C-14: Assured Timestamps and Flight Paths for Drone Videos

ABSTRACT

Inexpensive and highly capable unmanned aerial vehicles (aka drones) have enabled people to contribute high-quality, aerial videos at a global scale. However, a key challenge exists for accepting videos from untrusted sources: establishing when and how a particular video was taken. Once a video has been received or posted publicly, it is evident that the video must have been created before that time, but there are no current methods for establishing how old it is or what flight pattern the drone took while creating the video.

We propose C-14,[1] *a system that prescribes taking a video using a particular flight pattern, which C-14 uses to verify the earliest time a video could have been taken while also verifying the prescribed flight pattern. Using recent developments in optical flow and camera pose estimation, we show that we can establish both the time and flight pattern using both unconstrained flights with many waypoints and drone motions, as well as constrained flights that employ steady, easy to verify motions. Through extensive experiments we show how to verify a 59-second unconstrained video with eight motions in 91 seconds of computation with a false positive rate of one in* 10^{13} *and no false negatives. We also verify a 190-second constrained video with 4 motions in 158 seconds of computation with a false positive rate of one in one hundred thousand and no false negatives.*

1. INTRODUCTION

The continuous proliferation of drone-mounted, high resolution video cameras is ushering in an era of *global scale video sensing.* For instance, drones have enabled citizens to provide video coverage of land areas that were previously inaccessible. In areas of Latin America and Asia, citizen-powered drones are already proving crucial in providing imagery and video to monitor large areas of land vulnerable to unauthorized development such as deforestation [40]. Similarly, freelance journalists are producing invaluable evidence from war zones and other difficult to cover areas from the air [1].

[1]The name C-14 comes from the radioactive isotope used for carbon dating organic matter.

However, a key challenge is ensuring the trustworthiness of videos. For instance international agreements on climate change will require detailed monitoring of land-use changes around the globe, something that can be aided by user-contributed, aerial videos. However, relying on such videos for policy decisions requires knowing *when* and *how* the videos were taken. If the video is older than claimed or if the drone flight did not follow a prescribed pattern for collecting the data it may not be useful.

A number of efforts have addressed two other crucial problems: *(i)* location establishment: the video was taken at a particular place [21,35,43], and *(ii)* integrity assurance: the video was unaltered after recording (e.g. spliced, re-encoded, etc.) [23,25,33,35,41]. However, we are unaware of techniques that establish how old a video is or what flight motions created it.

We propose a system, C-14, that assures a video was not created before some time t_b and that the drone flew a particular pattern to gather that video. It is not sufficient to merely post a video to a public forum as the video may be much older than the posting time. To establish that a video was taken after a time t_b, C-14 requires a pattern of movements in the video that could only be known by the creator after time t_b. This pattern of movements can be provided by a trusted party only after t_b or could be derived from values publicly known after t_b. If the video matches that pattern this demonstrates that the video could only have been produced after t_b and that it followed the prescribed motion pattern during flight. This pattern can either be placed in the middle of a typical drone flight to establish just a time stamp, or the motion can be used as the drone flight itself to establish a timestamp and verify the entire flight.

The pattern is a series of translations (move up/down, left/right, forward/backwards), or rotations (yaw, roll, tilt), and combinations of the two. We demonstrate the system on two types of patterns: *unconstrained* highly variable drone motions over wide areas, and *constrained* motions over a small area. Unconstrained motions can be incorporated into typical drone flights, however constrained motions are more amenable to sampling small parts of the video, which speeds verification.

C-14 incorporates a number of techniques to speed verification: compression, skipping frames, spatial sampling, and temporal sampling. A full analysis of the video to verify the motion runs in 2000x real-time (a 2 minute video takes 3 days of GPU time to verify), but through compression and sampling we can reduce the amount of time needed. C-14 can verify a 59-second unconstrained video with 8 motions in 91 seconds of computation with a false positive rate of 1 in 10^{13} and a 190-second constrained video with 4 motions in 158 seconds of computation with a false positive rate of 1 in one hundred thousand and no false negatives. The overhead for incorporating C-14 into drone flights is minimal: either no overhead in the case of unconstrained flights and at most 15% of the flight time for constrained flights.

2. ASSUMPTIONS AND THREAT MODEL

We have made certain assumptions in building C-14:

Self-contained system: To establish a timestamp, at the beginning of the flight the drone could video a large screen on the ground showing a 2-D barcode of a signed timestamp. However, this requires equipment external to the drone, complicating the deployment. Further, it only establishes a timestamp and does not verify that the drone flew in the requested pattern. Since the 2-D barcode only appears at the beginning of the flight it becomes a single point of vulnerability to possible video editing attacks. C-14 encodes patterns into a much larger part of the video, marrying the timestamp, the flight pattern, and the subject matter.

No Trusted Hardware: C-14 does not rely on any trusted hardware on the drone, such as a TPM. Such hardware is not yet readily available on drones and it would need to encompass the clock, flight controller, and video sensor to be effective at establishing a timestamp. Instead we have targeted off-the-shelf devices.

Flight Pattern Unknown Before t_b: The flight pattern must have sufficient entropy so as to be unguessable. The pattern can be derived from a public source, such as a blockchain, or from a trusted third party that reveals a random seed or full flight pattern at a certain time.

Limited Motion: The system has been built and tuned around largely static scenes, such as buildings, forests, etc. Drone flights in the USA cannot be done over live subjects, which makes quantifying this limitation difficult. However, techniques to remove independent object motion from scenes [12,13,18,38] could be used to increase the robustness of C-14.

Featured Scenes: We assume that the subject of the video is not mono-tone or mono-textured, such as a field of snow or a featureless desert. Such scenes pose difficulties for the optical flow algorithms we use in C-14.

Speed-Agnostic: We do not make assumptions about the speed of the drone. Verification of the video is done on a frame basis, not based on time.
We assume a particular threat model to the system, and classify many attacks as solvable through other mechanisms. Most importantly C-14 can only assure that the video was created after t_b and the motion pattern of the camera. It cannot, on its own, assure the integrity of the video, or its location or subject matter. Here we detail a few of these attacks:

Video Splicing: An attacker could take a large number of videos from a location, each of which performs one of the movements required from the authenticated video. Once the sequence of movements is known, one can splice together those movements into a video and present it as authentic. However, such splices can be detected in videos through a number of techniques [9–11,19,20].

3D Rendering/Fake Video: If an attacker can generate a video that is a full reconstruction of a scene, they can arbitrarily create any pattern (rotations and translations). Detecting videos created from whole-cloth is outside of the scope of our work, but the image forensics community has worked on detecting such reconstructions based on a number of techniques, for example camera noise [16], the smoothness of images [28] and machine learning classification [29]. A similar technique would encompass such videos created by so-called "bullet-time" or 3D reconstructions from large numbers of photographs.

Altered Optical Flow: Similar to a 3D rendering, one might be able to take a genuine video, and frame-by-frame create pixel movements that create the correct optical flow. However, we are unaware of anyone who has successfully shown how to manipulate a video this way. The closest known attack is to add a well chosen small patch in the video, which will result in large modifications of the optical flow [34]. But this method does not create a specific optical flow. We also believe that such heavy manipulation of the video will result in a very distorted video as occlusions and disocclusions occur in real videos that are difficult to fake.

Most importantly it is best to think of C-14 in the same light as a CAPTCHA—we can provide some assurance and raise the bar for a malicious actor. C-14 is only one piece of a system for assuring properties of drone videos.

3. DRONE BACKGROUND

Here we provide background on how drones fly and collect videos. We focus on 'copter' drones which typically have four rotors allowing the drone to translate in three dimensions, as well as rotate around three axes, as labeled in Figure 1. The drone is equipped with a front-

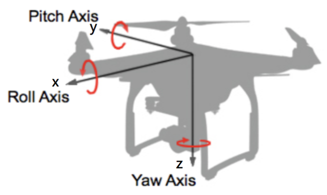


Figure 1: This figure demonstrates the axes of movement found in a copter drone. Image from DJI documentation [2].

facing video camera, mounted on a three-axis gimbal. The gimbal can be manually pointed in any direction, but typically it is set to *yaw-follow* meaning that the gimbal tries to maintain a constant pitch angle and zero rotation with respect to the world horizon. The yaw of the camera follows the drone's heading, though it does so with some elasticity to prevent sudden motions in the video.

For the remainder of the paper we use a frame of reference we refer to as the *ideal drone* frame of reference. This frame of reference is the drone without roll and pitch induced by aircraft motion. Consider what happens when the drone moves to the right (a positive y-axis translation). The aircraft adjusts the speed of the propellers to roll slightly right (a positive roll), which makes the aircraft move to the right. The gimbal counteracts this motion and the resulting video has no roll. In the ideal drone frame of reference the drone would always appear to have no roll and pitch, but it does yaw.

4. MOTION PROGRAM

C-14 depends on the untrusted creator of the video not knowing the sequence of motions, which we refer to as a *motion program*, before time t_b. There are two general ways to create the motion program: *unconstrained* and *constrained* programs. In an unconstrained program we use a hand-crafted sequence of motions over a wide-area including any type of motion a drone is capable of, such as yawing at different rates while changing the gimbal pitch. In contrast, a constrained program uses constant motions, meaning that the translation and rotation does not vary during the motion. For instance if the drone translates forward, it does so without changing direction, or if it yaws, it does so at a constant rate. The advantage of a constrained program is that it is more amenable to sampling because the drone's rate of rotation or translation can be verified by looking only at randomly selected parts and assuming the motion is the same over that period. The advantage of an unconstrained program is that it may fit more naturally into systems where a particular path needs to be followed over a wide area, rather than a computed path.

4.1 Unconstrained Program

In unconstrained programs, we create motion programs that are sufficiently complicated such that guessing the program ahead of time would be improbable. As long as we only divulge the program to the video creator after t_b, we can show that the video was created after that time and that the drone flew in a prescribed series of motions.

We have chosen to describe unconstrained programs using a popular drone flight planning system named Litchi [4]. Figure 2 shows an example motion program—each plain numbered pin represents a *waypoint* for the drone to fly to, and each numbered pin containing a camera icon represents a *point of interest* for the drone to focus on. The point of interest has an altitude as well, creating yaw and gimbal pitch throughout the motion. The curves around waypoints represent the actual flight path to be taken to smooth out the drone's motion. Recall that C-14 does not verify where the video was taken, only the drone's motions, so this flight plan could be applied anywhere and the video will be accepted as valid.

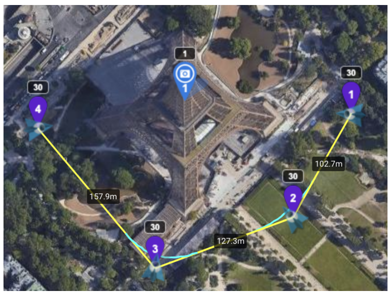


Figure 2: A sample unconstrained mission from the Litchi flight planning software.

Such unconstrained motions are highly-complex, and thus there are a very large number of distinct possible drone paths. C-14 measures both the yaw between waypoints, as well as an average translation vector to verify the video. As we show in Section 7.6 even small deviations can be detected in the resulting video, ensuring that the number of distinct drone paths is very large.

A side benefit of using Litchi missions is that many users publicly post their flight plans and the resulting video from the drone, giving us a varied dataset to work with for our evaluation.

4.2 Constrained Program

The disadvantage of an unconstrained program is that the motion between waypoints is not a constant rate. So when verifying the unconstrained flights we have to sample more of the video, slowing down verification. An alternative to the unconstrained program is a constrained one that uses a small number of short, steady, and mathematically derived motions. This motion program can be used at some point of a flight to establish a timestamp for the whole video.

The constrained motion program is a sequence of motions, $m_1, m_2, ...$ deterministically derived from a number, R, with sufficient entropy, where R is not known before t_b. To keep the system understandable m_n is always a combination of a rotation on one axis and/or a translation along one axis. An example motion would be flying in a circle while remaining pointed at a center point, which is a combination of a yaw rotation with translation to the side of the drone. This is commonly called a *hotpoint* motion: the drone circles, and points the camera at a point of interest. We use hotpoints as the key motion in the constrained program.

The hotpoint motion also must execute for a certain magnitude: the total number of degrees over the time period of the motion. To derive the number of degrees of rotation in the video, C-14 requires the *field of view* (FOV) of the camera. If the video creator supplies this parameter, it can scale the total amount of rotation perceived in the video. However, as we note in 6.3.3 we can use a typical FOV for a drone and achieve good results as drones typically use very similar cameras. One attack on the system is to dynamically zoom the video, which could modify the total rotation in a hotpoint. However, zooming will create different views when looking at the object at different times (which happens many times in the constrained program)—this would be detectable. We leave a thorough exploration of this issue as future work.

We also make the motions time-independent and only measure motions based on sequence of motions that occurred. This makes the system portable across drones with varying capabilities in speed and frame rate.

We have created a simple constrained program based on a single, high-entropy number, R, that would only be known after t_b. As an example we used the block hash from the Ethereum blockchain from a block that occurs shortly after t_b, but any distributed or centralized trust could produce such a number.

The drone flies a series consisting of two motions: a motion to fly towards the point of interest to an inner radius and then back out again (called a fly in and out) followed by one hotpoint motion for *angle* degrees randomly chosen from 0 to 360 degrees along an outer radius, either clockwise or counter-clockwise (called a hotpoint). The purpose of the translation is to provide a separation between hotpoint motions. If there is no separation, the verifier cannot attribute the yaw to each required hotpoint. During the fly in and out motion we only need to know that the drone did not appreciably yaw for some number of frames. The motion program ends with a fly in and out to bookend the last hotpoint.

All of the motions are done with the drone pointing it's camera at one point of interest at the center of an circle, but with constant gimbal pitch. An example motion program is shown in Figure 3. The angle and direction of the hotpoint motion is determined by the high-entropy number R by seeding a random number generator that produces a series of random numbers $r_0, r_1, ...$ The motion program is a series of hotpoint motions for a certain number of degrees, as: $hotpoint_i = (angle = r_{2i}\%360,$ $clockwise = (abs(r_{2i+1})\%2) == 1)?true : false)$.

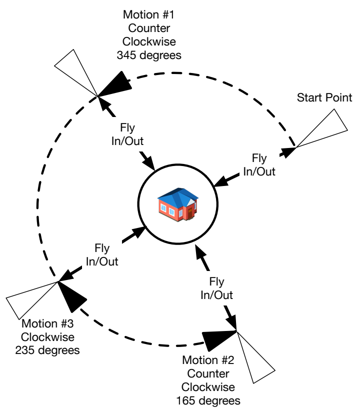


Figure 3: This figures shows the sequence of motions found in our example constrained motion program. The motions are a sequence of hotpoint motions and fly in and out motions, all centered on a point of interest.

4.3 Probability of False Positives/Negatives

An attacker could perform a brute-force attack: producing videos conforming to a random motion program with the same length and then submitting them as authentic. A false positive is defined as: *the probability a video with a random set of N motions is accepted as valid.* While other stop-gaps would prevent a large number of videos being produced and submitted, it is valuable to know how likely it is that a random video would be accepted. Assuming that the number of motions N is known, a random attack will pick N random rotations in [-360, 360] degrees (-360 is a counter-clockwise full circle, and 360 is the same, but clockwise). As we show later, some tolerance (aka threshold) is needed to prevent false-negatives. A false-negative is defined as: *the probability that a video is rejected even though it is based on the random value R or the flight plan matching time t_b.*

In the constrained programs there is a series of N rotations that the video must match. So given a true rotation θ_t, a random rotation θ_r is considered correct if it is in the range $[\theta_t - rotThreshold, \theta_t + rotThreshold]$. If we consider N rotations, the probability p_N for the N rotations to be correct is:

$$p_N = \left(\frac{2 * rotThreshold}{720}\right)^N$$

In a constrained video with 4 rotations with a threshold of 20 degrees, that is approximately one chance in one hundred thousand. However, we must also consider that the video creator can look back in time to find a motion program that matches a given video. For instance, if we draw a random seed from a blockchain block, an attacker could generate as many motion programs as there are past blocks and claim that the video is from a matching block. One simple solution is to only accept videos within a small window after t_b, for instance a few hours or days. This means that the attacker can only search a window of time after t_b to find a program that matches the video. Alternatively, the system can limit the number or granularity of random seeds released to video contributors. However, it is relatively easy to increase the entropy of the motion in a constrained program. For instance if the hotpoint motion goes up or down in altitude and moves in or out (a spiral), we increase the possibilities by a factor of 4 and the probability of a false positive decreases to four in one hundred million.

In the unconstrained setting, the entropy of a flight is higher as the flights are longer and contain more motions. Also, as they have long translation periods we also consider the direction during translations (see Section 5.4) and apply a rotation and translation threshold to prevent false negatives. The probability for a random N-waypoint flight to match the flight plan becomes:

$$p_N = \left(\frac{2 * rotThreshold}{720}\right)^{N-1} \times \left(\frac{2 * transThreshold}{360}\right)^{N-1}$$

For a 10-waypoints flight the probability for a random video to match a given flight plan using a 30° rotation threshold and a 40° translation threshold is then smaller than 1 in 10^{17}.

5. VERIFIER

The goal of C-14 is to verify that the motion depicted in the video is consistent with a set of flight instructions given at the beginning of the flight. We leverage recent results from computer vision to estimate the camera motion, and thus the motion of the drone, based on the video. Using this estimated motion, we can verify that the drone has translated and rotated in the specified direction, and in the specified order.

The C-14 verifier takes in an untrusted video, a timeline description of when each motion element occurred (the metadata) and the timestamp claimed with the video. The verifier first ensures that the timestamp claimed for the video, t_b is consistent with the metadata. If that is true, then it must verify that the video matches the metadata. The verifier then produces a pass/fail determination based on the results of each test. We explain each step in detail here.

5.1 Verifying the Metadata

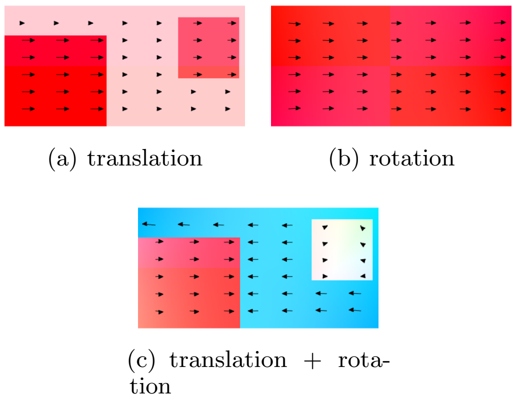

(a) translation (b) rotation (c) translation + rotation

Figure 4: Optical flow due to camera motion of a static scene with static objects located at different depths. The hue of the background shows the angle of flow, and the intensity (or saturation) of the color shows the magnitude of motion.

The metadata provided with the video describes the motions that should be contained in the video. The verifier checks that these claimed motions are consistent with the timestamp claimed in the video. For the unconstrained video the process is straightforward, it must check that the flight plan in the metadata is one that was not revealed until after time t_b. For the constrained videos, the verifier uses the timestamp to fetch the correct random number, R, from the trusted source of timestamped random numbers. Given R, the verifier computes the motion program using the same algorithm used by the video creator and ensures that it matches the motions contained in the metadata for the video.

Additionally, the metadata describes where in the video each motion occurs. The verifier operates on the whole video with no gaps between motions. If gaps were allowed, then part of a motion would be ignored by the verifier, allowing the attacker to modify the motion without changing the video.

An alternative is to ignore the metadata entirely and use the computed motion in the video to recreate the metadata. For instance the verifier can look for where the drone stopped executing a yaw and started to go forward. This transition time would represent the change from a hotpoint motion to a pure translation fly-in motion. However, this requires computing all, or a large part, of the motion estimation from the video, something that is computationally expensive.

5.2 Optical Flow and Motion Estimation

To compute the drone's motion from the video we draw on recent results from computer vision in optical flow and motion estimation. Optical flow is the process of analyzing a video to show how pixels move from one frame to the next. If the scene is largely static (i.e., there are no moving objects) and the camera has no component of forward or backward motion, then the pixels move in a direction opposite that of the camera motion. For example if the camera moves to the right, then the pixels appear to move toward the left (see Figure 4(a)). The output of an optical flow algorithm is a two dimensional vector field that gives the magnitude and direction of the movement of each pixel in the scene. The current best performing optical flow algorithm is PWC-Net [36], which uses a deep convolutional neural network to estimate the optical flow, which we use in our implementation.

Once optical flow has been estimated, we use it to estimate the motion of the camera. Camera motion occurs in three dimensions and can be broken into two components: rotation and translation. Consider the relationship between translation and rotation of the camera. When a camera rotates, it only changes what it is looking at, but objects at different depths do not appear to move in relation to each other. However, when translating, new parts of a scene become visible (disocclusions) or become hidden (occlusions), and objects that are closer appear to move faster than objects further away (see Figure 4). Results from photogrammetry and computer vision have shown that it is possible to disambiguate translation from rotation and thus discover how the pose of the camera is changing from frame to frame [12,13,30,42]. We use a recent camera pose estimator [13] to output a six-valued vector of the camera motion - the three rotation parameters pitch, yaw and roll and a three-dimensional unit vector defining the 3D translational motion direction. Note that this unit vector gives the direction, but not the magnitude (speed) of the translation direction.

The verifier computes the optical flow and motion estimation on the untrusted video which outputs an estimate of the camera motion on three translation axes and three rotation axes. It then matches the camera motion to the *ideal* motion the drone should have followed according to the flight plan or constrained program.

5.3 Translating Frame of Reference

Before making that match we must translate the video into the correct frame of reference. Recall that we describe everything from the *ideal drone* frame of reference described in Section 3. In the ideal drone frame of reference the only rotation is yaw (no pitch or roll), but the drone is free to translate on three axes. However, the video from the drone is taken from a frame of reference of the camera, which may be pitched by θ degrees (generally it is pointed down towards the ground) with respect to the ideal drone. In the constrained videos we fix θ at one value that is provided with the metadata (or could be a globally fixed value). In the unconstrained videos θ needs to be generated as part of the flight plan. This angle is typically not constant. The camera does not roll with respect to the horizon as it has a gimbal, and thus the camera and the ideal drone have no roll. The camera is also set to *yaw follow* mode which means that the yaw generally matches the ideal drone with some elasticity. In the interests of space, we omit the details of this translation here and will include it in the documentation of the source code at publication.

5.4 Verifying Motion in Unconstrained Videos

Figure 5 shows an example of the output of the camera motion in the drone frame of reference compared to the drone motion from the flight plan.

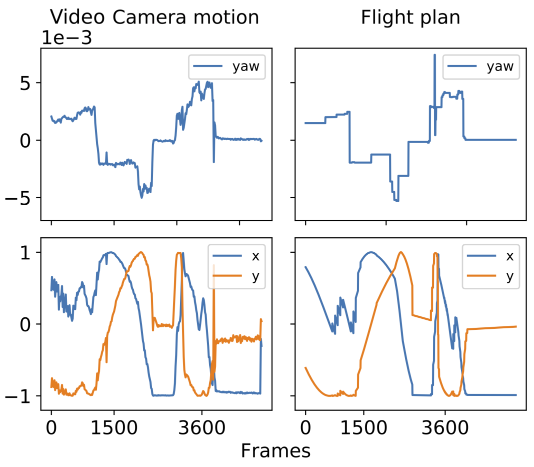


Figure 5: Comparison of the camera motion computed from the video (left) and the Litchi flight plan (right). The camera motion has been computed from a 3-minute video at a resolution of 384*216 and a frame rate of 2 FPS.

To verify unconstrained videos, we check that between each pair of waypoints the camera motion output and the flight plan match. Specifically we check that *(i)* the drone's total yaw and *(ii)* the average error of the angle between the two translation vectors, are both within some threshold.

Let's define two sequences of motions expressed in the ideal drone frame of reference: *(i)* the sequence of rotations and translations at each frame of video from the camera motion estimator and *(ii)* the same sequence of rotations and translations from the flight plan. While the flight plan does not contain speed information, we can still make a correspondence between the output of the camera motion and the flight plan by matching the waypoints in each and then interpolating between points. This requires knowing where in the video the drone reached each waypoint, which we discuss in Section 6.1.

The verifier then sums the yaw in the flight plan between each pair of waypoints and compares it to the sum of the yaw between two waypoints from camera motion computed from the video. If the difference of the two sums are within a threshold for all waypoints, then the verifier is satisfied.

To verify the translation, we average the translation vectors in the flight plan between two waypoints and compares it to the averages of the translation vectors between two waypoints from the camera motion. The verifier then checks if the angle between the two resulting vectors is smaller than a threshold. Note that the average of the translations does not have any physical meaning because each translation vector is expressed in the ideal drone frame of reference at their respective times.

5.5 Verifying Motion in Constrained Videos

The constrained motion program instructs the drone to complete a total amount of yaw during the hotpoint. To find the yaw, the verifier samples frames, chosen uniformly randomly in the interval, fits a curve to the samples using piecewise linear fit, and then integrates the curve. It then checks if the total yaw is within a threshold of the target and a large percentage of the yaw samples are the correct sign (for a counterclockwise motion the yaw should be negative). We discuss the thresholds we use in Section 6.

During this motion the drone is also flying in a positive or negative y-axis direction, which corresponds to a counter-clockwise or clockwise hotpoint respectively. Recall this is in the drone frame of reference, so while doing a hotpoint the drone yaws between frames and moves on the y-axis only. Similar to yaw, a large percentage of the samples must be the correct sign.

Recall that the fly in and out motion is only there to bookend the hotpoint motions. To verify the fly in/out motion, the verifier checks that there is no appreciable yaw for a certain number of frames.

5.6 Sampling

To verify a 133 second video (3993 frames) at 4K UHD resolution (3840x2160) and 29.97 fps, the computation time required for optical flow and camera motion is 26 *hours* on a high-end Xeon processor and a GPU.

We use four techniques to reduce the amount of computation needed to approximately 1 minute (a factor of more than 1500): video compression, frame skipping, spatial sampling, and temporal sampling. Video frame compression simply reduces the resolution of the frames. Verifying lower resolution videos is faster, but may prevent optical flow from recognizing corresponding pixels across subsequent frames. Frame skipping reduces the frames per second (fps) of the video, reducing computation as well, but skipping too many frames poses similar difficulties for optical flow. However, we have found skipping some frames generally has a *positive* effect as it has a smoothing effect on the estimated motion (up to a point where optical flow breaks down). Further, the camera motion algorithm can sample the optical flow by taking the center point from a square grid. Larger grid squares reduce computation, but only up to a point.

For temporal sampling we only examine the motion in discrete parts of the video frames that supposedly contain that motion. This is similar to skipping frames, and the verifier samples after reducing the frame rate, but it does so using uniform random sampling to prevent an attacker from exploiting the reduced sampling.

We show in the results that for both unconstrained and constrained videos we can reduce the resolution by a factor of 144, skip up to 15 frames, and sample the optical flow in a 7x7 pixel grid. For constrained videos we can process just 4% of the frames (skip 15 frames and sample 60% of those) over the whole video. This combination of techniques reduces the computation time to less than three minutes.

6. IMPLEMENTATION

Our implementation of C-14 consists of two parts: a method for capturing videos and metadata from flights, and a system for verifying that videos match the intended motion program. We gathered videos from two sources: *(i)* public videos of drone flights with flight plans and *(ii)* an implementation of a motion program using the DJI SDK [2]. At publication we will make all of the source code of the drone program and verifier public, as well as links to all of the data sets we use in the evaluation.

6.1 Unconstrained Videos

To collect unconstrained motion programs and videos we take advantage of a popular drone flight planning application (Litchi) which allows users to publicly post their flight plans on the Litchi website and the resulting video on YouTube. Using videos from third parties helps eliminate potential bias in data collection and gives us access to a large variety of scenes (rural, nature, cities, etc.), lighting conditions, and DJI drone models.

A disadvantage of this data set is that we do not have the metadata (GPS, heading, speed, etc.) so we do not know when in the video the drone executed each movement (such as flying around a waypoint). In a deployed C-14 system we would have access to the metadata as it would be submitted to the verifier with the video. However, by using the camera motion results obtained from a full video, we can manually label when each motion starts.

The Litchi flight plan only contains the waypoints, heading, and gimbal position. To translate this into a motion in the ideal drone frame of reference, we first obtain the drone poses in the world frame of reference using Virtual Litchi Mission [6]. The drone poses are a list of timestamps describing the longitude, latitude, altitude, heading, tilt and roll of the camera. From those poses, the motions in the ideal drone frame of reference can be derived. By definition, there is no tilt and no roll in the ideal drone frame of reference. The yaw corresponds to the difference of heading 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.

$$t_{(y_{t,t+1})} = altitude_{(t+1)} - altitude_{(t)}$$

To compute the translations 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 vectors to match the normalized translation vectors coming from the camera motion estimator.

6.2 Constrained Videos

To collect constrained videos we implemented the C-14 motion program using the DJI Mobile SDK, which is compatible with many DJI drones. We implemented the motion program using DJI's mission control API that uses a mobile device to program the drone with a series of mission elements (waypoint, hotpoint, etc.).

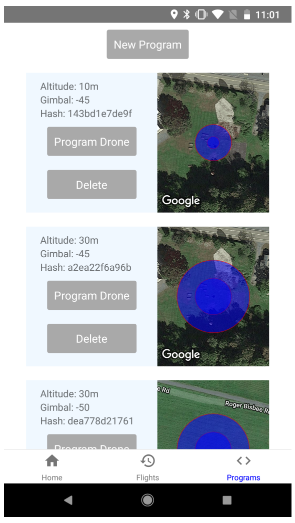
One disadvantage of using the mission control API is that transitions between motions (such as a hotpoint to a fly in and out motion) have a multi-second pause. Future systems could eliminate this pause by more directly controlling the drone through the virtual stick [3].

Our interface to the DJI SDK was built into a React Native application using a wrapper for the DJI SDK [5]. We contributed a number of changes to the library to implement capabilities required for C-14. Our C-14 program runs on Android and connects to the drone via WiFi and the DJI remote controller. The app, shown in Figure 6 has functions to create new motion programs for a particular location chosen on a map and run the program on the drone.

While working on the system we discovered two issues: the hotpoint missions in the DJI SDK do not accurately fly the given number of degrees, instead flying too many or too few degrees by many 10s of degrees. As C-14 depends on the flight to be accurate, we re-implemented the hotpoint mission to measure the GPS location of the drone and stop just short of the required number of degrees. Nonetheless as we show in Section 7.4 our implementation cannot perfectly fly the target number of degrees, but is typically within 10 degrees.

6.3 Verifier

The verifier consists of several stages: computation reduction, segmentation, and motion estimation & verification. We implemented the verifier in Python. It invokes an implementation of the optical flow estimator, PWC-Net, optimized for GPUs [27]. We use a Matlab implementation of the camera motion estimator [13] obtained from the authors.



Altitude: 10m
Gimbal: -45
Hash: 143bd1e7de9f

Altitude: 30m
Gimbal: -45
Hash: a2ea22f6a96b

Altitude: 30m
Gimbal: -50
Hash: dea778d21761

Figure 6: This figure shows a screen from the constrained video application running on an Android device.

6.3.1 *Computation Reduction*

As described in Section 5.6 we compress the videos and reduce the frame rate to speed up verification. For compression we use OpenCV's resize() with INTER_AREA interpolation. To reduce the frame rate, we maintain one frame among every s frames. For example, if the skip rate is 5, we only maintain the frame whose indexes are the multiples of 5, i.e. frame #0, #5, #10.

6.3.2 *Segmentation*

Using the metadata, the verifier divides each video into segments, one for each expected motion in the program (fly-in, fly-out, hotpoint, etc). We do not allow gaps between the motions as it would allow an attacker to possibly modify the results. For instance, an attacker could say that a hotpoint or turn started later than it does in the video which could reduce the yaw value to match a target. Using the metadata we separate the video into segments: for unconstrained videos this is the time between waypoints, and for constrained videos it is each fly in, fly out, and hotpoint motion.

6.3.3 *Motion Estimation & Verification*

After segmentation, the verifier checks if each segment conforms to its corresponding motion as described in Section 5. For constrained videos we only sample parts of the motions (temporal sampling) as described in Section 5.6. The evaluation section shows how many samples are needed to achieve good results. The verifier computes the optical flow with spatial sampling for all of the samples first and then computes the camera estimation. If any of the segments fail the verifier fails.

As discussed in Section 4.2 the verifier requires knowing the FOV of the drone camera—it is specified as a ratio of the focal length to the sensor width. For all of the videos we use a parameter of 0.82, which we measured from our DJI Mavic Air. However, we did not change this parameter for the unconstrained videos because we do not know the model of the drone used. Fortunately, the system is relatively insensitive to the FOV parameter because most drones have a similar camera.

Also, we must add some amount of tolerance to the verifier to account for inaccuracies in the camera motion estimation process and sampling noise. The angle tolerance is measured in the evaluation section. During a hotpoint we check that 60% of the samples have the correct Y-axis sign. For the fly in and out, we check that the integral of the yaw is no greater than 15 degrees.

7. EVALUATION

To evaluate C-14 we collected two data sets: unconstrained videos taken from Litchi and constrained videos taken using our custom drone program. We downloaded 10 unconstrained flight plans and videos from Litchi and YouTube respectively. The videos with links to the flight plans are shown in Figure 7 and were collected internationally with a variety of drones. We also collected eight unconstrained videos ourselves. For constrained videos we collected videos from a number of different settings, at different altitudes, and different radius and number of motions, shown in Figure 8. We collected the constrained videos using a DJI Mavic Air drone. This is a relatively inexpensive drone ($900USD) with a three-axis gimbal, a 4K UHD (3840x2160) 30 fps camera, and capable of speeds of 30 km/h in obstacle avoidance mode.

Name	FlightID	Length/Motions
Church	j6uTc0Qvha	3:27/13
Vegetation	mQUaT4UHLA	3:01/16
Small town	bNqRdjSFmo	3:44/10
Garden	qBV1h0veQ2	3:46/15
Village	bHg7fNYTSW	3:27/13
Ruins	u6UgiEDrp5	4:54/10
School	cSBHZth7L2	2:22/7
Ice hockey	mVCQVhqy0c	1:47/6
Tree	IFaQrqidsy	2:08/8
Ice rink	k7e4Hmi9Gm	1:06/7
House(x8)	withheld	approx. 0:56/8

Figure 7: This table lists the 18 flights used in our unconstrained data set. We did not create these flights and videos (except the House video). The flight plans can be obtained from http://flylitchi.com/hub?m=FlightID. The videos can be obtained from the Litchi links by double clicking on the yellow "eyeball" icon. We will provide a full data set upon publication.

We ran the experiments on a cluster with various CPUs and GPUs. All of the timing experiments were completed on a machine with two Xeon E5-2620 v3 2.40 GHz CPUs, 256G of RAM, and an NVIDIA TITAN X GPU.

Name	Altitude(m)	Radius	Length(s)/Motions
Forest Road	55	15/30	88/2
Creek	55	15/30	86/2
House A	55	15/30	119/2
Roadway(x2)	40	15/30	94,102/2
House B(x2)	80	20/40	91,106/2
Barn	40	15/30	78/2
Soccer Field	30	15/30	95/2
Farm Field	30	15/30	100/2
Snow House(x2)	30	15/30	101,106/2
Library(x3)	40	15/30	93,190,184/4
School(x4)	40	15/30	171,196,189,173/4
River(x3)	55	15/30	176,156,157/4
RecCenter(x2)	30	15/30	177,176/4
Parking lot(x2)	30	15/30	216,212/4

Figure 8: This lists 26 flights in our constrained data set with their altitude, the radius of the inner and outer circles, the length of the video in seconds, and number of individual motions(fly in/out+hotpoint). Roadway and House B were at the same location with the same program. The remaining multiple videos used different programs.

7.1 Resizing and Spatial Sampling

All of the sampling methods create a trade-off between *(i)* the accuracy of the camera motion estimator and *(ii)* the run time of the system. To evaluate the accuracy of the camera motion estimator, we use a constrained video and it's associated metadata, which records the drone's location via GPS, as well as it's heading via an onboard compass, and speed. We compute the Mean Squared Error (MSE) of the drone's yaw at each frame between the camera motion estimation and the metadata. The results are shown in Figure 9.

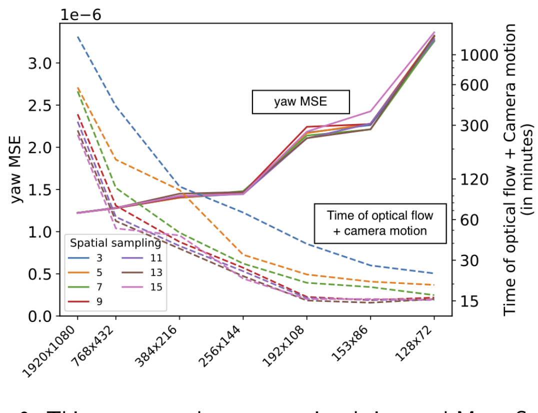


Figure 9: This compares the computational time and Mean Squared Error of the yaw between the metadata and the camera motion estimator for different video resolutions and different spatial sampling. Each data point is the average of three runs on a 2-minute video.

This shows that the computational time increases with the resolution of the video, while the error decreases exponentially. Using these results as a guide, we choose to resize the videos to 320x180 (a factor of 144 fewer pixels) for the rest of the results in the paper.

Similarly, we evaluate the impact of the camera motion spatial sampling. As shown in Figure 9, all the spatial sampling values seem to give very similar accuracy results. Based on measuring the impact on the runtime of camera motion (not shown) we choose a spatial sampling of 7 as beyond that the motion estimator does not run appreciably faster.

7.2 Frame Rate

We collected videos at 30 fps, however accurate optical flow does not require all of the frames and fewer frames means less processing time. Using the same video as before we compute the MSE between the the camera motion estimator and the metadata. The results are shown in Figure 10.

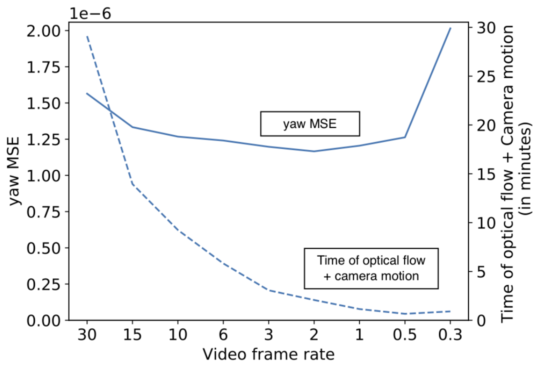


Figure 10: This shows the comparison of the yaw Mean Squared Error and the computational time for different frame rates. We used 320x180 video and a spatial sampling of 7. Each data point is the average of three runs on a 2-minute video.

In addition to computational time benefits, skipping frames reduces the MSE up to a point. This is because skipping frames smooths the motion between sample points by measuring a larger motion. However, once the system skips too many frames optical flow has a harder time making pixel correspondences between frames and the MSE increases. Based on these we results we choose to use a frame rate of 2 fps.

7.3 Sampling Rate

In constrained videos we additionally use temporal sampling to reduce the amount of computation required. To find a reasonable rate we sample a portion of the frames (after reducing the frame rate), and compute the error between the yaw estimation from the camera motion estimator and the motion program target angle. A histogram of the results is shown in Figure 11. The results show that below 60% sampling the yaw error grows outside of the bounds of [-20,20]. Thus we fix the sampling rate at 60%.

7.4 Yaw Error

A key component of verifying videos is checking the total yaw of the drone during a hotpoint motion in constrained videos, or between waypoints in unconstrained ones, matches the motion program. Even for valid videos there will be a difference between the estimate computed from the video and the target angles due to three sources: *(i)* GPS/Compass errors in the drone, *(ii)* imperfect control of the drone, and *(iii)* error introduced by optical flow and camera motion. We discount *(i)* as GPS/Compass units typically used in drones have a clear view of the sky and give location and heading accuracy of 1-3 meters and 0.3 degrees [7]. To measure *(ii)* we compare the target angles to the angle flown by the drone according to GPS. We measure *(iii)* by comparing the estimate from camera motion to the GPS reading. We measure the total error by comparing the estimate from camera motion to the target angle. We use the hotpoint motions (except House) from the dataset in Figure 7 (104 motions) and Figure 8 (38 motions) and plot the error in Figure 12.

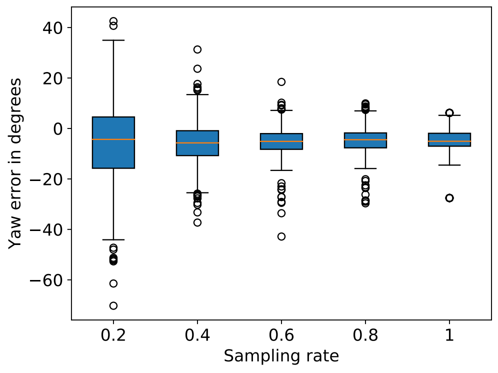


Figure 11: This shows the yaw error for various sampling rates. For a sampling rate larger than 60%, the yaw errors remain in the same range of values. A smaller sampling rate increases the yaw error.

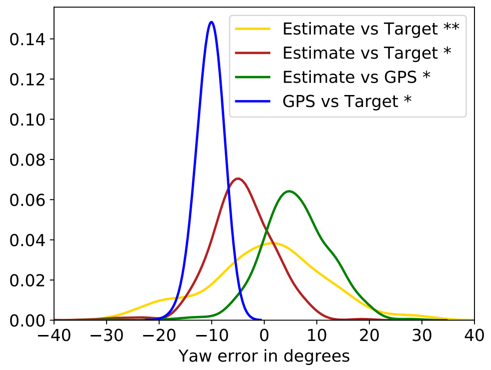


Figure 12: (*: constrained, **: unconstrained) This is the distribution of yaw error for constrained and unconstrained videos. For the constrained videos, we decompose the error in the camera motion estimation error (Estimate vs GPS) and the drone control error (GPS vs target).

The results show that the drone is able to fly to an angle within 10 degrees of the target as measured by GPS/Compass. Nonetheless, the drone under-flies the target angle consistently, which we hope to improve in future implementations. When comparing the estimate of the angle from camera motion to the GPS angle, we see that it overestimates what the drone flew—these two errors partially cancel each other out when comparing the overall constrained and unconstrained results to the target angles. Overall, thresholds on the order of [-20, 20] for constrained videos and [-30,30] for unconstrained will ensure that a high percentage of yaw angles will be classified as correct.

Looking further into the data (Figures 11 and 12) we find that all of the errors in the constrained and unconstrained videos that fall outside of a threshold of 20 degrees are due to a single hotpoint in five different videos: Forest Road, two River videos, one Rec-Center, and one School video. We believe that optical flow has trouble making accurate pixel correspondences due to the textures in those videos (leafless trees in the first three and lots of snow in the fourth and fifth). However, if we change the frame rate of these videos to 3 frames per second and 80% sampling, the error decreases dramatically (and well within 20 degrees). In a deployed system we can rerun any detected negatives with higher sampling rates, frame rates, and lower compression to truly verify negatives and eliminate such a case, and we apply this technique in the next section.

7.5 False Negative Rate

We evaluate the false negative rate with different thresholds in both the constrained and unconstrained settings. As detailed in Section 4.3, a false negative occurs when a video is created using a motion program or flight plan and it is rejected by the verifier. For constrained videos we sample at various rates, processing each video 10 times as the sampling is random. We plot the resulting false negative rate in Figure 13 for various yaw thresholds and sampling rates. The results show that for constrained videos we can achieve a 0 false negative rate at a sampling rate of 60% a frame rate of 2fps (with the exception of the five videos mentioned previously at 80% sampling and 3fps), and a threshold of 20 degrees for yaw.

Verifying unconstrained videos does not use sampling and is therefore deterministic. We plot the false negative rate in Figure 14 using various thresholds for both yaw and translation vectors. With a threshold of 30 degrees for the rotations and a threshold of 35 degrees for the translations, the system achieves a 0 false negative rate.

7.6 False Positive Rate

As detailed in Section 4.3, a false positive occurs when a video matches some other random seed R and time t_b. For constrained videos we confirm that given the motion programs in the dataset in Figure 8, each only matches the video it was used for. However, given the size of the dataset the analytical evaluation of the false positive rate for constrained videos is a better measure (see Section 4.3).

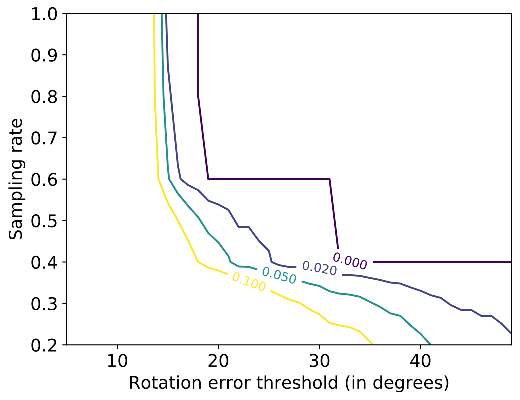


Figure 13: This shows the false negative rate with respect to the rotation error threshold and the sampling rate.

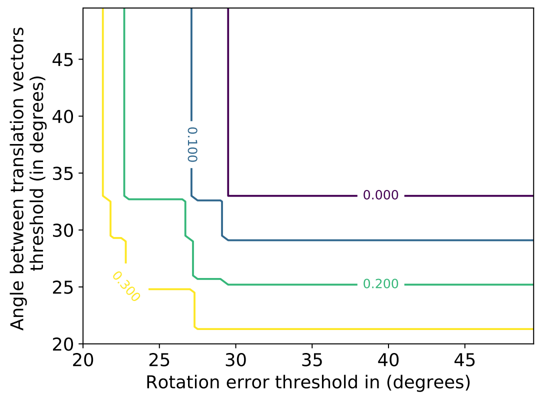


Figure 14: This figure shows the false negative rate at a particular rotation threshold and translation threshold.

For unconstrained videos a false positive will only occur if a video from some flight plan A matches some other random flight plan B. We evaluate the chance of this happening in Section 4.3 as well. However, the creator of the video might not want to create a completely random flight plan, but rather modify an existing one by “enough” such that previous videos will not match the new plan. We evaluate what “enough” is by modifying one waypoint in the flight plan. We use the chosen thresholds for unconstrained video i.e. 30 degrees for the rotation angle and 35 degrees for the translation angle. In Figure 15 we show the false positive data points in terms of how large the maximum rotation difference between the computed angle from the camera motion and the target angle from the flight plan are. This demonstrates that modifying a single waypoint in the flight plan to change the yaw of the drone by 30 degrees or more will eliminate false positives.

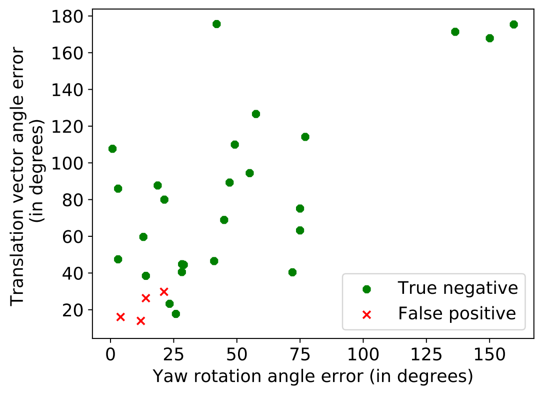


Figure 15: This figure demonstrates how much a flight plan must change to eliminate false positives. If a single waypoint is changed to produce 30 degrees more or less yaw and 35 degrees of average translation vector direction, no false positives occur.

We also confirm that subtle changes in an unconstrained flight plan creates a distinguishable video. We programmed our drone with four Litchi flight plans, approximately 340 meters of flight, and took two videos from each plan. The second flight plan differed from the first by moving just one of the waypoints by 17.5 meters. The third moved the same waypoint by 56 meters and the fourth by 63.5 meters. The second flight plan produces a video that is a false positive with the first flight plan, but the other two flight plans verify as true negatives. This shows that moving just waypoint by 50 meters was sufficient to prevent false positives.

7.7 Computational Time

We evaluate computational time using two Xeon CPUs and a TITAN X GPU. Processing a 59 second unconstrained video takes 91 seconds and a 190 second constrained video (with sampling 60%) takes 158 seconds. 60% of the time is camera motion estimation, 35% is optical flow and 5% for compression. Verifying the motion is within the tolerance takes negligible time.

7.8 Overhead

One concern is how long it takes the drone to complete the motion program. An assumption with unconstrained videos is that the motion program is part of the flight plan that would have been used anyway, so there is no overhead for incorporating C-14.

For the constrained videos there is also an assumption that the motions sequence's center point is a subject of interest, and therefore the motions are not purely overhead. However if we conservatively take the entire motion sequence as overhead the overhead is $N*(180/R+2*D/V)$, where there are N motions, the average rotation is 180 degrees, the copter completes the hotpoint at R degrees/sec and a fly in/out motion covers D meters between the outer and inner radius at V m/s. Our copter can hotpoint at approximately $R = 10 degrees/sec$ at a radius of 30m and can fly at $V = 8.3 meters/s$. So a flight with 4 motions at an outer and inner radius of 30m and 15m, will take $4*(180/10+2*15m/8.3) = 86$ seconds. However, the drone must accelerate to full speed, does not stop on a dime, and due to limitations of the DJI SDK pauses for many seconds between motions. Thus a 4 motion sequence takes approximately 3 minutes (see Figure 8), which is 15% of the Mavic Air's flight time.

8. RELATED WORK

Drones have inspired a great deal of work in mobile systems, including testbeds [8], control algorithms [15, 22], and detecting the presence of a drone [26]. However, we are unaware of any system that attempts to place a timestamp on a drone video or verify a flight pattern.

A different property is that of "liveness": the provider of a video is the original creator of a video. In two systems, Vamos [31] and Movee [32], Rahman et. al. verify a user's claim they created a video. When providing proof to a trusted third party, the user provides acceleration measurements, which can be matched to the movements in the video. In contrast we are not proving ownership, we show that a video was taken in a particular time window with particular motions.

An alternative to using optical flow is to use monocular Simultaneous Localization and Mapping (SLAM) [17,24]. SLAM algorithms estimate the drone's position while simultaneously estimating a map of the unknown 3D environment - while none of these are known beforehand. Instead, we solely estimate the drone's camera rotation and translational motion direction to verify unmodified drone videos - an easier problem.

Providing assurance for videos does depend on the videos being unaltered, specifically not spliced together from multiple videos. The more dynamic the pattern, the more difficult it is to create a convincing fake video. Much has been made recently of "deep fake" videos that change what a person is saying, such as in Face2Face [37]. New techniques can detect such alterations [23,25,39,41]. Work has also been conducted on detecting computer graphics [33]. We are not aware of such systems applied to drone videos, but once fakes emerge, we believe researchers will combat them with similar techniques.

9. CONCLUSIONS

We consider C-14 to be an important first step in opening up the problem to defenses, forensics, and anti-forensics research [14]. While it significantly raises the bar for assuring the age and flight pattern of drone videos, the basic technique can be used for other purposes, such as ensuring the quality of videos resulting from drone flights, or reconstructing a flight plan from videos with no other information.

10. REFERENCES

[1] Using Drones to Shoot War Zones. https://petapixel.com/2018/02/20/using-drones-shoot-war-zones/, 2018.

[2] DJI Mobile SDK. https://developer.dji.com/mobile-sdk/documentation/introduction/flightController_concepts.html, 2019.

[3] DJI Mobile SDK. https://developer.dji.com/mobile-sdk/documentation/introduction/component-guide-flightController.html#virtual-sticks, 2019.

[4] Litchi. https://flylitchi.com/, 2019.

[5] React Native Wrapper Library For DJI Mobile SDK. https://github.com/Aerobotics/react-native-dji-mobile, 2019.

[6] Virtual Litchi Mission. https://mavicpilots.com/threads/virtual-litchi-mission.31109/, 2019.

[7] NEO-M8 u-blox M8 concurrent GNSS modules Data Sheet. https://www.u-blox.com/sites/default/files/NEO-M8-FW3_DataSheet_%28UBX-15031086%29.pdf, 2020.

[8] Mikhail Afanasov, Alessandro Djordjevic, Feng Lui, and Luca Mottola. Flyzone: A testbed for experimenting with aerial drone applications. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 67–78. ACM, 2019.

[9] Javad Abbasi Aghamaleki and Alireza Behrad. Inter-frame video forgery detection and localization using intrinsic effects of double compression on quantization errors of video coding. *Signal Processing: Image Communication*, 47: 289–302, 2016.

[10] Jamimamul Bakas and Ruchira Naskar. A digital forensic technique for inter–frame video forgery detection based on 3d cnn. In *International Conference on Information Systems Security*, pages 304–317. Springer, 2018.

[11] Jamimamul Bakas, Ruchira Naskar, and Rahul Dixit. Detection and localization of inter-frame video forgeries based on inconsistency in correlation distribution between haralick coded frames. *Multimedia Tools and Applications*, 78(4): 4905–4935, 2019.

[12] Pia Bideau and Erik Learned-Miller. It's moving! a probabilistic model for causal motion segmentation in moving camera videos. In *European Conference on Computer Vision (ECCV)*, 2016.

[13] Pia Bideau, Aruni RoyChowdhury, Rakesh R Menon, and Erik Learned-Miller. The best of both worlds: Combining cnns and geometric constraints for hierarchical motion segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 508–517, 2018.

[14] Rainer Böhme and Matthias Kirchner. Counter-forensics: Attacking image forensics. In *Digital Image Forensics*, pages 327–366. Springer, 2013.

[15] Endri Bregu, Nicola Casamassima, Daniel Cantoni, Luca Mottola, and Kamin Whitehouse. Reactive control of autonomous drones. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 207–219. ACM, 2016.

[16] Sintayehu Dehnie, Taha Sencar, and Nasir Memon. Digital image forensics for identifying computer generated and digital camera images. In *Image Processing, 2006 IEEE International Conference on*, pages 2313–2316. IEEE, 2006.

[17] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[18] Suyog Dutt Jain, Bo Xiong, and Kristen Grauman. Fusionseg: Learning to combine motion and appearance for fully automatic segmentation of generic objects in videos. In *2017 IEEE conference on computer vision and pattern recognition (CVPR)*, pages 2117–2126. IEEE, 2017.

[19] Shan Jia, Zhengquan Xu, Hao Wang, Chunhui Feng, and Tao Wang. Coarse-to-fine copy-move forgery detection for video forensics. *IEEE Access*, 6:25323–25335, 2018.

[20] Staffy Kingra, Naveen Aggarwal, and Raahat Devender Singh. Inter-frame forgery detection in h. 264 videos using motion and brightness gradients. *Multimedia Tools and Applications*, 76(24):25767–25786, 2017.

[21] Vincent Lenders, Emmanouil Koukoumidis, Pei Zhang, and Margaret Martonosi. Location-based trust for mobile user-generated content: applications, challenges and implementations. In *Proceedings of the 9th workshop on Mobile computing systems and applications*, pages 60–64. ACM, 2008.

[22] Wenguang Mao, Zaiwei Zhang, Lili Qiu, Jian He, Yuchen Cui, and Sangki Yun. Indoor follow me drone. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 345–358. ACM, 2017.

[23] Falko Matern, Christian Riess, and Marc Stamminger. Exploiting visual artifacts to expose deepfakes and face manipulations. In *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)*, pages 83–92. IEEE, 2019.

[24] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate

monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[25] Huy H Nguyen, Junichi Yamagishi, and Isao Echizen. Capsule-forensics: Using capsule networks to detect forged images and videos. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2307–2311. IEEE, 2019.

[26] Phuc Nguyen, Hoang Truong, Mahesh Ravindranathan, Anh Nguyen, Richard Han, and Tam Vu. Matthan: Drone presence detection by identifying physical signatures in the drone's rf communication. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 211–224. ACM, 2017.

[27] Simon Niklaus. A reimplementation of PWC-Net using PyTorch. https://github.com/sniklaus/pytorch-pwc, 2018.

[28] Feng Pan, JiongBin Chen, and JiWu Huang. Discriminating between photorealistic computer graphics and natural images using fractal geometry. *Science in China Series F: Information Sciences*, 52(2):329–337, 2009.

[29] Fei Peng, Jiao-ting Li, and Min Long. Identification of natural images and computer-generated graphics based on statistical and textural features. *Journal of forensic sciences*, 60(2):435–443, 2015.

[30] Clement Pinard, Laure Chevalley, Antoine Manzanera, and David Filliat. Learning structure-from-motion from motion. In *The European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[31] Mahmudur Rahman, Mozhgan Azimpourkivi, Umut Topkara, and Bogdan Carbunar. Video liveness for citizen journalism: Attacks and defenses. *IEEE Transactions on Mobile Computing*, 16(11):3250–3263, 2017.

[32] Mahmudur Rahman, Umut Topkara, and Bogdan Carbunar. Seeing is not believing: Visual verifications through liveness analysis using mobile devices. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pages 239–248. ACM, 2013.

[33] Nicolas Rahmouni, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. Distinguishing computer graphics from natural images using convolution neural networks. In *2017 IEEE Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2017.

[34] Anurag Ranjan, Joel Janai, Andreas Geiger, and Michael J. Black. Attacking optical flow, 2019.

[35] Stefan Saroiu and Alec Wolman. Enabling new mobile applications with location proofs. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, page 3. ACM, 2009.

[36] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *CVPR*, 2018.

[37] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of rgb videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2387–2395, 2016.

[38] P. Tokmakov, K. Alahari, and C. Schmid. Learning motion patterns in videos. In *CVPR*, 2017.

[39] Weihong Wang and Hany Farid. Exposing digital forgeries in video by detecting double quantization. In *Proceedings of the 11th ACM workshop on Multimedia and security*, pages 39–48. ACM, 2009.

[40] John Wihbey. The drone revolution - uav-generated geodata drives policy innovation. *Land Lines Magazine*, pages 14–21, October 2017.

[41] Xin Yang, Yuezun Li, and Siwei Lyu. Exposing deep fakes using inconsistent head poses. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8261–8265. IEEE, 2019.

[42] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1851–1858, 2017.

[43] Zhichao Zhu and Guohong Cao. Applaus: A privacy-preserving location proof updating system for location-based services. In *2011 Proceedings IEEE INFOCOM*, pages 1889–1897. IEEE, 2011.