Homework 5 - due 10/08/2003

15-491: CMRoboBits

1 Introduction

This lab introduces high level vision and builds on the basics that you learned last week with low level vision. You will implement filters to recognize part of Aibo's charging station and the speed board, being careful to disambiguate between the speed board wheels and the orange ball. You will also return a pose estimate for the speed board (be aware that this is very difficult and we only expect a rough estimate in practice) Example images from the robot are provided and you will work exclusively under Linux using the vis_test program. This means that you have full access to DDD, print statements, and any other tools you wish to use for debugging.

2 Downloading the sample images

A tar.gz file containing all of the images and .inf files for this assignment is available from the course webpage. Download this file and expand it in the ∼/dogs directory by running: "tar -zxvf images.tar.gz". Tar will create several directories in ∼/dogs/images containing the sample data for each object. Look through these with xzgv to familiarize yourself with the data.

3 Running vis_test

Vis_test is a program that runs the vision code from the robot under Linux. It performs exactly the same processing and uses all of the filters from the robot to extract the location of objects in the images that you pass to it. It can either accept raw vision frames or RLE frames and each frame must have a matching .INF file that gives the camera position. Since the camera position is available for each image, it is possible to reconstruct the ego-centric position of each object in the scene from the recorded data. In this lab you will pass raw images to vis_test and it will threshold them using the thresh.tm file in agent/config/config.

3.1 Basic use of vis_test

To make and run vis_test:

- cd ~/dogs/util/vision_test
- make
- *You must run vis_test from this directory as it looks for its configuration file in this location. Note that this file points to a colors.txt file and thresh.tm (minus the extension in the config file for obscure reasons). Currently it points to ~/dogs/agent/config/config which means you should copy your thresh.tm file generated in your previous assignment to that location for now. Later on, we will add thresholds to CVS once we create them from your previous homework.*
- Read the above warning again and copy thresh.tm to the specified location if we have not yet distributed the thresholds. Otherwise, do a "cvs update" to retrieve the new ones from the repository. (We will send e-mail later this week when they are ready)
- Run vis_test with something like: "./vis_test ~/dogs/images/ball/i*.ppm".
- The GUI window must have the focus in order to receive commands. If it stops responding to keyboard input, click on its title bar.
- To control the GUI, you may press SPACE to advance an image, 'b' to move back, 'q' to quit, and 'r' to redraw the current frame.
- The GUI will display the thresholded image. The blue line represents the "Up" vector of the image.
- Watch the print outs as you browse through the images to figure out how well your object detectors are working.
- Remember to run make again after you make changes in the source. (We'll detail which files to change in another section)

3.2 Advanced vis_test

Vis_test can use the output of object recognition to divide files into several groups. This is useful when batch processing a large number of images files; vis_test can automatically write out the filenames of misclassified images to a file so that you can inspect problematic images individually after doing batch processing on the rest.

- In your code (in Vision.cc) set the DebugClass global variable to a numeric value. [This has been done for you in this assignment. Tower = 1; Bullseye = 2; Wheels = 3; Skateboard = 4; Nothing = 0]
- Copy an additional comment for the class into DebugInfo with sprintf or something similar [Again, this has been done for you, although you may add additional information if you wish or remove the default.].

- When you run vis_test, pass it the −−sort flag. This flag will cause it to create output files of the form 0.vfl, 1.vfl, 2.vfl, ..., n.vfl. Each file contains the filenames of the images that were assigned to each class along with some other information. Here is a sample: "01 images/skateboard/i0004.ppm DebugInfoStringGoesHere" The first item is the class number, the second the filename, and the last is the program specified info in DebugInfo.
- Note that you must set the confidence for objects in order to use the facility; that is how the program determines if an object is present in the image. By default, the confidence must be above .5.

Since the images are presorted, the individual VFL files are not particularly interesting on their own [unless you are looking for omissions]. However, they may be used in many ways with UNIX command line tools. There are no line breaks in the following commands, even though some will take several lines to write out.

To quickly generate a list of files that contain/don't contain some object: (requires a mostly working object detector)

- Set DebugClass=1 to indicate presence of object, DebugClass=0 to indicate it's abscence in Vision.cc.
- vis_test −−sort image_files_go_here
- cp 0.vfl to no.vfl
- cp 1.vfl to yes.vfl
- Run one of the following:
  - cut -d ' ' -f 2 yes.vfl | xargs xzgv
  - The above displays all of the images in yes.vfl. You can use this to check for errors and manually move the files from yes.vfl to no.vfl using a text editor (there should be few errors with a good object detector)
  - cut -d ' ' -f 2 no.vfl | xargs xzgv
  - The above is like the first one, except you look for false negatives instead of false positives. If you want to make a list of images and simply delete the false positives or negatives, use one of the following:
  - cut -d ' ' -f 2 yes.vfl >tmp.vfl
  - cp tmp.vfl yes.vfl
  - OR
  - cut -d ' ' -f 2 no.vfl >tmp.vfl
  - cp tmp.vfl no.vfl

To quickly test the accuracy of an object detector given a list of images that contain the object and a list of images that don't contain the object, you can do the following:

- run vis_test −−sort ImageFiles
- This is just like above and generates n.vfl where n is a debug class. Now, if you have yes.vfl and no.vfl from the previous step, you may run one of:
  - cut -d ' ' -f 2 1.vfl | diff -U 0 - yes.vfl
  - The above will give a list of all the images which have errors
  - cut -d ' ' -f 2 1.vfl | diff -U 0 - yes.vfl | grep -E '∧[+]'
  - The above will give a list of all the images which are false negatives.
  - cut -d ' ' -f 2 1.vfl | diff -U 0 - yes.vfl | grep -E '∧[-]'
  - The above will give a list of all the images which are false positives.
  - cut -d ' ' -f 2 1.vfl | diff -U 0 - yes.vfl | grep -E '∧[+]' | tail -n +2 | cut -b 2- | xargs xzgv
  - The above will show all of the images which are false negatives
  - cut -d ' ' -f 2 1.vfl | diff -U 0 - yes.vfl | grep -E '∧[-]' | tail -n +2 | cut -b 2- | xargs xzgv
  - The above will show all of the images which are false negatives

4 Files to view and edit

You will need to view and edit several files in this assignment:

- agent/Vision/VisionInterface.h
  Specifically, you should look at the VObject structure. This is what you will be filling with information about the objects you are detecting. Specifically, you must fill in *confidence, loc, distance, and orientation*. Set orientation to 0 for all objects other than the speed board. You will do the actual work of filling in these fields in instances of VObject that are accessible from Vision.cc.
- Vision.cc and Vision.h declare and define the following object detectors (this is the bit you will write; stub functions are already in place)
  - void findChargingTower(VObject *tower);
  - void findChargingBullseye(VObject *bullseye);
  - void findSkateboardWheels(VObject *wheels);
  - void findSkateboard(VObject *skateboard);
- Look at findBall or findMarkers for examples of how to calculate the locations of objects in space based on their pixel position in the frame.
- It is very helpful to add print statements to your object detector's method.

5 Procedure

- If your group number is odd, complete the findChargingTower method in Vision.cc using the above background information. You must fill in the confidence, loc, and distance fields of the buffer that is passed to your method.
- If your group number is even, complete the findChargingBullseye method in Vision.cc using the above background information. You must fill in the confidence, loc, and distance fields of the buffer that is passed to your method.
- All groups: Complete findSkateboardWheels. Fill in the same fields as you did above.
- All groups: Complete findSkateboard. Fill in all the files above *and* the orientation field.

6 What to hand in

Hand in a complete list of your changes to the source files by running:

- cd ∼/dogs
- cvs diff -U 5 -N > handin.txt
- enter password
- klog userid@andrew.cmu.edu
- cp handin.txt /afs/andrew/course/15/491/dropbox/groupn/
- unlog

Also, copy Vision.cc and Vision.h to your group's directory.

Additionally, run the wlan_setup command and pass it no arguements to convert your memory stick to use DHCP for wavelan. This doesn't matter for this assignment, but please do it before we forget to ask for next week.

7 Grading

We will evaluate your object detectors on test images that you were not given as examples. Grading will be based on the percentage of the examples that your detector correctly classifies (misclassified examples - e.g. returning a wheel location with high confidence for images of the ball - will could against you).

The procedure orders tasks by difficulty. Our grading will become more lenient as it progresses so that harder tasks are evaluated differently than the easy, initial tasks. For example, the best skate board orientation detector will receive full credit no matter how poorly it does.

- Performance on charging station part - 40 points
- Performance on skate board wheels - 40 points
- Performance on skate board orientation - 20 points