

4C-Dog.vi
 Fiamma:Users:andy:Professional:UW Design:Art 483, Winter 2007:Robot Exercises:4C-Dog.vi
 Last modified on 12/13/06 at 8:48 PM
 Printed on 12/13/06 at 8:54 PM

The Dog [C]

- Port 1: touch sensor
- Port 2: light sensor
- Port 3: sound sensor
- Port 4: distance sensor

- Port A: motor
- Port B: lamp
- Port C: motor

This robot crudely models the behavior of a dog. Being a canine, it is a highly sentient, intelligent, and perceptive creature, of course. Our dog is a hunter and tracker.

This dog's mission is to find something ahead of it in the bush. Since we don't have smell sensors for this robot, instead we'll have him follow a visual trail on the ground—a black line on a light background.

The robot does this by using a light sensor pointed at the ground to evaluate reflected light and determine its intensity, and then steering itself to stay on the black line.

In addition to this activity, the robot is simultaneously doing two other things: stopping if it makes contact with anything, and listening for commands to pause and continue its search.

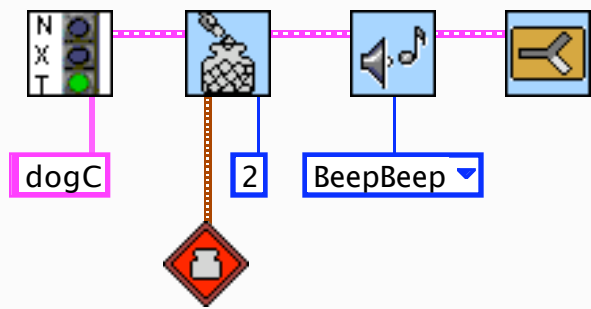
In this program, the robot has three separately executing tasks, operating simultaneously to accomplish each of the four goals. Each task continuously monitors one of the input sensors, and takes appropriate action when something is detected by that sensor.

In some cases, this requires communication between separate tasks, which is done by setting an indicator value in a container accessible to all tasks in the program.

The details of each task's methodology (*LIGHT*, *TOUCH*, *SOUND*, and *DISTANCE Monitors*) are explained in the boxes to the right.

Initialization

The *Initialization* is explained in the box to the right . ==>



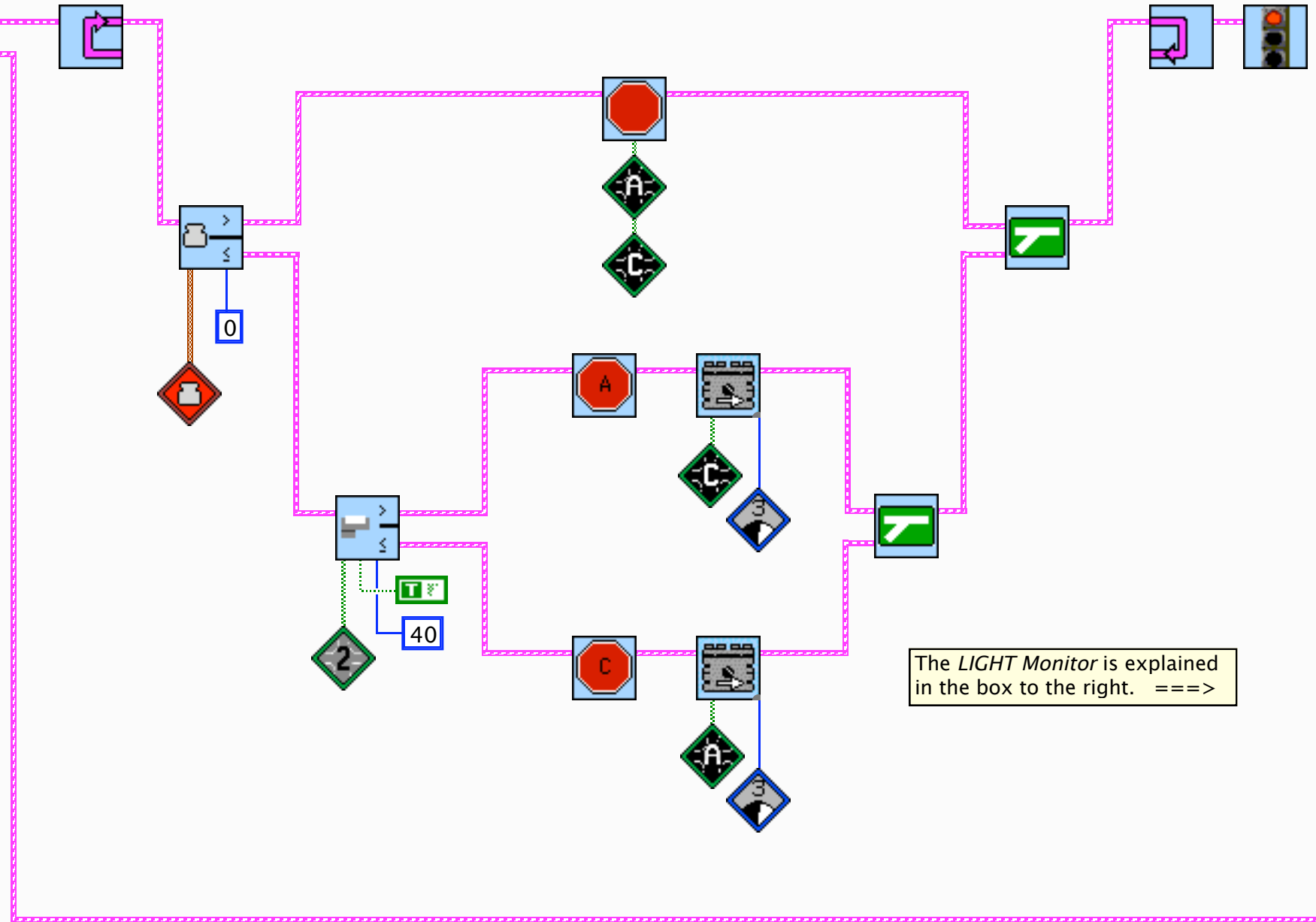
The two commands (with sideways "Y"-shaped icons) after the initialization commands serve to start the three tasks of the program operating simultaneously.

Each starts executing the commands from the topmost exit wire and then creates a new task (from the bottom command exit wire) that begins executing in parallel to the top sequence.

So the first task command starts the main program (the *TOUCH Monitor*) and then creates a second task. This second task immediately starts the *LIGHT Monitor* in parallel, and then starts a new task as the *SOUND Monitor*.

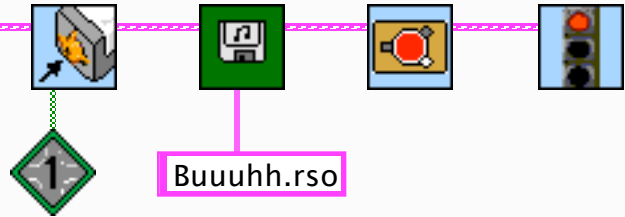
With this method of cascading task creation, we start three separate tasks almost instantaneously after the initialization of the program.

LIGHT Monitor



The *LIGHT Monitor* is explained in the box to the right. ==>

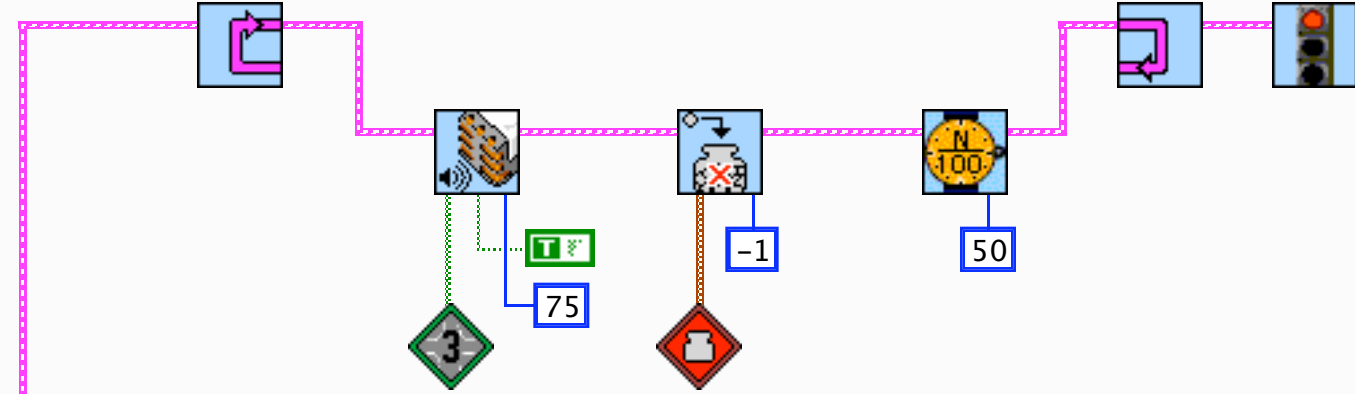
TOUCH Monitor



This is the main program. It has the responsibility to monitor the touch sensor and stop everything if it is hit.

When contact is made, a sound is played (note that this is a sound file, not just a synthesized tone), and then all other tasks are halted.

SOUND Monitor



The *SOUND Monitor* is explained in the box to the right. ==>

Designing Behaviors
Fundamentals of Interface Design
UW Division of Design
ART 483, Winter 2007
Davidson/Roesler

Initialization

First store the value "2" in the red container and then beep to signal that the robot is ready.

The red container is simply a storage location that holds a number. We will use the value in that container to indicate whether the robot should pause (the dog stops tracking) or move (track the line).

A number greater than zero (> 0) means pause, while zero or a number less than zero (≤ 0) indicates it is ok to run. This is an arbitrary convention that we set, not any sort of technical or mystical value.

The initial value of "2" (or any positive number) means the robot will initially be paused, and wait for a command to start moving.

LIGHT Monitor

This task has the responsibility to monitor the light sensor and direct the robot to follow a black line. It also obeys the setting of the "pause/track" switch while following. It runs continuously until stopped by the main program.

The following "pseudo-code" summarizes the method in a textual form of logic.

```
-- follow a black line on the ground, only if robot is not paused
loop forever
  if (red container > 0) then
    -- pause in effect
    stop both motors
  else
    -- tracking, so check the light sensor value
    if (light is bright) then
      -- off the track, turn left
      stop left motor
      run right motor
    else
      -- on the track, turn right
      stop right motor
      run left motor
    endif -- light sensor check
  endif -- container check
```

The method is to continuously check the light sensor and evaluate the intensity of the reflected light by comparing its value to a threshold level. If the sensor detects a value above the threshold (here we set it to 40% brightness, but this can be changed depending the specific environment), then it is assumed to be white and the robot is off the track. If it is 40% or less, then it is assumed to be black and on the track.

If the robot is on the track, it turns a little to the right and continues forward. If it is off the track, it instead turns a little to the left and proceeds forward also. Since it is doing this continuously, in this way it inches forward (at a given speed, also modifiable according to taste), constantly adjusting its course to the left or right to stay on the black line.

This scheme seems like magic, but it actually works. There is a flaw in the logic which you will undoubtedly discover if you give the robot a complicated track to follow, but it works especially well for a circular track. Improving the scheme to follow a more general track is, as math professors love to say, left as an exercise for the student.

Within the task, each time through the loop there is an overarching test that checks to see if the robot is paused or allowed to be moving (tracking the line).

To do this, we test the value of the number stored in the red container to decide whether to move (read the light sensor and run the motors) or just stop both motors. Since this happens each time through the loop, it makes the robot very responsive.

If the red container value is positive (> 0), it means we should be paused and so we just stop both motors. Otherwise, we go on and check the light sensor and move in the right direction.

SOUND Monitor

This task has the responsibility to monitor the sound sensor and control whether the robot pauses or runs. It runs continuously until stopped by the main program.

Each time a loud noise (a master's command) is detected, it toggles (flips) a "switch" that indicates whether the robot should be paused or tracking. The switch is simply a container that holds a number indicating "pause" or "track."

The following "pseudo-code" summarizes the whole method in a textual form of logic.

```
-- toggle pause/track switch each time a loud noise is detected
loop forever
  wait for loud noise
  multiply value in red container by -1
  wait 1/2 second
```

The key to this task's method is that the only significant aspect of the value stored in the red container is whether the number is positive (> 0) or not (≤ 0). Positive indicates paused, and negative indicates stopped.

So (as long as the number is not actually equal to zero), we can use a trick of basic algebra and simply multiply it by -1 each time a noise is detected. Since any positive number multiplied by -1 will give the negative version of that number, AND multiplying any negative number by -1 will give the positive equivalent, this effectively toggles the setting of the red container with each multiplication.

A little algebraic summary of this trick:

$$\begin{aligned} +2 * -1 &= -2 \\ -2 * -1 &= +2 \end{aligned}$$

During the initialization of the program, a value must be stored in the red container; otherwise it would default to zero and this scheme would collapse in a clever programmer's pile of trickery. (Why?) The reason to use it is that it makes this task very efficient; it only has to do a single multiply command when the sound triggers it. This allows the dog to rapidly respond to its master's commands!

An equivalent but longer piece of logic to accomplish the same thing could be:

```
if (red container < 0) then
  put "2" in red container
else
  put "-2" in red container
endif
```

By the way, the purpose of waiting for 1/2 second after toggling the switch is to allow the sound to die down a little before looping back and checking it again. Otherwise the robot would toggle the switch a few extra times after each sound is detected.