Formal Description of the C-14 Verifier

The verifier of C-14 takes as input a video, its metadata, and a claimed timestamp t_b. Using the claimed timestamp, the verifier retrieves the corresponding motion program or the random seed R from the trusted public source. In case of R, the verifier needs a further action to map R to a motion program. The verifier then compresses the video with frame resizing and frame skipping techniques, verifies whether the metadata is consistent with the motion program, estimates the motion from the compressed video, and verifies if the motion estimation matches the metadata. We will describe the technicalities of these stages in the following text.

1 Acquiring the Motion Program

With the claimed timestamp t_b, the verifier first needs to acquire the corresponding motion program, and then verifies the consistency of the motion program, the metadata, and the motion depicted in the video. The verifier uses t_b to retrieve either a full flight plan or a number R from the trusted public source. The full flight plan (e.g. the flight plan from Litchi) is used directly as the manual motion program, while the number R is used to seed a known algorithm to generate a series of motions, which forms the algorithmic motion program. For our example algorithmic program, we draw the seed R from the block hash of Ethereum blockchain. R then is used to seed a random number generator which produces a sequence of random numbers r_0, r_1, ... The series of hotpoint motions, which forms the motion program, is derived from $hotpoint_i = (angle = r_{2i}\%360, clockwise = (abs(r_{2i+1})\%2) == 1)?true : false)$.

2 Compressing the video

In this stage, we compress the video with two techniques: frame resizing and frame skipping. Frame resizing reduces the resolution of all frames of the video using the function resize() with INTER_AREA interpolation from OpenCV. Frame skipping reduces the frame rate of the video by maintaining one frame among every s frames. For example, if the skip rate is 5, the frames whose indexes are the multiples of 5 are maintained, i.e. frame #0, #5, #10,... Compressing the video can significantly reduce the time expense on verification.

3 Verifying the Metadata

Metadata is the log of the flight during which the input video is taken. Different implementations of the drone control program have different formats of the metadata, but generally we require it to include the description of the motions that should be contained in the video along with the time period each motion runs. In this stage, we verify if the series of motions claimed in the metadata matches the motion program.

Manually planned programs is consists of a series of waypoints, each of which specifies the latitude, longitude, altitude, headings, tilt and roll of the camera, and etc. that the drone should follow during flight. The metadata of this type of programs records a series of waypoints that the drone followed. The verification of the metadata for manual programs is straightforwardly by comparing if these two matches.

For programs generated by our example algorithm, we check if every hotpoint motion claimed in the metadata exactly matches that of the motion program. More specifically, the degrees of the angle should be equivalent to each other, and the angle should be both clockwise or both counterclockwise.

Additionally, we also extract from the metadata the time each motion begins and ends in this stage. In most cases, the timestamp recorded in the metadata is from the system clock, which, in our control program for DJI Mavic Air, keeps count of the number of mini-seconds elapsed since the epoch. We convert the timestamp t (in mini-seconds) to "framestamp" f, which represents the frame in the video that corresponds to the timestamp, simply by $f = FPS * (t - t_s)/1000$, where t_s is the timestamp (in mini-seconds) the camera launches, and FPS is the frame per second of the input video. Notice that f is the "framestamp" in the original video. If we compress the video with skip rate s, the "framestamp" should be further divided by s since each frame in the compressed video represents s times as large the timespan of the original video.

4 Estimating the Motion

After matching the metadata and the motion program, we need to verify if the video matches the metadata to finally establish equivalence between the video and the motion program. Before doing that, we need to estimate the motion in the video, which involves two steps: computing optical flow and inferring the motion from optical flow.

Computing optical flow is the process of analyzing a video to show how pixels move from one frame to the next. The output of an optical flow algorithm is a two dimensional vector field that gives the magnitude and direction of the motion of each pixel for each pair of adjacent frames. The state-of-the-art algorithm is PWC-Net, which we use in our implementation.

After establishing the optical flow for each pair of adjacent frames, we infer the motion depicted in the video using a recent camera pose estimator, whose output is a three rotation parameters defined by Euler angle, and a three-dimensional unit vector defining the direction of translation. Since the motion in the video is all based on the pose of the camera, the output of the pose estimator is all expressed in the camera frame of reference. The estimator requires the focal length in pixels to make a decent estimation, which can be derived by the field of view (FoV) of the camera:

$$f[\text{in pixel}] = \text{Width}_{\text{image}}\ [\text{in pixel}] \cdot \frac{f[\text{in mm}]}{\text{Width}_{\text{sensor}}\ [\text{in mm}]} = \text{Width}_{\text{image}}\ [\text{in pixel}] \cdot \frac{1}{2 \cdot \tan(\text{FoV}/2)}$$

The estimator can still infer the motion without knowing the optical flow for all pixels. Inspired by this, we use a technique, named spatial sampling, to reduce the amount of computation. We divide the optical flow vector field into a grid, and randomly sample pixels from grid squares (one pixel per grid), to make sure sampled pixels are spread evenly. The chosen pixels are then input to the camera pose estimator.

For our example algorithmic programs, we also use another acceleration technique, called temporal sampling. Due to the constant rate of rotation and translation of the hotpoint and fly in/out motion, we don't need to compute the optical flow and infer the motion for the whole video. Instead, for each hotpoint or fly in/out, we can uniformly randomly sample many small video segments, each of which contains several consecutive frames, and compute their optical flow and pose estimation. These segments are enough to prove whether the motion depicted in the video matches the required motion program described in the metadata. We will describe how segments are used for verification in Section 6.2.

5 Translating the Frame of Reference

As we mentioned in Section 4, the output of the camera pose estimator is all in the camera frame of reference, but for convenience of verification, we need to translate the rotation and the translation from the camera frame of reference to the ideal drone frame of reference.

We first focus on the harder part of the problem, i.e. rotation. The estimation of rotation outputted by the camera pose estimator is an Euler Angle $e_{camera_1,camera_2} = (\alpha, \beta, \gamma)$ between two adjacent frames in the video, say frame #1 and #2. Starting from the frame of reference of frame #1, after rotating the z-axis by angle α, the new y'-axis by angle β, and then, the new x''-axis by angle γ, we get the frame of reference of frame #2 [1].

The problem can be formally defined as: Assume, for two adjacent frames in the video, i.e. frame #1 and #2, the pitch angles between the ideal drone and the camera are θ_1 and θ_2, respectively. Given the Euler angle $e_{camera_1,camera_2}$, i.e. the output of the camera motion estimation, which transforms the camera frame of reference from frame #1 to frame #2, how do we derive $e_{drone_1,drone_2}$, which transforms the ideal drone frame of reference from frame #1 to frame #2?

We can derive the solution to this problem with the bridge of rotation matrix. Rotation matrix $R_{a,b}$ represents a rotation from the frame of reference a to b. Two important properties of the rotation matrix are $R_{a,c} = R_{a,b}R_{b,c}$ and $R^{-1}_{a,b} = R_{b,a}$. We assume $R(e)$ converts the Euler angle to its corresponding rotation matrix, while $e(R)$ converts the rotation matrix to its corresponding Euler angle. Euler angle and the rotation matrix can be converted to each other easily.

The formula for the problem is then a direct application of the two properties of the rotation matrix:

$$\begin{aligned} & e_{drone_1,drone2} \\ =& e(R_{drone_1,drone_2}) \\ =& e(R^{-1}_{camera_1,drone_1} R_{camera_1,camera_2} R_{camera_2,drone_2}) \\ =& e(R(e_{camera_1,drone_1})^{-1} R(e_{camera_1,camera_2}) R(e_{camera_2,drone_2})) \end{aligned}$$

[1]We followed the standard of ZYX Euler Angle.

where $e_{camera_1,drone_1} = (0, \theta_1, 0)$ and $e_{camera_2,drone_2} = (0, \theta_2, 0)$. This is because we only need to rotate the y-axis of the camera frame of reference by θ to get the ideal drone frame of reference.

As for translation, based on the definition of rotation matrix, it is not difficult to translate t_{camera}, the vector of translation in the camera frame of reference, to t_{drone}, the corresponding vector in the ideal drone frame of reference:

$$t_{drone} = R(e_{camera,drone})^{-1} \cdot t_{camera}$$

6 Verifying the Video

In this stage, we verify the correspondence between the video and the metadata, thus finally establishing the equivalence between the motion depicted in the video and the motion required by the program via the bridge of the metadata. We verify if every motion claimed in the metadata matches the corresponding timespan of the video. Manual programs and example algorithmic programs have different verification steps, so we describe them in two subsections, respectively.

6.1 Manual Program

The metadata of manual motion programs should contain a series of waypoints and the time when the drone arrives at them. Each waypoint describes the latitude, longitude, altitude, headings, tilt and roll of the camera, and etc., for the drone to follow. After the verification of metadata, the waypoints claimed in the metadata should be the same as the flight plan. The task of verifier in this stage is to check if the motion between every two adjacent waypoints is depicted in the video. It completes this task by comparing both the yaw component of rotation and the direction of translation for each pair of adjacent waypoints.

We first need to convert the data recorded in the metadata to the ideal drone frame of reference so that we can compare them with the motion estimated from the video, which has also been translated to the ideal drone frame of reference. We only need to get the yaw and the translation direction for comparison.

By definition, the tilt and the roll are 0 in the ideal drone frame of reference. The yaw corresponds to the difference of headings between two consecutive timestamps:

$$yaw_{(t,t+1)} = heading_{(t+1)} - heading_{(t)}$$

The translation along the z-axis is the difference of the altitudes.

$$translation_{z(t,t+1)} = altitude_{(t+1)} - altitude_{(t)}$$

To compute the translation along the x-axis and the y-axis, we use the geodesic distance with the WGS-84 ellipsoid model. The translation along the x-axis is given by the distance between the point $(lat_t, long_{t+1})$ and the point $(lat_t, long_t)$. The translation along the y-axis is given by the distance between the point $(lat_{t+1}, long_t)$ and the point $(lat_t, long_t)$. To express the translation vector in the ideal drone frame of reference, we rotate the translation vector by the heading of the drone. We then normalize the translation vector to match the normalized translation vector coming from the motion estimator.

Once we get the yaw and the translation direction for each point recorded in the metadata, we can start comparing them with the video. The verifier sums the yaw between each pair of adjacent waypoints, and compares it to the total yaw of the corresponding timespan of the video. If the difference of the two sums is within a threshold for every pair of adjacent waypoints, the verifier is satisfied. Then, the verifier averages all the normalized translation vectors between each pair of adjacent waypoints, compares it with the corresponding average translation vector from motion estimation, and see if the angle between these two average translation vectors are all smaller than a threshold for every pair of adjacent waypoints.

For videos from Litchi, their companion metadata files are missing. We manually recover them using visual correspondence between the videos and the flight plan. The Litchi flight plan contains a sequence of waypoints. Between each pair of adjacent waypoints, there is a series of drone poses generated by Virtual Litchi Mission, a software that plan the trajectory between waypoints, which instructs the drone to transit from one waypoint to the next. The verifier sums up the yaw error of every drone pose to get the total yaw error for the waypoint pair. Similarly, the verifier averages the translation vector of every pair of drone poses to get the average translation vector for the waypoint pair.

6.2 Example Algorithmic Program

The example algorithmic program is consists of a series of hotpoint and the fly in/out motions. The verifier's task in this stage is to verify if the hotpoint and fly in/out motions claimed in the metadata appear in the corresponding timespan of the input video. Notice that we don't allow gaps between motions to be ignored since it may pose risk, so in our implementation, we also consider the transition between fly in/out and hotpoint as a part of the hotpoint motion. In other words, the hotpoint begins at the end of the last fly in/out and ends at the start of the next fly in/out.

As mentioned in Section 4, we use temporal sampling to reduce the amount of computation. We sample a series of small video segments from each motion for verification. In our implementation, each video segment only contains two adjacent frames from the input video. The camera pose estimator estimates rotation and translation only for these segments, instead of the whole video, which saves a significant amount of time. The number of samples can be controlled by the sample rate, which is the ratio of the number of sampled frames to the total number of frames in a certain motion. Specifically, if the sample rate is 100%, the verifier examines the whole video without temporal sampling.

Once the motion in each segment is estimated and translated into the ideal drone frame of reference, the verifier starts to verify the hotpoint and fly in/out, respectively.

The verification of hotpoint takes two steps: *(i)* checking if the motion depicted in the video is hotpoint *(ii)* checking if the hotpoint in the video is the hotpoint claimed in the metadata.

Hotpoint can be decomposed into yaw and y-axis translation with constant rate if seen in the ideal drone frame of reference. Thus, in the output of camera pose estimator for a typical hotpoint, the absolute value of yaw in the Euler angle should be a large constant and the absolute value of the y component of the translational vector should be ideally equal to 1. If we consider the orientation of the yaw, a clockwise yaw should have positive yaw and negative y, while a counterclockwise yaw should have negative yaw and positive y. To confirm the motion is indeed a hotpoint, we verify if the magnitude of yaw and y in most of the segments satisfy the property.

Checking if the hotpoint is the one claimed in the metadata involves examining both the orientation and the angle. The orientation can be checked by examining if most of the segments are of the correct sign as described in the above-mentioned property. As for the yaw angle, inspired by the definite integral, it can be estimated by fitting the segments with the piece-wise linear function and calculate the area below the function. Imagine there is a coordinate system where x-axis is the "framestamp" of the video and y-axis is the magnitude of yaw [2]. Assume we have computed all yaw values for a certain hotpoint motion. Put these values to the coordinate system according to their corresponding framestamps. Then, we have a way to visually represent the yaw as a function F, and the area enclosed by F and the x-axis is the magnitude of the yaw angle of the hotpoint. However, we have only computed the yaw values for sampled segments. If we also put these computed yaw values to the coordinate system, and use a piece-wise linear function F' to fit these points. As long as we have sampled enough segments, thus enough computed yaw values, F' would be a good approximation of F. In our implementation, we simply connect the points along the x-axis to form the piece-wise linear function F'. In the extreme case, if the sample rate is 100%, our implementation will leads to $F' = F$. We can use F' to estimate the yaw angle by calculating the area enclosed by F' and the x-axis because this area will be approximately equivalent to the area enclosed by F and x-axis as long as $F' \approx F$. If the error between the estimated yaw angle and the angle claimed in the metadata is within a threshold, the verifier accepts the angle of the hotpoint.

Remember that fly in/out plays the role of bookending hotpoint motions. We only need to check there is no appreciable yaw during this motion. We use the same method as the hotpoint motion to estimate the total yaw value during the fly in/out motion, i.e. fitting a piece-wise linear function. If the estimated angle is too large, the verifier will not pass the verification.

The video compression, spatial sampling, and temporal sampling all reduce the amount of computation significantly. However, they sacrifice the accuracy of the verifier, and increase the false negative. To boost accuracy, we can always try to run the verifier again with less video compression, smaller spatial sampling and/or less sample rate.

[2] In fact, each framestamp represents two adjacent frames in the original video, and the yaw represents the yaw component of rotation between these two frames.