Assignment 5: Vision
CS 393R: Robotics

Due Date: Thursday, November 5, 2009

Your task: Write sanity checks for ball detection. Write code to detect field lines.

This assignment is to be done **individually**.

In this assignment, we will be using UTNaoTool, the tool developed by Austin Villa to develop and debug code on the humanoid Nao robots. You will be using UT's Nao codebase, and filling in missing parts of the vision module. There are two main components:

First, the **ball**:
You will implement checks to verify that an orange blob that is seen is in fact the ball. These checks include checking the ball's size, how much orange is in it, its elevation, if its above green, and if its not inside red or yellow (maybe seeing it in a shadow of a robot uniform.
For the ball checks, you will edit core/vision/BallDetection.cpp. Search for comments that say TODO inside the checkBallBlob() method. Here you check if the given blob is a reasonable ball. If it is, return a confidence greater than 1, otherwise, return a confidence less than 1. You have to fill in 5 sanity checks: ball size, color ratio, if its inside another color blob, that's its not floating in the air, and that it has some green/white below it.

Second, **lines**:
You will implement line detection. First you will detect which pixels are possible lines by looking for white pixels with green on either side. Then you will perform line fitting to determine which ones the lines are. Finally, you will determine which of these lines have intersections and identify them. For extra credit, detect the center circle separately from the rest of the lines.
For the line detection, you will edit core/vision/LineDetection.cpp. Search for comments that say TODO. In FormLines(), you need to fill in code to check if a pixel is a possible line point. Then you need to write code to fit lines to those points. Finally, perform some sanity checks on the lines. After you've decided you have a good line, you need to call CalLineDetails, like CalLineDetails(fieldLines[FieldLinesCounter]). In FormCorners(), you need to find the intersections of the lines, determine if their intersection is on the screen, and set valid ones as cornerPoints.

I have tested that the code works on the department machines, therefore I would recommend that you work on them. It does NOT work on the machines in the lab. If you prefer to sit in the lab, you can always ssh into one of the department machines and run it that way.

Steps:
1. Copy the code from my folder to wherever you are going to use it: scp -r emos.cs.utexas.edu:/u/todd/cs393r/vision .
2. Remember to set your LD_LIBRARY_PATH to include the folder the tool is in.
3. Go to the lib/lua directory. Type make clean, then make.
4. Compile the code for the first time. Go into the tools/UTNaoTool directory, and type make.
5. Take a look at the tool. In the tools/UTNaoTool directory, type ./UTNaoTool. Now open a log from the file menu. Click on 'Run Core'. Now click on Vision. Here you can see what the robot saw.

6. You will edit the following two files: core/vision/BallDetection.cpp and core/vision/LineDetection.cpp.
7. Once you've edited the file, you can re-compile the tool with your changes by typing make in the tools/UTNaoTool directory.
8. Then re-open the tool and open the log file. On the main window, be sure to click 'Run Core' to see what your vision is processing.

You will find the vision window of the tool useful for debugging. It has one large image on the left and 4 smaller ones on the right. Clicking on any of the small images make it the big image. The first image is the raw camera image; the second is the segmented image. The third image shows all blobs, all possible lines (valid or not), and all line points (red dots). The fourth image shows objects, i.e. the ball and valid lines. By selecting tooltips on the bottom left, you can hover your mouse over the big image to get information about that pixel, blob, or object.

You have two logs containing real images from RoboCup 2009 in Austria: vision.log and vision2.log. These logs contain all the real issues we have to deal with: people wearing orange, orange signs, skin coming up as orange, orange-red flower pots near the field, orange seats in the stands, white signs everywhere, odd shadowing, shiny red uniforms, etc. Good luck!

Checklist:

[____] (1 points) Implement a size check on the possible ball blobs.

[____] (1 points) Implement a check on the ratio of orange pixels within the ball.

[____] (2 points) Implement a check to see if the orange blob is inside a red or yellow blob.

[____] (2 points) Implement a check that the ball is above green pixels.

[____] (2 points) Identify which pixels are possible line candidates

[____] (2 points) Fit lines to these pixels.

[____] (2 points) Identify line intersections.

[____] (2 points) Clarity and quality of your memo. Turn it in at the time your assignment is graded.

[____] (2 points) **Extra Credit:** Implement code to detect the center circle on the field separately from the regular field lines.

Documentation:

- Blobs
    - Defined in core/common/Blob.h
    - This is a struct. Main things you need to use:
        - int minX, maxX, minY, maxY: these tell you the left, right, top, and bottom edges of the blob.
        - int pixels: how many pixels are in this blob.
        - int correctPixels: how many pixels are of the correct color.
        - int width, height, area: self-explanatory
        - int color: the color of the blob (defined in core/vision/Vision.h).
        - bool ignoreBlob: true if this blob should be ignored for some reason (possibly subsumed by a larger blob).
- Segmented Image
    - This is an array of the segmented colors of all the pixels in the image. Incrementing a char pointer to this array will bring you to the next pixel.
    - The colors are defined in: core/vision/Vision.h. Key ones: c_WHITE, c_ORANGE, c_FIELD_GREEN, c_ROBOT_RED, c_YELLOW, c_UNDEFINED.
- LinePoint
    - Defined in core/vision/LineDetection.h
    - This is a struct. Main things you need to use:
        - double PosX
        - double PoxY
        - bool ValidPoint
- FieldLine
    - Defined in core/vision/LineDetection.h
    - This is a struct. Main things you need to use:
        - int Points
        - double MinX, MinY, MaxX, MaxY
        - bool ValidLine, CompLine
        - LinePoint** PointsArray: this is an array of all the points you've used for this line
- CornerPoint
    - Defined in core/vision/LineDetection.h
    - This is a struct. Main things you need to use:
        - double PosX
        - double PoxY
        - bool Valid
        - FieldLine* Line[2]: this is an array of the two lines used to form this corner
        - short CornerType
- World Objects
    - These are defined in core/common/WorldObject.*
    - These objects describe where everything is on the field, as well as tell you if something was seen and where it was seen.
    - What the robot sees:
        - If you get the world object, the seen boolean tells if the robot saw that object that frame.
        - visionDistance and visionBearing tell you the distance and bearing that the robot saw the object.

        - Actual objects
            - The object's loc variable (a Point2D) tells you where the object is located on the field.
- Geometry, Point2D
    - You may want to use some of the existing code in Point2D and Geometry. This is located in core/common/Geometry.*
- Debug
    - You can add print statements that will show up in the tool's 'Log' window at the frame they were printed by using: memory->log(int, "something something"); where the int is the verbosity level from 0-100.
    - The log window will display all messages from that frame, and in the bottom left you can set the range of message levels you want to see.