Robotics Final Project Report

Karassayev Magzhan
Aubakirov Sanzhar

February 27, 2013

Abstract

1 Image Preprocessing

The first task that we did is thresholding image. We have a colored image as an input and to calculate a path we need to deal with binary image. So firstly we converted the image to grayscale and then to binary. It is done in 2 classes: ThresholderSimple and ThresholderOtsu. The first class just does threshhold with some constant value. Value should be between 0 and 255. each point is true if average value of blue, green and red component is more than a treshold. The second classes uses the first one. First it calculate a theshold using Otsu method and pass a result to the first class.

Otsu method is: FIXME.

The interface has only one method:

public boolean[][] threshold(CImage ci)

In our case obstacles are black and background lighter so we need inverse our array of booleans. We use ImProcUtils.inversedThreshold for this. It just inverse every element of the given array.

The third operation is what it called dilation we named it as extendObstacles in ImProcUtils class. We used mask k*k. and a mask is a sphere which is described as

$x^2 + y^2 < r^2$

So the method tries to expand limits and if limits that it tries to expand is out of bound it ignores it. We need that technique to fill space in obstacles if any.

2 Graph Building, Preparing

There are a lot of pixel of obstacles. We take randomly first 1000 points and consider them as corners using function getFirstRandomPoints. Of course we could use Hough transform method to find real corners, but we believe that result would be almost the same or even better using random approach.

We also add 4 fictive points in each corner as corners into our coolection.

After that we trying to get all point which located exaclty between 2 corner points for each corner point crossproduct. So if we had 100 corner points we will get 10000 middle points. We use function pointCrossings for that.

These middle points are point we want to use to find the path we are looking for. But the resulting points in some places are very concentrated. That is why we need to reduce number of point but we need not to loose quality. The next function unconcentrateCrossings filter points in specific radius such that after applying this function there will be no two point such that distance between them less or equal than that specific radius.

To calculate distance between two points we always use euclidean distance:

$\sqrt{((x0 - x1)^2 + (y0 - y1)^2)}$

Each of the resulted points represented as tuple of 3 points: two obstacle point and a point exactly between them.

After we have unconcentrated points we wanted to exclude points on obstacles. filterBadCrossings filters points where bad crossing point is a point such that there are no obstacle point in the middle of the line between those two obstacle point. To find the line we use bresenham algorithm which is described in function: bresenhamLine.

3 Graph Building, Building

At this point we have point and we need a graph to find the path in.