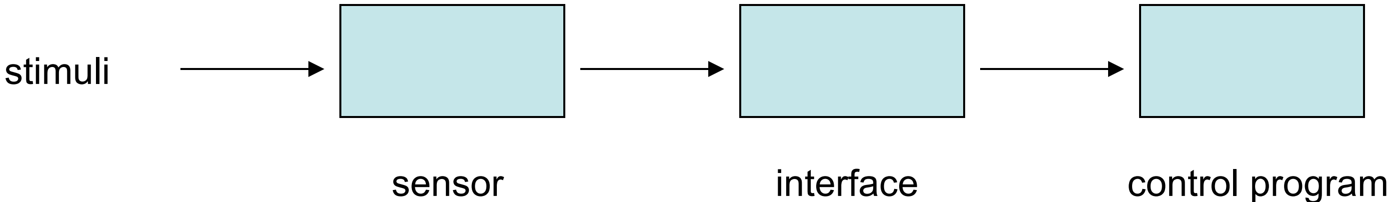
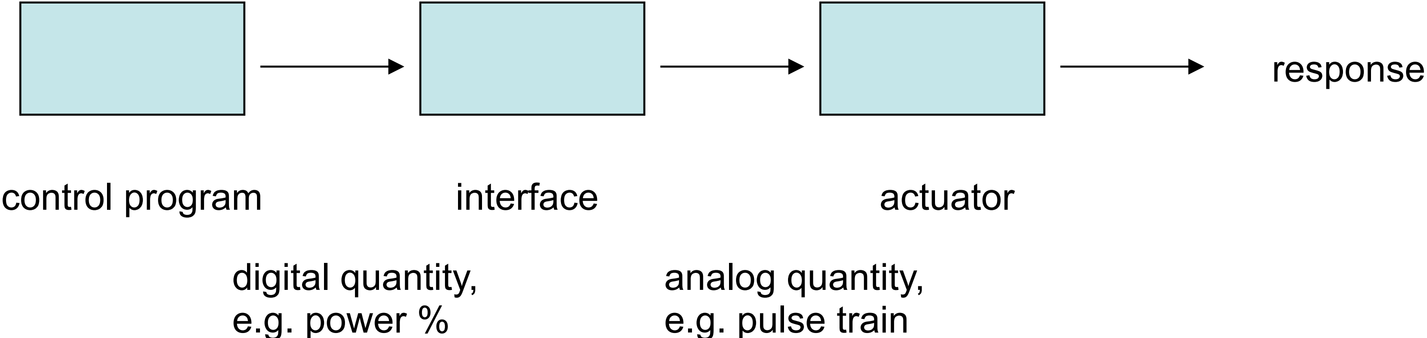
Sequential and Reactive Strategies

analog quantity,
e.g. light

digital quantity,
e.g. light %

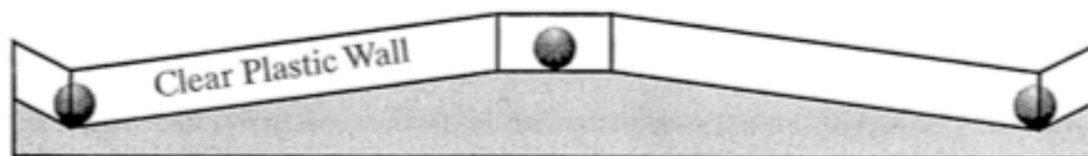




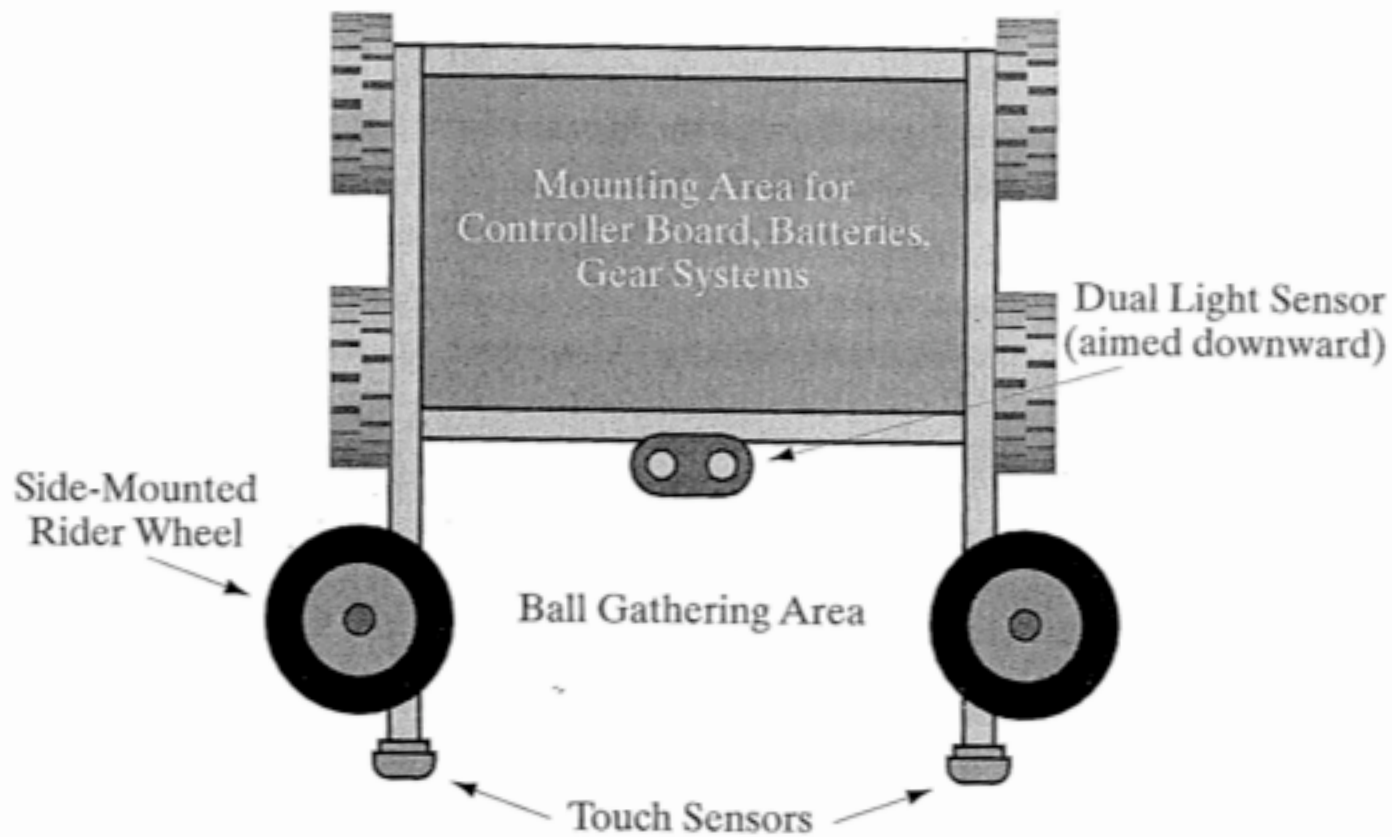
sensor space → actuator space

1991 MIT "Robo-Pong" Contest Table

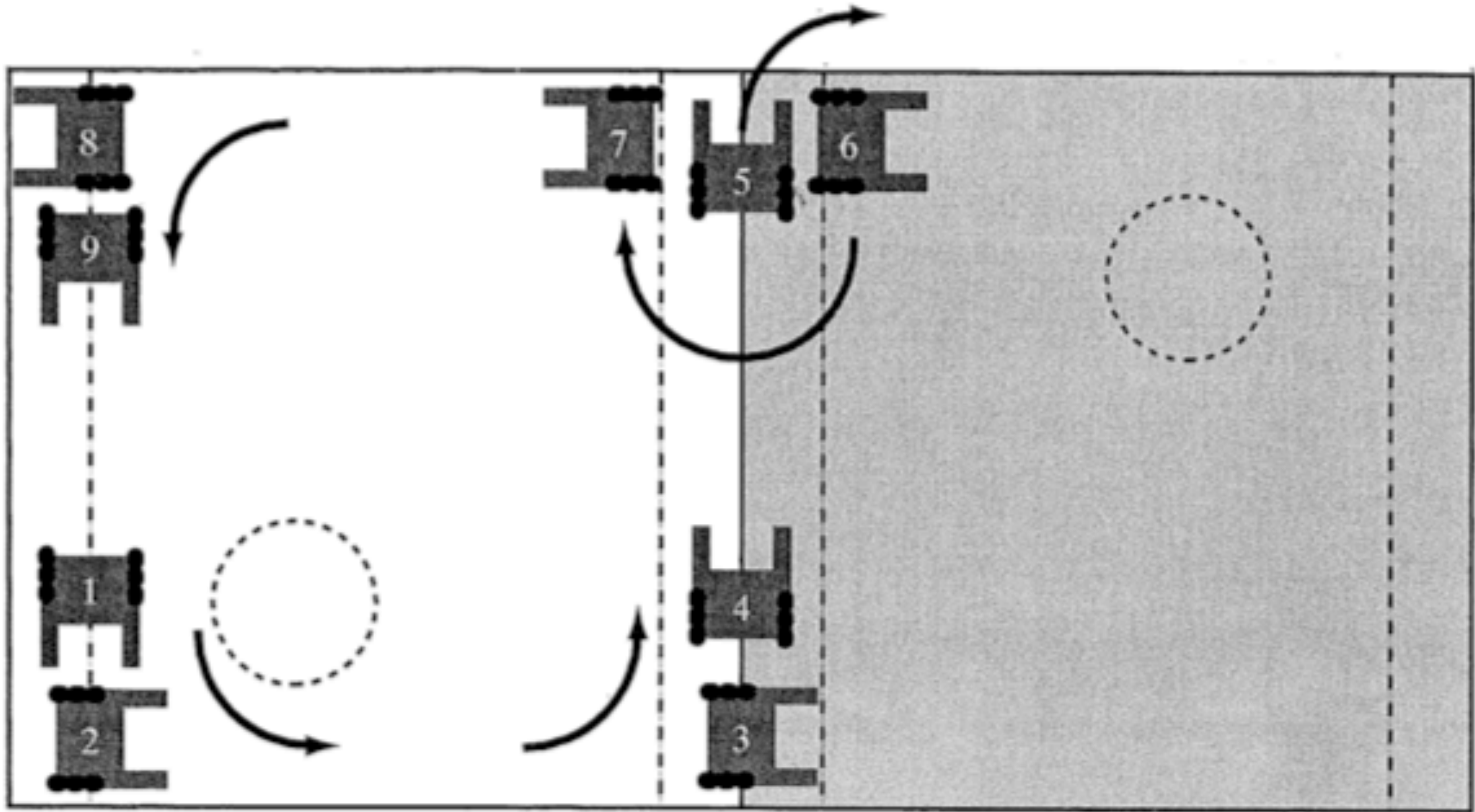
Aerial View



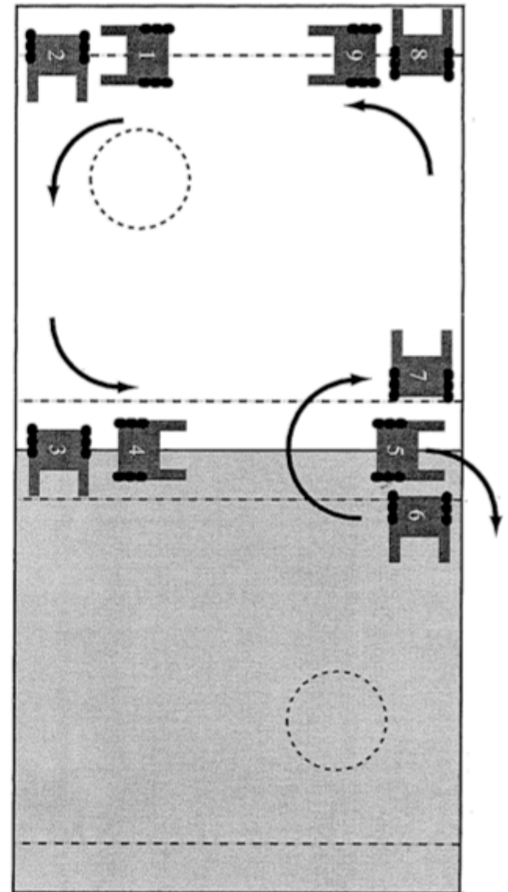
Side View


Mounting Area for
Controller Board, Batteries,
Gear Systems
Dual Light Sensor
(aimed downward)
Side-Mounted
Rider Wheel
Ball Gathering Area
Touch Sensors

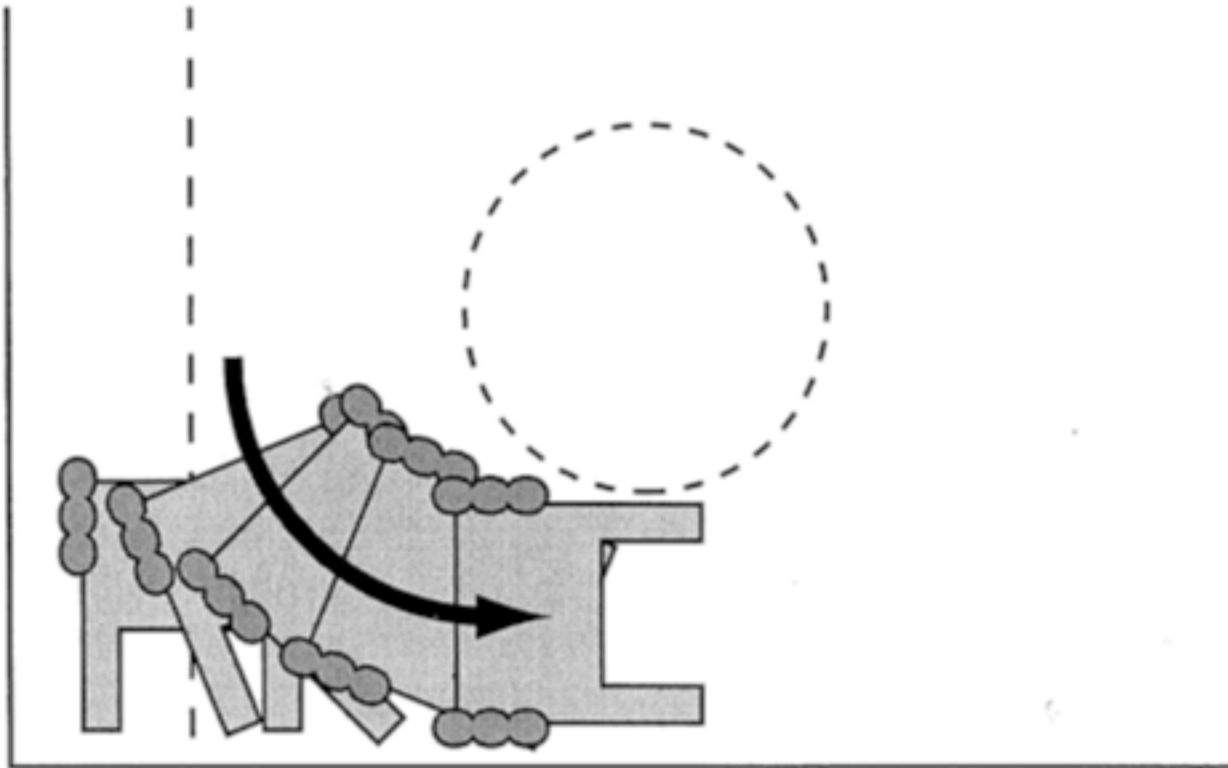
Sequential Strategy



```
void groucho() {

  while (1) {                    /* loop indefinitely */

    /* for simplicity, assume robot starts at position 1 */
    forward();
    waituntil_hit_wall();
    rotate_left_ninety();    /* now at position 2 */

    forward();
    waituntil_see_black();   /* position 3 */

    rotate_left_ninety();    /* position 4 */

    forward();
    waituntil_hit_wall();    /* position 5 */

    rotate_left_ninety();    /* position 6 */

    rotate_onehundred_eighty();  /* position 7 */

    forward();
    waituntil_hit_wall();    /* position 8 */

    rotate_left_ninety();    /* position 9 */
  }
}
```



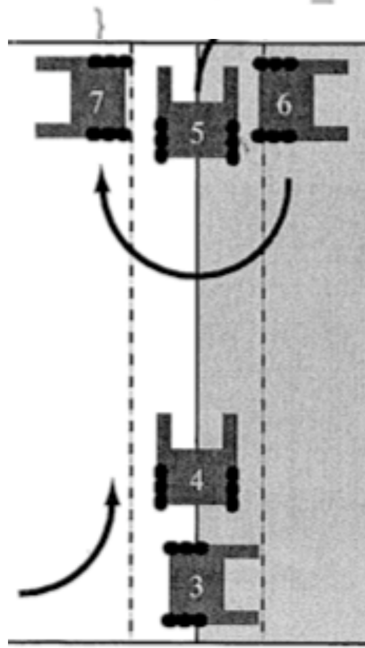
Open loop/ closed loop control



```
/* "eye" sensors return 1 if above light, 0 if above dark */
/* try to keep left on light, right on dark */

while (1)  {
  /* if left eye sees black, turn left */
  if (left_eye() == 0) veer_left();

  /* if right eye sees white, turn right */
  else if (right_eye() == 1) veer_right();

  /* otherwise, go straight */
  else forward();

  /* check for touch sensors */
  if (left_touch() || right_touch()) break;
}
```

closed loop

exit condition



Detect when something goes wrong

```
/* declare and initialize timeout variable */
long timeout= mseconds() + 4000L;

while (1)  {
  if (left_eye() == 0) veer_left();
  else if (right_eye() == 1) veer_right();
  else forward();

  if (left_touch() || right_touch()) break;


  /* check for timeout */
  if (mseconds() > timeout) break;
}
```

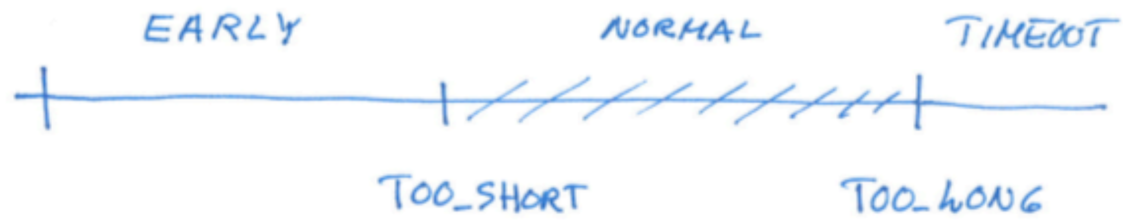
4 sec timeout limit

→ "wall"

→ timeout

```
/* define exit codes */
int NORMAL= 0;
int TIMEOUT= 1;

int follow_edge_to_wall()  {

  long timeout= mseconds() + 4000L;

  while (1)  {
    if (left_eye() == 0) veer_left();
    else if (right_eye() == 1) veer_right();
    else forward();

    if (left_touch() || right_touch()) return NORMAL;

    if (mseconds() > timeout) return TIMEOUT;
  }
}
```

```
/* define exit codes */
int NORMAL= 0;
int TIMEOUT= 1;
int EARLY= 2;

/* sample timing parameters */
long TOO_LONG= 4000L;
long TOO_SHORT= 1500L;

int follow_edge_to_wall()  {

  long start= mseconds();
  long timeout= start + TOO_LONG;

  while (1)  {
    if (left_eye() == 0) veer_left();
    else if (right_eye() == 1) veer_right();

    else forward();

    if (left_touch() || right_touch())
      if (mseconds() < (start + TOO_SHORT))
        return EARLY;
      else return NORMAL;

    if (mseconds() > timeout) return TIMEOUT;
  }
}
```



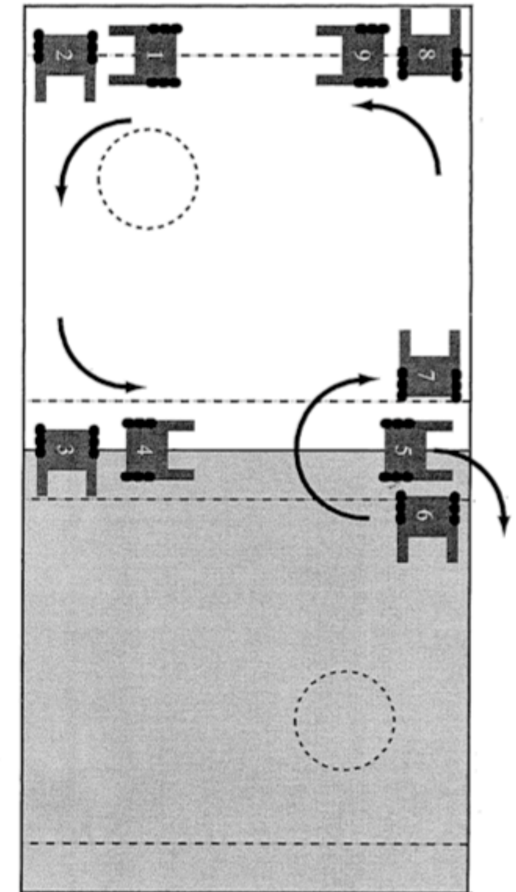
Timeout in all infinite loops

```
/* define exit codes */
int NORMAL= 0;
int TIMEOUT= 1;
int EARLY= 2;
int VL_STUCK= 3;
int VR_STUCK= 4;
int GS_STUCK= 5;

/* sample timing parameters */
long TOO_LONG= 4000L;
long TOO_SHORT= 1500L;
long VL_TIME= 2000L;
long VR_TIME= 2000L;
long GS_TIME= 3000L;

int follow_edge_to_wall()  {

  long start= mseconds();
  long timeout= start + TOO_LONG;
  int last_mode= 0;
  long last_time= 0;
```

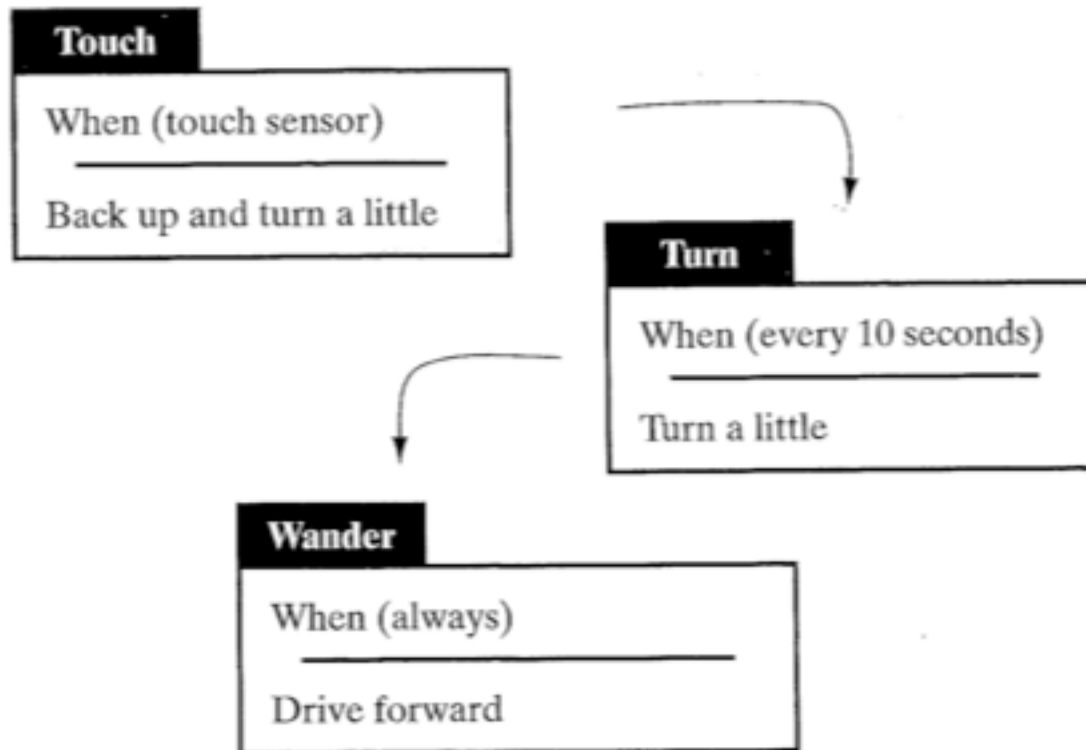
```
while (1)  {
  if (left_eye() == 0)  {
    veer_left();
    if (last_mode == VL_STUCK)
      if ((mseconds() - last_time) > VL_TIME)
        return VL_STUCK;
    else  {
      last_mode= VL_STUCK;
      last_time= mseconds();
    }
  }

  else if (right_eye() == 1) {
    veer_right();
    if (last_mode == VR_STUCK)
      if ((mseconds() - last_time) > VR_TIME)
        return VR_STUCK;
    else  {
      last_mode= VR_STUCK;
      last_time= mseconds();
    }
  }

  else {
    forward();
    if (last_mode == GS_STUCK)
      if ((mseconds() - last_time) > GS_TIME)
        return GS_STUCK;
    else  {
      last_mode= GS_STUCK;
      last_time= mseconds();
    }
  }

  if (left_touch() || right_touch())
    if (mseconds() < (start + TOO_SHORT))
      return EARLY;
    else return NORMAL;

  if (mseconds() > timeout) return TIMEOUT;
}
```

Sequential strategy

```
void groucho() {

  while (1) {                    /* loop indefinitely */

    /* for simplicity, assume robot starts at position 1 */
    forward();
    waituntil_hit_wall();
    rotate_left_ninety();   /* now at position 2 */

    forward();
    waituntil_see_black();  /* position 3 */

    rotate_left_ninety();   /* position 4 */

    forward();
    waituntil_hit_wall();   /* position 5 */

    rotate_left_ninety();   /* position 6 */

    rotate_onehundred_eighty();  /* position 7 */

    forward();
    waituntil_hit_wall();   /* position 8 */

    rotate_left_ninety();   /* position 9 */
  }
}
```



Reactive Strategy



```java
public static void main(String [] args)  throws Exception {

    int touchThreshold = 500;
    TouchSensor touch = new TouchSensor(SensorPort.S1);
    int turnInterval = 10000; // 10 sec in msec
    int lastTurnTime;
    int power = 70;

    LCD.drawString("Simple car", 0, 0);
    Button.waitForAnyPress();

    lastTurnTime = (int)System.currentTimeMillis();

    while ( ! Button.ESCAPE.isDown())
    {
        /* Touch */
        if ( SensorPort.S1.readRawValue() < touchThreshold )
        {
            /* Back up and turn a little */
            LCD.drawString("Back up and turn ", 0, 1);
            Car.backward(power, power);
            Thread.sleep(2000);
            Car.forward(0, power);
            Thread.sleep(1000);
            LCD.drawString("                 ", 0, 1);
        }
        else
        /* Turn */
        if ( lastTurnTime + turnInterval < (int)System.currentTimeMillis() )
        {
            /* Turn a little */
            LCD.drawString("Turn", 0, 2);
            lastTurnTime = (int)System.currentTimeMillis();
            Car.forward(0, power);
            Thread.sleep((int)(Math.random()*2000));
            LCD.drawString("    ", 0, 2);
        }
        else
        /* Wander */
        {
            /*  Drive forward */
            LCD.drawString("Forward", 0, 3);
            Car.forward(power, power);
        }

    }
}
```

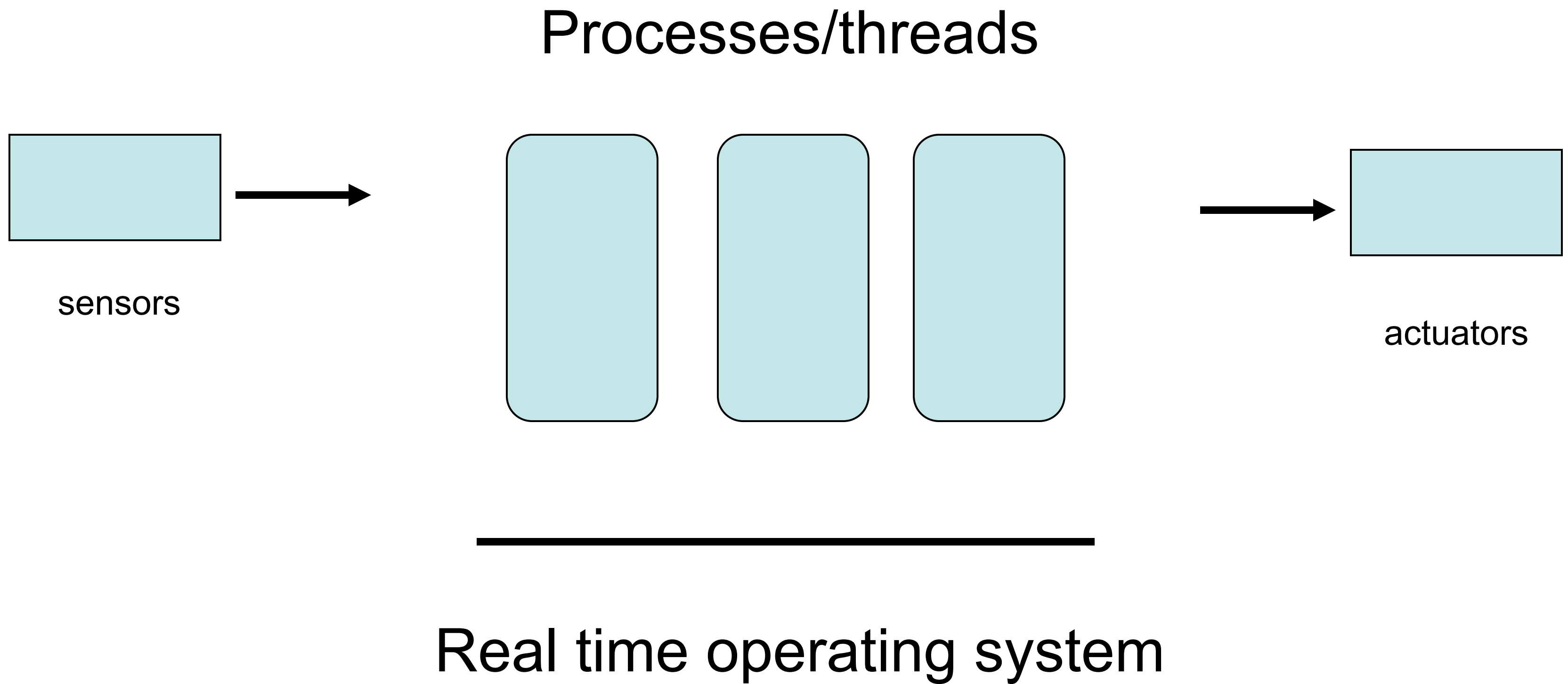
```java
public static void main(String [] args)  throws Exception {

    int touchThreshold = 500;
    TouchSensor touch = new TouchSensor(SensorPort.S1);
    int turnInterval = 10000; // 10 sec in msec
    int lastTurnTime;
    int power = 70;

    LCD.drawString("Simple car", 0, 0);
    Button.waitForAnyPress();

    lastTurnTime = (int)System.currentTimeMillis();

    while ( ! Button.ESCAPE.isDown())
    {
        /* Touch */
        if ( SensorPort.S1.readRawValue() < touchThreshold )
        {
            /* Back up and turn a little */
            LCD.drawString("Back up and turn ", 0, 1);
            Car.backward(power, power);
            Thread.sleep(2000);
            Car.forward(0, power);
            Thread.sleep(1000);
            LCD.drawString("                 ", 0, 1);
        }
        else
        /* Turn */
        if ( lastTurnTime + turnInterval < (int)System.currentTimeMillis() )
        {
            /* Turn a little */
            LCD.drawString("Turn", 0, 2);
            lastTurnTime = (int)System.currentTimeMillis();
            Car.forward(0, power);
            Thread.sleep((int)(Math.random()*2000));
            LCD.drawString("    ", 0, 2);
        }
        else
        /* Wander */
        {
            /*  Drive forward */
            LCD.drawString("Forward", 0, 3);
            Car.forward(power, power);
        }

    }
}
```

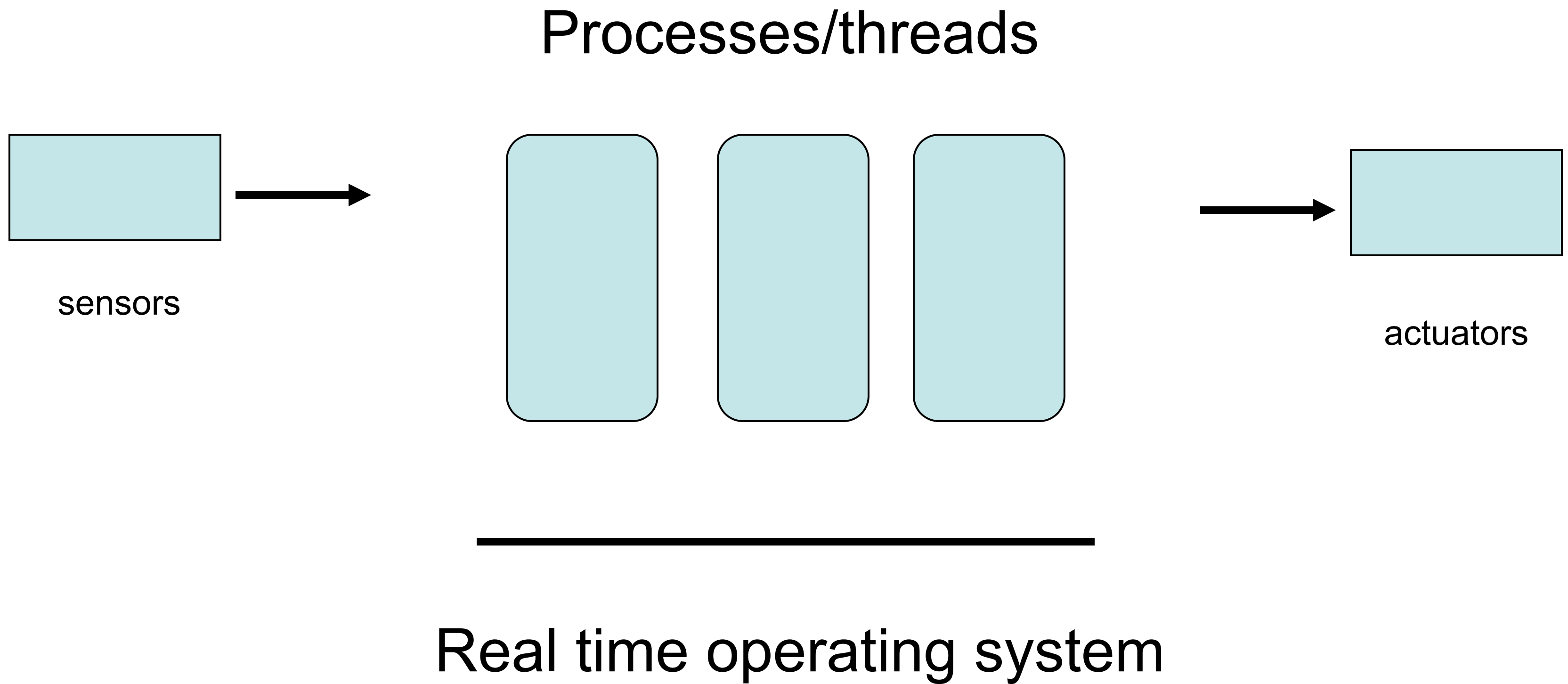
blocked

blocked

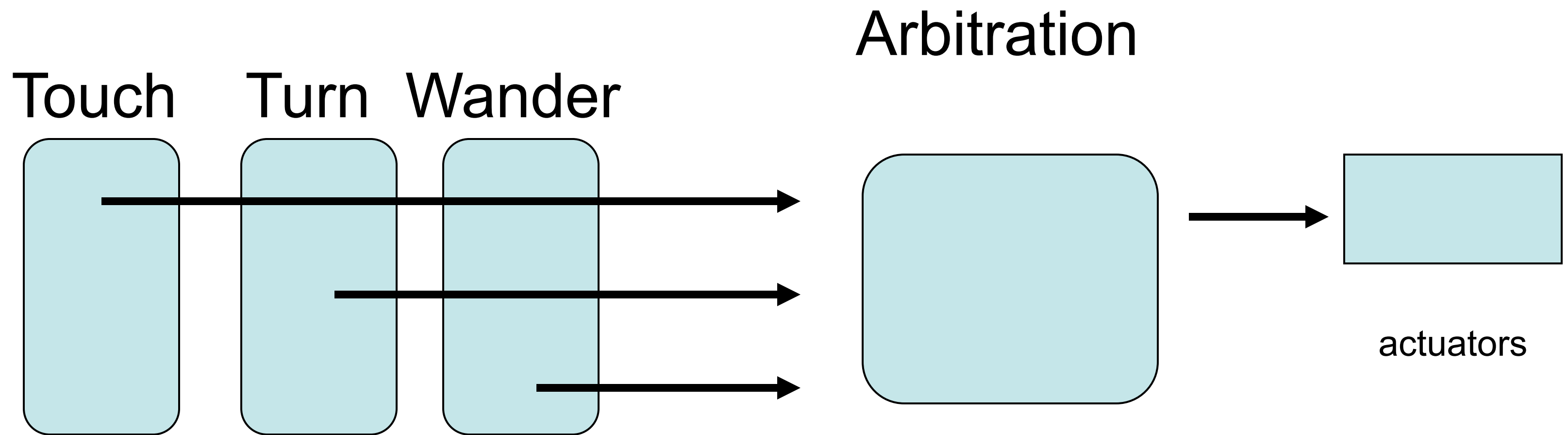
Each control loop can be programmed as a process

```
/* generic touch sensor program */
void touch () {
  while (1) {
    if (left_touch()) {
      backward(); msleep(500L);
      right(); msleep(500L);
    } else if (right_touch()) {
      backward(); msleep(500L);
      left(); msleep(500L);
    }
  }
}
```


Processes/threads
sensors
actuators
Real time operating system



Common resources?



Real time operating system

Embedded Java

The leJOS API classes provide real time events, sensor events and threads:

TimerListener

SensorListener

Thread

Timer

```
public interface TimerListener
```

Listener used with Timer.

Author:

Ryan VanderBijl

See Also:

`Timer`

Method Summary

`void`	**`timedOut`**`()` Called every time the Timer fires.

Method Detail

timedOut

```
void timedOut()
```

Called every time the Timer fires.

lejos.nxt

Interface SensorPortListener

```
public interface SensorPortListener
```

Interface for monitoring changes to the value for an Analogue/Digital sensor (such as a Touch, Light or Sound sensor) on a SensorPort.

Method Summary

void	**stateChanged**(SensorPort aSource, int aOldValue, int aNewValue) Called when the raw value of the sensor attached to the port changes.

Method Detail

stateChanged

```
void stateChanged(SensorPort aSource,
                  int aOldValue,
                  int aNewValue)
```

Called when the raw value of the sensor attached to the port changes.

Parameters:

`aSource` - The Port that generated the event.
`aOldValue` - The old sensor raw value.
`aNewValue` - The new sensor raw value.

```java
import lejos.nxt.*;
import lejos.util.Timer;
import lejos.util.TimerListener;
/*
 * Fred Martins simple reactive car from chapter 5
 */
public class SimpleCar2 {

   public static void main(String [] args)  {

       int turnInterval = 10000; // 10 sec in msec
       final int power = 70;

       LCD.drawString("Simple car 2", 0, 0);
       Button.waitForAnyPress();

       SensorPort.S1.addSensorPortListener(new SensorPortListener()
       {
           private int touchThreshold = 500;

           public void stateChanged(SensorPort port, int value, int oldValue)
           {
               if (port == SensorPort.S1 )
               {
                   /* Touch */
                   if (value < touchThreshold & oldValue > touchThreshold)
                   {
                       /* Back up and turn a little */
                       LCD.drawString("Back up and turn", 0, 1);
                       Car.backward(power, power);
                       try { Thread.sleep(2000); }
                       catch (Exception e){}

                       Car.forward(0, power);
                       try { Thread.sleep(1000); }
                       catch (Exception e){}
                       LCD.drawString("                ", 0, 1);
                   }
               }
           }
       });
```

```
TimerListener tl = new TimerListener()
{
    public void timedOut()
    {
        /* Turn a little */
        LCD.drawString("Turn", 0, 2);
        Car.forward(0, power);
        try { Thread.sleep((int)(Math.random()*2000)); }
        catch (Exception e){}
        LCD.drawString("    ", 0, 2);

    }
};
Timer timer = new Timer(turnInterval,tl);
timer.start();

while ( ! Button.ESCAPE.isDown())
{
    /*  Drive forward */
    LCD.drawString("Forward", 0, 3);
    Car.forward(power, power);
}

timer.stop();
```

java.lang

Class Thread

java.lang.Object
  └ **java.lang.Thread**

All Implemented Interfaces:

Runnable

Direct Known Subclasses:

Keyboard, LCPResponder, NavPathController.Nav, NXTRegulatedMotor.Controller, RConsole,

---

```
public class Thread
extends Object
implements Runnable
```

A thread of execution (or task). Now handles priorities, daemon threads and interruptions.

Field Summary	
static int	**MAX_PRIORITY** The maximum priority that a thread can have.
static int	**MIN_PRIORITY** The minimum priority that a thread can have.
static int	**NORM_PRIORITY** The priority that is assigned to the primordial thread.

Method Summary

static Thread	currentThread()
String	getName() Returns the string name of this thread.
int	getPriority()
void	interrupt() Set the interrupted flag.
static boolean	interrupted()
boolean	isAlive()
boolean	isDaemon() Set the daemon flag.
boolean	isInterrupted()
void	join() Join not yet implemented
void	join(long timeout)

void	**run**() When an object implementing interface `Runnable` is used to create a thread, starting the thread causes the object's `run` method to be called in that separately executing thread.
void	**setDaemon**(boolean on)
void	**setName**(String name) Sets the string name associated with this thread.
void	**setPriority**(int priority) Set the priority of this thread.
static void	**sleep**(long aMilliseconds)
void	**start**()
static void	**yield**()

setPriority

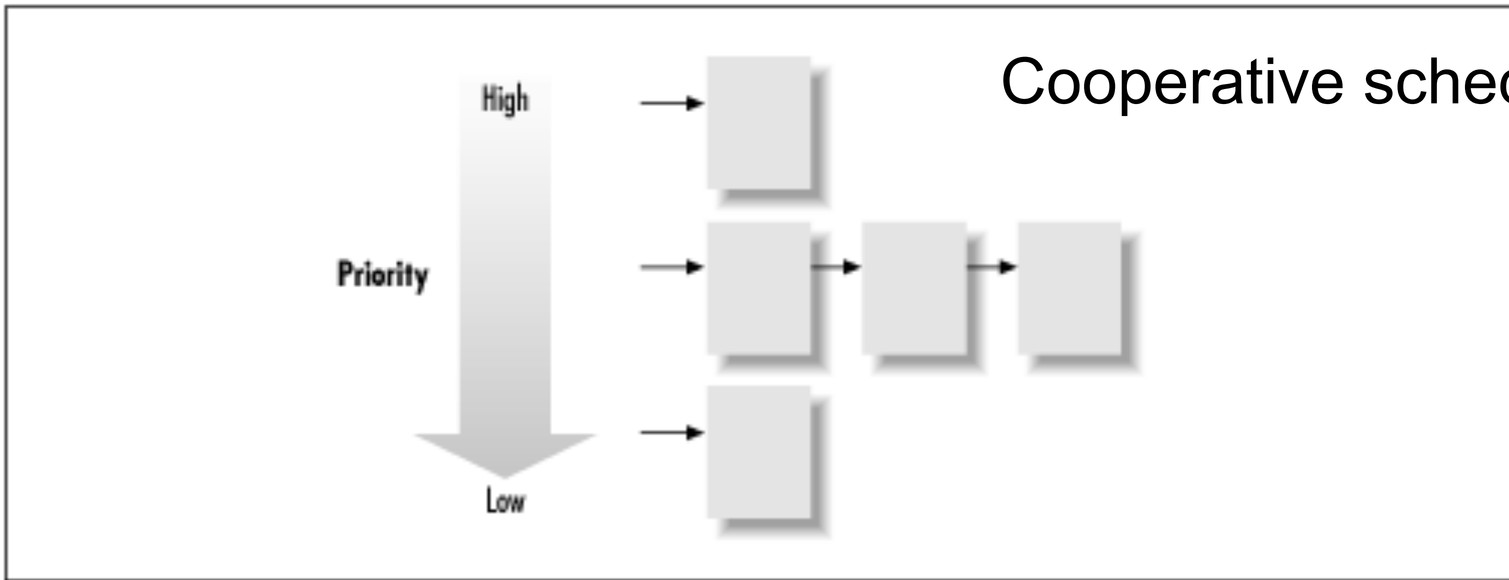
```
public final void setPriority(int priority)
```

Set the priority of this thread. Higher number have higher priority. The scheduler will always run the highest priority thread in preference to any others. If more than one thread of that priority exists the scheduler will time-slice them. In order for lower priority threas to run a higher priority thread must cease to be runnable. i.e. it must exit, sleep or wait on a monitor. It is not sufficient to just yield.

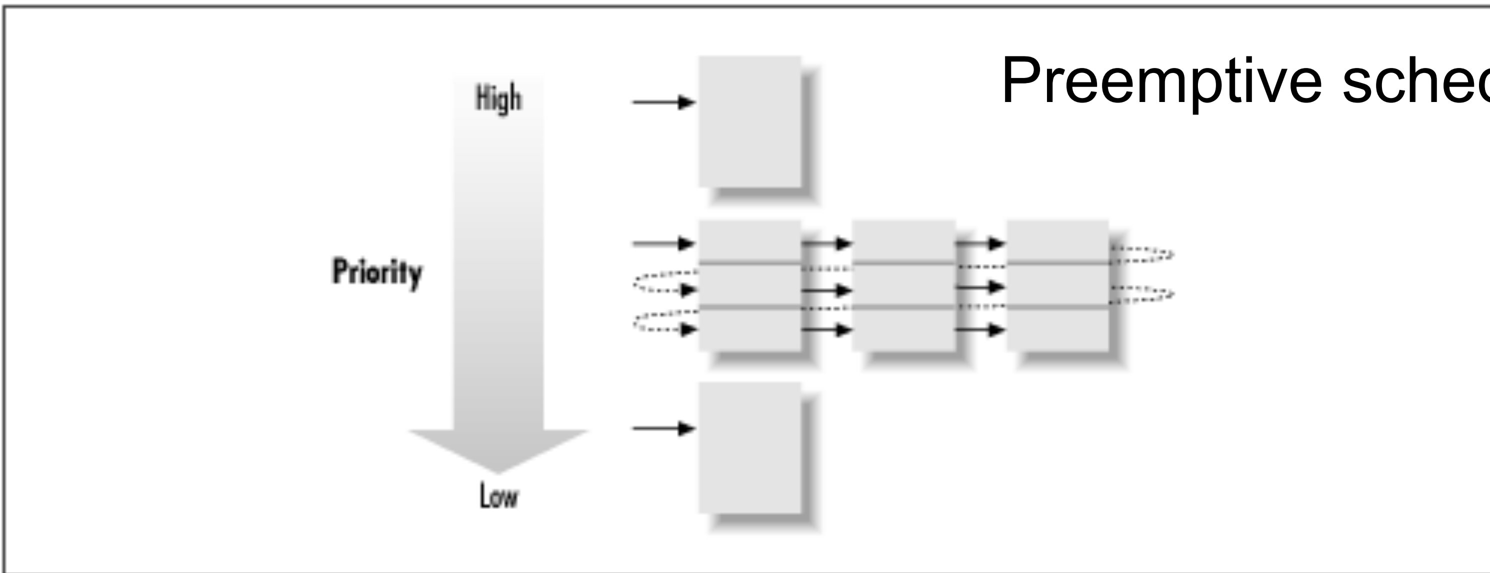
Threads inherit the priority of their parent. The primordial thread has priority NORM_PRIORITY.

Parameters:

`priority` - must be between MIN_PRIORITY and MAX_PRIORITY.

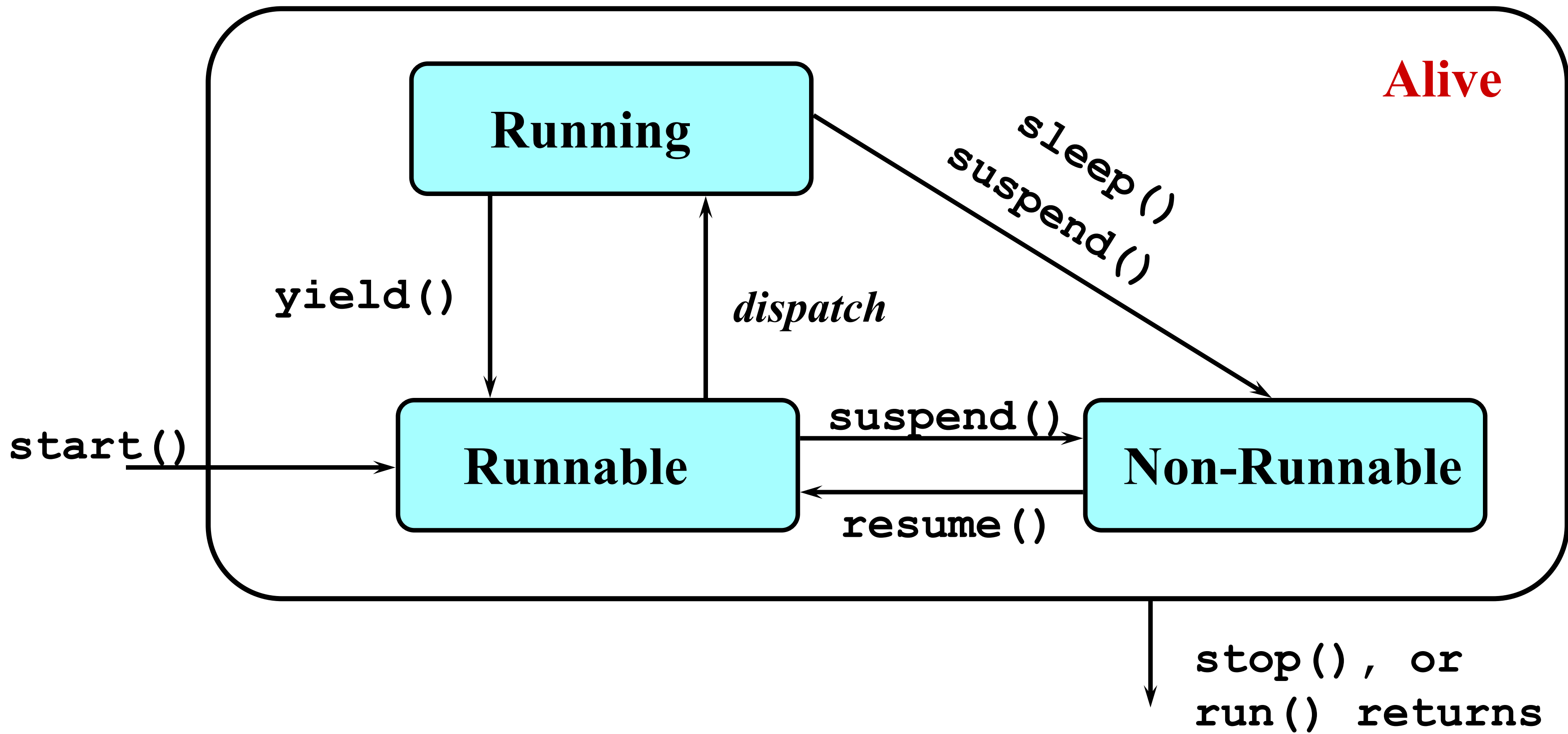


Cooperative scheduling



Preemptive scheduling

a simple round robin 2ms time slice for threads of equal priority...


Alive
Running
sleep()
suspend()
yield()
dispatch
start()
Runnable
suspend()
resume()
Non-Runnable
stop(), or
run() returns

```java
/**
 * Timer object, with some similar functionality to java.Swing.Timer.
 *
 * @author <a href="mailto:rvbijl39<at>calvin<dot>edu">Ryan VanderBijl</a>
 */
public class Timer
{
    private TimerListener myListener;
    private Thread        myThread  ;
    private int           delay     ;
    private boolean       running   ;

    /**
     * Create a Timer object. Every theDelay milliseconds
     * the el.timedOut() function is called. You may
     * change the delay with setDelay(int). You need
     * to call start() explicitly.
     */
    public Timer(int theDelay, TimerListener el)
    {
    running    = false;
    delay      = theDelay;
    myListener = el;

    myThread   = new Thread() {
        public void run() {
        int     d;
        boolean r;
        while(true) {
            synchronized(Timer.this)
            {
                d = delay;
                r = running;
            }
            if (r)
            {
                try
                {
                  Thread.sleep (d);
                }
                catch (InterruptedException e)
                {
              // ignore
                }
                myListener.timedOut();
            } else {
                yield();
            }
        }
        }
    };

    myThread.setDaemon(true);
    }
```

```
/**
 * access how man milliseconds between timedOut() messages.
 */
public synchronized int getDelay() {
return delay;
}
/**
 * Change the delay between timedOut messages. Safe to call
 * while start()ed. Time in milli-seconds.
 */
public synchronized void setDelay(int newDelay) {
delay = newDelay;
}

/**
 * Stops the timer. timedOut() messages are not sent.
 */
public synchronized void stop() {
running = false;
}

/**
 * Starts the timer, telling it to send timeOut() methods
 * to the TimerListener.
 */
public synchronized void start() {
running = true;
if (!myThread.isAlive())
  myThread.start();
}
```

Line following

Black White Detection

First, you should mount the sensor on the LEGO 9797 car as described in LEGO Mindstorms Education NXT Base Set 9797 building instruction page 32 to page 34. Second, make a program that use and test the class `BlackWhiteSensor.java`. After calibration, place the car with the light sensor over different dark and bright areas and investigate how well the `BlackWhiteSensor` works.

Line Follower with Calibration

As an application of the `BlackWhiteSensor` try the program `LineFollowerCal.java`. The program uses the simple class `Car.java` to move the car.

ThreeColorSensor with Calibration

Use the idea and structure of the `BlackWhiteSensor` to program a slass `ThreeColorSensor` that can detect three colors: black, green and white. Make a test program that investigate the usefulness of the class.

Line Follower that stops in a Goal Zone

Use the `ThreeColorSensor` to make a line follower that stops in a green goal zone.

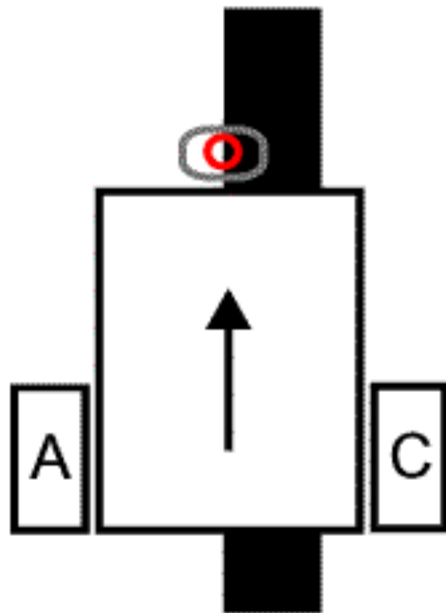
PID Line Follower

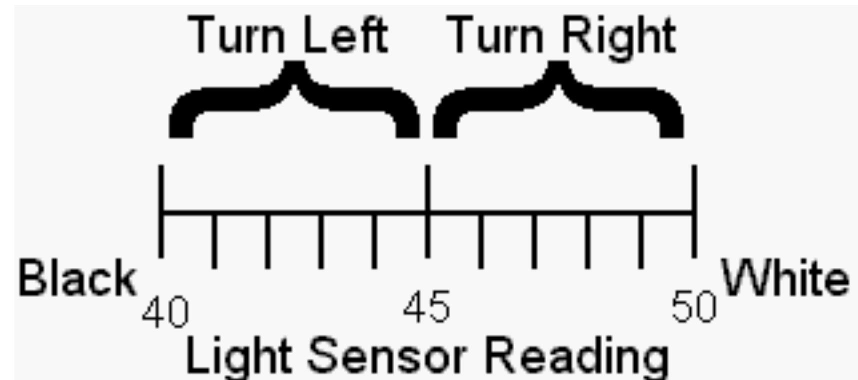
It is possible to make a line follower with just one light sensor that follows the line more smoothly and drive faster if a PID regulator is used, [1]. Try this.

Color Sensor

In the LEGO Mindstorms series there is also a color sensor. Use the test program `ColorSensorSensor.java` to investigate the information that the class `ColorSensor` provide in Full mode. Place the color sensor over black, white and green to figure out if the information provided can be used to distinguish the three colors.

A PID Controller For Lego Mindstorms Robots






```
Kp = 10                               ! Initialize our three variables
offset = 45
Tp = 50
Loop forever
   LightValue = read light sensor     ! what is the current light reading?
   error = LightValue - offset        ! calculate the error by subtracting the offset
   Turn = Kp * error                  ! the "P term", how much we want to change the motors' power
   powerA = Tp + Turn                 ! the power level for the A motor
   powerC = Tp - Turn                 ! the power level for the C motor
   MOTOR A direction=forward power=powerA ! issue the command with the new power level in a MOTOR block
   MOTOR C direction=forward power=powerC ! same for the other motor but using the other power level
end loop forever                      ! done with this loop, go back to the beginning and do it again
```

Color