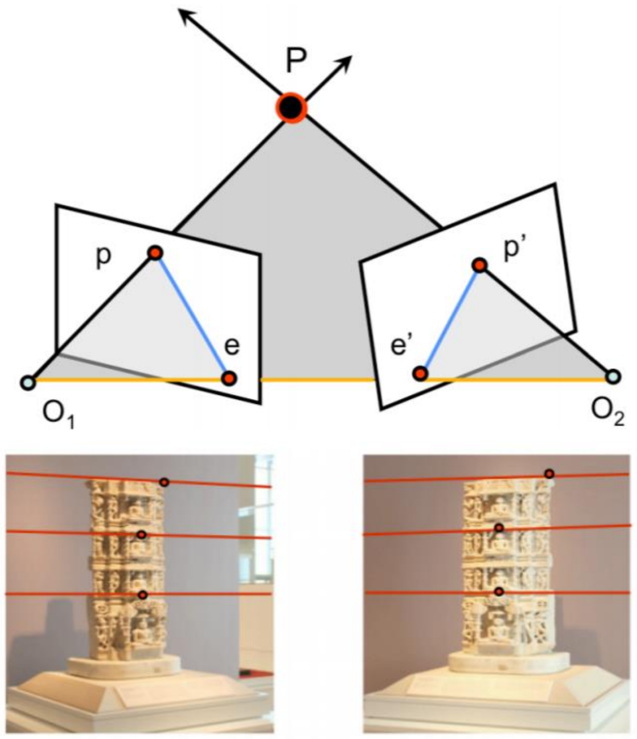
Epipolar Geometry

Functions: matrix math, feature detection, k-nearest-neighbors, fundamental matrix estimation, epipole determination/epipolar line calculation
Inputs: stereo image pairs
Metrics: per-stage and full-pipeline time

1. Overview

Epipolar geometry defines the relationships between points across multiple perspectives on a three-dimensional scene. By identifying points in one image that correspond positionally to points in another image (epipoles), it is possible to construct a representation of the 3D space in question by means of the epipolar plane (intersecting the same point as it appears in each image plus its true position in the scene itself) and epipolar lines (defined by the intersection of the epipolar plane and the two image planes). For further reading, see https://web.stanford.edu/class/cs231a/course_notes/03-epipolar-geometry.pdf or https://web.eecs.umich.edu/~jjcorso/t/598F14/files/lecture_1022_epipolar.pdf.



https://web.stanford.edu/class/cs231a/course_notes/03-epipolar-geometry.pdf

MilSpec's epipolar geometry reference implementation is adapted from https://github.com/mbeyeler/opencv-python-blueprints/blob/master/chapter4/scene3D.py. This is an archetypal epipolar geometry implementation written in Python that detects and matches keypoints in image pairs based on a configurable sensitivity threshold, computes the fundamental matrix that defines correspondence between the two images, and draws epilines. The MilSpec version of this code translates the Python code to C++, enhances configuration options, adds performance measurement, and adds rudimentary result sanity checks. The feature detection and fundamental matrix estimation methods are also configurable: feature detection may use AKAZE,[1] BRISK,[2] KAZE,[3] or ORB,[4] and fundamental matrix estimation may use RANSAC[5] or LMEDS.[6] These are selected in testcase-specific configuration files (see section 3). A future revision will further improve the facilities for testcase definition, output image generation, and result validation.

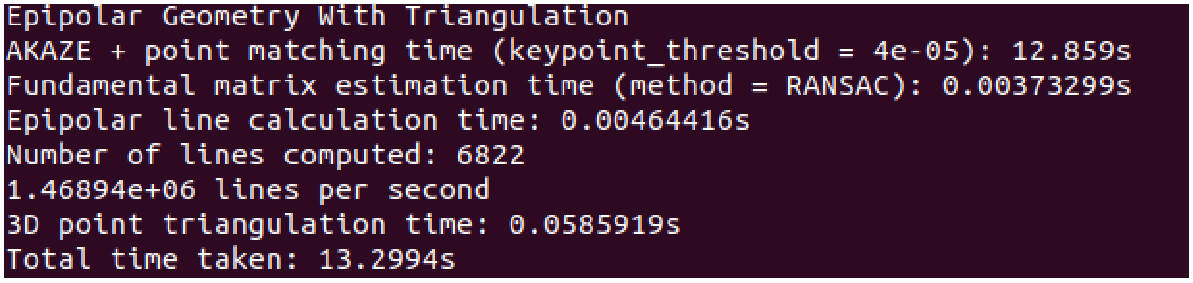
2. System requirements

Platform: Ubuntu 18.04 LTS with g++ 7.4.0. Code may build and run successfully on other versions/platforms, but has not been tested with them.
Storage: ~85MB for code and sample input images.
Dependencies: OpenCV 3.2.0 (may work with later versions, but not tested).

3. Build and run

To benchmark:

- Install OpenCV: `sudo apt-get update` and `sudo apt-get install libopencv-dev`
- Verify compatible OpenCV version: `pkg-config --modversion opencv`
- Download and extract the zipfile from www.adacenter.org/milspec
- From the `EpipolarGeometry/` directory, `make clean && make`
- Choose a testcase: `standard.in` (simple baseline testcase) or `triangulation.in` (includes the depth mapping step). Note that the keypoint threshold parameter is interpreted differently for different feature detection methods; see the links in the footnotes for details.
- From the `EpipolarGeometry/` directory, `./epipolar [testcase]`
- Results are displayed in the terminal, as below:



```
Epipolar Geometry With Triangulation
AKAZE + point matching time (keypoint_threshold = 4e-05): 12.859s
Fundamental matrix estimation time (method = RANSAC): 0.00373299s
Epipolar line calculation time: 0.00464416s
Number of lines computed: 6822
1.46894e+06 lines per second
3D point triangulation time: 0.0585919s
Total time taken: 13.2994s
```

4. Code structure

Coming soon!

[1] https://docs.opencv.org/3.4/db/d70/tutorial_akaze_matching.html
[2] https://docs.opencv.org/trunk/de/dbf/classcv_1_1BRISK.html
[3] https://docs.opencv.org/3.4/d3/d61/classcv_1_1KAZE.html
[4] https://docs.opencv.org/4.1.2/d1/d89/tutorial_py_orb.html
[5] https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#gae420abc34eaa03d0c6a67359609d8429
[6] Ibid.

5. MilSpec development notes, errata, changelog

v0.9:

- Changes from baseline epipolar geometry code for MilSpec:
    - Translated from Python to C++
    - Created makefile
    - Added basic configuration options and file-based testcase input
    - Created basic testcases
    - Added performance metrics
    - Added selectable feature detection and fundamental matrix estimation methods

6. Acknowledgements

MilSpec is a project under development at the University of Michigan – Ann Arbor by Pete Ehrett, Bing Schaefer, Adrian Berding, John Paul Koenig, Pranav Srinivasan, Todd Austin, and Valeria Bertacco. This project is supported by the Applications Driving Architectures Center, one of six centers of JUMP, a Semiconductor Research Corporation program co-sponsored by DARPA.

The original epipolar geometry implementation upon which the MilSpec version is based was written by Michael Beyeler,[7] and is available at https://github.com/mbeyeler/opencv-python-blueprints/blob/master/chapter4/scene3D.py. Input images are drawn from http://vision.middlebury.edu/stereo/data/scenes2014/ and https://drivingstereo-dataset.github.io/.

[7] M. Beyeler, (2015). OpenCV with Python Blueprints: Design and develop advanced computer vision projects using OpenCV with Python. Packt Publishing Ltd., London, England, 230 pages, ISBN 978- 178528269-0.