Subspace Constraint Method of Feature Tracking

Abstract

Feature tracking in video is a crucial task in computer vision. Usually, the tracking problem is handled one feature at a time, using a single-feature tracker like the Kanade-Lucas-Tomasi algorithm, or one of its derivatives. While this approach works quite well when dealing with high- quality video and "strong" features, it often falters when faced with dark and noisy video containing low-quality features. We present a framework for jointly tracking a set of features, which enables sharing information between the different features in the scene. We show that our method can be employed to track features for both rigid and non- rigid motions (possibly of few moving bodies) even when some features are occluded. Furthermore, it can be used to significantly improve tracking results in poorly-lit scenes (where there is a mix of good and bad features). Our approach does not require direct modeling of the structure or the motion of the scene, and runs in real time on a single CPU core.

1 Introduction

Feature tracking in video is an important computer vision task, often used as the first step in finding structure from motion or simultaneous location and mapping (SLAM). The celebrated Kanade-Lucas-Tomasi algorithm tracks feature points by searching for matches between templates representing each feature and a frame of video. Despite many other alternatives and improvement, it is still one of the best video feature tracking algorithms. However, there are several realistic scenarios when Lucas-Kanade and many of its alternatives do not perform well: poor lighting conditions, noisy video, and when there are transient occlusions that need to be ignored. In order to deal with such scenarios more robustly it would be useful to allow the feature points to communicate with each other to decide how they should move as a group, so as to respect the underlying three dimensional geometry of the scene.

This underlying geometry constrains the trajectories of the track points to have a low-rank structure for the case when tracking a single rigid object under an affine camera model, and for non-rigid motion and the perspective camera. In this work we will combine the low-rank geometry of the cohort of tracked features with the successful non-linear single feature tracking framework of Lucas and Kanade by adding a low-rank regularization penalty in the tracking optimization problem. To accommodate dynamic scenes with non-trivial motion we apply our rank constraint over a sliding window, so that we only consider a small number of frames at a given time (this is a common idea for dealing with non-rigid motions). We demonstrate very strong performance in rigid environments as well as in scenes with multiple and/or non- rigid motion (since the trajectories of all features are still low rank for short time intervals). We describe experiments with several choices of low-rank regularizers (which are local in time), using a unified optimization framework that allows real time regularized tracking on a single CPU core.

2 On Low-Rank Feature Trajectories

Under the affine camera model, the feature trajectories for a set of features from a rigid body should exist in an affine subspace of dimension 3, or a linear subspace of dimension 4. However, subspaces

.

corresponding to very degenerate motion are lower-dimensional those corresponding to general motion.

Feature trajectories of non-rigid scenarios exhibit significant variety, but some low-rank models may still be successfully applied to them. we consider a sliding temporal window, where over short durations the motion is simple and the feature trajectories are of lower rank. The restriction on the length of feature trajectories can also help in satisfying an approximate local affine camera model in scenes which violate the affine camera model. In general, depth disparities give rise to low-dimensional manifolds which are only locally approximated by linear spaces.

At last, even in the case of multiple moving rigid objects, the set of trajectories is still low rank (confined to the union of a few low rank subspaces). In all of these scenarios the low rank is unknown in general.

3 Feature Tracking

Notation: A feature at a location z1 $\in R^2$ in a given $N_1 \times N_2$ frame of an $N_1 \times N_2 \times N_3$ video is characterized by a template T, which is an n × n sub-image of that frame centered at z_1 (n is a small integer, generally taken to be odd, so the template has a center pixel). If z_1 does not have integer coordinates, T is interpolated from the image. We denote Ω = 1, ..., n × 1, ..., n and we parametrize T so that its pixel values are obtained by $T(u)_u \in \Omega$.

A classical formulation of the single-feature tracking problem is to search for the translation x_1 that minimizes some distance between a feature's template T at a given frame and the next frame of video translated by x_1; we denote this next frame by I. That is, we minimize the single-feature energy function c(x_1):

$$c(x_1) = \frac{1}{2} \sum_{u \in \Omega} \psi(T(u) - I(u + x_1))$$

where, for example, ψ(x) = |x| or ψ(x) = x^2. To apply continuous optimization we view x_1 as a continuous variable and we thus view T and I as functions over continuous domains (implemented with bi-linear interpolation).

3.1 Low Rank Regularization Framework

If we want to encourage a low rank structure in the trajectories, we cannot view the tracking of different features as separate problems. For f $\in$ 1, 2, ..., F, let x_f denote the position of feature f in the current frame (in image coordinates), and let x = (x_1, x_2, ..., x_F) $\in R^{2F}$ denote the joint state of all features in the scene. We define the total energy function as follows:

$$C(x) = \frac{1}{F_{n^2}} \sum_{f=1}^{F} \sum_{u \in \Omega} \psi(T_f(u) - I(u + x_f))$$

where T_f(u) is the template for feature f. Now, we can impose desired relationships between features in a scene by imposing constraints on the domain of optimization.

Instead of enforcing a hard constraint, we add a penalty term to, which increases the cost of states which are inconsistent with low-rank motion. Specifically, we define:

$$C(x) = \alpha \sum_{f=1}^{F} \sum_{u \in \Omega} \psi(T_f(u) - I(u + x_f)) + P(x)$$

where P(x) is an estimate of, or proxy for, the dimensionality of the set of feature trajectories over the last several frames of video (past feature locations are treated as constants, so this is a function only of the current state, x). Notice that we have replaced the scale factor 1/(Fn^2) from with the constant α, as this coefficient is now also responsible for controlling the relative strength of the penalty term. We will give explicit examples for P in section 3.2.

This framework gives rise to two different solutions, characterized by the strength of the penalty term (definition of α). Each has useful, real-world tracking applications. In the first case, we assume that most (but not necessarily all) features in the scene approximately obey a low rank model. This is appropriate if the scene contains non-rigid or multiple moving bodies. We can impose a weak constraint by making the penalty term small relative to the other terms. If a feature is strong, it will confidently track the imagery, ignoring the constraint (regardless of whether the motion is consistent with the other features in the scene). If a feature is weak in the sense that we cannot fully determine its true location by only looking at the imagery, then the penalty term will become significant and encourage the feature to agree with the motion of the other features in the scene.

In the second case, we assume that all features in the scene are supposed to agree with a low rank model (and deviations from that model are indicative of tracking errors). We can impose a strong constraint by making the penalty term large relative to the other terms. No small set of features can overpower the constraint, regardless of how strong the features are. This forces all features to move is a way that is consistent with a simple motion. Thus, a small number of features can even be occluded, and their positions will be predicted by the motion of the other features in the scene.

3.2 Specific Choices of the Low-Rank Regularizer

There is now a large body of work on low rank regularization. We will restrict ourselves to showing results using three choices for P described below. Each choice we present defines P(x) in terms of a matrix M. It is the 2(L + 1) × F matrix whose column f contains the feature trajectory for feature f within a sliding window of L + 1 consecutive frames (current frame and L past frames). Specifically, M = $[m_{i,j}]$, where $(m_{0,f}\ , m_{1,f})^T$ is the current (variable) position of feature f and $(m_{2l+1,f}\ , m_{2l+2,f})^T$, l = 1, ..., L contains the x and y pixel coordinates of feature f from l frames in the past (past feature locations are treated as known constants). One may alternatively center the columns of M by subtracting from each column the average of all columns. Most constraints derived for trajectories actually confine trajectories to a low rank affine subspace (as opposed to a linear subspace). Centering the columns of M transforms an affine constraint into a linear one. Alternatively, one can forgo centering and view an affine constraint as a linear constraint in one dimension higher. We report results for both approaches.

3.2.1 Explicit Factorizations

A simple method for enforcing the structure constraint is to write M = BC, where B is a 2(L+1)×d matrix, and C is a d × F matrix. However, as mentioned in the previous section, because the feature tracks often do not lie exactly on a subspace due to deviations from the camera model or non- rigidity, an explicit constraint of this form is not suitable.

However, an explicit factorization can be used in a penalty term by measuring the deviation of M, in some norm, from its approximate low rank factorization. For example, if we let

$$M = U\Sigma V^T$$

denote the SVD of M, we can take P(x) to be ||BC M||, where B is the first three or four columns of U, and C is the first three or four rows of V^T. Then this P corresponds to penalizing M via $\sum_{i=d+1}^{F} \sigma_i$, where σ_i = λ_{ii} is the i^{th} singular value of M. As above, since the history is fixed, U, Σ, and V^T are functions of x.

This approach assumes knowledge of the low-rank d. For simplicity, we assume a local rigid model and thus set d = 3 when centering M and d = 4 when not centering.

3.2.2 Nuclear Norm

A popular alternative to explicitly keeping track of the best fit low-dimensional subspace to M is to use the matrix nuclear norm and define

$$P(x) = ||M||_* = ||\sigma||_1$$

This is a convex proxy for the rank of M. Here σ = $(\sigma_1\ \sigma_2\ .\ .\ .\ \sigma_{2(L+1)\wedge F})^T$ is the vector of singular values of M, and || · ||1 is the l_1 norm. Unlike explicit factorization, where only energy outside the first d principal components of M is punished, the nuclear norm will favor lower-rank M over higher-rank M even when both matrices have rank d. Thus, using this kind of penalty will favor simpler track point motions over more complex ones, even when both are technically permissible.

3.2.3 Empirical Dimension

Empirical Dimension refers to a class of dimension estimators depending on a parameter $\epsilon \in$ (0,1]. The empirical dimension of M is defined to be:

$$d_\epsilon(M) = \frac{||\sigma||_1^\epsilon}{||\sigma||_\epsilon^\epsilon}$$

Notice that we use norm notation, although $\| \cdot \|_\epsilon$ is only a pseudo-norm. When ϵ = 1, this is sometimes called the "effective rank" of the data matrix.

Empirical dimension satisfies a few important properties. First, empirical dimension is invariant under rotation and scaling of a data set. Additionally, in the absence of noise, empirical dimension never exceeds true dimension, but it approaches true dimension as the number of measurements goes to infinity for spherically symmetric distributions. Thus, d_ϵ is a true dimension estimator (whereas the nuclear norm is a proxy for dimension). To use empirical dimension as our regularizer, we define P(x) = d_ϵ(M).

Empirical dimension is governed by its parameter, ϵ. An ϵ near 0 results in a "strict" estimator, which is appropriate for estimating dimension in situations where you have little noise and you expect your data to live in true linear spaces. If ϵ is near 1 then d_ϵ is a lenient estimator. This makes it less sensitive to noise, and more tolerant of data sets that are only approximately linear. In all of the experiments we present, we use ϵ = 0.6, although we found that other tested values also worked well.

3.3 Implementation Details

We fix L = 10 for the sliding window and let (x) = |x|. We use this form for so that all terms in the total energy function behave linearly in a known range of values. If our fit terms behaved quadratically, it would be more challenging to balance them against a penalty term. We also tested a Huber loss function for and have concluded that such a regularization is not needed.

We fix a parameter m for each penalty form (selected empirically - see the supplementary material for our procedure), which determines the strength of the penalty. The weak and strong regularization parameters are set as follows:

$$\alpha_{weak} = \frac{1}{mn^2} \text{ and } \alpha_{strong} = \frac{1}{mFn^2}$$

The weak scaling implies that a perfectly-matched feature will contribute 0 to the total energy, and a poorly-matched feature will contribute an amount on the order of 1/m to the total energy. The penalty term will contribute on the order of 1 to the total energy. Since we do not divide the contributions of each feature by the number of features, the penalty terms contribution is comparable in magnitude to that of a single feature. The strong scaling implies that the penalty term is on the same scale as the sum of the contributions of all of the features in the scene.

3.3.1 Minimization Strategy

The total energy function we propose for constrained tracking is non-convex since the contributions from the template fit terms are not convex (even if P is convex); this is also the case with other feature tracking methods, including the Lucas-Kanade tracker. We employ a 1st-order descent approach for driving the energy to a local minimum.

To reduce the computational load of feature tracking, some trackers use 2nd-order methods for optimization. This works well when tracking strong features, but in our experience it can be unreliable when dealing with weak or ambiguous features. Since we are explicitly trying to improve tracking accuracy on poor features we opt for a 1st-order descent approach instead.

The simplest 1st-order descent method is (sub)gradient descent. Unfortunately, because there can be a very large difference in magnitude between the contributions of strong and weak features to our total energy, our problem is not well-conditioned. If we pursue standard gradient descent, the strong features dictate the step direction and the weak features have very little effect on it. Ideally, once the strong features are correctly positioned, they will no longer dominate the step direction. If we were able to perfectly measure the gradient of our objective function, this would be the case. In practice, the error in our numerical gradient estimate can be large enough to prevent the strong features from ever relinquishing control over the step direction. The result is that in a scene with both very strong and very weak features, the weak features may not be tracked.

To remedy this, we compute our step direction by blending the gradient of the energy function with a vector that corresponds to taking equal-sized gradient descent steps separately for each feature. We use a fast line search in each iteration to find the nearest local minimum in the step direction. This compromise approach allows for efficient descent while ensuring that each feature has some control over the step direction (regardless of feature strength).

Because the energy is not convex, it is important to choose a good initial state. We use a combination of two strategies to initialize the tracking: first, we generate our initial guess of x by registering an entire frame of video with the previous (at lower resolution). Secondly, we use multi-resolution, or pyramidal tracking so that approximate motion on a large scale can help us get close to the minimum before we try tracking on finer resolution levels.

We now explain the details of the algorithm. Let I denote a full new frame of video and let x_{prev} be the concatenation of feature positions in the previous frame. We form a pyramid for I where level 0 is the full-resolution image and each higher level m (1 through 3) has half the vertical and half the horizontal resolution of level m 1. To initialize the optimization, we take the full frame (at resolution level 3) and register it against the previous frame (also at resolution level 3) using gradient descent and an absolute value loss function. We initialize each features position in the current frame by taking its position in the previous frame and adding the offset between the frames, as found through this registration process). Once we have our initial x, we begin optimization on the top pyramid level. When done on the top level, we use the result to initialize optimization on the level below it, and so on until we have found a local minimum on level 0. On any given pyramid level, we perform optimization by iteratively computing a step direction and conducting a fast line search to find a local minimum in the search direction. We impose a minimum and maximum on the number of steps to be performed on each level (min_i and max_i, respectively). Our termination condition (on a given level) is when the magnitude of the derivative of C is not significantly smaller than it was in the previous step. To compute our search direction in each step, we first compute the gradient of C (which we will call D_C) and set a =

This is done by breaking it into a collection of 2-vectors (elements 1 and 2 are together, elements 3 and 4 are together, and so on) and normalizing each of them. We then recombine the normalized 2-vectors to get b. We blend a with c to compute our step direction. Algorithm 1 summarizes the full process.

3.3.2 Efficiency and Complexity

We have found that our algorithm typically converges in about 20 iterations or less at each pyramid level (with fewer iterations on lower pyramid levels). In our experiments, we used a resolution of 640-by-480 (we have also done tests at 1000 × 562), and we found that 4 pyramid levels were sufficient for reliable tracking. Thus, on average, less than 80 iterations are required to track from one frame to the next. A single iteration requires one gradient evaluation and multiple evaluations of C. The complexity of a gradient evaluation is $k_1Fn^2 + k_2LF^2$, and the complexity of an energy evaluation is $k_3Fn^2 + k_4L^2F$. Our C++ implementation (which makes use of OpenCV) can run on 35 features of size 7-by-7 with a temporal window of 6 frames (L = 5) on a 3rd-generation Intel i5 CPU at approximately 16 frames per second. SIMD instructions are used in places, but no multi-threading was used, so faster processing rates are possible. With a larger window of L = 10 our algorithm still runs at 2-5 frames per second.

4 Experiments

To evaluate our method, we conducted tests on several real video sequences in circumstances that are difficult for feature tracking. These included shaky footage in low-light environments. The resulting videos contained dark regions with few good features and the unsteady camera motion and poor lighting introduced time-varying motion blur.

In these video sequences it proved very difficult to hand-register features for ground-truth. In order to present a quantitative numerical comparison we also collected higher-quality video sequences and synthetically degraded their quality. We used a standard Lucas-Kanade tracker on the non-degraded videos to generate ground-truth (the output was human-verified and corrected). We therefore present qualitative results on real, low-quality video sequences, as well as quantitative results on a set of synthetically degraded videos.

4.1 Qualitative Experiments on Real Videos

In our tests on real video sequences containing low- quality features, single-feature tracking does not provide acceptable results. When following a non-distinctive feature, the single-feature energy function often flattens out in one or more directions. A tracker may move in any ambiguous direction without realizing a better or worse match with the features template. This results in the tracked location drifting away from a features true location (i.e. “wandering”). This is not a technical limitation of one particular tracking implementation. Rather, it is a fundamental problem due to the fact that the local imagery in a small neighborhood of a feature does not always contain enough information to deduce the features motion between frames. This claim can be verified by attempting