Disparity Image Segmentation For ADAS

Viktor Mukha & Inon Sharony, L4B Automotive

2018-06-05

Contents

1 Abstract 1

2 Introduction 1
  2.1 Disparity image segmentation 1
  2.2 Previous work 2
  2.3 Our attempts 2

3 Solution 3
  3.1 Algorithm: Adaptive Non-binary CCL 3
  3.2 Implementation 3
  3.3 Parameters 4
    3.3.1 Definitions 4

4 Results 5

5 Conclusion 6
  5.1 Future Work 6

6 References 7

1 Abstract

We present a simple solution for segmenting grayscale images using existing Connected Component Labeling (CCL) algorithms (which are generally applied to binary images), which was efficient enough to be implemented in a constrained (embedded automotive) architecture. Our solution customizes the region growing and merging approach, and is primarily targeted for stereoscopic disparity images where nearer objects carry more relevance. We provide results from a standard OpenCV implementation for some basic cases and an image from the Tsukuba stereo-pair dataset.

2 Introduction

In various computer vision contexts, we are required to perform Blob Extraction or Image Segmentation on an image. This image may be binary (i.e. each pixel either black or white), or not. ADAS usually entails some sort of sensor fusion, with two common sensors being thermal imaging and stereoscopic cameras. Both these sensors output false “color” grayscale images, in which pixel value expresses either the temperature or parallax (also known as disparity), respectively.
In the following we will give examples from Image Segmentation of a Disparity Image. The higher the disparity value, the closer the object. An object at an infinite distance away from the camera will have a disparity value of zero. In the context of Advanced Driver Assistance Systems (ADAS), we use disparity image segmentation to detect objects in the vehicle’s path, primarily of interest are the closer objects (this consideration will guide us throughout the following). In thermal imaging we are also interested only in foreground objects, but can bring more domain knowledge to bear in thresholding a certain part of the temperature spectrum (in contrast, we don’t know the relevant disparity values in advance).

2.1 Disparity image segmentation

The task of image segmentation in our context is to extract the blobs from a post-processed disparity image. Each blob is described using a bounding box, a rectangle tightly encompassing the blob. Additionally, this module determines the maximum disparity value in the bounding box and assigns it to the object description struct.

It is hard to reliably estimate the “thickness” of an object from stereo-derived depth maps, so we should rather concentrate on the most important distance for us: the closest distance to the object. If the maximum value is found too rough an estimate, we should throw away outliers by doing something similar to median filtering, but only in order to find the value of interest. That is, we don’t need to generate a full matrix of disparity values for each rectangle. Basically, this is a direct estimation of the distance to the object, so this will be thoroughly investigated elsewhere.

The algorithm sorts the results in the order of decreasing disparity value, since we are mostly interested in the closer objects.

2.2 Previous work

CCL is well researched Image Segmentation method, with many efficient implementations. However, the vast majority of Image Segmentation algorithms such as Connected Component Labeling (CCL) only work on a binary image, while a disparity image is a grayscale image.

There is one work which has shown that it is possible to extend different CCL algorithms to grayscale images, by Yapa & Harada. However the implementation is not publicly available, and these ideas have been implemented in MATLAB and are lacking proper evaluation for embedded systems. Therefore, we have decided to derive a more portable solution.

2.3 Our attempts

The first try was to use the cv::SimpleBlobDetector class, which is the only blob extraction algorithm in OpenCV which works on grayscale images instead of binary images. It uses multiple thresholds for generating binary images and then applies the cv::findContours() algorithm to binary images. cv::findContours(), in turn, uses a border following algorithm by Suzuki & Abe.

The results of directly applying cv::SimpleBlobDetector to the industry-standard Tsukuba ground truth disparity images were not satisfactory. The following are results for parameters which allow all found keypoints to be shown. On the left is the original disparity image, and on the right is a negated image (since by default the algorithm looks for dark circles in bright image):
Tweaking all parameters only filtered some of these keypoints away and did not help finding other keypoints. Thus, we decided to implement our own multiple-threshold algorithm using OpenCV’s existing CCL implementations.

3 Solution

The idea behind the algorithm is fairly simple: We apply a binarizing threshold, run CCL, extract objects, and repeat this with a lower threshold, until objects start to merge or we do not detect any objects anymore. We decided not to use the histogram of the grayscale image since we are interested in the the nearest objects more than the shape of the global histogram.

3.1 Algorithm: Adaptive Non-binary CCL

1. Determine the range of disparity in the input disparity image using cv::minMaxLoc and take thresholdStepSize (percent) of it as a threshold step.
2. Initialize an empty storedObjects vector.
3. Apply a binary threshold using cv::threshold to get a binary image for CCL.
4. Run CCL with this binary image.
5. Update the vector of stored objects with the newly detected objects using the following criteria (in this order):
   - if the new object is almost fully inside the bounding box of any stored object, discard it;
   - if the new object does not intersect any of the stored objects, add it;
   - if the new object contains the centers of more than one stored objects, this means two stored objects have merged, so discard this new object;
   - if an old object is almost fully inside of any new object and the center of stored object is inside it too, update the stored object’s bounding box and center with those from the new object.

3.2 Implementation

The algorithm is implemented as a single pass over the threshold values, then a single pass over any new objects each of which is running over all stored objects to check the criteria. When checking for a merge, complexity is added since we need to check if any other stored object’s center is inside of the new object. This adds another pass over the stored objects.

The main factors of complexity are therefore the number of objects we want to detect and the stopping criteria.
The “blobs” output is written to `std::vector<DetectedObjects<int>>`. Basically, a DetectedObject contains the 2D coordinates of the center, the disparity value, and a bounding box (`cv::Rect`). Please note that `cv::Rect<unsigned short>` is ill-defined for images with a dimension larger than `ushort` pixels, since the intersection operator `&` can overflow the buffer. This is the reason for choosing the `int` data type.

The implemented algorithm is iterative in nature because it considers objects from each of the previous steps. Under some circumstances (zero value of the `numSameIterationsToStop` parameter, in the following) we can parallelize the CCL computation using, e.g., `cv::parallel_for()`.

For the rest of the algorithm logic, we would still need to process all objects detected by CCL sequentially, so before parallelizing, it is important to profile how much time running CCL takes, and how much time the logic takes. If both are taking too much time, it might be better to opt for another approach from Yapa & Harada.

### 3.3 Parameters

| Position | Name                          | Type       | Domain                | Default |
|----------|-------------------------------|------------|-----------------------|---------|
| 1        | thresholdStepSize             | float      | [0, 1]                | 0.05    |
| 2        | numSameIterationsToStop       | uchar      | [0, 2^8)              | 0       |
| 3        | minObjDimension               | ushort     | [0, max(width, height) − 1] | 10      |
| 4        | maxObjDimension               | ushort     | [0, max(width, height) − 1] | 400     |
| 5        | commonAreaToConsiderBackground| float      | [0, 1]                | 0.9     |
| 6        | commonAreaToConsiderGrowing   | float      | [0, 1]                | 0.9     |

The rest of the parameters are the arguments to `cv::connectedComponentsWithStats()` CCL method:

| Position | Name               | Type       | Domain                          | Default          |
|----------|--------------------|------------|---------------------------------|------------------|
| 7        | connectivity       | int        | 8 or 4                          | 8                |
| 8        | ltype              | int        | CV_32SC1 or CV_16UC1            | CV_16UC1         |
| 9        | ccltype            | int        | see the `cv::ConnectedComponentsAlgorithmsTypes` | CCL_GRANA        |

#### 3.3.1 Definitions

1. **thresholdStepSize**: The algorithm proceeds in descending order of disparity values moving the threshold by this step size. The default is 5% of the disparity range (maximum disparity minus minimum disparity). Smaller steps help to separate objects which are close in depth to each other, bigger steps save calculation time since this parameter directly determines the number of CCL runs.
2. **numSameIterationsToStop**: This is a stop criteria of the algorithm. The default value of 0 lets the algorithm go over the complete range of disparity values, which is slow, and will only help to detect objects with lower disparity, which are probably not that important in our context. A positive-definite value will stop if nothing changed in the last `numSameIterationsToStop` steps. This parameter is to be adjusted together with `thresholdStepSize` if the algorithm takes too long to go over all thresholds.
3. **minObjDimension**: Minimal object dimension in pixels. Both height and width of the detected blob must satisfy this criteria for an object to be stored in the list of detected objects. Note that very small dimensions will allow the algorithm to accept noise as objects.
4. **maxObjDimension**: Maximal object dimension in pixels. Both height and width of a detected blob must satisfy this criteria for an object to be stored in the list of detected objects. Note that setting this parameter too low will make the detection of objects which are close to the cameras impossible. The default value is one less than the minimal frame dimension (height or width), so that the complete frame will not be recognized as an object.
5. **commonAreaToConsiderBackground**: Criteria for discarding a *newly detected* object if it is at least \( \text{commonAreaToConsiderBackground} \) (ratio) inside of any *stored* object. In most cases, such a detected object is behind an already detected object. This parameter is only needed for compensating for slightly shifted object detection.

6. **commonAreaToConsiderGrowing**: Criteria for updating a *stored* object if it is at least \( \text{commonAreaToConsiderGrowing} \) (ratio) inside of a *newly detected* object. This means that the object is growing. This parameter is only needed for compensating for slightly shifted object detection.

7. **connectivity**: 8-way or 4-way connectivity, respectively.

8. **ltype**: Output image label type. Currently \texttt{CV\_32S} and \texttt{CV\_16U} are supported.

9. **ccltype**: Connected components algorithm type.

### 4 Results

We have implemented four unit test cases for segmentation. The following are the resulting images with rectangles around detected objects in corresponding order:

| Test case | Description                                      | Result   | Evaluation                  |
|-----------|--------------------------------------------------|----------|-----------------------------|
| 1         | 3 filled rectangles of different disparity values, and 1 black rectangle (faking failed disparity) | ![Image](image1.png) | 3 objects are recognized correctly |
| 2         | 3 filled circles                                 | ![Image](image2.png) | 3 objects are recognized correctly |
| Test case | Description | Result | Evaluation |
|-----------|-------------|--------|------------|
| 3         | Disparity Ground Truth image from Tsukuba dataset | Objects are recognized, the bust is recognized as 3 separate objects with different disparity values since it actually has some volume, and parts of it are deeper in the image. For now, the criteria of passing this test is that we detected between 1 and 10 objects. | |
| 4         | A real calculated disparity image. It is noisier, so we set the minimum object dimension to 40px for this image. | The amount of detected blobs and their bounding boxes are adequate. For now, the criteria of passing this test is that we detected between 1 and 100 objects. | |

## 5 Conclusion

We have demonstrated an adaptive algorithm for performing Image Segmentation on a non-binary image, such as a false-color (grayscale) disparity image, using CCL. Our initial results show potential, especially considering that they are portable to constrained systems such as embedded ADAS. The current approach yields qualitatively better results than OpenCV’s `cv::SimpleBlobDetector`, which are sensitive.

### 5.1 Future Work

1. Better control using the stop criteria. For instance, stopping when we have found $N$ closest objects and none of them has “grown” in last $M$ frames. This way we can balance between performance and the
number of detected objects.

2. Implement Yapa & Harada’s proposal for at least one of the three mentioned CCL algorithms, and benchmark the quality and speed.

3. Consider alternatives to CCL for segmentation (Watershed, Region Growing, Clustering, etc.).

4. Improve the object-depth estimation algorithm beyond zeroth order (i.e. taking the maximum disparity value in the bounding box).

6 References

1. Lifeng He; Xiwei Ren; Qihang Gao; Xiao Zhao; Bin Yao; Yuyan Chao, “The connected-component labeling problem: A review of state-of-the-art algorithms,” Pattern Recognition, Elsevier BV, ISSN: 0031-3203, Vol: 70, Page: 25-43 (2017), DOI: 10.1016/j.patcog.2017.04.018

2. Martin Peris Martorell, Atsuto Maki, Sarah Martull, Yasuhiro Ohkawa, Kazuhiro Fukui, “Towards a Simulation Driven Stereo Vision System”. ICPR2012, pp.1038-1042, 2012. Sarah Martull, Martin Peris Martorell, Kazuhiro Fukui, “Realistic CG Stereo Image Dataset with Ground Truth Disparity Maps”, ICPR2012 workshop TrakMark2012, pp.40-42, 2012.

3. Yapa, R.D. and Harada, K., 2008. “Connected component labeling algorithms for gray-scale images and evaluation of performance using digital mammograms,” International Journal of Computer Science and Network Security, 8(6), pp.33-41.

4. Suzuki, S. and Abe, K., “Topological Structural Analysis of Digitized Binary Images by Border Following,” CVGIP 30 1, pp 32-46 (1985)

5. Nobuyuki Otsu, “A Threshold Selection Method from Gray-Level Histograms,”IEEE Transactions on Systems, Man, and Cybernetics, Volume 9, Issue 1, pp. 62 - 66 (Jan. 1979), DOI: 10.1109/TSMC.1979.4310076

6. Grana, Costantino; Bolelli, Federico; Baraldi, Lorenzo; Vezzani, Roberto “YACCLAB - Yet Another Connected Components Labeling Benchmark” Proceedings of the 23rd International Conference on Pattern Recognition, Cancun, Mexico, pp. 3109 -3114 , Dec 4-8, 2016, DOI: 10.1109/ICPR.2016.7900112