Ninth Hungarian Conference on Computer Graphics and Geometry, Budapest, 2018

Plane Matching Between Camera Images and 3D Point Clouds

Soma Szeier[1,2] and Levente Hajder[1,2]

[1] Machine Perception Research Laboratory, MTA SZTAKI, Budapest, Hungary
[2] Department of Algorithms and Their Applications, ELTE IK, Budapest, Hungary

Abstract
Detecting the surrounding environment is one of the most researched areas in self-driving technology. For this purpose, a typical solution is to detect planes from various sensors (cameras, LIDARs, etc.) as planes can be detected in different modalities. Using fitted planes, the relative position of the sensors can be computed. In this paper, we propose a novel method to find corresponding planes detected by cameras and LIDARs to calibrate them.

1. Introduction

In recent years, self driving technology has become one of the most interesting and fastest growing fields of research. The ability to not rely on human limits greatly increases what these vehicles can achieve: they can drive continuously without the driver becoming tired or unfocused, safe driving speed is increased due to near instant reaction time and multiple vehicle communication, etc.

One of the most important features of a self driving car is the way it analyzes its surroundings. Most commonly, this is done by mounting LIDAR sensors and cameras onto the vehicle.

A Light Detection and Ranging (LIDAR) sensor measures the distance to a target with laser light and measuring the reflected light with a sensor. With multiple such readings, we form a 3-D image of the target. Most often, we use these images to detect planes near the vehicle because planes appear in nearly all man-made environments. A plane is defined by three points on it, however not every three points represent a plane on the target, therefore plane detecting algorithms locate the "best" planes on the target. What exactly makes a plane good varies for each algorithm.

Cameras are also used to detect planes or obstacles. Unlike LIDARs, a single camera image is not enough to detect planes on the target. Atleast two images are required, although having more will increase the accuracy of the planes. In camera images, a plane is defined by four points on one image and their corresponding ones on the second image. Using these points we can calculate the homography matrix of the plane. A homography matrix is a 3×3 transformation matrix that converts the location of a point on a plane on the first image to it's location on the second.

The idea behind this paper is to combine the data from LIDARs and cameras since most self driving vehicles are equiped with both. By combining them, our data should be more acurate than either data set by itself.

2. Related work

Plane detection is a heavily researched area of computer vision. In this paper, we introduce PEARL and Multi-H, two state-of-the-art methods used to detect planes in point clouds and images respectively.

2.1. PEARL

PEARL [5] is a multi-model fitting algorithm that uses α-expansion [3] and the least squares method to find the model with the lowest possible energy, which is calculated from a given function. Here, we use it to find two-dimensional planes in three-dimensional space. PEARL consists of three sections, where the second and third are iterated until convergence.

An energy function evaluates the model set and penalizes it, giving the set's energy. By minimizing the energy, we arrive at a set that is considered the “best” by the energy function.

There are three basic ideas about what we consider a "good" result that must be addressed in the energy function. The first one is that each point should belong to a plane that a short distance away. The next one is that neighboring points should be on the same plane. The last idea is that the point cloud consists of only a few planes, each having many inliers, instead of many planes, each having only a few outliers. These ideas are realized in the following equation:

$$E(L) = \sum_p ||p - L_p|| + \lambda \cdot \sum_{p,q \in N} w(p,q) \cdot \delta(L_p \neq L_q) + \beta \cdot |\Omega_L| \tag{1}$$

where L is the current labeling, each label presenting a plane, λ and β are constants that are used to change the relative weight of each term, N is the neighbors matrix, w is a weight function, $\delta(x)$ is one if x is true and zero if false, and $|\Omega_L|$ is the number of labels that contain atleast one point.

In the first step, we randomly generates planes on the pointcloud. This is done by randomly selecting three points and finding the plane that they determine. The number of planes that should be generated for an accurate result depends on several factors including how many points each plane contains, the outlier ratio of the dataset, how accurate the result should be, etc.

Next, we use alpha-expansion to decrease the energy of the model set. Alpha-expansion iterates through every model in the set, naming it "alpha" and the others "not alpha" and attempts every possible graph cut, until finally selecting the "best" one. This step is iterated a predetermined number of times or until the set's energy converges.

Lastly, using the models alpha-expansion has created, we optimizes the energy of each individual model by assigning them a new plane by applying the least squares method to the points assigned to them. This step has no effect on the second and third part of the energy function, while minimizing the first, therefore we can determine that it truly decreases the set's energy.

2.2. Multi-H

Multi-H requires two images as input and returns with the planes detected and their homographies. It solves a common problem with plane detection, that planes with few inliers cannot be identified. Instead of a plane being defined by its inliers, in Multi-H its defined by a single point and that point's local affine transformation.

There are several methods to estimate the local affine transformations. Multi-H uses an affine covariant feature point detector [6], which returns with the coordinate of the point and its local affine transformation. For the detector, we use the Matching On Demand with view Synthesis (MODS) [7] algorithm, which calculates the affine transformations and discards those that aren't compatible with an estimated F fundamental matrix.

Next, Multi-H calculates the H homography matrix for each point correspondence using it's local affine transformation with the Homography from Affine transformation and Fundamental matrix (HAF) [2] method.

Finally, much like PEARL, Multi-H iterates three steps until convergence: using Mean shift [4] on the homographies, finding the best homography for each point using α-expansion [3], and optimizes the homographies using the least squares methods.

Mean shift is a feature-space analysis technique for locating the maximum of a density function. By converting homopgraphies to vectors, we use Mean shift to cluster them. The noise from the data leads to noisy homographies, but we assume that they should cluster around the actual value. We then replace every homography with the point it clusters around.

α-expansion step is similar to what is used in PEARL. The energy function is

$$E(L) = \frac{1}{\lambda} E_d(L) + \lambda E_s(L) \tag{2}$$

where L is the current labeling, $E_d(L)$ is the error of points from their homography, and $E_s(L)$ is the error from neighboring points.

$E_s(L)$ is the sum of the distance between each point's homography projected position and their pair, thus penalizing points for belonging to incorrect homographies. The function is

$$E_d(L) = \sum_{i=1}^{n} ||p_2^i - \frac{H^{l_i} p_1^i}{H_{31}^{l_i} p_1^{i,x} + H_{32}^{l_i} p_1^{i,y} + H_{33}^{l_i}}|| \tag{3}$$

where $l_i \in L$ is p^i's label and H^{l_i} is l_i's homography. $E_s(L)$ term is based on the idea that neighboring points are likely to belong to the same homography.

$$E_s(L) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} \delta(l_i \neq l_j) \tag{4}$$

where A is the neighbors matrix, A_{ij} is one if p^i and p^j are neighbors and zero if they aren't, and $\delta(x)$ is one, if x is true and zero if false.

For every homography, we use least squares and HAF on the points on it to recalculate the optimal homography. This step does not affect the first term in the energy function since the number of homographies remains unchanged and minimizes the value of the second term, thus the overall value also decreases.

3. Plane Matching Algorithm

In this paper, we propose a way to calibrate cameras and LIDARs using their detected planes' normal vectors. The advantage of this algorithm is that beyond the readings of the equipment, the only additional information required is the camera's calibration matrix.

The first step is to run the PEARL and Multi-H algorithms and calculate the normal vector of each plane. With the point cloud planes, this poses no difficulty if we define a plane as ax + by + cz + d = 0 where we see that the normal vector is [a,b,c]. However, finding the normals in the image planes is more difficult, because using the homography of a plane (calculated by Multi-H) and the camera's calibration matrix, we can calculate four normals vectors, but are unable to further narrow our search.

Next, we must evaluate every possible result and find the optimal answer. To do this, we must iterate through every possible way that image planes and point cloud planes can be paired. This is done with an array whose size is the number of point cloud planes and where A[i] = j represents the number i point cloud plane being paired with the number j image plane. However, because each image plane has four normals and a point cloud plane can not have a pair on the images, we use a new list of image planes where each plane appears four times, once with each different normal vector, and with an extra plane for planes without a pair. Thus, for each iteration, we have a list of pairs between point cloud and image normals.

To find the best combination of plane pairs one from the images, one from the point cloud, that represent the same plane in the real world. We iterate through every possible combination of pairings until, determining how accurate the each of them are. Some can be discarded as incorrect early on. These are the ones where multiple normal vectors of the same image plane appear in the pairing, where there are too few (two or less) planes being matched, or where the angle between two image planes differentiates too much from the angle between their pointcloud counterparts.

If a pairing isn't eliminated in the previous part, we determine the error of the matching by using Euclidean transformation [1]. By converting the normals vectors into three-dimensional points, we get two point clouds, one containing the image normal vectors and the other containing the point cloud normal vectors, labeled o_i and p_i respectively. The Euclidean transformation is

$$p_i = qRo_i + t \tag{5}$$

where R is a 3x3 orthonormal rotation matrix, t is a translation vector, and q is a scaling variable. However, because these points were created from normal vectors the origin of both point clouds is the same and every point is at a distance of one from the origin, the translation vector is $\underline{\mathbf{0}}$ and the scaling variable is one. The translation can be simplified

$$p_i = Ro_i. \tag{6}$$

Where only R is unknown. Because the data is noisy, we estimate the R matrix with the lowest possible value for the energy function

$$E = \sum_{i=1}^{n} ||p_i - Ro_i||^2. \tag{7}$$

Expanded, the function is:

$$E = \sum_{i=1}^{n} p_i^T p_i + o_i^T o_i - 2p_i^T Ro_i. \tag{8}$$

To minimize the result, we have to maximize

$$\sum_{i=1}^{n} p_i^T Ro_i. \tag{9}$$

This is solved using the Lemma

$$Trace(AA^T) \geq Trace(RAA^T), \tag{10}$$

where A is a general matrix, R is an orthonormal matrix, and

$$Trace(RAA^T) \geq Trace(RA^T A) = \sum_{i=1}^{n} a_i^T Ra_i. \tag{11}$$

Let $H = \sum_{i=1}^{n} p_i o_i^T$. We calculate H's Singular Value Decomposition: $H = UDV^T$, where U and V are 3×3 orthonormal matrices and D is a 3×3 diagonal matrix. Let $X = VU^T$. Since X is the product of two orthonormal matrices, therefore it is also orthonormal. By multiplying X and H, we get

$$XH = VU^T UDV^T = VDV^T. \tag{12}$$

If $A = V\sqrt{D}$, then $AA^T = XH$. Using the Lemma, we get

$$Trace(XH) \geq Trace(R'XH). \tag{13}$$

The maximum value of the right side of the equation is at $R' = I$, therefore the maximum value of $\sum_{i=1}^{n} = p_i^T Ro_i$ is when $R = UV^T$

For each iteration, we calculate the average error of each pairing

$$E = \frac{1}{n} \sum_{i=1}^{n} ||p_i - Ro_i|| \tag{14}$$

. However, we must also take into account the number of pairs being matched in the iteration.This is because if we take any pairing that contains more than three pairs, we can decrease the average error by dropping the worst pair. If the worst pair had a low enough error, then the original pairing is the better result.

4. Results

The algorithm has been tested with noiseless synthetics tests and on a real world example. In the synthetics tests, the planes were matched correctly without an error appearing. The real world test can be seen on Fig. 1.

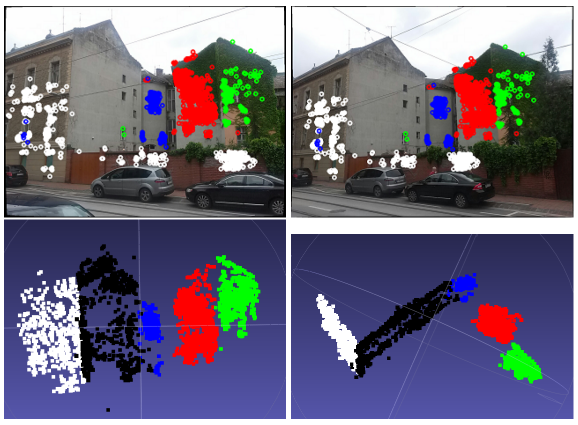

Figure 1: Result of the algorithm. Corresponding planes have the same color. The black plane is unmatched.

5. Further Development

There are multiple ways to improve the algorithm. First of all, since it only uses normals vectors when pairing, two or more parallel planes can easily be mixed up. To avoid this, all parallel planes should be checked if they are wrongly paired based on their position relative to the rest of the data set.

Another improvement would be for the algorithm to take multiple images as input, to calculate the image plane normals more precisely.

Lastly, after the algorithm has found the best pairings, we can use the fact that the camera and LIDAR are now calibrated to check for planes exist that exist on both data sets, but were only detected on one. Because the calibration is already known, we can match planes with less acuracy than was needed before. This could, for example, find that the black plane on Figure 1 is indeed present on the images.

References

1. K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, May 1987. 3

2. D. Barath and L. Hajder. A theory of point-wise homography estimation. *Pattern Recogn. Lett.*, 94(C):7–14, July 2017. 2

3. Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, Nov. 2001. 1, 2

4. D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, May 2002. 2

5. H. Isack and Y. Boykov. Energy-based geometric multi-model fitting. *Int. J. Comput. Vision*, 97(2):123–147, Apr. 2012. 1

6. K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of affine region detectors. *Int. J. Comput. Vision*, 65(1-2):43–72, Nov. 2005. 2

7. D. Mishkin, J. Matas, and M. Perdoch. Mods. *Comput. Vis. Image Underst.*, 141(C):81–93, Dec. 2015. 2