# Python Port of “nister2” Two-View Triangulation Method

**Introduction / Motivation:** Let a point $X \in R^3$ be visible across two images. If the camera’s calibration matrix is $\boldsymbol{P}$ and $x_i\ st\ \{i = 1,2\}$ is its 2D projection onto the image plane then the problem of determining the 3D position of $\boldsymbol{X}$ from its 2D projections is called the **Triangulation** problem. This problem is formulated as

$$x_i = \boldsymbol{P_i} * X \tag{1}$$

where $\boldsymbol{P_i}$ are the pair of camera matrices for images 1 and 2 respectively.

In practice, due sensory noise and other disturbances, solving this problem requires a non-trivial solution that satisfies the non-convex geometric constraint called *epipolar constraint* which states that if two points $x \to \hat{x}$ and $x' \to \widehat{x'}$ are obtained in the Image domain such that $\hat{x}F\widehat{x'} = 0$ then a globally optimal solution to Equation 1 is achieved. The most popular method to solution is [1] which requires finding all roots of a six DOF polynomial. In practice, while near-global optimal, this “direct” method is computationally expensive. Iterative methods such as [2,3] while more performant, is held back by the high number of iteration required, especially in unstable camera configurations and the need to test for convergence. Lindstorm in [4] developed a novel iterative, image-space iterative method that takes optimal steps by solving a quadratic problem and is guaranteed to be quasi-optimal by satisfying epipolar constraint to $10^{-12}$ degree accuracy. Named as “nister2”, this method outperformed [1] and is comparable in performance to several newer methods [2,3]. The method was written for C++ projects but to the best of this author’s knowledge, no official python port of this method exists.

**Objective:** The objective of this project is to create an **open-source Python port of nister2** and test its performance against OpenCV 4.0’s optimized implementation of [1] on real-world dataset(s).

**Proposed Approach:** The proposed approach is to first convert the C++ head-only library it into a full Python module (or package if time permits) usable with OpenCV 4.0 API. Then using the large image dataset “NotreDam” compare “Points/sec” processed by Hartle’s method and nister2. Then using OpenCV 4.0’s compute_pose() method, compare the camera pose obtained between the two methods.

**References:**


1. Hartley and Zisserman, (2004), “Multi-view Geometry: Triangulation”
2. Kanazawa and Kannatani, (1995), “Reliability of 3-D Reconstruction by Stereo Vision”
3. Lee and Civera, (2018) “Triangulation: Why Optimize”
4. Peter Lindstorm, (2010), “Triangulation Made Easy”