Mind Pointeye CV Test Report

Han Runbing

1.Problem Statement

Given sequential rectified stereo images with camera information, the objects (car and person) should be detected and tracked across image frames and the tracking ID, distance from the bounding box center to the camera optic center should be attached and calculated.

In this case, the subtasks of this project are:

1. Multiple objects detection
2. Multiple objects tracking
3. Multiple objects distance estimation

2. Method

2.1 Objects Detection

2.1.1 Method Selection

Existing object detection methods are traditional detection methods and deep learning-based detection method. Compared to the traditional detection methods, deep learning based methods with higher accuracy and perform speed are commonly used. Among these methods, YOLOv3 has the advantages of detection speed and accuracy and meets the real-time requirements for objects detection**(Figure 1).** In mAP measured at .5 IOU YOLOv3 is on par with Focal Loss but about 4x faster. Thus, the yolov3 was selected here as object detection methods[1].

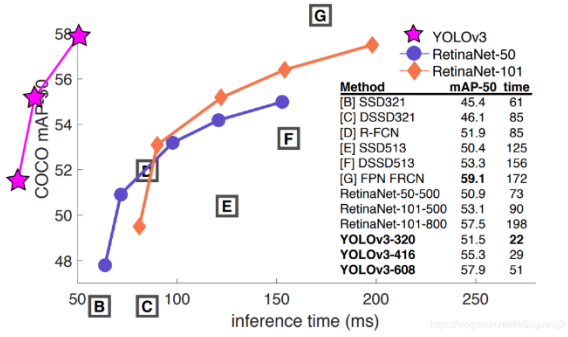

Figure 1. Yolov3 performance

2.1.2 implement of yolov3

The yolov3 model here is already trained with fixed yolov3.weight. Thus, the model can be used to predict and detect the objects immediately. The requirement is to detection all person and car in the images. Based on the classes of yolov3, the classes name list is ["person", "car","bus","truck","train"].

1. implement of network darknet53

The backbone feature extraction network used by YoloV3 is Darknet53, which has two important features **(Figure 2):**

a. An important feature of Darknet53 is the use of the residual network ResidualThe characteristic of the residual network is that it is easy to optimize and can increase the accuracy by adding considerable depth[1]. The internal residual block uses jump connections to alleviate the problem of gradient disappearance caused by increasing depth in the deep neural network.

b. Each convolution part of Darknet53 uses the unique DarknetConv2D structure, l2 regularization is performed during each convolution, and BatchNormalization standardization and LeakyReLU are performed after the convolution is completed. Ordinary ReLU sets all negative values to zero, while Leaky ReLU assigns a non-zero slope to all negative values.

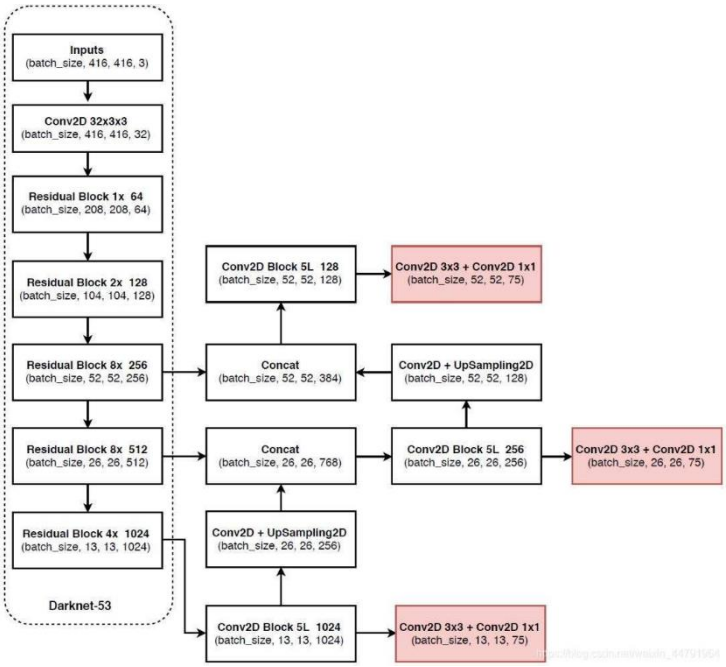

Figure 2a. Yolov3 architecture

```python
@wraps(Conv2D)
def DarknetConv2D(*args, **kwargs):
    """Wrapper to set Darknet parameters for Convolution2D."""
    darknet_conv_kwargs = {'kernel_regularizer': l2(5e-4)}
    darknet_conv_kwargs['padding'] = 'valid' if kwargs.get('strides')==(2,2)
    darknet_conv_kwargs.update(kwargs)
    return Conv2D(*args, **darknet_conv_kwargs)

def DarknetConv2D_BN_Leaky(*args, **kwargs):
    """Darknet Convolution2D followed by BatchNormalization and LeakyReLU."""
    no_bias_kwargs = {'use_bias': False}
    no_bias_kwargs.update(kwargs)
    return compose(
        DarknetConv2D(*args, **no_bias_kwargs),
        BatchNormalization(),
        LeakyReLU(alpha=0.1))

def resblock_body(x, num_filters, num_blocks):
    '''A series of resblocks starting with a downsampling Convolution2D'''
    # Darknet uses left and top padding instead of 'same' mode
    x = ZeroPadding2D(((1,0),(1,0)))(x)
    x = DarknetConv2D_BN_Leaky(num_filters, (3,3), strides=(2,2))(x)
    for i in range(num_blocks):
        y = compose(
                DarknetConv2D_BN_Leaky(num_filters//2, (1,1)),
                DarknetConv2D_BN_Leaky(num_filters, (3,3)))(x)
        x = Add()([x,y])
    return x

def darknet_body(x):
    '''Darknent body having 52 Convolution2D layers'''
    x = DarknetConv2D_BN_Leaky(32, (3,3))(x)
    x = resblock_body(x, 64, 1)
    x = resblock_body(x, 128, 2)
    x = resblock_body(x, 256, 8)
    x = resblock_body(x, 512, 8)
    x = resblock_body(x, 1024, 4)
    return x
```

2b. Implement of darknet53

2. Obtain prediction results from features

The process of obtaining prediction results from features can be divided into two parts, namely:

a. Construct FPN feature pyramid for enhanced feature extraction.

YoloV3 extracts multiple feature layers for target detection, and extracts a total of three

feature layers. The three feature layers are located at different positions of the main part of Darknet53, which are located in the middle layer, middle and lower layer, and bottom layer. The feature pyramid can merge feature layers of different shapes, which is conducive to extracting better features.

b. Use Yolo Head to predict the three effective feature layers.
Using the FPN feature pyramid, we can obtain three enhanced features, and then we use the feature layer of these three shapes Pass in Yolo Head to get the prediction result. Yolo Head is essentially a 3x3 convolution plus a 1x1 convolution. The function of 3x3 convolution is feature integration, and the function of 1x1 convolution is to adjust the number of channels. The three feature layers are processed separately.

```
def make_last_layers(x, num_filters, out_filters):
    '''6 Conv2D_BN_Leaky layers followed by a Conv2D_linear layer'''
    x = compose(
            DarknetConv2D_BN_Leaky(num_filters, (1,1)),
            DarknetConv2D_BN_Leaky(num_filters*2, (3,3)),
            DarknetConv2D_BN_Leaky(num_filters, (1,1)),
            DarknetConv2D_BN_Leaky(num_filters*2, (3,3)),
            DarknetConv2D_BN_Leaky(num_filters, (1,1)))(x)
    y = compose(
            DarknetConv2D_BN_Leaky(num_filters*2, (3,3)),
            DarknetConv2D(out_filters, (1,1)))(x)
    return x, y


def yolo_body(inputs, num_anchors, num_classes):
    """Create YOLO_V3 model CNN body in Keras."""
    darknet = Model(inputs, darknet_body(inputs))
    x, y1 = make_last_layers(darknet.output, 512, num_anchors*(num_classes+5))

    x = compose(
            DarknetConv2D_BN_Leaky(256, (1,1)),
            UpSampling2D(2))(x)
    x = Concatenate()([x,darknet.layers[152].output])
    x, y2 = make_last_layers(x, 256, num_anchors*(num_classes+5))

    x = compose(
            DarknetConv2D_BN_Leaky(128, (1,1)),
            UpSampling2D(2))(x)
    x = Concatenate()([x,darknet.layers[92].output])
    x, y3 = make_last_layers(x, 128, num_anchors*(num_classes+5))

    return Model(inputs, [y1,y2,y3])
```

```
def yolo_head(feats, anchors, num_classes, input_shape):
    """Convert final layer features to bounding box parameters."""
    num_anchors = len(anchors)
    # Reshape to batch, height, width, num_anchors, box_params.
    anchors_tensor = K.reshape(K.constant(anchors), [1, 1, 1, num_anchors, 2])

    grid_shape = K.shape(feats)[1:3] # height, width
    grid_y = K.tile(K.reshape(K.arange(0, stop=grid_shape[0]), [-1, 1, 1, 1]),
        [1, grid_shape[1], 1, 1])
    grid_x = K.tile(K.reshape(K.arange(0, stop=grid_shape[1]), [1, -1, 1, 1]),
        [grid_shape[0], 1, 1, 1])
    grid = K.concatenate([grid_x, grid_y])
    grid = K.cast(grid, K.dtype(feats))

    feats = K.reshape(
        feats, [-1, grid_shape[0], grid_shape[1], num_anchors, num_classes + 5])

    box_xy = K.sigmoid(feats[..., :2])
    box_wh = K.exp(feats[..., 2:4])
    box_confidence = K.sigmoid(feats[..., 4:5])
    box_class_probs = K.sigmoid(feats[..., 5:])

    # Adjust preditions to each spatial grid point and anchor size.
    box_xy = (box_xy + grid) / K.cast(grid_shape[::-1], K.dtype(feats))
    box_wh = box_wh * anchors_tensor / K.cast(input_shape[::-1], K.dtype(feats))

    return box_xy, box_wh, box_confidence, box_class_probs
```

Figure 3. Implement of prediction

3. Decoding of prediction results

From the second step, we can obtain the prediction results of the three feature layers. Since each grid point has three a priori boxes, the above prediction result can be reshaped as: (N,13,13,3,85) (N,26,26,3,85) (N,52,52,3,85) The 85 can be divided into 4+1+80, which represent x_offset, y_offset, h and w, confidence, and classification result, respectively.

However, this prediction result needs to be decoded before it can be completed. The decoding process of YoloV3 is divided into two steps:

- First add each grid point its corresponding x_offset and y_offset, and the result after adding is the center of the prediction box.
- Then use the combination of a priori box and h and w to calculate the width and height of the prediction box.

In this way, the position of the entire prediction box can be obtained. After obtaining the final prediction results, score ranking and non-maximum suppression screening must be performed:

- Take out the boxes and scores of each category whose score is greater than self.obj_threshold.

- Use frame position and score for non-maximum suppression.

```python
def yolo_correct_boxes(box_xy, box_wh, input_shape, image_shape):
    '''Get corrected boxes'''
    box_yx = box_xy[..., ::-1]
    box_hw = box_wh[..., ::-1]
    input_shape = K.cast(input_shape, K.dtype(box_yx))
    image_shape = K.cast(image_shape, K.dtype(box_yx))
    new_shape = K.round(image_shape * K.min(input_shape/image_shape))
    offset = (input_shape-new_shape)/2./input_shape
    scale = input_shape/new_shape
    box_yx = (box_yx - offset) * scale
    box_hw *= scale

    box_mins = box_yx - (box_hw / 2.)
    box_maxes = box_yx + (box_hw / 2.)
    boxes =  K.concatenate([
        box_mins[..., 0:1],  # y_min
        box_mins[..., 1:2],  # x_min
        box_maxes[..., 0:1],  # y_max
        box_maxes[..., 1:2]  # x_max
    ])

    # Scale boxes back to original image shape.
    boxes *= K.concatenate([image_shape, image_shape])
    return boxes


def yolo_boxes_and_scores(feats, anchors, num_classes, input_shape, image_shape):
    '''Process Conv layer output'''
    box_xy, box_wh, box_confidence, box_class_probs = yolo_head(feats,
        anchors, num_classes, input_shape)
    boxes = yolo_correct_boxes(box_xy, box_wh, input_shape, image_shape)
    boxes = K.reshape(boxes, [-1, 4])
    box_scores = box_confidence * box_class_probs
    box_scores = K.reshape(box_scores, [-1, num_classes])
    return boxes, box_scores
```

Figure 3. Decoding of prediction results

2.2 Objects Tracking

2.2.1 Method Selection

Common multi-target tracking algorithms can generally be divided into detection-based tracking (Detection-Based Tracking) and detection-free tracking (Detection-Free Tracking). DBT requires a target detector to first detect the target in each frame of image, while DFT requires that the position of each target be known for the first time, and then each target is tracked separately. Obviously, the setting of the former is closer to actual application scenarios, and it is also the mainstream of practical application.

Among Detection-Based Tracking methods, SORT[2].and deepsort[3]are commonly used tracking methods. SORT (Simple Online and Realtime Tracking)is a prototype of the mainstream Tracking-by-Detection framework and many subsequent works have similar frame. DeepSORT (Simple Online and Realtime Tracking with a Deep Association Metric.) is an improved version of SORT authors based on SORT. Its biggest contribution is the use of deep CNN to extract target features as Match criteria. In addition, DeepSORT defines a cascading matching method so that tracks with higher recent activity are matched first. Here DeepSORT is applied method as object tracking method.

2.2.2 Implement of DeepSORT

1.The Hungarian algorithm

In DeepSORT, the Hungarian algorithm is used to associate the tracking frame tracks in the previous frame with the detection frame detections in the current frame, and calculate the cost matrix through appearance information and Mahalanobis distance, or IOU [4].

```
def min_cost_matching(
        distance_metric, max_distance, tracks, detections, track_indices=None,
        detection_indices=None):
    """Solve linear assignment problem.
    if track_indices is None:
        track_indices = np.arange(len(tracks))
    if detection_indices is None:
        detection_indices = np.arange(len(detections))

    if len(detection_indices) == 0 or len(track_indices) == 0:
        return [], track_indices, detection_indices  # Nothing to match.

    cost_matrix = distance_metric(
        tracks, detections, track_indices, detection_indices)
    cost_matrix[cost_matrix > max_distance] = max_distance + 1e-5
    indices = linear_assignment(cost_matrix)

    matches, unmatched_tracks, unmatched_detections = [], [], []
    for col, detection_idx in enumerate(detection_indices):
        if col not in indices[:, 1]:
            unmatched_detections.append(detection_idx)
    for row, track_idx in enumerate(track_indices):
        if row not in indices[:, 0]:
            unmatched_tracks.append(track_idx)
    for row, col in indices:
        track_idx = track_indices[row]
        detection_idx = detection_indices[col]
        if cost_matrix[row, col] > max_distance:
            unmatched_tracks.append(track_idx)
            unmatched_detections.append(detection_idx)
        else:
            matches.append((track_idx, detection_idx))
    return matches, unmatched_tracks, unmatched_detections
```

Figure 4. Implement of Hungarian algorithm

2. Kalman Filter

Kalman Filter is widely used in the fields of unmanned aerial vehicle, autonomous driving, satellite navigation, etc. In simple terms, its function is to update the predicted value based on the measured value of the sensor to achieve a more accurate estimation.

In target tracking, the following two states of track need to be estimated:

- Mean: Represents the position information of the target, which is composed of the center coordinates of the bbox (cx, cy), the aspect ratio r, the height h, and the respective speed change values.
- Covariance (Covariance): Represents the uncertainty of the target location information, represented by an 8x8 diagonal matrix. The larger the number in the matrix, the greater the uncertainty, and it can be initialized with any value.

Kalman filtering is divided into two stages:

(1) predict the position of the track at the next moment,

(2) update the predicted position based on the detection.

```python
def predict(self, mean, covariance):
    # 相当于得到t时刻估计值
    # Q 预测过程中噪声协方差
    std_pos = [
                self._std_weight_position * mean[3],
                self._std_weight_position * mean[3],
                1e-2,
                self._std_weight_position * mean[3]
              ]
    std_vel = [
                self._std_weight_velocity * mean[3],
                self._std_weight_velocity * mean[3],
                1e-5,
                self._std_weight_velocity * mean[3]
              ]
    # np.r_ 按列连接两个矩阵
    # 初始化噪声矩阵Q
    motion_cov = np.diag(np.square(np.r_[std_pos, std_vel]))
    # x' = Fx
    mean = np.dot(self._motion_mat, mean)
    # P' = FPF^T+Q
    covariance = np.linalg.multi_dot((self._motion_mat, covariance, self._motion_

    return mean, covariance
```

```python
def project(self, mean, covariance):
    # 将状态分布投影到测量空间
    std = [
        self._std_weight_position * mean[3],
        self._std_weight_position * mean[3],
        1e-1,
        self._std_weight_position * mean[3]
        ]
    # 初始化噪声矩阵R
    innovation_cov = np.diag(np.square(std))
    # 将均值向量映射到检测空间，即Hx'
    mean = np.dot(self._update_mat, mean)
    # 将协方差矩阵映射到检测空间，即HP'H^T
    covariance = np.linalg.multi_dot((self._update_mat, covariance, self._update_mat.T))
    return mean, covariance + innovation_cov

def update(self, mean, covariance, measurement):
    # 计算 Kalman Filter的Correction Step==>Update Step
    # 将mean和covariance映射到检测空间，得到Hx'和S
    projected_mean, projected_cov = self.project(mean, covariance)
    # 矩阵分解
    chol_factor, lower = scipy.linalg.cho_factor(projected_cov, lower=True, check_finite=False)
    # 计算卡尔曼增益K
    kalman_gain = scipy.linalg.cho_solve((chol_factor, lower), np.dot(covariance, self._update_mat.T).
    # z - Hx'
    innovation = measurement - projected_mean
    # x = x' + Ky
    new_mean = mean + np.dot(innovation, kalman_gain.T)
    # P = (I - KH)P'
    new_covariance = covariance - np.linalg.multi_dot((kalman_gain, projected_cov, kalman_gain.T))
    return new_mean, new_covariance
```

Figure 5. Implement of Kalman Filter

3.DeepSort workflow

The core idea of the tracking method based on trajectory prediction is:

- Use Yolo to detect the target in the first frame
- Pass the Kalman filter to predict the trajectory state of the next frame (u, v, r, h)
- use Yolo detects the target in the second frame
- Match the detected target in the second frame with the predicted trajectory state
- Repeat this process

where: Yolo is used to detect the target frame by frame, and Kalman filter is used for prediction. The Hungarian algorithm is used to associate the data of the preceding and following frames[5].

```python
def predict(self):
    for track in self.tracks:
        track.predict(self.kf)

def update(self, detections):
    # Run matching cascade.
    matches, unmatched_tracks, unmatched_detections = \
        self._match(detections)

    # Update track set.
    for track_idx, detection_idx in matches:
        self.tracks[track_idx].update(
            self.kf, detections[detection_idx])
    for track_idx in unmatched_tracks:
        self.tracks[track_idx].mark_missed()
    for detection_idx in unmatched_detections:
        self._initiate_track(detections[detection_idx])
    self.tracks = [t for t in self.tracks if not t.is_deleted()]

    # Update distance metric.
    active_targets = [t.track_id for t in self.tracks if t.is_confirmed()]
    features, targets = [], []
    for track in self.tracks:
        if not track.is_confirmed():
            continue
        features += track.features
        targets += [track.track_id for _ in track.features]
        track.features = []
    self.metric.partial_fit(
        np.asarray(features), np.asarray(targets), active_targets)
```

```python
def _match(self, detections):
    # 主要功能是进行匹配，找到匹配的，未匹配的部分
    def gated_metric(tracks, dets, track_indices, detection_indices):
        # 功能：用于计算track和detection之间的距离，代价函数，需要使用在KM算法之前
        features = np.array([dets[i].feature for i in detection_indices])
        targets = np.array([tracks[i].track_id for i in track_indices])
        # 1. 通过最近邻计算出代价矩阵 cosine distance
        cost_matrix = self.metric.distance(features, targets)
        # 2. 计算马氏距离，得到新的状态矩阵
        cost_matrix = linear_assignment.gate_cost_matrix(self.kf, cost_matrix, tracks, dets,

        return cost_matrix

    # 划分不同轨迹的状态
    confirmed_tracks = [i for i, t in enumerate(self.tracks) if t.is_confirmed()]
    unconfirmed_tracks = [i for i, t in enumerate(self.tracks) if not t.is_confirmed()]

    # 进行级联匹配，得到匹配的track、不匹配的track、不匹配的detection
    '''
    # gated_metric->cosine distance
    # 仅仅对确定态的轨迹进行级联匹配
    matches_a, unmatched_tracks_a, unmatched_detections = \
        linear_assignment.matching_cascade(
            gated_metric, self.metric.matching_threshold, self.max_age,
            self.tracks, detections, confirmed_tracks)

    # 将所有状态为未确定态的轨迹和刚刚没有匹配上的轨迹组合为iou_track_candidates，
    # 进行IoU的匹配
    iou_track_candidates = unconfirmed_tracks + [
        k for k in unmatched_tracks_a if
        self.tracks[k].time_since_update == 1]
    # 未匹配
    unmatched_tracks_a = [
        k for k in unmatched_tracks_a if
        self.tracks[k].time_since_update != 1]
    '''
```

Figure 6. Implement of DeepSort

2.3 Distance Estimation

2.3.1 Method Selection

Monocular and binocular ranging (Stereo Vision) are two distance estimation methods. Compared to Monocular ranging, Stereo Vision could achieve higher accuracy. Besides, multiple cameras can be used to cover different ranges of scenes and solve the problem of recognition clarity at different distances at one time. Thus I choose binocular ranging as distance estimation method.

2.3.2 Implement of Stereo Vision

Stereo Vision, also called binocular stereo vision, its research can help us better understand how human eyes perceive depth. The general process of binocular ranging is:

- Dual target determination
- Distortion Elimination and Stereo correction
- Stereo matching and disparity map calculation
- Depth calculation/3D coordinate calculation

1. Dual target determination

The goal of dual target determination is to obtain the internal parameters, external parameters and distortion coefficients of the left and right cameras. Since these parameters are given in the test file, I omitted this step.

2. Distortion Elimination and Stereo correction

The light passing through the camera's optical system often cannot be projected onto the sensor in an ideal situation, that is, so-called distortion will occur. The purpose of stereo correction is to mathematically transform the left and right views shot in the same scene so that the two imaging planes are parallel to the baseline, and the same point is located in the same row in the left and right images[6].

In OpenCV, “cv2.stereoRectify” can be directly applied to return the correction matrix. “cv2initUndistortRectifyMap” is used to calculate the distortion and correct the conversion relationship. And “cv2.remap” is to match two image after the Stereo correction.

```python
# 获取畸变校正和立体校正的映射变换矩阵、重投影矩阵
# @param：config是一个类，存储着双目标定的参数:config = stereoconfig.stereoCamera()
def getRectifyTransform(height, width, config):
    # 读取内参和外参
    left_K = config.cam_matrix_left
    right_K = config.cam_matrix_right
    left_distortion = config.distortion_l
    right_distortion = config.distortion_r
    R = config.R
    T = config.T

    # 计算校正变换
    height = int(height)
    width = int(width)
    R1, R2, P1, P2, Q, roi1, roi2 = cv2.stereoRectify(left_K, left_distortion, right_K, right_distortion, (width, he

    map1x, map1y = cv2.initUndistortRectifyMap(left_K, left_distortion, R1, P1, (width, height), cv2.CV_32FC1)
    map2x, map2y = cv2.initUndistortRectifyMap(right_K, right_distortion, R2, P2, (width, height), cv2.CV_32FC1)

    return map1x, map1y, map2x, map2y, Q


# 畸变校正和立体校正
def rectifyImage(image1, image2, map1x, map1y, map2x, map2y):
    rectifyed_img1 = cv2.remap(image1, map1x, map1y, cv2.INTER_AREA)
    rectifyed_img2 = cv2.remap(image2, map2x, map2y, cv2.INTER_AREA)

    return rectifyed_img1, rectifyed_img2
```

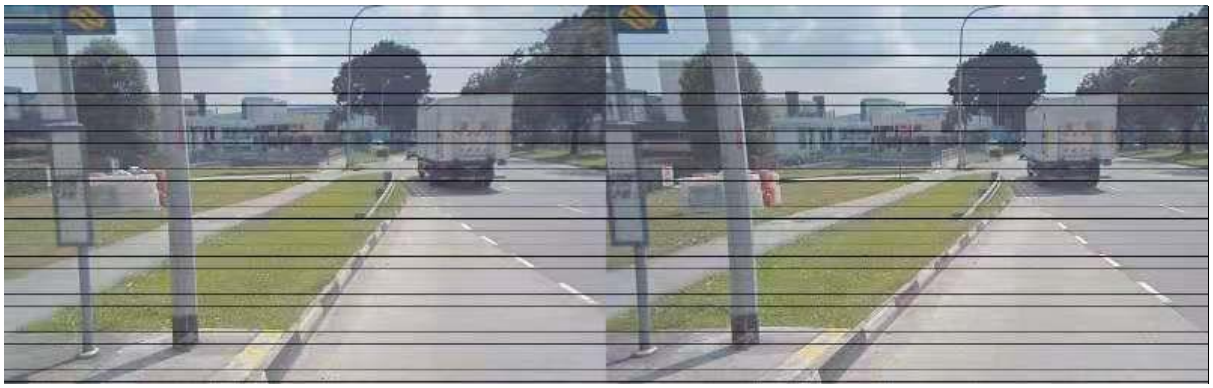

Figure 7. Stereo correction visitation

3. Stereo matching and disparity map calculation

The purpose of stereo matching is to find the corresponding point for each pixel in the left image in the right image, so that the parallax can be calculated: The calculation process can be divided into the following stages: 1) matching cost calculation,2) cost aggregation, 3) disparity optimization, and 4) disparity refinement [6].

Commonly used stereo matching methods can be basically divided into two categories: local methods, and global methods. local methods are computationally expensive and the matching quality is relatively low. The global method omits cost aggregation and uses the method of optimizing the energy function and the matching quality is higher. Thus I choose one global matching algorithm (SGBM) in OpenCV as matching method.

```python
# 视差计算
def stereoMatchSGBM(left_image, right_image, down_scale=False):
    # SGBM匹配参数设置
    if left_image.ndim == 2:
        img_channels = 1
    else:
        img_channels = 3
    blockSize = 3
    paraml = {'minDisparity': 0,
              'numDisparities': 128,
              'blockSize': blockSize,
              'P1': 8 * img_channels * blockSize ** 2,
              'P2': 32 * img_channels * blockSize ** 2,
              'disp12MaxDiff': 1,
              'preFilterCap': 63,
              'uniquenessRatio': 15,
              'speckleWindowSize': 100,
              'speckleRange': 1,
              'mode': cv2.STEREO_SGBM_MODE_SGBM_3WAY
              }

    # 构建SGBM对象
    left_matcher = cv2.StereoSGBM_create(**paraml)
    paramr = paraml
    paramr['minDisparity'] = -paraml['numDisparities']
    right_matcher = cv2.StereoSGBM_create(**paramr)
```

Figure 8. Implement of Stereo matching

After the disparity map is obtained, the pixel depth can be calculated, the formula is as follows.

$$Z = \frac{b * f}{X_R - X_T} = \frac{b * f}{d}$$

Where f is the focal length (pixel focal length), b is the baseline length, d is the parallax, and is the column coordinates of the principal points of the two cameras.

```python
def depthTry(dismap,fx,baseline):
    (height,width) = dismap.shape
    depthmap = np.zeros((height,width))
    for y in range(height):
        for x in range(width):
            if dismap[y][x] != 0:
                depthmap[y][x] = fx * baseline/dismap[y][x]
    return depthmap
```

Figure 9. Implement of distance calculation

2.4 Radar Map Generation

Here is the additional task. I would like to generate radar map based on the output of object tracking and distance estimation, which could clearly observe the traffic conditions in the road in the real time.
The output Radar map is in the World coordinate system with the shape of 200 * 200 pixels represent 100 * 100 meter in the real world (1 pixel = 0.5m). The camera is at the bottom center and each rectangular represent one identified object.

The generation of one radar map contains three steps:

1. Given the centers of objects from the tracking output, calculate the angle of object and horizontal line in the pixel coordinate system
2. Combined the distance and angle, generate the real-world coordinate of each object
3. Mark the object location in the radar map

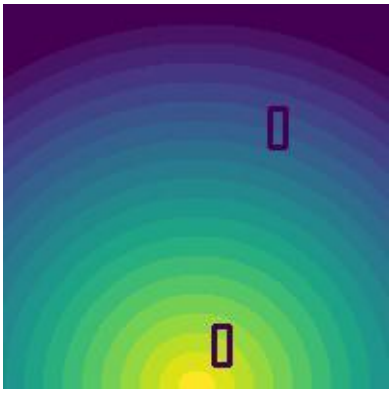

Figure 10. Radar map example

3. Summary of problems

1. Class label could not be matched to the objects.

Reason: The reason of this problem is that the bounding box (location) of the objects are drew from the result of tracker. However, the class label is generated from the detector. Thus cause the matching of label and location is difficult.

Solution: I labelled the bounding box with both class name and location from the detector. Since the location in the detector output is close to the one from tracker. The only problem is the label might absent in some frames without the tracking. But it didn't have big impact on the result.

2. Absence of rotation and shift matrix

Reason: Rotation and shift matrix are in need for the input of cv2.stereoRectify to generate Q matrix and other rectified matrixes. The baseline and focal length are necessary for the calculation of distance, which can be read from the Q and shift matrix separately.

Solution: The rotation and shift matrix can be extracted from projection matrix with QR decomposition. After doing through the mathematical equations, with the help of QR decomposition

methods in MATLAB, I wrote the mathematical script and successfully extracted R,T matrix.

```
K:
914.755 -8.98126e-12 500.581
-1.7053e-13 914.755 323.874
-5.55112e-17 -2.77556e-17 1

R:
1 4.90931e-15 -1.47775e-14
-4.90931e-15 1 -9.52108e-15
1.47775e-14 9.52108e-15 1

T:
-0.500346
1.21295e-31
0
```

Figure 11. Result of rotation and shift matrix extraction

3. Stereo Matching Problem

Reason: The distance estimation replies on the high accuracy of disparity map, which reply on the result on the stereo matching. However, it is difficult to get good stereo matching. It might because the object in images is too far, and such long distance may have bad impact on the stereo matching results.

Solution: I first tried different matching methods (SAD,SGBM,BM). Among these algorithms, although the SGBM perform best, still it does not work for distant objects. Then I tuned the parameters of the methods. And gradually improve the results of disparity map.

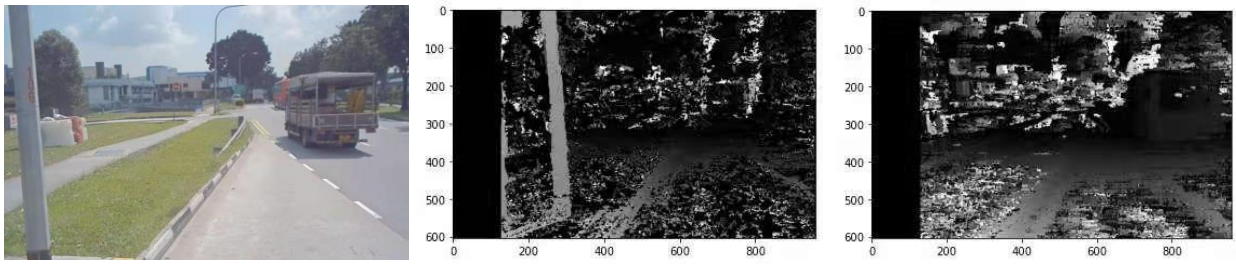

Figure 12. Improvement of stereo matching

4. Distance problem

Reason: Initially I used distance of the center of objects as distance estimation result. However, from the result, the distance is very unstable and the value changes largely across images.

Solution: Instant use the center distance, I added a filter of the distances of whole objects. I first collected all the distance values of with in object bounding box and sorted the values. After that I only keep the median 50% distances and calculate their average values as the distance result.

Figure 13a. Filter area visitation

```
# bbox_center_point(x,y)
center = (int(((bbox[0]) + (bbox[2])) / 2), int(((bbox[1]) + (bbox[3]))

# calculate the average distance
dis_all = depth_map[int(bbox[1]):int(bbox[3]),int(bbox[0]):int(bbox[2])]
dis_all = dis_all.flatten()
dis_all.sort()
dis_sort = [k for k in dis_all if k != 0]
start = int(len(dis_sort)*0.25)
ave_dis = np.mean(dis_sort[start:-start])
```

13b. Implement of distance filter

4. **Result**

49 objects (persons and cars) are identified in the test dataset as the result. The tracking ID, class label and distance of each object are marked in the output video. The corresponding radar map is generate as well providing great visitation of real time road condition.
From the result the objects near the camera are more stable, whereas the object far from the camera are more likely encounter lost tracking problem and thus the bounding boxes of them are jumped in the frames.

5. Future Work

Due to the time limitation, there are some further steps have not implemented of this project.
1. Evaluation of model

- Use the average accuracy index to evaluate the target detector.
- The precision/recall rate (PR) curve emphasizes the accuracy of the detector under different recall levels
- Use mAP to evaluate the target detector

2. Improvement of tracking result
As statement previously the detector does not perform well for the distant objects. This problem may be solved by:

- Using part of test dataset to train the detector
- Adjusting the parameters of tracker

3.Improve the accuracy of distance estimation
The current accuracy of distance estimation is not so stable. The accuracy can be further improved by adjusting the parameters in the stereo matching methods to drive a better disparity map.

4. Refine the radar map
Since in the current pixel to meter ratio in radar map (0.5m/pixel) is too large, the objects near the camera are too close to each other. (eg. the distance of two cars is 5m which is only 10 pixels in the radar map). The size of radar map needs to be adjusted to have a better visitation of real time road condition.

Reference

[1] Redmon, Joseph and Ali Farhadi. “YOLOv3: An Incremental Improvement.” ArXiv abs/1804.02767 (2018): n. pag.
[2] Bewley, Alex, et al. "Simple online and realtime tracking." 2016 IEEE International Conference on Image Processing (ICIP). IEEE, 2016.
[3] Wojke, Nicolai, Alex Bewley, and Dietrich Paulus. "Simple online and realtime tracking with a deep association metric." 2017 IEEE International Conference on Image Processing (ICIP). IEEE, 2017.
[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] Danelljan, Martin, et al. “Accurate scale estimation for robust visual tracking.” British Machine Vision Conference, Nottingham, September 1-5, 2014. BMVA Press, 2014.
[6] W. Jang, C. Je, Y. Seo, and S. W. Lee. Structured-Light Stereo: Comparative Analysis and Integration of Structured-Light and Active Stereo for Measuring Dynamic Shape. Optics and Lasers in Engineering, Volume 51, Issue 11, pp. 1255-1264, November, 2013.
[7] Lazaros, Nalpantidis; Sirakoulis, Georgios Christou; Gasteratos1, Antonios (2008). "Review of Stereo Vision Algorithms: From Software to Hardware". International Journal of Optomechatronics. 2 (4): 435–462. doi:10.1080/15599610802438680. S2CID 18115413.