Human Guided Machine Vision for Road Detection
Team AddNameHereWhenWeFinallyDecide

Kelsey DiPietro, Richard Frnka, Brian Hunter, Shaked K, Khanh Nguyen, Scott Spencer

Problem Statement

Detection of roads from an aerial view is an important problem, yet one with an elusive general solution. Road detection is essential to developing and maintaining a database of roads, especially with the increased use of GPS devices. Humans are particularly good at picking out roads from an image, but can only do so at a limited speed. Computers operate at the other end of the spectrum, lacking in a general algorithm to detect roads, but able to do so quickly. To get the best of both worlds, it is useful to combine a computer program with human input to fill in where a computer missed a road.

The task was to create a program which finds roads in an aerial image and contains a user interface for a human to highlight specific roads and identify any missed roads. All user interface was to be done in a graphical component which allows the user to easily click on the image and have the computer do the rest of filling in and highlighting.

Initial Setup

The project was divided into three subprojects: A pre-processing part which involved initially segmenting the image into candidate road component and an object detection part which involved ranking the components for their likeliness to be a road and finding obstructions that blocked view of the road. After pre-processing was complete, there was a graphical user interface piece which would allow a human user to select and highlight roads and identify any roads missed by the pre-processing.

All programming was done using MATLAB which has built in libraries for image segmentation and graphical user interface. Only simple images consisting of a single road with no cars were considered initially, but as the segmentation methods were developed, more complicated road configurations were used. Since no single algorithm performed well on all roads, many were considered, each with a unique segmentation process.

Segmentation Methods

Below is a list of the segmentation functions with pseudocode and a brief description:

basicSegmentation

J ← grayscale image
BW ← edge(J,canny)
se0 ← structural horizontal line of length 3
se90 ← structural vertical line of length 3
BW ← dilate(BW,[se0,se90])
BW2 ← ∼BW
BW, BW2 ← fillHoles(BW), fillHoles(BW2)
BW, BW2 ← BW, BW2 minus small pieces
CC1 ← components(BW)
CC2 ← components(BW2)
return CC1 + CC2

This method is a bareboned road finder. It finds the edges, thickens them, fills the holes, cleans up noise, and returns the components.

blurSegmentation

J ← grayscale image
se ← structural square
Io ← erode(J,se)
Io ← dilate(J,se)
Ie ← erode(Io,se)
Iobr ← reconstruct(Ie,∼J)
Iobrd ← dilate(Iobr, se)
blur1 ← reconstruct(∼Iobrd,∼Iobr)
blur2 = ∼blur1
fgm, fgm2 ← localmax(blur1), localmax(blur2)
fgm, fgm2 ← fillHoles(fgm), fillHoles(fgm2)
CC1, CC2 ← components(fgm), components(fgm2)
return CC1 + CC2

This method blurs the image a significant amount to create a more even color distribution then looks for local maximums. It fills in any holes that were not local maxima and returns the components of the corrected image.

colorBasedSeg

```
J ← grayscale image
x,y ← dimensions(J)
blank ← zeroMatrix(x,y)
se ← structural square
Io ← erode(J,se)
Io ← dilate(J,se)
Ie ← erode(Io,se)
Iobr ← reconstruct(Ie,∼J)
Iobrd ← dilate(Iobr, se)
Iobrcbr ← reconstruct(∼Iobrd,∼Iobr)
CC ← components(null)
for i in range 0 to 255:
        C ← Iobrcbr == i
        C ← fillHoles(C)
        C ← C minus small pieces
        C ← components(C)
        for j in range 1 to length(C):
                CC += C
        rof
rof
return CC
```

Similar to blurSegmentation, this method first blurs the image to get a more even color distribution. It then gets components by taking pixels of each color index, filling the holes, cleaning up noise, and merging them into one component list to return.

Connected_Comp_Edges

```
I ← grayscale image
gsI ← gaussFilter(I)
I2 ← edge(gsI,canny)
hy ← sobelFilter
Iy ← filter(I,hy)
Ix ← filter(I,hy')
se ← structural square
Io ← dilate(J,se)
Io ← erode(J,se)
Ie ← erode(Io,se)
Iobr ← reconstruct(Ie,∼J)
```

Iobrd ← dilate(Iobr, se)
Iobrcbr ← reconstruct(∼Iobrd,∼Iobr)
fgm ← localmax(Iobrcbr)
fgm ← edge(fgm,sobel)
fgm ← thicken(fgm)
se ← structural disk
cfgm ← erode(fgm,se)
cfgm ← dilate(cfgm,se)
inv ← ∼cfgm
CC ← components(inv)
return CC

crudeAFSegmentation

rgb ← color image
rgb ← gaussFilter(rgb)
I ← grayscale(rgb)
I ← thicken(I)
se ← structural square
I ← erode(I,se)
I ← dilate(I,se)
I ← ∼shrink(I)

ext_grad_seg

ibw ← grayscale image
se ← structural square
eg ← dilate(ibw,se) - ibw
rec ← reconstruct(eg,ibw)
f1 ← edge(rec,canny)
fgm ← localmax(rec)
fgm ← fgm minus small pieces
return components(fgm)

This method enhances external boundaries, darkens internal boundaries, and uses Canny edge detection to split the image up into components.

rgbSeg

```
J ← color image
x,y ← dimension(J)
apform ← L*a*b form of J
apd ← reshape apform to x*y by 2
cluster ← kmeans(apd) with k = 5
pixels ← reshape cluster to x by y
CC ← empty component list
for i in range 1 to 5:
        temp ← x by y zero matrix
        temp ← (pixels == i) minus small pieces
        CC += components(temp)
rof
return CC
```

This method takes an rgb image and segments it into 5 parts using k-means. It then finds components for each individual cluster and merges them together in a master list.

thinRoad

```
I ← grayscale image
I2 ← ∼I
x,y ← dimensions of I
BW1, BW2 ← I ¿ 150, I2 ¿ 150
BW1, BW2 ← fillHoles(BW1), fillHoles(BW2)
BW1, BW2 ← BW1, BW2 minus very small pieces
CC1, CC2 ← components(BW1), components(BW2)
return CC1 + CC2
```

This method segments a grayscale image and it's complement by separating by intensity and filling holes. It removes only very small components to accommodate for the small area of thin roads.

User Interface

Object and Road Detection

Code Call Chart

Future Work