Learn to Scale

alex.kreimer, ilan.shimshoni

May 2016

Chapter 1

Infinite Odometry

1.1 Introduction

Visual odometry refers to the problem of recovering camera motion based on the images taken by it. This problem naturally occurs in robotics, wearable computing, augmented reality and automotive.

Wheel odometry, recovers the motion of the vehicle by examining and integrating the wheel turns over time. In a similar manner, visual odometry operates by estimating relative motion of the camera between subsequent images by observing changes in them. Later, these estimates are combined into a single trajectory. Just as in wheel odometry, visual odometry is subject to error accumulation over time. Contrary to wheel odometry, visual odometry is not affected by wheel slip in a rough terrain. Visual odometry is able to produce motion estimates with errors that are lower than those of the wheel odometry. Another advantage of visual odometry is that cameras are low cost and low weight sensors. All these make visual odometry a viable supplement to other motion recover methods such as global positioning systems (GPS) and inertial measurement units (IMUs).

Visual odometry becomes a harder problem as the amount of detail in the images diminishes. The images should have sufficient overlap and the scene needs to be illuminated. In the stereo setup, the scene must be static or the images taken at the same time. Also, the video processing incurs a computational burden.

Visual odometry is an active fields of research with a large amount of published work. We review only the most pertinent works. [**?**] provides a more complete survey.

Similar to [**?**] we partition visual odometry algorithms by four traits:

1. Feature-based vs direct
2. Global vs local
3. Filter based vs bundle adjustment based

4. Monocular vs stereo

Visual odometry algorithms use large number of corner detectors (e.g., Moravec [**?**], Harris [**?**], Shi-Tomasi [**?**], Fast [**?**]) and blob detectors (e.g., SIFT [**?**], SURF [**?**]). Corners are faster to compute and usually are better localized, while blobs are more robust to scale change. The choice of a specific feature point depends mainly on the images at hand. Motion estimation results for different feature points are presented in [**?**]. In this work we choose Harris [**?**] corners, but this choice is not crucial. We view the feature point choice as a parameter, which needs to be determined from the data (e.g., by cross-validation).

The features are either tracked [**?**] or matched [**?**] (i.e., freshly detected in each new frame) between subsequent images. While the early works chose to track features, most of the current works detect and match them. The output of this stage are pairs of the image features, which are the projections of the same 3-D point.

Matched features are used as an input for a motion estimation procedure. Whether the features are specified in 2-D or 3-D, the estimation procedures, may be classified into 3-D-to-3-D [**?**], 3-D-to-2-D [**?**] and 2-D-to-2-D [**?**]. Most of the early works were of the 3-D-to-3-D type. More recent works [**?**] claim that this approach is inferior to the latter two. Popular techniques that participate in most algorithms in some way are essential matrix estimation and (possibly) its subsequent decomposition [**?**], perspective 3-point algorithm [**?**], and re-projection error minimization [**?**].

Global methods [**?**], [**?**] keep the map of the environment and make sure that motion estimates are globally consistent with this map, while local methods do not. Some local methods [**?**] also keep track of a (local) map, but the underlying philosophy is different: global vs local. Global methods usually more accurate since they make use of a vast amount of information (which, of course, comes at a computational price). Note that accuracy does not imply robustness, since outliers that made their way into the map may greatly skew subsequent pose estimates.

Methods that explicitly model system state uncertainty tend to use filtering mathematical machinery, e.g., [**?**], [**?**], [**?**]. Another alternative to maintain map/pose estimate consistency is to use the bundle adjustment approach [**?**]. Monocular systems [**?**] make use of a single camera, while stereo systems [**?**] rely on a calibrated stereo rig. In the monocular setup the translation of the camera may only be estimated up to scale, while in stereo all six motion parameters may be recovered. An additional advantage of the stereo setup is that more information is available at each step, which may be one of the reasons why stereo algorithms perform better.

1.1.1 Our method

In this work we present a novel algorithm for camera motion estimation. The novelty of the algorithm is in camera rotation estimation procedure. We rely on the fact that for scene points that are infinitely far from the camera, the

motion of the projected (image) points may be described by an homography (the infinite homography). For distant points this assumption is nearly true. Our algorithm starts by partitioning the scene points into two sets: distant and near-by. Then, camera rotation is estimated from the distant points and, subsequently, the translation is recovered from the near-by points.

We present two versions of the algorithm: one for the monocular and the other for the stereo settings. These versions differ in the way we partition points into the distant and the near-by ones and in the way the algorithms estimate translation.

With respect to the classification of the visual odometry methods given in the introduction, our work is local, feature based, stereo odometry. We do not use bundle adjustment, however the results of our algorithm may be subsequently improved with some form of bundle adjustment.

The outline of the our method:

1. Feature detection. We use Harris [**?**] corners.
2. Feature matching. The matching is done both across the stereo pair images as well as previous vs. current pair. We enforce epipolar constraint, chierality and use circle heuristics similar to [**?**] to reject outliers.
3. Partition the scene points into two sets: distant and near-by.
4. Estimate the rotation of the camera from the distant points.
5. Estimate the translation of the camera from the near-by points.

We choose the work [**?**] as our baseline (our implementation of their work). The results in the Section 1.4 show that on the KITTI dataset our rotation estimation method outperforms the baseline.

1.2 Preliminaries and Notation

1.2.1 Image Point Mapping Related to Camera Motion

Suppose the camera matrices are those of a calibrated stereo rig P and P′ with the world origin at the first camera

$$\mathrm{P} = \mathrm{K}[\mathrm{I} \mid 0], \quad \mathrm{P}' = \mathrm{K}'[\mathrm{R} \mid \mathbf{t}]. \tag{1.1}$$

Consider the projections of a 3D point $\mathbf{X} = (X, Y, Z, 1)^T$ into the image planes of both views:

$$\mathbf{x} = \mathrm{P}\mathbf{X}, \quad \mathbf{x}' = \mathrm{P}'\mathbf{X}. \tag{1.2}$$

If the image point is normalized as $\mathbf{x} = (x, y, 1)^T$ then

$$\mathbf{x}Z = \mathrm{P}\mathbf{X} = \mathrm{K}[\mathrm{I} \mid 0]\mathbf{X} = \mathrm{K}(X, Y, Z)^T.$$

It follows that $(X, Y, Z)^T = \mathrm{K}^{-1}\mathbf{x}Z$, and:

$$\mathbf{x}' = \mathrm{K}'[\mathrm{R} \mid \mathbf{t}](X, Y, Z, 1)^T \tag{1.3}$$
$$= \mathrm{K}'\mathrm{R}(X, Y, Z)^T + \mathrm{K}'\mathbf{t} \tag{1.4}$$
$$= \mathrm{K}'\mathrm{R}\mathrm{K}^{-1}\mathbf{x}Z + \mathrm{K}'\mathbf{t}. \tag{1.5}$$

We divide both sides by Z to obtain the mapping of an image point $\mathbf{x}$ to image point $\mathbf{x}'$

$$\mathbf{x}' = \mathrm{K}'\mathrm{R}\mathrm{K}^{-1}\mathbf{x} + \mathrm{K}'\mathbf{t}/Z = \mathrm{H}_\infty\mathbf{x} + \mathrm{K}'\mathbf{t}/Z = \mathrm{H}_\infty\mathbf{x} + \mathbf{e}'/Z. \tag{1.6}$$

H_∞ is the infinite homography that transfers the points at infinity to the points at infinity. If $\mathrm{R} = \mathrm{I}$ (e.g., pure translation) the point $\mathbf{x}$ will undergo a motion along a corresponding epipolar line:

$$\mathbf{x}' = \mathbf{x} + \mathrm{K}'\mathbf{t}/Z = \mathbf{x} + \mathbf{e}'/Z. \tag{1.7}$$

If $\mathbf{t} = \mathbf{0}$ the motion of the point may be represented by a homology:

$$\mathbf{x}' = \mathrm{H}_\infty\mathbf{x}. \tag{1.8}$$

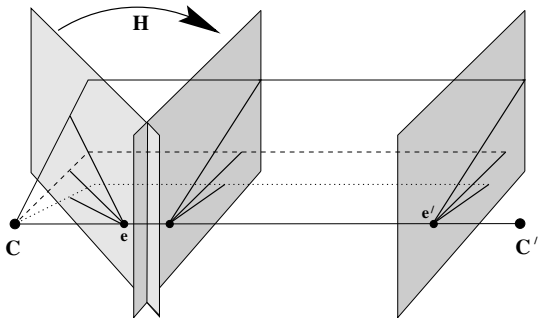


Figure 1.1: (Adapted from [**?**]) The effect of the camera motion on the image points may be viewed as a two-step process: mapping by a homography H_∞ followed by a motion along the corresponding epipolar lines.

In a general case the mapping of an image point $\mathbf{x}$ into $\mathbf{x}'$ may be viewed as a two step process: transformation by a homology (a specialization of homography which has two equal eigenvalues) H_∞ which simulates a pure rotational motion of the camera followed by an offset along the epipolar line which simulates a pure translational motion of the camera, see Figure 1.1.

1.3 Motion Estimation

Our strategy to attack the problem is to separate it into two smaller sub-problems: rotation estimation and translation estimation. The algorithm relies

on the ability to partition the scene points into two sets: the distant and the near-by ones. The distant points are used for rotation estimation while the near-by ones take part in the translation estimation.

First, the stereo algorithm is presented, followed by the monocular one. The main difference is in the translation estimation part. While it is possible to implement a stereo-like algorithm in the monocular setting as well, it suffers from the scale drift. Thus, we propose a different technique.

1.3.1 Stereo

Partitioning the points To partition the points in the stereo case we hard-threshold their Z-coordinates (the threshold is a parameter of the algorithm). The depth of the points was computed by a stereo triangulation.

Rotation Estimation: We use distant points to estimate rotation R (i.e., near-by points do not take part in rotation estimation). As Eq. (1.6) states:

$$\mathbf{x}' = \mathrm{KRK}^{-1}\mathbf{x} + \mathrm{K}\mathbf{t}/Z. \tag{1.9}$$

The total motion of the feature point in the image plane may be viewed as a two-step process (the order is not important): transformation by homography $\mathrm{H}_\infty = \mathrm{KRK}^{-1}$ followed by displacement along the line defined by the epipole e' and the point $\mathrm{H}_\infty\mathbf{x}$. The magnitude of the displacement along the epipolar line depends on the the camera translation and the inverse depth of the point, see Figure 1.2.

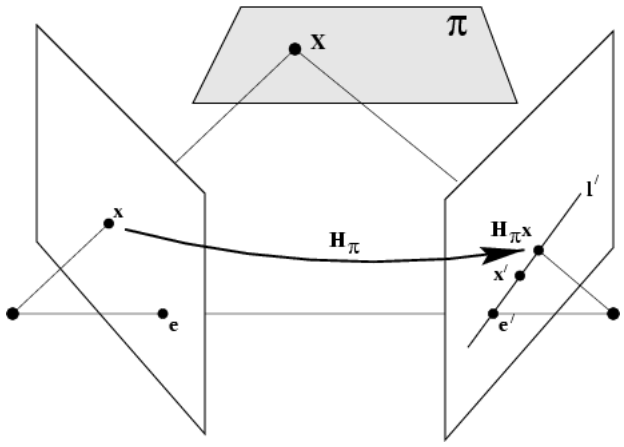


Figure 1.2: (Adapted from [**?**]) The homography H_π transfers the point onto a corresponding epipolar line. The displacement along the epipolar line depends on the inverse depth of the point and the camera translation magnitude.

Our estimation algorithm consists of initialization and non-linear refinement.

Initialization: to compute the initial estimate of the rotation parameters we assume that for the distant points (s.t., $\|\mathbf{t}\|/Z \ll \|\mathrm{H}_\infty \mathbf{x}\|$):

$$\mathbf{x}' \approx \mathrm{KRK}^{-1}\mathbf{x}. \tag{1.10}$$

This assumption is justified by the fact that for the distant points the displacement along the epipolar line is small. We multiply both sides of Eq. (1.10) by K^{-1} and denote $\mathbf{u}' = \mathrm{K}^{-1}\mathbf{x}$ and $\mathbf{u} = \mathrm{K}^{-1}\mathbf{x}$:

$$\mathbf{u}' = \mathrm{K}^{-1}\mathbf{x}' \approx \mathrm{RK}^{-1}\mathbf{x} = \mathrm{R}\mathbf{u}. \tag{1.11}$$

Since $\mathbf{u}$ and $\mathbf{u}'$ are projective quantities, only their directions are of importance, we normalize them to unit length and denote normalized quantities by $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{u}}'$ respectively. We choose a sample of n points ($n = 3$) and stack them as columns of matrices $\tilde{\mathrm{U}}$ and $\tilde{\mathrm{U}}'$ respectively. We search for R that solves the following minimization problem:

$$\underset{\mathrm{R}}{\operatorname{argmin}} \|\tilde{\mathrm{U}}' - \mathrm{R}\tilde{\mathrm{U}}\|_2. \tag{1.12}$$

Eq. (1.12) is known as the absolute orientation problem (see e.g., [**?**]) and its solution provides an initial estimate for the subsequent non-linear optimization problem.

Refinement: The idea of the refinement is this: the residual vector $\mathrm{H}_\infty\mathbf{x} - \mathbf{x}'$ may be viewed as a sum of a vector orthogonal to the epipolar line and the vector parallel to it. We search for the camera rotation that ignores the parallel component (we view it as a "legal" bias) while trying to minimize the orthogonal one. We define the point residual as the orthogonal distance to the corresponding epipolar line and minimize the sum of squared residuals for all points. We do so, because, as Eq. (1.6) suggests, after we compensate for a rotation, the point is still allowed to move along the epipolar line.

Consider the objective:

$$\begin{gathered}\mathrm{R}(\mathrm{v},\theta) = \underset{v,\ \theta}{\operatorname{argmin}} \sum_{i=1}^{N} {r_i}^2 \quad \text{s.t.} \quad r_i = n_i \cdot (\mathbf{x}_i' - H_\infty(v,\theta)\mathbf{x}_i) \\ \text{where } n_i = (\mathrm{F}\mathbf{x}_i)_\perp\end{gathered} \tag{1.13}$$

$\mathrm{H}_\infty(v,\theta) = \mathrm{KR}(\mathrm{v},\theta)\mathrm{K}^{-1}$ is the homography that transforms the image points in Eq. (1.6). It depends on a known camera intrinsic matrices K and a rotation matrix R. We choose to parameterize the rotation by an angle θ and an axis v. F denotes the fundamental matrix that corresponds to two subsequent stereo rig poses and is computed elsewhere. $\mathrm{F}\mathbf{x}_i$ denotes the epipolar line that corresponds to $\mathbf{x}_i$ in the second image and $(\mathrm{F}\mathbf{x}_i)_\perp$ is the normal to this line. We solve the minimization problem (1.13) by means of the Levenberg-Marquardt optimization algorithm.

To make the estimation robust we wrap the initialization procedure into the RANSAC iterations. We choose the strongest consensus estimate and its support set as an input for the solution of the Eq. (1.13).

Translation Estimation (1-point algorithm) To estimate the translation we use only the near-by points. First, we triangulate these points in the previous stereo pair to obtain their 3-D locations, and then iteratively minimize the sum of reprojection errors into the current frame.

The reprojection of point $\mathbf{X} = (X, Y, Z, 1)^T$ into the current left image is given by:

$$\pi^{(l)}(\mathbf{X}; \mathbf{t}) = K\Big[\mathrm{R} \mid \mathbf{t}\Big]\mathbf{X}. \tag{1.14}$$

and the reprojection of the same point into the current right image (b is the baseline of the stereo-rig) is given by:

$$\pi^{(r)}(\mathbf{X}; \mathbf{t}) = K\Big[\mathrm{R} \mid \mathbf{t}\Big](\mathbf{X} - (b, 0, 0, 0)^T), \tag{1.15}$$

We use the Levenberg-Marquardt algorithm to iteratively minimize the sum of squared reprojection errors (starting from $\mathbf{t} = \mathbf{0}$):

$$\|\mathbf{x}' - \pi^{(l)}(\mathbf{X}; \mathbf{t})\|^2 + \|\mathbf{x}' - \pi^{(r)}(\mathbf{X}; \mathbf{t})\|^2. \tag{1.16}$$

There are three unknown parameters, since $\mathbf{t} = (t_x, t_y, t_z)^T$, thus a single 3-D point provides enough constraints to determine $\mathbf{t}$.

1.3.2 Mono

The stereo setup has an advantage over the monocular one in the sense that it provides the algorithm with more information (e.g., the calibration and the additional image at each camera position). These advantages come at a price, e.g., the cameras need to be synchronized and the computational resource requirements climb. These make monocular setups and the related algorithms more attractive. In the following section we present the version of our algorithm, adapted for the monocular setup.

Given two sets of matching image points $\mathbf{x}_i, \mathbf{x}'_i$ from two subsequent frames I_1, I_2 respectively, we estimate the camera motion between these frames. Similar to the stereo algorithm in Section 1.3.1 the algorithm first partitions the points and then estimates the rotation followed by the estimation of the translation direction (it is well known that the magnitude of the camera translation is unavailable in the monocular setting).

Partitioning the points To estimate the rotation of the camera as in the Eq. (1.3), it is required to partition the set of the image points into the distant ones and the near-by ones. While in the stereo setting we may triangulate the points and threshold their depths, in the monocular setting this can not be done.

This section proposes a method to perform the aforementioned partition in the monocular setting.

Consider the subsequent images I_1, I_2, I_3 taken by a moving camera at the locations O_1, O_2 and O_3 respectively. We assume the image points $\mathbf{x}_i$ in I_1 are known to be distant relative to the camera at O_1. The magnitude of the camera translation is small relative to the distant points depths, thus we assume that these points are distant w.r.t. the camera at O_2 and O_3 as well. Some of these points will be lost in I_3, thus it is desirable to known which of the points tracked from I_2 to I_3 (which are not part of $\mathbf{x}_i$'s) are distant (denote these by $\mathbf{y}_j$).

The real baselines $t_1 = \|O_1 - O_2\|$ and $t_2 = \|O_2 - O_3\|$ are unknown and thus we can not use them to obtain real depths of the points.

We use the following procedure to classify the newly tracked points in I_2 as distant:

1. Set $t_1 = 1$ and triangulate the distant points $\mathbf{x}_i$ to obtain the depths Z_i
2. Set $t_2 = 1$ and triangulate the points $\mathbf{y}_j$ to obtain the depths Z_j
3. Classify the point y_j to be distant if $Z_j > \min_i Z_i$.

While the assumption $t_1 \approx t_2$ is acceptable for the KITTI dataset, it may be improved on by computing the t_1/t_2 ratio (by minimizing the reprojection errors of $\mathbf{x}_i$ into I_3, similar to the translation estimation described in the Section 1.3.1).

To initialize the monocular algorithm we may further assume that the initial motion is a pure translation and thus the points with small disparity are the distant ones (disparity being the magnitude of the motion in the image plane).

Rotation Estimation is exactly as in the Section 1.3.1. Denote the estimated rotation by R and the corresponding homography by H.

Translation Estimation is as follows. Compensate the rotation by computing $\mathbf{y}_i = \mathrm{H}_\infty \mathbf{x}_i$. We optimize over the location of the epipole e and minimize the orthogonal distances of the points to their corresponding epipolar lines:

$$\operatorname*{argmin}_e \sum_i d(l_i, \mathbf{y}_i) + d(l_i, \mathbf{x}'_i) \text{ s.t. } d(e, l_i) = 0 \text{ while } l_i = \operatorname*{argmin}_l d(l, \mathbf{x}'_i) + d(l, \mathbf{y}_i) \tag{1.17}$$

We define the epipolar line l_i to be the line that passes through the epipole and its distance to $\mathbf{x}'_i$ and $\mathbf{y}_i$ is minimal. $d(l, \mathbf{x})$ denotes the distance from the line l to the point $\mathbf{x}$. The epipole provides us with the translation direction of the camera.

1.4 Experimental Results

1.4.1 The Choice of Features

We chose to evaluate our algorithm on the KITTI dataset [**?**], which is a de-facto standard for the visual odometry research works.

Feature Detector/Descriptor: We use Harris [**?**] corner detector. It is fast, well localized and (most important) Harris corners are abundant in urban scenes we work with. We detect corners in each new image and then match them to obtain putative matches. We tune sensitivity threshold of the detector in such a way, that we are left with about five hundred putative matches after matching and pruning. We extract a square patch of 7×7 pixels centered at the corner point and use this vector as feature descriptor.

We would like to point out that our method may be used with any feature detector that would allow to match features across images. The choice of feature detector should be viewed as a parameter to the algorithm and mainly depends on the images at test.

Feature Matching: We use sum-of-square differences (SSD) of feature descriptors as a metric function when matching features. For each feature we choose a single best match w.r.t. the metric function in the other image. We employ a number of heuristics to prune outliers:

- Reciprocity: features a and b match only if a matches b and b matches a
- Epipolar constraint: we work with calibrated stereo pair. When we match features across images of stereo pair, the search is one-dimensional, i.e., along the horizontal epipolar line. This heuristic is not used when matching features across subsequent frames.
- Chierality (visibility): also used when matching features across stereo pair images. We triangulate the features to obtain the 3-D point and keep the match only if the 3-D point is visible in both cameras.
- Circular match: similar to [**?**] we keep only those matches that form a circle.

1.4.2 Experimental Results

The Tables 1.1 and 1.2 present the results of the experiments for the KITTI dataset. The columns denote the number of the sequence. The rows denote the algorithm: SS is the baseline, HX is the stereo version of the algorithm, HG is the monocular version. The numbers are the mean error for the corresponding sequence with the last column is the mean error for the dataset. The error computation method is described in [**?**].

Table 1.1: Rotation errors for the KITTI sequences [deg/m]

	00	01	02	03	04	05	06	07	08	09	10	mean
SS	3.95e-04	1.98e-04	4.11e-04	1.07e-03	8.03e-04	3.49e-04	4.72e-04	2.96e-04	3.69e-04	3.44e-04	4.82e-04	4.71e-04
HX	**2.70e-04**	**1.75e-04**	**4.10e-04**	**6.51e-04**	**6.04e-04**	3.95e-04	**3.77e-04**	**2.37e-04**	**3.23e-04**	**3.22e-04**	5.66e-04	**3.93e-04**
HG	8.72e-04	3.89e-04	6.28e-04	1.07e-03	5.99e-04	6.96e-04	3.31e-04	8.12e-04	8.13e-04	6.82e-04	5.23e-04	6.74e-04

Table 1.2: Translation errors for the KITTI sequences %

	00	01	02	03	04	05	06	07	08	09	10	mean
SS	4.40e+00	9.25e+00	4.03e+00	1.22e+01	5.06e+00	2.80e+00	4.37e+00	2.21e+00	4.12e+00	5.25e+00	5.60e+00	5.39e+00
HX	3.07e+00	1.08e+01	3.80e+00	7.94e+00	3.82e+00	4.06e+00	3.99e+00	1.67e+00	3.28e+00	3.77e+00	5.65e+00	4.72e+00
HG	1.21e+01	1.48e+01	8.72e+00	1.33e+01	8.62e+00	8.37e+00	4.46e+00	7.93e+00	9.76e+00	1.16e+01	8.36e+00	9.82e+00

Stereo In this set of experiments we ran our algorithm in its stereo mode as described in the Section 1.3.1. Table 1.1 and 1.2 present the rotation and the translation errors respectively in the row HX. The columns marked bold are those that our algorithm outperforms the baseline (on 9 of 11 sequences). The results show that our algorithm improves the rotation results over the benchmark algorithm and successfully competes with it in the translation estimation.

Mono In an additional set of experiments we ran our algorithm in a monocular mode. Monocular motion estimation lacks a scale parameter. In order to compare the results we set the scale of the translation to be that of the stereo algorithm. Feature point selection/partition was done without using any stereo information and the motion estimation was done as explained in the Section 1.3.2.

1.5 Conclusions and Discussion

This paper presents a novel visual odometry algorithm. The novelty of the algorithm is in its rotation estimation method. The rotation is estimated by means of the infinite homography. The algorithm may be used both in the stereo and in the monocular setting.

The strengths of the presented algorithm are in its ability to split the motion estimation problem into two smaller problems and to operate directly on the image points instead of on the computed 3-D quantities. Splitting the problem helps because each sub-problem is easier to solve. The ability to partition the points into the distant and the near-by ones is what allows us to separate the rotation and the translation estimation.

The stereo version of the algorithm shows better performance, but the monocular version has the advantage of being a more practical one. Indeed the authors in [**?**] report that they re-calibrate the cameras before each drive, which is hardly possible in real world installations.

Chapter 2

Monocular Scale

2.1 Introduction

Recovering camera 6-DOF ego-motion from images is a well studied problem. It arises in various practical contexts (e.g. virtual/augmented reality application, autonomous or aided navigation, etc.). The problem was studied in both stereo and the monocular setups. To recover the full 6-DOF motion, the previous works resorted to the stereo setup, used auxiliary sensors (e.g. IMU) or made assumptions about the camera pose and the scene. All of these have their drawbacks: stereo pairs are fragile and require careful calibration procedures, additional sensors are not always available and also require calibration, scene assumptions don't always hold. Motion estimation from images of a single moving camera is probably the hardest setup, as well as the most desirable one, because of its simplicity. It is well known that the translation scale parameter is not directly observable for a motion of a single camera.

We argue, that for natural scenes the scale information is present in the images and may be extracted. Our method learns a regressor, that is capable to prediction the motion scale.

2.2 Related work

2.2.1 Others

2.2.2 Our method

We assume that a single camera moves through space and takes images. We treat the initial camera pose (at time $t = 0$) as the world coordinate frame. We denote the pose of the camera at time t by $\hat{\mathbf{T}}_t$ described by the rotation matrix $\hat{\mathbf{R}}_t$ and the translation vector $\hat{\mathbf{t}}_t$ as seen in the world coordinate frame. We denote camera image taken at time t by I_t. To facilitate the discussion, we also introduce notation for camera pose $\mathbf{T}_t = [\mathbf{R}_t \mid \mathbf{t}_t]$ as seen from the coordinate

frame associated with camera pose at time $t-1$. Most of the time, we will omit the time index, since its clear from the context. By translation scale (or simply, scale) we refer to the norm of the translation vector $\mathbf{t}$ (e.g., $s = \|\mathbf{t}\|$)

We pose the scale estimation problem as a regression problem and search for a good regressor model.

2.3 Random forest

2.3.1 Decision trees

In this section we briefly describe tree-based methods for regression. These involve splitting the feature space into a number of small regions. The prediction for a sample is made by computing a mean or a mode of training samples that belong to the same region. Since the set of rules used to split the feature space into smaller regions may be described by a tree, these methods are referred to as *decision tree* methods.

Building a decision tree may be described by a two step procedure:

1. Split the feature space, e.g., the set of all possible values for $X_1, X_2, \ldots, X_n$ into J distinct regions $R_1, R_2, \ldots, R_J$.
2. For every sample that belongs to the regions R_i we make the same prediction, which is a mean of the responses of training samples that belong to this region.

The regions R_i are usually chosen to be multidimensional boxes (axis aligned). We would like to find such a partition of the feature space that minimizes

$$\sum_{j=1}^{J} \sum_{x \in R_j} (x - \bar{x}_j)^2 \tag{2.1}$$

Where $\bar{x}_j$ denotes the mean of the response values of the samples in region R_j. Unfortunately, solving the optimization problem 2.1 is computationally hard. Usually it is replaced with a greedy algorithm, called *recursive binary splitting*. This approach starts at the top of the tree and greedily chooses the best split at that point that minimizes the variance of its sub-trees. To be more precise, for each j and s we define the hyper-planes:

$$R_1(j,s) = \{X | X_j < s\} \quad \text{and} \quad R_2(j,s) = \{X | X_j \geq s\}, \tag{2.2}$$

We seek such j and s that minimize the equation:

$$\sum_{i: x_i \in R_1(j,s)} \left(y_i - \hat{y}_{R_1}\right)^2 + \sum_{i: x_i \in R_2(j,s)} \left(y_i - \hat{y}_{R_2}\right)^2, \tag{2.3}$$

Where y_{R_1}, y_{R_2} are the average responses of the samples in $R_1(j,s), R_2(j,s)$ respectively. Once we found the j and s we recursively split each sub-tree in a similar manner. The process is repeated until stopping criterion (e.g., number of nodes in the leaf) is reached.

2.3.2 Bagging

The decision trees tend to suffer from *high variance*. This means that if we split the training set into a number of random subsets and fit random tree into each sub-sample, we would likely to get much different answers from these trees when asked the same question. It is known that the variance of a mean of a set of independent random variables is $\frac{1}{n}$. Thus, in order to improve the variance of the estimator, it is possible to fit n estimators, each to its training-set and the average their predictions. Since, usually, we don't have n training sets, we would use *bootstrapping* (e.g., sample independently with replacement from the data set).

2.3.3 Random Forest

Random forest suggest additional improvement over bagging, by decorrelating the random trees. The issue they attempt to address is this: lets say there is a very dominant feature w.r.t. to task at hand for a given data-set. Bagging ensures that we use different training sets, but yet, most trees will tend to first split on this dominant feature. In this case the trees will resemble each other. In random forest, the trees are constructed by using only a subset of features (e.g., $m = \sqrt{p}$). This means that at calculating the splits, the algorithm is allowed to consider only a subset of features.

2.4 Convolutional Neural Networks

Convolutional neural networks (CNN) leverage the availability of the computational power and the abundance of data. CNN have become a method of choice in a number of computer vision areas (e.g., image classification, object recognition). They were also shown capable of per-pixel tasks, such as semantic segmentation and depth estimation from a single image. Works like (e.g., flownet) show that CNN can do well for task that require feature matching as well as feature extraction.

With these encouraging works in mind, we turn to CNN for our task. There are lots of network architectures out there to choose from. We have decided to use the ZF [**?**] object recognition network as a basic building block for our work.

The network architecture consists of 5 convolutional layers, kernel sizes 7, 5, 3, 3, 3. Four initial convolutional layers are each followed by the relu, max-pooling and local response normalization operations. Conv5 is followed by two fully connected layers separated by non-linearity. The fully connected layers are of size 4096. Finally, there is a last fully connected layer that reduced the network to a single output, which we are interested to learn. We use euclidean loss to train the network.

2.5 Recurrent Neural Networks

Many problems require passing information through time. Since the traditional neural networks are stateless, they are poorly suited for this purpose. Recurrent Neural Networks have loops in them, so that the information can persist.

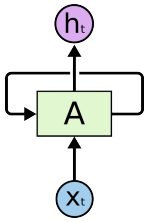


Figure 2.1: RNN

In the Figure 2.1 network A accepts input x_t and outputs h_t. The loop allows the network to pass information from one step of to another.

The RNN may be thought of as a chain of multiple copies of the same neural network where each node passes a message to its successor. This is called *network unrolling*, depicted in Figure 2.2. Unrolled network stresses the relation of the RNN to the sequential data streams. This is an architecture of choice when modeling such data.

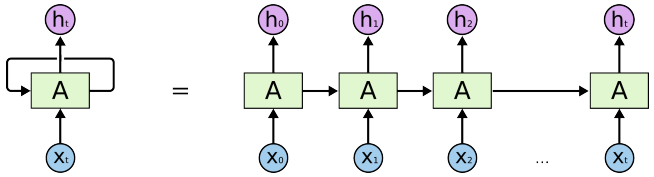


Figure 2.2: An unrolled RNN

It turns out that the vanilla RNN are hard to train due to the exploding/imploding gradients. Fortunately, Long Short Term Memory Networks (LSTM), which is a special kind of RNN addresses these issues.

LSTM were introduced by [**?**], they are specifically designed to model long term dependencies. Similarly to RNN, LSTM posses a chain-like structure. Each repeating node has an internal structure depicted in Figure 2.3.

To use the LSTM for the image data we combine the convolutional network with the LSTM. X_t vectors are produced by the ZF convnet. This is a fairly common representation used previously. The convnet and the LSTM are trained jointly.

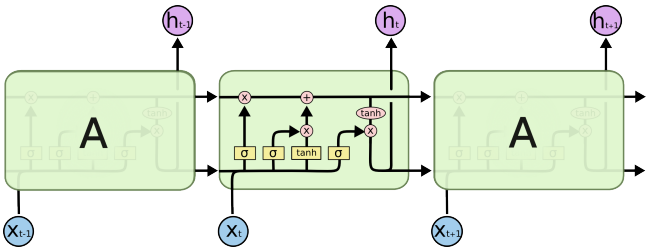


Figure 2.3: Long Short Term Memory

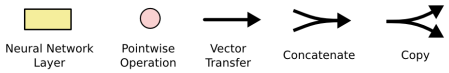
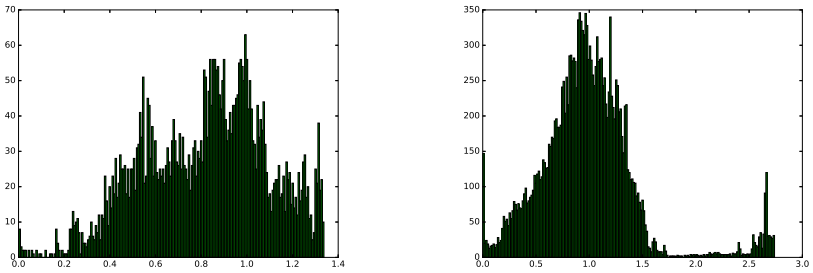


Figure 2.4: A notation used in Figure 2.3

2.6 Experiments

2.6.1 Data-set

We train and test on the KITTI data-set [**?**]. The data-set consists of 11 sequences with ground truth data. We arbitrarily use sequence 00 for testing and sequences 01-10 for training. There are 4540 and 18650 images in the test and the train sets respectively. We only use the data from the left color camera. Figure 2.5 depicts the distribution of the scale values over the train and the test sets.



(a) The test set (b) The train set

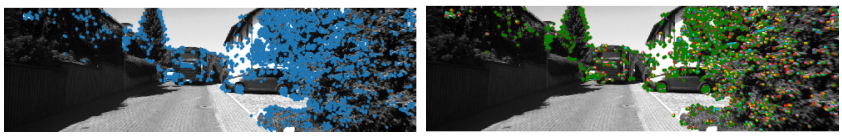
Figure 2.5: Scale distributions

2.6.2 Random Forest

2.6.3 Feature Extraction

In order to train the random forest we need to represent subsequent image pairs as feature vectors, which are created as follows: we extract and match sparse salient points in both input images. The output of this stage is a set of a corresponding pixel locations. Then, we compute the point displacement magnitudes. Finally, we bin all the interest pundits according to a grid and create histogram of displacement magnitudes for each bin. Concatenating all histograms together produces a feature vector. We use Harris corners and square 11×11 patches as corner descriptors. Sum of square differences is used as a distance measure with the winning pair declared a match. To prune the outliers we fit the fundamental matrix into the matched corner sets and remove the corners that do not agree with the model. Figure 2.6 shows a typical example of extracted and matched corners.

Some statistics of the features is presented in the Figure 2.7. We expect the peaks, that correspond to a closer image regions have a distributions shifted to the right (i.e., larger displacements) and the peaks that correspond to a regions farther away should be closer to zero. This behavior can be observed especially well for the feature vectors that correspond to larger camera displacements (e.g. Figure 2.7c).



(a) (b)

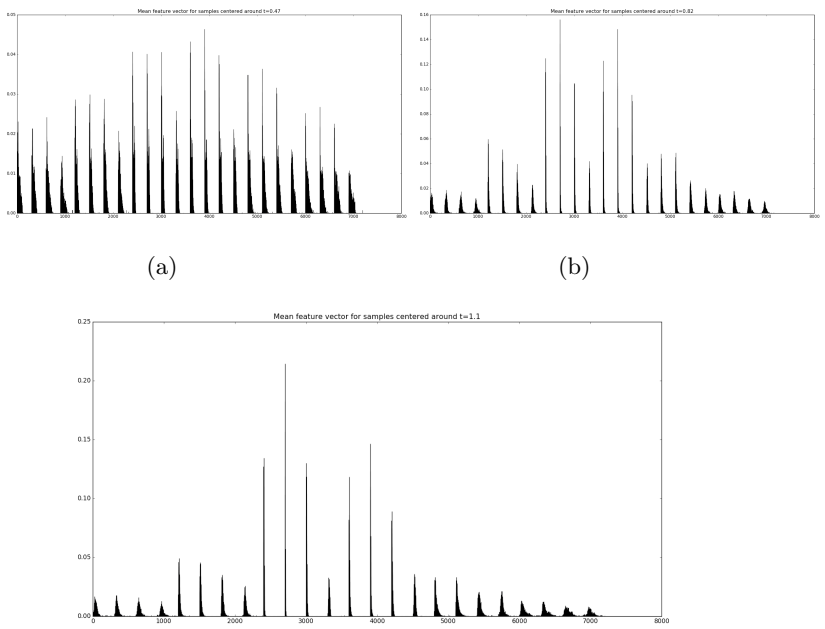
Figure 2.6: Typical corner extraction and matching. Figure a shows the raw extracted corners, while Figure b shows pruned and matched corners.

We bin each image into 6×4 grid. For each bin in the image we compute the histogram of corner disparities. By disparity we denote the displacement of the corner in the image. We use 300-bin histogram for disparities (e.g, feature vector length is 7200).

We use the extracted features to fit a random forest, by means of recursive note splitting with sub-node variance minimization. The mean absolute error is 0.174 meters with a standard deviation of 0.148 meters. The visualization of a prediction is in the Figure 2.8.

2.7 Convolutional neural networks

We use Caffe [**?**] framework to train and test our models. The mean absolute error is 0.2 meters with a standard deviation of 0.16 meters.



(a) (b)

(c)

Figure 2.7: Average feature vectors for samples centered around the specific camera translation magnitude. Each peak corresponds to a grid cell (e.g. here the grid is 4 rows by 6 columns by 300 bins, so the feature vector is of the dimension 6*4*300=7200). The grid is sampled in a column-major mode. So the first four peaks correspond to the leftmost column of the image grid.

2.8 LSTM

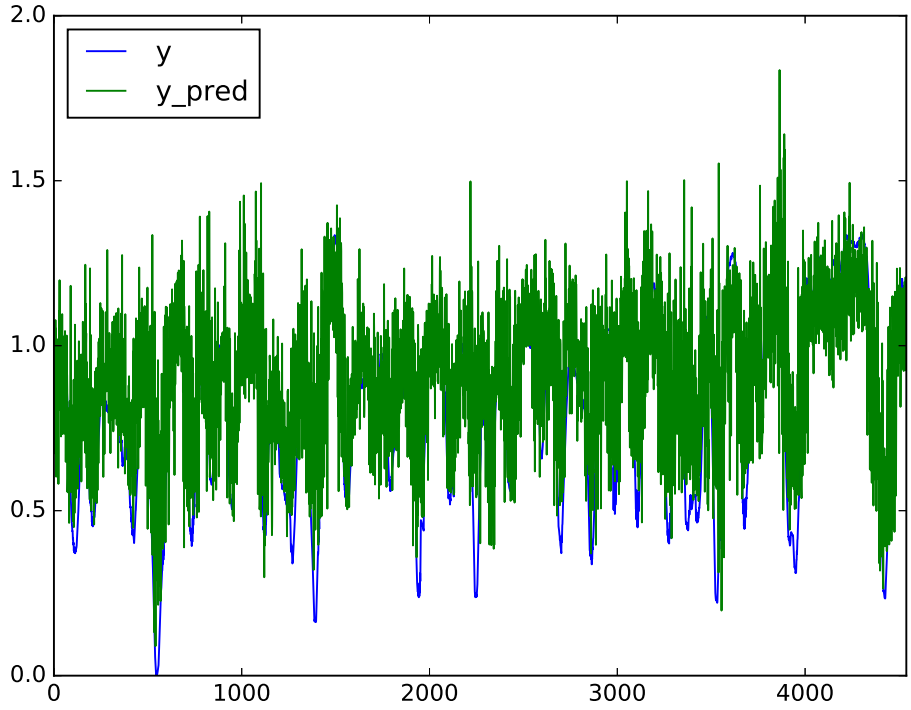


Figure 2.8: Visualization of the random forest prediction

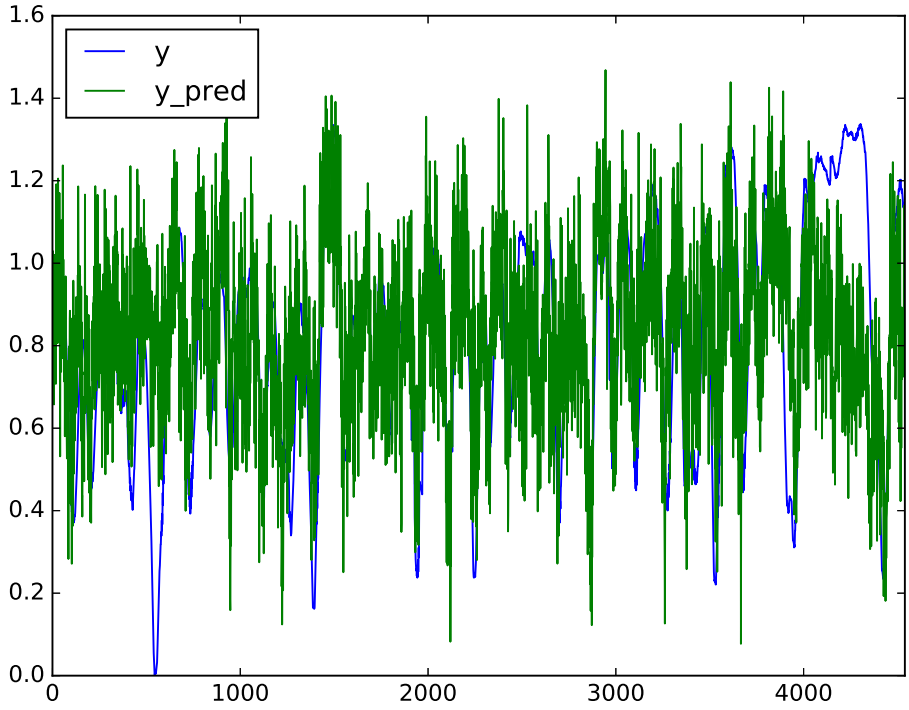


Figure 2.9: Visualization of the ZF CNN prediction