2019-05-25

An Iterative Five-Point Algorithm with Application to Multi-Target Tracking

Jacob H. White
Brigham Young University, jacob.white@byu.net

Randal W. Beard
Brigham Young University, beard@byu.edu

An Iterative Five-Point Algorithm with Application to Multi-Target Tracking

Jacob H. White[1], Randal W. Beard[1]

***Abstract*— We present ReSORtSAC: Recursively-seeded optimization, refinement, sample, and consensus. ReSORtSAC is a novel algorithm that can be used to estimate the relative pose between consecutive frames of a video sequence. Relative pose estimation algorithms typically generate a large number of hypotheses from minimum subsets and score them in order to be robust to noise and outliers. The relative pose is often represented using the essential matrix. Previous methods calculate essential matrix hypotheses directly without utilizing prior information. These equations are complex to evaluate and can return up to ten essential matrix solutions for each minimum subset, all of which must be scored.**

Instead, we calculate relative pose hypotheses by optimizing the rotation and translation between frames, rather than calculating the essential matrix directly. The equations used in our optimization are simpler to evaluate, resulting in faster computation speeds. We also reuse the best hypothesis to seed the optimizer which reduces the number of relative pose hypotheses which must be generated and scored. These advantages greatly speed up performance and enable the algorithm to run in real-time, while sharing resources with other computer vision algorithms. We show application of our algorithm to visual multi-target tracking (MTT) in the presence of parallax and demonstrate its real-time performance on a 640×480 video sequence. Video results are available at https://youtu.be/HhK-p2hXNnU.

I. Introduction

Motion estimation from a video sequence has many applications in robotics including target tracking, visual odometry, and 3D scene reconstruction. These applications often require on-board processing of the video sequence in real-time and have size, weight, and power (SWAP) constraints. Furthermore, the motion estimation algorithms often must share computational resources with other computer vision algorithms on the robot.

One method of estimating motion from a video sequence is by calculating the essential matrix between consecutive frames. The essential matrix relates the normalized image plane coordinates in one frame to the next frame using the epipolar constraint. The essential matrix can be decomposed into a rotation and a normalized translation to determine the motion of the camera. The essential matrix is typically calculated by generating a large number of hypotheses from five-point minimum subsets of matching features in order to be robust to noise and outliers. This is often done using random sample consensus (RANSAC) [1] or least median of squares (LMedS) [2]. When using RANSAC, these hypotheses are scored by counting the number of inlier points from the entire set. When using LMedS, these hypotheses are scored by calculating the median error.

Previous methods calculate essential matrix hypotheses directly from each five-point minimum subset. One of the most well-known of these algorithms is Nister's algorithm [3]. Nister showed that for five matching points, the essential matrix can have up to ten solutions, all of which can be found by solving a tenth-order polynomial, where some of the roots are complex and can be discarded. Since the number of complex roots come in pairs, there always remain an even number of real roots. On average there are typically four valid solutions. There are a number of open-source implementation of Nister's five-point algorithm. One of the most popular implementations is OpenCV's `findEssentialMat` function [4]. However, constructing, solving, and extracting the essential matrix from this tenth-order polynomial is complex and can be computationally expensive. Furthermore, since each minimum subset produces up to ten hypotheses, it can be time consuming to score them.

As an alternative to directly calculating essential matrix solutions, some authors [5], [6], [7], [8], [9] propose solving for the essential matrix using non-linear optimization algorithms such as Gauss-Newton (GN) and Levenberg-Marquardt (LM). Since the essential matrix has nine entries but only five degrees of freedom, the optimization is done on the essential matrix manifold to ensure that the result is valid. There are a number of ways to define the essential matrix manifold. Some authors define the manifold using a rotation and translation unit vector, which are elements of $SO(3)$ and S^2 respectively [5], [6]. Others define the manifold using two elements of $SO(3)$, in an SVD-like product [8], [9].

During each iteration of the optimization algorithm, the optimizer step is solved for in terms of the five degrees of freedom along the manifold. Since $SO(3)$ and S^2 are or can be represented using Lie groups, the matrix exponential can then be used to map the optimizer step back to the manifold in order to perform the update. Each iteration of the optimization is quite fast because the cost function and its derivatives are mathematically simple. The most time-consuming part of the iteration is inverting a 5×5 matrix, which is faster than solving a tenth-order polynomial. For this reason, these optimization-based solvers have the potential to be faster than direct essential matrix solvers.

*This work has been funded by the Center for Unmanned Aircraft Systems (C-UAS), a National Science Foundation Industry/University Cooperative Research Center (I/UCRC) under NSF award Numbers IIP-1161036 and CNS-1650547, along with significant contributions from C-UAS industry members.

[1]J. H. White, and R. W. Beard are with the Department of Electrical and Computer Engineering, Brigham Young University, Provo, UT 84602, USA

However, one weakness of these optimization-based solvers is that they only find one of the ten possible solutions to the essential matrix at a time. Finding all solutions requires additional optimization runs with different initialization points. The optimization method is also sensitive to initial conditions, which can cause the optimizer to fail to produce a valid solution. For example, GN may diverge if the initial guess is too far from the true solution. LM can be used to prevent increases in the cost function, but may occasionally still fail to converge. Because of the need to run the optimizer more times to produce the same number of hypotheses, these existing optimization-based solvers might not necessarily be faster than the direct essential matrix solvers if the same level of accuracy is desired.

However, not all of the ten possible solutions are needed in order achieve comparable accuracy to direct essential matrix solvers, if the best solution can be found the first time. The main contribution of this work is a novel optimization-based relative pose solver that leverages prior information in order to find the desired solution without requiring additional optimization runs. Building on previous work, our algorithm utilizes LMedS to produce robust estimates in the presence of noise and outliers. We use the LM optimizer to solve for the relative pose from each minimum subset.

The main difference between our work and previous work is that during each iteration of LMedS, we seed the optimizer with an informed initial guess instead of a random seed. In video sequences, the rotation and translation between consecutive frames is similar to nearby frames, so we can use the relative pose estimate from the previous time step as a good initial guess of the relative pose at the current time step. Then in subsequent LMedS iterations, we update this initial guess whenever a better relative pose hypothesis is found. We show that this change significantly reduces the number of hypotheses that must be generated and scored to achieve the same level of accuracy. This allows the algorithm to run in real-time while sharing resources with other computer vision algorithms.

After the best LMedS hypothesis is found, it is refined using only inlier points. The same optimizer described above can easily be extended to accept more than five points. In contrast, direct essential matrix solvers are unable to refine the best hypothesis and thus require a separate algorithm to do so.

The remainder of the paper is outlined as follows. Section 2 finalizes the problem description. Section 3 develops our iterative five-point algorithm. Section 4 applies the algorithm to target tracking in the presence of parallax. Section 5 presents results of both the iterative five-point algorithm and the target tracking algorithm. Conclusions are given in section 6.

II. Problem Definition

Given two consecutive video frames with point correspondences detected in each frame, the goal is to find the rotation R_1^2 and translation $t_{1/2}^2$ between the two frames. The relevant geometry is shown below in Figure 1.

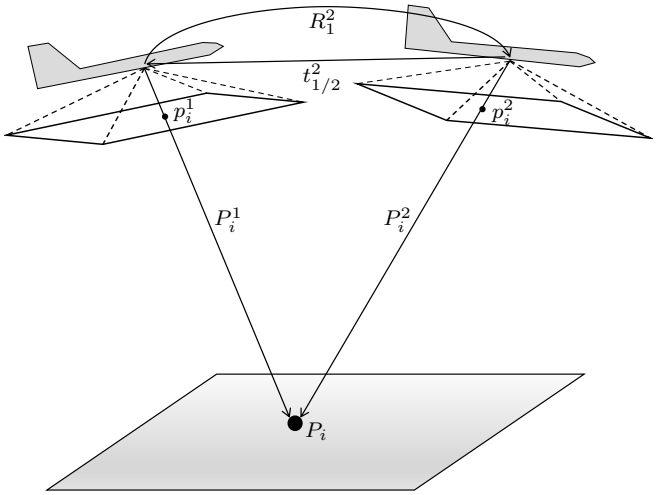

Fig. 1. The geometry for the derivation of the essential matrix

Let $P_i^1 = \begin{bmatrix} X_i^1 & Y_i^1 & Z_i^1 \end{bmatrix}$ and $P_i^2 = \begin{bmatrix} X_i^2 & Y_i^2 & Z_i^2 \end{bmatrix}$ be the 3D position of point i with respect to camera frames 1 and 2. The equation describing the relationship between the 3D coordinates of each point in frame 1 and 2 is

$$P_i^2 = R_1^2 P_i^1 + t_{1/2}^2,$$

where $R_1^2 \in SO(3)$ is the rotation matrix from frame 1 to 2 and $t_{1/2}^2$ is the translation vector from frame 2 to frame 1, resolved in frame 2.

Left multiplying each side of the equation by $\left(t_{1/2}^2\right)_\times$, where

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix},$$

gives

$$\left(t_{1/2}^2\right)_\times P_i^2 = \left(t_{1/2}^2\right)_\times R_1^2 P_i^1 + \left(t_{1/2}^2\right)_\times t_{1/2}^2.$$

The last term on the right is zero, thus

$$\left(t_{1/2}^2\right)_\times P_i^2 = \left(t_{1/2}^2\right)_\times R_1^2 P_i^1.$$

We can then left-multiply each side of the equation by $\left(P_i^2\right)^\top$ to get

$$\left(P_i^2\right)^\top \left(t_{1/2}^2\right)_\times P_i^2 = \left(P_i^2\right)^\top \left(t_{1/2}^2\right)_\times R_1^2 P_i^1.$$

However, the left side of the equation is always zero because the cross product $\left(t_{1/2}^2\right)_\times P_i^2$ gives a vector perpendicular to P_i^2, and therefore

$$\left(P_i^2\right)^\top \left(t_{1/2}^2\right)_\times R_1^2 P_i^1 = 0.$$

Since the right side of the equation is zero, any scalar multiple of P_i^1 and P_i^2 will also satisfy the equation.

This removes the depth dependency and allows us to use normalized image plane coordinates in the equation. Let p_i^1 and p_i^2 be the normalized homogeneous image coordinates of point i in camera frame 1 and 2 respectively, i.e.

$$p_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{X_i}{Z_i} \\ \frac{Y_i}{Z_i} \\ 1 \end{bmatrix} = \frac{P_i}{Z_i}.$$

This gives

$$\left(p_i^2\right)^\top \left(t_{1/2}^2\right)_\times R_1^2 p_i^1 = 0. \tag{1}$$

The essential matrix is defined as

$$E \triangleq \left(t_{1/2}^2\right)_\times R_1^2. \tag{2}$$

In our method, we optimize the rotation and translation on the manifold to satisfy Equation (1). The essential matrix is only produced from the rotation and translation when it is needed for cost functions and derivatives. In contrast, many approaches in the literature solve for the essential matrix directly, which requires decomposing the essential matrix later to obtain the desired rotation and translation.

III. New Iterative Five-Point Algorithm

In this section, we drop the frame notation and simply write the rotation as R and the translation as t, where $R \in SO(3)$ and $t \in S^2$. To estimate the rotation and translation between frames, we generate a large number of relative pose hypotheses from five-point minimum subsets and score them using LMedS. However, instead of calculating the essential matrix in closed-form, we optimize the rotation and translation components of the essential matrix to find the solution.

The optimization is done on the $SO(3) \times S^2$ manifold. We will call this the essential matrix manifold. This manifold can be represented using Lie groups, as discussed in subsections III-A-III-C

At each iteration, the optimizer requires derivatives with respect to each of the five degrees of freedom. Additionally, since the optimizer step is represented in these five degrees of freedom, it must be mapped back to the Lie group using the matrix exponential. These derivatives and the exponential map are also discussed in subsections III-A-III-C

The optimization algorithm minimizes the Sampson error using the Levenberg-Marquardt algorithm. This error and its derivatives are described in subsection III-D. The Levenberg-Marquardt optimization algorithm used is described in subsection III-E.

At each iteration of the LMedS algorithm, the optimization produces a relative pose hypothesis from a random five-point minimum subset. The best hypothesis is always used to seed the optimizer. The LMedS algorithm is described in subsection III-F and the optimizer seeding method is described in subsection III-G.

The best relative pose hypothesis is refined using only inlier points. This refinement is performed using the same optimization algorithm used to generate the rotation and translation hypotheses. The refinement is described in subsection III-H.

Lastly, since there are two rotations and two translations for which the Sampson error produces the same cost value, the correct rotation-translation pair must determined. Since the cheirality check often gives the wrong rotation matrix, we instead pick the rotation matrix with the smallest angle. We only use the cheirality check to determine the correct translation. This is described in subsection III-I.

A. The $SO(3)$ Manifold

The $SO(3)$ manifold is the three-dimensional Lie group consisting of all 3×3 rotation matrices. To be able to use elements of $SO(3)$ in an iterative algorithm, these three degrees of freedom must be defined and the derivatives in each of these directions calculated. These will later be used to form the Jacobian in the LM optimization. The LM algorithm returns updates to the Lie group represented in the Lie algebra. Thus we also need a method to map between the Lie group and the Lie algebra. This can be done using the matrix exponential and the matrix logarithm. Thus to update elements of the Lie group by an incremental amount $\delta_R = \begin{bmatrix} \delta_{R,1} & \delta_{R,2} & \delta_{R,3} \end{bmatrix}^\top \in \mathfrak{so}(3)$, we will use

$$R_{k+1} = e^{(\delta_R)_\times} R_k.$$

To simplify notation, we will define this update using the boxplus operator, as is done in [10],

$$\boxplus : SO(3) \times \mathbb{R}^3 \to SO(3),$$

$$R_k \boxplus \delta_R \triangleq e^{(\delta_R)_\times} R_k.$$

For skew symmetric 3×3 matrices the matrix exponential can be calculated efficiently using the Rodrigues formula

$$e^{\omega_\times} = I + \sin(\theta)\,\hat{\omega}_\times + (1 - \cos\theta)\,\hat{\omega}_\times^2, \tag{3}$$

where

$$\hat{\omega} = \frac{\omega}{\|\omega\|}$$

and

$$\theta = \|\omega\|.$$

The derivative of the rotation matrix R_{k+1} along the direction $\hat{\omega}$, evaluated at $\theta = 0$ is given by

$$\begin{aligned} \frac{\partial}{\partial\theta} R_{k+1} &= \frac{\partial}{\partial\theta} e^{\theta\hat{\omega}_\times} R_k \Big|_{\theta=\mathbf{0}} \\ &= \hat{\omega}_\times e^{\theta\hat{\omega}_\times} R_k \Big|_{\theta=\mathbf{0}} \\ &= \hat{\omega}_\times R_k. \end{aligned}$$

Thus the derivatives of $R \in SO(3)$ along each of the three degrees of freedom are

$$\frac{\partial}{\partial\delta_{R,1}} R_{k+1} = (e_1)_\times R_k,$$

$$\frac{\partial}{\partial \delta_{R,2}} R_{k+1} = (e_2)_\times R_k,$$

and

$$\frac{\partial}{\partial \delta_{R,3}} R_{k+1} = (e_3)_\times R_k,$$

where

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

$$e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix},$$

and

$$e_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

B. The S^2 Manifold

The S^2 manifold is the set of all 3×1 unit vectors and has two degrees of freedom. However, S^2 is not a Lie group, so instead we will represent it internally using elements of $SO(3)$. To obtain an S^2 unit vector, we will multiply this internal rotation matrix by a unit vector in the $\hat{z}$ direction to give

$$t = Q^\top e_3.$$

We will define an incremental update on the internal rotation matrix Q by $\delta_t \in \mathbb{R}^2$ as

$$Q_{k+1} = e^{\begin{bmatrix} \delta_{t,1} \\ \delta_{t,2} \\ 0 \end{bmatrix}_\times} Q_k.$$

Once again, we will define a boxplus operator to condense notation,

$$\boxplus : S^2 \times \mathbb{R}^2 \to S^2,$$

$$Q_{k+1} = Q_k \boxplus \delta_t \triangleq e^{\begin{bmatrix} \delta_{t,1} \\ \delta_{t,2} \\ 0 \end{bmatrix}_\times} Q_k.$$

Taking derivatives in both degrees of freedom gives

$$\begin{aligned} \frac{\partial}{\partial \delta_{t,1}} Q_{k+1} &= \left((e_1)_\times Q_k\right)^\top e_3 \\ &= Q_k^\top \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} e_3 \\ &= Q_k^\top e_2 \end{aligned}$$

and

$$\begin{aligned} \frac{\partial}{\partial \delta_{t,2}} Q_{k+1} &= \left((e_2)_\times Q_k\right)^\top e_3 \\ &= Q_k^\top \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} e_3 \\ &= -Q_k^\top e_1. \end{aligned}$$

C. The Essential Matrix Manifold

The essential matrix is the product of the skew symmetric matrix of a translation unit vector and a rotation matrix:

$$E = t_\times R.$$

Therefore, define the essential matrix manifold as the Cartesian product of the rotation matrix and the unit vector translation

$$\boldsymbol{E} \in \left\{SO(3) \times S^2\right\}.$$

As before, we will define a boxplus operator to update elements of the essential matrix manifold by an incremental update $\delta \in \mathbb{R}^5$,

$$\boxplus : \left\{SO(3) \times S^2\right\} \times \mathbb{R}^5 \to \left\{SO(3) \times S^2\right\},$$

$$\boldsymbol{E} \boxplus \delta \triangleq \left\{ R \boxplus \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{bmatrix}, Q \boxplus \begin{bmatrix} \delta_4 \\ \delta_5 \end{bmatrix} \right\}. \tag{4}$$

The derivatives of the essential matrix in each of the five degrees of freedom are

$$\frac{\partial E_{k+1}}{\partial \delta_1} = t_\times (e_1)_\times R_k, \tag{5}$$

$$\frac{\partial E_{k+1}}{\partial \delta_2} = t_\times (e_2)_\times R_k, \tag{6}$$

$$\frac{\partial E_{k+1}}{\partial \delta_3} = t_\times (e_3)_\times R_k, \tag{7}$$

$$\frac{\partial E_{k+1}}{\partial \delta_4} = \left((e_1)_\times Q\right)_\times^\top e_3 R_k, \tag{8}$$

and

$$\frac{\partial E_{k+1}}{\partial \delta_5} = \left((e_2)_\times Q\right)_\times^\top e_3 R_k. \tag{9}$$

D. Sampson Error

The Sampson error is a well-known approximation of the reprojection error in both images [11]. This error will be used as the residual function in the Levenberg-Marquardt optimization. The Sampson error for the ith point is

$$r_i = \frac{\left(p_i^2\right)^\top E p_i^1}{\sqrt{(Ep_i^1)_1^2 + (Ep_i^1)_2^2 + (E^\top p_i^2)_1^2 + (E^\top p_i^2)_2^2}}. \tag{10}$$

The derivative of the Sampson error with respect to the jth degree of freedom is

$$\frac{\partial r_i}{\partial \delta_j} = \frac{\sqrt{s}\left(p_i^2\right)^\top \frac{\partial E}{\partial \delta_j} p_i^1 - \left(p_i^2\right)^\top E p_i^1 \frac{1}{\sqrt{s}} \frac{\partial s}{\partial \delta_j}}{s}, \tag{11}$$

where

$$s = \left(Ep_i^1\right)_1^2 + \left(Ep_i^1\right)_2^2 + \left(E^\top p_i^2\right)_1^2 + \left(E^\top p_i^2\right)_2^2,$$

and $\frac{\partial E}{\partial \delta_j}$ is given in equations (5)-(9).

E. Levenberg-Marquardt Optimization

Both the Gauss-Newton and Levenberg-Marquardt optimizers begin with an initial guess of the rotation matrix and the translation unit vector. The Sampson error (10) can be evaluated at this initial guess for each point to construct the residual vector

$$r = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix}.$$

The Jacobian of the residual vector is constructed from the residual derivatives (11) in each of the five degrees of freedom along the essential matrix manifold described in section III-C. This gives

$$J = \begin{bmatrix} \frac{\partial r_1}{\partial \delta_1} & \frac{\partial r_1}{\partial \delta_2} & \frac{\partial r_1}{\partial \delta_3} & \frac{\partial r_1}{\partial \delta_4} & \frac{\partial r_1}{\partial \delta_5} \\ \frac{\partial r_2}{\partial \delta_1} & \frac{\partial r_2}{\partial \delta_2} & \frac{\partial r_2}{\partial \delta_3} & \frac{\partial r_2}{\partial \delta_4} & \frac{\partial r_2}{\partial \delta_5} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_m}{\partial \delta_1} & \frac{\partial r_m}{\partial \delta_2} & \frac{\partial r_m}{\partial \delta_3} & \frac{\partial r_m}{\partial \delta_4} & \frac{\partial r_m}{\partial \delta_5} \end{bmatrix}.$$

This Jacobian is a first-order Taylor-series approximation of the error function. In other words, the residual can be approximated by

$$r \approx r_0 + J_0 \delta.$$

The goal is to bring the residual to zero. Based on this linear approximation of the error, we can solve for the Gauss-Newton incremental update δ_{GN} using least squares, so that

$$\delta_{GN} = -\left(J_k^\top J_k\right)^{-1} J_k^\top r_k.$$

The essential matrix manifold element is then updated using the boxplus operator defined in Equation (4).

Gauss-Newton iterations are repeated until the norm of the residual is lower than a threshold, or the maximum number of iterations is reached. In our implementation we use a maximum of ten iterations.

However, if the initial guess is too far from the true solution, the Gauss-Newton algorithm can take longer to converge or become unstable. The Levenberg-Marquardt algorithm is an improvement on the Gauss-Newton algorithm that performs better in these conditions. It is essentially a hybrid between the Gauss-Newton algorithm and gradient descent, with a parameter λ to mix the two algorithms, so that

$$\delta_{LM} = -\left(J_k^\top J_k + \lambda \left(J_k^\top J_k\right)_{diag}\right)^{-1} J_k^\top r_k,$$

where $(\cdot)_{diag}$ indicates a matrix formed by only using the diagonal elements of the matrix.

F. Consensus Algorithms

The two consensus algorithms used in this paper are random sample consensus (RANSAC) [1] and least median of squares (LMedS) [2]. At each iteration of RANSAC or LMedS, a minimum subset of size n is chosen from which to generate a model. In the case of solving for the relative pose between frames, the subset is chosen from the m point correspondences at the current time step. The subset is called a minimum subset because it is the smallest number of points that provide enough constrains to generate a solution, or model. In the case of the solving for the relative pose, $n = 5$ point correspondences are used, since there are five degrees of freedom and each point correspondence gives one constraint. For each minimum subset, one or more models that fit the points are solved for.

After generating N minimum subsets, each with their corresponding solutions, all of these models are scored to find the best one. The difference between RANSAC and LMedS is how the models are scored. For RANSAC, the score is the total number of inliers with a residual lower than the RANSAC threshold τ. This can be written as a cost function if the number of outliers are counted instead of the number of inliers, so that

$$cost = \sum_{i=0}^{m} \begin{cases} 1 & |r_i| > \tau \\ 0 & \text{otherwise.} \end{cases}$$

For LMedS, the cost is instead calculated by finding the median of the square of the residuals, so that

$$cost = \underset{i=0}{\overset{m}{\text{median}}} \; r_i^2.$$

The advantage to using LMedS is that the threshold does not need to be known before-hand. However, it only works if less than half of the points are outliers.

The number of iterations required when using RANSAC or LMedS is typically solved for by treating each matched pair of points as a binomial random variable representing whether the point is an inlier or an outlier [12]. For example, achieving a 99% confidence ratio when the outlier ratio is 50% requires 145 RANSAC or LMedS iterations.

G. Seeding the Optimizer

The method in which the optimizer is seeded is essential to achieving good performance. It is important that the optimizer be seeded with a good initial guess. When no prior information is known about the camera motion, the algorithm can be seeded with the identity rotation and a random translation unit vector. This method we will call the random initialization method.

Prior information can significantly increase performance. One way to incorporate prior information is to seed the optimizer at each LMedS iteration with the best hypothesis from the previous time step. We will denote this method as the "prior" initialization method.

Another approach is to seed the optimizer with the best hypothesis so far from the current time step. For the first iteration of LMedS, there is no best hypothesis and so the optimizer must be seeded randomly. However, for each subsequent iteration, the best hypothesis can be used to seed the optimizer. Since each hypothesis depends on the previous best hypothesis, we will denote this method as the "random recursive" initialization method.

These two approaches can also be combined. This results in one long continuous stream of hypotheses which depend on the previous best hypothesis. We will denote this method as the "prior recursive" initialization method.

H. Refining Relative Pose Hypotheses

Even the best hypothesis from RANSAC and LMedS usually has some error due to error on the individual points in the minimum subset used to create the hypothesis. The estimate can be improved by using least-squares optimization on the manifold over inlier points.

Since our algorithm already uses LM to optimize relative pose hypotheses, it can easily be extended to accept more than five points to refine the best relative pose hypothesis. In contrast, the OpenCV five-point algorithm does not have any built-in optimization method or separate function to perform the optimization.

Naive optimization using least-squares will often make the estimate worse if the data contains outliers. Robust least-squares optimization methods either explicitly find the inliers [13], or use a robust cost function that essentially gives higher weights to measurements with lower residuals [14].

For robust refinement we will find the inliers by first estimating the standard deviation $\hat{\sigma}$ [13]. For an explanation of these cryptic numbers the reader is referred to [15], page 202. The robust standard deviation is given by

$$\hat{\sigma} = 1.4826 \left[1 + \frac{5}{n-p}\right] \sqrt{m},$$

where $p = 5$ is the number of points used to create the model, n is the total number of points, and m is the median squared from LMedS. Inliers are defined to be any points within two and a half standard deviations. Thus the weights or inlier mask is

$$w_i = \begin{cases} 1 & r_i^2 < 2.5\hat{\sigma} \\ 0 & \text{otherwise.} \end{cases}$$

Even after attempting to remove outliers, however, the refined estimate can sometimes be worse than the original. To prevent this, we score the refined hypothesis using the same LMedS Sampson error metric to see if it is better or worse than the original. If the new hypothesis has a lower LMedS error it is kept, otherwise it is discarded.

I. Determining the Correct Rotation and Translation

For any relative pose hypothesis, there are two possible rotations R_1 and R_2, as well as two possible translations t and $-t$, that will produce the same Sampson cost. This is because the Sampson cost is based on the essential matrix. Each of these possible rotation and translation pairs produce equivalent essential matrices, and thus our hypothesis generation and scoring algorithms cannot distinguish between them.

Direct essential matrix solvers usually find these rotation matrices and the translation vector by extracting them from the essential matrix using the singular-value decomposition (SVD). However, since our optimization already produces one possible rotation matrix and translation vector, we can determine the other possible rotation by rotating the first rotation 180 degrees about the translation vector. This is simpler than taking the SVD of the essential matrix. Using the Rodrigues formula (3) to rotate the first rotation matrix R_1 by 180 degrees gives

$$R_2 = \left(I + 2\hat{t}_\times^2\right) R_1.$$

After finding the two rotation matrices and the translation vector, the cheirality check is often used to determine which of rotation matrices and which translation sign is correct. The cheirality check involves triangulating each point to determine its 3D position. In our implementation of the cheirality check we count the number of triangulated points with positive depth and then add the totals from both cameras. The rotation and translation pair with the largest sum is chosen. However, with four possible choices, the cheirality check often gives spurious results.

Alternatively, since the two possible rotations are always 180 degrees apart, we can pick the rotation with the smallest angle and use the cheirality check to find the correct translation. In a continuous video sequence the true rotation between frames is usually very small. Thus one possible rotation angle will be close to zero and the other possible rotation angle will be close to 180 degrees. As long as the true rotation is never more than 90 degrees, the correct rotation will always be the one with the smaller angle.

Taking the trace of the matrix is a computationally efficient method of picking the rotation with the smaller angle. Notice that the first and third terms of the Rodrigues formula (3) are the only terms with diagonal components. Thus the trace of the rotation matrix is

$$\begin{aligned} tr\left(R\right) &= 3 - \left(1 - cos\left(\theta\right)\right)\left(2\hat{\omega}_1^2 + 2\hat{\omega}_2^2 + 2\hat{\omega}_3^2\right) \\ &= 3 - 2\left(1 - cos\left(\theta\right)\right) \\ &= 1 + 2cos\left(\theta\right). \end{aligned} \tag{12}$$

Since $cos\left(\theta\right)$ is monotonically decreasing on $0 < \theta < \pi$, taking the trace of the matrix $\mathrm{tr}\left(R\left(\theta\right)\right)$ will also be monotonically decreasing. Thus the trace can be used to determine which rotation matrix has the smallest angle, giving

$$R = \begin{cases} R_1 & \text{if } \mathrm{tr}\left(R_1\right) > \mathrm{tr}\left(R_2\right) \\ R_2 & \text{otherwise.} \end{cases}$$

J. Complete System

The complete system is shown in Figure 2. At each time step, features are detected in one frame using good features to track [16] and then matched to the next frame using LK optical flow [17]. These point pairs are the inputs to ReSORtSAC.

At each iteration of LMedS, the LM optimizer is seeded with the best initial guess, using the prior recursive method described in section III-G. Ten LM iterations are used, with iterations being repeated if the error is not successfully decreased (Section III-E). After each iteration of LMedS, the new hypothesis is scored. If the median squared error is lower than the median squared error of the best hypothesis so far, the new hypothesis is kept (Section III-F).

The best hypothesis is then refined using inliers (Section III-H) and its rotation and translation disambiguated (Section III-I). The result is the final essential matrix estimate.

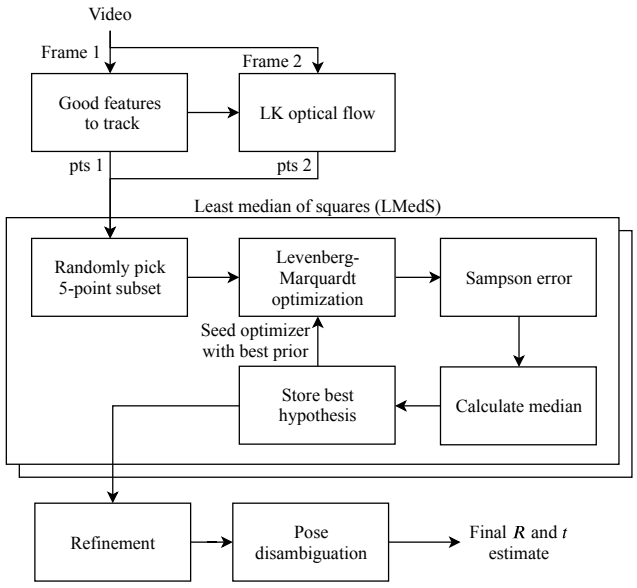

Fig. 2. Block diagram of ReSORtSAC

K. Computational Considerations

There are a couple of computational considerations in implementing the ReSORtSAC relative pose estimator. The most important of these considerations is to reduce the number of matrices allocated on the heap, as heap allocations are computationally expensive. One reason the OpenCV polynomial essential matrix solver is slow is because it allocates a large number of dynamic matrices on the heap.[1] Our implementation allocates fixed-size matrices on the stack and initializes dynamic matrices from raw buffers created once at the beginning of the program. These initialization options are available in both Eigen and OpenCV, but we have chosen to implement our algorithm using Eigen.

[1]Based on our own observation of the publicly available OpenCV source code [4]

IV. Motion Detection and Tracking in the Presence of Parallax

One application of relative pose estimation is motion detection and tracking in the presence of parallax. Motion detection is a valuable source of information in tracking. It can be used to track objects without any prior knowledge about their appearance, in contrast to many trackers that are designed to track specific classes of objects.

There are many successful image-based background subtraction techniques in the literature that work on stationary cameras. In order for image differencing techniques to work on a moving camera, the images must first be aligned using a homography. This works well for planar scenes. But if there is parallax, artifacts can appear in the difference image. If the parallax is small enough in comparison to the movement of objects in the scene, the effects of parallax can be reduced using simple morphological operations and gradient suppression, as is done in [18].

In the presence of strong parallax, however, a better motion model that accounts for depth variation must be used. There are several methods in the literature that use a geometric model to describe the motion of tracked points in the scene over time. A KLT tracker or other tracker can be used to generate sparse point trajectories. These can then be used in the motion model.

With these tracked points, one approach is to approximate the camera as orthographic instead of perspective [19]. With this approximation, the motion of the camera and the points in the scene over time can be solved using a system of linear equations. Points that are not consistent with this motion must be moving points. This approach works well if the camera is far from the scene or only observes a small field of view, but does not work well for perspective transformations, because the linear orthographic assumption is violated.

Another approach is to maintain an affinity matrix over time. Each cell of this matrix stores the dissimilarity between two points and is calculated from the maximum difference of position and velocity between these two points up until the current frame. This affinity matrix can be mapped into an embedding space. The points can then be clustered in the embedding space to distinguish between foreground and background points and to segment moving objects [20]. Another approach is to use matrices to describe multiple-frame geometric constraints [21], [22].

These approaches are well developed, but the papers describing these methods do not include computation times. In this paper, we use a simple method for motion detection using only the two-frame essential matrix constraint. The advantages of the method are computational simplicity and real-time performance.

A. Problem Setup

Given two consecutive frames, with point correspondences detected in each frame, the objective is to determine which points are from stationary objects and which are from moving objects. It is assumed that the essential matrix E, along with

its rotation R and translation t have already been calculated using the iterative method described in Section III.

This is a much simpler problem when the camera is stationary, since all motion observed in the image plane comes from moving objects in the world frame. However, for a moving camera, the problem becomes more difficult because object motion and camera motion both cause apparent motion in the image plane. The challenge is determining the source of this motion. In other words, the goal is to design a detector $\phi(p_i)$ which returns 1 if the ith point is moving and 0 otherwise. The output of the motion detector is used as an input to a tracking algorithm in order to produce target estimates.

B. Motion Detection Algorithm

The essential matrix relates points in one image to the other image with the epipolar constraint. In other words, the essential matrix maps a point in one image to a line in the other image. Where the point in the other image appears along this line depends on the real-world depth of the point from the camera. As the camera translates, points that are closer to the camera will appear to move more than the points that are far away. This effect is known as parallax.

There are two degrees of freedom for the apparent motion of each point in the image plane. One of these degrees of freedom can be explained by the epipolar constraint if the real-world point is stationary. However, motion along this degree of freedom can also be explained by object motion in the world frame. Hence the source of any movement along this degree of freedom is ambiguous without additional information. The second degree of freedom for apparent motion of points in the image plane is perpendicular to the epipolar constraint. Thus the only possible source of motion along this degree of freedom is movement in the real-world frame.

Note that except in degenerate cases, each point in the image will move in a different direction due to parallax. For example, when the camera is moving towards the scene or away from the scene, the points move along radial epipolar lines which intersect at the center of the image. As a result, it can be helpful to describe the motion of the points due to parallax using a vector field. Define the parallax field as a unit vector assignment to each point in the image plane that points in the direction the point would move due to parallax. The second degree of freedom is perpendicular to the parallax field, thus we will define the perpendicular field as a 90 degree clockwise rotation of each of these vectors.

These two vector fields give basis vectors at each point that will be used to decompose apparent motion in the image plane into the two degrees of freedom described above. We will denote the velocity in these two directions as the parallax velocity and the perpendicular velocity respectively.

Before considering the effects of parallax, however, we must compensate for the rotation of the camera. One method of compensating for the rotation of the camera is to use a Euclidean homography. Like the regular homography matrix, the Euclidean homography also maps points in one image to points in the other image using homogeneous coordinates, but operates on image plane coordinates instead of pixel coordinates. The Euclidean homography is given by

$$H_e = \left(R + \frac{tn^\top}{d} \right) \gamma,$$

where t is a translation vector, n is the normal vector of the plane which the homography describes, d is the distance to this plane, and γ is a scale factor.

The homography requires normalizing each homogeneous point after left-multiplying the point by the homography matrix. To condense notation, define $g(p)$ as an operator that normalizes homogeneous points, so that

$$g(p) \triangleq \frac{p}{p_z} = \begin{bmatrix} \frac{p_x}{p_z} \\ \frac{p_y}{p_z} \\ 1 \end{bmatrix}.$$

Compensating for the rotation alone gives

$$\hat{p}_i^2 = g\left(H_e p_i^1\right) = g\left(R p_i^1\right),$$

where $\hat{p}_2$ is the estimated location of the ith point in the next camera frame.

Note that this estimated location is where the point would be in the second image if it had infinite depth. The remaining point velocity is the sum of the true velocity of the point and the apparent velocity due to parallax. Subtracting the estimated point and true point in the second image gives the remaining velocity

$$v_i^2 = p_i^2 - \hat{p}_i^2.$$

We can then use the essential matrix to calculate the parallax and perpendicular fields. The essential matrix equation (1) can be rewritten as a line in standard form, giving

$$ax_i^2 + by_i^2 + \gamma = 0,$$

where $a = \left(Ep_i^1\right)_1$, $b = \left(Ep_i^1\right)_2$, and $\gamma = \left(Ep_i^1\right)_3$.

It can be shown that the vector $\begin{bmatrix} a & b \end{bmatrix}^\top$ is perpendicular to this epipolar line and the vector $\begin{bmatrix} -b & a \end{bmatrix}^\top$ is parallel to it. Assuming the essential matrix is formed from the relative pose as given in Equation (2), this parallel vector will have the correct sign and will be pointing in the direction of parallax. Making these vectors unit vectors gives the parallax field

$$f_\parallel \triangleq \begin{bmatrix} b \\ -a \end{bmatrix} / \sqrt{a^2 + b^2}$$

and the perpendicular field

$$f_\perp \triangleq \begin{bmatrix} a \\ b \end{bmatrix} / \sqrt{a^2 + b^2}.$$

The component of apparent velocity in the image plane in each direction can be found using the dot product with each field. There is no need to divide by the magnitude since the fields are already unit vectors. This gives

$$v_{\parallel} = \left\langle f_{\parallel}, \begin{bmatrix} v_x \\ v_y \end{bmatrix} \right\rangle$$

and

$$v_{\perp} = \left\langle f_{\perp}, \begin{bmatrix} v_x \\ v_y \end{bmatrix} \right\rangle.$$

The perpendicular velocity can then be thresholded to determine if a particular object is moving. The parallel velocity can also sometimes help distinguish between moving and stationary objects. Since parallax always makes stationary points appear to move in the same direction as camera translation, if the parallax velocity is negative it indicates that the object is moving. Thus the decision function for whether a point is moving can be written as

$$\phi(p_i) = \begin{cases} 1 & \text{if } |v_{\perp}| > \tau \text{ or } v_{\parallel} < -\tau \\ 0 & \text{otherwise,} \end{cases}$$

where τ is the moving point threshold. Due to small errors in calculating the essential matrix and lack of camera calibration, a threshold of one pixel is often the tightest constraint that can be used.

C. Recursive-RANSAC

Moving points found using motion detection are then fed into recursive-RANSAC, a newly proposed algorithm for multi-target tracking [23]. At each time step recursive-RANSAC searches for new models using RANSAC. When a sufficient number of inliers are detected, a new track is initialized.

Existing tracks are propagated forward using a Kalman filter. Probabilistic data association [24] is used to account for measurement association uncertainty. Each track is given an inlier score based on the percentage of time steps in which the track is detected. Recursive-RANSAC also has a track management system that merges similar tracks and removes tracks that have an inlier score lower than the minimum threshold.

V. Results

Two video sequences were used to test the two algorithms presented in this paper. The ReSORtSAC relative pose algorithm was tested on a synthetic video sequence of a UAV inside a city. The synthetic video sequence has no moving objects, so the complete motion detection and tracking algorithm was tested on a video sequence taken from a UAV camera.

A. Iterative Five-Point Algorithm

The ReSORtSAC relative pose algorithm was tested on a synthetic video sequence of a UAV inside a city generated using the BYU Holodeck simulator [25]. The two-minute video sequence (3600 frames) includes aggressive rotational and translational motions. A screenshot of the video sequence is shown in Figure 3.

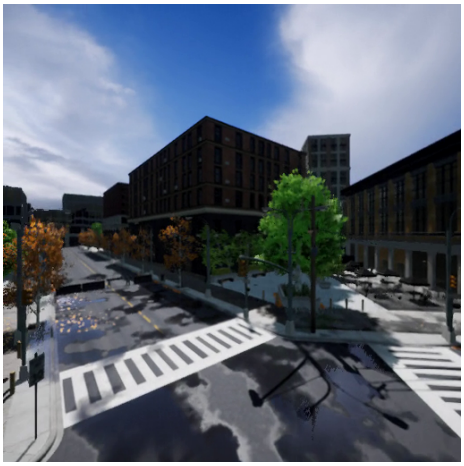

Fig. 3. Screenshot of the holodeck video sequence

There are three error metrics for each algorithm comparison. The rotational error is the smallest rotation angle between the true rotation and the estimated rotation. This angle is the norm of the matrix logarithm and can be efficiently calculated by solving (12) for θ, giving

$$\begin{aligned} e_R &= \left\| \log R_{true} R_{est}^{-1} \right\| \\ &= \frac{\operatorname{tr}\left(R_{true} R_{est}^{-1}\right) - 1}{2}. \end{aligned}$$

The translational error is computed by finding the angle between the true unit vector translation and the estimated unit vector translation, so that

$$e_t = \cos^{-1}\left(\hat{t}_{true}^{\top} \hat{t}_{est}\right).$$

Note that when calculating the rotation and translation error metrics, we do not penalize pose disambiguation errors. We consider the rotation and translation error metrics to be independent from the pose disambiguation error metrics, since there are four possible rotation-translation pairs for which the Sampson error produces the same cost value. The rotation and translation errors are thus computed for all possible rotations and translations and smallest of these errors is returned. The pose disambiguation error metrics show the percentage of the time that the relative pose algorithm chooses the correct rotation and translation.

Both the rotational and translational error are measured in radians. The LMedS Sampson error is also computed. Unless otherwise noted, all error metrics are averaged over the entire video sequence of 3599 frame pairs.

The error over time for the OpenCV LMedS polynomial solver and the ReSORtSAC solver is shown in Figure 4. Both the OpenCV and ReSORtSAC solvers give low error for the UAV trajectory. Notice how the rotation error seems to be proportional to the total rotation, while the translation error becomes very large as the true translation approaches zero.

Various methods of initializing the LM optimizer were tested and compared against the OpenCV five-point polynomial solver. To compare these methods, the LMedS algorithm was run for 100 iterations at each time step for

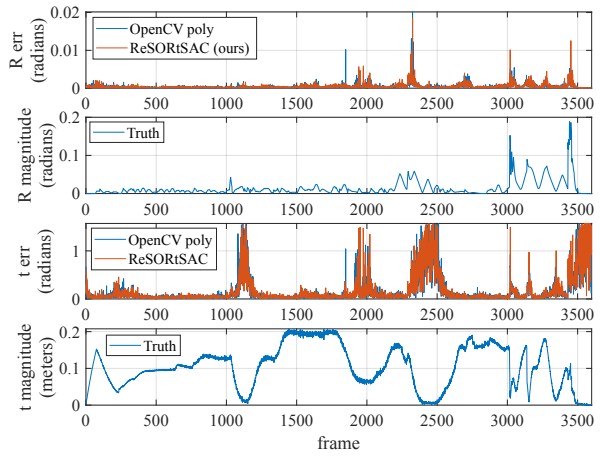

Fig. 4. Error over entire video sequence

the complete video sequence. At each LMedS iteration the error of the best hypothesis was recorded. The mean error across the entire video sequence is plotted in Figure 5.

This result shows the importance of initializing the optimizer with a prior. The random initialization method performs the worst out of all four methods, while initializing the optimizer with a prior from the previous time step or the best LMedS hypothesis so far from the current time step significantly reduces the error. After 100 iterations, the LMedS error for the initialization methods that use prior information is comparable to the OpenCV five-point polynomial solver, despite the fact that only one hypothesis is generated per subset instead of an average of about four.

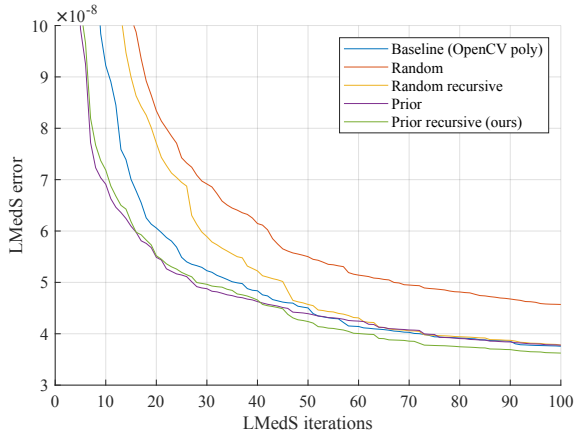

Fig. 5. ReSORtSAC Levenberg-Marquardt seeding methods

Similarly, Figure 6 shows the error of our algorithm and the OpenCV algorithm, but with the x-axis changed to be time instead of number of iterations. When under a time constraint, ReSORtSAC significantly outperforms the OpenCV solver.

Table I compares the error of the LMedS Gauss-Newton and Levenberg-Marquardt methods. Levenberg-Marquardt has a noticeably lower error. This is likely because Levenberg-Marquardt does a better job of dealing with large

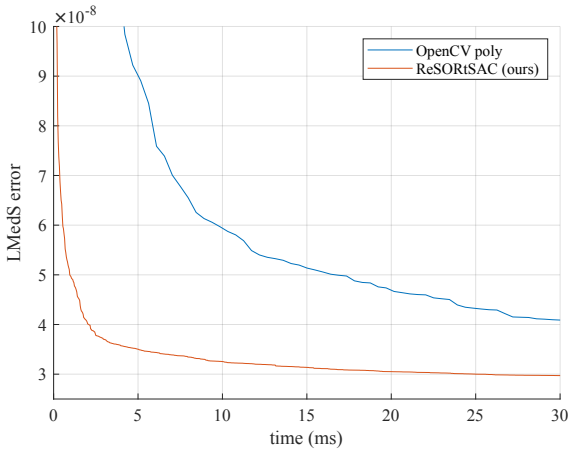

Fig. 6. ReSORtSAC and OpenCV error over time comparison

TABLE I
GAUSS-NEWTON AND LEVENBERG-MARQUARDT ERROR COMPARISON

Relative pose solver	Rotation error (radians)	Translation error (radians)	LMedS error (Sampson)
Baseline (OpenCV)	4.843e-04	1.742e-01	3.769459e-08
GN + LMedS	4.806e-04	1.731e-01	3.773465e-08
LM + LMedS	4.573e-04	1.706e-01	3.623720e-08

gradients and non-linearities.

RANSAC and LMedS were also compared. For RANSAC the algorithm was tested with 19 different thresholds. For LMedS, the algorithm was run once, since there is no threshold parameter to tune. For each run the average truth rotation and translation error over the entire video sequence were calculated. As shown in Figure 7, LMedS performs well without requiring a threshold. However, in order for RANSAC to perform as well as LMedS, the threshold must to be tuned to within an order of magnitude of the optimal threshold.

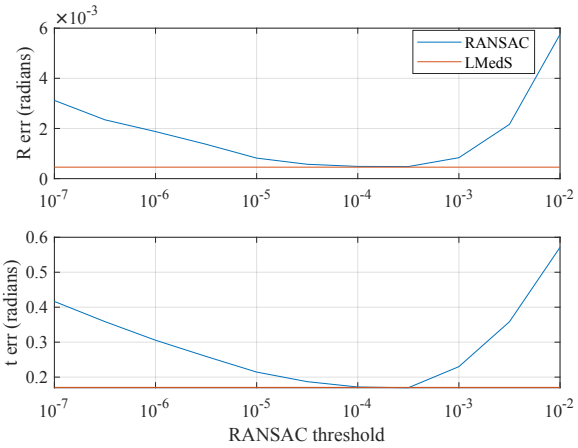

Fig. 7. Average rotation and translation errors for RANSAC and LMedS

Table II shows the results of the rotation and translation disambiguation algorithms. The first row within each group of relative pose solvers shows a baseline comparison, where no method was used for pose disambiguation. The baseline

TABLE II

POSE DISAMBIGUATION COMPARISON

Solver	Pose disambiguation method	Rotation correct	Translation correct	Both correct
OpenCV	none	50.2%	14.7%	6.8%
OpenCV	cheirality	54.0%	93.2%	52.3%
OpenCV	trace + cheirality	100.0%	96.5%	96.5%
ReSORtSAC	none	100.0%	53.1%	53.1%
ReSORtSAC	cheirality	40.9%	92.1%	40.8%
ReSORtSAC	trace + cheirality	100.0%	96.5%	96.5%

TABLE III

RESORTSAC RELATIVE POSE REFINEMENT

Relative pose solver	Refine success	Rot err (radians)	Trans err (radians)	LMedS err (Sampson)
Baseline (OpenCV)	-	4.843e-04	1.742e-01	3.769e-08
Before refinement	-	4.573e-04	1.706e-01	3.624e-08
After refinement	55.0%	3.779e-04	1.596e-01	3.519e-08

method gives poor results. However, it is worth nothing that ReSORtSAC is able to keep the correct rotation, even without any form of pose disambiguation. This is likely because it is seeded at the first frame with the identity rotation, and every frame thereafter the best hypothesis from the previous is reused as a seed to the optimizer.

The second row in each group shows the results when the cheirality check was used (Section III-I) to determine the best out of the four possible rotation translation pairs. Though the translation direction is often correct, the rotation is correct only about half of the time. The third row in each group shows the results of using the matrix trace to determine which rotation is correct, with the cheirality check to determine the correct translation direction. This third pose disambiguation method consistently outperforms the other methods.

Table III compares the average error before and after refinement using only inlier points (Section III-H). The refinement is defined to be successful if the new relative pose has a lower LMedS Sampson error. The new relative pose is only kept if the refinement was successful. Refining the best relative pose hypothesis significantly reduces all three error metrics.

The computation times for the relative pose solvers is shown in Table IV. Notice that the OpenCV polynomial solver takes much longer to generate hypotheses and also requires scoring four times as many hypotheses. The time required to score each hypothesis is proportional to the number of points detected in each frame, which for this video sequence is on average about 400 points. The relative pose solvers were benchmarked on a laptop with a 2.1 GHz Intel i7 CPU running Linux. The breakdown of the time required to generate each hypothesis set is shown in Figure 8. The most time-consuming part of the OpenCV solver is finding the zeros of the tenth-order polynomial. The most time-consuming part of the GN and LM solvers is the Eigen matrix solver.

TABLE IV

COMPUTATION TIME

	OpenCV poly	ReSORtSAC
Hypothesis generation	$100 * 0.400$ ms $= 40.0$ ms	$100 * 22.7$ us $= 2.27$ ms
Hypothesis scoring	$400 * 17.2$ ns $= 6.89$ ms	$100 * 9.22$ ns $= 0.92$ ms
Refinement	-	5.98 ns
Pose disambiguation	0.32 ms	0.15 ms
Total	47.2 ms	3.94 ms

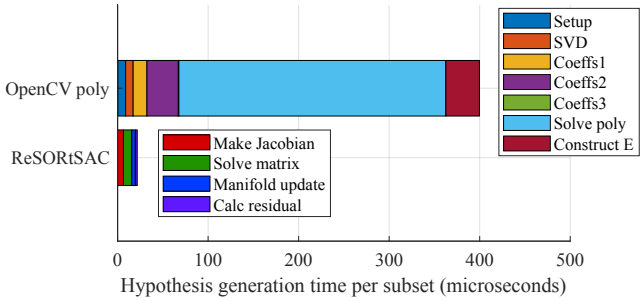

Fig. 8. Time required to generate each hypothesis set

B. Motion Detection and Tracking Results

The motion detection algorithm was tested on a moving camera video sequence taken from a UAV. Figure 9 shows the results of the motion detection algorithm. Notice how the stationary points have zero perpendicular velocity and a positive parallax velocity, while the moving points have a non-zero perpendicular velocity component.

The computation times of the motion detection and tracking algorithm are shown in Table V. For faster processing the video was scaled to 640×480. The motion detection and tracking algorithm is running on a Linux desktop computer with a 4GHz Intel i7 CPU. On average about 800 points are detected, tracked, and fed to the relative pose solver each frame. Notice that the OpenCV feature detection and tracking are the most time-consuming components of the tracking algorithm and consume 70% of the total CPU usage. The complete algorithm takes 29 milliseconds to run per frame, which means it is capable of running in real-time at 34 frames per second (FPS).

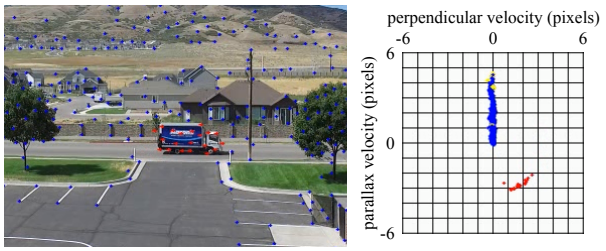

Fig. 9. Video motion detection results. Each point position (left) and its corresponding net velocity (right) are plotted. Points with a net perpendicular velocity greater than one pixel are classified as moving points (red), while points with a velocity below this threshold are classified as stationary points (blue).

Fig. 10. Recursive RANSAC tracks

TABLE V
MOTION DETECTION AND TRACKING COMPUTATION TIMES

Tracking Component	Computation Time
Good features to track	9.2 ms
LK optical flow	12 ms
Calc E (ReSORtSAC)	3.0 ms
Recursive RANSAC	0.4 ms
Other	4.4 ms
Total	29 ms (34 FPS)

VI. CONCLUSION

In this work, we have presented an iterative five-point algorithm for solving the rotation and translation between consecutive frames capable of running in real-time. We show the importance of seeding the Levenberg-Marquardt optimizer with an initial guess and demonstrate that this initial guess significantly improves the performance to the algorithm. We have applied this algorithm to detecting motion and tracking multiple targets from a UAV and demonstrated real-time performance of this tracking algorithm on a 640×480 video sequence.

Future work includes using an IMU to improve the initial guess of the rotation and translation from the previous time step. Future work also includes applying the principles of using an initial guess to seed the optimizer to 3D scene reconstruction and more complex tracking methods. The depth of moving objects can be estimated by using nearby 3D reconstructed points, building on the method used in [26], in order to estimate the 3D position of moving targets.

REFERENCES

[1] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[2] P. J. Rousseeuw, "Least median of squares regression," *Journal of the American statistical association*, vol. 79, no. 388, pp. 871–880, 1984.

[3] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[4] "OpenCV 3.1: Open source computer vision library," https://github.com/opencv/opencv/releases/tag/3.1.0, 2015.

[5] Y. Ma, J. Košecká, and S. Sastry, "Optimization criteria and geometric algorithms for motion and structure estimation," *International Journal of Computer Vision*, vol. 44, no. 3, pp. 219–249, 2001.

[6] T. Botterill, S. Mills, and R. Green, "Refining essential matrix estimates from RANSAC," in *Proceedings Image and Vision Computing New Zealand*, 2011, pp. 1–6.

[7] ——, "Fast RANSAC hypothesis generation for essential matrix estimation," in *Digital Image Computing Techniques and Applications (DICTA), 2011 International Conference on*. IEEE, 2011, pp. 561–566.

[8] U. Helmke, K. Hüper, P. Y. Lee, and J. Moore, "Essential matrix estimation using gauss-newton iterations on a manifold," *International Journal of Computer Vision*, vol. 74, no. 2, pp. 117–136, 2007.

[9] M. Sarkis, K. Diepold, and K. Hüper, "A fast and robust solution to the five-point relative pose problem using gauss-newton optimization on a manifold," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 1. IEEE, 2007, pp. I–681.

[10] C. Hertzberg, R. Wagner, U. Frese, and L. Schröder, "Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds," *Information Fusion*, vol. 14, no. 1, pp. 57–77, 2013.

[11] S. Belongie, "Cse 252b: Computer vision ii, lecture 11," https://cseweb.ucsd.edu/classes/sp04/cse252b/notes/lec11/lec11.pdf, 2006.

[12] F. Fraundorfer and D. Scaramuzza, "Visual odometry: Part II: Matching, robustness, optimization, and applications," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 78–90, 2012.

[13] Z. Zhang, R. Deriche, O. Faugeras, and Q.-T. Luong, "A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry," *Artificial Intelligence*, vol. 78, no. 1-2, pp. 87–119, 1995.

[14] B. Triggs, P. F. McLauchlan, R. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis," in *International Workshop on Vision Algorithms*. Springer, 1999, pp. 298–372.

[15] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*. John Wiley & Sons, 2005, vol. 589.

[16] J. Shi *et al.*, "Good features to track," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*. IEEE, 1994, pp. 593–600.

[17] S. Baker and I. Matthews, "Lucas-Kanade 20 years on: A unifying framework," *International Journal of Computer Vision*, vol. 56, no. 3, pp. 221–255, 2004.

[18] V. Santhaseelan and V. K. Asari, "Moving object detection and tracking in wide area motion imagery," in *Wide Area Surveillance*. Springer, 2014, pp. 49–70.

[19] Y. Sheikh, O. Javed, and T. Kanade, "Background subtraction for freely moving cameras," in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 1219–1225.

[20] A. Elqursh and A. Elgammal, "Online moving camera background subtraction," in *European Conference on Computer Vision*. Springer, 2012, pp. 228–241.

[21] J. Kang, I. Cohen, G. Medioni, and C. Yuan, "Detection and tracking of moving objects from a moving platform in presence of strong parallax," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 1. IEEE, 2005, pp. 10–17.

[22] S. Dey, V. Reilly, I. Saleemi, and M. Shah, "Detection of independently moving objects in non-planar scenes via multi-frame monocular epipolar constraint," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7576 LNCS, no. PART 5, pp. 860–873, 2012.

[23] P. C. Niedfeldt and R. W. Beard, "Multiple target tracking using recursive RANSAC," in *2014 American Control Conference*, June 2014, pp. 3393–3398.

[24] Y. Bar-Shalom, F. Daum, and J. Huang, "The probabilistic data association filter," *IEEE Control Systems*, vol. 29, no. 6, 2009.

[25] "BYU holodeck: A high-fidelity simulator for deep reinforcement learning," https://github.com/byu-pccl/holodeck, 2018.

[26] J. Nielsen and R. W. Beard, "Relative target estimation using a cascade of extended Kalman filters," *Proceedings of the 30th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2017)*, pp. 2273–2289, 2017.