Visual Odometry from Stereo Using Point-Set Matching

KOTWAL ALANKAR SHASHIKANT
12D070010

ANAND KALVIT
12D070032

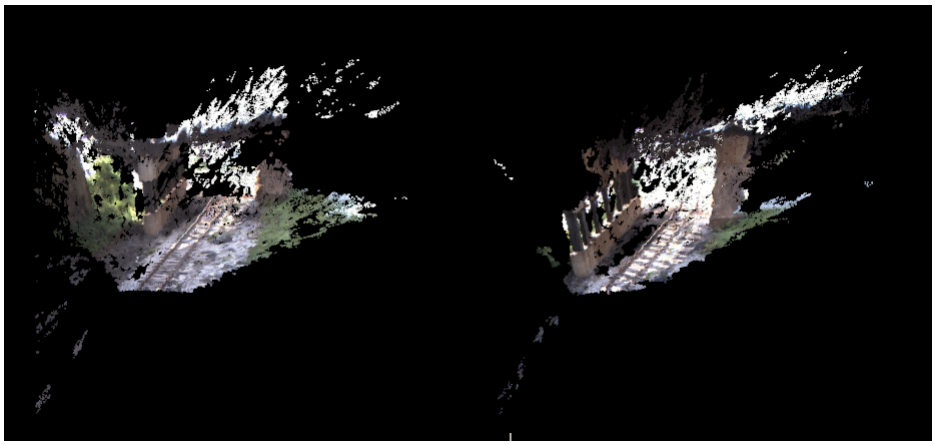
April 29, 2015

1 Introduction

Odometry refers to the use of data from motion sensors to estimate change in position over time. One of the most reliable ways of estimation of 3-D structure using cameras is to use a calibrated stereo pair. Given the sequence of 3-D structures generated by the stereo camera, we can estimate the motion of the camera with respect to its environment as well as generate a 3-D map of the environment. This is usually referred to as visual SLAM (simultaneous localization and mapping), which has wide applications in robotics and remote sensing.

We plan to implement a 6 DoF pose estimation algorithm using a calibrated stereo pair and generate a 3-D map of the environment simultaneously. We assume that scene illumination doesn't change much and most of the field of view of the camera is occupied by static parts of the environment.

In a nutshell, our job is to figure out what the translation and rotation of the camera is between two scenes as shown below.



2 Method

We closely follow the method outlined in [1]. The method involves modelling both the reference and the moving pointclouds as Mixture Models. We choose spherical Gaussian Mixtures and equal weights per-point for simplicity and easy updates. Our model for the pointclouds, thus, is given by

$$\mathcal{S}(\mathbf{x}) = \sum_{i=1}^{N} \mathcal{G}(\mathbf{x}; \mathbf{s}_i, \Sigma), \mathcal{M}(\mathbf{x}) = \sum_{i=1}^{M} \mathcal{G}(\mathbf{x}; \mathbf{m}_i, \Sigma)$$

where $N = |\mathcal{S}|, M = |\mathcal{M}|, \Sigma = diag(\sigma^2)$. We then maximise the kernel correlation between a translated and rotated version of the moving pointcloud and the reference pointcloud, with respect to the rotation and the translation, yielding the optimization problem

$$\mathbf{R}, \mathbf{t} = arg \max_{\mathbf{R},\mathbf{t}} \sum_{i=1}^{M} \sum_{j=1}^{N} e^{-\frac{||\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_j||^2}{2\sigma^2}}$$

such that $\mathbf{R}^T\mathbf{R} = I$. This is a convex program and the optimization is carried out using gradient ascent on the components of $\mathbf{t}$ and a parametrization of the rotation matrix $\mathbf{R}$, as described in the implementation section below.

3 Implementation

We chose to code the method in C++ using PCL to handle pointcloud representation and i/o, and Eigen to handle the matrix algebra. With this, we chose to represent rotations using unit quaternions to make gradient-descent optimization convenient and avoid the orthogonality constraint on $\mathbf{R}$.

Gradient ascent requires us to calculate the gradient of the objective function with respect to the optimizing variables. With the help of [2] (see appendix), we get, with $F(\mathcal{S}, \mathcal{M}, \mathbf{R}, \mathbf{t})$ being the objective function defined above,

$$\frac{\partial F}{\partial \mathbf{t}} = \mathbf{G}^T \mathbf{1}_m$$

$$\frac{\partial F}{\partial r_i} = \mathbf{1}_d^T \left(\mathbf{G}^T \mathbf{M}_0 \otimes \frac{\partial \mathbf{R}}{\partial r_i} \right) \mathbf{1}_d$$

where (r_0, r_1, r_2, r_3) is the unit-quaternion representation of the rotation matrix.

The derivatives of $\mathbf{R}$ with respect to r_i are calculated, from [3] as

$$\frac{\partial \mathbf{R}}{\partial r_0} = 2\begin{pmatrix} r_0 & -r_3 & r_2 \\ r_3 & r_0 & -r_1 \\ -r_2 & r_1 & r_0 \end{pmatrix}$$

$$\frac{\partial \mathbf{R}}{\partial r_1} = 2\begin{pmatrix} r_1 & r_2 & r_3 \\ r_2 & -r_1 & -r_0 \\ r_3 & r_0 & -r_1 \end{pmatrix}$$

$$\frac{\partial \mathbf{R}}{\partial r_2} = 2\begin{pmatrix} -r_2 & r_1 & r_0 \\ r_1 & r_2 & r_3 \\ -r_0 & r_3 & -r_2 \end{pmatrix}$$

$$\frac{\partial \mathbf{R}}{\partial r_3} = 2\begin{pmatrix} -r_3 & -r_0 & r_1 \\ r_0 & -r_3 & r_2 \\ r_1 & r_2 & r_3 \end{pmatrix}$$

Apart from this, we used adaptive step-size to prevent overshoot and slow convergence.

4 Results

4.1 Generating pointclouds from stereo images

See pointclouds in *pointclouds/**. The original images had a resolution of 640×512 pixels, so the original pointclouds had of the order of 10^5 points. This is too much for us to process, so we downsample the pointclouds by a factor of $\sim$100 to reduce the number to a managable quantity.

4.2 Estimating identity transformations

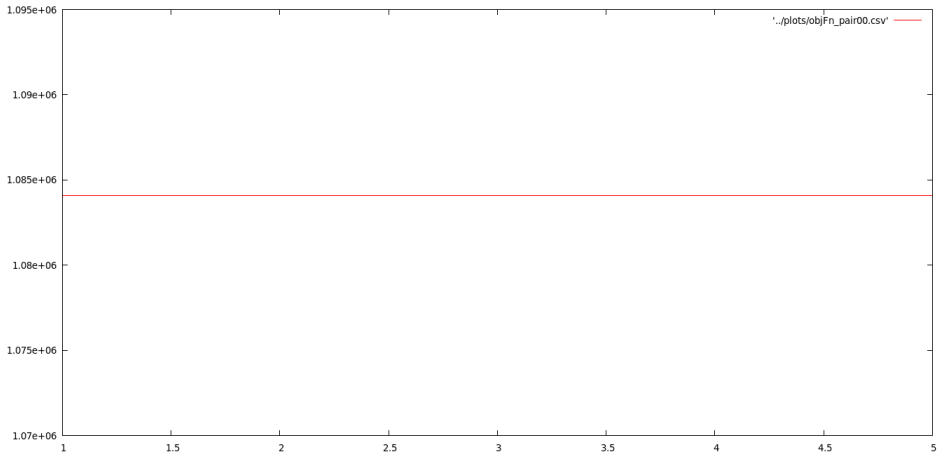
The first checkpoint is checking if the code estimates the transformation between identical pointclouds correctly. We save the output rotation matrix in *data/tfxxR.transform* and the output translation in *data/tfxxt.transform* where x is the pointcloud number. For example, the estimated identity transformation for the pointcloud *pair00.pcd* is

$$\hat{\mathbf{R}} = \begin{pmatrix} 1 & -7.72142 \times 10^{-10} & 7.72142 \times 10^{-10} \\ 7.72142 \times 10^{-10} & 1 & 1.26552 \times 10^{-8} \\ -7.72142 \times 10^{-10} & -1.26552 \times 10^{-8} & 1 \end{pmatrix}$$

and

$$\hat{\mathbf{t}} = \begin{pmatrix} 8.77619 \times 10^{-10} \\ 1.20335 \times 10^{-11} \\ 4.22396 \times 10^{-11} \end{pmatrix}$$

As is clear, these are very close to the identity transformation, confirming that the method estimates identity transformations properly. We sometimes have convergence problems (especially for large rotation angles). The plot of the objective function (starting from some initialization) is shown below:

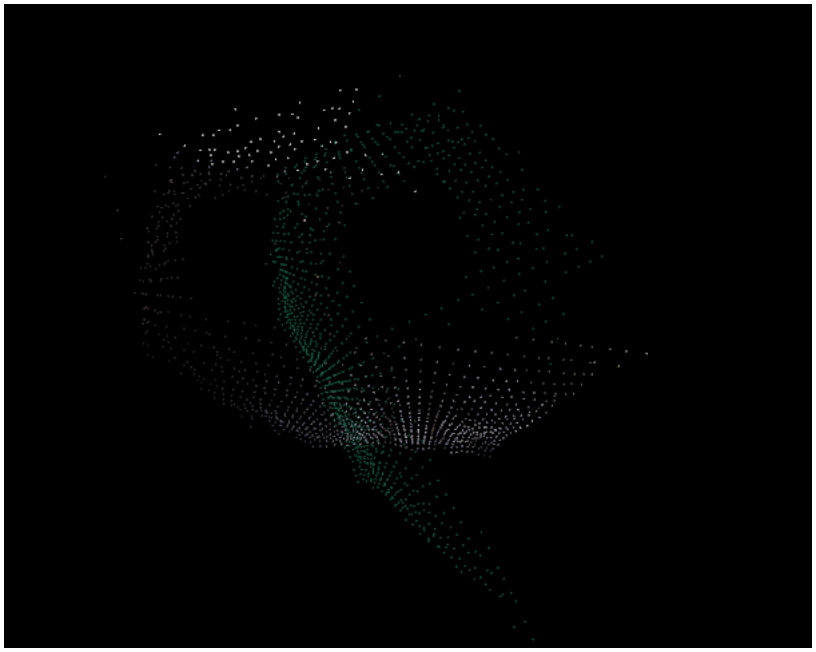


4.3 Estimating synthetic transformations

As the next step, we 'synthesize' transformed pointclouds (by translating and rotating one of the pointclouds by known quantites). For example, the transformation matrix

$$T = \begin{pmatrix} 0.500001 & -0.866025 & 0 & 2.5 \\ 0.866025 & 0.500001 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

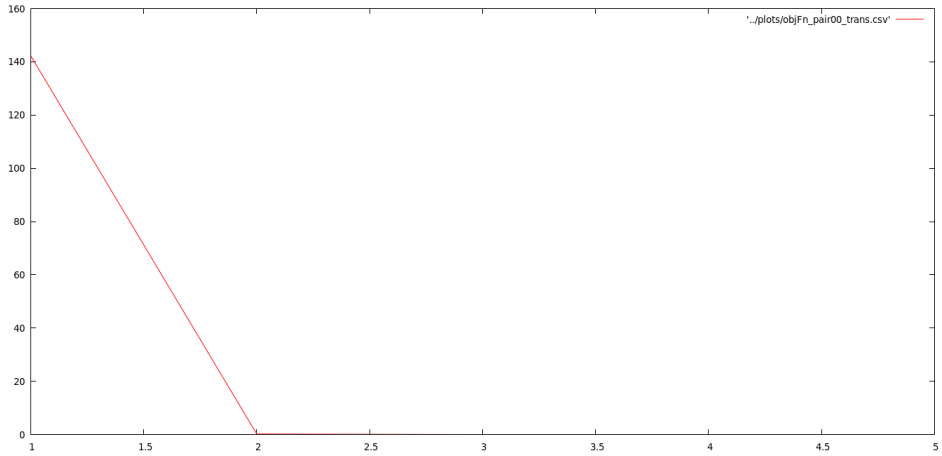
on the pointcloud *pair00.pcd* yields the two pointclouds (one white and one green) as shown:



We then run the matching algorithm on these pointclouds and compare the output to the (known) transformation matrix. For this particular example, the estimated transformation after five iterations is

$$T = \begin{pmatrix} 0.60301 & -0.797733 & 0 & 2.256 \\ 0.797733 & 0.60301 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

which is close enough to the original value. A plot of the objective function with iteration number is shown below:



4.4 Estimating non-identity transformations

Now we try to estimate the transformation between different pointclouds. Here we have significant convergence problems if the transformation is not initialised sufficiently close to the actual transformation. We think this is because the pointclouds used have relatively little overlap, and this is not sufficient to make the algorithm converge to the actual transformation. We do not include the results since we do not think this part is complete.

5 Other Methods

We also tried an implementation of Coherent Point Drift, see [5] and [6], which we did not get time to debug and test. This is in a non-working state right now.

6 Concluding Remarks

We showed the algorithm working for the identity transformation and transformations that don't differ much from identity. The objective function is bounded below by zero, and almost always (at least in the synthetic transformation case) drops down to zero by the first five iterations. This shows that the solution obtained is (almost) optimal. We did not explore the parameter space (changing Gaussian variance and step size) much. However, estimation of small transformations is useful enough since two pointclouds which are captured in succession are related by transformations that are not very different from each other.

References

[1] Bing Jian and Vemuri, B.C., *Robust Point Set Registration Using Gaussian Mixture Models.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2010.

[2] Pre-print of the above paper here.

[3] George Terzakis, Phil Culverhouse *et al*, *A Recipe on the Parameterization of Rotation Matrices for Non-Linear Optimization using Quaternions.* Technical Report of the School of Marine Science and Engineering, Plymouth University, 2012.

[4] Documentation for the OpenCV, Point Cloud and Eigen libraries in C++ here,here and here respectively

[5] Andriy Myronenko and Xubo Song, *Point Set Registration: Coherent Point Drift*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2010.

[6] The Wikipedia article on Point Set Registration here.

Appendix 1

Here we derive the expression for the kernel correlation between two Gaussian mixtures. Following [1] we want to evaluate the integral

$$\int_{\mathcal{R}^3} f_{\mathbf{R},\,\mathbf{t}} g d\mathbf{x}$$

with f and g being Gaussian mixtures with means $\mathbf{s}_i$ and $\mathbf{m}_i$ respectively. We assume uncorrelated Gaussians, so covariance everywhere is $\sigma^2\mathbf{I}$. Thus, the correlation becomes, *modulo* constants,

$$\sum_{i=1}^{N}\sum_{j=1}^{M}\int_{\mathcal{R}^3} e^{\frac{||\mathbf{x}-\mathbf{R}\mathbf{s}_i-\mathbf{t}||^2+||\mathbf{x}-\mathbf{m}_j||^2}{2\sigma^2}}$$

The numerator of the exponential can be written as

$$(\mathbf{x}-\mathbf{R}\mathbf{s}_i-\mathbf{t})^T(\mathbf{x}-\mathbf{R}\mathbf{s}_i-\mathbf{t})+(\mathbf{x}-\mathbf{m}_j)^T(\mathbf{x}-\mathbf{m}_j)$$

With $\mathbf{R}\mathbf{s}_i+\mathbf{t}\mathbf{n}_i$,

$$(\mathbf{x}-\mathbf{m})^T(\mathbf{x}-\mathbf{m})+(\mathbf{x}-\mathbf{n})^T(\mathbf{x}-\mathbf{n})$$

which is the same as

$$(\mathbf{x}-(\mathbf{m}+\mathbf{n}))^T(\mathbf{x}-(\mathbf{m}+\mathbf{n}))$$

which makes the correlation the sum of Gaussian functions with mean $\mathbf{RS}_i+t-m_j$ evaluated ar zero. So

$$\int_{\mathcal{R}^3} f_{\mathbf{R},\,\mathbf{t}} g d\mathbf{x} = \mathcal{K}\sum_{i=1}^{N}\sum_{j=1}^{M} e^{-\frac{||\mathbf{R}\mathbf{s}_i+\mathbf{t}-\mathbf{m}_j||^2}{2\sigma^2}}$$

where $\mathcal{K}$ is a positive constant independent of $\mathbf{R}$ or $\mathbf{t}$.

Appendix 2

Here we derive the expressions for the gradient descent technique used in the registration method.

The optimization problem under consideration is

$$\mathbf{R}, \mathbf{t} = arg \min_{\mathbf{R},\mathbf{t}} \sum_{i=1}^{M} \sum_{j=1}^{N} e^{-\frac{||\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_j||^2}{2\sigma^2}}$$

such that $\mathbf{R}^T\mathbf{R} = I$. To calculate the derivative of this quantity with respect to the optimization variables $\mathbf{R}$ and $\mathbf{t}$, we use the chain rule:

$$\frac{\partial F}{\partial q} = \frac{\partial F}{\partial M}\frac{\partial M}{\partial q}$$

where M is the transformed data-set and q is the optimization variable under consideration. Now with $\mathbf{m}_t$ being the transformed version of $\mathbf{x}_t$,

$$\frac{\partial F}{\partial \mathbf{m}_t} = \sum_{j=1}^{N} \frac{\partial}{\partial \mathbf{m}_t} e^{-\frac{||\mathbf{m}_t - \mathbf{y}_j||^2}{2\sigma^2}}$$

thus yielding

$$\frac{\partial F}{\partial \mathbf{m}_t} = -\sum_{j=1}^{N} \frac{e^{-\frac{||\mathbf{m}_t - \mathbf{y}_j||^2}{2\sigma^2}}}{\sigma^2} \left(\mathbf{m}_t - \mathbf{y}_j\right)^T$$

The matrix consisting of M of these row vectors is what we called $\mathbf{G}^T$.

Clearly,

$$\frac{\partial \mathbf{m}_i}{\partial t_x} = \frac{\partial \left(\mathbf{R}\mathbf{s}_i + \mathbf{t}\right)}{\partial t_x}$$

which is just 1. So, the derivative of F with respect to $\mathbf{t}$ is just $\mathbf{G}^T\mathbf{1}_M$ where $\mathbf{1}_M$ is the column vector of all ones, corresponsing to adding up the derivatives with respect to all $\mathbf{m}_t$s.

Figuring out the derivative with respect to rotation is a bit more difficult since we have a constraint on the rotation matrix. To avoid having to explicitly enforce this constraint we use a quaternion parametrization of the rotation matrix. Following the excellent treatment in [3], we have the rotation matrix induced by the quaternion $\mathbf{q} = (q_0, q_1, q_2, q_3)$ is

$$\mathbf{R} = \begin{pmatrix} q_0^2 + q_1^2 + q_2^2 + q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix}$$

Now clearly

$$\frac{\partial \mathbf{m}_i}{\partial q_t} = \frac{\partial \mathbf{R}}{\partial q_t}\mathbf{m}_{i0}$$

where $\mathbf{m}_{i0}$ is the corresponding point in the untransformed reference point-cloud. Now we directly differentiate the objective function and then convert to the desired form using matrix calculus, just for a change.

$$\frac{\partial F}{\partial q_t} = -\sum_{i=1}^{M}\sum_{j=1}^{N} \frac{e^{-\frac{||\mathbf{m}_i - \mathbf{y}_j||^2}{2\sigma^2}}}{\sigma^2}(\mathbf{m}_i - \mathbf{y}_j)^T \frac{\partial \mathbf{m}_i}{\partial q_t}$$

thus

$$\frac{\partial F}{\partial q_t} = -\sum_{i=1}^{M}\sum_{j=1}^{N} \frac{e^{-\frac{||\mathbf{m}_i - \mathbf{y}_j||^2}{2\sigma^2}}}{\sigma^2}(\mathbf{m}_i - \mathbf{y}_j)^T \frac{\partial \mathbf{R}}{\partial q_t}\mathbf{m}_{i0}$$

Writing this out explicitly and interchanging the sum of the elements of a matrix expressed by $(\mathbf{m}_i - \mathbf{y}_j)^T \frac{\partial \mathbf{R}}{\partial q_t}\mathbf{m}_{i0}$ and the matrix product expressed by the double summation yields

$$\frac{\partial F}{\partial q_i} = \mathbf{1}_d^T \left(\mathbf{G}^T \mathbf{M}_0 \otimes \frac{\partial \mathbf{R}}{\partial q_i} \right) \mathbf{1}_d$$