CS100R Assignment 4

out: 12:00AM Oct 16th, 2007
due: **6:00PM Oct 26th, 2007**

1 Previously on Assignment 3

Write a function which drives a robot based on the position of the largest connected component and its area. More specifically, if the area of the blob increases, move the robot forward; if it decreases, move the robot backward. If the centroid moves to the left, rotate the robot to the left; if the centroid moves to the right, rotate the robot to the right. You will need to use the following robot functions `robotInit`, `robotDriveStraight`, `robotDriveCircle` `robotGetSensors`, and `robotGetFrame`. Look them up using Matlab's `help` command for information about how they're used. Place your code in `CC_robots_student.m`.

2 Introduction

We have already seen how to capture a desired region (e.g. red pixels) in an image using thresholding and connected components, but now we will examine a strategy for finding the orientation of a captured region. In this assignment, we will determine a simple orientation for an object by first finding a bounding polygon, and then by choosing a major axis along the length of the object and a minor axis along the width. Then, applying this algorithm to a real-time camera feed, we will be able to provide throttle control to the robots.

3 Convex Hull

As defined in the lecture, a convex hull is the *smallest convex polygon* (see Fig. 1) containing a set of points—this means that it is possible to draw a straight line from any point to any other point in the hull without ever crossing the boundary or coming out of the shape. The question is, of course, if we are given a set of points (e.g. the big red blob you found using connected components in A3), how do we find its convex hull?
Answer: Use the *Gift Wrapping* algorithm.

- Start with a point that is guaranteed to lie on the border of the hull (we choose the bottom right in our reference implementation, but you can choose top-left, or anything else, provided that you're consistent).
- Find a new point such that all the other points lie to the left of the line formed between that point and the point most recently added to the hull. Add that point to the convex hull. *Hint: you will want to very carefully consider how you'll handle colinear points.*
- Repeat.

There are various ways to check whether a point (x_2, y_2) lies to the left of a line from (x_0, y_0) to (x_1, y_1). We discussed a particularly cool cross product rule in the class. You can find a cross product using the following formula:

$$(x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0). \tag{1}$$

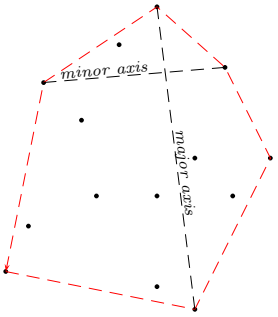

Figure 1: A convex hull.

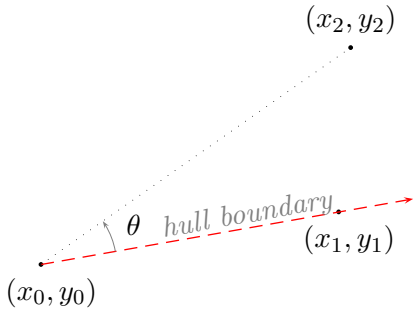

Figure 2: Left-ness of (x_2, y_2).

Since the cross product is dependent on $\sin\theta$ (see Fig. 2), it follows that its sign will indicate the side of the line on which the point falls: if the result is positive, the point is on the left of the line, and if the result is negative, the point is on the right. In the case that the cross product returns zero, we may conclude that the points are colinear.

4 Orientation

We define the orientation of a polygon (a convex polygon, in our case) by defining a major and a minor axis.

4.1 Major axis

The major axis of a polygon is the pair of vertices that are the farthest from each other. To find these points, find the distance between each pair of vertices and select the pair with the largest distance. Simply find the two points (x_1, y_1) and (x_2, y_2) that maximize $(x_2 - x_1)^2 + (y_2 - y_1)^2$.

4.2 Minor axis

We define the minor axis of a polygon as the closest pair of points such that each point of the pair is on the opposite side of the major axis. To find the minor axis, choose all possible pairs of points and sort[1] them in increasing order of distance. Start with the closest pair and check if the points lie on opposite sides of the major axis. If so, then they form the minor axis; otherwise, pick the next-smallest distance and repeat the same steps until you get a pair that satisfies this condition.

5 Linked List

Recall from lecture a way we can implement linked lists using arrays. First, we create an array M of size n. Then we allocate $M(1)$ and $M(2)$ as the header of the linked list. $M(1)$ contains a pointer to the first element of the linked list, and $M(2)$ contains the size of the linked list. Elements $M(3)$ through $M(n)$ contain the actual linked list, where $M(k)$ is the element and $M(k+1)$ is the pointer to the next element. If $M(k+1)$ has a value of 0, that denotes that $M(k)$ is the last element of the list. For example:

```
>> M = {5, 2, 4, 0, 8, 3}
```

$M(1) = 5$ places the first element at $M(5)$, $M(2) = 2$ indicates that the list has 2 elements in it. We can then see that the first element has a value of 8 and a pointer to $M(3)$, which has a value of 4. In other words, the items in the linked list are thus:

```
[8]->[4]
```

6 Task List

6.1 Functions to Write

We have provided template files for each of these functions, so grab them from `M:\wand\part5_template` and copy them to your working directory to get started.

1. Write a function that implements convex hull as described in Sec. 3 to find the shape of the points in the given binary image. Your function should be named `convex_hull_student`. It will accept a binary image as an argument and return an n-by-2 matrix of the form `[x1 y1; x2 y2; ...; xn yn]` containing a list of the vertices that form the shape of the convex hull. The points on your hull may be listed in either clockwise or counterclockwise order, and may or may not contain colinear

[1] You may use the builtin Matlab `sort` function: `sorted_array = sort(array);`

points. Your start point should appear both at the beginning and the end of the list. You may use the `test_convex_hull` function to test the correctness of your convex hull.

2. Write a function `poly_axes_student` that accepts an n-by-2 matrix of points which represent vertices of a polygon (the same kind that `convex_hull_student` returns) and returns two 2-by-2 matrices of the form `[x1 y1; x2 y2]`: the first matrix must contain the two points which form the major axis, and the second matrix must contain the two points which form the minor axis.

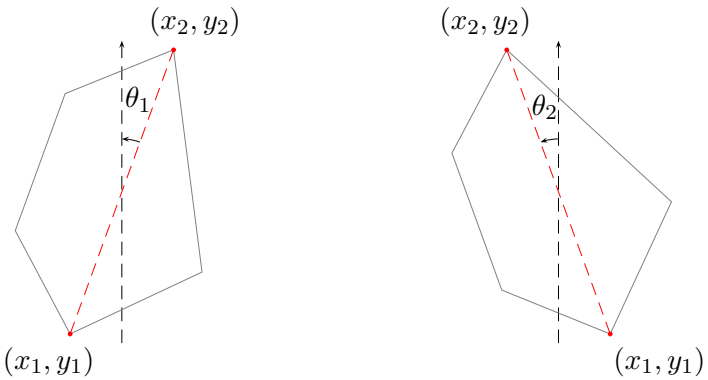

Figure 3: `major_angle_student` should measure the angle of the major axis relative to the vertical—in whatever direction will yield the smallest positive result; so, in this example, θ_1 and θ_2 would both be positive.

3. Write a function `major_angle_student` that accepts a 2-by-2 matrix `[x1 y1; x2 y2]` (i.e. one of the two results from `poly_axes_student`) and returns the angle of the major axis relative to the vertical (see Fig. 3).

   Hint: this will, of course, require some trigonometry; you may therefore find the `atan2` *function useful . . .*

4. Write a function that returns a list of the current running totals of a linked list (i.e. each element of the new list is the sum of all the elements before it in the old list). For example:

```
Original list: [8]->[3]->[4]->[1]
New list: [8]->[11]->[15]->[16]
```

5. Write a function that finds the average of all the non-negative numbers in a given linked list.