Argoverse by Argo.ai

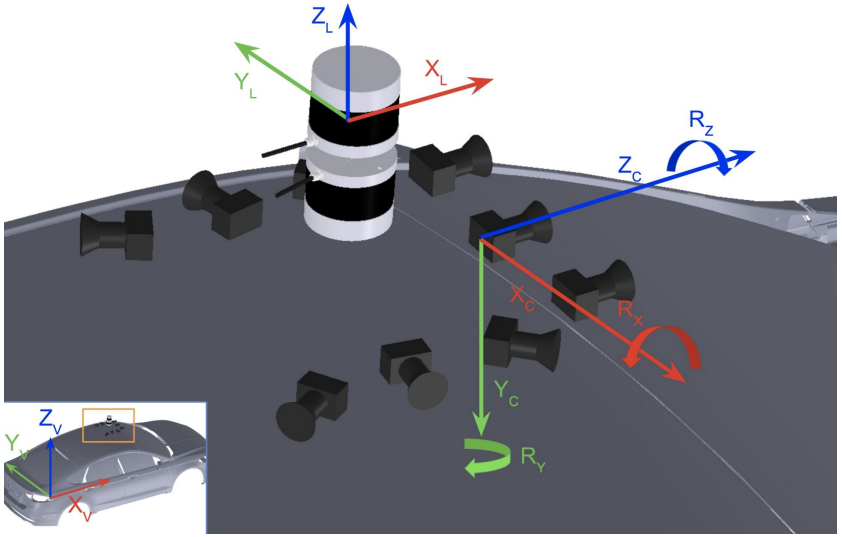
Mehul Varma

What is Argoverse?

- A set of three datasets designed to support autonomous vehicle perception tasks including 3D tracking and motion forecasting.
  - One dataset with 3D tracking annotations for 113 scenes
  - One dataset with 324,557 unusual vehicle trajectories extracted from over 1000 driving hours (motion forecasting)
  - Two high-definition (HD) maps with lane centerlines, traffic direction, ground height, and more
- Data collected from Miami (204 linear km) and Pittsburgh (86 linear km)
- One API to connect the map data with sensor information

3 Parts to Argoverse's Dataset

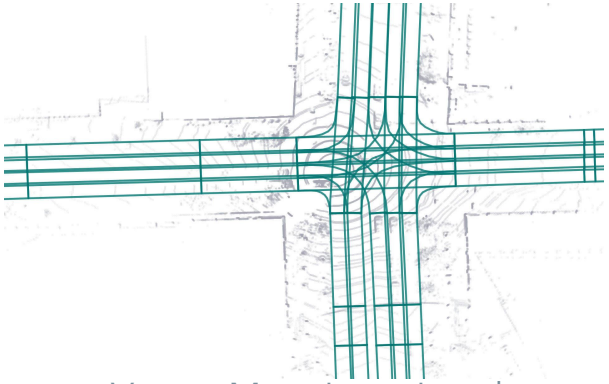
- High-definition maps
- 3D tracking
- Motion forecasting



Schematic of car's sensors (2 roof-mounted LiDAR sensors, 7 ring cameras, 2 forward stereo cameras

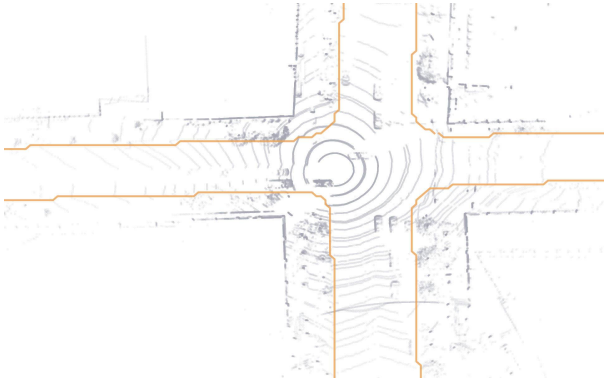
#1 Maps

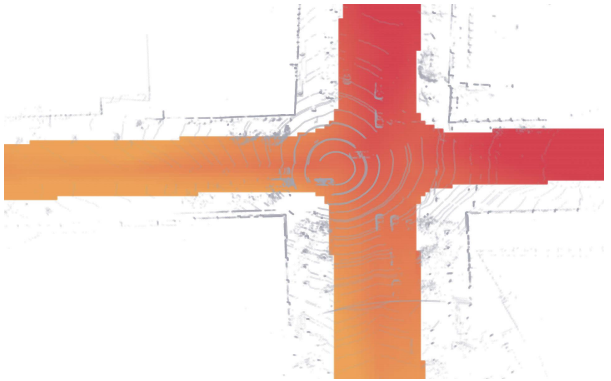
- “180 miles of mapped lanes contain rich geometric and semantic metadata not currently available in any public dataset”



Vector Map: Lane-Level Geometry

Rasterized Map: Drivable Area

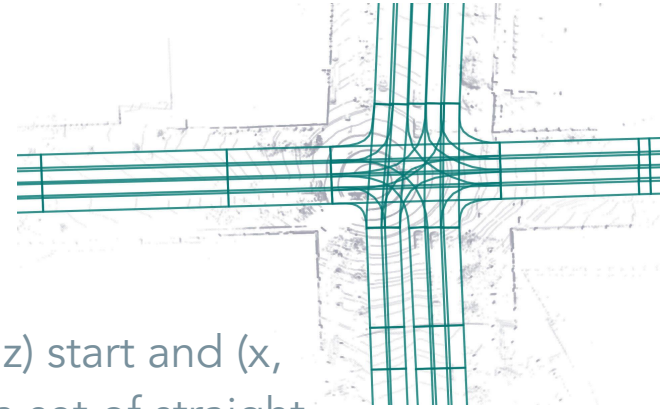




Rasterized Map: Ground Height

Vector Map: Lane-Level Geometry

- Provide a number of semantic attributes
  - lane centerlines (split into lane segments)
  - traffic direction (left, right, or none)
  - whether a lane is located within an intersection
  - has an associated traffic control measure (boolean values)
  - unique identifiers for the lane's predecessors
- Each straight segment is defined by 2 vertices: (x, y, z) start and (x, y, z) end. Thus, curved lanes are approximated with a set of straight lines
- Observations include that vehicle trajectories generally follow the center of the lane
- Can classify roads based on their suitability of self-driving



Code & Data for Vector Maps

```python
class LaneSegment:
    def __init__(
        self,
        id: int,
        has_traffic_control: bool,
        turn_direction: str,
        is_intersection: bool,
        l_neighbor_id: Optional[int],
        r_neighbor_id: Optional[int],
        predecessors: Sequence[int],
        successors: Optional[Sequence[int]],
        centerline: np.ndarray,
    ) -> None:
        """Initialize the lane segment.

        Args:
            id: Unique lane ID that serves as identifier for this "Way"
            has_traffic_control:
            turn_direction: 'RIGHT', 'LEFT', or 'NONE'
            is_intersection: Whether or not this lane segment is an intersection
            l_neighbor_id: Unique ID for left neighbor
            r_neighbor_id: Unique ID for right neighbor
```

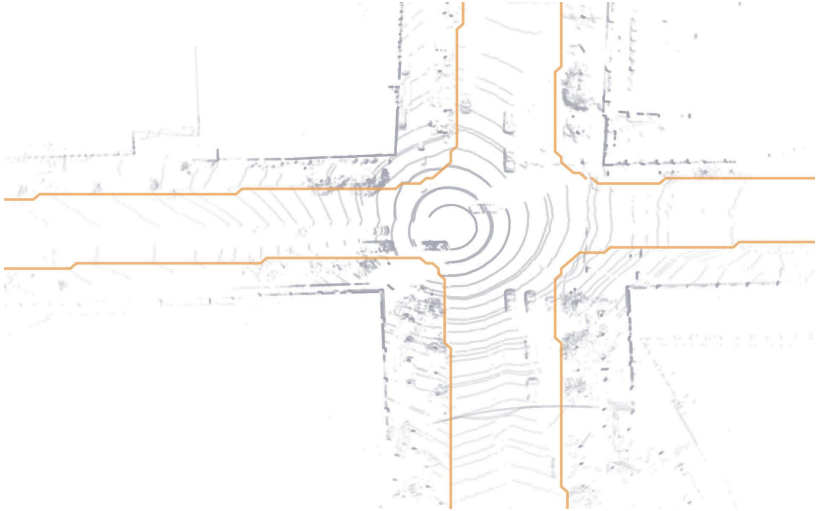
```python
class Node:
    """
    e.g. a point of interest, or a constituent point of a
    line feature such as a road
    """

    def __init__(self, id: int, x: float, y: float, height: Optional[float] = None):
        """
            Args:
                id: representing unique node ID
                x: x-coordinate in city reference system
                y: y-coordinate in city reference system

            Returns:
                None
        """
        self.id = id
        self.x = x
        self.y = y
        self.height = height
```

```xml
<?xml version='1.0' encoding='UTF-8'?>
<ArgoverseVectorMap>
  <node id="0" x="252.05857658141758" y="1783.094199654879"/>
  <node id="1" x="252.2491120869268" y="1781.4253478064977"/>
  <node id="2" x="252.43964759243602" y="1779.7564959581164"/>
  <node id="3" x="252.6301830979452" y="1778.0876441097353"/>
  <node id="4" x="252.82071860345442" y="1776.4187922613542"/>
  <node id="5" x="253.01125410896364" y="1774.7499404129726"/>
  <node id="6" x="253.20178961447286" y="1773.0810885645915"/>
  <node id="7" x="253.39232511998205" y="1771.4122367162104"/>
  <node id="8" x="253.58286062549126" y="1769.743384867829"/>
  <node id="9" x="253.77339613100048" y="1768.0745330194477"/>
  <node id="10" x="257.386197283864" y="1768.1909449379891"/>
```

Coordinates of Each of the Road Segments ~ Nodes

Rasterized Map: Drivable Area

- Converting raw data from LiDAR sensors to visual ‘rasterized data’
- Include binary drivable area labels at one-meter grid resolution (+/- 0.5 m)
- Drivable Area: an area in which it is possible for a vehicle to drive (but not necessarily legally)
    - Ex. A road’s shoulder
- Track annotations extend to five meters beyond the drivable area, called the “region of interest”

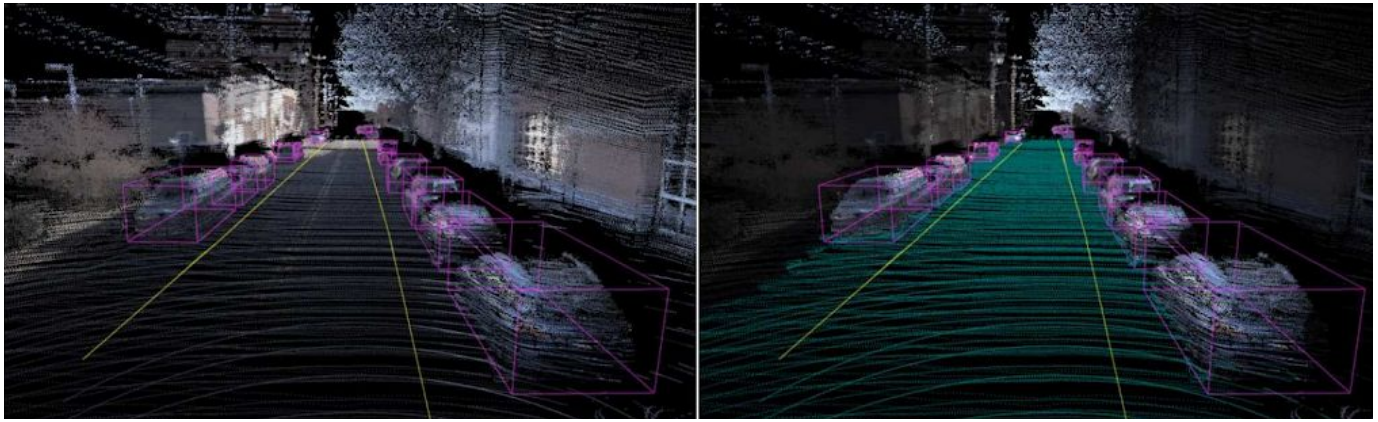


Code for Rasterized Map: Drivable Area

```
def build_city_driveable_area_roi_index(self) -> Mapping[str, Mapping[str, np.ndarray]]:
    """
    Load driveable area files from disk. Dilate driveable area to get ROI (takes about 1/2 second).

    Returns:
        city_rasterized_da_dict: a dictionary of dictionaries. Key is city_name, and
                value is a dictionary with driveable area info. For example, includes da_matrix: Numpy array of
                shape (M,N) representing binary values for driveable area
                city_to_pkl_image_se2: SE(2) that produces takes point in pkl image to city coordinates, e.g.
                p_city = city_Transformation_pklimage * p_pklimage
    """
    city_rasterized_da_roi_dict: Dict[str, Dict[str, np.ndarray]] = {}
    for city_name, city_id in self.city_name_to_city_id_dict.items():
        city_id = self.city_name_to_city_id_dict[city_name]
        city_rasterized_da_roi_dict[city_name] = {}
        npy_fpath = MAP_FILES_ROOT / f"{city_name}_{city_id}_driveable_area_mat_2019_05_28.npy"
        city_rasterized_da_roi_dict[city_name]["da_mat"] = np.load(npy_fpath)

        se2_npy_fpath = MAP_FILES_ROOT / f"{city_name}_{city_id}_npyimage_to_city_se2_2019_05_28.npy"
        city_rasterized_da_roi_dict[city_name]["npyimage_to_city_se2"] = np.load(se2_npy_fpath)
        da_mat = copy.deepcopy(city_rasterized_da_roi_dict[city_name]["da_mat"])
        city_rasterized_da_roi_dict[city_name]["roi_mat"] = dilate_by_l2(da_mat, dilation_thresh=ROI_ISOCONTOUR)
```

Accumulating LiDAR points and projecting them to a virtual image plane

LiDAR points beyond driveable area are dimmed. Points near the ground are in cyan. Cuboid object annotations & road centerlines are shown in pink and yellow.

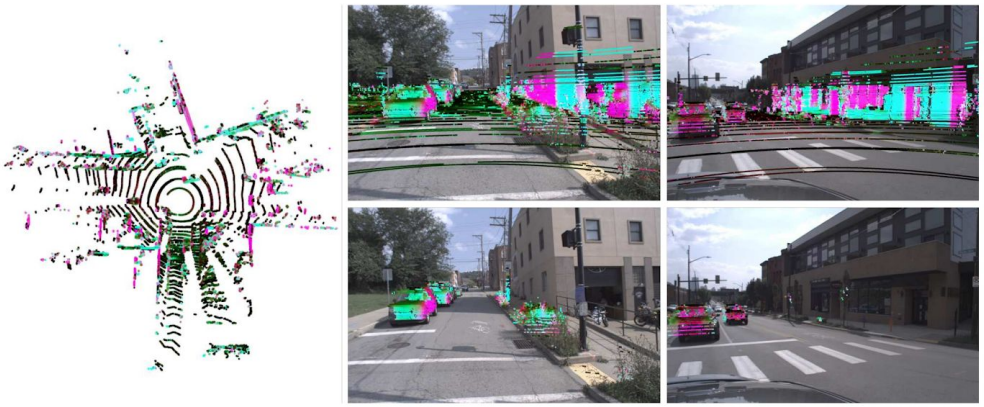


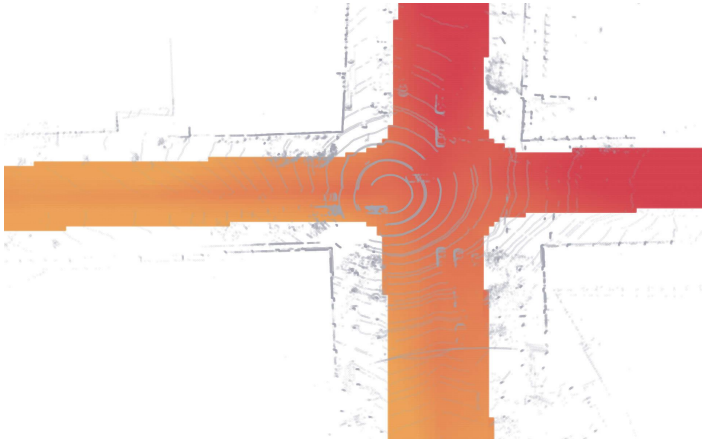
```
934e 554d 5059 0100 7600 7b27 6465 7363
7227 3a20 277c 7531 272c 2027 666f 7274
7261 6e5f 6f72 6465 7227 3a20 4661 6c73
652c 2027 7368 6170 6527 3a20 2833 3034
332c 2034 3235 3929 2c20 7d20 2020 2020
2020 2020 2020 2020 2020 2020 2020 2020
2020 2020 2020 2020 2020 2020 2020 2020
2020 2020 2020 2020 2020 2020 2020 200a
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

Drivable Data Visualized from Data in the Matrix

Rasterized Map: Ground Height

- Uses LiDAR to determine the real-valued ground height at one-meter resolution (+/- 0.5 m)
- Able to identify cars, obstructions, slant/rise of roads, etc.
- Scenes containing uneven ground are removed through processing techniques (the explicit assumption is that the ground is flat/planar)

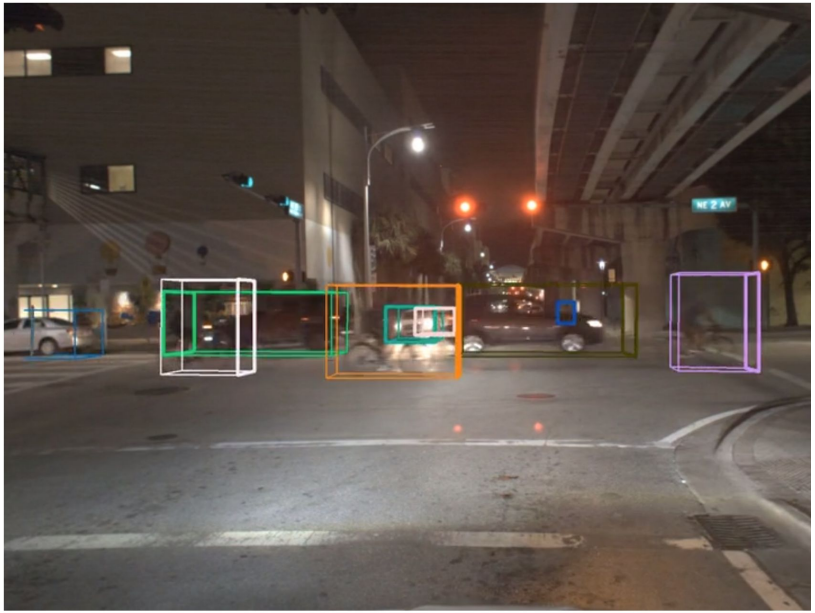




An intersection with a slight slant
(depicted by color gradient)

#2 3D Tracking

- These short video log segments are 15-30 seconds, which help us apply CNNs that aid in object segmentation.
- Help us with the understanding of the movement of the objects on the road.
- 30 fps, 360 degree view
- 10,000 tracked objects (like cars, pedestrians, signs, etc)
- https://www.argoverse.org/data.html#tracking-link

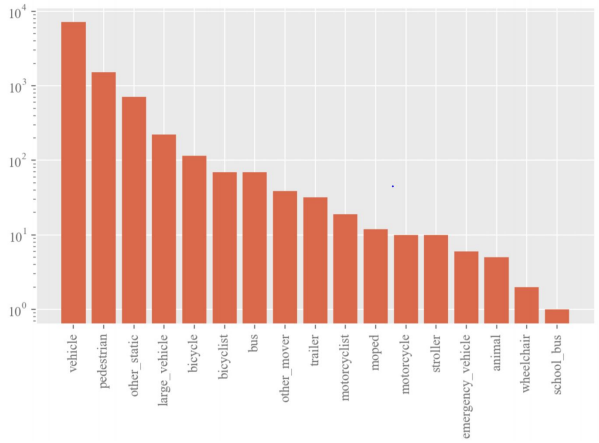


How the Data is Trained

Given a sequence of F frames, each frame contains set of 3D points from LiDAR {Pi | i = 1, ..., N}, where Pi ∈ R3 of x, y, z coordinates, we want to determine a set of track hypothesis {Tj | j = 1, ..., n} where n is the number of unique objects in the whole sequence, and Tj contains the set of object center locations at frames f for f = {fstart, ..., fend}, the range of frames where the object is visible

- From f frames, we get a set of points that have 3 dimensions to them, and we want to obtain a hypothesis where we classify each of the objects into different classes and train our model to find the center points and learn to make the bounding boxes
- It uses Mask R-CNNs to classify each of the objects. This classification is based on Faster/ Fast CNNs which learns bounding boxes for objects.

- The model doesn’t use IoU which is more commonly used for object classification, it rather uses, the Euclidean distance between the objects to identify multiple objects in one image. It is more effective in this case.
- It also uses drivable area, ground removal, lane direction to assist the model in 3d tracking as we see




```
""" Object Class mapping dictionary."""
OBJ_CLASS_MAPPING_DICT = {
    "VEHICLE": 0,
    "PEDESTRIAN": 1,
    "ON_ROAD_OBSTACLE": 2,
    "LARGE_VEHICLE": 3,
    "BICYCLE": 4,
    "BICYCLIST": 5,
    "BUS": 6,
    "OTHER_MOVER": 7,
    "TRAILER": 8,
    "MOTORCYCLIST": 9,
    "MOPED": 10,
    "MOTORCYCLE": 11,
    "STROLLER": 12,
    "EMERGENCY_VEHICLE": 13,
    "ANIMAL": 14,
    "WHEELCHAIR": 15,
    "SCHOOL_BUS": 16,
}
```

Results of 3D Tracking

Baseline tracker is compared with 3 modifications

- ★ Mask-RCNN dramatically improves our detection performance by reducing false positives.
- ★ Map-based ground removal leads to slightly better detection performance (higher MOTA) than a plane-fitting approach at longer ranges
- ★ lane direction from the map doesn't affect our metrics (based on centroid distance), but it helps initialize vehicle direction

(a) without lane information (b) with lane information

Figure 7: **Tracking with orientation snapping.** Using lane direction information helps to determine the vehicle orientation for detection and tracking. Ground truth cuboids are green.

RANGE THRESHOLD	USE MASK-RCNN	USE MAP LANE	GROUND REMOVAL	MOTA	MOTP	IDF1	MT(%)	ML(%)	# FP	#FN	IDSW	#FRAG
100 m	Y	Y	map	**37.98**	**0.52**	**0.46**	0.10	0.51	105.40	2455.30	32.55	22.35
	N	Y	map	16.42	0.54	**0.46**	**0.16**	**0.41**	1339.95	**1972.95**	43.30	29.65
	Y	N	map	37.95	**0.52**	**0.46**	0.10	0.51	105.30	2454.85	32.35	22.45
	Y	Y	plane-fitting	37.36	0.53	**0.46**	0.10	0.53	**105.20**	2484.00	**31.10**	**21.25**
50 m	Y	Y	map	**52.74**	**0.52**	**0.58**	0.22	0.29	99.70	1308.25	31.60	21.65
	N	Y	map	21.53	0.54	0.55	**0.38**	**0.18**	1197.30	**897.90**	37.85	24.60
	Y	N	map	52.70	**0.52**	**0.58**	0.22	0.29	99.50	1307.75	31.40	21.75
	Y	Y	plane-fitting	52.05	0.53	**0.58**	0.20	0.31	**98.10**	1335.65	**30.15**	**20.45**
30 m	Y	Y	map	**73.02**	**0.53**	0.73	0.66	0.08	92.80	350.50	19.75	12.80
	N	Y	map	23.28	0.56	0.63	**0.78**	**0.04**	837.45	**238.80**	**19.10**	**11.25**
	Y	N	map	72.99	**0.53**	0.73	0.66	0.09	92.80	349.90	19.65	12.95
	Y	Y	plane-fitting	72.82	**0.53**	**0.74**	0.66	0.09	**92.00**	363.35	19.75	12.85

```
    metrics=[
        "num_frames",
        "mota",
        "motp",
        "idf1",
        "mostly_tracked",
        "mostly_lost",
        "num_false_positives",
        "num_misses",
        "num_switches",
        "num_fragmentations",
    ],
    name="acc",
)
```

#3 Motion Forecasting

The forecasting task is framed as: *given the past input coordinates of a vehicle trajectory* V_i *as* $X_i = (x_i^t, y_i^t)$ *for time steps* $t = \{1, \ldots, T_{obs}\}$, *predict the future coordinates* $Y_i = (x_i^t, y_i^t)$ *for time steps* $\{t = T_{obs+1}, \ldots, T_{pred}\}$.

- First step is to localize the object on the vector map
- The next steps in forecasting are
  - Hypothesis phase
  - Generation phase
- We use BFS to generate trajectories in the map which makes generation phase easy to run
- The problem is that trajectories are more complicated because of the multimodal nature of the problem, e.g. it's difficult to know which lane segment a vehicle will follow in an intersection.
- Demonstration: https://www.argoverse.org/index.html

```
def bfs_enumerate_paths(graph: Mapping[str, Sequence[str]], start: str, max_depth: int = 4) -> Sequence[Sequence[str]]:
    """Run Breadth-First-Search. Cycles are allowed and are accounted for.

    Find (u,v) edges in E of graph (V,E)

    Args:
       graph: Python dictionary representing an adjacency list
       start: key representing hash of start/source node in the graph search
       max_depth: maximum depth to traverse in graph from start node

    Returns:
      all_paths: list of graph paths
    """
    dists: MutableMapping[str, float] = {}

    # mark all vertices as not visited
    for k, neighbors in graph.items():
        dists[k] = float("inf")
        for v in neighbors:
            dists[v] = float("inf")

    dists[start] = 0
    paths: MutableSequence[MutableSequence[str]] = []
    # maintain a queue of paths
    queue: MutableSequence[MutableSequence[str]] = []
    # push the first path into the queue
    queue.append([start])
    while queue:  # len(q) > 0:
        # get the first path from the queue
        path: MutableSequence[str] = queue.pop(0)
        # get the last node from the path
        u: str = path[-1]
        # max depth already exceeded, terminate
        if dists[u] >= max_depth:
            break
        # enumerate all adjacent nodes, construct a new path and push it into the queue
        for v in graph.get(u, []):
            if dists[v] == float("inf"):
                new_path: MutableSequence[str] = list(path)
                new_path.append(v)
                queue.append(new_path)
                dists[v] = dists[u] + 1
                paths.append(new_path)

    return remove_duplicate_paths(paths)


def remove_duplicate_paths(paths: Sequence[Sequence[Any]]) -> Sequence[Sequence[str]]:
```

Motion Forecasting: Multimodal Evaluation

- Forecasting Task: Observe 20 past frames (2 seconds) and then predicting 10-30 frames (1-3 seconds) into the future
- Incorporate both social and spatial context to predict outcome
- Output: a semantic graph
  - Important to predict *many* possible outcomes and not just the *most likely* outcome
- On average, the heuristics generate 5.9 separate hypotheses for possible vehicle trajectories
  - Compact yet diverse set of forecasts

Motion Forecasting: Trajectory Prediction

- Evaluated the effectiveness of numerous models in correctly predicting the outcome.
  - Ex. Evaluate the effect of adding social context
- Different combinations of Constant Velocity, Nearest Neighbor (NN), Map (with various centerlines), LSTM (Long Short-Term Memory) Encoder-Decoder Model, Social Context
  - LSTM Encoder-Decoder: A specific RNN designed to address sequence-to-sequence problems (forecasting the next value in a real-valued sequence)
  - Social Context: Interacting with pedestrians, leaving space between the next car, etc. (social norms but with vehicles)

Constant Velocity
NN
NN+map(oracle)
NN+map
LSTM ED
LSTM ED+social
LSTM ED+map(oracle)
LSTM ED+map
LSTM ED+social+map(oracle)

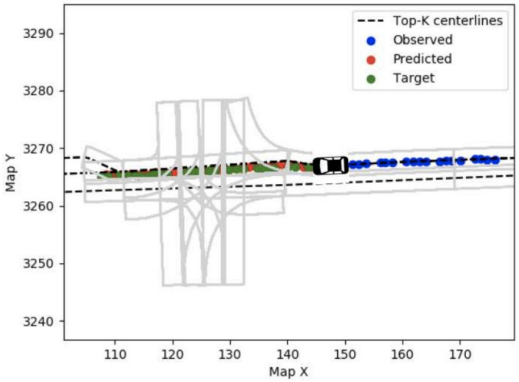
Trajectory Prediction - Results

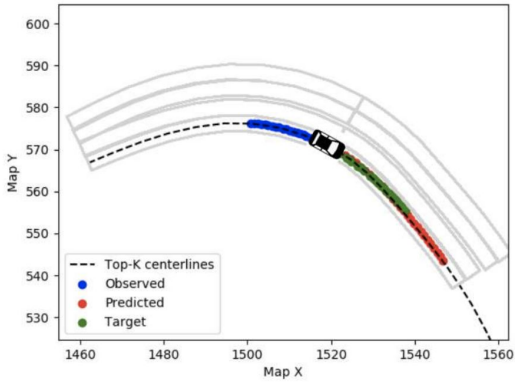
- Constant Velocity was outperformed by all other behaviors
  - Failed to capture typical driving behavior (acceleration, deceleration, turns, etc.)
- LSTM ED + social performs similar to LSTM ED
  - Social context does not add significant value to forecasting
- NN+map has a lower error than LSTM ED+social and LSTM ED
  - Even a shallow model working on top of a vector map works better than a deep model with social features and no vector map.

	1 SECOND		3 SECONDS	
BASELINE	ADE	FDE	ADE	FDE
Constant Velocity	1.04	1.89	3.55	7.89
NN	0.75	1.28	2.46	5.60
NN+map(oracle)	0.82	1.39	2.39	5.05
NN+map	0.72	1.33	2.28	4.80
LSTM ED	**0.68**	1.78	2.27	5.19
LSTM ED+social	0.69	**1.20**	2.29	5.22
LSTM ED+map(oracle)	0.82	1.38	2.32	4.82
LSTM ED+map	0.80	1.35	**2.25**	**4.67**
LSTM ED+social+map(oracle)	0.89	1.48	2.46	5.09

Example: LSTM Encoder-Decoder + Map

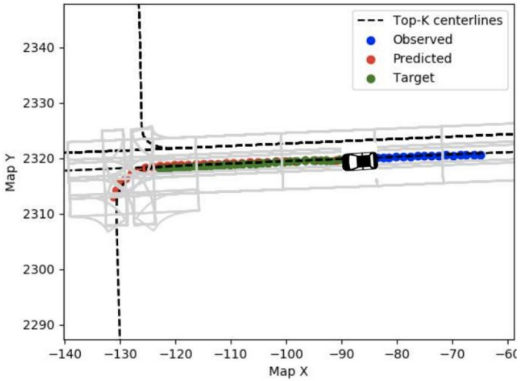
The model correctly predicts that the car will go straight at the intersection.

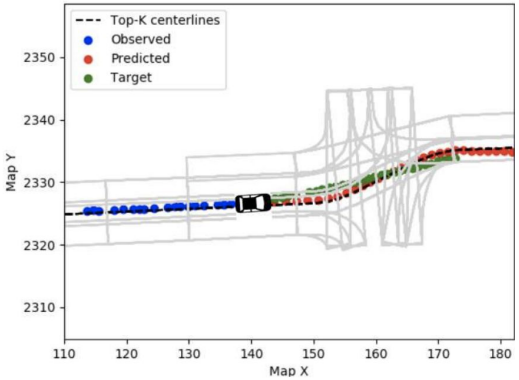




The model correctly predicts a smooth right turn never going out of the lane, which might have been difficult if there were no map.

Demonstration of the multimodal nature of predictions, where the model considers all top-K possibilities.





The predictions are on a non-typical lane which takes a slight left and then a slight right. Again, this is hard to predict without a map.

Next Steps

- Training the Argoverse data
- Adjusting our Unreal model to Argoverse and building direct simulations taking parts of their projections