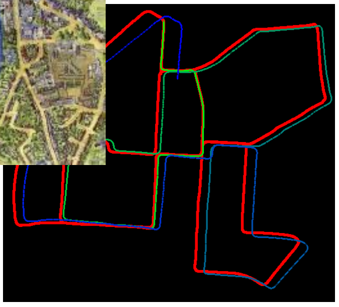
L4 Assignment – SSA – Advanced Computer Vision

Using Object Detection with Visual Odometry for Future Autonomous Vehicle Guidance



Background

Visual odometry has many potential applications for future autonomous road vehicles, robotic sensing and even autonomous drone applications. Key comparators for the various available visual odometry approaches are their robustness to varying environmental conditions and the accuracy of the vehicle motion recovered via the technique – especially when other objects in the environment are moving also. This assignment will perform a practical investigation of these aspects.

This is a real-world task, operating at or very near to the current boundaries of research in the area based both on recent research and leveraging the power of existing open source software implementations in this field that aid rapid prototyping and development.

Task Specification – *Comparing Visual Odometry Approaches*

You are required to develop a working prototype of **one variant** of the visual odometry approaches outlined in lectures and compare the effect of using/ignoring features from dynamic (moving scene objects) providing a summary of their performance using a specified dataset provided to you. This comparison can be carried out using any form of qualitative or quantitative measure (ideally both).

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

- *Available open-source implementations* – a number of OpenCV Python visual odometry solutions exist on-line (e.g. *on github – [1], [2])*. You may use (and of course *fully acknowledge*) any part of these solutions within the task but will need to understand and adapt them to the task in-hand (and the available data set).

  Make full use of these and do not re-invent the wheel – but avoid *"look no hands"* use.

- *Existing OpenCV components* – the OpenCV library contains dedicated routines for the detection and matching of feature points (see lab session on *mosaicking*), the tracking of features between image frames and the calculation of the Essential matrix, E, between camera positions. The outline provided in [3] and [4] may also be useful to assist here.

- *Advanced Odometry Features* – feature binning, frame selection, N-point VO, optimization.

- *Comparison against Ground Truth* – noisy ground truth information is available from on-board GPS (latitude, longitude, altitude) and IMU (acceleration, velocity, compass) sensors.

You may carry out your comparison using either monocular or stereo visual odometry.

For your chosen visual odometry approach, compare its operation by using/ignoring features that correspond to dynamic objects (e.g. vehicles/pedestrians) – this can be done by integrating an existing object detection approach (for which an example is provided – *yolo.py*) and ignoring features in the regions where objects are detected (for which the built in *"point in polygon"* test – *cv2.pointPolygonTest()* may also be useful).

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-sub5"
  ```
  from which it will cycle through either mono or stereo imagery.

- Your program must work with **OpenCV 4.x** on the lab PCs.

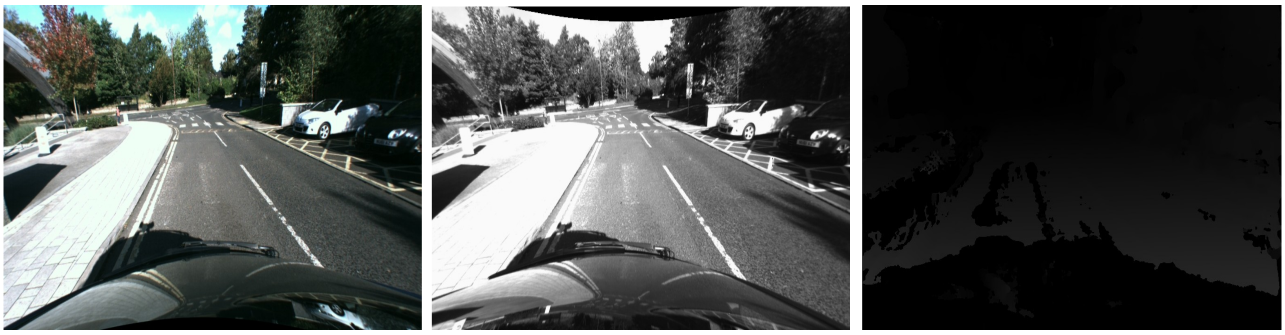


Figure 1: Example left (colour), right (greyscale, rectified) and corresponding disparity calculated using the example python code provided for the assignment.

The sample data provided is a set of 2898 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.

  TTBB-durham-02-10-17-sub5.zip
  Be aware that this data set is still large! (~4Gb, this is the nature of this business).

  Ground Truth sensor data is available (one set of measurements per image) as two simple CSV format files: ***GPS.csv, IMU.csv*** *(inside main dataset zip file)*

  [This dataset is at a subsampling of 5 – i.e. every 5th frame from the original sequence – on DUO you will also find a link for one with a subsampling of 2 – i.e. every 2nd frame from the original sequence – which you may also use instead. This second dataset is ~9.4Gb.]

  N.B. these datasets differ from that used in L3 that had a subsampling of 10 (...sub10.zip).

A set of example python scripts are also provided as follows:

- *stereo_disparity.py* – cycles through the dataset and calculates the disparity from the left and right stereo images provided.
- *stereo_to_3d.py* – projects a single example stereo pair to a 3D, writes a point cloud of this data to file for reference and shows an example back-projection from 3D to the 2D image.
- *mono_stream.py*– cycles through single image set (monocular), provides a full set of camera parameters and reads/displays the corresponding GPS + IMU information (using *gyro.py*).

  All available in: https://github.com/tobybreckon/stereo-disparity

- *yolo.py* – an example object detection approach which you can use *"out of the box"* for your object detector for the purposes of this assignment (at the moment this can be treated as a black box detection component, although the more details will be taught in lectures 9/10 and the operation of this component relates heavily to the deep learning content of L3 SSA).

  All available from - https://github.com/tobybreckon/python-examples-cv

Marks

The marks for this practical will be awarded as follows:

- Overall design and implementation of your solution including aspects of:
  - use feature detection (or similar first stage processing)
  - feature selection and Essential matrix calculation/estimation
  - effective re-use of existing open source software / components *(as appropriate)*
  - use of ground truth information to provide performance evaluation 30%

- Comparative performance using/ignoring features that correspond to dynamic objects in the scene** (quality of position estimation, robustness to scene conditions)
  30%

- Clear, well documented program source code 5%

- Brief performance report stating approach used, performance achieved and showing illustrate examples of your visual odometry output (as figures) 15%

- Additional credit will be given for one or more of the following:
  - comparison of variant heuristics and/or object feature filtering approaches
  - advanced visualization of the ground truth data against VO
  - visualization with overlay onto external mapping / satellite imagery
    (for any of the above up to a maximum, dependent on quality) 20%

Total: 100%

*[** as supporting evidence for this part you must also submit a video file of your system in operation over a sample of the data – make sure your video shows both the input images and odometry output. This can be constructed using OpenCV directly or any tool of your choice. File size must be less than 20Mb in size, video format in use must playback in the VLC tool – www.videolan.org]*

Submission:

You must submit the following:

- Full program **source code** for your solution to the above task as a **working python script** meeting the above ***"additional program specifications"*** for testing.

- *Optional* video file as supporting evidence (< 10Mb) as detailed above,

- **Brief Report (max. 750 words!, remember - it can always be shorter than this)** 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 produce *(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.* 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 program. Your code must run on one of the lab based PCs (with OpenCV 4.x on Windows or Linux) – 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. 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. Automated tools will be used to detect plagiarism.

To submit your work create a directory named by your username (e.g. *cxfh123*). Place all required files in this directory, ZIP up this entire directory structure (not rar or .7z or anything else, *please*) and submit it via DUO *(late submissions will be penalised following department policy).*

Submission Deadline (ALL) : 2pm (UK TIME) –13th December 2019

Relevant Resources (in addition to materials + reading for visual odometry lecture):

1. *A simple monocular visual odometry project in Python*
   https://github.com/uoip/monoVO-python
2. *Py-MVO: Monocular Visual Odometry using Python*
   https://github.com/Transportation-Inspection/visual_odometry
3. *Monocular Visual Odometry using OpenCV - https://avisingh599.github.io/vision/monocular-vo/*
4. *Visual Odometry from scratch - A tutorial for beginners - https://avisingh599.github.io/vision/visual-odometry-full/*