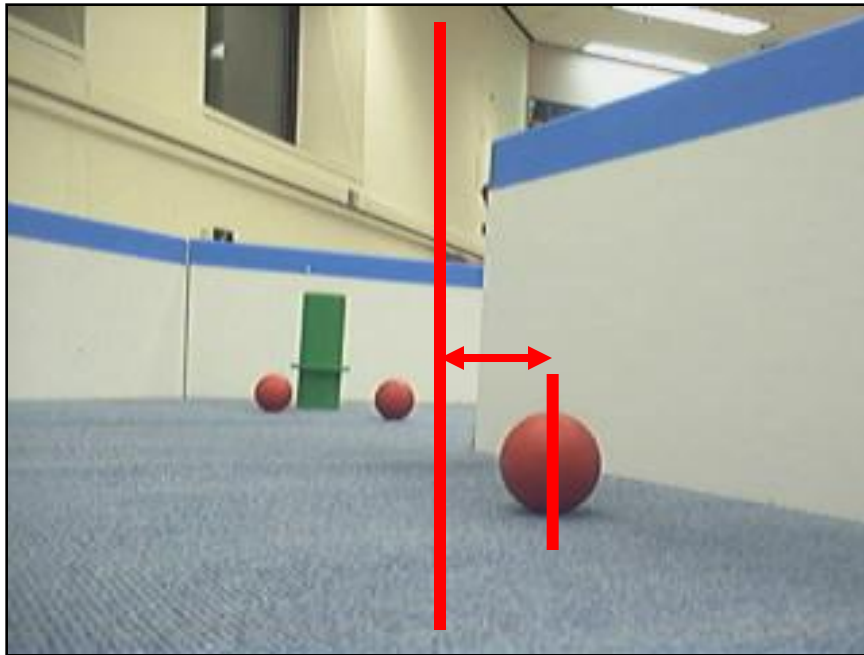
- Gunakan kontroler yg sederhana untuk mencapai yang diinginkan
- Tuning PID constants is very tricky, especially for integral constants

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!