Car Tracking Algorithm-v2

Algorithm 1 tracking

Input Parameters:
imagespath:(List of paths to all images we want to perform tracking on.)
boundingbox:(A box around car at first frame. This box will update as we move to next images.)
IntervalSize: (Interval Size to calculate forward and backward flow error.)
GeomatricThreshold: (Threshold to filter points on base of geomatric distance.)
ForwardZeroFlowThreshold: (Threshold to filter points on base of zero flow while moving forward.)
BackwardZeroFlowThreshold: (Threshold to filter points on base of zero flow while moving backward.)
DescriptorThreshold: (Threshold to filter points on base of descriptor matching.)
flowinboxmethod: (method to computer flow in box from one frame to other .)
ransacmethod: (method to compute ransac vector for flow vectors..)
maxcorners: (specfiy how many corner you want to find at max.)
qualitylevel: (threshold to get or ignore corner in corner detector)
mindistance: (Minimum possible Euclidean distance between the returned corners.)
useHarrisDetectorboolean: (boolean to check whether to use harris detector or not)
k: (free parameter in harris corner detector.)
Output:
featurespointineveryframe: (The list of list of good points which we track in all images

My Algorithm:
$featurespointineveryframe = []$
$ImageCounter \leftarrow 0$
while $ImageCounter < Len(imagespath)$ **do**
    $ImageCounter \leftarrow 0$
    $images, grayimages \leftarrow getframes(imagespath, ImageCounter, IntervalSize)$
    $pointsat1stframe \leftarrow Initialization(BoundingBox, grayimages[0], method = 'gCorners', maxcorners, qualitylevel, mindistance, useHarrisDetectorboolean, k)$
    $pointsatkframe, filteredpointsat1stframe \leftarrow getforwardbackwardflow(grayimages, pointsat1stframe, pointsat1stframe, ForwardZeroFlowThreshold, 'forward')$
    $pointsat1stframebackward, filteredpointsat1stframe \leftarrow getforwardbackwardflow(Images, pointsatkframe, filteredpointsat1stframe, BackwardZeroFlowThreshold, 'backward')$
    $GoodPoints \leftarrow applyallfilters(filteredpointsat1stframe, pointsatkframe, pointsat1stframebackwar, grayimages, GeomatricThreshold, DescriptorThreshold, method = 'ssd')$
    $featurespointineveryframe.append \leftarrow GoodPoints$
    $Flow, FlowPoints \leftarrow CalculateBoxFlow(grayimages[ImageCounter], grayimages[ImageCounter], GoodPoints, flowinboxmethod, ransacmethod)$
    $BoundingBox \leftarrow UpdateFlowvector$
    if $ImageCounter == len(AllImages) - 2$ **then**
        $featurespointineveryframe.append \leftarrow FlowPoints$
    $ImageCounter++$
return $featurespointineveryframe$

Algorithm 2 getframes

Input Parameters:
imagespath: (This is the list of images path in same order as images in video stream.)
firstimageindex: (starting index to read images from path given in list of images path.)
intervalsize: (Specifies how many frames to read)
images: (queue of numpy arrays and each index specfiy a rgb image.)
grayimages: (queue of numpy arrays and each index specfiy a image in gray scale.)
Output:
images: (queue of numpy array and each index specify a rgb image.)
grayimages: (queue of numpy array and each index specify a gray image.)
Status:(1 for successful and -1 in case of no frame found)

My Algorithm:

if $firstimageindex$ *or* $intervalsize < 0$ **then return** $images, grayimages, status \leftarrow$ *-1*
else
    $a \leftarrow$ *firstimageindex + intervalsize*
    if $a > len(imagespath) - 1$ **then**
        $a \leftarrow len(imagespath) - 1$
    if *images.empty* **then**
        while $firstimageindex < a$ **do**
            img ← read(imagespath[firstimageindex])
            grayimg ← convertimgtograyscale
            images.enqueue ← img
            grayimages.enqueue ← grayimg
            firstimageindex++
    else
        images ← images.dequeue
        grayimages ← grayimages.dequeue
        img ← read(imagespath[a-1])
        grayimg ← convertimgtograyscale
        images.enqueue ← img
        grayimages.enqueue ← grayimg
    if $len(images) == 0$ **then return** $images, grayimages, status \leftarrow$ *-1*
    else
        return $images, grayimages, status \leftarrow$ *1*

Algorithm 3 initialization

Input Parameters:
boundingbox: (This contains the starting and ending index of box.[xmin,ymin,xmax,ymax].)
grayimage: (Image of which we want to initilize points.)
method: (Method which specify, how to get Image Points)
maxcorners: (specfiy how many corner you want to find at max.)
qualitylevel: (threshold to get or ignore corner in corner detector)
mindistance: (Minimum possible Euclidean distance between the returned corners.)
useHarrisDetectorboolean: (boolean to check whether to use harris detector or not)
k: (free parameter in harris corner detector.)
Output:
Points: (List of points in range of given box)
Status:(1 for successful and -1 in case of no points found)

My Algorithm:

$Status = -1$
if $Method == "grid"$ **then**
    $Points \leftarrow$ *points between(xmin,ymin) and (xmax,ymax)*
    return $Points, status \leftarrow 1$
if $Method == "gCorners"$ **then**
    $patch = im[ymin : ymax, xmin : xmax]$
    $corners \leftarrow goodFeaturesToTrack(patch, maxcorners, qualitylevel, mindistance, useHarrisDetector = useHarrisDetectorboolean, k)$
    $corners \leftarrow corners.reshape((-1, 2))$
    $corners[:, 0], corners[:, 1] \leftarrow corners[:, 0] + xmin, corners[:, 1] + ymin$
    return $Points \leftarrow corners, status \leftarrow 1$

Algorithm 4 getforwardbackwardflow

Input Parameters:
grayimages: (queue of numpy array of gray Images)
$featurepointsin1stimage$: $(Thislistcontainsthefeaturespointsonfirstimage, ofwhichwewanttocompute$
$filteredfeaturepointsin1stimage$: $(Thislistcontainsthefeaturespointsonfirstimage, ofwhichwewanttoc$
thresholdzeroflow: (This value specify the theshold value to filter background points from object points)
direction: (this specifiy the direction to compute flow on given images)
Output:
$trackedpointsinkimage$: $(givenfeaturepointsatlastframeofgivenimages.)$
$trackedpointsin1stimage$: $(Givenfeaturespoint, whichwewanttotrackongivensetofimages)$
Status:(1 for successful and -1 in case of no points found after zero flow filter)

My Algorithm:

$Status = -1$
$listofgrayimages \leftarrow list(grayimages.queue)$
if $direction ==' Forward'$ **then**
    $a \leftarrow 0$
    while $a < len(listofgrayImages)$ **do**:
        $next \leftarrow a++$
        $currentImage - Gray \leftarrow listofgrayimages[a]$
        $nextImage - Gray \leftarrow listofgrayimages[next]$
        $flowpoints, st, err \leftarrow cv2.calcOpticalFlowPyrLK(currentImage - Gray, nextImage-Gray, featurepointsin1stimage.astype(np.float32), None, **lk_params)$
        $a++$
        if $a \leftarrow (len(listofgrayimages) - 2)$ **then**
            $trackedpointsinkimage, trackedpointsin1stimage \leftarrow CalculateZeroFlow(flowpoints, featurepointsin1stimage, thresholdzeroflow)$
else
    $a \leftarrow len(listofgrayImages) - 1$
    while $a > 0$ **do**:
        $next \leftarrow a--$
        $currentimage - Gray \leftarrow listofgrayimages[a]$
        $nextImage - Gray \leftarrow listofgrayimages[next]$
        $flowpoints, st, err \leftarrow cv2.calcOpticalFlowPyrLK(currentImage - Gray, nextImage-Gray, featurepointsin1stimage.astype(np.float32), None, **lk_params)$
        $a--$
        if $a \leftarrow (len(listofgrayimages) - 2)$ **then**
            $trackedpointsinkimage, trackedpointsin1stimage \leftarrow CalculateZeroFlow(flowpoints, filteredfeaturepointsin1stimage, thresholdzeroflow)$
if $len(trackedpointsin1stimage) == 0$ **then return** $trackedpointsinkimage, trackedpointsin1stimage, Status \leftarrow$ *-1*
else
    return $trackedpointsinkimage, trackedpointsin1stimage, Status \leftarrow 1$

Algorithm 5 CalculateZeroFlow

Input Parameters:
pointsinKimage:(Points at the last frame after optical flow as an numpy array.)
pointsin1stimage:(Given Points at the first image as an numpy array)
threshold:(Threshold to filter points on base of zero flow)
Output:
filteredpointsinKimage: (Last Image point after zero flow filter)
filteredpointsin1stimage: (First Image point after zero flow filter)

My Algorithm:

$filteredpointsinKimage = []$
$filteredpointsin1stimage = []$
$difference \leftarrow np.linalg.norm(pointsin1stimage - pointsinkimage, 2, 1)$
$t \leftarrow 0$
while $t < len(distance) - 1$ **do**
    if $difference < threshold$ **then**
        $filteredpointsinKimage.append \leftarrow pointsinKimage[t]$
        $filteredpointsin1stimage.append \leftarrow pointsin1stimage[t]$
    $t++$
return $filteredpointsinKimage, filteredpointsin1stimage$

difference = [0.1, 2, 0.5, 10, 15, 30, 10, ...]
t_difference = difference < threshold = 2
# t_difference will be a binary array
e.g. t_difference = [true, false, true, false, ...]
this binary array can be used to access elements of an array.

filtered_points_in_kth_image = points_in_kth_image[not t_diff]
filtered_points_in_1st_image = points_in_1st_image[not t_diff]

Algorithm 6 applyallfilters

Input Parameters:
pointsin1stimage:(Features Points at start of image)
pointsinkimageforward:(Given Points tracked to given interval size.)
pointsin1stimagebackward:(Given forward Points tracked back to first image.)
grayimages:(Set of images of given interval size, on which we want to track points.)
GeomatricThreshold:(Threshold to filter points on base of geomatric distance.)
DescriptorThreshold:(Threshold to filter points on base of descriptor SSD or Dot Product)
Output:
filteredpoints: (Return Good features points which pass criteria of filters.)
Status: (Return -1 if no good feature point found otherwise 1.)

My Algorithm:

$varGoodPoints = []$
$varForwardPoints = []$
$filteredpoints = []$
$Status \leftarrow -1$
$geomatricdisplacement \leftarrow np.linalg.norm(pointsin1stimage - pointsin1stimagebackward, 2, 1)$
$i \leftarrow 0$
while $i < len(geomatricdisplacement) - 1$ **do**
    if $geomatricdisplacement[i] <= GeomatricThreshold$ **then**
        $varGoodPoints.append \leftarrow pointsin1stimage[i]$
        $varForwardPoints.append \leftarrow pointsinkimageforward[i]$
    $i++$
$orb \leftarrow cv2.ORB_create()$
$FirstImageGray \leftarrow grayimages[First]$
$LastImageGray \leftarrow grayimages[Last]$
$keyPoints_Good \leftarrow KeyPoints(varGoodPoints)$
$keyPoints_Farword \leftarrow KeyPoints(varFarwordPoints)$
$Goodkp, GoodptsDes \leftarrow orb(FirstGrayImage, keyPoints_Good)$
$Farwordkp, FarwordptsDes \leftarrow orb(LastGrayImage, keyPoints_Farword)$
$t = 0$
$gdpt_des \leftarrow NormalizedGoodptsDes$
$forward_des \leftarrow NormalizedFarwordptsDes$
$ssd_distance \leftarrow np.linalg.norm(forward_des - gdpt_des, 2, 1)$
while (**do**$t < len(ssd_distance) - 1$)
    if $ssd_distance <$ "$DescriptorThreshold$" **then**
        $FinalGoodPoints.append \leftarrow varGoodPoints[t]$
    $t++$
if $len(FinalGoodPoints) == 0$ **then return** $FinalGoodPoints, Status \leftarrow$ *-1*
else
    return $FinalGoodPoints, Status \leftarrow 1$

Algorithm 7 CalculateBoxFlow

Input Parameters:
firstimage:(Current Image from which we want to calculate flow)
nextimage: (Very next image to current image)
goodpoints: (Features points on current image to compute flow as numpy array)
method: (Method to compute box flow.)
ransacmethod: (Method to compute flow vector through ransac.)
Output:
Flow: (This contain the change in x and y coordinates, which specifies the change in flow of box.)
ForwardPoints: (This list contain flow of points from current image to next image)

My Algorithm:

$ForwardPoints, st, err \leftarrow cv2.calcOpticalFlowPyrLK(currentImage, nextImage, FirstImageFeatureP$
$lk_params)$
$flow \leftarrow (forwardpoints - goodpoints)$
if $method ==' median'$ **then**
    $shape \leftarrow flow.shape$
    $rowsize \leftarrow shape(0)$
    $flowinbox = np.divide(np.sum(flow, 0), rowsize)$
if $method ==' ransac'$ **then**
    $flowinbox = getRansac(flow, threshold, ransacmethod)$
return $flowinbox, forwardpoints$

this is not 'median'
this is avrage flow

Another example

B = [0,0,1,5,4.5,5,5,5,5,6,10,0,0,15]

sort_B = [0,0,0,0,1,4.5,5,5,5,5,5,6,10,15]

len=14

median_B = 5

average_B = 4.4

Median eg. A = [1,2,2,3,4]

↳ A = sort_array in ascending order (A)

↳ median: pick the mid point of A i.e. value at $A[\frac{Len(A)}{2}]$ if len(A) is odd.

↳ In above example median = 2

Algorithm 8 GetRansac

Input Parameters:
flow:(Flow of good points, we get as output of calculate flow method)
Threshold:(Threshold to filter inlairs from outlairs)
Method:(method to get ransac. One option is to return best hypothesis and second is to return medium of
Output:
FlowPoint: (Return a single point which has the most number of inlairs.)

My Algorithm:

$Inlairs = []$
$NormalizedFlowPoints \leftarrow Normalized\ to1$
$samplesize \leftarrow 20$
if $len(NormalizedFlowPoints) - 1 < samplesize$ **then**
    $samplesize = len(NormalizedFlowPoints) - 1$
$randomsample = random.sample(range(0, len(NormalizedFlowPoints)-1), samplesize)$
while $t < samplesize$ **do**
    $tempInlairs = []$
    $i \leftarrow randomsample[t]$
    $hypothesis \leftarrow NormalizedFlowPoints[i]$
    $product \leftarrow np.dot(NormalizedFlowPoints, hypothesis)$
    loop:till len(product)
    for $j \leftarrow inproduct$ **do**
        if $j >= threshold$ **then**
            $tempInlairs.append \leftarrow j$
    if $len(tempInlairs) >= len(Inlairs)$ **then**
        $Inlairs \leftarrow tempInlairs[]$
        if $method ==' besthypothesis'$ **then**
            $FlowPoint \leftarrow flow[i]$
        if $method ==' median'$ **then**
            $shape \leftarrow flow.shape$
            $rowsize \leftarrow shape(0)$
        $FlowPoint = np.divide(np.sum(tempInlairs, 0), rowsize)$
return *FlowPoint*

can be vectorized using comments on previous pages.

not median