Multi-Object tracking and Trajectory Prediction for Autonomous Vehicles

Vinit Awale
Department of Electrical Engineering
Indian Institute of Technology, Bombay
Mumbai, India
awale.vinit@gmail.com

Prof. Naveen Arulselvan
AI & Robotics Technology Park
ARTPARK, IISC Bangalore
Karnataka, India
naveen@artpark.in

***Abstract*—This document specifies the implementation details of the perception module using cameras of a self-driving car. This project was completed as a part of the Summer Internship program at ARTPARK, IISC Bangalore, under the mentorship of Prof. Naveen Arulselvan. The perception module consists of multi-object detection, tracking and trajectory prediction. The multi-object detection is based on the *YOLO v5* algorithm [3]. We have used the *Deep Sort* algorithm for multi-object tracking based on the paper "Simple online and realtime tracking with a deep association metric". The trajectory prediction module is implemented using *PEC Net* based on the paper "It is not the journey but the destination: Endpoint conditioned trajectory prediction" [5]. Python and PyTorch framework have been used for the code implementation.**

***Index Terms*—Object detection, multi-object tracking, trajectory prediction, YOLOv5, Deep Sort, PEC Net, Autonomous vehicles**

I. Introduction

Perception is a central problem for any autonomous agent, be it humans, robots or self-driving vehicles. This module helps for a smoother and more reliable control of the car using the path-planning module of the autonomous agent. It can also aid in pose estimation. For our project, we have included the following sub-modules for the perception:

- Multi-object detection using the *YOLOv5* algorithm.
- Multi-object tracking using the *Deep Sort* algorithm.
- Trajectory prediction using the *PEC Net* algorithm.

Object detection in the context of autonomous driving refers to detecting the objects present in the scene (making use of the camera sensors on the autonomous vehicle) by making bounding boxes surrounding the detected objects. This is followed by identifying the class of the objects. The family of YOLO (You Only Look Once) models are the most popular object detection models for autonomous driving. The YOLOv5 model is a state-of-the-art object detection model that is capable of detecting 80 classes of objects. The model is trained on the MS COCO dataset, containing over 1.2 million images and over 20,000 bounding boxes for the 80 classes of objects.

Multi-object tracking refers to the problem of tracking the objects detected across frames. For this project, we are implementing the *Deep Sort* algorithm for tracking the bounding boxes. Simple Online Realtime Tracking *SORT* is an approach of multi-object tracking using simple and effective algorithms such as Kalman Filter. Including an association metric (using deep learning) for the detected object across frames leads to a more robust and accurate multi-object tracking called Deep Sort.

The problem of trajectory prediction (as is already apparent from the name) involves predicting the agents detected by the YOLOv5 model. PEC Net uses an encoder-decoder architecture for predicting the agents detected by the YOLOv5 model. The encoder is a neural network that encodes the input image into a vector of fixed size. The decoder is a fully connected neural network that decodes the vector into a high-dimensional vector. For the implementation of PEC Net, we need an aerial view of the scene. However, we are working with datasets of camera images taken in the ego-centric view in our implementation. Hence for a complete end-to-end perception pipeline, we need to move the detections of our object detection module to birds-eye view. This is a part of future work.

The code for the project is available at https://github.com/TheShiningVampire/PERCEPTION_ARTPARK

II. Background and Related Works

A. Object Detection

The problem of object detection and object tracking is quite well known in the field of computer vision. Modern Object detectors are composed of two main components:

- The backbone: This is the part of the detector that is responsible for extracting the features of the objects in the image using a deep convolutional neural network.
- The head: This is the part of the detector that is responsible for classifying the objects in the image.

On the basis of head component, we have two main approaches for object detection:

- Two stage Object Detector: This approach leads to high localization and object detection accuracy of the results.
  Example: R-CNN, fast R-CNN, faster R-CNN, R-FCN, Libra R-CNN, etc

- One stage Object Detector: This approach leads to a higher reference speed.
  Example: SSD, YOLO, RetinaNet, etc

We have chosen the One-stage Object Detector approach since we need a higher reference speed for autonomous driving applications. Specifically, we have used YOLO detection for our application.

B. Multi-Object Tracking

Multi-object tracking methods usually include a tracking network and a detection network. The tracking network is responsible for updating the position and orientation of the object in the scene. The detection network is responsible for detecting the object in the scene. The tracking network is responsible for the data association across the frames.
The detections from the object detector can be processed in one of the following ways:

- Batch method: In this approach, the data association problem is solved by storing the image frames with detections in a batch. The batch is then processed one by one.

- Online method: In this approach, the data association problem is solved by processing an image frame with a detection one at a time.

Practically, the batch method performs better than online methods for the tracking network. However, the batch method is computationally expensive, which means that the tracking network needs to process the entire batch of detections before starting the next frame. Hence, this approach is not viable for our project. Therefore, we have chosen the online method for our project.
Famous object tracking methods include:
KCF tracker, SORT tracker, MOSSE, KCF+SORT, MOSSE+KCF, MOSSE+KCF+SORT, MedianFlow, etc.

For our project, we have chosen the Deep SORT, an online multi-object tracking algorithm that incorporates a deep association metric for the tracking network.

C. Trajectory Prediction

Amongst all the tasks mentioned, the trajectory prediction is the most challenging one. The trajectory prediction is the task of predicting the future trajectory of the object. The challenges in trajectory prediction are:

- Destination is an intrinsic parameter of the agent.
  The agents in the surrounding have some end goal to be achieved, however, we can never know the exact end goal of the agent until it is achieved.
- Agent Environment interactions.
  The agent interacts with the environment and hence the trajectory taken is affected by it.
- Agent-agent interactions.
  The agents also interact among themselves and hence the trajectory taken is affected in such a way that all the agents move in a socially compliant manner.
- Personal preferences.
  Apart from all the challenges mentioned above, the agent has some personal preferences that affect the trajectory taken.

However, there have been several approaches for trajectory prediction in the past. These include:

- Social Force Model (1998)
  In this model, interactions among the agents and the environment are taken into account by modelling them as forces. And the trajectory of a particular agent is predicted based on the forces acting on it by using Newton's laws.
- LSTM In this model the output of each agent is obtained by a LSTM network. The LSTM network is trained to predict the future trajectory of the agent based on the past trajectory of the agent.
- Social LSTM This model was an improvement over the basic LSTM model. In this model social interactions were also incorporated in the LSTM network.

For this project we will be using the PEC Net architecture (abbreviation for the term Predicted Endpoint Conditioned Network) for trajectory prediction.

III. Datasets

Autonomous driving is an emerging field; hence there are limited datasets that can be used for training and testing which would resemble the real world. The datasets that we have used for our project are:

- Lyft level 5 Prediction dataset [4]
  This is a self-driving dataset for motion prediction, containing over 1,000 hours of data. This was collected by a fleet of 20 autonomous vehicles along a fixed route in Palo Alto, California, over a four-month period. It consists of 170,000 scenes, where each scene is 25 seconds long and captures the perception output of the self-driving system, which encodes the precise positions and motions of nearby vehicles, cyclists, and pedestrians over time.
- KITTI dataset [1]
  The dataset comprises of Raw (unsynced+unrectified) and processed (synced+rectified) color stereo sequences (0.5 Megapixels, stored in png format), captured and synchronized at 10 Hz. This dataset was only used for testing the perception modules of our project.
  Along with camera sequences, the dataset also contains 3D Velodyne point clouds, 3D GPS/IMU data and 3D object tracklet labels. All of these will be useful for the proposed future works of the project.

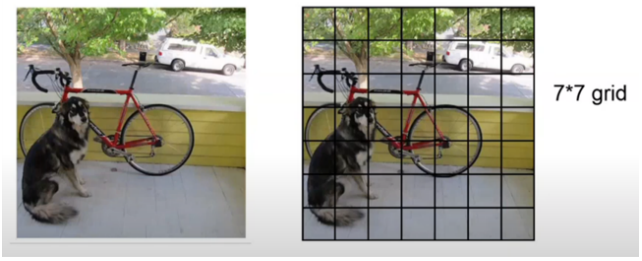
IV. Procedure, Experiments and Results

A. Object detection

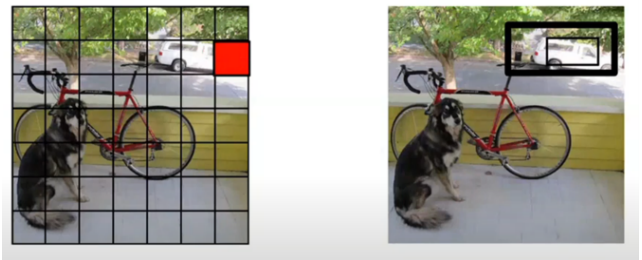
1) ***YOLO algorithm****:* YOLO is an abbreviation for the term 'You Only Look Once'. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images. YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects.

Working of YOLO*:* In this subsection we will discuss the working of YOLO algorithm in brief.

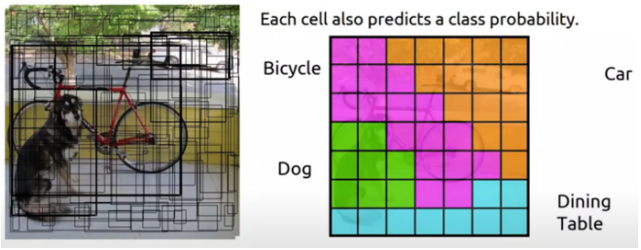
- First split the image into smaller S x S grid cells. This S is a hyper-parameter that can be tuned to improve the performance of the algorithm. However, the larger the S, the more computationally expensive the algorithm will be but the results will be more accurate. In the original paper, the S is set to 7 (as shown in the following figure).



- For each grid cell make predictions of B bounding boxes $(x_{centre}, y_{centre}, width, height)$ and confidence for each bounding box (P[object]). The value of B was set to 2 in the original paper. Here, multiple bounding boxes are predicted for each grid cell. This helps in the case when a single grid cell has the center of multiple objects. However, as the number of grid cells (B) increases, the probability that multiple objects are detected in a single grid cell decreases.

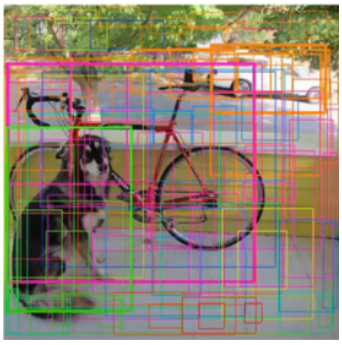


- Along with this the model also predicts conditional class probability (out of the C classes) i.e. $P[class_i|object]$ for each grid cell.



- The model then predicts the class label for each bounding box using Bayes Theorem

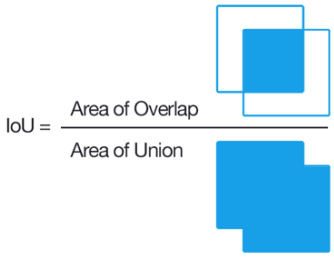
$$P[class_i] = P[class_i|object] * P[object]$$



From the above image we can see that there are several bounding boxes detected for each class. Now, to get rid of these bounding boxes, we need to apply non-maximal suppression.

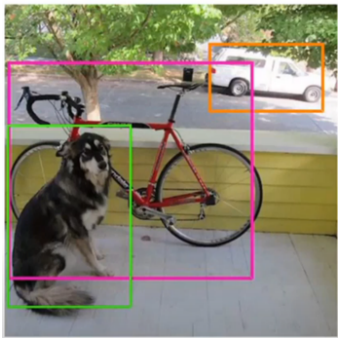
Non-maximal suppression*:* The steps involved in applying non-maximal suppression are as follows:

- Find the IoU (Intersection over Union) for all the bounding boxes of the same class.

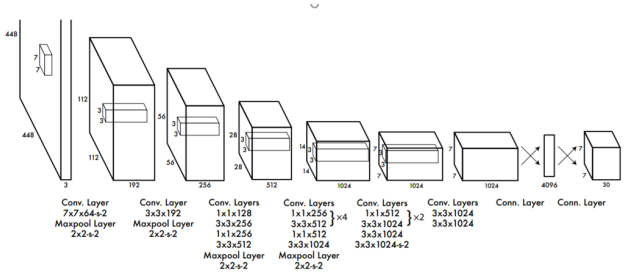


- Keep the bounding box with maximum class probability and discard other bounding boxes with IoU > threshold.

Hence, using this we get rid of the multiple bounding boxes of the same class and only preserve the one with the highest class probability. Finally, the result we obtain is shown in the following figure.



Architecture of YOLO: The architecture of YOLO is shown in the following figure.



The architecture of YOLO comprises of 24 convolutional layers followed by 2 fully connected layers. In the architecture, alternating 1×1 convolutional layers reduce the feature space from the preciding layers. In the original paper the architecture was trained on the ImageNet dataset with all the images resized to 224 x 224 pixels.

Loss Function: The loss function used while training the YOLO model is as follows:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

where, $\mathbb{1}_i^{obj}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{obj}$ denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.

The first two terms in the loss function correspond to the localization loss. The next two terms correspond to the confidence loss. The last term is the classification loss.

2) ***YOLO v5***: For our project we have used pretrained YOLOv5 model, which is an improve version of the basic YOLO architecture. Since, we decided to implement the entire module using PyTorch, we decided to change the object detection sub module to YOLOv5 written in PyTorch from YOLOv3 which has been implemented using the Darknet framework. Also, according to various studies [2] regarding the comparison of the two models, it has been found that YOLO v5 is the most optimal model for deployment due to its speed and accuracy. Following is the result of one frame on the Lyft dataset when the YOLOv5 algorithm was implemented on it.

Fig. 1: Original camera frame from the Lyft Level 5 dataset

Fig. 2: Result of YOLO v5 object detection

When implemented on the KITTI dataset, the results we as follows:

Fig. 3: Original camera frame from the KITTI dataset

Fig. 4: Result of YOLO v5 object detection

FPS obtained: We tested the Object detection module on KITTI dataset and the Lyft Level 5 dataset. Also, two GPUs were used for testing. One being Nvidia Geforce RTX 3060 (mobile) GPU and the other being Nvidia Tesla T4 GPU (server). The following table shows the FPS obtained on the KITTI dataset.

	LYFT LEVEL 5 DATASET		KITTI DATASET	
	AVG FPS	MIN FPS	AVG FPS	MIN FPS
NVIDIA GEFORCE RTX 3060	22.19	14.33	25.86	16.13
NVIDIA TESLA T4	30.15	21.88	31.35	19.27

B. Multi-Object Tracking

For this project, we have implemented Deep SORT algorithm for multi-object tracking. The algorithm is based on the paper [6] which is a state-of-the-art algorithm for multi-object tracking. Now, let us look at the working of the algorithm.

Working Deep Sort: The tasks included in the implementation of Deep SORT are as follows:

- OBJECT DETECTION
  We have already implemented YOLO v5 algorithm for the object detection tasks
- ESTIMATION OF THE BOUNDING BOXES OF THE OBJECTS IN IMAGE SPACE
  We assume a very general tracking scenario where the camera is uncalibrated and where we have no ego-motion information available. We use a Standard Kalman Filter with constant velocity motion and linear observation model on a eight dimensional space $(x, y, \gamma, h, \dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$
- ASSIGNMENT PROBLEM
  We solve the association problem using the Hungarian Algorithm. For that we need to form a cost matrix using an association metric. Deep Sort incorporates two association metrics which are as follows:
  - Motion Information
    To incorporate the motion information, we use the Mahalanobis distance between the predicted and the newly measured bounding boxes.

    $$d^1(i,j) = (d_j - y_i)^T \Sigma^{-1} (d_j - y_i)$$

    where, d_j is the predicted bounding box and y_i is the newly measured bounding box. The Σ matrix is the covariance matrix of the motion model. The Mahalanobis distance is a robust distance metric in the case when there is correlation between the n-dimensional vectors.
  - Apperance Information
    For each bounding box detection dj we compute an appearance descriptor rj with $\|r_j\| = 1$. To find these appearance descriptors we use a CNN architecture that has been trained on a large-scale person re-identification dataset that contains over 1,100,000 images of 1,261 pedestrians.

Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

Fig. 5: CNN architecture used for feature extraction

Further, we keep a gallery (R_k) of past 100 associated appearance descriptor for each track (k). Then, our second metric measures the smallest cosine distance between the ith track and jth detection in appearance space.

$$d^2(i,j) = \min 1 - r_j^T r_k^{(}i) | r_k^{(}i) \in R_k$$

where, r_j is the appearance descriptor of the jth detection and $r_k^{(}i)$ is the ith track in the gallery.
To build the association problem we combine both metrics using a convex combination.

$$c_{ij} = \alpha d^1(i,j) + (1-\alpha) d^2(i,j)$$

We call an association admissible if it is within the gating region of both metrics:

$$b_{ij} = \prod_{m=1}^{2} b_{i,j}^{(m)}$$

where, $b_{i,j}^{(m)}$ is the gating region of the mth metric.

$$b_{i,j}^{(m)} = \mathbb{1}[d^m(i,j) \leq t^{(m)}]$$

where, $t^{(m)}$ is the threshold of the mth metric.

- TRACKING HANDLING
  For each track k we count the number of frames since the last successful measurement association . This counter is incremented during Kalman filter prediction and reset to 0 when the track has been associated with a measurement. Tracks that exceed a predefined maximum age A_{max} are considered to have left the

scene and are deleted from the track set. New track hypotheses are initiated for each detection that cannot be associated to an existing track. These new tracks are classified as tentative during their first three frames.

Results: When implemented on the sample dataset, the results we as follows:

Fig. 6: Original camera frame from the sample dataset

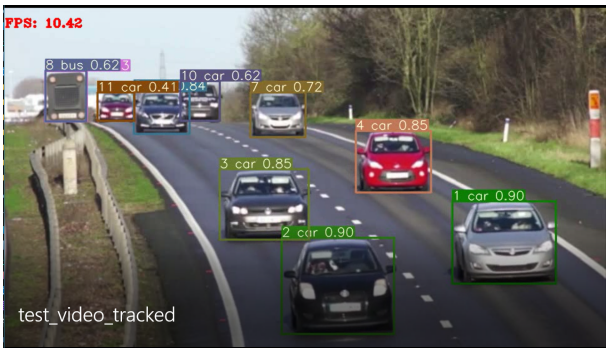


Fig. 7: Result of Deep Sort tracking

When tested on the KITTI dataset, the results we as follows:

Fig. 8: Original camera frame from the KITTI dataset

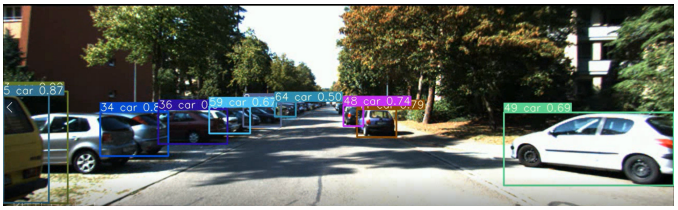


Fig. 9: Result of Deep Sort tracking on KITTI dataset

When tested on the Lyft Level 5 dataset, the results we as follows:

Fig. 10: Original camera frame from the Lyft Level 5 dataset

Fig. 11: Result of Deep Sort tracking on Lyft Level 5 dataset

	LYFT LEVEL 5 DATASET		KITTI DATASET	
	AVG FPS	MIN FPS	AVG FPS	MIN FPS
NVIDIA GEFORCE RTX 3060	12.96	8.72	13.14	7.21
NVIDIA TESLA T4	14.19	11.69	14.04	9.36

Fig. 12: FPS obtained using Deep Sort

FPS obtained: We can easily observe that the FPS obtained after object tracking is roughly the half of the FPS obtained after simple object detection.

V. Trajectory Prediction

A. Modelling the problem

Given sequence of observations:

$$X_i^t = (x_i^t, x_i^t) \qquad for \quad t = 1,2,3,...,T^{obs}$$

Tasks to be done:

- Make prediction of the trajectory until the next T^{pred} frames.

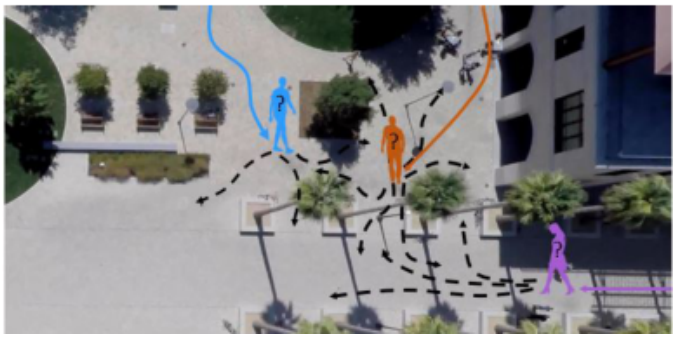
$$\hat{Y}_i^t = \mathscr{F}(X_i^t) \qquad for \quad t = T^{obs},...,T^{pred}$$

Such that it is as close as possible to the actual trajectory.

$$Y_i^t = \mathscr{F}(X_i^t) \qquad for \quad t = T^{obs}, ..., T^{pred}$$

Approach: The approach used by PEC Net for trajectory prediction can be understood from the following images:

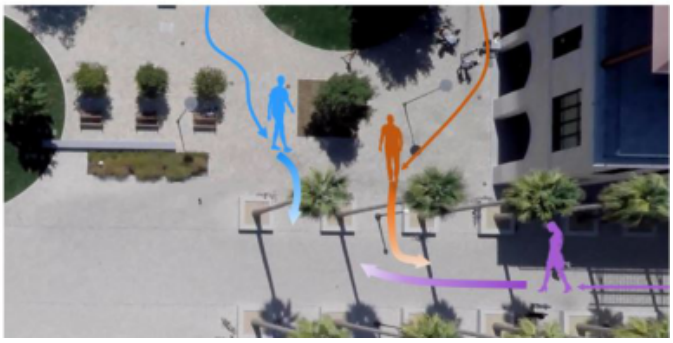
- We know the past trajectory of the agent as shown below and we attempt to predict the future trajectory of the agent.



- We first try to find the multi modal distribution of the agent's possible endpoint. (We are basically trying to model the intent of the agent in this step)

- From the generated distribution we sample some of the possible endpoint and then conditioned on the past trajectory, we try to predict the future trajectory.

- Further we use a social pooling layer so that all the predicted trajectories of the agents in the scene are socially compliant.



Architecture of PEC Net: The architecture of PEC Net is as follows:

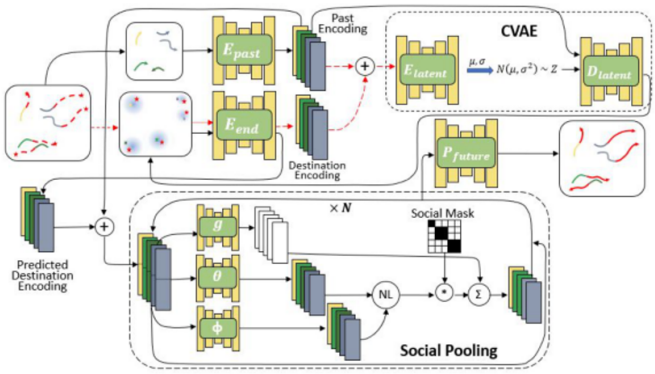


Fig. 13: Architecture of PEC Net

The various blocks in the PEC Net architecture can be understood as:

- **While training**
  - We first encode past trajectory and Ground truth information
  - Concatenate and Encode to CVAE latent space with the encoder.
  - Decode the sampled latent vector with CVAE decoder conditioned on the past trajectory encoding
  - Re-encode estimated endpoints and form pool vectors for neighbours.
  - Pool context for neighbours with social pooling module.
  - Jointly predict the future positions for all the neighbours with social context pooled information
- **While inference**
  Since while inference we do not have the information of the ground truth trajectory all the blocks including the ground truth information are removed. In fig. 13, these blocks correspond to the ones connected with a red arrow. All of these block are not used during Inference.
  The exact steps for inference are as follows:
  - Encode past trajectory

- Decode the sampled latent vector with CVAE decoder conditioned on the past trajectory encoding
- Re-encode estimated endpoints and form pool vectors for neighbours.
- Pool context for neighbours with social pooling module.
- Jointly predict the future positions for all the neighbours with social context pooled information

Please refer to the original paper [5] for the mathematical details.

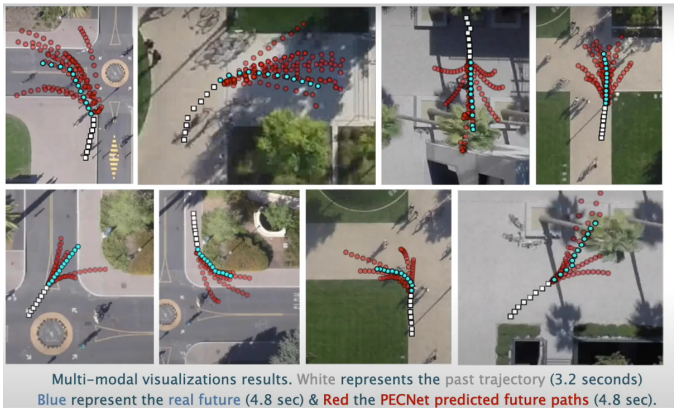


Fig. 14: Results of PEC Net

Results: The above are the results obtained using PEC Net on the Stanford Drone dataset. We have been unable to use PEC Net on our datasets since our datasets consists of images from the ego-centric, however, PEC Net requires them to be in Aerial perspective.

VI. Applications and Future work

Firstly, since this project was a part of remote summer internship, none of the implementations have been tried on a real autonomous system. Hence, this is to be done in the future work for this project. As mentioned earlier the PEC Net architecture hasn't been used on our datasets since our datasets consists of images from the ego-centric, however, PEC Net requires them to be in Aerial perspective. Hence, projecting our obtained detections to the Birds-eye view is view is one of the possible way to overcome this problem. Also, we have only used camera as our primary sensor. Using other sensors like lidar or radar is also possible. Fusing the information from multiple sensors is also possible. The final aim of such a pipeline is to make the agent more social and more autonomous. This can be achieved by making an intent prediction subsystem, which may use Reinforcement Learning techniques to learn the intent of the agent.

With all the possible directions in which this project can be carried forward, there are also several applications to the tasks that have been accomplished in this project. The tasks accomplished in this project will supplement other subsystems of a self-driving car such as localization and path-planning. Also, it can be used for surveillance applications.

VII. Acknowledgment

I would like to thank ARTPARK to present me with the opportunity of working on this project. Also, I will like to thank Prof. Naveen Arulselvan for guiding me through this project.

References

[1] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[2] Rakkshab Varadharajan Iyer, Priyansh Shashikant Ringe, and Kevin Prabhulal Bhensdadiya. Comparison of yolov3, yolov5s and mobilenet-ssd v2 for real-time mask detection. 2021.

[3] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, Adam Hogan, lorenzomammana, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Francisco Ingham, Frederik, Guilhen, Hatovix, Jake Poznanski, Jiacong Fang, Lijun Yu , changyu98, Mingyu Wang, Naman Gupta, Osama Akhtar, PetrDvoracek, and Prashant Rai. ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements, October 2020.

[4] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Level 5 perception dataset 2020. https://level-5.global/level5/data/, 2019.

[5] Karttikeya Mangalam, Harshayu Girase, Shreyas Agarwal, Kuan-Hui Lee, Ehsan Adeli, Jitendra Malik, and Adrien Gaidon. It is not the journey but the destination: Endpoint conditioned trajectory prediction. In *European Conference on Computer Vision*, pages 759–776. Springer, 2020.

[6] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE, 2017.