C-14: Assured Timestamps for Drone Videos

Zhipeng Tang
University of Massachusetts Amherst
Amherst, Massachusetts
zhipengtang@cs.umass.edu

Fabien Delattre
University of Massachusetts Amherst
Amherst, Massachusetts
fdelattre@cs.umass.edu

Pia Bideau
University of Massachusetts Amherst
Amherst, Massachusetts
pbideau@cs.umass.edu

Mark D. Corner
University of Massachusetts Amherst
Amherst, Massachusetts
mcorner@cs.umass.edu

Erik Learned-Miller
University of Massachusetts Amherst
Amherst, Massachusetts
elm@cs.umass.edu

ABSTRACT

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

We propose C-14[1], a system that assures the earliest timestamp, t_b, of drone-made videos. C-14 provides a challenge to an untrusted drone requiring it to execute a sequence of motions, called a motion program, revealed only after t_b. It then uses camera pose estimation techniques to verify the resulting video matches the challenge motion program, thus assuring the video was taken after t_b. We demonstrate the system on manually crafted programs representing a large space of possible motion programs. We also propose and evaluate an example algorithm which generates motion programs based on a seed value released after t_b. C-14 incorporates a number of compression and sampling techniques to reduce the computation required to verify videos. We can verify a 59-second video from an eight-motion, manual motion program, in 91 seconds of computation with a false positive rate of one in 10^{13} and no false negatives. We can also verify a 190-second video from an algorithmically derived, 4-motion program, in 158 seconds of computation with a false positive rate of one in one hundred thousand and no false negatives.

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies → Computer vision tasks**;

KEYWORDS

security and privacy, drone video, UAV video, video forensics

ACM Reference Format:
Zhipeng Tang, Fabien Delattre, Pia Bideau, Mark D. Corner, and Erik Learned-Miller. 2020. C-14: Assured Timestamps for Drone Videos. In *The 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20), September 21–25, 2020, London, United Kingdom.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3372224.3419196

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 videos to monitor large areas of land vulnerable to unauthorized development such as deforestation [45]. Similarly, freelance journalists are producing invaluable evidence from war zones and other difficult-to-cover areas from the air [1].

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* the videos were taken. For instance, a developer could remove trees from an area at some time, but provide older videos and claim that it was still being preserved.

A number of efforts have addressed two crucial problems: *(i)* location establishment: the video was taken at a particular place [24, 39, 48], and *(ii)* integrity assurance: the video was unaltered after recording (e.g. spliced, re-encoded, etc.) [27, 29, 37, 39, 46]. However, we are unaware of techniques that establish how *old* a video is.

We propose a system, C-14, that provides a "challenge" to an untrusted drone to assure a video was not created before some time t_b. Although we can assure the video was not created after some time t_e by verifying the time it was posted to a public forum, the video may be much *older* than that posting time. To establish that a video was taken after t_b, C-14 requires a sequence of motions, called a *motion program*, to be incorporated into a drone flight that could only be known by the untrusted drone after time t_b.

One alternative is for the drone to 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

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MobiCom '20, September 21–25, 2020, London, United Kingdom
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7085-1/20/09…$15.00
https://doi.org/10.1145/3372224.3419196

deployment. 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.

The assurance of timestamp t_b relies on the untrusted drone knowing the motion program only after t_b. There are multiple potential sources of programs. One is to derive the motions using an algorithm seeded by a public, timestamped value, R, known by the drone only after t_b. For instance, a hash taken from a blockchain published at t_b can serve as a seed. Another is to use a trusted party to reveal the motions. For instance, the trusted party can manually, or algorithmically, create the motion program and then reveal it to the untrusted drone after t_b.

The motion program is a series of motions, including translations (move up/down, left/right, forward/backwards), rotations (yaw, roll, tilt), and combinations of the two. The program can either be placed in the middle of a free-form drone flight or be used as the entire flight. We demonstrate the system on two types of programs. The first is a set of manually generated flight motion sequences. We use these manual programs to show that C-14 can be applied to a general class of patterns generated by algorithms or by hand. C-14 verifies that the drone matched the program's translation and rotation sequence. We also provide an example of one such algorithm, which generates motion programs consisting of a series of *hotpoint* motions where the drone circles a center point with its camera always pointed to the center. The algorithm produces a series of target angles, either clockwise or counterclockwise, for the untrusted drone to follow. C-14 then verifies that the drone video rotated the correct number of degrees in the correct sequence.

In order to verify that the untrusted video demonstrates the correct motion program, C-14 uses state-of-the-art techniques from computer vision to measure the optical flow of the video and derive the motion of the camera. However, such methods are typically slow, thus C-14 incorporates a number of techniques to speed verification: frame compression, skipping frames, spatial sampling, and temporal sampling. A full analysis of the video to verify the motion runs in 700x real-time (a 2 minute video takes 1 day to verify), but through compression and sampling we can reduce the amount of time needed. C-14 can verify a 59-second video from a complex, manually generated motion program with 8 motions in 91 seconds of computation with a false positive rate of 1 in 10^{13}. Using programs generated by the example algorithm, C-14 can verify a 190-second video with 4 hotpoint 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: in the most conservative sense, it uses at most 15% of the flight time.

2 TRUST AND THREAT MODEL

C-14 is split across three parties shown in Figure 1: *(i)* a trusted public source of either a number R or a motion program, and *(ii)* an untrusted drone operator who provides an untrusted video to *(iii)* a trusted verifier.

The trusted source can either provide a number R, or a full motion program as a "challenge" to an untrusted drone operator. In both cases, this source must ensure it reveals them after a timestamp t_b. This t_b then becomes the earliest time the drone can claim the resulting video was made. In the case of revealing R, this random number should be taken from a large range of values and should not be duplicated from some earlier timestamp. The trusted public source which generates R could be a distributed trust system such as an existing blockchain, or a trusted server that produces timestamped random numbers. The drone then uses that random number to seed a known algorithm that produces a motion program for the drone to follow while gathering a video. The trusted source can alternatively reveal a full motion program after t_b.

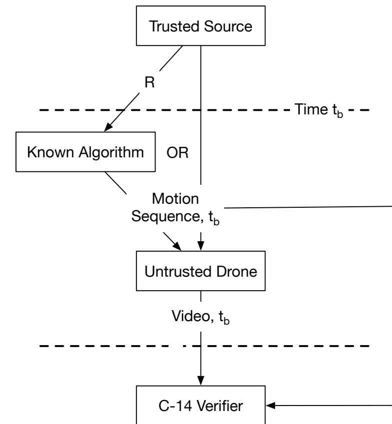


Figure 1: This figure shows an overview of the C-14 system.

The untrusted drone flies the motion program, and produces a video which it then provides to a verifier with its claim of t_b. The drone is fully untrusted and C-14 does not rely on any special hardware on the drone, such as a Trusted Platform Module. 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.

The trusted verifier then analyzes the video to determine if the claimed t_b provided with the video matches the same motion program provided by the trusted source or as derived from R using the same known algorithm. It then produces a determination which could be published with the video as a signature or other attestation.

An attacker's goal is to provide a video that is older than t_b and get C-14 to accept it as newer than t_b. To do this, the attacker would need to alter an old video to contain the motions revealed after t_b. C-14 relies on the assumption that the video provided by the untrusted drone is *unaltered*. While at first this assumption may seem unrealistic, the kinds of alterations needed to create a different motion pattern in one video to match another motion pattern are detectable. Video forensics is a field unto itself and we rely on external techniques from that community, such as:

Video Splicing (aka Frame Insertion): An attacker could take a large number of videos from a location, each of which performs one of the motions required from the authenticated video. Once the sequence of motions in the program is known, the attacker can splice together those motions into a video and present it as authentic. However, such splices can be detected in videos through a number of techniques [10–12, 21, 23]. Any such splices would need

to maintain the same illumination, texture and geometry across the splices, which would be extremely difficult.

3D Rendering/Fake Video: If an attacker can generate a 3D reconstruction of a scene, they can arbitrarily create any motion (rotation and translation) in the video. Detecting videos created from whole-cloth is outside of the scope of our work, but the image forensics community has worked on detecting such videos based on a number of techniques, for example, camera noise [17], the smoothness of images [32] and machine learning classification [33]. A similar technique would encompass such videos created by so-called "bullet-time" or 3D reconstructions from large numbers of photographs. 3D re-rendering and deep fakes are increasingly prevalent; however, to the best of our knowledge, this is still well-beyond current adversarial systems.

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 [38]. But this method does not create any specific optical flow, just a random motion pattern. We also believe that such heavy manipulation of the video will result in a very distorted video as occlusions and disocclusions that occur in real videos are difficult to fake.

Field of View Manipulation: An essential metric we use to verify the timestamp is matching the rate of rotation in the video to the motion program. Due to how camera pose estimation works, the rate of rotation can be manipulated in an existing video by cropping the video. For instance, by cropping the video to a smaller frame and scaling the frames up to the original resolution, objects will move faster across the same number of pixels thus increasing the measured rate of rotation. An attacker could possibly dynamically adjust this cropping throughout an old video to create a particular pattern of rotations. However, this dynamic scaling process will leave detectable artifacts in the video, and there have been several papers on detecting such manipulations of videos [22, 25, 40].

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 2. The drone is equipped with a front-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.

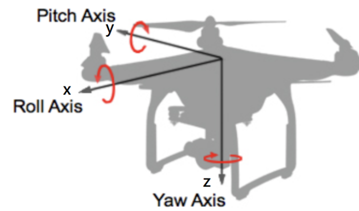


Figure 2: This figure demonstrates the axes of motion in a copter drone. Image from DJI documentation [2].

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 many ways to create such a program, including algorithmic methods and manual ones. We use a set of manual programs taken from typical flight plans to represent the wide variety of possible motion programs. We also present an example algorithmic method which uses *hotpoint* angles derived from a random seed.

In the manual programs, 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. These motion programs are flexible enough to represent various potential motion programs designed by the trusted party or generated from different algorithms.

In contrast, our example algorithmic method uses constant motions, meaning the rate of translation and rotation does not vary during the motion. For instance, if the drone translates forward, it does so without changing direction and speed, or if it yaws, it does so at a constant rate. This constant rate makes these algorithmic programs 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. Note, however, that this is one example algorithm to create motion programs. One could conceive of many algorithmic ways of creating a sequence of unique motions for a drone to follow, such as incorporating altitude changes, gimbal motions, and other complex maneuvers as represented in the manual programs.

4.1 Manual Programs

To represent a wide class of motion programs, we use manually generated sequences that are sufficiently complicated such that

guessing the program ahead of time t_b would be improbable. As long as C-14 only divulges the program to the video creator after t_b, we can assure that the video was created after that time.

By leveraging a popular drone flight planning system named Litchi [4], we can assemble a varied collection of possible flight plans, which we use as motion programs. Figure 3 shows an example flight plan—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 3: A sample manual flight plan from the Litchi flight planning software.

Such hand-planned motions can be highly-complex, and thus there are a very large number of distinct possible sequences. C-14 measures both the yaw between waypoints, as well as the 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 motion sequences 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 Example Algorithmic Programs

As an example method for automatically generating a motion program, we propose an algorithm which generates sequences of a common drone motion called a *hotpoint*. In a hotpoint motion, the drone flies in a circle while remaining pointed at a center point. This is a combination of a yaw rotation with translation to the side of the drone.

We use a hotpoint as the key motion for several reasons: *(i)* it keeps the camera centered on one subject of interest that may be needed to be part of the verified video, *(ii)* it provides us with a straightforward algorithmic way of creating the motion programs: we use a random sequence of total rotation angles as the program, *(iii)* it restricts the drone to a relatively small geographic area, which makes it straightforward to ensure the drone is flying safely, and *(iv)* in a hotpoint, the drone is translating and rotating at a constant rate throughout the motion which allows C-14 to sample at a lower rate in comparison to general programs.

In the motion program from our algorithm, the drone flies a series of motions of two types: 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 a number of degrees randomly chosen from 0 to 360 degrees along an outer radius, either clockwise or counter-clockwise. The purpose of the fly in and out 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 a circle, but with constant gimbal pitch. An example motion program is shown in Figure 4. The angle and direction of the hotpoint motion is determined by using a high-entropy number R to seed a random number generator that produces a series of random numbers $r_0, r_1, \ldots$. The motion program is a series of hotpoint motions for a certain number of degrees, as: $hotpoint_i = (angle = r_{2i}\%360, clockwise = (abs(r_{2i+1})\%2) == 1)?true : false)$.

This algorithm can generate the sequence from any single, high-entropy number, R, that would only be known after t_b. As an example, we used the block hash from a block from the Ethereum blockchain that occurs shortly after t_b, but any distributed or centralized trust could produce such a number.

We 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.

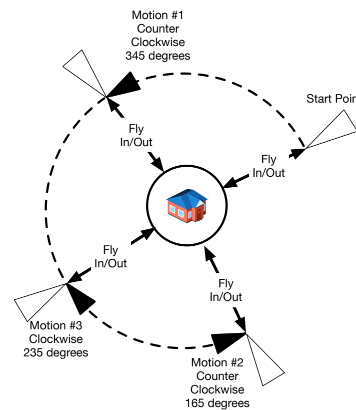


Figure 4: This figure shows the sequence of motions found in our example algorithmic motion program. The motions are a sequence of hotpoint motions and fly in/out motions, all centered on a point of interest.

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, an 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 a motion program known only after* t_b.

In our example algorithmic program, there is a series of N rotations that the video must match. Given a true rotation θ_t, a random rotation θ_r is considered correct if it is in the range $[\theta_t - rotThreshold, \theta_t + rotThreshold]$. For instance, consider a motion program with only one hotpoint motion of 100 degrees, and a threshold of 20 degrees, any video that shows a hotpoint motion of 80 to 120 degrees would be accepted as correct. 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 video with 4 rotations with a threshold of 20 degrees, that is approximately one chance in one hundred thousand.

However, we must also consider the reverse problem: if untrusted video creators can find an old motion program that matches their video they can claim the video is older than it is. For instance, if the random seed, and thus motion program, comes from a blockchain, an attacker could generate as many motion programs as there are past blocks, and find one that matches the video. One simple solution is to only accept videos within a small window after the claimed t_b, for instance, a few hours or days. This means that in the worst case the attacker would be able to claim the video is from t_b minus that window if a matching program exists in that window. Alternatively, the system can limit the number or granularity of random seeds released to video contributors. It is also relatively easy to increase the entropy of the motion in an algorithmic program. For instance, if the hotpoint motion goes up or down in altitude and moves in or out (a spiral), we decrease the probability by a factor of four and the probability of the false positive decreases to one in four hundred thousand for a program with four hotpoints.

In manual programs, the entropy of a motion program is higher as the time to execute the program is larger and it contains more motions. We also consider the direction during translations (see Section 5.4) and apply a rotation and a translation threshold to prevent false negatives. The probability for a video from a random N-waypoint flight to match the motion program becomes:

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

For a 10-waypoint program the probability for a random video to match a given flight plan using a 30° rotation threshold and a 35° translation threshold is smaller than 1 in 10^{16}.

4.4 Field of View Parameter

To measure rotations in the video, C-14 requires the *field of view* (FOV) of the camera. However, as the untrusted video creator supplies this parameter, it could attack the system by supplying a false FOV. The false FOV would scale the total amount of rotation perceived in the video. Intuitively, the attacker could choose a FOV that makes our system misclassify the first hotpoint of the video as valid. For the following hotpoints, the FOV will then be fixed, thus, scaling these hotpoints simultaneously. That brings us back to the case where there is no field of view attack with one less hotpoint.

5 VERIFIER

The goal of C-14 is to verify that the video is no older than time t_b by showing that the motion depicted in the video is consistent with a set of flight instructions given after t_b. 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 sequence dictated by the motion program.

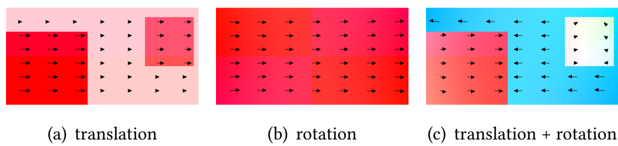
The C-14 verifier takes in an untrusted video, a timeline description of when each motion element occurred (the metadata) and the timestamp claimed with the video. The verifier first ensures that the timestamp claimed for the video, t_b, is consistent with the metadata. If that is true, then it must 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 and provide a more detailed, formal explanation in a companion document [8].

5.1 Verifying the Metadata

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 with the video. For a manually planned motion, the process is straightforward: it must check that the motion program recorded in the metadata is the one that was not revealed until after time t_b. For the algorithmically derived flights, 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 known 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 starts and the next begins. 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.



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

Figure 5: 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.

5.2 Optical Flow and Motion Estimation

To compute the drone's motion from the video, we draw on recent results in camera motion estimation, which usually has two steps: identifying correspondences among real-world points in different video frames and inferring motion from those correspondences. We chose optical flow to establish correspondences of points. Computing 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 5(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 [41], which uses a deep convolutional neural network to estimate the optical flow, which we use in our implementation.

There are other techniques that could be used to establish point correspondences instead of the optical flow technique we chose for C-14. For instance, many solutions to SLAM (simultaneous *localization* and *mapping*) use keypoint matching techniques [18, 28]. The key difference between keypoint matching and optical flow techniques is the number of matches made between frames. In keypoint methods, a subset of locations in the image (anywhere from a few dozen to hundreds or thousands of points) are matched from one frame to another. In optical flow, *every point in the image* is mapped to the next frame (unless it is occluded in the next frame). Thus, optical flow can be seen as a *dense keypoint* method.

There are trade-offs between keypoint matching and optical flow techniques. Keypoint matching is faster and works very well in many situations. But if there is motion in the image, such as waving leaves on trees, these can cause significant errors in the analysis of motion by keypoints, especially when there are relatively few keypoints identified. Because optical flow uses a much large number of correspondences, such motion in the image may be identified and removed [14]. The disadvantage of optical flow is the extra time required to compute the additional correspondences and the additional time required to process the larger number of points.

Based on our knowledge of the performance of sparse keypoint matching algorithms and dense keypoint (optical flow) methods, we chose to start with optical flow methods. However, the debate about the trade-offs between dense and sparse keypoint matching is an active area of research in computer vision. Recent results in computer vision have shown that techniques with no explicit correspondences may outperform keypoint matching [19, 47], something we plan to investigate in the future.

Once optical flow has been estimated, we use it to estimate the motion of the camera. Camera motion is decomposed into two components, rotation and translation, each of which has three dimensions. 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 5). 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 [13, 14, 34, 47]. We use a recent camera pose estimator [14] 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 motion program.

5.3 Translating Frame of Reference

Before making that match, we must translate the estimate of the camera motion 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. 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 include it in the companion document of the source code [8]. In general (such as in the manual motion programs we use from Litchi), θ is typically not constant, while in the motion program from our example algorithm, we fix θ at one value.

5.4 Manual Motion Programs

Figure 6 shows an example of the output of the motion estimator and its corresponding drone motion from a Litchi flight plan, which we use to recover the metadata as shown in Section 6.1. Both are expressed in the ideal drone frame of reference.

The verifier takes as input two sequences of motions: *(i)* the result of motion estimation, i.e. the sequence of rotations and translations between two adjacent video frames and *(ii)* a series of motions claimed in the metadata, which should be the same as the flight plan, i.e. the sequence of rotations and translations between two

adjacent waypoints. C-14 translates both into the same, ideal drone frame of reference. Before comparing them, we need to align the sequence from the motion estimator with that from the metadata, making correspondences, and interpolating between points. For each pair of adjacent waypoints, the verifier then checks the error between the two aligned sequences.

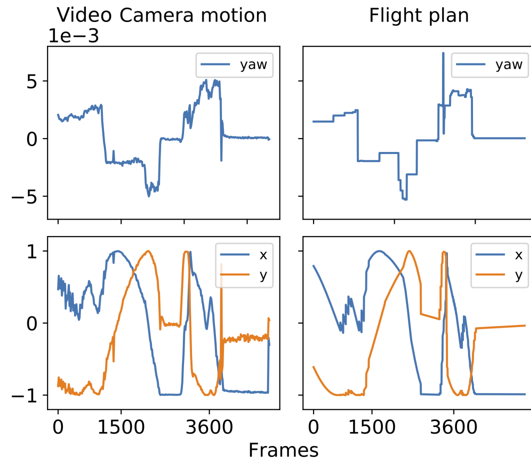


Figure 6: Comparison of the camera motion extracted from the video (left) and the Litchi flight plan (right). The camera motion has been computed from a 3-minute video at a resolution of 384*216 and a frame rate of 2 FPS.

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

Second, to check the translation, the verifier averages the translation vectors in the metadata between two waypoints and compares it to the average translation vectors between two waypoints from the motion estimation and 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 just the direction of translation expressed in the ideal drone frame of reference at their respective times.

5.5 Example Algorithmic Motion Program

Our example algorithmic motion program instructs the drone to complete several hotpoints, each of which can be decomposed into yaw and y-axis translation, if translated into the ideal drone frame of reference.

The drone travels a certain amount of yaw during each hotpoint. To compute 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 whether the total yaw (i.e. the estimated area under the curve) is within a threshold of the target value of the hotpoint, and whether a large percentage of the yaw samples are the correct sign (for a clockwise hotpoint, the yaw should be positive, while for counterclockwise, 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. Similar to yaw, a large percentage of the samples must be the correct sign.

Recall that the fly in/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 time needed to approximately 1 minute: video frame compression, frame skipping, spatial sampling, and temporal sampling. Video frame compression (aka frame resizing) simply reduces the resolution of 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 motion estimator does not need the optical flow for every pixel to infer the motion. Instead, it can sample as well. In spatial sampling, to guarantee sample pixels are spread across the frame, the estimator divides each frame into a grid and randomly chooses one pixel from each grid square. A larger grid reduces the number of samples, thus reducing computation, but with diminishing returns.

For temporal sampling, C-14 only examines the motion in short sequences of video frames (see Section 5.5). 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 manual and algorithmic motion videos we can reduce the resolution by a factor of 144, skip up to 15 frames/sec, and sample the optical flow in a 7x7 pixel grid. For algorithmic motion videos, we can process just 4% of the frames (skip 15 frames/sec and sample 60% of those) over the whole video. This combination of techniques significantly reduces the computation time.

5.7 Limitations

The system has been built and tuned around largely static scenes, such as buildings, forests, etc. Drone flights in the USA cannot easily be done legally over live subjects, which makes quantifying this limitation difficult. However, techniques to remove independent object motion from scenes [13, 14, 20, 43] could be used to increase the robustness of C-14.

We also 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.

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 that we use as motion programs and *(ii)* an implementation of our example algorithmic motion program using the DJI SDK [2]. All of the source code for the drone control program, the verifier, as well as links to all of the data sets we use in the evaluation are publicly available [8].

6.1 Manual Motion Videos

To collect manual 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 motion (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 visual correspondence between the video and the flight plan, we manually label when each motion starts.

The Litchi flight plan consists of waypoints. Each pair of waypoints has a series of drone poses generated by Virtual Litchi Mission [6] describing the instructions (including longitude, latitude, altitude, heading, and tilt and roll of the camera) the drone has to follow during the transition between these two waypoints. 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.

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

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 motion estimator.

6.2 Algorithmic Motion Videos

To collect videos based on our algorithmic motion program, we implemented the C-14 drone control program using the DJI Mobile SDK [2], which is compatible with many DJI drones. We implemented the drone control 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/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 7, has functions to create new motion programs for a particular location chosen on a map and run the program on the drone.

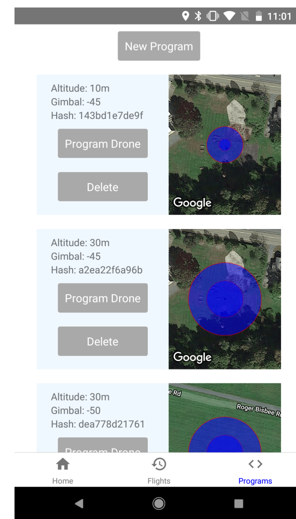


Figure 7: This figure shows a screen from the drone control application for the algorithmic motion videos running on an Android device.

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, with errors of tens 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 execute hotpoints more accurately. Nonetheless, as we show in Section 7.4, our implementation cannot perfectly fly the target number of degrees, but is typically within 20 degrees.

6.3 Verifier

The verifier consists of three stages: computation reduction, segmentation, and motion estimation & verification. We implemented the verifier in Python. It invokes an optical flow estimator, PWC-Net, optimized for GPUs [31]. We use a Matlab implementation of the motion estimator [14] obtained from the authors.

6.3.1 Computation Reduction. As described in Section 5.6, we compress the video frame resolution and reduce the frame rate to speed up verification. For frame compression, we use the function resize() with INTER_AREA interpolation from OpenCV. 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 frames 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. For manual videos, this is the time between waypoints, and for algorithmic videos, this is each fly in/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 modify the metadata to truncate a hotpoint, which could reduce the yaw value to match a target.

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 algorithmic 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 segments (or samples for algorithmic videos) first and then computes the motion estimation. If any of the segments fail, the verification fails.

As discussed in Section 4.2, the verifier requires knowing the FOV of the drone camera to approximate the ratio of focal length to the sensor width. The ratio can be used to derive the focal length in pixels, an essential parameter for motion estimator. For all of the videos, we use a ratio of 0.82 as derived in the companion document [8]. We use the same value for the manual motion videos as we do not know the model of the drone used—fortunately, most drones have a similar FOV. Nonetheless, we believe using this "typical" value for the FOV likely leads to some additional error, leading to a slightly larger threshold for these videos.

Also, we must add some amount of tolerance to the verifier to account for inaccuracies in the drone control program, camera motion estimation process and sampling noise. The angle tolerance is measured in the evaluation section.

7 EVALUATION

To evaluate C-14, we collected two data sets: manually planned flight videos taken from Litchi and algorithmically planned flight videos taken using our custom drone program. We downloaded 10 manual flight plans and videos from Litchi and YouTube respectively. The videos with links to the flight plans are shown in Table 1 and were collected internationally with a variety of drones. We also collected eight manually planned videos ourselves. For our example algorithm, we collected videos from a number of different settings, at different altitudes, and different radius and number of motions, shown in Table 2. We collected the videos using a DJI Mavic Air drone. This is a relatively inexpensive drone ($900 USD) with a three-axis gimbal, a 4K UHD (3840x2160) 30 fps camera, and capable of speeds of 30 km/h in obstacle avoidance mode.

The videos we used are either largely static or contain some limited motion—the Ice Rink video contains hockey players and many of the videos have trees with leaves moving due to wind. However, the motion does not comprise a large portion of the video frames, something that would complicate estimating camera motion from optical flow. Unfortunately, US FAA regulations prohibit flying drones over people or crowds so we were unable to gather videos with large amounts of motion.

Table 1: This table lists the 18 flights used in our manually planned 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 provide a full data set in the project webpage [8].

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

Table 2: This table lists 26 flights in our algorithmic motion 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.

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

We ran the experiments on a cluster with various CPUs and GPUs. All of the timing experiments were completed on a machine with Xeon E5-2620 v3 2.40 GHz CPU with 6 cores (only 2 cores are used), 8G of RAM, and an NVIDIA TITAN X GPU.

7.1 Resizing and Spatial Sampling

Frame resizing, frame skipping, and spatial sampling 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 three algorithmic motion videos and their 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 yaw errors of the drone at each frame between the camera motion estimate and the metadata. The results are shown in Figure 8.

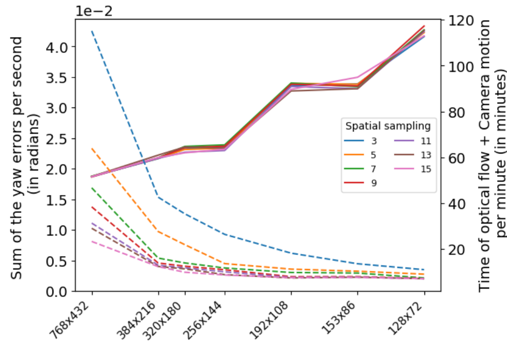


Figure 8: This figure compares the computational time and the yaw errors between the metadata and the camera motion estimator for different video resolutions and different spatial sampling. Each data point is the average of three runs on a set of three videos.

This figure 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 compared to 4K UHD videos) for the rest of the results in the paper.

Similarly, we evaluate the impact of spatial sampling. As shown in Figure 8, all the spatial sampling values 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 set of videos as before, we compute the yaw error between the the camera motion estimate and the metadata. The results are shown in Figure 9.

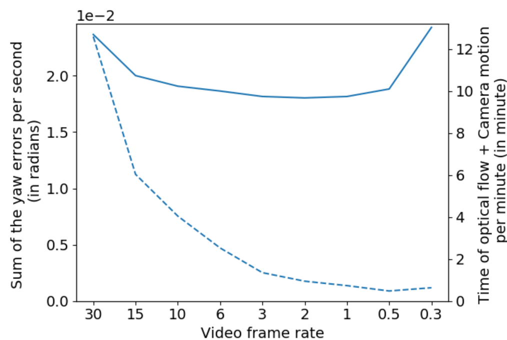


Figure 9: This figure shows the comparison of the yaw errors and the computational time for different frame rates. We used 320x180 resolution and a spatial sampling of 7. Each data point is the average of three runs on a set of videos.

In addition to computational time benefits, skipping frames reduces the yaw errors 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 yaw error increases. Based on these results, we choose to use a frame rate of 2 fps.

7.3 Sampling Rate

In algorithmic planned 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 estimate from the camera motion estimator and the motion program target angle. A histogram of the results is shown in Figure 10. 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%.

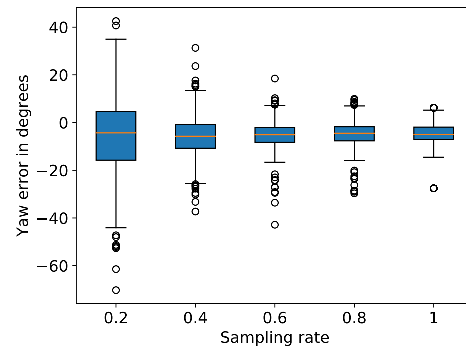


Figure 10: This figure shows the yaw error for various sampling rates. For a sampling rate larger than 60%, the yaw errors remain in the same range of values. A smaller sampling rate increases the yaw error.

7.4 Yaw Error

A key component of verifying videos is checking the total yaw of the drone during a hotpoint motion in our algorithmic flight videos, or between waypoints in manually planned 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 estimator. 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 motion estimator 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 Table 1 (104 motions) and Table 2 (38 motions) and plot the error in Figure 11.

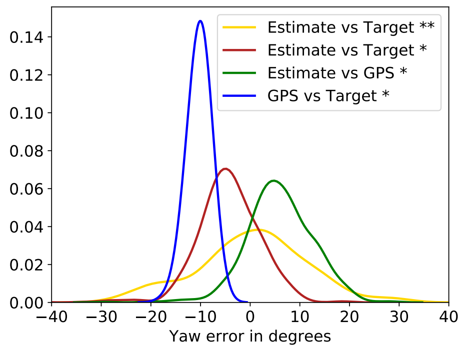


Figure 11: (*: algorithmic, **: manual) This figure shows the distribution of yaw error for algorithmic and manual flights. For the algorithmic ones, we decompose the error into the the motion estimator error (Estimate vs GPS) and the drone control error (GPS vs target).

The results show that the drone is able to fly to an angle within 20 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 results to the target angles. Overall, thresholds on the order of [-20, 20] for algorithmic flight videos and [-30,30] for manual flight videos will ensure that a high percentage of yaw angles will be classified as correct.

Looking further into the data (Figures 10 and 11), we find that all of the errors 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 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 subsection.

7.5 False Negative Rate

We evaluate the false negative rate with different thresholds in both the algorithmic and manually planned settings. As detailed in Section 4.3, a false negative occurs when a video is created using a legal motion program or flight plan but is still rejected by the verifier. For algorithmic flight videos, we sample at various rates, processing each video 10 times as temporal sampling is random. We plot the resulting false negative rate in Figure 12 for various yaw thresholds and sampling rates. The results show that for algorithmic flight 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 the yaw error.

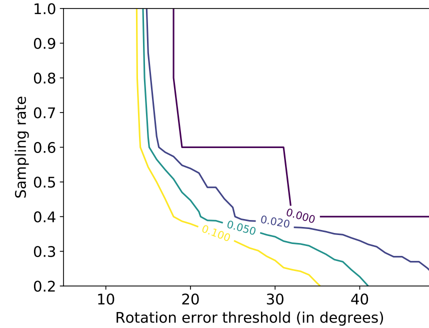


Figure 12: This figure shows the false negative rate with respect to the rotation error threshold and the sampling rate for algorithmic flights.

Verifying manually planned flight videos does not use temporal sampling and is therefore deterministic. We plot the false negative rate in Figure 13 using various thresholds for both the yaw error and the translation vector deviation. With a threshold of 30 degrees for the yaw rotations and a threshold of 35 degrees for the translations, the system achieves a 0 false negative rate.

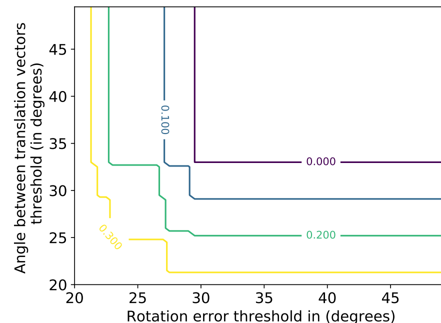


Figure 13: This figure shows the false negative rate at a particular rotation threshold and translation threshold for manual flights.

7.6 False Positive Rate

For algorithmic flight videos, as detailed in Section 4.3, a false positive occurs when a video matches the motion program derived from some other seed *R*. We confirm that each motion programs in the dataset in Table 2, only matches its corresponding video. However, given the size of the dataset, the analytical evaluation of the false positive rate for algorithmically planned videos is a better measure (see Section 4.3).

For manually planned flight videos, a false positive will only occur if the motions found in a video from one flight plan matches the motions in a different flight plan. We evaluate the chance of this happening in Section 4.3 as well. However, the entity who provides the challenge to the untrusted drone operator 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 manually planned flight video, i.e. 30 degrees for the rotation angle and 35 degrees for the translation angle. In Figure 14, we show the false positive data points in terms of how large the maximum rotation and translation difference between the computed angle from the motion estimator and the target angle from the flight plan are. This demonstrates that modifying a single waypoint in the flight plan to either change the rotation or translation direction by 30 and 35 degrees or more respectively will eliminate false positives.

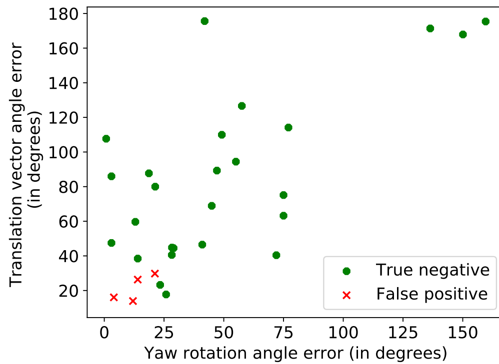


Figure 14: This figure demonstrates how much a flight plan must change to eliminate false positives. If a single waypoint is changed to produce 30 degrees more or less yaw and 35 degrees of average translation vector direction deviation, no false positives occur.

We also confirm that subtle changes in a 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 waypoint by just 50 meters was sufficient to prevent false positives.

7.7 Computational Time

We evaluate computational time using two cores of a Xeon CPU and a TITAN X GPU. Processing a 59 second manually planned flight video takes 91 seconds and a 190 second algorithmic planned 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. We did use a high-end machine with an expensive GPU for benchmarking, but the computational resources needed for verification will continue to decline in price. While cameras are increasing in resolution, C-14 can operate on a relatively low resolution version of the video, thus we expect verification to become less expensive over time.

7.8 Overhead

One concern is how long it takes the drone to complete the motion program. An assumption with manually planned flight programs 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 algorithmically planned flights, there is an assumption that the motion sequence's center point is a subject of interest, thus the motions are not pure 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 Table 2), 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 [9], control algorithms [16, 26], and detecting the presence of a drone [30]. However, we are unaware of any system that attempts to place a timestamp on a drone video.

A different property is that of "liveness": the provider of a video is the original creator of a video. In two systems, Vamos [35] and Movee [36], 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 show that a video was taken in a particular time window, instead of proving ownership.

Providing assurance for videos does depend on the videos being unaltered, specifically not spliced together from multiple videos. The more dynamic the pattern is, the more difficult it is to create a convincing fake. Much has been made recently of "deep fake" videos that change what a person is saying, such as in Face2Face [42]. New techniques can detect such alterations [27, 29, 44, 46]. Work has also been conducted on detecting computer graphics [37]. We are not aware of 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 [15]. While it significantly raises the bar for assuring the age 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.

ACKNOWLEDGMENTS

The authors would like to thank to the anonymous Shepherd and reviewers, who provided numerous suggestions on this paper. The authors also would like to thank to Massachusetts Technology Collaborative, which funds the cluster C-14 used for computation.

REFERENCES

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

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

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

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

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

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

[7] 2020. 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.

[8] 2020. Project website. https://github.com/zptang1210/C-14.

[9] Mikhail Afanasov, Alessandro Djordjevic, Feng Lui, and Luca Mottola. 2019. FlyZone: A Testbed for Experimenting with Aerial Drone Applications. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 67–78.

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

[11] Jamimamul Bakas and Ruchira Naskar. 2018. A Digital Forensic Technique for Inter–Frame Video Forgery Detection Based on 3D CNN. In *International Conference on Information Systems Security*. Springer, 304–317.

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

[13] Pia Bideau and Erik Learned-Miller. 2016. It's Moving! A Probabilistic Model for Causal Motion Segmentation in Moving Camera Videos. In *European Conference on Computer Vision (ECCV)*.

[14] Pia Bideau, Aruni RoyChowdhury, Rakesh R Menon, and Erik Learned-Miller. 2018. 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*. 508–517.

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

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

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

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

[19] Ariel Gordon, Hanhan Li, Rico Jonschkowski, and Anelia Angelova. 2019. Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras. In *Proceedings of the IEEE International Conference on Computer Vision*. 8977–8986.

[20] Suyog Dutt Jain, Bo Xiong, and Kristen Grauman. 2017. 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)*. IEEE, 2117–2126.

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

[22] Rong Jin, Yanjun Qi, and Alexander Hauptmann. 2002. A probabilistic model for camera zoom detection. In *Object recognition supported by user interaction for service robots*, Vol. 3. IEEE, 859–862.

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

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

[25] Vishnu Vardhan Makkapati. 2007. Robust Camera Pan and Zoom Change Detection Using Optical Flow.

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

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

[28] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. 2015. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE transactions on robotics* 31, 5 (2015), 1147–1163.

[29] Huy H Nguyen, Junichi Yamagishi, and Isao Echizen. 2019. 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)*. IEEE, 2307–2311.

[30] Phuc Nguyen, Hoang Truong, Mahesh Ravindranathan, Anh Nguyen, Richard Han, and Tam Vu. 2017. 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*. ACM, 211–224.

[31] Simon Niklaus. 2018. A Reimplementation of PWC-Net Using PyTorch. https://github.com/sniklaus/pytorch-pwc.

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

[33] Fei Peng, Jiao-ting Li, and Min Long. 2015. Identification of Natural Images and Computer-Generated Graphics Based on Statistical and Textural Features. *Journal of forensic sciences* 60, 2 (2015), 435–443.

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

[35] Mahmudur Rahman, Mozhgan Azimpourkivi, Umut Topkara, and Bogdan Carbunar. 2017. Video Liveness for Citizen Journalism: Attacks and Defenses. *IEEE Transactions on Mobile Computing* 16, 11 (2017), 3250–3263.

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

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

[38] Anurag Ranjan, Joel Janai, Andreas Geiger, and Michael J. Black. 2019. Attacking Optical Flow. arXiv:1910.10053 [cs.CV]

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

[40] Raahat Devender Singh and Naveen Aggarwal. 2017. Detection of upscale-crop and splicing for digital video authentication. *Digital Investigation* 21 (2017), 31–52.

[41] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. 2018. PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume. In *CVPR*.

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

[43] P. Tokmakov, K. Alahari, and C. Schmid. 2017. Learning Motion Patterns in Videos. In *CVPR*.

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

[45] John Wihbey. 2017. The Drone Revolution - UAV-Generated Geodata Drives Policy Innovation. *Land Lines Magazine* (October 2017), 14–21.

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

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

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