HW1 Report

1. Implement
    a. get_relative_pos

```
# TODO
'''
Extracts the translation vectors from both matrices and subtracts them element-wise.
This gives the relative position of the other vehicle in global coordinates.
'''
relative_pos = vehicle_matrix[:3, 3] - ego_matrix[:3, 3]

# This is done to get the inverse rotation, effectively transforming from the global coordinate system to the ego coordinate system.
rot = ego_matrix[:3, :3].T

# It rotates the relative position from the global coordinate system into the ego coordinate system.
relative_pos = rot @ relative_pos
return relative_pos
```

    b. extract_yaw_from_matrix

```
# TODO

# Extract submatrix represents the rotation component of the transformation.
rotation_matrix = np.array(matrix)[:3, :3]

# Extract yaw using the rotation matrix
yaw = normalize_angle(math.atan2(rotation_matrix[1, 0], rotation_matrix[0, 0]))

# raise NotImplementedError('extract_yaw_from_matrix not implemented.')

return yaw
```

    c. get_corner

```
# TODO
translation = np.array([[box[0], box[1]]]) # Get the coordinate of the bbox
width = box[2]
height = box[3]
yaw = box[4]

rotation_matrix = np.array([[np.cos(yaw), -np.sin(yaw)], [np.sin(yaw), np.cos(yaw)]]) # Compute the rotation matrix
corners = np.array([[-width, -height], [width, -height], [width, height], [-width, height]]) # Get the corners
corner_global = (rotation_matrix @ corners.T).T + translation # Get the corner in the global coordinate system
corner_global = corner_global.astype(np.int64)

return corner_global
```

    d. vehicle_coor_to_image

```
# TODO
# Multiply position and extent by pixels_per_meter to convert the unit from meters to pixels
box[:4] = box[:4] * pixels_per_meter


# Compute pixel location that represents 0/0 in the image
translation = np.array([-(min_x * pixels_per_meter), -(min_y * pixels_per_meter)])

# Shift the coordinates so that the ego_vehicle is at the center of the image
box[0] = -box[0]
box[:2] = box[:2] + translation
box[4] = -box[4]
```

    e. iou_bbs

```
def rect_polygon(x, y, width, height, angle):
  p = Polygon([(-width, -height), (width, -height), (width, height), (-width, height)])
  return shapely.affinity.translate(shapely.affinity.rotate(p, angle, use_radians=True), x, y)

def iou_bbs(bb1, bb2):
  '''
    Just do the IoU calculation
  '''
  # TODO
  a = rect_polygon(bb1[0], bb1[1], bb1[2], bb1[3], bb1[4])
  b = rect_polygon(bb2[0], bb2[1], bb2[2], bb2[3], bb2[4])
  intersection_area = a.intersection(b).area
  union_area = a.union(b).area
  iou = intersection_area / union_area
  return iou
```

f. forward

```
# TODO
"""
    Use the formula to predict the future position of cars
"""
x_current_positions = bounding_boxes[:, :, 0]
y_current_positions = bounding_boxes[:, :, 1]
yaw = bounding_boxes[:,:,4]
velocities = bounding_boxes[:, :, 5]

future_boxes = bounding_boxes
future_boxes[:,:,0] = x_current_positions + velocities * self.dt * torch.cos(yaw)
future_boxes[:,:,1] = y_current_positions + velocities * self.dt * torch.sin(yaw)

return future_boxes
```

g. L2-distance

```
L2_dis = None
# TODO
"""
    Just calculate the L2 distance, and I notice that
    the size of gt box and future box may have different size, thus, I discard the exceeding part
"""
mLen = min(len(gt_bbs[:, :2]), len(future_bbs[:, :2]))

gt_bbs1 = gt_bbs[:mLen]
gt_xy = gt_bbs1[:, :2]

future_bbs1 = future_bbs[:mLen]
pred_xy = future_bbs1[:, :2]


L2_dis = np.linalg.norm(gt_xy - pred_xy, axis=1)
L2_dis = L2_dis[0] if len(L2_dis) != 0 else 0
if L2_dis is None:
    raise NotImplementedError('calculate L2_dis not implemented')
# END TODO
```

h. Visualize the boxes

```
for idx, box in enumerate(gt_bbs):
    if future_bbs is not None:
        future_center = future_bbs[idx]
        avg_iou += iou_bbs(box, future_center)

    # TODO
    # Draw predicted and ground truth bounding boxes on BEV image
    # P.S. use vehicle_coor_to_img(), draw_box()
    vehicle_coor_to_img(box, loc_pixels_per_meter, config.min_x, config.min_y)
    topdown = draw_box(topdown, box, (255, 255, 0))
    vehicle_coor_to_img(future_center, loc_pixels_per_meter, config.min_x, config.min_y)
    topdown = draw_box(topdown, future_center, (0, 0, 255))
```

2. Discussion

a. Compare the result of constant velocity and learning-based forecasting models at different forecasting steps.

i. Quantitative result (Metric: IOU and L2 distance)

- Constant velocity & Learning-based

Under the same short focasting time (0.5), the IoU and L2 distance of both approach are
Constant velocity: 0.902, 0.315
Learning-based: 0.479, 1.071

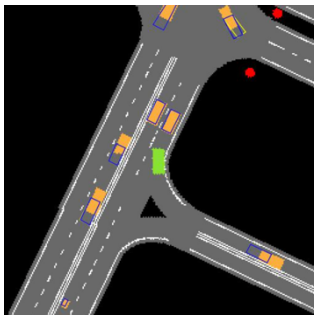
Under the same long focasting time (2), the IoU and L2 distance of both approach are
Constant velocity: 0.543, 2.687
Learning-based: 0.298, 3.487

ii. Qualitative result (Visualization)

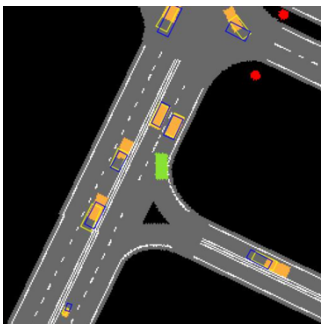
- Constant velocity & Learning-based

Under the same short focasting time (0.5), the IoU and L2 distance of both approach are

Constant:

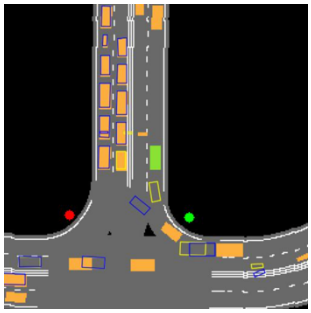


Learning-based:

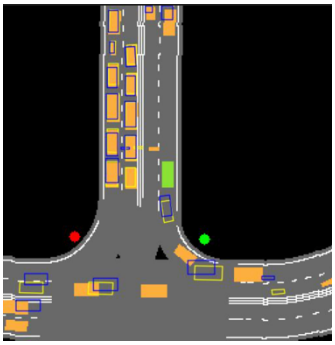


Under the same long focasting time (2), the IoU and L2 distance of both approach are

Constant:



Learning-based:



iii. Which one is better, why?

While the constant velocity model maintains consistent accuracy across different forecasting times, it may struggle with complex maneuvers like turns. In contrast, the learning-based model, though potentially less accurate overall, could excel in predicting such maneuvers due to its ability to learn from diverse datasets. Therefore, the decision ultimately depends on the importance of accurately capturing complex motion patterns versus the need for consistent and reliable predictions across various scenarios.

iv. Your conclusion

In conclusion, the constant velocity model offers consistent accuracy but may struggle with complex maneuvers like turns. In contrast, the learning-based model, while potentially less accurate overall, excels in predicting such maneuvers due to its ability to learn from diverse datasets. The choice depends on whether accurately capturing complex motion patterns or ensuring consistent and reliable predictions across various scenarios is more important. For example, the constant velocity model may suffice for typical driving scenarios, while the learning-based model could be crucial in urban environments with frequent turns. So we should combine these two models and weighted them if we put these to real-world application. Like when driving on highway, using constant, and when in the environemnt with frequent turns, we can use learning-based model.

3. Question Answering
    a. We quantize the label when training the learning-based forecasting model, converting the task into a classification problem. What would this design affect the performance of the model ?
        Classification models typically provide deterministic predictions, whereas regression models can naturally capture uncertainty through probabilistic outputs or confidence intervals. By converting the task into a classification problem, the model may struggle to express uncertainty in its predictions, which can be crucial for decision-making in real-world applications.

    b. What else can we do to improve the performance of the forecasting model?
        Enhance the more features used for prediction. This might include additional data sources such as vehicles' accerlation, road congestion, historical traffic patterns, etc. Extracting relevant features and transforming them appropriately can significantly enhance model performance.

    c. How would you use this representation to facilitate planning?
        I can use these representation to avoid vehicle collision when planning.

    d. Did you encounter any problems with this assignment?
        Yes, I spent some time knowing the each element of bounding box. And studied some coordination system transformation.

4. Please provide any reference you take
    a. https://carla.readthedocs.io/en/latest/python_api/
    b. https://motion.cs.illinois.edu/RoboticSystems/CoordinateTransformations.html#Homogeneous-coordinate-representations