Behavior Based Control

Institute for Personal Robots in Education (IPRE)
Behavior Based Control
Behavior Based Control

- Behavior based control allows you (the programmer) to specify robot behaviors.
- Multiple behaviors “fight it out” for control of the robot.
Each Behavior

- May use sensor values to determine what to ask the robot to do, or may not.
- Makes a recommendation to the arbitrator about what action the robot should take.
  - or....tells the arbitrator that it doesn't have an opinion.
The Arbitrator

- Asks each behavior what the robot should do.
- Decides which behavior(s) to use.
  - Advanced Arbitrators may even allow the robot to do a mixture of different behaviors at the same time!
Implementing a Behavior based control system

- You need to specify each behavior
  ...and...
- An arbitrator that decides which behavior to follow at any particular point in time and makes the robot's actuators follow that behavior's recommendation.
Problem Statement:

- Your robot should move continuously looking for a bright light. When it finds a bright light, it should go towards it. The robot should turn to avoid hitting obstacles.
Finding Behaviors (look for verbs!)

- Your robot should **move** continuously **looking** for a bright light. When it finds a bright light, it should **go towards it**. The robot should **turn to avoid** hitting obstacles.
Summarized:

- Move
- Look (for light)
- Go to the light
- Avoid obstacles
Combined/Simplified:

- Move
- Go towards light
- Avoid obstacles
Prioritized:

- 1st: Avoid obstacles
- 2nd: Go towards light
- 3rd: Move
Questions about behaviors

- How does the behavior communicate with the arbitrator?
  - How does it indicate what action it wants the robot to take?
  - How does it indicate that it doesn't have a recommendation?
- All of these questions can be answered by building the Arbitrator.
The Arbitrator

- Calls each behavior.
- Sees what each behavior wants to do.
- Selects the recommendation from the highest priority behavior.
- Tells the robot to do that behavior's recommended action.
The Arbitrator/Behavior interface

- Each behavior needs to return the same data (in the same order) to the Arbitrator.
- You can pick any data and any order you want, but you must be consistent in all of your code.
- For this example, we choose to return:
  - Boolean – Tells if a behavior has a recommendation (True) or not (False)
  - Translate Value – Float from -1.0 to 1.0 telling the recommended translate amount. (0.0 is stopped)
  - Rotate Value – Float from -1.0 to 1.0 telling the recommended rotate amount. (0.0 is no rotation)

\[ [ \text{True}, \ 0.0, \ 0.0 ] \] - Represents a recommendation to stop completely.
Behavior 1: Cruse

#Move behavior: Always recommends that the robot move.
def move():

    return( [ True, -1.0 , 0.0 ] )
#Avoid behavior, makes the robot turn away from obstacles.
def avoid():
sensorVals = getIR()
leftV = sensorVals[0]
rightV = sensorVals[1]

if (leftV == 0) or (rightV == 0):
    return([True, 0.0, 1.0])
else:
    return([False, 0.0, 0.0])
# Calls each behavior in the order listed. If a behavior gives a recommendation, 
# arbitrate passes the recommended action on to the robot and then skips the 
# rest of the behaviors. (So avoid has a higher priority than move.)

def arbitrate():
    behaviors = [avoid, move]

    for behavior in behaviors:
        values = behavior()

        hasRec = values[0]
        transVal = values[1]
        rotVal = values[2]

        if (hasRec == True):
            translate(transVal)
            rotate(rotVal)
            return()
Behavior 3: Seek Light

#seekLight behavior: turns towards (bright) light

def seekLight():
    values = getLight()
    leftV = values[0]
    rightV = values[2]

    if (leftV > 200) and (rightV > 200):
        return( [False, 0.0, 0.0]  )
    else:
        if (leftV < rightV):
            return( [True, -0.5, 0.33]  )
        else:
            return( [True, -0.5, -0.33]  )
def arbitrate():
    behaviors = [avoid, seekLight, move]

    for behavior in behaviors:
        values = behavior()

        hasRec = values[0]
        transVal = values[1]
        rotVal = values[2]

        if (hasRec == True):
            translate(transVal)
            rotate(rotVal)
            return()