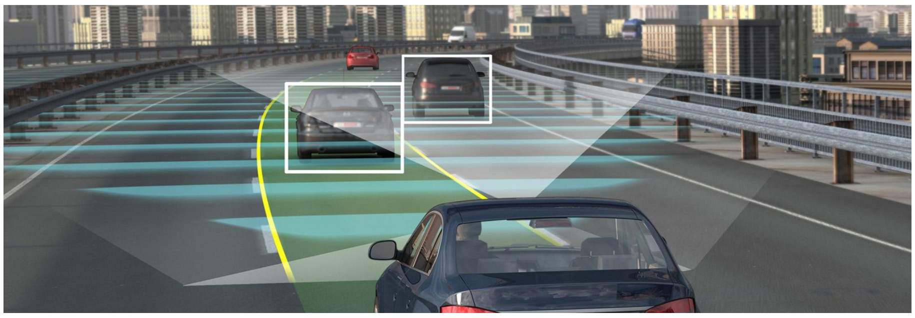
Autonomous Vehicles: Integrating Object Detection and Distance Ranging



Background

Autonomous road vehicles and advanced driver assistance systems are fast becoming a reality. Computer Vision is increasingly being used to allow such vehicles to understand the road environment around them based on imagery from on-board forward facing cameras.

In this assignment we are dealing with the automatic detection of objects, and the estimation of their distance from the vehicle (i.e. ranging), within stereo video imagery from an on-board forward facing stereo camera. This can be performed by integrating the use of depth (disparity) information recovered from an existing stereo vision algorithm with one or more object detection algorithms. Knowledge of the distance of objects that have the potential to move within the scene (i.e. dynamic objects, such as pedestrians/vehicles) assists both automatic forward motion planning and collision avoidance within the overall autonomous control system of the vehicle.

The low cost and high granularity (i.e. full-scene) 3D information available from stereo vision means that classification (i.e. type) and distance of objects in front of the vehicle the vehicle can be detected more readily than with radar or lidar (laser) sensing technologies.

As such, we have a set of still image pairs (left and right) extracted from on-board forward facing stereo video footage under varying illumination conditions and driving conditions. Your task is to design and prototype a computer vision system to detect dynamic objects and their distance from the vehicle at any given point in the journey. You will develop this prototype system using Python with the OpenCV library and the techniques covered in the module.

This is a real-world task, comprising a real-world image set. As such, this is an open-ended challenge type task to which a perfect solution that works perfectly over all the images in the provided data set may not be possible.

Task Specification – *Object Detection and Distance Ranging*

You are required to develop to an object detection system that correctly detects one or more types of dynamic objects within the scene in-front of the vehicle and estimates the range (distance in metres) to those objects.

In constructing your solution you may wish to consider the following aspects of your design:

- **the use of one or more object detection approaches** covered in lectures for the detection of pedestrians, and possibly other, types of dynamic object within the scene.
- exploring **the use of additional image pre-processing to improve the contrast** of the grayscale images used as an input to both object detection and stereo vision disparity (depth) estimation.
- **selection of a region of interest**, possibly adaptively, within the image (including possibly

areas of road, pavement, other or not) that represents the region directly in-front of the vehicle where such dynamic objects are likely to occur.

- the **use of selective search object bounding box identification**, within this region of interest, in place of exhaustive search for objects using a sliding window based approach.

Your solution must make use of a recognised object detection approach, ideally comparing multiple approaches or variants, and use stereo disparity (depth) information to provide ranging information of one or more types of dynamic objects within the scene. For the avoidance of doubt, *limited* credit will be given for a solution based only on the existing built-in pedestrian capabilities of OpenCV (although these can be used as part of a wider overall detection solution) or that does not provide object range (distance).

Additionally, some example images may not have significant noise-free disparity (depth) available for scene objects which may also be partially occluded. The road scene itself will change in terrain type, illumination conditions, clutter and road markings – ideally your solution should be able to cope with all of these. All examples will contain a clear front facing view of the road in front of the vehicle only – your system should report all appropriate objects instances it can detect *recognising this may not be possible for all cases within the data set provided.*

Initially you are advised to **target one type (class) of dynamic object – pedestrians, with later extension to other types** (vehicles, …) as time allows and with consideration of the available credit in the marking scheme provided.

As this is only a prototype – efficiency of your approach is less important than performance.

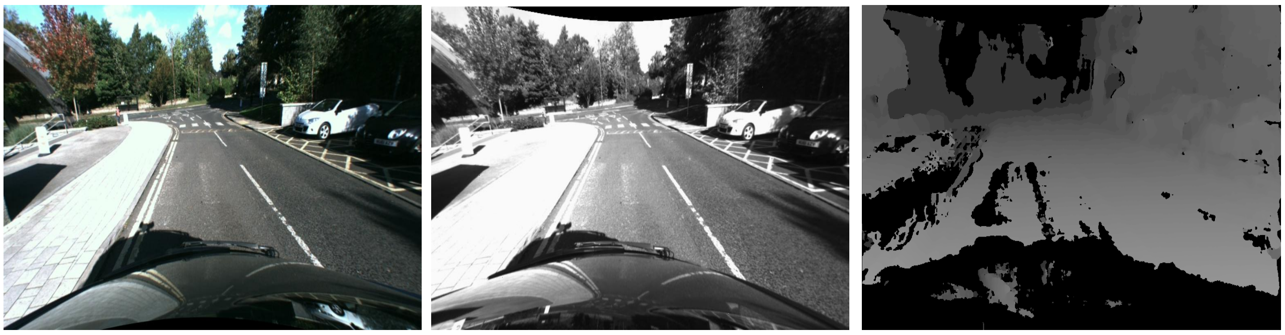


Figure 1: Example left (colour), right (greyscale, rectified) and corresponding disparity calculated using the example python code provided for the assignment.

Additional Program Specifications

Additionally, to facilitate easy testing, your prototype program **must** meet the following **functional requirements:**

- Your program must contain an obvious variable setting in the top of the main code file that allows a directory containing images to be specified. e.g.

  ```
  master_path_to_dataset = "TTBB-durham-02-10-17-sub10"
  ```

  from which it will cycle through each stereo pair in turn processing it for object detection and distance ranging prior to displaying it. A basic example (*stereo_disparity.py*) for cycling through the data set of images and computing the stereo disparity is provided.

- When dynamic objects, such as pedestrians and/or vehicles, are detected within a scene your solution must display a coloured polygon on the left (colour) image highlighting where the object is and also a distance estimate to the object obtained from the corresponding stereo depth information of the scene (see example in Figure 2).

- Furthermore, for each image file it encounters in the directory listing it must display the following to standard output:

```
filename_L.png
filename_R.png : nearest detected scene object (X.Xm)
```

  where "filename" is the current image filename and *X.X* is the distance in metres to the current nearest dynamic scene object detected within the scene. When no objects can be detected, output a zero distance for dynamic objects. *Your final program must run through all the files as a "batch" without requiring a user key press or similar.*

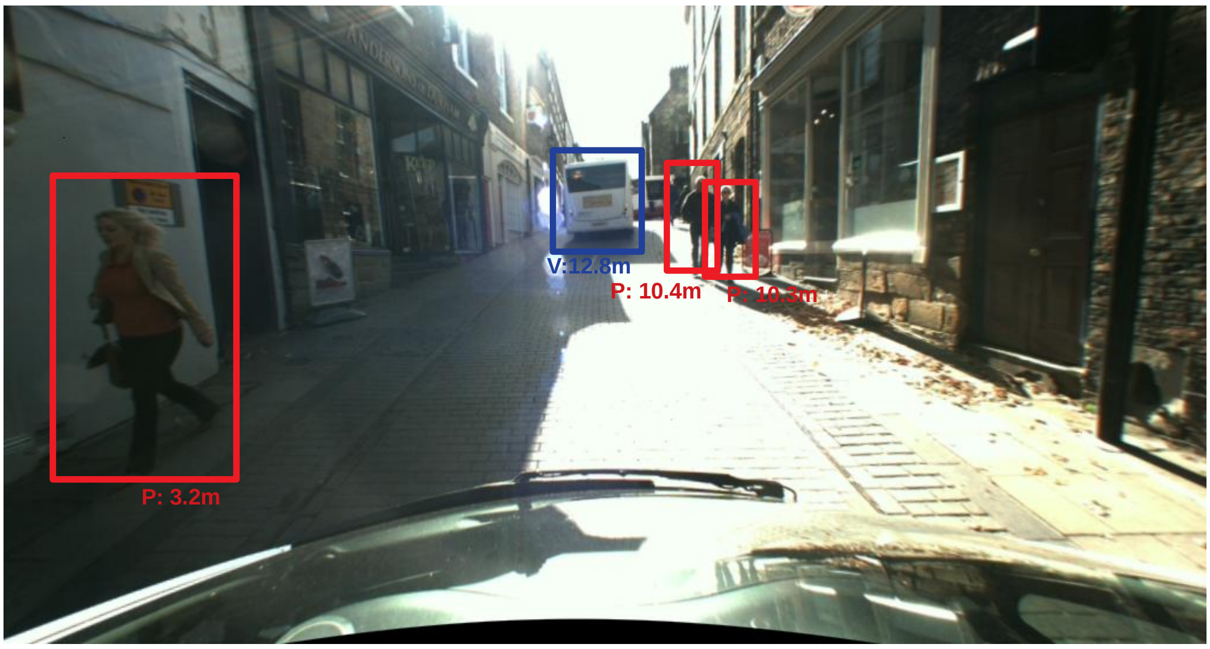


Figure 2: Illustrative polygon outline of the detected scene objects, with distance displayed and abbreviated class label inset, drawn on the left (colour) image.

- You may use any heuristics you wish to aid/filter/adjust the performance.
- Your approach must combine the use of object detection and stereo vision based ranging.
- Your program must compile and work with **OpenCV 3.4.x** on the lab PCs.

Sample Data & Example Software

The sample data provided is a set of 1449 sequential still image stereo pairs extracted from on-board stereo camera video footage (see example – Figure 1). These images have been rectified based on the camera calibration and you do not need to perform stereo calibration yourself.
The full set of images is available as a single ZIP file from DUO as follows:

TTBB-durham-02-10-17-sub10.zip
Be aware that this data set is still large! (~2Gb, this is the nature of this business).

Furthermore a training/testing dataset for pedestrian detection is provided as follows:

INRIAPerson-DU.zip *(560Mb)*

Two sets of example python scripts are also provided as a starting point as follows:

- *stereo_disparity.py* – cycles through the stereo dataset *(TTBB-durham-02-10-17-sub10)* and calculates the disparity from the left and right stereo images provided (lecture 4)
- *stereo_to_3d.py* – projects a single example stereo pair to a 3D in order to show how to obtain 3D distance information for the scene, write a point cloud of this data to file for reference and shows an example back-projection from 3D to the 2D image (lecture 4)

  Available from - https://github.com/tobybreckon/stereo-disparity
- *hog_{train|test|detector}.py* – a set of training/testing/detector scripts for the detection of pedestrians using a combination of HOG features of SVM classification (lecture 3).
- *bow_{train|test|detector}.py* – a set of training/testing scripts for the detection of pedestrians using a combination of Bag of Words (BoW) features of SVM classification (lecture 5/6).
- *selective_search.py* - an example implementation of using a selective search approach to identify possible object locations prior to classification of these scene regions (lecture 3).

  These latter examples all operate on the pedestrian detection dataset (*INRIAPerson)* by default. Available from - https://github.com/tobybreckon/python-bow-hog-object-detection

Marks

The marks for this assignment will be awarded as follows:

- Overall design and implementation of your solution including aspects of:
  - any image pre-filtering performed (or similar first stage processing)
  - choice of object detection methodology (including search strategy)
  - effective integration of object range estimation from stereo vision 30%
- General performance on object detection and distance ranging **
  (taking into account accuracy, false detection, missed detection, failures etc.) 30%
- Clear, well documented program source code 10%
- Report:
  - Discussion / detail of solution design and choices made 5%
  - *Statistical* evidence of the performance of system at the task 5%
- Additional credit will be given for one or more of the following:
  - extension of object detection to more than one type of object (people, vehicles …)
  - comparison of one or more object detection methodologies (or variants)
  - automatic parameter adjustment based on some form of preliminary analysis image
  - the successful use of any heuristics to speed up processing times or improve overall detection performance.

  (for any of the above up to a maximum, dependent on quality) 20%

Total: 100%

*[** as supporting evidence for this part* ***you are required to submit a video file of your system in operation*** *over a sample of the data – make sure your video shows both the colour and disparity images. This can be constructed using OpenCV directly or any tool of your choice. File size must be less than 10Mb in size, video format in use must playback in the VLC tool –* https://www.videolan.org/*]*

Submission :

You must submit the following:

- Full program **source code together with any required classifier/dictionary files** for your final solution to the above task as a **working python script** meeting the above ***"additional program specifications"*** for testing. Include all supporting python files and clear instructions (*e.g. in a README.txt)* on how to run it on the stereo dataset *(TTBB... images).*

- **Example video file** showing general performance on some of the example data (see above)

- **Report (max. 750 words)** detailing your approach to the problem and the success of your solution in the task specified. Provide any illustrative images (as many as you feel necessary) of the intermediate results of the system you produc*e (overlays, results of processing stages etc.) Remember that any titles, captions, tables, references, and graphs do not count towards the total word count of the report.*

  Summarise the success of your system in detecting and range estimating scene objecte in the data set. Submit this as a PDF (not in any other format).

Make it clear in the initial comments of your source code how to run your Python script. Your executable must run on one of the lab based PCs (either on Windows or Linux via OpenCV 3.4.x) – ensure compatibility before submission.

Plagiarism: You must not plagiarise your work. You may use program source code from the provided course examples, the OpenCV library itself or any other source BUT this usage must be acknowledged in the comments of your submitted file. Automated software tools (e.g. https://theory.stanford.edu/~aiken/moss/) may be used to initially detect cases of potential source code plagiarism in this practical exercise which will include automatic comparison against code from previous year groups. Attempts to hide plagiarism by simply changing comments/variable names will be detected.

You should have been made aware of the Durham University policy on plagiarism. Anyone unclear on this must consult the course lecturer prior to submission of this practical.

To submit your work create a directory named as your username (e.g. *cxfh123*). Place all required files in this directory using, **ZIP** compress/archive this entire directory structure (not rar or .z7 or anything else please - as this breaks the automated extract/test tools) and submit it via DUO *(late submissions will be penalised following departmental policy).*

Submission Deadline:– 2pm (UK time) on 7th December 2018