Science Arena Publications
Specialty Journal of Electronic and Computer Sciences
ISSN: 2412-7485
Available online at www.sciarena.com
2020, Vol, 6 (2): 1-9

Comparison of API trackers in OPENCV using Raspberry Pi hardware

Hassan Darvishi

Master of Science in Electrical-Telecommunication System, Islamic Azad University, Sirjan Branch, Sirjan, Iran.

Abstract: *OpenCV3.0 has a tracking API involving the implementation of various single-object tracking algorithms. These tracking algorithms include BOOSTING, MIL, KCF, TLD, MEDIANFLOW, MOSSE, and CSRT. As long as we observe the current position of the object, we know its motion, that is, the parameters of the object motion model. Furthermore, based on the location, velocity, and direction of the object in the prior frames, we can predict the new position of the object based on its current motion model. But we require more data. We know what the object looks like in each previous frame. In more detail, we can build an appearance model that simulates the object. This visual model can be employed to search for a small spatial area near the location predicted by the motion model to more precisely predict the position of the object. The motion model predicts the rough location of the object. This accurate calculation is provided for a more precise assessment based on the model. In this study, we first characterized these trackers and then simulated the performance of them using Raspberry Pi hardware. Plus, with the outcomes of the simulations, we compared these trackers.*

Keywords: *real-time tracking, API trackers, object model, Raspberry Pi minicomputer, obstruction*

INTRODUCTION

Now, scientific advancements have deeply affected human life. One of the most essential issues presented in telecommunications today is image processing methods for detecting, tracking, and following the moving objects. Detecting and tracking moving targets in back-to-back frames of a video stream has many applications in robotics, the military industry, security systems, and machine vision. The meaning of tracking a target is to particularize the desired area in a sequence of consecutive frames. Different hardware and software platforms are used for image processing and machine vision systems. The hardware platforms used include industrial computers, home computers, microcontrollers, mobile or tablet development boards, X86 small laptops or netbooks, X86 laptops with dGPU, FPGA hardware design, single-board computers (SBCs), System on Chip (SoC), etc. For each of these hardwares, relevant software platforms are used. Windows and Linux operating systems are among the most famous ones in this domain. A system on a chip (SoC) is an integrated circuit that involves the components of a computer on one chip. SoCs, along with electronic devices, are very prevalent for their low power consumption and versatility. SoCs are broadly used in SBC cell phones and embedded hardware. A SOC includes all the hardware and software needed for that operation. The best benefit of using an SoC is its size. If you use a CPU, creating a compact computer will be quite difficult, just because of the number of separate chips and other parts you need to settle on board. Yet, when using SoCs, you can put complete special application computing systems in smartphones and tablets. You can still have

plenty of space for batteries, antennas, and other add-ons required for telephony and data communications. Owing to the extremely high level of integration and compressed size, an SoC consumes significantly less power than a standard CPU. This is a vital advantage of SoCs when joined with mobile and portable systems. Furthermore, decreasing the number of chips by removing additional ICs on the computer board yields a compact size for the board.

Raspberry Pi is a class of low-cost SBCs in the size of credit cards developed in the UK by the Raspberry Pi Foundation. This Foundation was founded in 2009. The idea of developing Raspberry Pi was to improve rudimentary computer science education in schools and growing countries by providing a low-cost computing platform. Raspberry Pi has developed its position well beyond its planned purpose by entering the market for embedded systems and research. The Raspberry Pi Foundation published Raspberry Pi in 2012. Raspberry Pi primarily uses Linux-based Unix-like operating systems, like Debian and Fedora. Raspberry Pi models A, A +, B, and B + are based on the ARM11 class, which implements the ARM version 6 instruction set. ARM version 6 instruction set does not support Ubuntu 3 and Windows 4 operating systems. However, Raspberry Pi 2, which is capable of supporting Windows 10 and Ubuntu (Snappy Core), runs on ARM Cortex A7.

In this study, we want to implement API trackers in OPENCV on Raspberry Pi hardware using Python software. We also want to display and compare the results individually for each algorithm by tracking moving targets in real-time.

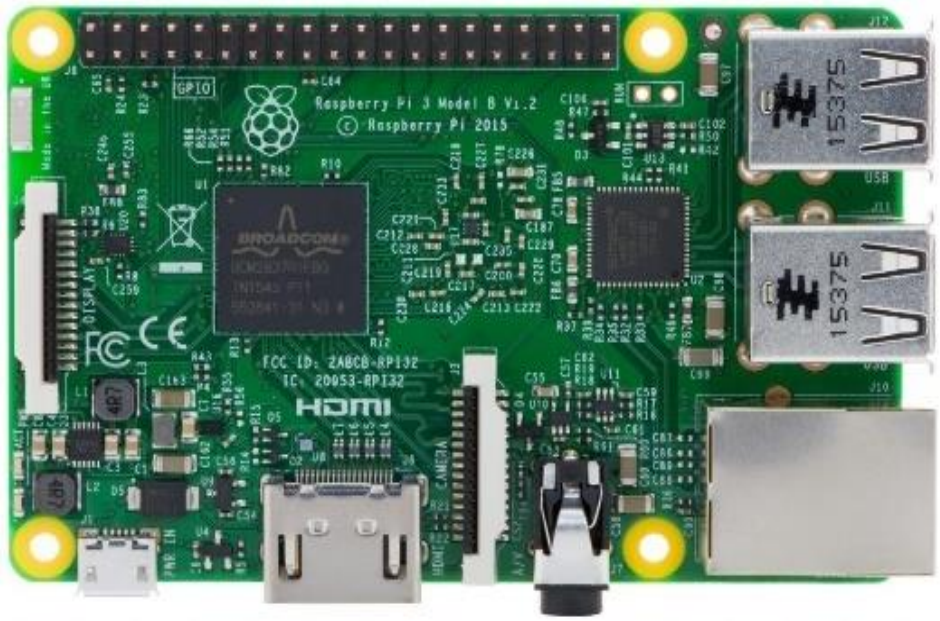

Figure 1: Picture of Raspberry Pi Model 3B used in the article

Raspberry Pi single-board minicomputer

In this article, a Raspberry Pi 3B minicomputer is employed to simulate API algorithms using Python software on the RASPBIAN operating system. The characteristics of this single-board minicomputer are listed in Table (1). Also in Figure (1), the Raspberry Pi
used in this article is displayed. Raspberry Pi has input and output pins for hardware interaction with various modules called GPIO.

API trackers in OpenCV 3.0

OpenCV3 has a new tracking API, including the implementation of various single-object tracking algorithms. These algorithms involve BOOSTING, MIL, KCF, TLD, MEDIANFLOW, MOSSE, and CSRT. As long as we

observe the current location of the object, we know its motion, that is, the parameters of the object's motion model.

Furthermore, based on the location, velocity, and direction of the object in the prior frames, we can predict the new position of the object based on its current motion model. But we require more data. We know what the object looks like in each previous frame. In more detail, we can build an appearance model that simulates the object. This visual model can be employed to search for a small spatial area near the location predicted by the motion model to more precisely predict the position of the object. The motion model predicts the rough location of the object. This accurate calculation is provided for a more precise assessment based on the model.

Table 1: Specifications of Raspberry Pi Mini Computer 3B

CPU	Broadcom BCM2837, 1.2 GHz 64-bit quad-core ARM Cortex-A53
GPU	Broadcom VideoCore IV @ 400 MHz
Memory (SDRAM)	1 GB (shared with GPU)
Onboard Storage	MicroSDHC slot, USB Boot Mode
Onboard Network	10/100 Mbit/s Ethernet, 802.11n wireless, Bluetooth 4.1
Video Input (Camera)	15-pin MIPI camera interface (CSI) connector
Video Outputs	HDMI (rev 1.3), composite video (3.5 mm TRRS jack), MIPI display interface (DSI) for raw LCD panels
Audio inputs	via I²S
Audio outputs	Analog via 3.5 mm phone jack; digital via HDMI
USB 2.0 ports	4 (via on-board 5-port USB hub)
Low-level peripherals	17× GPIO plus UART, I²C bus, SPI bus with two chip selects, I²S audio +3.3 V, +5 V, ground.
Power source	5 V via MicroUSB or GPIO header
Size	85.60 mm × 56.5 mm × 17 mm (3.370 in × 2.224 in × 0.669 in)
Supported OS	Raspbian (Debian-based Linux), Ubuntu MATE, Windows 10 IoT Core, RISC OS, KODI

If the object is very plain, and its appearance does not vary much, we can use a simple template as an appearance model. But this is not the case: the appearance of the target can change dramatically. To approach this problem, in many modern trackers, this modeling suggests an online-trained classifier whose job is to classify a rectangular section of an image as an object or background. The classifier takes a part of the image as input and sets a score of 0 or 1 to specify whether the snippet includes an object. A score of 0 suggests that the snippet contains only the background, while a score of 1 is when the snippet contains an object. A classifier is trained using positive (object) and negative (background) examples.

- **BOOSTING tracker**

This tracker is based on the online version of AdaBoost. An algorithm that practices the HAAR cascade-based face detector inside. This classification needs training at runtime with positive and negative examples of the object. The initial constraining box presented by the user (or defined by another detection algorithm) is considered a positive example for the object. Parts of the image outside the constraint box are considered as the background. When entering a new frame, the classifier reviews all the pixels in the vicinity of the previous object and stores the classification score. The new position of the object is the one that has the highest score. So now, there is a more positive example for classification. With the addition of more frames, the classification updates with this new information. One of the drawbacks of this algorithm is that its tracking performance is average, and it cannot recognize well that the tracking has failed (Grabner and Bischof, 2006).

- **MIL tracker**

The idea of this tracker is similar to the BOOSTING tracker specified above. The big distinction is that instead of just looking at the current position of the object as a positive example, there is a small area around the current location to create some potentially positive examples. At first look, this seems like a bad plan. Because in several of these positive examples, the object is not focused.

To overcome this problem, multiple instance learning (MIL) is considered. In MIL, you do not define positive and negative samples, but "positive and negative containers". A set of images in a positive basket are not positive examples. Alternatively, just one image in the positive basket should be a positive example. For illustration, a positive basket has parts of the image that are in the current location of the object and also in a tiny area around it. Even if there is no reliable tracking of the current position of the object, when samples of the current location range are in the positive basket, there is a good chance that the basket includes at least one image in which the object exists. The performance of this algorithm is quite solid, and it works well in case of partial obstruction. Impediments include complete obstruction, in which case the algorithm does not work properly (Babenko et al., 2009).

- **KCF tracker**

KCF is the abbreviation of Kernel Correlation Filters. This tracker is based on the concepts presented in the past two trackers and is formed based on the fact that various positive examples used in the MIL tracker have overlapping ranges. This overlapping data yields desirable mathematical properties that are used by this tracker to make tracking faster and more reliable at the same time. The precision and speed of this tracker are better than MIL, and the performance of its failure reports in tracking has been more agreeable than BOOSTING and MIL. One of the downsides of this tracker is that it does not recover from total obstruction (Henriques et al., 2014).

- **TLD Tracker**

TLD is the short form of Tracking, Learning, and Detection. As the name suggests, this tracker breaks down long-term tracking into three short-term ones: tracking, learning, and detection. The tracker tracks the target frame by frame. The detector classifies all items observed so far and corrects the tracker if necessary. The Learning section calculates detector errors and renews them to stop these errors in the future. The output of this detector tends to move around very little. For instance, if we track down a walker and there are other pedestrians there, this tracker may sometimes briefly track a pedestrian other than what we intended and proceed to track it quickly when the target reappears.

On the positive side, the tracker seems to be able to track the target on a larger scale, move, and blocking. If we have a video series in which the object is concealed behind another object, this tracker can be a reliable choice.

Long-term tracking can be performed by tracking or detection. Tracking algorithms measure the motion of an object. Trackers only require initialization as they are fast and provide the right paths. On the other hand, they normally make an error if the object vanishes from the camera. Tracking research intends to develop very, very strong trackers that track objects over a longer period. Detection-based algorithms determine the location of an object in each frame individually. Detectors do not disappear if the object disappears from the camera sight. Still, they need an offline training phase and, hence, cannot be applied to unknown objects.

The opening point for an investigation into the TLD algorithm is the recognition of the fact that tracking and detection cannot do long-term tracking work alone. But, if they work at the same time, each can use the other. A tracker can give tagged training data for a detector, thus, advancing its runtime performance. A detector can restart a detector, thus lessening detection errors (Kalal et al., 2011).

- **MEDIANFLOW tracker**

This tracker follows the object in both forward and backward directions in time and measures the differences between these two paths. Reducing this Forward Backward error allows it to readily detect tracking failures and pick reliable paths in video sequences.
This tracker runs best when the motion is anticipated and small. Unlike other trackers that proceed even when the tracking has clearly failed, it recognizes the failure. The tracking reports with this tracker have been great. One of the benefits of this tracker is that when motion is predictable, and there is no impediment, this tracker operates very well. One of its weaknesses is that under large and fast movements of the object, it fails and becomes ineffective (Kalal et al., 2010).

- **MOSSE tracker**

Minimum Output Sum of Squared Error (MOSSE) uses compatibility correlation for object tracking that regulates stable correlation filters when using an initial frame. MOSSE Detector is immune to changes in light, scale, appearance, and non-rigid changes. It additionally detects barriers based on the ratio of the peak to the side edge. MOSSE Detector can work at high fps (450 frames per second or even higher). Another positive characteristic is its smooth implementation and its very high speed compared to other complicated tracking algorithms. However, in terms of performance, this tracker is behind the trackers based on deep learning (Bolme et al., 2010).

- **CSRT tracker**

In the Discriminative Correlation Filter with Channel and Spatial Reliability (DCF-CSR), we use spatial reliability to set the filter support to a portion of the selected area of the frame for tracking. This expands and localizes the selected zone and tracks non-rectangular areas or objects. This tracker practices two standard features (HoGs and Colornames). It additionally works at lower frames (25 fps) but is more precise at tracking objects (LuNežič et al., 2018).

Simulation

The hardware system defined in section 2 was used to simulate the algorithms specified in this paper. These algorithms were run on standard sequences and sequences of frames. These frame-by-frame sequences include intricate and specific tracking conditions across various frames. The IoU evaluation standard was used to examine these algorithms and assess the performance and tracking results.
In this research, the code for each algorithm was run on three benchmark datasets in the tracking literature, i.e jumping, bear_front, and zcup_move_1, and the results were examined. Table (2) gives the specifications of these datasets together with the number of frames estimated and their overall condition. In the simulation, the frames are processed sequentially. The first frame is taken and the tracking target is chosen by the user by entering the coordinates of the primary constraint box or by forming a constraint box around it with the command cv2.SelectRoi (). The frames are then inserted sequentially into the algorithm. Next, the object chosen by the user in each frame is analyzed and detected and tracked by creating a restrictive box. This process lasts until the final frame. At the end of processing each frame, the analytical items of that frame are measured and presented.

Table 2: Specifications of the benchmark data set with the number of frames evaluated and its status

Sequence name	Number of frames	The camera motion	Partial obstruction	Thorough obstruction	Change of position	Similar objects	Change of environment	Change of size
Jumping	313	No	Yes	Ni	Yes	Yes	Yes	No
bear_front	295	No	Yes	Yes	Yes	No	No	No
z_cup_move1	150	Yes	No	No	Yes	Yes	Yes	No

- **IoU evaluation criteria**

To assess the performance quality of the algorithm employed in this article for tracking and detection, we used the evaluation criterion named overlapping, or Intersection over Union (IoU).
This criterion settles the precision of localization by comparing the degree of overlapping of the reference constraint box and the predicted constraint box. The value of IoU, the overlapping, is obtained from the relationship given in Figure (2). Overlap values greater than 0.5 are usually a good predictor. In reality, it appears very improbable that the (y, x) coordinates of our predicted limit box agree exactly with the (y, x) coordinates of the reference constraint box.

- **Simulation results**

In this part, the results of the implementations made to simulate the algorithm are presented individually. In each section, the result tables and images of the frames are given.

✓ **Execution of algorithms on the Jumping dataset**

This dataset includes 313 frames in JPG format with a frame size of 358 x 288 pixels from the evaluation benchmark datasets. The target tracked in this dataset is a person doing jumping ropes at high speed. This dataset in some frames includes fast motion, glitch, and fading of the tracked object. Figure (3) displays the first frame of this sequence (right image), target selection (left image), and successful tracking (bottom image) by one of the detectors, respectively.

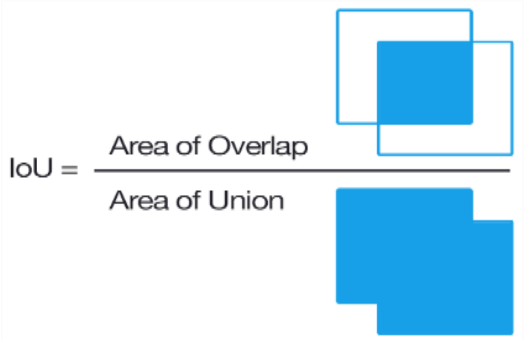

Figure 2: Calculating the amount of overlap

Figure 3: Images of the first frame (right image), target selection (left image) and successful tracking (bottom image) of the jumping data set

Table 3: Results of tracker performance evaluation on jumping data set

Tracker	Number of frames	Number of successful trackings	Number of unsuccessful trackings	Success percentage	Frame tracking per second
BOOSTING	313	35	296	11.8	6
MIL	313	313	-	100	2

KCF	313	125	188	40	16
TLD	313	270	43	86.26	3
MEDIANFLOW	313	104	209	33.2	15
MOSSE	313	110	203	35	21
CSRT	313	309	4	98.72	4

- ✓ **Execute the algorithm on the bear_front dataset**

This sequence has 295 frames in png format, and the size of each frame is 640 x 480 pixels. The striking point in this sequence is the presence of partial and total obstructions and half-done axial rotations along this sequence of frames. Frames 30 to 60 have the first partial and complete obstructions in this data set. Since the goal of implementing the detectors on this dataset in this essay was to compare the performance of the detectors in complete and partial obstruction conditions, the implementation was done on 30 to 60 frames. Table (4) shows the tracking results. From frame 41 to frame 45, the object is tracked in total obstruction. Figure (4) displays an image of pre-blocking, during-blocking, and successful post-blocking tracking in this dataset by the TLD tracker.

Figure 4: Before, during and after blocking by TLD tracker in bear_front dataset

Table 4: Results of tracker performance evaluation in 30 to 60 frames of bear_front data set

Tracker	Number of frames	Number of successful trackings	Number of unsuccessful trackings	Success percentage	Frame tracking per second
BOOSTING	31	16	15	51.61	2
MIL	31	5	26	Tracker failure	2
KCF	31	25	6	80	4
TLD	31	31	-	100	2
MEDIANFLOW	31	4	27	Tracker failure	6

MOSSE	31	4	27	Tracker failure	8
CSRT	31	3	28	Tracker failure	5

✓ **Execute the algorithm on the dataset z_cup_move1**

This sequence holds 140 frames in png format with a size of 640 x 480 pixels. In this dataset, both the object being tracked and the camera are moving. Figure (5) displays an image of the initial constraint box and the predicted constraint box in the following frames. This condition impedes the tracking process. But by nearly all trackers, the object is tracked. In the frames where the camera was moving, the tracking of all the detectors was assessed as almost successful. Table (5) exhibits the results of this tracker implementation for the frames of this dataset, which involve quick camera movement.

Figure 5: Image of the primary (left) and predicted (right) constraint boxes in the z_cup_move1 dataset

Table 5: Results of tracker performance evaluation on z_cup_move1 data set frames

Tracker	Number of frames	Number of successful trackings	Number of unsuccessful trackings	Success percentage	Frame tracking per second
BOOSTING	140	140	-	100	2
MIL	140	140	-	100	2
KCF	140	140	-	100	3
TLD	140	136	4	97.14	2
MEDIANFLOW	140	132	8	94.28	5
MOSSE	140	136	4	97.14	8
CSRT	140	140	-	100	2

Following performing the simulation and showing the performance results of the detectors for each dataset, the qualitative performance of each in various conditions like fading and swift displacement of the target, retrieval of the detector in partial and complete obstruction mode, and also with the camera moving were examined. The corresponding results are shown in Table (6).

Table 6: Investigating the performance of algorithms in different conditions according to the results

Tracker	Fading and quick movement	Retrieving of the tracker in partial obstruction	Retrieving of the tracker in complete obstruction	Moving camera
BOOSTING	Unsuccessful	Successful	Successful	Successful
MIL	Successful	Successful	Unsuccessful	Successful

KCF	Successful	Successful	Unsuccessful	Successful
TLD	Successful	Successful	Successful	Successful
MEDIANFLOW	Successful	Successful	Successful	Successful
MOSSE	Unsuccessful	Unsuccessful	Unsuccessful	Successful
CSRT	Successful	Unsuccessful	Unsuccessful	Successful

Conclusion

In this article, we first presented a brief explanation of API detectors in OPENCV3.0 and then compared them by executing this algorithm. These detectors include BOOSTING, MIL, KCF, TLD, MEDIANFLOW, MOSSE, and CSRT.

As stated, the hardware used for the simulation in this article was a Raspberry Pi Model 3B. The simulations were performed on three datasets: jumping, bear_front, and z_cup_move1. The jumping dataset presents a person doing jump ropes in which the target has rapid and sudden movements and fades at various times. The berar_front dataset comprises multiple partial and total obstructions, while the z_cup_move1 dataset includes the movement of the camera and target. The results achieved from the trackers were compared with the reference truth data of each set and illustrated as tables. According to the received results, the top tracker in each dataset was identified. Among them, the TLD detector showed the best performance in dealing with partial and complete obstructions, and the MIL detector showed the best performance in dealing with fast displacements and fading of the target. When the camera was moving, all the detectors performed well, and there were few differences.

One thing to do in this regard is to present solutions to increase the performance of detectors that have been strong in one field and have significant weaknesses in other ones. The MIL tracker, for example, is quite powerful in fast target movements but very ineffective in the event of complete obstructions. Additionally, the TLD algorithm, which is quite strong against obstruction, can be improved for more reliable performance in conditions of fading and quick movement of the target. CSRT tracker can also be a novel research subject. Because it has performed with high accuracy and speed in the event of fading and rapid displacement of the tracked object. However, this tracker failed in the case of obstruction and was not able to recover the target.

References

1. Babenko, B., Yang, M. H., & Belongie, S. (2009, June). Visual tracking with online multiple instance learning. In *2009 IEEE Conference on computer vision and Pattern Recognition* (pp. 983-990). IEEE.
2. Bolme, D. S., Beveridge, J. R., Draper, B. A., & Lui, Y. M. (2010, June). Visual object tracking using adaptive correlation filters. In *2010 IEEE computer society conference on computer vision and pattern recognition* (pp. 2544-2550). IEEE.
3. Grabner, H., & Bischof, H. (2006, June). On-line boosting and vision. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (Vol. 1, pp. 260-267). Ieee.
4. Henriques, J. F., Caseiro, R., Martins, P., & Batista, J. (2014). High-speed tracking with kernelized correlation filters. *IEEE transactions on pattern analysis and machine intelligence*, *37*(3), 583-596.
5. Kalal, Z., Mikolajczyk, K., & Matas, J. (2010, August). Forward-backward error: Automatic detection of tracking failures. In *2010 20th International Conference on Pattern Recognition* (pp. 2756-2759). IEEE.
6. Kalal, Z., Mikolajczyk, K., & Matas, J. (2011). Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, *34*(7), 1409-1422.
7. LuNežič, A., Vojíř, T., Čehovin Zajc, L., Matas, J., & Kristan, M. (2018). Discriminative Correlation Filter TracNer with Channel and Spatial Reliability. *International Journal of Computer Vision*, *126*(7), 671-688.