Reconstruction of 3D flight trajectories from tracking via on-line AdaBoost algorithm

CSE327[Fundamentals of Computer Vision] Final Project Report

Yongho Son

June 14, 2021

Abstract

In this final project, we used the concepts of on-line AdaBoost feature selection algorithm for tracking, Kalman filter for trajectory optimization, and ad-hoc camera networks that provide the 3D trajectory of unmanned aerial vehicles (UAVs). The distinct advantage of our tracking method is its capability of on-line training. This allows the classification to adapt while tracking the object. Then we applied the Kalman filter on each of the 2D trajectory outputs from each of the videos, which performs a reduction of noise introduced by inaccurate detections. Feeding the 2D trajectory information into the ad-hoc camera network and reconstruct trajectories in 3D by overcoming unsynchronization of the sequential data.

1 Introduction

The efficient tracking of an object in complex environments is important for a variety of applications including autonomous driving, drone trajectory tracking, or video surveillance. Thus, we have to come up with robust visual tracking methods which can manage inevitable variations that occur in natural scenes. Especially, the multi-view drone tracking dataset 4 contains fast motion and moving clouds, which makes the detection harder. Thus, the success of detection of the drone was highly relied on how distinguishable an object is from its background. As a result, we used on-line selection of local Haar-like features to handle possible variations in appearance.

AdaBoost [1] approaches the tracking problem as a classification problem between object and background, which is also known as a binary classification problem. It selects discriminative features by combining an ensemble of weak classifiers into a strong classifier. This algorithm performs on-line updating of the ensemble of features of the target object during tracking drones from the dataset, which is capable of managing appearance changes of the target object. Moreover, AdaBoost uses the background as negative examples in the update, so it is more robust against the changes in the background. Therefore, the method can deal with both appearance variations of the drone and different backgrounds due to fast clouds in the dataset.

2 Tracking

The main idea of tracking is to formulate the problem as a binary classification task and continuously update the classifier of the target to gain robustness. We assume that the image region is a positive image sample for the tracker while negative examples are extracted from the surrounding background. This initialization allows on-line boosting algorithm to have a first model that is already stable. Then we analyze the confidence value from the current tracker and shift the target window to the new location of the global maximum in the map. The current region is also used to have a positive update and the surrounding background is used to make a negative update on the tracker. As new frames arrive, the whole procedure is repeated.

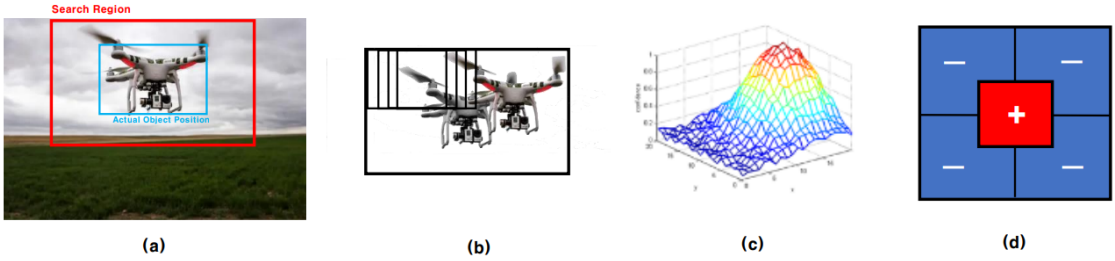


Figure 1: The four main steps of the tracker. (a) The initial position of the target object in time t is given, (b) the tracker evaluates many possible positions in a search region in frame t+1. (c) The confidence map is used to analyze the most probable position and (d) the tracker is updated

2.1 On-line AdaBoost

In this section, we briefly introduce the details of the on-line boosting algorithm with the following definition terms:

- **Weak classifier:** A classifier that can perform slightly better than random guessing. In case of binary classification, the error rate must be less than 50 percent.
- **Selector:** Given a set of N weak classifiers, a selector selects exactly one of those. The choice decision is made by the estimated error of each weak classifier.
- **Strong classifier:** Given a set of M weak classifiers, a strong classifier is computed by a linear combination of selectors.

The main idea of the algorithm is done by *selectors*. Each of them holds a separate feature of weak classifiers and the weak classifier with the lowest error is selected by the selector. When a new frame of the video arrives, the weak classifiers of each selector are updated. The selectors consecutively change the best weak classifier with respect to the weight passed on to the next selector (see Figure 2). Finally, the strong classifier is available at each time step of the video for object tracking.

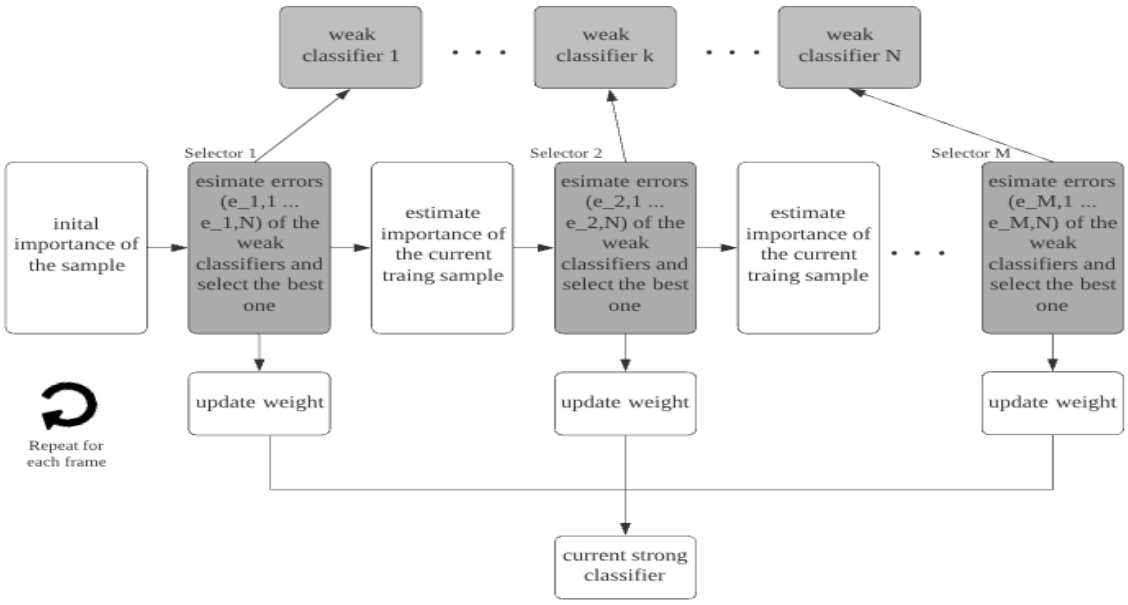


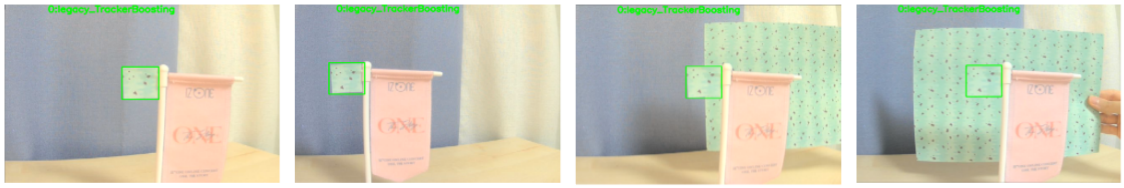
Figure 2: Principle of on-line boosting for feature selection

3 Experiments

This section demonstrates two properties of the tracker. Our experiments have been initialized by manually setting the target object in the first frame. Tracking has been applied to sequences with hundreds of frames.

3.1 Adaptivity

The adaptivity of the tracker is defined by selecting the best features depending on the background. Thus, we made a scene where the target object is a small textured patch (see Figure3). The objective is to demonstrate how the on-line feature selection method can adapt proficiently to the current tracking problem. We changed the background to the same texture, which is the same as the patch. The result shows that the target object was still successfully tracked by the algorithm although the background was the same texture as the object.



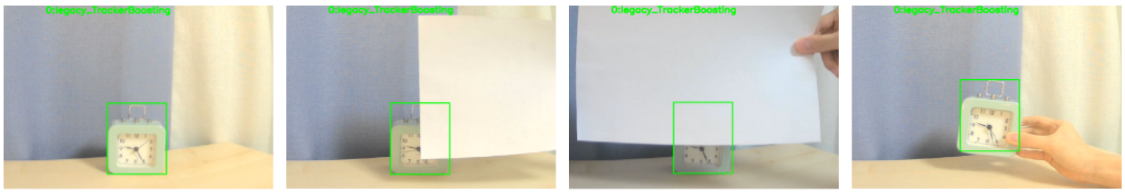
(a) Frame 9 (a) Frame 43 (a) Frame 78 (a) Frame 112

Figure 3: Each frames captured by a static camera with a resolution of 1920×1080. The tracked object is marked with a green rectangle.

This experiment illustrates the powerful role of on-line learning because it is not impossible to train for all different possible backgrounds available in the video.

3.2 Robustness

A successful tracker should manage various appearance changes of the target object, including occlusions, rotations, and movement. Figure 4 demonstrates the behavior of the AdaBoost algorithm in various conditions. The sequence shows a clock keeps tracked by the tracker although a significant portion of the object has been occluded. It implies that the tracker is continuously adapting to the background. Also, the tracker keeps track of the object with the movement in an upward direction and rotation by the hand. It shows that the tracker has been updated the object's feature and confidence score in real-time.



(a) Frame 7 (a) Frame 38 (a) Frame 73 (a) Frame 92

Figure 4: Tracking results on combination of appearance changes on target object (i.e. occlusion, movement, rotation, illumination)

This experiment shows that the tracker is capable of all kinds of appearance variations of the target object and always finds the best features for discriminating the target object from the background.

4 Kalman Filter

Kalman filter [3] is an algorithm that estimates the states of a system given the observations. Thus, it is a useful tool for a variety of different applications, especially in object tracking. During the process of object tracking, we used the Adaboost algorithm to produce a 2D trajectory path of the drone. It is a powerful object tracker, but the dataset contains some videos that include the fast movement of the drone and the disappearance from the scene. These properties produce some cut-offs in the trajectory. Thus, we applied the Kalman filter on the trajectory output from AdaBoost to produce a smoother trajectory for better trajectory estimation. The basic idea of the Kalman filter is using the prior knowledge of the state, which is the output from Adaboost in our project.

The filter makes a forward projection state and predicts the next state. Estimating the state of a system at time k using the linear stochastic difference equation assuming that the state k evolved from the prior state at time $k-1$ is like the following:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{1}$$

Moreover, we also need measurement model z_k that describes a relationship between the state and measurement at the current step k like the following:

$$z_k = Hx_k + v_k \tag{2}$$

- **A**: It is a transition matrix relating the previous time step $k-1$ to the current state k (nxn matrix).
- **B**: It is a control input matrix applied to the optional control input u_{k-1} (nxl matrix).
- **H**: It is a transformation matrix, which performs transformation of the state in the measurement domain.
- $\mathbf{w}_k$: It is a process noise vector, which is assumed statistically independence with the normal probability distribution.
- $\mathbf{v}_k$: It is a measurement noise vector, which is assumed statistically independence with the normal probability distribution.

4.1 Prediction

In order to update time, we need to predict a priori state $\hat{\mathbf{x}}_k^-$ and predict a priori error covariance $\mathbf{P}_k^-$. The priori state is predicted by using the equation $\hat{\mathbf{x}}_k^- = A\hat{\mathbf{x}}_{k-1}^- + B\hat{\mathbf{u}}_{k-1}^-$ ($\hat{\mathbf{x}}_{k-1}$ is the previous estimated state).
Then the error covariance matrix is predicted by $\mathbf{P}_k^- = A\mathbf{P}_{k-1}^- A^T + Q$ ($\mathbf{P}_{k-1}$ is the previous estimated error covariance matrix).

4.2 Update

During the update, we compute the Kalman gain $\mathbf{K}_k$ using the equation $\mathbf{K}_k = \mathbf{P}_k^- H^T (H\mathbf{P}_k^- H^T + R)^{-1}$.
Then we need to calculate measurement residual, which is the difference between the true measurement z_k and the previous estimated measurement $H\hat{\mathbf{x}}_k^-$, which becomes $\mathbf{z}_k - H\hat{\mathbf{x}}_k^-$.
Updating the predicted state $\mathbf{x}_k^-$ achieve by the summation of the previous updated state $\mathbf{x}_k^-$ and the product of the Kalman gain and the measurement residual, which is $\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - H\hat{\mathbf{x}}_k^-)$
Finally, we can update the error covariance $\mathbf{P}_k$ by using the next time step, $\mathbf{P}_k = (I - \mathbf{K}_k H)\mathbf{P}_k^-$ (I is an identity matrix).

4.3 Application of the Kalman Filter

This section mainly shows the result of applying the Kalman filter on the stored 2D trajectory from the AdaBoost to improve tracking estimation. Kalman filter is essentially a set of mathematical equations that consists of a predictor-corrector type estimator, which demonstrates outstanding performance on minimizing the estimated error covariance. Taking these properties into the advantage, we could successfully improve the trajectory estimation. Figure 5 shows that there were some break portions in the original trajectories on the left side. After applying the Kalman filter, all of the distinct break portions of the trajectory are recovered. The result shows that the Kalman filter is robust to cope with several cur-offs in the trajectory in a non-linear motion.

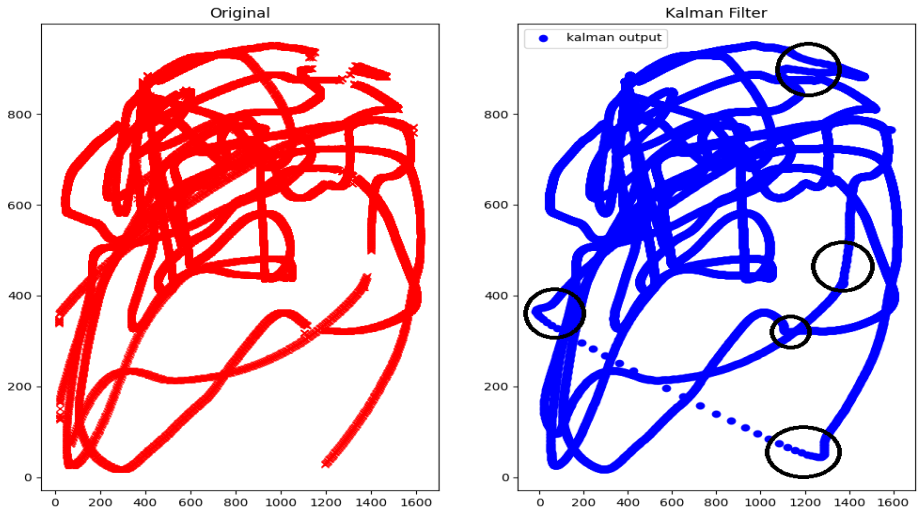


Figure 5: Kalman filter output from the original trajectory. Cut-offs that are recovered from the filter are specified by a black circle.

5 Ad-hoc Camera Network

Until this section, we only produced the 2D trajectory of the drone in each of the seven videos from the dataset that were filmed with cheap and easy-to-deploy types of equipment. Now it's time to reconstruct 3D trajectories from unsynchronized consumer cameras. We only assume intrinsic parameters for calibration for each camera, such as focal length and radial distortion. All other parameters of the seven camera setup are recovered during the operation, which is synchronization between different cameras and camera poses. Thus, the network observes flying object itself as a feature to calibrate all the cameras. In the following sections, we will demonstrate how initial geometry computation works.

5.1 2D Correspondence

We use a spline approximation of the trajectory in 2D image space to obtain 2D $\leftrightarrow$ 2D correspondence. Suppose there is an image point x_i and check whether three consecutive overlapping points are generated between two cameras i and k. If this is the case, then we fit a spline to those points,

$$S_k(t) = \sum_m b_m(t) K_{km} \tag{3}$$

$S_k(t)$ are the coordinates of the trajectory at time t, $b_m(t)$ are the basis functions and K_{km} are the spline coefficients.

5.2 Time Shift Estimation

We have to fit the unknown temporal offset between the cameras. We approached the relative time shift under a linear approximation of the 2D image point trajectory,

$$(\hat{x}_k^j + \beta_{ik} v_k^j)^T F x_i^j \tag{4}$$

F_{ik} is the fundamental matrix between two cameras. Also, β_{ik} performs the relative time shift. Estimation of β_i applies to all camera pairs to obtain improved values.

5.3 Building Trajectory

The sequence of 3D spline curves parametrized by the time t creates the drone's trajectory like below:

$$T_r(t) = \sum_u b_u(t) K_u \tag{5}$$

Whenever a new camera arrives, the algorithm tries to find image points that are not part of the already constructed parts, and construct 2D $\leftrightarrow$ 2D correspondences. Then triangulate those points and extends the spine until it covers all available cameras. Figure 6 shows the trajectory constructed by applying all the steps explained so far on dataset 4.

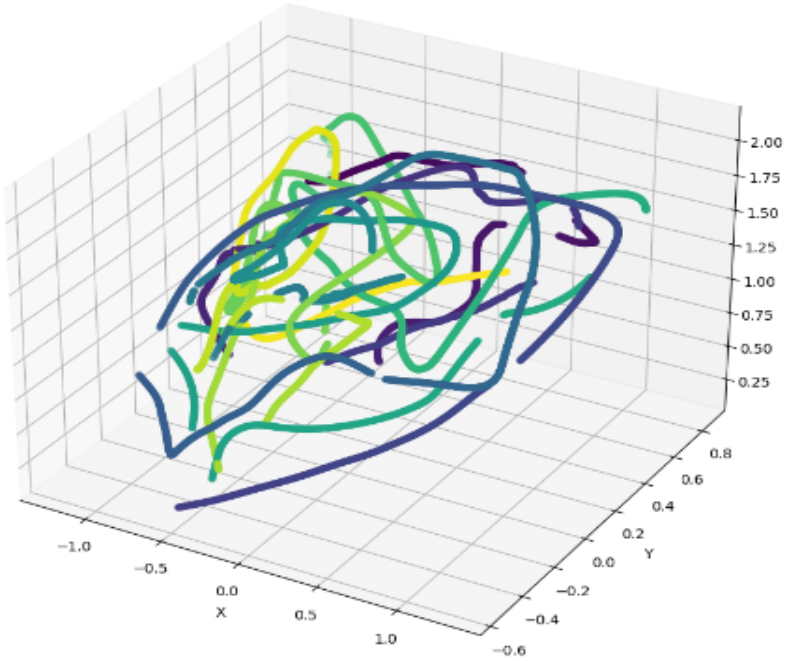


Figure 6: 3D trajectory obtained from the ad-hoc network using the 2D trajectory information from dataset 4

5.4 Bundle Adjustment

To optimize the estimated parameters, which are the camera poses P_i, 3D trajectory parameters $T_r(t)$, time offsets $\beta_1, ..., \beta_n$, time scales $\alpha + a, ...\alpha_n$, and rolling shutter scan speeds r_i, we need to minimize the image re-projection errors. To achieve this, we introduced two types of regularizing the trajectory. Before that, we would like to define spline trajectory as optimizing parameters,

$$argmin_{P_i,\alpha_i,\beta_i,r_i,K_r} \sum_i \sum_j \left\| x_i^j - \mu(P_i T_r(t_i^j)) \right\|^2 \tag{6}$$

where $t_i^j = \alpha_i j + \beta_i + r_i x_{2i}^j$, due to the rolling shutter effect in our optimization where x_{2i}^j is the y-coordinate of the image as majority of cameras are equipped with rolling shutters (RS).

Fisrt type is the ***least kinetic energy trajectory***. This model makes good performance for dynamic 3D objects. Since the kinetic energy of an object with mass is defined as $E_k = \frac{1}{2}mv^2$, if we assume the object is a point mass with $m = 1$, then the instant velocity at the t_i^j is $\left\|(X_i^{j+1} - X_i^j)/(t_i^{j+1} - t_i^j)\right\|^2$. It leads to optimizing,

$$\sum_i \sum_j \left\| \frac{X_i^{j+1} - X_i^j}{t_i^{j+1} - t_i^j} \right\|^2 \tag{7}$$

However, estimating the motion based on the instant velocity does not always guarantee the proper trajectory reconstruction. Thus, the least kinetic energy approach remains as the above spline trajectory definition as in equation (6).

Second type is the ***least force trajectory***. The force needed to accelerate an object with mass m is $F = ma$. If we assume the mass of the object is $m = 1$, then the instant acceleration is $a = \left\|(X_i^{j+1} - X_i^j)/(t_i^{j+1} - t_i^j) - (X_i^j - X_i^{j-1})/(t_i^j - t_i^{j-1})\right\|$. Then it leads to optimizing,

$$\sum_i \sum_j \left\| \frac{X_i^{j+1} - X_i^j}{t_i^{j+1} - t_i^j} - \frac{X_i^j - X_i^{j-1}}{t_i^j - t_i^{j-1}} \right\|^2 \tag{8}$$

We perform the optimization whenever a new camera has been added during the bundle adjustment process starting from the first two cameras' geometry estimation.

By using the two different types of regularizing methods, we can compute the mean absolute errors and the root mean square error (RMSE). We first fit a euclidean transform to align the reconstructed trajectory to the ground truth from the real-time kinematic (RTK) file provided from dataset 4. The mean error of the reconstructed camera positions was about 64cm, which is a very good localization accuracy considering the baselines in the order of 100m. Also, we obtain an RMSE slightly above 47cm, which is a significant improvement compared to the original 3D trajectory errors proposed in [2]. The visual example of the trajectory is shown in Figure 6. We do want to point out that the comparison is not completely exact because we missed some portions of the trajectory, which are the start and end of the drone near the ground.

		Least kinetic		Least force	
		w/o RS	RS	w/o RS	RS
Dataset 4	Mean	63.90	63.89	63.98	63.95
	RMSE	47.03	47.02	47.22	47.09

Figure 7: 3D trajectory errors on dataset 4 with ground truth, in centimeters.

6 Keyhole Markup Language

This section introduces how we draw trajectories in an Earth browser such as Google Earth using the KML file. KML is a file format to display geographic data based on the XML standard. We used *LineString* tag to demonstrate our trajectory, which allows us to define a connected set of line segments. KML file stores coordinate information in latitude, longitude, and optionally altitude format. Thus, our 3D trajectory points are replaced with the corresponding format.

```
1: procedure GENERATEKML
2:     data ← read("filename")
3:     file ← open("filename.kml")
4:     i ← 0.
5:     loop:
6:     if len(data[i]) > 0 then
7:         file.write(" < coordinates > ")
8:         file.write(data[i][0], data[i][1], data[i][2])
9:         file.write(" < /coordinates > ")
10:        i ← i + 1.
11:        goto loop.
12:    file.close()
```

(a)

(b)

Figure 8: (a) Pseudocode of generating KML file from 3D data (b) Trajectory visualization on Google Earth

7 Conclusions

In this paper, we have demonstrated a robust real-time tracking technique that formulates the tracking as a binary classification between object and background. Since coping with the variations in appearance during tracking was the key, we brought on-line AdaBoost algorithm which updates features of the tracker during tracking the object. Then we presented a two-step based implementation of the Kalman filter, which is a very powerful tool when our 2D trajectory detections have some noises like cut-offs. After improving trajectory estimation, we reconstruct the trajectory of the drone using seven different 2D paths obtained from the external cameras. The network takes care of calibration, rolling shutter effects, and geometry computation to produce cm-accurate trajectories. This implemented system can be applied to any computer vision application for moving object detection and tracking. A logical next step would be implementing multi-target tracking to reconstruct the trajectory online.

References

[1] H. B. Helmut Grabner, Michael Grabner. Real-time tracking via on-line boosting, 2006.

[2] D. I. K. S. Jingtong Li, Jesse Murray and C. Albl. Reconstruction of 3d flight trajectories from ad-hoc camera networks, 2020.

[3] M. Laaraiedh. Implementation of kalman filter with python language, 2012.