How to Teach an Autonomous Robot to Play Catch

Final Project Writeup

submitted to the faculty

of the

UNIVERSITY OF MASSACHUSETTS AT LOWELL

in partial fulfillment of the requirements for

91.549: Mobile Robotics

by

Patrick Hoey _____

19-December-2005

_____

Holly Yanco, Advisor

Table Of Contents

Abstract

This paper will discuss the background of the project research found in literature. The implementation will be discussed in regards to teaching a Pioneer autonomous robot to play catch with varying types of balls. The artificial intelligence methods used in this process include neural networks and vision. The results of the research will be evaluated along with the deviation from the original intent of the project due to certain limitations. Future work on the project will be mentioned to enhance the research currently done. Finally, a code snapshot used to implement the project will be attached as a reference in the appendix.

1 Introduction

There are two specific methods, specifically neural networks and vision, used within the artificial intelligence community that could potentially create a powerful combination. Neutral networks are one method of artificial intelligence used by modeling how the human brain receives and transmits information via synapses. Vision is another method of artificial intelligence used by modeling how humans see and interpret objects.

Used together, this project will attempt to emulate the behavior of a small child during their early years of development, specifically when they learn to catch and return a ball.

This project will attempt to use vision so that the Pioneer robot will be able to detect the varying types of balls and keep its attention on them. This project will also attempt to use neural networks so that the robot can learn to play catch with only certain types of balls, not all objects. The robot will also be trained to return the ball or place it in the correct storage device.

2 Background

The general concept for this project was from Dean Pomerleau's paper entitled "Neural Network Vision For Robot Driving", where a system was described for visual-based autonomous driving. Artificial neural networks have displayed promising performance and flexibility in other domains characterized by high degrees of noise and variability, such as handwritten character recognition and speech recognition and face recognition [10]. The intent of this project was to take a small subset of this vision problem and using the third party libraries available, produce a proof of concept.

The other aspect of this project was to simulate the interactions of teaching a child to play catch. Autonomous agents are systems that inhabit a dynamic, unpredictable environment in which they try to satisfy a set of time-dependent goals or motivations. Agents are said to be adaptive if they improve their competence at dealing with these goals based upon experience [1]. There was a requirement that intelligence be reactive to dynamic aspects of the environment, that a mobile robot operate on time scales similar to those of animals and humans, and that intelligence be able to generate robust behavior in the face of uncertain sensors, an unpredicted environment, and a changing world [2].

The Pioneer was to react to the ball is such a way as to demonstrate intelligence. This model was based upon the behavioral approach which limited the amount of state information. By eliminating internal state the reactive approach avoids the problem associated with maintaining the state, but runs headlong into the problem of extracting reliable information about the world through sensors.

3 Implementation

During the initial project proposal, there was a general idea of how to implement a solution to the problem for getting an autonomous robot to learn to pickup a specific type of ball. In the actual development and given certain time constraints, the scope of the project was constrained to just detecting a blue ball and being able to pick up the ball and throw it back.

3.1 Original Implementation

The original intent of the implementation of this project was to develop a framework based in Python, utilizing the Pyrobot platform. This framework will allow a vision system and a neural network system to work cooperatively. The robot will be able to use the sonar sensors to

detect distance from objects so that the robot does not bump into walls. Using the vision system, the robot will be able to determine the color and forms of the objects. Different algorithms will be used for image processing including blobbing and edge detection. Using the neural network system, the robot will be trained to follow and return certain types of balls as well as avoiding collisions with walls. The Pioneer robot will also utilize the grippers in order to catch and carry the different types of objects.

3.2 Environment

- Operating System: Linux (Fedora Distribution)
- Libraries:
    - Conx: Neural network processing
    - Phission: Image Processing/Capture
    - Pyrobot: Robot environment platform
- Language: Python
- Robot: Pioneer 2

3.3 Actual Implementation

There are five specific steps needed in order to accomplish the reevaluated implementation. First, be able to detect a blue color ball. This is done via the Phission library with blobbing. Second, create the train data for offline training for the Neural Net. Third, process the training data to create the weights with the Conx API. Fourth, start the robot with the learned behavior for detecting a blue color ball via Pyrobot. Fifth, given a certain probability that the object is a blue colored ball, pick it up and send it back

In the first step, the Pioneer needed to detect a blue colored ball so that it could differentiate this from all other objects. In order to accomplish this, the Phission library was used because of its performance over the Pyrobot platforms vision library. Phission is developed using a pipelined architecture with multithreading capabilities for heavy computation as well as being compiled natively with C++. This reduces significant overhead of Pyrobot's vision library due to being interpreted as well as not being multithreaded.

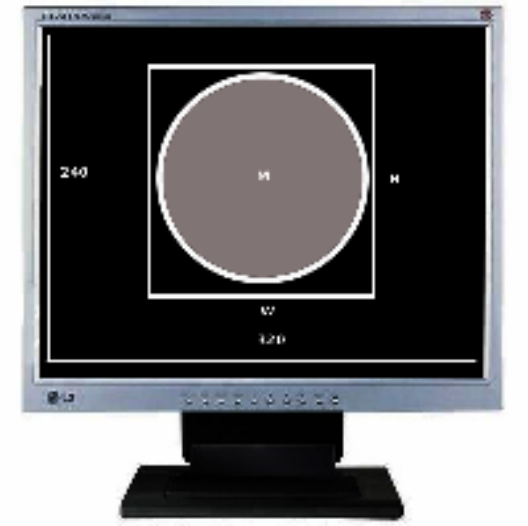

Figure 1 Blob Information

Figure 1 describes the essential components used in this project for deriving information from an image in real time. The Phission library utilizes a pipelined architecture where filters can be added to the pipeline so that multiple filters can be used at once. In this research, we are only concerned with the Blob filter from the Phission library. This filter is used to extract information from a live video feed (or image) based upon certain pixel values. This is used in order to match up the visual characteristics of a blue colored ball.

The M value represents the total mass of the object within the bounding rectangle. Given specific criteria for the hue, saturation and value (HSV), the blob filter will attempt to derive all relevant pixels that fit. There is a threshold which determines which blobs to keep based upon the mass value. For this project, we are only concerned with the largest blob mass because there will only be one object that the camera on the Pioneer will focus on.

This mass will change depending how the distance the object is from the camera and so a threshold value was chosen based upon the actual mass values of the blue colored ball from six to 36 inches. The distance will work in the case of this blue colored ball because we are extracting out the largest mass blob.

The mass is also determined by the resolution of the display in which the pixels are being evaluated. In the case of this project, a 320 by 240 sized display was used. The total amount of pixels in this display is 76,800 (320*240). This

information is used later on in scaling this mass to a value that can be input in the neural network.

The H value represents the height of the blob. The W value represents the width of the blob. These two values used together will determine the specific ratio or relationship between the height and the width of the largest blob. This is important because we can assume since we are looking at a round ball, that all round balls are symmetrical. The threshold is set close to one because this represents the nature of symmetry.

Figure 2 Blue Colored Ball

Figure 2 shows the Pioneer 2 robot nicknamed "Trinity". A blue ball is held in front of the camera of Trinity so that it can be processed.

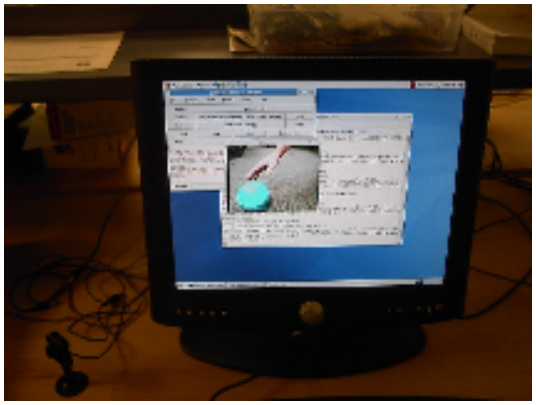

Figure 3 Blob of the Blue Colored Ball

Figure 3 represents what the blob of the blue colored ball looks like within the project application. The blob information is exported as a light cyan color to differentiate from the actual ball color itself. In looking at this image, the blue ball actually has very large blob information and shows up very distinctly in this image.

Figure 4 Orange Colored Ball

Figure 4 demonstrates the test to verify that the blob information is correctly identified and that the algorithm can distinguish between an orange ball and the blue ball correctly.

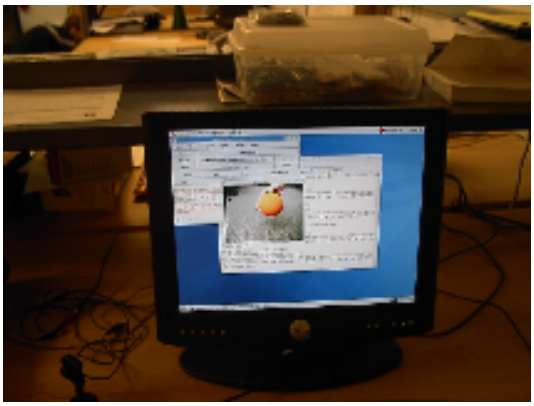

Figure 5 No Blob for Orange Ball

Figure 5 demonstrates that the blob filter including all the threshold values does not pick up on the orange ball at all. This shows that the blob information and threshold values do work in this environment.

The second step is to create the training data for offline training for the Neural Net. In order to feed in values into the Conx API neural net, the values needed to be rescaled to values between 0 and 1. The mass of the blob was scaled using the following formula:

(actual mass)/(total mass)

where total mass was 76,800, or all the pixels represented by the resolution of 320x240.

The height and width ratio was scaled by the following formula:

(height)/(width)

The width needed to account for the case of zero (to avoid divide by zero error) and therefore if the width had a value of 0 this would be updated to the value of 1.

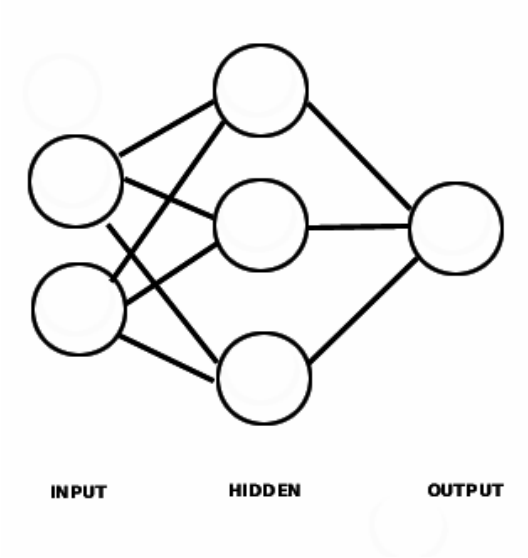

Figure 6 Neural Net Layout

Figure 6 describes the general neural net layout used to "teach" the Pioneer to look for a specific colored ball. There are two inputs, one is the scaled value of the mass of the blob and the second input is the ratio between the height and width of the blob. The hidden layer is represented in this graph by three nodes. This piece of the neural network configuration was done via trial and error as to the most optimal path for the training to use. Finally, the output value represents the percentage that the neural network can derive from an image that the image shows a blue colored ball.

This data was trained on 1000 steps of data. Each step essentially returns either true for the case that a blue colored ball is detected or false for the case where a blue colored ball is not detected. The threshold for this determination is based upon a certain range of the mass of the largest blob and a range of the height/width ratios.

The third step is to process the training data to create the weights with the Conx API. The training data produced was run through 1000 epochs via the Conx API. This allows the neural network library to parse the information from the training data and evaluate the weights. These weights are the learned responses from the data provided. In the case that the object has been determined to be a blue ball, the weight will be larger than in the opposite case.

The fourth step is to start the robot with the learned behavior for detecting a blue color ball via Pyrobot. This was done to verify the results of the learned behavior and to adjust any values that do not agree with the actual results. These adjusted values are then fed back into the neural net for processing to get stronger weights. This is the process for guaranteeing the there is good test data available for the neural network to process.

The fifth step is that given a certain probability that the object is a blue colored ball, pick it up and send it back. The threshold for determining if the object is a blue colored ball was set to 75%, since there is a margin of error that needs to be accounted for in these models. The actual action of picking up the ball is done via the grippers of the Pioneer. The functionality included picking up the ball and then putting the ball back down. Once the ball is on the ground, the Pioneer will move forward to push the ball forward. The Pioneer would then go back to a rest state and process the camera information again looking for a blue colored ball.

4 Results

The results include the robot which can distinguish between a blue colored ball and all other colored balls and that the robot will grab the ball in front of it, pick it up and then push it back via the grippers.

This is a deviation based upon the original idea for the implementation but due to time constraints and the dynamic nature of the development environment (including dependencies on third party libraries) these results are a proof of concept to the idea of combining imaging with neural network processing.

5 Limitations and Issues

The first issue encountered is that the Pyrobot core vision library was very slow. There is not a

specific metric to describe the slowness factor but it was significant enough that no real image processing could be accomplished in real time. The solution was to switch to the Phission API for all image capturing and image processing.

The second issue encountered was specific to the cameras on the robots. The cameras did not work because they needed updated drivers. The solution was to have the drivers updated for the cameras. These enabled the cameras to capture input but did not allow for panning or zooming itself.

The third issue encountered during this project is that the HSV values of the ball could not be determined via trial and error. The solution was to use an internal QT based histogram application (QT-Phission) to calibrate for the HSV values. This application was modified for this project in order to take an actual snapshot of the image in real time and write out the HSV values out to a file. This file would then be read in by the main application in order to determine the HSV values from the calibration. This would allow for real time calibration of the lighting conditions so that the vision problems with lighting could be minimized. The QT-Phission application was nice because it visualized the actual values of the objects in the cameras and allowed the user to select the regions of interest via an intuitive interface.

The fourth issue encountered was the use of the Phission library itself. The reason for the difficulty in using the Phission package was due to the lack of documentation. Reviewing the header files for the interface and looking through the source code of an implementation is one method for figuring out how the API methods work but simple descriptions of what the methods tried to accomplish would save a lot of time for the end user developer. The solution for this was to sit with the developer of the Phission library for a question and answer session.

The fifth issue encountered was that Phission could not be run correctly on the local development machine. The solution was to set specific environment variables which set the library path. These environment variables are not documented and therefore there was time spent troubleshooting the problem.

The sixth issue encountered was that Phission could not xhost back to local development machine from the Pioneer robot. This was a vital piece in order to troubleshoot issues with the camera or calibration issues. The solution was to rebuild the Phission library on the Pioneer robot with specific undocumented flags in order to allow the display to xhost back to the client machine.

The seventh issue encountered was that the camera image was not displayed correctly from the Pioneer robot. This was due to a slightly different configuration of the cameras on the Pioneer robots as opposed to the development machines. The solution was to accommodate these differences in the initialization code on the Pioneer robot.

The eighth issue encountered was that the grippers did not start. The solution was to start the grippers as a device. Initializing the grippers from the code did not work. There was a manual process of starting the gripper device from the Pyrobot application.

The ninth issue encountered was that there was no straightforward method for retrieving the state information from the grippers. The solution was the use sleep commands between the motions instead of relying on the state of the grippers themselves.

The tenth issue encountered was that the QT-Phission application could not be used to calibrate on the Pioneer robot itself due to the lack of appropriate libraries. The solution was to not use the calibration program on the Pioneer and just use the values calibrated from the development machines.

The eleventh and final issue encountered was that the Pioneer could not detect the color on ball due to the change in lighting conditions. The solution was to attach a light source to the robot so that the objects received a consistent light source.

Figure 7 Pioneer 2

Figure 7 describes the additional light source needed to provide consistent and ambient light to the objects for the camera.

There are also limitations on the image processing piece itself. Vision for robots in general is a hard problem to solve because of all the variables that occur in typical image capturing. This becomes evident when trying to set threshold values based upon the current environment because once the environment changes the variables get skewed.

6 Future Work

For the implementation itself, this project should include an edge detection algorithm so that the robot can distinguish between shapes. This is important because even with the symmetry ratio and the mass of the blob, there still needs to be additional processing on the image for specific areas of interest, specifically a round shape. One method for this would be to create some mask and apply it to the image at every frame for the edge detection. This method would be extremely difficult to maintain and therefore a proposed approach would to modify the current method of the learning process of the robot.

The idea is to create a good set of data which represents the blue color ball and a bad set of data which represents everything else. Training on the good data would differentiate itself from the bad data. This would allow the training to occur without worrying about the specifics of the data since it changes depending on the environment.

The other idea is to implement movements in a closed loop via state variables with the Pioneer's gripper. This would allow more robustness with the object interaction with the gripper. If the object was not in the gripper, the gripper could then open up the grippers and wait for the object. This would simulate a more intelligent interaction with object handling than just a standard iterative loop where the grippers are run on a set pattern.

7 Conclusion

This project attempts to use vision so that the Pioneer robot will be able to detect the varying types of balls and react accordingly. This project also attempts to use neural networks so that the robot can learn to play catch with only certain types of balls, not all objects. The robot is also trained to return the ball or place it in the correct storage device.

There were limitations in the approach taken with this project, such as trying to process images upon specific threshold values. This can quickly become a problem when dealing with multiple variables to process in an image. It would also become computationally expensive to derive the necessary information on these images in real time using this method.

The future work should address these concerns and be able to add more robust code to deal with the varying environments.

8 Appendices

Code

NNCollectBallData.py

```
# Patrick Hoey
# Final Project

#HACK include because apparently Python does division with integers, so
of course it will round to 0. Doh!
from __future__ import division
```

```
import sys
from time import *

sys.path.append("/usr");
sys.path.append("/usr/local");
sys.path.append("../../../../");
sys.path.insert(0,"/home/phoey/installs");
#/usr/local/pyrobot/bin/pyrobot -r RobotStub.py -b BlobTest.py

#-----------------------------------------------------------------------
-------
# Load brain stuff
from pyrobot.brain.behaviors.core import *  # Stop
# MUST BE AFTER ANY OTHER brain imports
from phission.examples.pyro.brains import Brain

#-----------------------------------------------------------------------
-------
# import the modules from the plugins directory.
from phission import *

def saveListToFile(ls, file):
    for i in range(len(ls)):
        file.write(str(ls[i]) + " " )
    file.write("\n")


#-----------------------------------------------------------------------
-------
class NNCollectBallData(Brain):

    def determineBallObject(self, blobMass, blobMassRatio):
        #Logic for determining the values
        #First we want to determine if the color is blue
        if blobMass > self.maxBlobMass or blobMassRatio >
self.maxBlobHW:
            return 0
        #The blue ball will be between 6 inches to 24 inches away
        elif blobMass > 5000 and blobMass < 50000 and blobMassRatio >
.7 and blobMassRatio < 1:
            print "Good size mass"
            return 1
        #When all else fails
        else:
            return 0


    #-------------------------------------------------------------------
-----------
    # The setup function is called right after the module is __init__'d
    #-------------------------------------------------------------------
-----------
    def setup(self):
        #Threshold values for CANNY filter (edge detection)
        self.cannyLow = 40
        self.cannyHigh = 255
```

```
        #HSV Values for BLOB
        self.hsvH = 141
        self.hsvS = 121
        self.hsvV = 187

        #Height and width of the display
        self.windowW = 320
        self.windowH = 240

        #This mass is dependent on the size of the window
        self.maxBlobMass = self.windowW*self.windowH
        self.maxBlobHW = 1.5

        self.counter = 0
        self.maxSteps = 1000

        self.datafile1 = open("BDvalues.dat", "w")
        self.datafile2 = open("BDtranslatevalues.dat","w")

        # phSystem manages starting/stopping/halting of displays,
pipelines
        # and captures; use addCapture,addPipeline,addDisplay to add to
the
        # system
        self.system = phSystem();

        # Don't let the robot sleep when it's actively running
        self.setSleepTime(0.01)

        #-------------------------------------------------------------
-------
        # Create the SDL display window
        self.display =
[X11Display(self.windowW,self.windowH,"Unfiltered"),

X11Display(self.windowW,self.windowH,"BlobView"),

X11Display(self.windowW,self.windowH,"CannyView")];

        #self.system.addDisplay(self.display[0])
        self.system.addDisplay(self.display[1])
        #self.system.addDisplay(self.display[2])

        #-------------------------------------------------------------
-------
        # Capture device
        self.capture = V4LCapture();
        self.system.addCapture(self.capture);

        self.capture.set(self.windowW,self.windowH);
        self.capture.setChannel(0);

        #-------------------------------------------------------------
-------
        # Filters
        self.gauss = gaussianBlur_Filter();
        self.blob = blob_Filter();
```

```
        self.cannyBlob = blob_Filter();
        self.canny = canny_Filter();
        self.inverse = inverse_Filter();

        #canny parameters
        self.canny.set(self.cannyLow,self.cannyHigh);

        #blob1 parameters
        self.incolor    =
phColorHSV24_new(self.hsvH,self.hsvS,self.hsvV);
        self.threshold  = phColorHSV24_new(20,50,50);
        self.outcolor   = phColorRGB24_new(0,255,255);

        self.blob.set(  self.incolor,
                        self.threshold,
                        1, # number of colors to match
                        self.outcolor,
                        1, # Highlight blobs
                        1);# Draw blob rectangles

        self.cannyBlob.set( self.incolor,
                        self.threshold,
                        1, # number of colors to match
                        self.outcolor,
                        1, # Highlight blobs
                        1);# Draw blob rectangles

        self.blob_min_size = 100;
        self.blobData = phBlobData();

        #-------------------------------------------------------------
-------
        # Pipeline
        self.pipeline = phPipeline();
        self.pipeline2 = phPipeline();

        self.system.addPipeline(self.pipeline);
        self.system.addPipeline(self.pipeline2);

        self.pipeline.add(self.gauss);
        self.pipeline.add(self.blob);

        self.pipeline2.add(self.cannyBlob);
        self.pipeline2.add(self.canny);
        self.pipeline2.add(self.inverse);

        #-------------------------------------------------------------
-------
        # Connect objects
        # Capture Output -> Pipeline Input

self.pipeline.setLiveSourceInput(self.capture.getLiveSourceOutput());

self.pipeline2.setLiveSourceInput(self.capture.getLiveSourceOutput());

        #Capture Output ->Display[0] Input Unfiltered
```

```
#self.display[0].setLiveSourceInput(self.capture.getLiveSourceOutput())
;

        # Pipeline Output -> Display[1] Input Blob

self.display[1].setLiveSourceInput(self.pipeline.getLiveSourceOutput())
;

        #Pipeline Output ->Display[2] Input Canny

#self.display[2].setLiveSourceInput(self.pipeline2.getLiveSourceOutput(
));

    #need to scale for neural net conx library
    def scaleBlobMass(self,val):
        #This mass is determined at setup. Essentially, the entire mass
of the current window
        x = val / self.maxBlobMass
        if x > 1:
            print "scaled > 1"
            return 1
        else:
            return x

    def scaleBlobHW(self,val):
        x = val / self.maxBlobHW
        if x > 1:
            return 1
        else:
            return x


    #-----------------------------------------------------------------
-----------
    # The step method is called serially when the robot is running
    #-----------------------------------------------------------------
-----------
    def step(self):
        robot = self.getRobot();

        # Get the most recent blob data
        self.blob.getBlobData(self.blobData);

        #The blobs are sorted by mass in descending order with the
largest at 0
        self.largestBlob = self.blobData.getBlob(0);

        #We want to look for a specific mass and a height/width ratio
(close to 1) for the ball
        #since the ball is semetrical.

        #Since we are determining a ratio we want to guarantee we do
not divide by 0
        if self.largestBlob.w == 0:
            self.largestBlob.w = 1;
```

```
        #Return a tuple with the values I am looking for
        isBlue = self.determineBallObject(self.largestBlob.mass,

self.largestBlob.h/self.largestBlob.w );

        #This is the code from my lab for collecting Data
        if self.counter > self.maxSteps:
            self.datafile1.close()
            self.datafile2.close()
            print "Done collecting data"
            self.stop()
        else:
            #saveListToFile(
[self.scaleBlobMass(self.largestBlob.mass),
            #
self.scaleBlobHW(self.largestBlob.h/self.largestBlob.w)],
self.datafile1)
            #saveListToFile([isBlue], self.datafile2)
            #self.move(translation, rotation)
            #print "Step: ",self.counter, " of ", self.maxSteps
            self.counter += 1


        phThread.msleep(200);

    #-----------------------------------------------------------------
-----------
    # The 'begin' method is called once at the beginning when the "Run"
button is
    # pressed and before the first call to the 'step' method.
    #-----------------------------------------------------------------
-----------
    # 'begin' can return an error code to stop a robot run session from
begining
    # in the case a fatal error occurs
    #-----------------------------------------------------------------
-----------
    def begin(self):
        return self.system.startup();

    #-----------------------------------------------------------------
-----------
    # The 'stop' method is called when the "Stop" button is pressed
    #-----------------------------------------------------------------
-----------
    def stop(self):
        return self.system.halt();

    #-----------------------------------------------------------------
-----------
    # The 'destroy' method should be called whenever the "Reload"
button is
    # pressed. It *should* be called when the program is shut down.
    #-----------------------------------------------------------------
-----------
    def destroy(self):
        return self.system.shutdown();
```

```
#-------------------------------------------------------------------------
-------
def INIT(robot):
    return NNCollectBallData('NNCollectBallData', robot)
```

NNTrainBallData.py

```
#Patrick Hoey

# Train a network offline
# Inputs: two scaled front sensor readings
# Outputs: one translate reading (unscaled)

from pyrobot.brain.conx import *
from pyrobot.system.log import *


def setFromFile(filename, cols = None, delim = ' '):
   fp = open(filename, "r")
   line = fp.readline()
   lineno = 1
   lastLength = None
   data = []
   while line:
      linedata = [float(x) for x in line.strip().split(delim)]
      if cols == None: # get em all
         newdata = linedata
      else: # just get some cols
         newdata = []
         for i in cols:
            newdata.append( linedata[i] )
      if lastLength == None or len(newdata) == lastLength:
         data.append( newdata )
      else:
         raise "DataFormatError", "line = %d" % lineno
      lastLength = len(newdata)
      lineno += 1
      line = fp.readline()
   fp.close()
   print "length of data array is", len(data)
   return data


def saveListToFile(ls, file):
   for i in range(len(ls)):
      file.write(str(ls[i]) + " ")
      file.write("\n")


totalEpochs = 1000

# Create the network
n = Network()
n.addThreeLayers(2,3,1)

# Set learning parameters
```

```
n.setEpsilon(0.3)
n.setMomentum(0.0)
n.setTolerance(0.05)

# set inputs and targets (from collected data set)
n.setInputs(setFromFile('BDvalues.dat'))
n.setTargets(setFromFile('BDtranslatevalues.dat'))

# Logging
log = Log(name = 'trainingWFLog.txt')
best = 0

for i in xrange(0,totalEpochs,1):
   tssError, totalCorrect, totalCount = n.sweep()
   correctpercent = (totalCorrect*0.1) / (totalCount*0.1)
   log.writeln( "Epoch # "+ str(i)+ " TSS ERROR: "+ str(tssError)+
                " Correct: "+ str(totalCorrect)+ " Total Count: "+
                str(totalCount)+ " %correct = "+ str(correctpercent))
   if best < correctpercent:
      n.saveWeightsToFile("trainingBDLog.wts")
      best = correctpercent
      print "done"
```

NNRunBallData.py

```
#Patrick Hoey

# Load in saved weights from offline training
# Inputs are the two front sensor readings
# Output is a translate value, used to control the robot

#HACK include because apparently Python does division with integers, so
of course it will round to 0. Doh!
from __future__ import division

from pyrobot.brain import Brain
from pyrobot.brain.conx import *
from time import *


import sys

sys.path.append("/usr");
sys.path.append("/usr/local");
sys.path.append("../../../../");
sys.path.insert(0,"/home/phoey/installs");
#/usr/local/pyrobot/bin/pyrobot -r RobotStub.py -b BlobTest.py

#-----------------------------------------------------------------------
-------
# Load brain stuff
from pyrobot.brain.behaviors.core import *  # Stop
# MUST BE AFTER ANY OTHER brain imports
from phission.examples.pyro.brains import Brain

#-----------------------------------------------------------------------
-------
# import the modules from the plugins directory.
```

```
from phission import *

class NNRunBallData(Brain):

    def setup(self):
        self.n = Network()
        self.n.addThreeLayers(2,3,1)
        self.doneLearning = 1

self.n.loadWeightsFromFile("/home/phoey/project_final/trainingBDLog.wts
")
        self.n.setLearning(0)

        #HSV Values
        self.hsvH = 141
        self.hsvS = 121
        self.hsvV = 187

        #Height and width of the display
        self.windowW = 320
        self.windowH = 240

        #This mass is dependent on the size of the window
        self.maxBlobMass = self.windowW*self.windowH
        self.maxBlobHW = 1.5

        self.counter = 0
        self.maxSteps = 1000

        # phSystem manages starting/stopping/halting of displays,
pipelines
        # and captures; use addCapture,addPipeline,addDisplay to add to
the
        # system
        self.system = phSystem();

        # Don't let the robot sleep when it's actively running
        self.setSleepTime(0.01)

        #-----------------------------------------------------------------
-------
        # Create the SDL display window
        self.display =
[X11Display(self.windowW,self.windowH,"Unfiltered"),

X11Display(self.windowW,self.windowH,"BlobView"),

X11Display(self.windowW,self.windowH,"CannyView")];

        #self.system.addDisplay(self.display[0])
        self.system.addDisplay(self.display[1])
        #self.system.addDisplay(self.display[2])

        #-----------------------------------------------------------------
-------
        # Capture device
        self.capture = V4LCapture();
```

```
        self.system.addCapture(self.capture);

        self.capture.set(self.windowW,self.windowH);
        self.capture.setChannel(0);

        #-------------------------------------------------------------
-------
        # Filters
        self.gauss = gaussianBlur_Filter();
        self.blob = blob_Filter();
        self.canny = canny_Filter();
        self.inverse = inverse_Filter();

        self.incolor    =
phColorHSV24_new(self.hsvH,self.hsvS,self.hsvV);
        self.threshold  = phColorHSV24_new(20,50,50);
        self.outcolor   = phColorRGB24_new(0,255,255);
        self.blob.set(  self.incolor,
                        self.threshold,
                        1, # number of colors to match
                        self.outcolor,
                        1, # Highlight blobs
                        1);# Draw blob rectangles

        self.blob_min_size = 100;
        self.blobData = phBlobData();

        #-------------------------------------------------------------
-------
        # Pipeline
        self.pipeline = phPipeline();
        self.pipeline2 = phPipeline();

        self.system.addPipeline(self.pipeline);
        self.system.addPipeline(self.pipeline2);

        self.pipeline.add(self.gauss);
        self.pipeline.add(self.blob);

        self.pipeline2.add(self.canny);
        self.pipeline2.add(self.inverse);

        #-------------------------------------------------------------
-------
        # Connect objects
        # Capture Output -> Pipeline Input

self.pipeline.setLiveSourceInput(self.capture.getLiveSourceOutput());

self.pipeline2.setLiveSourceInput(self.capture.getLiveSourceOutput());

        #Capture Output ->Display[0] Input Unfiltered

#self.display[0].setLiveSourceInput(self.capture.getLiveSourceOutput())
;

        # Pipeline Output -> Display[1] Input Blob
```

```
self.display[1].setLiveSourceInput(self.pipeline.getLiveSourceOutput())
;

        #Pipeline Output ->Display[2] Input Canny

#self.display[2].setLiveSourceInput(self.pipeline2.getLiveSourceOutput(
));

    #need to scale for neural net conx library
    def scaleBlobMass(self,val):
        #This mass is determined at setup. Essentially, the entire mass
of the current window
        x = val / self.maxBlobMass
        if x > 1:
            print "scaled > 1"
            return 1
        else:
            return x

    def scaleBlobHW(self,val):
        x = val / self.maxBlobHW
        if x > 1:
            return 1
        else:
            return x

    def grabBall(self):
        #Make sure we are in a reset state
        self.robot.gripper[0].deploy()
        time.sleep(5)
        self.robot.gripper[0].store()
        time.sleep(5)
        self.robot.gripper[0].deploy()
        #time.sleep(5)
        self.robot.move(0.5,0)
        time.sleep(1)
        self.robot.move(-0.5,0)
        time.sleep(1)
        self.robot.move(0.0,0)

    def step(self):
        # Set inputs
        robot = self.getRobot();

        # Get the most recent blob data
        self.blob.getBlobData(self.blobData);

        #The blobs are sorted by mass in descending order with the
largest at 0
        self.largestBlob = self.blobData.getBlob(0);

        #We want to look for a specific mass and a height/width ratio
(close to 1) for the ball
        #since the ball is semetrical.
```

```
        #Since we are determining a ratio we want to guarantee we do
not divide by 0
        if self.largestBlob.w == 0:
            self.largestBlob.w = 1;


self.n['input'].copyActivations([self.scaleBlobMass(self.largestBlob.ma
ss),

self.scaleBlobHW(self.largestBlob.h/self.largestBlob.w)])
        self.n.propagate()

        blobMassActual = self.n['output'].activation[0]
        #blobHWActual = self.n['output'].activation[1]

        if blobMassActual > .75:
            self.grabBall()


        #print "massActual", blobMassActual
        #print "HWActual", blobHWActual

    #-----------------------------------------------------------------
-----------
    # The 'begin' method is called once at the beginning when the "Run"
button is
    # pressed and before the first call to the 'step' method.
    #-----------------------------------------------------------------
----------
    # 'begin' can return an error code to stop a robot run session from
begining
    # in the case a fatal error occurs
    #-----------------------------------------------------------------
-----------
    def begin(self):
       return self.system.startup();

    #-----------------------------------------------------------------
-----------
    # The 'stop' method is called when the "Stop" button is pressed
    #-----------------------------------------------------------------
-----------
    def stop(self):
        return self.system.halt();

    #-----------------------------------------------------------------
-----------
    # The 'destroy' method should be called whenever the "Reload"
button is
    # pressed. It *should* be called when the program is shut down.
    #-----------------------------------------------------------------
-----------
    def destroy(self):
        return self.system.shutdown();

def INIT(engine):
    return NNRunBallData('NNRunBallData', engine)
```

trainingBDLog.wts

```
(lp0
F-14.754267541724463
aF25.064804972531284
aF13.786326673411923
aF-33.564956125475682
aF52.578034906850228
aF-10.169341570541869
aF11.187757132247263
aF18.468573289912772
aF-35.131248186317933
aF-25.155872677723565
aF20.57363161750802
aF18.218836685173404
aF11.335469741503482
a.
```

run.sh

```
#!/bin/sh

#Step 1: run calibratePhission to get correct HSV values

#Step 2: run this script

#pyrobot -s PlayerServer -w pioneer.cfg -r Player6665

export PATH=$HOME/installs/:$HOME/installs/bin:$PATH
export LD_LIBRARY_PATH=$HOME/installs/lib:$LD_LIBRARY_PATH

#pyrobot -w pioneer.cfg -r Test.py -b NNCollectBallData.py
#pyrobot -r Test.py -b NNRunBallData.py
#pyrobot -w pioneer.cfg -r Player.py -b NNRunBallData.py
pyrobot -s PlayerServer -w pioneer.cfg -r Player6665 -b
NNRunBallData.py
```

9 References

[1] "Modeling Adaptive Autonomous Agents," Pattie Maes, Artificial Life, 1(1-2), MIT Press, pages 135-162, 1994.

[2] "Intelligence Without Reason," Rodney A. Brooks, Proceedings of 12th International Joint Conference on Artificial Intelligence (IJCAI-91), Sydney, Australia, pages 569-595, August 1991.

[3] “On Three-Layer Architectures”, Eran Gat, Artificial Intelligence and Mobile Robotics, in D. Kortenkamp, R. P. Bonnasso and R. Murphy (eds.), AAAI Press, pages 195-210, 1998.

[4] A Robust Layered Control System for a Mobile Robot, Rodney A. Brooks, IEEE Transactions on Robotics and Automation, 2(1), pages 14-23, April 1986.

[5] A Mobile Automaton: An Application of Artificial Intelligence Techniques, Nils J. Nilsson, Technical Note 40. AI Center, SRI International, Mar 1969. Presented at IJCAI 1969.

[6] Vision for Mobile Robot Navigation: A Survey, Guilherme N. DeSouza and Avinash C. Kak, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 24, No 2, February 2002.

[7] Vision-Guided Flight Stability and Control for Micro Air Vehicles, Scott M. Ettinger, Michael C. Nechyba, Peter G. Ifju and Martin Waszak, IROS 2002.

[8] Vision-Guided Flight Stability and Control for Micro Air Vehicles, Scott M. Ettinger, Michael C. Nechyba, Peter G. Ifju and Martin Waszak, IROS 2002.

[9] Vision-Guided Flight Stability and Control for Micro Air Vehicles, Scott M. Ettinger, Michael C. Nechyba, Peter G. Ifju and Martin Waszak, IROS 2002.

[10] Neural Network Vision for Robot Driving, Dean Pomerleau, The Handbook of Brain Theory and Neural Networks, M. Arbib, ed., 1995.

[11] Evolving 3D Morphology and Behavior by Competition, Karl Sims, Proceedings of Artificial Life IV, Rodney Brooks and Pattie Maes (eds.), MIT Press/Bradford Books, pages 28-39, 1994.

[12] Generative Encodings for the Automated Design of Modular Physical Robots, Gregory S. Hornby, Hod Lipson and Jordan B. Pollack, IEEE Transactions on Robotics and Automation, 19(4), pages 703-719, 2003.