Robust Object Tracking with YOLO and AKAZE for SfM

M. Omer Farooq Bhatti
MS Data Science & AI
Asian Institute of Technology
Bangkok, Thailand
st122498@ait.asia

***Abstract*—This document describes a robust object tracking algorithm using YOLOv4-Tiny object detector and AKAZE features tracking. The described tracker has been used to track a car under noisy conditions and can be used to recover Pose and estimate homography.**

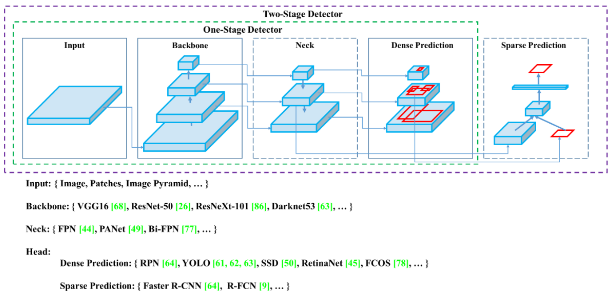
***Index Terms*—YOLO, AKAZE features, object tracking**

I. Introduction

Object tracking is used in many interesting applications such as robotics and autonomous vehicles. There are many existing approaches for object tracking. This paper describes a method for robust tracking of a car in autonomous driving application in order to recover camera pose and structure from motion.

II. Related Work

A. YOLO



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Fig. 1. Structure of an Object Detector, *reproduced from [1]*

YOLOv4 object detector has the same basic structure as the figure above. It utilizes CSPDarknet53 as backbone, Spatial Pyramid Pooling and Path Aggregation Network in the neck, and a YOLO head. Since we are working on an embedded system, we need to keep the resources required to a minimum. For that purpose, we use YOLOv4-Tiny instead of the full YOLOv4. YOLOv4-Tiny is a much smaller version of YOLOv4 which utilizes much less resources. Since we only require the YOLO detector for detecting cars, a pre-trained model suffices for the job. However, it would definitely boost performance if we fine-tuned the model on a car dataset with images taken under diverse conditions.

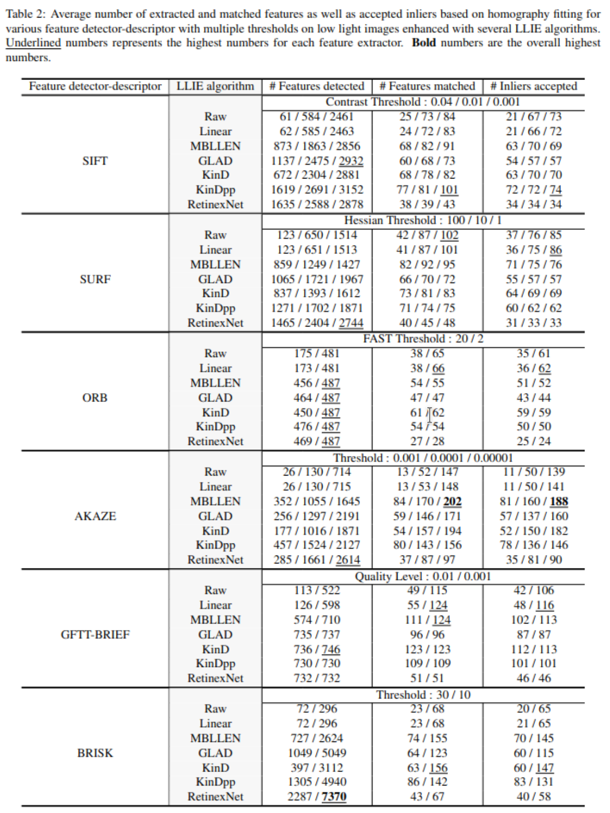


Table 2: Average number of extracted and matched features as well as accepted inliers based on homography fitting for various feature detector-descriptor with multiple thresholds on low light images enhanced with several LLIE algorithms. Underlined numbers represents the highest numbers for each feature extractor. **Bold** numbers are the overall highest numbers.

Feature detector-descriptor	LLIE algorithm	# Features detected	# Features matched	# Inliers accepted
		Contrast Threshold : 0.04 / 0.01 / 0.001		
SIFT	Raw	61 / 584 / 2461	25 / 73 / 84	21 / 67 / 73
	Linear	62 / 585 / 2463	24 / 72 / 83	21 / 66 / 72
	MBLLEN	873 / 1863 / 2856	68 / 82 / 91	63 / 70 / 69
	GLAD	1137 / 2475 / 2932	60 / 68 / 73	54 / 57 / 57
	KinD	672 / 2304 / 2881	68 / 78 / 82	63 / 70 / 70
	KinDpp	1619 / 2691 / 3152	77 / 81 / 101	72 / 72 / 74
	RetinexNet	1635 / 2588 / 2878	38 / 39 / 43	34 / 34 / 34
		Hessian Threshold : 100 / 10 / 1		
SURF	Raw	123 / 650 / 1514	42 / 87 / 102	37 / 76 / 85
	Linear	123 / 651 / 1513	41 / 87 / 101	36 / 75 / 86
	MBLLEN	859 / 1249 / 1427	82 / 92 / 95	71 / 75 / 76
	GLAD	1065 / 1721 / 1967	66 / 70 / 72	55 / 57 / 57
	KinD	837 / 1393 / 1612	73 / 81 / 83	64 / 69 / 69
	KinDpp	1271 / 1702 / 1871	71 / 74 / 75	60 / 62 / 62
	RetinexNet	1465 / 2404 / 2744	40 / 45 / 48	31 / 33 / 33
		FAST Threshold : 20 / 2		
ORB	Raw	175 / 481	38 / 65	35 / 61
	Linear	173 / 481	38 / 66	36 / 62
	MBLLEN	456 / 487	54 / 55	51 / 52
	GLAD	464 / 487	47 / 47	43 / 44
	KinD	450 / 487	61 / 62	59 / 59
	KinDpp	476 / 487	54 / 54	50 / 50
	RetinexNet	469 / 487	27 / 28	25 / 24
		Threshold : 0.001 / 0.0001 / 0.00001		
AKAZE	Raw	26 / 130 / 714	13 / 52 / 147	11 / 50 / 139
	Linear	26 / 130 / 715	13 / 53 / 148	11 / 50 / 141
	MBLLEN	352 / 1055 / 1645	84 / 170 / **202**	81 / 160 / **188**
	GLAD	256 / 1297 / 2191	59 / 146 / 171	57 / 137 / 160
	KinD	177 / 1016 / 1871	54 / 157 / 194	52 / 150 / 182
	KinDpp	457 / 1524 / 2127	80 / 143 / 156	78 / 136 / 146
	RetinexNet	285 / 1661 / 2614	37 / 87 / 97	35 / 81 / 90
		Quality Level : 0.01 / 0.001		
GFTT-BRIEF	Raw	113 / 522	49 / 115	42 / 106
	Linear	126 / 598	55 / 124	48 / 116
	MBLLEN	574 / 710	111 / 124	102 / 113
	GLAD	735 / 737	96 / 96	87 / 87
	KinD	736 / 746	123 / 123	112 / 113
	KinDpp	730 / 730	109 / 109	101 / 101
	RetinexNet	732 / 732	51 / 51	46 / 46
		Threshold : 30 / 10		
BRISK	Raw	72 / 296	23 / 68	20 / 65
	Linear	72 / 296	23 / 68	21 / 65
	MBLLEN	727 / 2624	74 / 155	70 / 145
	GLAD	1049 / 5049	64 / 123	60 / 115
	KinD	397 / 3112	63 / 156	60 / 147
	KinDpp	1305 / 4940	86 / 142	83 / 131
	RetinexNet	2287 / **7370**	43 / 67	40 / 58

Fig. 2. Comparison of Feature detector-descriptors, *reproduced from [2]*

B. AKAZE Local Features

Since our application is meant to work under all conditions which it may encounter, we need to take into account various scenarios which may be cause of some problems. Low illumination levels may be one such problem. Shyam et al. [2] explores various feature detector-descriptors for their performance under low illumination levels. They compare SIFT, SURF, ORB, AKAZE, BRIEF and BRISK for this task. Overall, they summarize that with low threshold levels, using AKAZE results in highest final inlier points. The table comparing these results is reproduced above.

The table in Fig. 3 is reproduced from [3]. The table shows features-matching times and feature-detection times for several

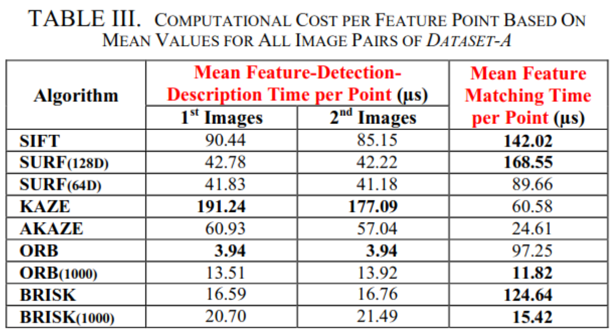


TABLE III. COMPUTATIONAL COST PER FEATURE POINT BASED ON MEAN VALUES FOR ALL IMAGE PAIRS OF *DATASET-A*

Algorithm	**Mean Feature-Detection-Description Time per Point (µs)**		**Mean Feature Matching Time per Point (µs)**
	1st Images	**2nd Images**	
SIFT	90.44	85.15	**142.02**
SURF(128D)	42.78	42.22	**168.55**
SURF(64D)	41.83	41.18	89.66
KAZE	**191.24**	**177.09**	60.58
AKAZE	60.93	57.04	24.61
ORB	**3.94**	**3.94**	97.25
ORB(1000)	13.51	13.92	**11.82**
BRISK	16.59	16.76	**124.64**
BRISK(1000)	20.70	21.49	**15.42**

Fig. 3. Comparison of computational efficiency Feature detector-descriptors, *reproduced from [3]*

feature-detectors. AKAZE is second to BRISK (1000) for being most efficient in feature-matching time, however it is not the best in terms of feature-detection times..

C. Structure from Motion

Structure from Motion (SfM) is a photogrammetry technique whereby a sequence of overlapping images is used to recover information about 3D geometry of the scene. A typical SfM pipeline, referenced from [4], proceeds as below.

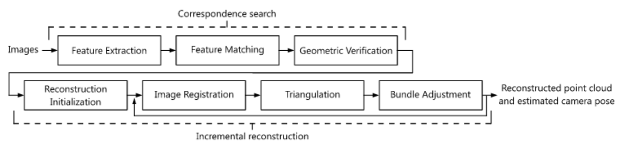


Fig. 4. Structure from Motion Pipeline, *reproduced from [4]*

III. METHODOLOGY

The application is based on using a single camera to track the car moving in front of the host vehicle. The camera is fixed at the front of the car. A YOLO-based detector is used to detect the car in the image taken from the camera. The bounding box prediction for the car is then used as a mask to identify AKAZE keypoints for the car. For the subsequent images taken from the camera, the keypoints are detected from the images using AKAZE detector and matched with the previously computed features from the car.

$$boundingbox = (x, y, width, height) \tag{1}$$

The YOLO-based detector is not always able to detect the car from the image in case of noisy conditions e.g low light, occlusion, blur etc. Complimenting it with AKAZE features enables robust tracking in noisy conditions as well as gives us exact point correspondences to extract geometric structure.

However as the car moves ahead, the changes in illumination and pose results in fewer inlier points from the SfM calculation and thus we aren't able to compute the structure. We compensate for that by using the detection from YOLO to compute/refresh new keypoints to be used for matching in the subsequent images. At every prediction by YOLO, which passes a pre-defined confidence threshold, we update the keypoints.

The detected keypoints are matched through a brute-force knn matcher using the Hamming distance. The matched pair of keypoints are then put through a distance ratio test[] where all keypoint pairs with nearest neighbour distance ratio less than 0.8 are eliminated.

The rest of the keypoint pairs are used to calculate a homography. A minimum of four corresponding points are required to compute the homography using the Direct Linear Transform and RANSAC method. This homography gives us the perspective transform from the previous position of the car to the current position. The homography is used to project the car bounding box to the next position.

The pair-correspondences are also used to calculate the Essential Matrix. A minimum of 5 points are required to get the Essential Matrix. The Essential Matrix and the inlier points used to calculate the Essential Matrix are then used to recover the Projection Matrices. The Projection Matrices can then be used for the 2D to 3D point transformations using the triangulation method. The resulting structure can be used to build a 3D model of the object in question.

Given corresponding 2D points x, x' from feature matching,

$$x'^T F x = 0 \tag{2}$$

$$AF = 0 \tag{3}$$

$$A = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \tag{4}$$

$$FundamentalMatrix = null(A) \tag{5}$$

$$EssentialMatrix = K'^T F K \tag{6}$$

IV. RESULTS

The application was tested on two videos, one taken during the day, the other at night. We are able to get 6 FPS from the video stream using this algorithm. We see that YOLO detector works very well during the well-lit environment of the day-video. As we get steady predictions from YOLO and are able to refresh the AKAZE features, our object detector works quite well and we are able to get very good corresponding points for computing homography and get projection matrices.

However, in the video taken during nighttime, the ambient lighting is low, there is glare coming from light sources and the image is noisy and blurry. In this case, the predictions from YOLO are sparse, and our detector is forced to mostly rely on diminishing number of matches from AKAZE features for computing homography. In this case, the inlier points are often not enough to compute a reliable homography. However, the detector is still able to detect and track the object, which would not be possible if we just relied on YOLO.

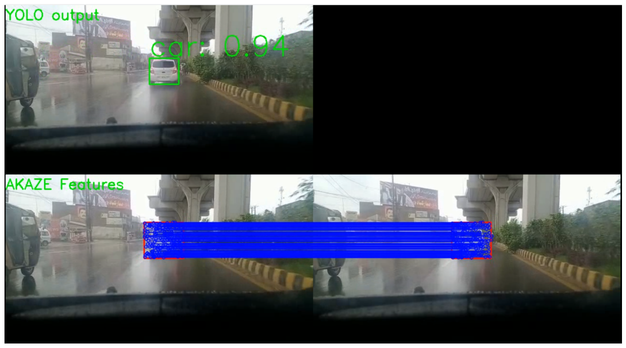


Fig. 5. Detection of car in well lit environment

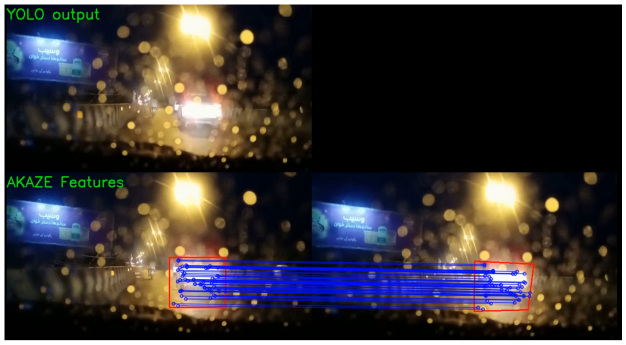


Fig. 6. Detection of car under very noisy conditions

V. Limitations

Currently the algorithm only accounts for single object detection and tracking. In case more than one cars are detected at the same time, the algorithm uses the boundary box computed using the last homography to choose the closest bounding box. This is not effective all of the time as the object we intend to detect is not always included in the YOLO predictions for every image frame.

An improvement over the current algorithm would be to employ a form of Kalman filter to take into account the noisy readings from the YOLO detector.

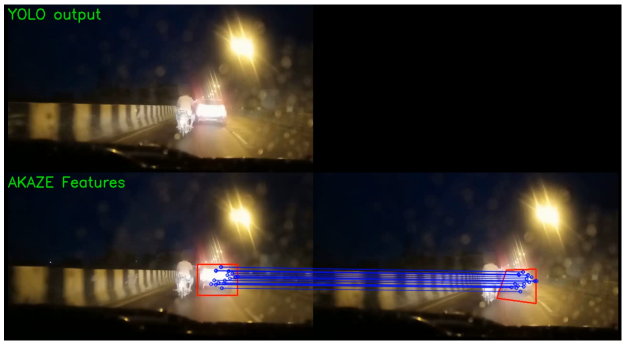


Fig. 7. Homography calculated from noisy data

References

[1] A. Bochkovskiy, C. Wang H. M. Liao, ”YOLOv4: Optimal Speed and Accuracy of Object Detection”, 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)

[2] P. Shyam, A. Bangunharcana and K. Kim, ”Retaining Image Feature Matching Performance Under Low Light Conditions”, 2020 20th International Conference on Control, Automation and Systems (ICCAS 2020) Oct. 1316, 2020; BEXCO, Busan, Korea

[3] S.A.K. Tareen, Z. Saleem, ”A Comparative Analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK”, 2018 International Conference on Computing, Mathematics and Engineering Technologies – iCoMET 2018

[4] S. Bianco, G. Ciocca D. Marelli, ”Evaluating the Performance of Structure from Motion Pipelines”, August 2018, Journal of Imaging 4(8):98