Robust Object Tracking Using Kalman Filters with Dynamic Covariance

Sheldon Xu and Anthony Chang
Cornell University

Abstract— **This project uses multiple independent object tracking algorithms as inputs to a single Kalman filter. A function for estimating each algorithm's error from related features is trained using linear regression. This error is used as the algorithm's measurement variance. With a dynamic measurement error covariance computed from these estimates, we attempt to produce an overall object tracking filter that combines each algorithm's best-case behavior while diminishing worst-case behavior. This filter is intended to be robust without being programmed with any environment-specific rules.**

I. Introduction

One very important perception task in robotics is tracking objects using a camera. The goal of object tracking is to find an object's location in consecutive video frames. Object tracking's many obvious applications include perception and control for autonomous surveillance systems, identifying and neutralizing threats in missile defense, optimizing traffic control systems, and improving human-computer interaction.

Many different algorithms have been proposed for object tracking, including mean-shift tracking, optical flow, and feature matching. Each algorithm has strengths in certain environments and weaknesses in others. This project aims to combine several such algorithms as inputs or "measurements" to a single Kalman filter for robust object tracking. The filter can favor the algorithm that is most applicable to the current environment by decreasing its measurement noise variance, and similarly ignore less suitable algorithms by increasing their measurement variances.

This project focuses on training a robust object tracking Kalman filter, then applying this filter to tracking varing objects in arbitrary environments using a Parrot Quadrotor robot. Training is done through regression of a function to calculate the filter measurements' error covariance. Training and validation data contains a mix of video captures and automatically generated data. No rules specific to any environment are used during algorithm development or training. The trained filter's success is measured by its performance in unfavorable environments, as well as its general ability to follow objects when deployed on the quadrotor.

This report will first summarize the input object tracking algorithms in Section II. Section III will cover how they are combined into a Kalman filter, as well as how the Kalman filter's output is used to control the quadrotor. Section IV will discuss the filter's preliminary training and test results, and Section V will conclude the report and results.

II. Object Tracking Algorithms

This section discussion the object tracking algorithms that we propose to use as measurements in our Kalman filter. These algorithms have varying strengths and weaknesses; we intend to train our Kalman filter to dynamically identify when each algorithm is weak, and penalize it accordingly with a higher measurement variance. A summary of each algorithms' weaknesses is presented in Table 1; these weaknesses are used to propose features for training in the next section.

A. *CamShift*

The Continuously Adaptive Mean Shift Algorithm (CamShift) is a lightweight object tracking algorithm based on a one-dimensional hue histogram. Originally designed for tracking faces or flesh tone, the algorithm computes the probability that any pixel is part of the tracked object as opposed to the background.

Given a color probability distribution, CamShift iteratively applies the Mean Shift algorithm to find the centroid of the probability image. The centroid is used as the center of a window for recalculating another color probability distribution, which will be used in the next frame. This process repeats fairly quickly every frame. CamShift thus continuously adapts its color histogram to each frame, hence its name. [1]

CamShift is fairly good at tracking a single foreground object even among multiple moving objects, as long as the object is not too multicolored. However, CamShift's results are not as good if the foreground object is colored too similarly to the background, if lighting conditions change, or if the foreground object's color changes rapidly.

B. *SURF*

SURF (Speeded Up Robust Features) is a feature detector and descriptor. It is based on sums of approximated 2D Haar wavelet responses and makes an efficient use of integral images.

"Interest points" are selected at distinctive locations in the image, such as corners, blobs, and T-junctions. The neighbourhood of every interest point is represented by a feature vector. The use of feature vectors allows SURF to be scale and rotation invariant. SIFT matches these feature vectors between different images (target versus scene). The matching can be based on a distance between the vectors. Fast approximate nearest neighbour search was used in this implementation.

TABLE I

OBJECT TRACKING ALGORITHM SUMMARY

Algorithm	dim	bright	lighting changes	moving object	stationary object	dropped frames	low resolution	similar background shapes	similar background colours
CamShift	bad	good	ok	good	good	ok	good	good	ok
SURF	good	good	good	N/A	good	good	bad	ok	good
Optical Flow	ok	ok	ok	good	bad	ok	bad	good	good

Given images of a target and a scene, SURF is very good at finding the location of the image even under suboptimal lighting or if there are similarly coloured objects in the scene. One crucial drawback is the computation time required to create interest points and match feature vectors, thereby rendering it incapable of producing real-time results for object tracking.

C. *Optical Flow*

Optical Flow tracking techniques allow the distinction between multiple objects and the background in a scene. This is computed based on the relative motion between an observer and the scene.

Optical Flow can be implemented in many ways, but commonly assume that neighbouring points in the scene will have similar motions given consecutive frames. Under certain circumstances, Optical Flow manages to track a moving object very well. For example, under normal lighting, it can locate a moving object.

III. KALMAN FILTERS

The Kalman filter is a framework for predicting a process's state, and using measurements to correct or 'update' these predictions.

A. *Time Update*

Discrete-time Kalman filters begin each iteration by predicting the process's state using a linear dynamics model.

1) State Prediction: For each time step k, a Kalman filter first makes a prediction $\hat{x}_k^-$ of the state at this time step:

$$\hat{x}_k^- = Ax_{k-1} + Bu_k$$

where $\hat{x}_{k-1}$ is a vector representing process state at time k-1 and A is a process transition matrix. u_k is a control vector at time k, which accounts for the action that the robot takes in response to state x_k; B converts the control vector u_k into state space. [2]

In our model of moving objects on 2D camera images, state is a 4-dimensional vector $[x, y, dx, dy]$, where x and y represent the coordinates of the object's center, and dx and dy represent its velocity. The transition matrix is thus simply

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We chose to include x and y velocities in our model because they are useful features. The objects that we intend to track on camera, such as people and cars, have slowly changing velocities; these velocities are easy to observe from a video stream. On the other hand, more complex features like acceleration are less useful, partly because they change suddenly and are harder to observe.

Our robot is able to orient itself using yaw rotation to track its target object. Its control is thus able to affect the object's x position: u_k is just a scalar representing how much the object is expected to move along the x axis in response to control. Converting u_k into state space is very simple:

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

because the yaw control u_k is only expected to shift the state's x-coordinate if the robot is approximately level.

2) Error Covariance Prediction: The Kalman filter concludes the time update steps by projecting estimate error covariance P_k^- forward one time step:

$$P_k^- = AP_{k-1}A^T + Q,$$

where P_{k-1} is a matrix representing error covariance in the state prediction at time $k-$, and Q is the process noise covariance (or the uncertainty in our model of the process). [2]

Intuitively, the lower the prediction error covariance P_k^-, the more we trust the prediction of the state $\hat{x}_k^-$. Prediction error covariance will be low if the process is precisely modeled, so the entries of Q are fairly low. Unfortunately, determining Q for any process model is often difficult – Q depends on hard-to-predict variables such as how often the target object changes velocity. We use the *Autocovariance Least-Squares* technique, whose implementation is freely available in MATLAB/Octave, to learn Q for our process model from training data. [3]

B. *Measurement Update*

After predicting the state $\hat{x}_k^-$ (and its error covariance) at time k using the time update steps, the Kalman filter next uses measurements to "correct" its prediction during the measurement update steps.

1) Kalman Gain: First, the Kalman filter computes a *Kalman gain* K_k, which is later used to correct the state estimate $\hat{x}_k^-$:

$$K_k = P_k^- H^T (HP_k^- H^T + R_k)^{-1}$$

where H is a matrix converting state space into measurement space (discussed below), and R is measurement noise covariance. [2] Like Q, determining R_k for a set of measurements is often difficult; many Kalman filter implementations statically analyze training data to determine a fixed R for all future time updates. We instead allow R to be dynamically calculated from the measurement algorithms' state. This procedure is detailed at the end of this section.

2) State Update: Using Kalman gain K_k and measurements z_k from time step k, we can update the state estimate:

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x_k}^-).$$

Conventionally, the measurements z_k are often derived from sensors. [2] In our approach, measurements z_k are instead the output of various tracking algorithms given the same input: one frame of a streaming video, and the most likely x and y coordinates of the target object in this frame (taken the first two dimensions of $\hat{x_k}^-$).

z_k thus contains two dimensions for every tracking algorithm, and has the form $[x_0, y_0, x_1, y_1, ..., x_{n-1}, y_{n-1}]$ for n different tracking algorithms. As a result, H has the form

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ ... & ... & ... & ... \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

as each measurement's x and y coordinates ideally equal the state's x and y coordinates, without any dependence on velocity.

3) Error Covariance Update: The final step of the Kalman filter's iteration is to update the error covariance P_k^- into P_k:

$$P_k = (I - K_k H)P_k^-. \text{ [2]}$$

The updated error covariance will be significantly decreased if the measurements are accurate (some entries in R_k are low), or only slightly decreased if the measurements are noise (all of R_k is high).

C. Control

1) Yaw: Our approach to control is fairly straightforward: the x coordinate of each a posteriori state prediction is used as an input to a *proportional controller*. The controller simply sets the robot's yaw based on $x - x_{center}$, where x_{center} is the x-coordinate of the image's center. Thus, the robot attempts to keep the object in the center of its vision.

2) Pitch: We originally intended to control pitch motion in addition to yaw motion, so the robot could follow objects in addition to tracking them. However, while this worked fairly well on certain classes of objects such as faces and hands, this proved difficult to generalize to a robust object tracker. Two possible approaches are summarized below.

a. Proportional size: Some tracking algorithms like CamShift output the target object's estimated size in addition to its estimated location. One approach to following is to use a P controller for pitch on these outputs, in an attempt to maintain a fixed object size. If the object appears to be shrinking fast or occupies too little of each frame, move closer to it. If the object appears to be growing fast or occupies too much of each from, move away.

Depending on the P controller's tuning, this approach works well on certain sizes of objects and poorly on others. For example, it can follow rolling chairs well at the expense of following the much smaller hands poorly.

b. Distance dimension: By adding object distance as another dimension of the Kalman filter's state, we may be able to get a better estimate of object size. This way, the rules for computing the object's distance (or changes in distance) from measurements can be learned through regression, like the rest of R. From here, using a P controller to maintain a reasonable distance is easy.

Unfortunately, computing distance using a monocular camera is not trivial. One major problem is inputting a starting distance to $\hat{x}_0$; this is difficult to do robustly.

3) Altitude: We considered allowing the robot to follow its target object vertically by adjusting its altitude. Much like yaw, altitude could be maintained by a simple P controller.

While this was fairly easy to implement, we decided against including vertical following in our final build, for fear of crashing the robot into the floor/ceiling, or flying it too high to safely land. Object tracking still works just as well without vertical following, as most objects don't move too much along the y-axis.

D. Tracking Algorithms as Kalman Filter Measurements

Kalman filters are easily able to take tracking algorithm outputs as measurements. However, the difficulty in combining arbitrary tracking algorithms as measurements comes from computing the Kalman gain: R_k, the measurement covariance matrix, is difficult to determine.

Our approach to this problem computes an error or noise estimate for each tracking algorithm. This computation is trained based on regression of image *features* that represent each algorithm's weaknesses. The features we propose are detailed below, and the actual training from these features is detailed in the next section. All features are computed from up to 2 consecutive video frames.

1) CamShift: Camshift tracking tends to fail when the target object is colored too similarly to the background, or when the target object's color histogram rapidly changes (for example, when a target person walks into a shadow).

These two failure conditions are possibly captured by two features: the 2-norm in the target's histogram change since the last frame, and the 2-norm of the difference between the target's histogram and the image background's histogram. Both features are easily calculated using the OpenCV functions calcHist, calcBackProject, and compareHist.

Another of CamShift's weaknesses is in tracking objects behind temporary obsctructions. We would like to introduce a feature to express this weakness, but it is somewhat infeasible to extract such a feature from only two frames.

2) SURF: Commonly, SURF's performance is mostly affected by the number of descriptors, which decreases with very poor lighting, low resolution, or a very plain object. A low number of descriptors in either target or scene images will degrade the performance. For example, a plain blue t-shirt in poor lighting and low resolution (34 x 53 pixels) gives only 4 descriptors. Even with 658 descriptors extracted from the scene (320 x 240 pixels), only 2 matches are made. Thus, the variance of SURF may be estimated from the feature:

$$1 - (matches/min(dO, dB)) * max(T - min(dO, dB), C)$$

where dO is the number of descriptors on the object, and dB is the number of descriptors on the background/scene. Our values of constants include thresholds T = 50 (minimum number of descriptors, related to the quality of the image), and C = 20 (maximum variance if the image meets quality requirements).

3) Optical Flow: Optical flow assumes brightness consistency, which is difficult to guarantee for the application of object tracking. It is also prone to image noise under very bright or very dark lighting, and will not return any useful data if the object is not moving. Optical Flow's variance may be estimated from the following features: (change in image luminosity)^2, (average image luminosity - 128)^2, dx, and dy.

Furthermore, Optical Flow will attempt locate the object in the scene that moves with the greatest velocity. Thus, if there are multiple objects moving at different velocities and the target object is not moving the fastest, it will be difficult to locate the target object. Unfortunately, this feature is difficult to capture without identifying multiple objects.

IV. Data and Results

A. Training

1) Data: Labeled training images allow us to perform *linear regression*, correlating the features of image pairs (described in section III) with tracking algorithm error.

Our *training data* consists of hundreds of image pairs (from sequential video frames), where the target object's location is known and labeled in both frames. *Validation data* similarly contains about a hundred image pairs. These images are meant to cover a wide range of objects and environments, as the training is intended to produce a robust Kalman filter for any object or environment without 'knowing' any special information about either one.

On a pair of training images (img_i, img_{i+1}), we first initialize all tracking algorithms using the target object's known location in img_i. Then we compute each algorithm's predictions of the object's location in frame img_{i+1}, and record each algorithm's error in Euclidean distance. The error-predicting features for each algorithm detailed at the end of section III are also recorded.

After error has been collected for each training pair, we use linear regression on the error-predicting features to find the least-squares fit to tracking error. This linear regression produces a rule for computing measurement error covariance R_k from images img_k and img_{k-1}. For simplicity, we assume that tracking algorithm errors are independent, so we only need to perform regression on each algorithm's own variance rather than an entire covariance matrix.

Since manually labelling training image pairs is expensive and prone to errors, and regression on so many dimensions requires a large set of data, we have supplemented our training data with some automatically generated training pairs. Fig. 1 shows an example training pair, containing a target object (red circle) on a black background with colored noise. A label (represented as a blue ellipse) is generated for both images. The label on the first image is used to initialize the tracking algorithms. The label on the second image is compared to the tracking algorithms' output (example shown in white). Our training set includes about eight thousand such automatically generated pairs, and our validation set includes about two thousand.

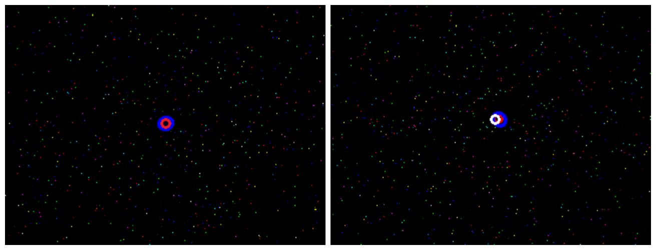
Fig. 1. Automatically generated and labeled training data. The previous frame is on the left.

2) Training Metrics: Training produces a linear function for computing a measurement error covariance matrix; we apply this rule in a Kalman filter over the training and validation image streams to calculate the resulting Kalman filter's tracking accuracy. Training and validation accuracy is calculated as the rate at which the Kalman filter's predicted location is within 48 pixels (Euclidean distance) of the real location (provided by label).

Accuracy on both training and validation data has been fairly consistent at 97.45%.

B. Testing

Testing had two components: Kalman filter robustness in certain environments, and robot deployment. All testing is compared to the CamShift algorithm alone as a baseline.

1) Kalman Filter Robustness: To test the Kalman filter's robustness, we ran the Kalman filter on video streams from three hand-picked 'difficult' environments. Our error metric for this test is more high-level and labor-intensive than the one used in training: each video is a trial, and each trial is successful only if a human judge decides that the Kalman filter has behaved correctly. The data sets are listed below, and a summary of these test results is presented in Table 2.

A. Lighting Changes: These videos start by tracking an object in one lighting environment. The object then moves into a differently-lit environment, and then back to the original environment. For example, the object may be under bright ambient lighting in the first frame, and then move against a harsh backlight before finally returning to its starting point. Success is achieved when the Kalman filter is obviously still tracking the object in the last frame.

Our Kalman filter performed poorly on these test sets, perhaps because of its heavy reliance on CamShift. It only

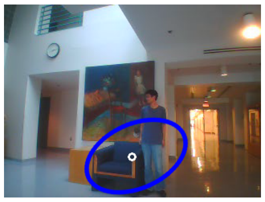

Fig. 2. Camshift on a distracting background. It was initialized to track the blue shirt, but eventually tracked the blue chair instead. The Kalman filter dynamically penalizes CamShift with a higher variance.

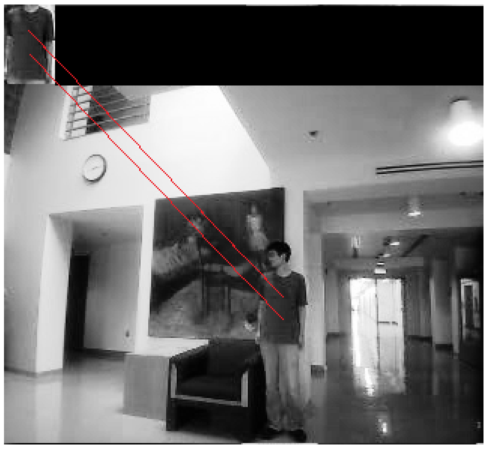

Fig. 3. SURF on the same distracting background as Fig. 2. SURF successfully follows two points on the shirt from one frame (top) to the next (bottom); the red lines link these points across the frames.

successfully tracked the object 1 in 5 times. A baseline CamShift tracker similarly succeeded 1 in 5 times.

B. Distracting Background: This set has videos in which the target object is very similarly colored to the background. For example, the target may be a dark blue shirt, while the background contains a similarly dark blue armchair. The filter is considered successful if it still tracks the object as the object moves through the background and back.

The trained Kalman filter was fairly good on these test sets, succeeding 11 of 13 times. The baseline CamShift alone had worse accuracy at 8 successes out of 13. SURF is particularly good at this test set, seeing that it is able to distinguish features from similarly colored objects (see Figs. 2 and 3).

C. Obscured Object: These videos involved the target object moving behind an obsctruction and emerging from the other side, then returning to its starting location. Trials are successful if the filter clearly tracks the object as it moves through the obstruction, then back to its starting point.

Our Kalman filter had a success rate of 8 in 14 on this data set. CamShift performed better at this task, scoring 10 of 14.

C. Robot Deployment

The robot deployment tests were fairly simple: each trial consisted of deploying the quadrotor running the Kalman filter and controller, and letting it physically track an object.

TABLE II
KALMAN FILTER ROBUSTNESS TEST RESULTS

Algorithm	Lighting	Background	Obscured
Kalman Filter	20%	84.62%	57.14%
Baseline (CamShift)	20%	61.54%	71.43%

Trials were successful if the robot clearly tracked the target object for at least 10 seconds.

Our tests consisted of letting the robot track shirts, jeans, hands, and faces. All trials were performed in basic, dim environments with distracting backgrounds. Of 15 recorded trials, 12 were successful; following usually lasted 30 or more seconds. The baseline CamShift alone had the same success rate of 12 in 15, though its tracking tended to last longer during successful trials.

Oddly, several different quadrotors used for these tests drifted even when control was disabled; this may have negatively affected the tests' accuracy. We hope to perform some trials on a more stable platform in the future.

V. Conclusions

We presented an approach for training a function for dynamically adjusting Kalman filter measurement error covariance, in attempt to tune a Kalman filter to favor better-suited tracking algorithms, and penalize ill-suited ones, during runtime. We implemented this approach using training and validation data drawn from real videos and automatically generated images, then tested its robustness in various environments.

Our experiment had mixed results. Though the Kalman Filter increased tracking accuracy in one type of environment, it was matched or outperformed by the baseline CamShift algorithm in most cases. Thus, we were only partly successful in increasing robustness over CamShift.

Future attempts to this approach may be able to improve upon our results by choosing more meaningful features for the regression (such as features that span more than just two frames), or by collecting a better set of training and validation data. Additionally, linear regression may not have been the best choice; perhaps a simpler logistic regression outputting a binary 'strong' or 'weak' rating may produce better results.

VI. Acknowledgements

We'd like to thank Cooper Bills for pointing us in the right direction and providing us with resources for this project.

References

[1] J. G. Allen et al., Object Tracking Using CamShift Algorithm and Multiple Quantized Feature Spaces, *Conferences in Research and Practice in Information Technology*, vol. 36, 2004, pp 1-4.
[2] G. Welch and G. Bishop. An Introduction to the Kalman Filter, *Proceedings of SIGGRAPH 2001*, pp 19-24.
[3] M. R. Rajamani. *Data-based Techniques to Improve State Estimation in Model Predictive Control*. University of Wisconson Press, 2007, pp ii-iii.