Real-time Car and Trackside Multi-Instance Advertisement Identification and Analysis for Televised Race Broadcasts

IN PROGRESS

Eric Q. Zeng [*]
ericzeng@seas.upenn.edu

Camillo J. Taylor [†]
cjtaylor@central.cis.upenn.edu

April 26, 2020

Abstract

We seek to create a computer vision tool to analyze streaming and prerecorded TV broadcast footage of motorsports events to better understand the distribution of airtime given to motorsports teams and drivers. Due to the inability to cover the whole venue or all action at a given moment in time with a since camera angle, there are inherent biases in which teams or drivers are shown to TV audiences. This has a huge impact on the efficacy of on-car advertisements ran by nearly all professional motorsports teams which rely heavily on such advertisement revenue. Our project will seek to create a computer vision tool to identify advertiser logos on-screen from race broadcast footage. We will use traditional feature detection techniques to extract keypoints and descriptors which will be matched between a sample image and frames from a video feed. Algorithms include SIFT, SURF, and ORB in conjunction with various filtering techniques to increase sensitivity and specificity rates. Additionally, we will be able to identify and aggregate data if there are multiple instances of an advertisement in a single frame. From this tool, we will deliver a summary of statistics with relevant measures such as percentage area coverage, time on-screen, and skewness of logo projection. We aim to make this tool easily usable and flexible by making relevant parameters tunable while offering a suggested starting point for those seeking a more straightforward use case.

***K*eywords** Computer Vision · OpenCV · Motorsports · Feature Matching · Homography

1 Introduction

The world of motorsports is unique among sports in that TV coverage is very dynamic. Camera angles are rarely standardized across venues and operators need to be extremely dynamic in covering their areas of the track. Subsequently, viewers of motorsports are exposed to constantly changing camera angles as TV broadcasters decided on which sections of a race to follow. This leads to a fundamental problem with motorsports coverage: not all on-track action can be shown to viewers. It is very common to have multiple on-track battles or incidents (crashes, radio transmissions, replays) to be happening at once in a race. Typically, broadcasters focus on the incidents most relevant to the context at hand. This may materialize as a bias towards teams who are neck-to-neck in a championship fight or a driver who has had recent off-track drama in regards to their career. These biases are nonetheless present in any other modern-day broadcast sport, but the inability to show all aspects of a motorsport event at once leads to these biases being ever more apparent to viewers.

From a commercial standpoint, TV airtime is extremely important to motorsports teams. Ever since the 1960s, teams in high-class racing series like Formula One have relied heavily on commercial sponsorships on their cars to run their cost-heavy operations. Without them, teams in such high-class series would not be able to financially survive with

[*]Website: ericqzeng.github.io
[†]Project Advisor

prize winnings alone. Thus, like with any form of advertisement, the exposure of on-car advertisements is critical to a motorsports team [1]; TV airtime is perhaps the most important of all delivery channels.

It comes at no surprise then, given the bias issue with motorsports broadcasts and the financial pressure having effective on-car advertisements, that many middle to bottom pack racing teams in Formula One are frustrated with the FOM (Formula One Management) broadcasting practices that tend to favor larger, front-running teams. The disparity in prize money already puts teams at the back of the pack at a disadvantage; the lack of TV coverage only compounds such financial issues.

As such, it is necessary to evaluate the effectiveness of motorsports TV broadcasts at providing equitable coverage to all teams and drivers, regardless of any kind of external bias the broadcasting body may have. Teams that feel disadvantaged may quantitatively measure their advertisement exposure through TV coverage, and present their case to the relevant motorsports governing body.

2 Description

We propose a computer vision system to detect and measure the effectiveness of motorsports TV broadcast coverage for a given single or set of teams and/or drivers so that advertisement effectiveness may be evaluated. This system will take in video footage from a past (or real-time streaming) motorsports broadcast, as well as relevant target logos, and output statistical results. To implement such a tool, we will use OpenCV to detect trackside and on-car advertisements with feature matching. Data for detection models would come in the form of previous seasons of relevant races. Although novel machine learning models may result in better detection accuracies, they require specific training sets and may not generalize well to motorsports footage. General car detection dataset and model are aplenty, and so are common advertisement logos [2]; we would like to offer a more general tool that does not rely on pre-built assumptions of what advertisements look like or be susceptible to false positives by wrongly identifying a different, but similar, logo. This is a special issue in motorsport broadcasts as many logos may appear together in a single video frame and misclassifying them is different from, say, misclassifying the brand of car.

Additionally, we would like to identify cases where there are multiple instances of the same logo. We understand that seeing an advertisement multiple times had diminishing additive returns to effectiveness [3]. Thus, we will report when this happens through additive metrics and sample images showing where each identified instance is within the video frame.

We plan on investigating a variety of well-known feature detection algorithms that may be used for this application. This process will involve feature detection, descriptor generations, filtering, feature matching, matching filtering, homography generation, homography filtering, and measurement of performance metrics. We set up our tool's pipeline to filter down relevant elements in each step so that later steps have less computational complexity and time requires. We will aim to have our tool run in as close to real-time as possible.

Additionally, we will expose the tunable parameters of this tool to allow the end-user to adjust the analysis to their specific use case. Although all intended users will generally be advertisers analyzing broadcast footage, we would like to allow this tool to be fit to a specific sport, race, or settings if need be. Thus, we will only present a general use case for this tool in the context of motorsports and provide minimum post-computer vision analysis. We will seek to deliver the data in a more direct yet understandable form.

We will measure advertisement effectiveness as a function of time on screen, average size throughout the detection period, and aggregate skewness of the advertisement from the perspective of the viewer. This method can be applied for each continuous "shot" the logo appears in or can be calculated over all detections in a race. It also assumes that the effects of size and time on advertisement effectiveness are linear. We will also export the raw data of detections so that it can be further analyzed from a more granular level.

3 Background Related on Computer Vision Tasks

3.1 SIFT/SURF

Algorithms for computer vision and object detection that we are interested in begin in 2004 with Lowe's paper on Scale Invariant Feature Transform, or SIFT [6]. In his paper, Lowe developed a method of corner detection and matching that can perform across different scaling and rotations of feature keypoints between query and target images. For example, the smaller representation of a corner below on the right may result in several separate defections under other methods such as Harris corner detection [4] [8]. By utilizing SIFT instead, this corner would be represented the same zoomed-in or not and be able to be matched with another instance of this feature better (see Figure 1).

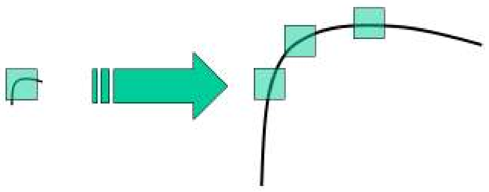

Figure 1: Why SIFT is needed. Different scaling (zoom) may result in inconsistent feature identification [4]

SIFT is able to achieve this by calculating corners and edges using Differences of Gaussians (DoG) [5]. These Gaussians blurs create a set of copies of the original image with different levels burring applied, dictated by σ values in [5] [4]:

$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi\sigma_1^2}} exp\left(-\frac{x^2 + y^2}{2\sigma_1^2}\right)$$

These DoG convolutions results in different levels of filtering applied to the image. By taking the difference, we are able to isolate out points in a middle σ layer from high-pass noise in as well as low-pass smooth areas [5]. This accentuates the corners and edges the most.

SIFT, like many feature extractors, will output two types of objects to be used for feature matching: keypoints and descriptors. Keypoints are found by [6]:

1. Calculate Differences of Gaussian blurrings with different σ values, approximating computationally heavy Laplacian of Gaussians
2. Filter out low-contrast edges and edge keypoints (we are only interested in corners)
3. Determine the orientation of each keypoint by analyzing a neighborhood around each local extrema identified.

SIFT will then calculate the descriptors for each keypoint by again looking at a “16x16 neighborhood” [4] around to be used for future feature matching. The steps of feature extraction and descriptor generation can be done separately in libraries like OpenCV but are commonly done together as many applications, like this project, are interested in feature matching.

Improvements have been made upon SIFT though since it’s creation and patent (which has expired already). Once such important improvement is Speeded Up Robust Features (SURF) developed by Bay, Tuytelaars, and Van Gool [9]. SURF improved upon nearly all steps in SIFT to provide significant speedup with little accuracy loss. It uses box filters instead of a Difference of Gaussians to approximate the Laplacian of Gaussian originally targeted in SIFT [4]. Additionally, SURF allows for skipping rotation invariance when calculating keypoint descriptors, a variant called Upright SURF (U-SURF). This offers and additional speed-up compared to normal SURF in applications that do not require matching to large rotations. We should expect that U-SURF may perform reasonably well within $\pm 15°$ of rotational difference between matched images [4].

A key insight from the OpenCV-Python documentation notes that SUFT runs “3 times faster than SIFT while performance is comparable to SIFT. SURF is good at handling images with blurring and rotation, but not good at handling viewpoint change and illumination change” [4]. This is crucial to keep in mind when comparing SIFT and SURF as we reflect on which two may be a better candidate in our application. Additionally, we should note that SIFT results in 128-dimension descriptors while SURF results in 64 dimensions [6][9]. This is beneficial for SURF in comparison to SIFT when it comes to feature matching in terms of runtime.

3.2 Alternatives and a Need for Real-time Speeds

3.2.1 FAST

Yet such a speedup may still not be good enough for real-time feature matching as broadcast footage has increased in resolution over the years. In 2006, Edward Rosten and Tom Drummond published a paper on FAST (Feature from Accelerated Segment Test) [10] which commented on the need for improvement upon SIFT to work on smaller platforms with limited computing resources. The basic principles of FAST avoid the costly Difference of Gaussian calculations or its variation in SURF. An overview of the algorithm simplified from the OpenCV-Python documentation [4]:

1. For a given pixel of interest, examine the 16 pixels around it in a circle (see 2)

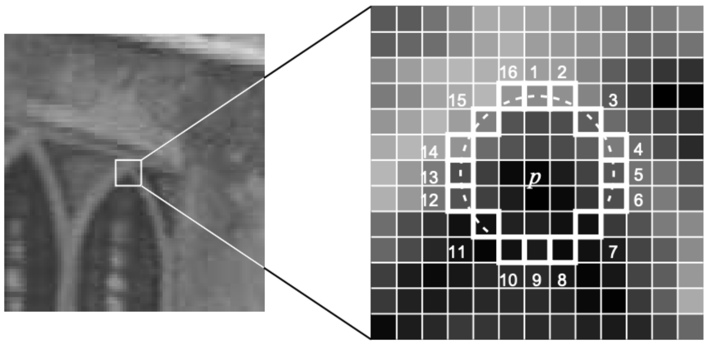

Figure 2: FAST keypoint detection by examining 16 pixels around a pixel of interest [10]

2. Set a threshold value of a minimum difference of intensity, t, and a minimum number of such neighbors, n
3. Evaluate if there are at least n contiguous pixels that exceed the threshold of t intensity compared to the center pixel, either all greater or less than in intensity

This simple approach gives FAST it's speed in identifying keypoints. This comes at a cost though as there are various issues that arise compared to SIFT/SURF; namely, FAST as presented above is sensitive to the value of n and also tends to result in many detected corners clustered together [4]. The latter issue makes intuitive sense as this algorithm would not work well in a large corner with noise, which may lead to many smaller corners being identified by FAST.

Thus, Rosten and Drummond propose (1) a machine learning speedup using ID3 decision trees to speed up corner classification, and (2) non-maximal suppression to remove redundant keypoints cluster together [10].

1. The ID3 decision tree involves taking training images or sub-samples of the image and uses the 16 pixels around each target pixel as a feature vector in training an ID3 decision tree. Once the tree is fully resolved, the tree can be used to speed up the classification of potential corner pixels for the rest of the image [4].

   To train this ID3 decision tree, we need to define entropy over set S of all test points as [11]:

   $$Entropy(S) = -p_{+}log(p_{+}) - p_{-}log(p_{-})$$

   At each stage of the tree, we will select the feature that most accurately splits the positive and negative corner identification (i.e. offers the largest Information Gain) [11]. We then have k descendants from this node, each representing one of k possible realizations of this feature. We continue this process until we are able to classify all test points using up to all 16 feature points if needed.
2. Non-maximal suppression involves summing the differences in intensity between the center pixel and all 16 surrounding pixels, for all identified corner pixels. Then, we compare a given pixel's sum with a nearby pixel's sum and remove the one with a smaller sum of the difference in intensities [4]. This fixed the overlapping/clustering keypoint problem while increasing the overall quality of each corner (each remaining keypoint will have great contrast, thus a higher likelihood to be a true corner).

Implementing FAST with improvements here results in significant speed-ps over SIFT or other corner detection methods. As mentioned in the original FAST paper, "[FAST] can fully process live PAL video using less than 7% of the available processing time. By comparison, neither the Harris detector (120%) nor the detection stage of SIFT (300%) can operate at full frame rate" [10].

3.2.2 BRIEF

FAST on its own though is just a keypoint finder; if we are to continue onto feature matching, we still need the descriptors. A key observation about SIFT descriptors is that they are floating point, requiring more space to store and computational resources to match than if they were binary descriptor [4]. Thus, BRIEF (Binary Robust Independent Elementary Features) was proposed by Calonder, et al. as a "an efficient feature point descriptor" using binary strings [12]. Binary string descriptors can be evaluated much faster using L2 Norm (Hamming distance) as it only involved examining which bits are different between two descriptors. This both decreases the size and complexity of descriptors. Additionally, it can be applied to a variety of feature extractors, like FAST, to create a much faster keypoint and descriptor generator process compared to SIFT/SURF.

3.2.3 ORB

Up until recently, both SIFT and SURF were under patents [7] that required payments for use. Thus, a free alternative was developed for OpenCV known as ORB. ORB utilized FAST corner detection and rotated BRIEF descriptors, along with several improvements from the OpenCV community, to gain large speed advantages over SIFT and SURF [4]. It, in turn, points out fewer keypoints in its analysis, keeping just the ones that are more likely to be significant. We will use ORB as a near real-time alternative to the more robust methods mentioned earlier, and examine results between it and slower, but arguable more thorough, methods like SIFT.

3.3 Comparison of Keypoint/Descriptor Algorithms

A study in 2017 compared the performance of SIFT, SURF, and ORB in various matching situations to see what algorithm would be best suited to what kinds of image transformations [13]. It found that, for their test data, SIFT was generally the slowest yet best performing across nearly all situations such as rotation, intensity, sheering, noise, and distortion; crucially, they noted that in the case of image scaling, ORB was nearly ten times faster than SIFT and had nearly a 13 percentages point advantage over SIFT in proportion of positive matches [13]. They also note that "In ORB, the features are mostly concentrated in objects at the center of the image while in SURF, SIFT and FAST key point detectors are distributed over the image" [13].

This is important given that we expect that most transformations between our query image and video frame will involve scaling. We also expect some degree of rotation, sheering, and intensity differences although not as often or as intense as scale differences. Thus we may propose the following guideline:

1. If we are interested in trackside advertisements, we should utilize SIFT as it searches evenly throughout the image for features. Trackside ads almost always appear near the edges of the screen, making ORB a poor choice.
2. If we are interested in on-car advertisements, we should ORB as it runs faster and focuses on find features near the center of the image.

We will test if this hypothesis is true by giving our tool several different query images and race feeds to analysis; measurement methods are discussed below.

4 Feature Matching Workflow

After identifying features and creating descriptors using one, or a combination, of the above methods, we can move onto comparing still images to feature match. This is the core of our tool as here we decide which features in one image match up with features in another, if at all. We will run the above keypoint and descriptor generators once for our query image logo, and every time for each frame of our target video feed.

4.1 Filtering Out Noise

We suspect that our query images (images of the logo of interest) and target video frames (race footage) will be quite noisy in the sense that there will be lots of features found in unimportant parts of the video. Thus, it is necessary to cut down on the details in our query images and video files. There are some simple heuristics we can use:

i) **Analyze only part of a video file**
Taking a glance at typical F1 race footage, we see that much of the display is taken up by on-screen metrics of the car positions, timings, and charts. These areas can all be ignored or cropped out in our input video files. This prepossessing can be done in common video editing tools like Adobe Premiere Pro, or cropped in as video frames are being pulled during analysis. The former may be a better option simply for performance reasons. In a similar vein, we may also elect to preprocess videos to be grayscale so that we do not need to convert in real-time. This may lead to issues if the grayscale conversion between two colors results in approximately the same shade of gray, but can be tuned by weighing the different RGB channels accordingly to increase color contrast.

ii) **Low Pass Filters to keep likely crucial features**
Similar to how in SIFT we filter for higher contrast points, we can increase the baseline contrast of our images and/or use Gaussian blurring kernels to filter out small “dots" of noise [18]. This is a trade-off of a bit of preprocessing for each video frame but may result in a huge increase in accurate identifications and speedup of feature matching.

iii) **Movement separator (Mixture of Gaussians)**
We will investigate if a Mixture of Gaussians background subtractors may benefit us in better isolating moving and stationary objects [16]. MOG subtractors are traditionally used on stationary cameras where it assumes stationary pixels are the background. The retained background model then guides the MOG subtractor to identify which pixels are moving in the image, thus separating the foreground motion from the background [17]

4.2 Feature Matching

After extracting our features and descriptor properly, we need a way to match our query logo to frames in our video file. The simplest way is to use a Brute-Force matcher that, for each keypoint in one image, calculates the distance to all other keypoints in the other image. This is an $O(n^2)$ given that n represents the $max\{$keypoints in query image, keypoints in video frame$\}$. Thus if we are to use the BFMatcher, we are highly incentivized to implement many of the aforementioned noise filtering techniques to reduce the number of keypoints we need to compare.

In OpenCV, there are parameter inputs into how BFMatcher will calculate the distance between. Descriptors calculated using Lowe's SIFT algorithm should use L2 Norm (Euclidean) distance whereas ORB features should use Hamming distance. This is due to the binary string representation of ORB descriptors, making Hamming distance a more apt choice [4].

Alternatively, we can elect to use FLANN (Fast Library of Approximate Nearest Neighbors) [19] matching to potentially speed up our matching step. According to OpenCV, FLANN is "a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features" [4]. FLANN matching comes with own parameters [19] that may also need to be tuned for performance depending on the feature types being matched.

i) **ORB**
Since ORB feature descriptors use Hamming distance measure, the documentation recommends Locality-Sensitive Hash tables (LSH) as search indexes for FLANN matching [4]. These are hash tables that, contrary to most hash tables, seek to maximize hash collisions within each bin. This is so that the index tree can be accessed using the descriptors as keys and the best potential neighbors can be found at the same hash address. This allows for amortized access times of $O(1)$ for each search operation, offering potential speed increases over simple brute force searching over all other descriptors.
There are several parameters that may be tuned [19]:
   a) Number of hash tables
   b) Key size
   c) Multi-probe level (can be set to 0 for stand LSH)

These parameters will be tuned accordingly for models that utilize ORB+FLANN feature matching

ii) **SIFT and SURF**
Documentation for FLANN usage in OpenCV-Python recommends using kd-trees as the search index for feature matching. Kd-trees are decision trees that allow for k-dimensional classification of feature sets, suitable for large dimension features like SIFT and SURF [4]. Due to their flexibility (and arguably, simplicity) compared to LSH, they can be used for SIFT. Additionally, it does not have the constraint of only Hamming distance comparisons like LSH does [19].

4.3 Evaluating Matches

With both Brute-Force and FLANN-based matching techniques, there is a need to evaluate the proposed matches we find. We cannot simply just take all matches between our query image and a given video frame as accurate and high quality (see Figure 4). When SIFT was introduced, Lowe suggested a ratio test between the distance of the first and second-best matches for each given match to be used as a guide for filtering for good matches. This ratio, as proposed in Lowe's original paper, is 0.7 [6]. For example, if we were to do Brute-Force matching on keypoint/descriptor pairs for a given query image of an advertiser logo, we would examine the closest and second closest distance measure for each keypoint/descriptor (distance measure type is dependent on the feature extractor as mentioned above). We take the ratio of the first to the second closest and only accept matches that are below 0.7. This value comes from tuning on Lowe's part on his image dataset [6], but is a good recommended starting point for most applications; we may allow this ratio to be tuned along with all other parameters in our final tool (see Figure 3).

This ratio test is quite simple and robust in that it systematically removes "controversial points" where the closest and second closet match distances are similar. Thus, only matches where the closest distance is clearly the closest are kept,

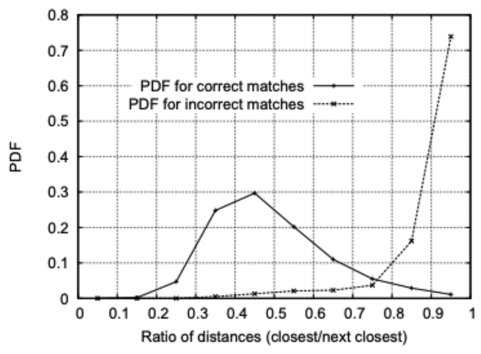

Figure 3: Histogram of distance ratios for filtering matches [6]

resulting in a smaller selection of highly confident match pairs. It also only requires the calculation of two difference of distances per match found, a linear $O(n)$ approach.

One edge case to consider though is when there is only one such candidate match for a given descriptor. In this case, there is no other match to evaluate a ratio based off of. Thus, we will simply ignore this feature match as it has no other competing match distances. We assume that this is simply noise as most keypoints will match to a few other keypoints in the other image given the sheer size of the image and video frames.

An alternative to Lowe's Ratio Test would be to implement cross-checking of best matches. This involves verifying that the best match for one descriptor is also the best match for the other descriptor [4]. This ensures that both descriptors agree on selecting each other as the best match to be paired with. This cross-check method is commonly used in place of Lowe's Ratio Test as a less computationally intensive process [4], but may be more dependent on the quality of features in both the query image and video frame. The quality of each frame in race broadcast footage is not guaranteed to be ideal as race conditions are changing and the nature of the broadcast is live. As such, we suspect that there may be more false positive matches passing through with this cross-check approach.

Regardless of the method used to evaluate feature matches, there is still a need to filter a step further before we can infer how our query image could be transformed in our video frame. Many good matches that pass Lowe's Ratio test or cross-check may still not be relevant in the context of all other good matches [4]. We could end up with many passing matches that do not relate very well to each other (i.e. they are scattered all over the place, with their relative positions seeming to have no relation to one another). In this case, we may expect that our good matches may indeed just be matching onto noise throughout our images. We need a way to not only ensure a single match is good, but also that all our passing matches make sense with respect to each other.

Since we expect there to be some skewing of the advertiser logo in the video frame, we cannot simply calculate relative distances between matched features and scale them accordingly; having rotations and other translations complicates this calculation. Thus, we will use the simple guideline of having a minimum number of matches required to continue onto the homography stage in addition to having to pass either the ratio test or cross-check [4]. This parameter may have to be tuned but can be kept relatively low (around 10 matches to start [4]).

Additionally, we note that if we are to use the ratio test with this minimum matches requirement, we should seek to find a balance between these two parameters. Both have similar purposes in our tool's workflow, but use different heuristics in principle. Their relationship will be inverse with respect to matching; more matches will be ultimately passed onto homography with a smaller minimum match requirement and larger threshold ratio.

4.4 Homography Calculation with RANSAC

Once descriptors are matched and filtered for good matches only, we can start classifying where our query image is within the video frame. This can be achieved through a homography transformation on matched keypoints. A homography transformation is a matrix operation on the (x, y) coordinates of an image to map them onto another plane through skewing, rotation, etc [30]. It is commonly used as a way to calculate perspective transformation in 3D to find an image viewed from a different camera position [21]. This technique can be used to create a matrix mapping describing how we transform our query image into what we have identified as our candidate image in a given video frame.

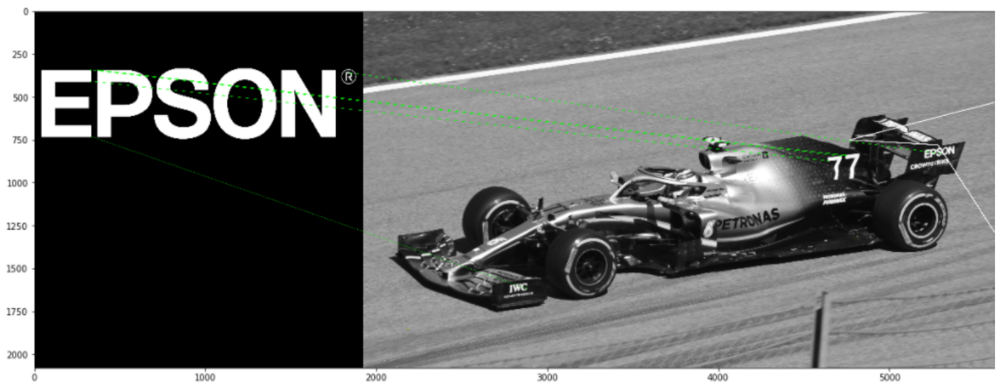

Figure 4: A Bad Match with Skewed Homography Transformation in White

Mathematically, we are interested in solving for the homography matrix H in the equation below; (x, y) is one of our features in our query image and (x', y') is the location of the matching feature in our video frame [21]:

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \text{ where } H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \tag{1}$$

This can be achieved by minimizing the least-square equation [21]:

$$min||\vec{A}H - \vec{B}||^2$$

Where $\vec{A}$ and $\vec{B}$ are our original and transformed coordinates. Our homography solution requires at least four matching points to do [21] because the four points provide eight variables, able to solve the nine variable H matrix.

OpenCV can calculate the homography matrix given the two sets of matched keypoint/descriptors [4]. We hope and expect that most true matches of our logo in the video frame will have more than four matches as that is an indicator of a true positive match. Thus, it is necessary to determine which four points to use, out of the many we could have. OpenCV can use a technique called RANSAC (Random Sample Consensus) [20] to determine the best feature points to solve for when running homography calculations [4].

In short, this algorithm will [21]:

1. Repeat
   (a) Arbitrarily select four feature corners as candidate features
   (b) Solve for the homography matrix H using these four points, and count the number of other features that are inliers (or outliers).
2. Compare the number of inliers (outliers) across all calculated homographies, and keep the H that produced the most inliers (least outliers).

A threshold parameter is used by OpenCV to determine what is or isn't an outlier. In each iteration of RANSAC, we take a point $p(x, y)$ not used in this iteration of RANSAC to calculate H, and we measure the distance from the true projected point $p_t(x_t, y_t)$ in the video frame to the homography transformation output of $p'(x', y')$ when given $p(x, y)$ [21]. If H is a perfect transformation for this feature, then our distance measure will be 0. We expect some noise so we give a threshold to allow inliers if the distance measure is within it. Mathematically, for each p, we count p as an inlier if [21]:

$$||p_t - p'|| < t$$

Once RANSAC selects the best H, our query image's actual corners will be passed through H to get the best guess corners of the query image in the video frame.

It is entirely possible though that RANSAC does not select any matrix, and an empty H is returned. This would mean that RANSAC could not find a set four points with sufficient inliers [22]. This may be due to either (1) the whole set of matches M are all outliers, thus no four matches picked could ever generate an H satisfying RANSAC or (2) there are at least four matches $\in M$ that are inliers but we were "unlucky" in that all sets of four matches picked by RANSAC did not generate an H with sufficient inliers [22][21]. The probability of (2) happening is quite low; specifically, if we take G to be the proportion of true inlier matches out of all matches and n to be the number of iterations in RANSAC, the probability of not finding a valid H is [21]:

$$(1 - G^4)^n$$

For example, if $G = 0.5$ and we ran $n = 100$ iterations, then there would be a 0.00157 chance of not finding a valid H despite our set of proposed matches truly having at least four proper matches [21]. Thus, we can safely assume any failure to output a solution for H by RANSAC is most likely due to reason (1), from which we will decide to classify this image as not having a match.

4.4.1 Filtering for Valid Homography Transformations

After calculating the homography transformation for our query image, we still need to check how sensible the projection is. Advertisers will not only want their logos to be visible but also easily recognizable; a heavily skewed representation of a query image logo is of no use. Thus, we still need to filter out strange homography shapes that are too extreme to be counted as true matches. This will reduce the number of false-positive matches onto noise and reduce the number of false-positive matches onto other similar logos.

There a few strategies we can employ to help us decide if a homography transformation is valid. The order of such homography filters will be applied in the order presented here as they go from least to most computationally intensive:

i) **Extreme Homography Coordinates**
Some homography transformed corner of our polygon may end up being either extremely high or low relative to the visible coordinate space of the video. Since most of our footage will be 720p or lower in resolution, we should expect coordinates within the 1280x720 positive x and y coordinate values. We do recognize that logos are frequently partially cropped off or obscured in the video frame as TV cameras are panning to follow on-track action and not advertisements. Thus, our matcher may report coordinates of the output homography transformation to be slightly beyond the video frame, but not too much so. Thus, we will limit the range of all propose homography coordinates to be between $-10,000 < x, y < 10,000$. This gives ample leeway to allow for the following filters to work properly while discounting a significant portion of false-positive matches caused by noise or low resolution.

ii) **Angle Measurement Minimums and Maximums**
We may take the arc cosine value of each of the four corners' cross products to determine if any corner angle is too sharp or large [23]. We expect some skewing in our video frames, but nothing too drastic. We can tune the upper and lower limits of how sharp each corner may be to better accept only reasonably shaped homography outputs. This is a very low computation cost check to add as it is simply a trigonometric function. To do this, we will use this formula. Say we are given P_1, P_2, P_3 and have the vector forms of the sides (V_{12}, V_{23}). We are interested in $\angle P_1, P_2, P_3$. the angle is [23]:

$$\theta = \begin{cases} 2\pi - arccos\left(\dfrac{V_{12} \cdot V_{23}}{|V_{12}||V_{23}|}\right) & \theta \text{ is obtuse} \\ \pi - arccos\left(\dfrac{V_{12} \cdot V_{23}}{|V_{12}||V_{23}|}\right) & \theta \text{ is acute} \end{cases}$$

We need a way to determine if the angle in question should be obtuse or acute. We should expect that we take the smaller of the two angles proposed, as it is more likely that, given we have a valid homography transformation, all angles are less than π radians. To be accurate though, we will use the sign of the cross product to drive the output of the formula above. As explained in [23], we can take the RHR (right-hand rule) by calculating the cross product sign of $V_{12} \times V_{23}$ and use that as the signed input to the arc cosine function above. We then examine the sum of all angles at the end. If the sum is 2π, then we have the correct angles with the correct assumption of acute/obtuse. Else, we take the other piece-wise option (which is just a difference of 4π) [23].

iii) **Area Minimum and Maximums**
In a similar vein, we can also quickly compute the area of the quadrilateral using the shoelace formula [24]. This involves a simple matrix operation on the coordinates of the four points. We can then filter out logos that have extremely small or extremely large areas; these extreme values represent probable false positives as we will disregard small, illegible logos and will never see a logo take up the whole video frame. Given that we have the coordinates of our quadrilateral in a list in counter-clockwise order (clockwise order for OpenCV's coordinate system [4] [24]):

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$$

We can evaluate the area using the shoelace formula [24]:

$$\frac{1}{2}\left|\sum_{i=1}^{3} x_i y_{i+1} + x_4 y_1 - \left(\sum_{i=1}^{3} x_{i+1} y_i + x_1 y_4\right)\right|$$

iv) **Intersections of a Polygon (Quadrilateral)**

We may also attempt to identify if there have been intersections between sides of our homography output. There are a few ways to do this, but the simplest way involves checking if and where two line segments intersect on a Cartesian Plane. We can achieve this by identifying the vector form of a line segment as $\vec{bc}$, and defining the orientation of a pair of endpoints a, d relative to $\vec{bc}$ [26]. This is a simple determinate operation to calculate the cross product of two vectors.

Say we have 3 points where $\vec{bc}$ is our segment of interest and we are trying to find the orientation of a relative to it.

$$a = (0, 0)$$
$$b = (4, 3)$$
$$c = (1, 2)$$

Thus, we can evaluate:

$$\begin{aligned}\vec{ab} \times \vec{ac} &= (4-0)(2-0) - (3-0)(1-0)\\ &= 8 - 3\\ &= 5 \geq 0\end{aligned}$$

We check the polarity of the cross product to determine the orientation of a with respect to $\vec{bc}$. Since it is positive, the orientation is clockwise. We repeat this process for the other endpoint d on the line segment a is an endpoint for. If both orientations are not the same, then $\vec{bc}$ "splits" $\bar{ad}$. Finally, we repeat this process for the endpoints b and c relative to $\vec{ad}$ and check that the orientation for b and c are also opposite. If that is the case, then $\bar{ad}$ splits $\bar{bc}$, and they must intersect [25].

This will need to be done for each of the six possible $\binom{4}{2}$ pairings of quadrilateral sides. Special cases involving co-linearity of the two line segments will also need to be considered [25]. Although six operations are not computationally intensive, we would like to be able to reduce the amount of work as much as possible. This because we need to do this filter for all frames where a homography transformation is found; thus, speeding up this filtering process is crucial for whichever method used.

4.4.2 Modified Bentley-Ottmann Algorithm for Line Segment Intersections

To cut down on the number of line segments comparisons we need, we will utilize the existing Bentley-Ottmann Algorithm [27] to find line segment intersections, but stop once one intersection is found. We also are not interested in saving the locations of each intersection, only if we found one.

As described in their paper, the Bentley-Ottmann algorithm allows for a fast evaluation of all intersections of line segments in a set of lines [27]. This prerequisite is clearly met by our setup of a simple, close polygon defining our homography transformation. A brief overview of the original algorithm [27]:

1. Sort all segments by their leftmost x-value
2. Add all endpoints to an Event Queue, noting down endpoints as the "Start" or "End" of a segment
3. Initialize a self-balance tree (AVL/Red-Black) to act as our "Sweep Line" that will accumulate line segments from left to right
4. Iterate over all events, add and removing line segments to an ordered Sweep Line
   For each event:
   (a) If we add a segment l, check if there is an intersection with the line above and l, as well as the line below and l, once l is in sorted order in the Sweep Line
   (b) If we remove a segment l, check if there is an intersection between the segments that were directly above and below l in the Sweep Line
   (c) If we determine there is ever an intersection, add that event to the Event Queue to be processed later

The Bentley-Ottmann Algorithm requires $O(nlg(n))$ space and time complexity to run [27] where n is the number of line segments. The complexity analysis formally has a k variable representing the number of intersections; we know that any quadrilateral will have at most one intersecting pair of line segments. Thus, $k < n^2$ and our complexity upper-bound is simplified to the above [27]. This is significantly faster than our default brute force comparison of $O(n^2)$.

To modify the algorithm to fit our application needs, we will not keep a running list of intersection points. We will still utilize a tree structure (such as AVL, Red-Black) to store line segments, sorted by the smaller x-coordinate from their two endpoints. This allows us to still perform the line sweep procedure in the increasing x-direction, and compare only those endpoints/corresponding line segments that are neighbors in y-coordinates. This is the key insight that will reduce the number of comparisons we need to do [27]. We will then utilize the intersections of line segments orientation algorithm mentioned above.

Additionally, we should note that because our line segments represent a closed polygon formed from a homography transformation, there will be at least four intersections detected the endpoints where segments connect. Therefore, we will have to modify our intersection detection algorithm to recognize and dismiss endpoint coincidence intersections.

4.5 Multiple Instances Detection

The above processes cover the required steps to detect one instance of a query image within the video frame if it truly exists and passes all tests and filters. Quite often though, we would like to detect if there are multiple instances of our query image. We can achieve this by removing the keypoints and descriptors belonging to the video frame that are used by the homography transformation from the first match. We keep the query image keypoint and descriptors the same. When we run the matcher, FLANN/Brute-Force will be blind to those descriptors from the first match. Thus, we can effectively find non-overlapping instances of the query image by repeating the whole workflow starting from the matching phase. Each additional match does require linearly more computation time, but we do not expect there to by more than a few instances per frame. Mathematically, given:

$$\begin{aligned} K &: \text{keypoints of video frame} \\ D &: \text{respective descriptors of video frame} \\ D_m &: \text{subset of } D \text{ that are inliers w.r.t valid homography} \\ K_m &: \text{corresponding keypoints to descriptors in } D_m \end{aligned}$$

We are interested in: $D \setminus D_m$ descriptors. To calculate D_m we can approximate the inlier/outlier status of each descriptor by their respective keypoint coordinates. Descriptors themselves not hold location information; thus, we need to keep K_m updated w.r.t D_m so that we know where is descriptor is. We will remove the keypoints/descriptor pairs in (K, D) whose coordinates are within the homography transformation box in the video frame.

OpenCV offers a point-in-contour checking function [29] that performs this task given the point and polygon (contour) coordinates). Applying this over all points in K and D is quite computationally intensive. Thus, we will only consider points that were passed the ratio/cross-check filter as these keypoint descriptors were the best matches; our inlier points are most likely within this subset.

(TODO) (TODO)

5 Technical Resources and Analysis Framework

We will implement our project in Python 3.7, utilizing standard libraries and others such as OpenCV [4] and sorted-containers [31]. Additionally, code snippets for simple operations, such as finding polygon areas and determining line segment intersections, have been heavily covered by open source communities; thus, their implementations will be used and credited where due [24] [25] [29] [23]. The relevant parts of the workflow under “Background Research” above are mostly all implemented with OpenCV [4]; these are represented by in blue in Figure 5. Filtering matches for high-quality ones (cross-check and Lowe’s Ratio Test), homography filtering, and multiple instance detections are a combination of known implementation available online and our own implementations; these are shown in purple above. Source code, with references, is available on GitHub[3].

Our race footage comes from community gathered video files compiled into a single BitTorrent stream [32], sourcing files through a P2P file-sharing network. There exists a formal archive for history F1 race footage, but it does not allow for downloads and is selective in its library collection [33]. Our community sourced torrent collection only has seasons

[3]https://github.com/ericqzeng/f1tvads

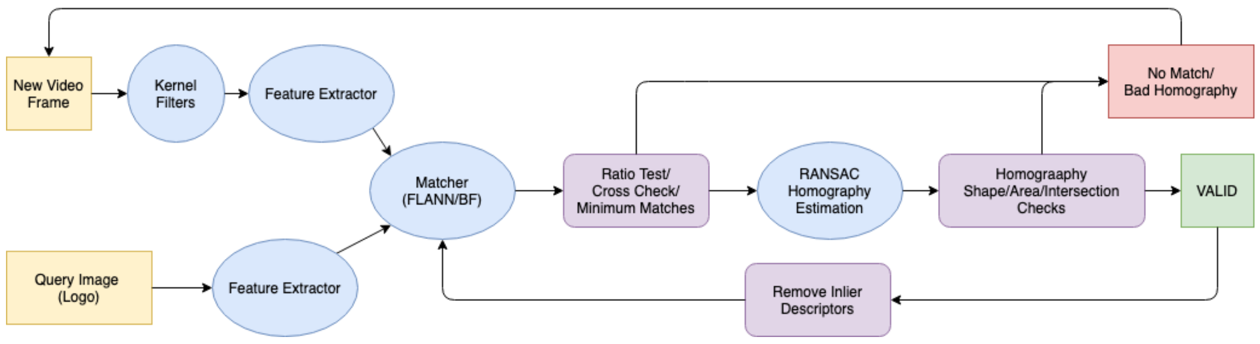
New Video Frame
Kernel Filters
Feature Extractor
Matcher (FLANN/BF)
Ratio Test/ Cross Check/ Minimum Matches
RANSAC Homography Estimation
Homograaphy Shape/Area/Intersection Checks
VALID
No Match/ Bad Homography
Query Image (Logo)
Feature Extractor
Remove Inlier Descriptors

Figure 5: General Tool Workflow for Analyzing a Query Image and Video

from 1978 to 2015, with more recent footage being higher quality (up to 720p 50fps). We will target all qualities of race footage available to investigate how resolution or frame rate sensitive our tool is.

Interfacing with our tool can be viewed as a function with a set of input parameters mentioned previously, and a set of defined output metrics:

1. **Inputs**
   Input Logo (file), Video File (file), Feature Extractor (SIFT/SURF/ORB), Matcher Frame Interval (num), Chain Length Requirement (num), Pass Ratio (num), Min/Max Area (num/num), Min/Max Angle(num/num), Filter Intersections (T/F)
2. **Outputs**
   Frame number, Total Area, Min Angle, Max Angle, Chain Weight Factor (CWF)

5.1 Data Output Format

Finally, our data on area, angles, and other performance measures will be outputted to a CSV file alongside sample images of frames with matches. This gives the advertiser a complete view of where, how long, and how large their advertisement was for the selected video frame. The sample images are highly compressed though as to save space (each usually <30KB). Thus, they are not suitable for directly running additional computer vision analytics; if more analysis is needed, the performance measures discussed below are available for that purpose.

The sample images will show valid matches that passed all tests in green; all saved images will have at least one such marking. There may be other colored boxes too signifying other potential matches that did not pass all homography checks. Red matches failed the initial area min/max check, yellow failed the second intersection check, and blue failed the final angle check. The checks are done in this order; thus, a red box only directly implies an area failure and the homography may have passed the other two. We do not check the subsequent matches in this case because we necessitate that all three be passed for a match to be counted as valid.

The performance measure can also be fed into an automated system to tune parameters of the tool if we are interested in specializing it. If we decide to iteratively tune the parameters, we need to acknowledge the risk of overfitting to a given pair of query image and race footage. This may be all right, and desired, for a single specific advertiser with one logo; we may end up tuning to perform extremely well only with this one query image. Such a use case is up to the end-user; thus, we will only tune manually and through a series of randomly sampled test query logos and test race footage clip to avoid overfitting to any one track venue or logo.

5.2 Measuring Performance

Once we have matches identified and validated, we need a way to report the actual effectiveness of the advertisement seen. Engagement is difficult to gauge especially when advertisements themselves are not the content that viewers are tuning in to focus on. Rather, we will do our best to approximate each ad's appearance effectiveness with a few, reasonable assumptions.

We can safely assume that size is critical to an ad's success; usually, engagement is positively correlated with size [34]. Thus, the size (area) will be our main measure of performance. We will simply use existing Python implementation for calculating the area of a closed polygon [24] given the four coordinates in Cartesian space.

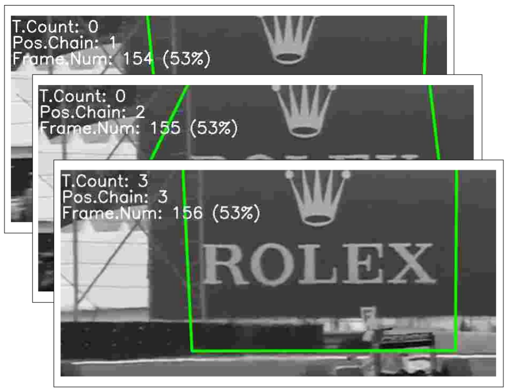

Figure 6: An example how a 3-chain requirement only records three positive frames if they are contiguous

In addition to area, we also need to consider how long the logo is present for on-screen as well as the continuity between frames where the image is found. For example, we cannot simply sum all pixel areas where a logo is identified to be across all frames of a video; that does not take into account the consistency of seeing a recognizable ad. We may achieve the same such summation measure of areas with two hypothetical situations: one where we have a few frames where the logo is large, and many frames where the logo is extremely small. Additionally, the less distorted an image is, whether that be through scale, rotation, or skewing, the more impactful that exposure is as a whole. Thus, how "upright" an image is also crucial to our method. We will address these issues here:

i) **Continuity between Successive Frames**

Having all identified instances of a logo in successive frames is more effective than having just all the frames separated from each other and spread throughout the race broadcast. Thus, we will only formally count frames as a positive "match" if we have a long enough running count of successive frames with the logo in it. For example, if we set this parameter to be 3 frames, then we wait until we see the third positive frame until we record down that these 3 frames have the logo it in. Since now we have a running record of 3, we will immediately record down a positive match for successive frames as long as there is no break in our homography matching. Once we lose sight of the logo, we restart our chain and only record until we find 3-in-row again (see Figure 6).

This parameter needs to be tuned and, potentially, may have large impacts on our project's performance. Additionally, we can extend a leeway period of frames in which we will allow our matcher to fail to find the logo between large continuous series of positive frames. We acknowledge that our homography matcher will not be perfect and we should be careful not to over discount the proposed homography matches.

An alternative to this filtering approach would be to simply count all positive frames regardless of the contextual information before and after it. Instead, we may elect to increase the measured success weight for larger continuous chunks and/or reduce the success value for individual stray frames. This would allow us to see all filtered out homography transformations but be more sensitive towards larger consecutive series. If enabled, this adjusted value is labeled as "Chain Weight Factor" (CWF).

ii) **Evaluating Transformed Logo Homography**

When measuring how upright an instance of a logo is in a video frame, we can use some of the previous work done when filtering for valid homography transformations.

We know the Cartesian coordinates of the four corners of our projected logo. We can take a look at the angle measures in each corner and determine how skewed the whole projection is, with no skewing being approximately 90 °in each corner. We can measure scale by examining the percentage of the total viewport covered by the logo; this is already calculated in our area of a polygon measure. For multiple instances, we can simply sum all valid areas as our multiple instance methodology does not allow for overlapping instances. Finally, we can measure rotation by examining the angle of the top line segment of our homography transformation quadrilateral. Regardless of the four corner angles that represent skewness, we can take the relative angle between the horizontal and this top line segment to be our overall rotation measure. We may also use the bottom segment as well with the horizontal, as well as the left or right segments compared with the vertical.

5.3 Frame Skipping during Analysis

Despite all the improvements to efficiency mentioned above, our tool will still rarely be able to run in real-time. That is, we are unlikely to be able to analyze a live broadcast video feed without a constantly increasing queue of footage. Therefore, we will give the user of our tool the ability to analyze every n frames of the video feed instead of every single one.

We accept this shortcut and argue that it does not affect the usability or principle of our tool. It is not necessary for us to know exactly in which frame a logo appeared or disappeared. Nor do we generally expect a large change in the image composition between successive frames, especially at higher frame rates. Much of the post-2012 race footage was recorded at a higher resolution and framerate (typically 1280x720 and 50fps) [32]. Given that the standard cinematic framerate is 24fps [35], we suggest that users target for 25fps in the playback footage (pre-2008 footage was standardized to 25fps [32]). Thus, for high-quality high-framerate race footage, we suggest setting n to 2.

We have considered other methods in place of frame skipping to speed up our tool without sacrificing raw frames. We could try to compress our footage or query images so that we have fewer pixels to draw features from. This process would make each individual extraction run faster, but sacrifices each individual frame's complexity instead. We would rather a few, intermittent high-quality frames over many lower quality ones.

Additionally, we recognize that the footage from our BitTorrent source is already compressed to some extent. Given their format, they are compressed using H.264 which already taken into account inter-frame lossy compression advantages [36]. This is where a majority of space gains can be made in video footage with relatively unchanging backgrounds [36]. Thus, compressing beyond this has diminished returns with respect to file size and computation complexity.

5.4 On-Screen Display and Speed Monitoring

To aid with efficiency measures when tuning parameters, we added a speed meter that reports the analysis speed of the tool reported in one second intervals. This speed meter can be seen in Figure 6 displayed as a percentage in the top left-hand corner. This speed meter takes into account the number of frames being skipped as well as the native framerate of the source footage. Thus, 100% would allow us the view the frames in real-time as the analysis works in the background, a gold-standard for live broadcast analytics.

5.5 Cloud Environment (AWS)

To speed the feature keypoint and descriptor generation process, we decided to run the whole tool in a cloud environment with greater processing power compared to a traditional laptop. We chose AWS (Amazon Web Services) for its ease of use and set-up. AWS also offers various EC2 (Elastic Computer 2) general-purpose and computer-orientated instance types, the latter of which we focused on. We started with a c5.large with two Intel Xeon Cores and 4 GB of RAM [38]; compared to our original test hardware of an early-2015 Macbook Pro 13" with a dual-core Intel laptop i5, this c5.large instance provided a substantial speed increase.

% of Real-time Speed (Positive Matches)	ORB/FLANN	SIFT/FLANN
2015 MBP (i5)	51%	6%
c5.large (Xeon)	92%	9%

Our starting c5.large instance was nearly twice as fast at ORB/FLANN feature detection and matching compared to our original testing platform. This may be due to a processor's ability to do binary XOR computations much faster than float point ones, giving and advantage to BRIEF-based descriptors like ORB [4]. Given the relatively inexpensive service offered here ($0.085/hr [37]), we will run all of our tests on AWS unless otherwise specified.

We also found that running with larger and more expensive computer-orientated instances did not over great improvements in speed (such as c5.xlarge and above). We suspect this may be due to SIFT/ORB's inability to scale with cores counts in the current implementation of OpenCV.

6 Test Results

For each set of comparison tests, we used the same set of logos and video files to control for our subject matter. We selected logos and clips that best represent the diversity of race broadcast variety that vary across logo sizes, video quality, and events during race. Our resources and parameters used are specified here:

1. **Logos**

Rolex, Fly Emirates, DHL, Formula 1 (1993-2017)

2. **Race Footage**

   2015 Brazilian GP - 5 minutes, 2008 European GP - 5 minutes

3. **Parameters**

   Chain Length: 3, Effective FPS (Source FPS/Frame Interval): 25, Pass Ratio: 0.7, Min Matches: 10, Min Area: 0.01, Filter Intersections: True, Min Angle: 45°, Max Angles: 135°

6.1 Comparisons across Feature Detectors

We compared SIFT to ORB detectors across various logos and video files. We found that ORB, on average, ran 8x as fast as SIFT as indicated by our speed meter throughout the analysis. SIFT was able to find more frames with positive matches, about 6x as many positive frames for the Rolex logo in our 5-minute clip.

Accuracy wise, both feature extractors were able to extract and correctly identify multiple logos trackside. These logos, such as the Emirates or Roles logos, are fairly large; the average size for both methods was between 13% and 26% of the frame.

SIFT was able to pick up on the small Rolex logos next to timing infographics that would occasionally appear when the broadcast would focus in on time splits between cars. This Rolex logo is completely flat with respect to the viewer, eliminating any major transformations needed. ORB was not able to pick up on these smaller instances at all.

Even in frames where both ORB and SIFT were able to find instances of logos, SIFT would often be more consistent (i.e. no breaking of the positive chain requirement) and have tighter angles and area statistics. The stand deviations (σ) for areas and angles for SIFT were 18.7 percentage points and $5.6°$; the respective σ values for ORB were 18.8 percentage points of area and $9.8°$ of angle. Thus we see SIFT offered more consistency between instances, more notably in angle measures. We can confirm this by taking a qualitative look at the sample images and noting that ORB tends to have skip frames and homographies that are more skewed when the logo should be mostly upright.

In the case of the DHL logo, ORB was unable to make any matches at all while SIFT was able to find them, albeit with a less tight homography box compared to the Rolex and Emirates logos (the σ for angle measure here was $9.9°$. Thus suggests that ORB does not do well when the (1) logo in the video frame is small and (2) when the source files have few features as is the case with the simpler block letters of the DHL logo.

(TODO)

6.2 Comparison across Feature Matches and Tuning

We ran trials with different parameters for LSH and kd-tree parameters during the matching stage of our workflow. Recommended values are used throughout this project, which can be found on the OpenCV-Python documentation guide [4]. A summary of the effects:

i) LSH: number of hash tables (default 6)

   This controlled the number of hash tables used in LSH [19]. This value worked well at the default as increasing or decreasing it had no significant impact on performance or speed.

ii) LSH: the size of the hash key (default 12)

   This controlled the size of the key for indexing in the LSH tables [19]. A longer key filtered down tangential matches so that only very close matches were ever passed onto the Lowe Ratio Test or cross-check step. Conversely, a smaller value allowed for more hash collision, resulting in a situation similar to brute-force matching. The default value worked well here while increasing it to about 15 showed slight increases in speed.

iii) kd-trees: number of trees (default 5)

   This controlled the number of trees using the kd-tree algorithm [19]. The documentation a value between [1..16], which was consistent with our finding; large values sometimes would appear to hang on certain frames [19].

We found that most of the default values suggested by the FLANN documentation and OpenCV-Python documentation worked well [4] [19]. We may elect to increase our LSH key size to speed up ORB/FLANN matching, though we risk classifying more false negatives. This is a difficult trade-off to take as ORB already has more false negatives as seen when compared to SIFT, all else equal.

6.3 Comparison across Match and Homography Filter Parameters

7 Generalization across Query Logos and Footage Quality

8 Miscellaneous Issues

8.0.1 Measuring Specificity and Sensitivity

Overall, the use of filters on matches and homography greatly reduced the false positive rate to near zero; by examining the sample images saved, nearly all images, regardless of feature or matcher type, had almost no false positive frames. This does not paint a whole picture though as we still are interested in our false-negative rate. Without manually marking each frame with a true positive/negative label, it is quite difficult to measure these statistics. We can approximate it by examining "blocks" of frames that should all be positive in the source footage, and (1) checking if a test run was able to detect any of those frames as positive and (2) how many in that block were positive.

Being able to achieve (1) is already a strong indication of a certain configuration's strength, but due to the CWF's measurement, that single (or minimum chain length) series of positive frames within the large block of true positive frames will not contribute as much weight to our final score compared to situations (2).

8.1 Small Overlapping Homographies

(TODO)

8.2 Implementation of Bentley-Ottmann Data Structures

(TODO)

8.3 Video Kernel Filters

(TODO)

9 Further Improvements

(TODO)

10 Importance

This project will provide motorsports teams around the world with a model that can be used to analyze how effective their advertisements reach TV audiences. Currently, some teams in Formula One feel that they are mistreated by the FOM in that their cars are not given the appropriate amount of TV coverage. Although it is within common reason that a more successful team will generally have more TV airtime, disregarding smaller teams results in a positive feedback loop of less and less financial resources leading to even worse relative performance in subsequent seasons [1].

Outside of advertising uses, teams can also use this project to mark when their cars are on-screen to get a visual indication of car status during a race. Modern motorsports teams have a myriad of telemetry data streaming from car to track-side engineers and even to offsite locations. Rarely though are non-sanctioned cameras allowed on cars; only officially mandated TV cameras pods (see Figure 7) and tracks-side broadcast TV cameras serve as visual aids for team engineers and strategists. Thus, knowing when the track-side broadcast TV cameras are pointing at a team's cars may be very beneficial for visually diagnosing issues.

References

[1] Mitchell, Scott. "Sainz: All midfield drivers frustrated by F1 not airing battles". September 29, 2019. https://us.motorsport.com/f1/news/sainz-midfield-battles-tv-airtime/4549776/

[2] Cambridge Spark. "50 Free Machine Learning Datasets: Self-Driving Cars." Medium, Cambridge Spark, 23 Dec. 2019, blog.cambridgespark.com/50-free-machine-learning-datasets-self-driving-cars-d37be5a96b28.

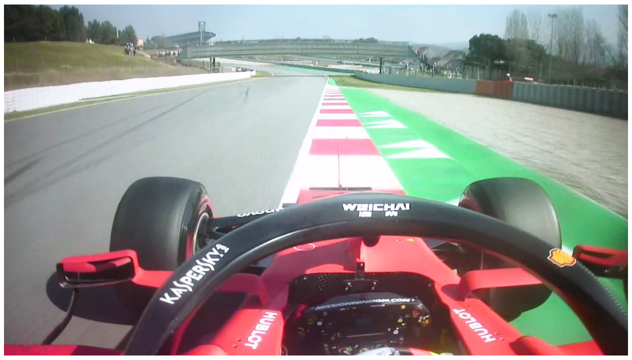

Figure 7: Typical Formula One TV pod viewport lacking a full view of the car

[3] Susanne Schmidt (Assistant Professor) & Martin Eisend (Professor) (2015) Advertising Repetition: A Meta-Analysis on Effective Frequency in Advertising, Journal of Advertising, 44:4, 415-428, DOI: 10.1080/00913367.2015.1018460

[4] "Feature Detection and Description." OpenCV, 20 Feb. 2020,

[5] Wang, Ruye. Difference of Gaussian (DoG). 12 Dec. 2012, fourier.eng.hmc.edu/e161/lectures/gradient/node9.html. docs.opencv.org/master/db/d27/tutorial_py_table_of_contents_feature2d.html

[6] Lowe, David G. "Distinctive image features from scale-invariant keypoints." International journal of computer vision 60.2 (2004): 91-110.

[7] Lowe, David G. Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image. US6711293B1, United States Patent and Trademark Office, 06 March 2000. USPTO, https://patents.google.com/patent/US6711293

[8] Harris, Christopher G., and Mike Stephens. "A combined corner and edge detector." Alvey vision conference. Vol. 15. No. 50. 1988.

[9] Bay H., Tuytelaars T., Van Gool L. (2006) SURF: Speeded Up Robust Features. In: Leonardis A., Bischof H., Pinz A. (eds) Computer Vision – ECCV 2006. ECCV 2006. Lecture Notes in Computer Science, vol 3951. Springer, Berlin, Heidelberg

[10] Rosten, Edward, and Tom Drummond. "Machine learning for high-speed corner detection." European conference on computer vision. Springer, Berlin, Heidelberg, 2006.

[11] Roth, Dan. Philadelphia, https://www.seas.upenn.edu/ cis519/fall2019/assets/lectures/lecture-2/Lecture2-DT.pptx.

[12] Calonder, Michael, et al. "Brief: Binary robust independent elementary features." European conference on computer vision. Springer, Berlin, Heidelberg, 2010.

[13] Karami, Ebrahim, Siva Prasad, and Mohamed Shehata. "Image matching using SIFT, SURF, BRIEF and ORB: performance comparison for distorted images." arXiv preprint arXiv:1710.02726 (2017).

[14] Mordvintsev, Alexander. K, Abid. "Getting Started with Videos." OpenCV, 2013, opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html.

[15] "Reading and Writing Images and Video.", OpenCV, 20 Feb. 2020, docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html

[16] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149), Fort Collins, CO, USA, 1999, pp. 246-252 Vol. 2.

[17] "MOG Background Reduction - OpenCV with Python for Image and Video Analysis 15." YouTube, Sentdex, 9 Jan. 2016, https://www.youtube.com/watch?v=8-3vl71TjDs.

[18] Mordvintsev, Alexander. K, Abid. "Smoothing Images." OpenCV, 2013, https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_mgproc/py_filtering/py_filtering.html

[19] Muja, Marius, and David Lowe. "Flann-fast library for approximate nearest neighbors user manual." Computer Science Department, University of British Columbia, Vancouver, BC, Canada (2009).

[20] M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981.

[21] MIT CSAIL. 2012. *Lecture 13: Homographies and RANSAC*. 6.869: Advances in Computer Vision. MIT, http://6.869.csail.mit.edu/fa12/lectures/lecture13ransac/lecture13ransac.pdf

[22] “Camera Calibration and 3D Reconstruction.” OpenCV,
docs.opencv.org/3.3.0/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780.

[23] Mastragostino, Robert. “How to Find the Interior Angle of an Irregular Pentagon or Polygon?” Mathematics Stack Exchange, 29 May 2012, math.stackexchange.com/questions/149959/how-to-find-the-interior-angle-of-an-irregular-pentagon-or-polygon.

[24] Semwal , Smitha Dinesh. "Area of a Polygon with given n Ordered Vertices." GeeksforGeeks, 22 June 2018, www.geeksforgeeks.org/area-of-a-polygon-with-given-n-ordered-vertices/.

[25] Riyal, Ansh. "How to Check If Two given Line Segments Intersect?" GeeksforGeeks, 24 Feb. 2020, www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/.

[26] Cormen, Thomas H., et al. Introduction to Algorithms, 3rd Edition. MIT Press and McGraw-Hill, 2009.

[27] Bentley and Ottmann, "Algorithms for Reporting and Counting Geometric Intersections," in IEEE Transactions on Computers, vol. C-28, no. 9, pp. 643-647, Sept. 1979.

[28] Sunday, Dan. “Intersections of a Set of Segments.” GeomAlgorithms, 2012, geomalgorithms.com/a09-_intersect-3.html.

[29] Mordvintsev, Alexander, and Abid K Revision. “Contours : More Functions.” OpenCV, 2013, opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_more_functions/
py_contours_more_functions.html#point-polygon-test.

[30] Taylor, Camillo J. Email Advising. March 2020.

[31] Jenks, Grant. Sorted Containers. 2019, http://www.grantjenks.com/docs/sortedcontainers

[32] F1Torrent. "Formula 1 - 1978 to 2015 Mega Torrent". BitTorrent, Reddit, 2016,
https://www.reddit.com/r/formula1/comments/3y5hv0/formula_1_1978_to_2015_mega_torrent_update/

[33] Formula 1 TV. Online Subscription and Race Archives. https://f1tv.formula1.com/

[34] “5 Charts That Prove Viewability and Audibility Together Are Key to Video Ad Effectiveness.” Google, Google, Mar. 2017, www.thinkwithgoogle.com/advertising-channels/video/effective-video-ads-viewability-audibility/.

[35] Har-Even, Benny. “Gemini Man And High Frame Rate (HFR)–Is It The New Face Of Cinema?” Forbes, Forbes Magazine, 11 Nov. 2019, www.forbes.com/sites/bennyhareven/2019/10/10/gemini-man-and-high-frame-rate-hfr-is-it-the-new-face-of-cinema#600c03182d07.

[36] “An Overview of H.264 Advanced Video Coding.” Vcodex, www.vcodex.com/an-overview-of-h264-advanced-video-coding/.

[37] “AWS EC2 On-Demand Pricing.” AWS, Amazon, aws.amazon.com/ec2/pricing/on-demand/.

[38] “AWS EC2 Instance Types.” AWS, Amazon, https://aws.amazon.com/ec2/instance-types/
(TODO, sources below may no longer be relevant?)

[39] W. Lu, J. Ting, J. J. Little and K. P. Murphy, "Learning to Track and Identify Players from Broadcast Sports Videos," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 7, pp. 1704-1716, July 2013.

[40] Chandan, G and Jain, Ayush and Jain, Harsh and Mohana, Mohana. (2018). Real Time Object Detection and Tracking Using Deep Learning and OpenCV. 1305-1308. 10.1109/ICIRCA.2018.8597266.

[41] Adrian Rosebrock, “Object detection with deep learning and OpenCV”, pyimagesearch.

[42] Ren, Shaoqing et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (2015): 1137-1149.

[43] Redmon, Joseph et al. “You Only Look Once: Unified, Real-Time Object Detection.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 779-788.

[44] Liu, Wei et al. “SSD: Single Shot MultiBox Detector.” ECCV (2016).