Detecting Drivable Regions in Monocular Images

Michael Fürst

***Abstract*—Detecting drivable regions in monocular images seems to be a hard task. By looking at different approaches and analysing histograms we found a simple way to detect regions a robot may drive on, assuming there is another collision avoidance algorithm for non-planar objects.**

I. Introduction

This paper was written with the robotour in mind. Its task is quite easy to describe but not simple to accomplish.

The task for the robots is to deliver payload in given 1 hour time limit to destination as far as 1km. Robots must be fully autonomous, not leave the road and choose correct path on junctions. The place of start and destination will be the same for all robots. [1]

Detecting drivable regions in camera images helps to stay on the road. It may be used for other challenges than the robotour as well such as self driving cars outside of cities or on roads with no lane markers.

II. Problem

A. Detecting Drivable Regions

Given an image, taken by a camera mounted at the front of the robot, it should calculate which regions in that image are drivable.

The output of the algorithm does not need to be pixel perfect. It is required that the information the algorithm outputs can be used to calculate the driving direction of the robot.

Avoidance of obstacles like cars, pedestrians, walls or other large objects is no requirement. It is assumed that there exist other sensors like Lidar which are better for those tasks.

B. Correctness of Current State

Many algorithms assume that the robot is in a correct state. This means the robot is on the road with all wheels or legs or other parts of its body. Furthermore, this means that the robot also looks at the road and knows a reference region on the image that is road for sure.

In reality of robotics your robot somehow does get into invalid states and you need to recover from them. So this is another requirement of this algorithm to even work when the robot itself is in an invalid state and does not know where a reference road is.

C. Images of Roads

The images try to give an overview over the situation. Image 1 shows a park in Pisek where you can clearly see drivable regions. Image 2 shows a road in Pisek where detecting drivable regions should work too. The last figure 3 is a detailed example transition between allowed and unallowed regions.

Fig. 1: Image of a park in Pisek.

Fig. 2: Image of a street in Pisek.

III. Solutions

A. Histogram Backprojection

This solution uses backprojection [2]. Backprojection is used to find objects in images by calculating the objects histogram and than backprojecting it on the image resulting in a black and white image representing probabilities for that pixel belonging to the object or not.

This method assumes that you have a reference image of an allowed region. Whereas you usually do not have one image that does represent all reference images well enough it is recommended to either apply all reference images on the given image or to define a reference region inside the image that should be tested.

Fig. 3: Close up of a border between grass and a valid region.

Each method has some problems. The method with testing all reference images is too slow for fast driving robots. Having less than 5 FPS on a robot moving with about 1 m/s or even more is not acceptable. If you decide to use a reference region you violate the assumption that the robot does get into invalid states and that it has to recover from them.

When ignoring those 2 constraints backprojection is an ideal solution (see image 4). However, in competitions you need to take these 2 problems serious or you will fail, because it only works in too few conditions.

Fig. 4: Backprojection of a reference region (marked in red on the left) into the image (left). The resulting probabilities in grayscale (right).

Implementing the reference region backprojection in python is easy.

```
from cv2 import *
import numpy as np

# Setup the variables.
width = 320
height = 240
dims = (width, height)
blur = 10
refWidth = 40
refHeight = 40
b = [5, 5]  # buckets for hn and sn
c = [0, 1]  # channels
r = [0, 180, 0, 256]  #ranges

# Prepare the image
img = resize(img, dims, 0, 0, INTER_CUBIC)
img = medianBlur(img, blur * 2 + 1)
img = cvtColor(img, COLOR_BGR2HSV)

# Set value to 127 for debug image.
# [...]

# Reference region mask.
mask = np.zeros(img.shape[:2], np.uint8)
left = int(width / 2 - refWidth / 2.0)
right = int(width / 2 + refWidth / 2.0)
top = int(height - refHeight)
bottom = int(height)
mask[top:bottom, left:right] = 255

# Calculate the histogram and backproject.
hist = calcHist([img], c, mask, b, r)
probs = calcBackProject([img], c, hist, r, 1.0)
```

B. Thresholding

Because the first solution presented had some problems in the real world. A faster and more robust approach is required. Finding a faster solution means looking at the simple solutions again, whereas they usually need low computation power and therefore run at up to 60 FPS. Thresholding seems to be a good candidate.

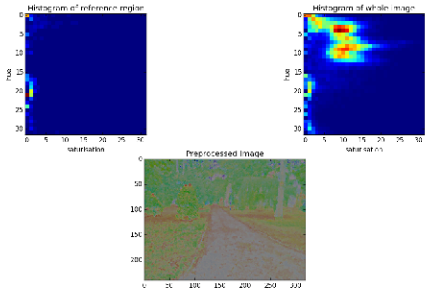

Fig. 5: The histogram analysis for a park image.

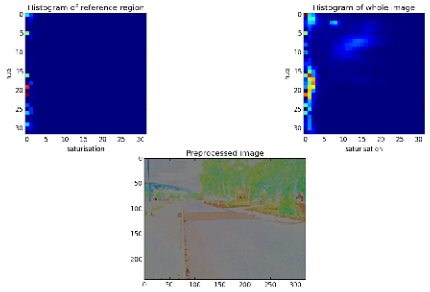

Fig. 6: The histogram analysis for a street image.

1) Analyzing Histograms: For good thresholding there is a good threshold required. Finding a good threshold can be a really tricky problem and sometimes it is impossible to find a threshold that solves the problem at all.

Analysis of histograms usually gives more insight. Analysing a lot of test data having good reference regions in them we realized that all histograms have something in

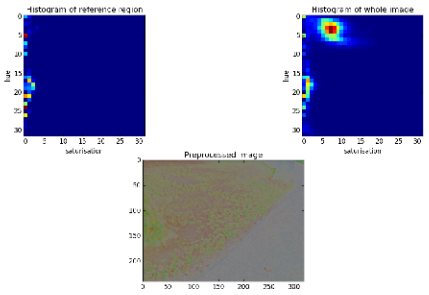

Fig. 7: The histogram analysis for a region transition.

common and that detecting drivable regions can be done pretty simple.

Pavements have a low saturation and tend to be not so greenish (see image 5, 6 and 7).

For further improvements preprocessing is applied to suppress shadows in the image feed to the thresholding algorithm. The preprocessing blurs the image, sets the value channel of the HSV image to 127. The latter is only for debugging visualisation.

2) Applying the threshold: Applying a threshold is simple in opencv.

```
from cv2 import *
import numpy as np

# Setup the variables.
width = 320
height = 240
dims = (width, height)
blur = 10

# Prepare the image.
img = resize(img, dims, 0, 0, INTER_CUBIC)
img = medianBlur(img, blur * 2 + 1)
img = cvtColor(img, COLOR_BGR2HSV)

# Set value to 127 for debug image.
# [...]

# Apply the threshold.
l = (0, 0, 0)
u = (255, 40, 255)
lower = np.array(l, dtype=np.uint8, ndmin=1)
upper = np.array(u, dtype=np.uint8, ndmin=1)
img = inRange(img, lower, upper)
```

Looking at the histograms and doing some testing, we discovered that with our camera a saturation range of (0,40) and no restrictions to hue and value are perfect (see 8, 9 and 10).

However with this method you have to watch out and test it in every new environment where you place your robot. And changing the camera might mean that you need new threshold.

IV. Conclusions

The images produced by the algorithms presented produce quite good images but have some issues. However there are more problems than the ones already described. Shadows

Fig. 8: Applying a ≤ 40 saturation threshold on a park image.

Fig. 9: Applying a ≤ 40 saturation threshold on a street image.

Fig. 10: Applying a ≤ 40 saturation threshold on a transition situation.

and small disturbances in the image can confuse algorithms working ontop of the image recognition.

To reduce these errors it is recommended to sum each column in the image. This represents the probability that a direction that is represented by a column in the image is drivable. This also means that you have fewer values to take into account for higher level algorithms making them a little bit faster.

V. KaMaRo-Engineering

This paper was research done in the context of KaMaRo-Engineering. For more information about the mission of KaMaRo-Engineering visit our website at: http://wordpress.kamaro-engineering.de/

References

[1] Robotour 2015. http://robotika.cz/competitions/robotour/2015/en. Accessed: 2015-11-17.

[2] Dana H. Ballard Michael J. Swain. Indexing via color histograms. *Proceedings, Third International Conference on Computer Vision*, pages 390 – 393, dec 1990.