**Deep Learning**

* Time series prediction
* Pipeline components using DL

**Reinforcement Learning**

* Inverse RL
* Imitation Learning

**Conventional Methods**

* Kalman Filter
* Probabilistic Models
* Markov Models

# IRL

Not relevant, but interesting concepts. Relies on an ‘expert’.  
**Inverse RL**: estimates a reward function motivating the expert demonstrator, and then is able to construct trajectories based on this loss function.  
**Imitation learning:** attempt to imitate an expert. We may want to do this in some manner, via learning through datasets that were gathered by human drivers. But directly learning through it – probably not what we want.

# Conventional Methods

## Methods Overview

### Physics Based

* Simplest models
* Assume the vehicle’s motion depends only on physical equations of motion
* Examples: constant velocity, constant acceleration
* Reliable for a short horizon

### Maneuver-Based Motion Models

* More sophisticated prediction methods
* Assumes motion is independent of other vehicles and factors
* Example: Kalman filter

### Interaction-Aware Motion Models

* Consider the reactive part of multiple vehicles

## Interacting Multiple Model Kalman Filter

### Overview

* Use IMM filter (interacting multiple model - extension of Kalman filter) that utilize different models (KFs) to model object trajectories and interactions
* Different models represent different intentions. At each time step, for each vehicle, a model is selected to estimate the next state. Model selected according to conditional probability  
  A close up of a word

  Description automatically generated
  + Longitudal Motion: velocity tracking (reference velocity), distance keeping (cruise-control)
  + Lateral Motion (lane change).  
    Note: for our purpose, we *know* the ego intention is overtake (2 lane changes). Maybe bake this into the system before hand, instead of predicting it, to evaluate validity (overtake feasibility) and other vehicle’s interaction based on our intent.
* Use most likely model for vehicle at each timestep to produce a trajectory comprised of points (mean) and uncertainty (covariance)  
  A screenshot of a map

  Description automatically generated
* Project trajectory to ensure collision-free trajectory (optimization problem)
* Note that IMM filter is not interaction-aware – it’s a way to model different manoeuvres for a single object. The interaction-aware aspect of the proposed scheme derives from the projection (the min distance projection is suboptimal, but perhaps improving is challenging).
* Challenge: update number of objects of interest

### IMM Filter

A diagram of a model

Description automatically generated

* Describes how to calculate (recursively) model selection probability – probability that model is selected at time step given the observations from time 1 to
* Model transition (prior probability) is modelled by a Markov chain given

# DL Methods

## Overview Idea

Given historical trajectories, form an input aggregating trajectories, local map information, ego control to predict future trajectory.  
This is largely seen as a time series problem, and the use of RNNs and LSTMs is prevalent.

A diagram of a network

Description automatically generated

## Challenges

### Use existing trained model (Inference)

Need to coordinate and match inputs i.e. map, label class, etc.  
Will probably perform poorly due to training and inference mismatch in inputs (not robust!)

### Train

* Most likely necessary if we go down this route.
* Most likely, training on existing open-source datasets should translate well into real settings.
* It might make more sense to construct our own dataset so inputs align better, including noise, object detection states, etc. See dataset construction below.
  + Will it be too noisy to train a model on? i.e. are our measurements good enough?

### Dataset Construction

* From trips, extract frames containing information:
  + Minimap vector
  + Objects
    - Type
    - Position
    - Heading
  + Optional: last ego control input
* Important to keep in mind: will chosen method rely on expert demonstration i.e. human driver?
  + If so, then using trips should be very counterproductive – why would want to mimic a faulty system?
  + Can we ignore ego, and focus on other agents for trajectory prediction? Possibly, we might not need to predict our own trajectory.

## Implementation

### CASPNet++: Joint Multi-Agent Motion Prediction

A diagram of a computer

Description automatically generated

* Discretize into grid
* Encode agent information (type, position, velocity, acceleration, heading, size)
* Encode maps (CNN)
* Output: Grid occupancy matrix. For each cell, produce probability that an agent of class C is there, and at which velocity it’s at

### TNT

A diagram of a process

Description automatically generated

* Generate targets (orange star) by generating
  + Given N candidates : can be chosen differently for object type (vehicle / pedestrian)
  + Produce scorer function (MLP) that can be compared to ground truth using cross entropy loss
  + Allow for offset of candidates using another MLP in a manner I didn’t quite understand, but isn’t necessary to understand the big picture at this stage.
* Given context (map) feature and target location (predicted target orange star), output predicted trajectory, and compare to ground truth (one trajectory per target)
* For M trajectories (corresponding to M targets), score them using softmax over output of some MLP that scores input path given context   
  + Construct CE loss that compares with “ground truth scores” (distance from ground truth trajectory)
* Inference:
  + Encode context
  + Sample N target candidates, take top M scored by MLP 1
  + Generate trajectory for each target using MLP 2
  + Score them using MLP3
  + Take top K

## Trajectron++

### Refs:

<https://www.youtube.com/watch?v=EVxS7tC9LMI>

<https://arxiv.org/pdf/2001.03093>

### Theory

**Big picture:** create distribution of trajectory (mean and variance of points).

**Their improvement upon previous work:** the consideration of dynamics (through MATS), their use of backpropagating error, including uncertainty in class prediction and position uncertainty.

#### Trajectory Prediction

1. **Construct Directed Graph**

Nodes (agents) are connected only if they are close to each other (closeness determined by class )  
Neighbors impact each other’s trajectory prediction (through input to encoder).

1. **Encoder**

Encode features and pass to CVAE (conditional variational autoencoder).

* 1. CVAE computes control mean and distribution

A diagram of a diagram

Description automatically generated

1. **Decoder**

The ‘decoder’ that predicts agent position is “dynamic-aware”: simply a prediction step of a Kalman filter given the controls

A diagram of mathematical equations

Description automatically generated

1. **Prediction Control**  
   Replace tracklets with MATS  
   Tracklets: previous de facto way of trajectory parameterization – set of waypoints (position, velocity, heading)  
   MATS produces a dynamical system as the trajectory forecasting output  
   A diagram of a cube with a tracklets and lines

   Description automatically generated

A diagram of a rectangular object with a black text

Description automatically generated with medium confidence  
  
A diagram of a variety of text

Description automatically generated with medium confidence

Note: control is given, not found (slightly counterintuitive to our purpose)

1. **Perception prediction**Incorporate class and prediction uncertainty from perception module into prediction uncertainty.
2. **Output**Note that latent variable is a vector.
   1. ML  
      take
   2. Generate predictions using
   3. Full
   4. Distribution

### Implementing

#### Train our own version

Time consuming, complex

#### Use trained version, but match inputs

* Check if inputs can be matched / configured.
* Check training dataset, and if we expect it to perform in our settings as well.

# API

## Trajectory Prediction

### Input

frames consisting of:

* Map
  + Road map
  + Maybe additional sensor maps, if we want to go down the DL route
* Objects
  + Type
  + Location (bounding box)
  + Heading

### Output

frames of trajectory horizon for objects present at time . This may be (depending on implementation) in the form of:

1. Discrete set of waypoints
2. Top K trajectories with probabilities
3. Probability function, of which a trajectory may be sampled

For 2+3, we may want to use the probabilities as input for the next module which selects a trajectory. Since we may want to include the probability of collision-free in the loss function of the scoring function.