Lecture 13/14, Feb 2, 2026

Planning for Autonomous Driving: Autonomoose Case Study

- Planning for complex problems usually involves a hierarchical structure, from mission planning, to behavioural planning, to local planning and finally vehicle control; we optimize at each level of abstraction
- The mission planner is the highest level and navigates at the road map level, using a graph search
- The behavioural planner decides what actions to take and when it's safe to proceed, e.g. looking at traffic signs and other traffic participants
  - For simple applications this can be handled well by a finite state machine
  - More complex applications use a rule-based system, which use a hierarchy of rules where higher priority rules have precedence
  - Modern systems may use reinforcement learning techniques trained in simulation
- The local planner generates feasible paths and comfortable velocity profiles, which is often further decomposed into path planning and velocity profile generation
  - Lattice planners often work well since actions (the *control set*) are often highly restricted in autonomous driving (e.g. driving in the current lane or switching to adjacent lanes)
    * This works less well in unstructured environments or when obstacles are close together
    * *Conformal* lattice planners fit the control actions to the road structure, i.e. the grid curves with the road
    * Doing this for only one step gives trajectory rollout
  - Sampling-based planners can be useful in parking or unstructured scenes but is generally less useful on the road since paths are not smooth
  - Variational planners optimize the trajectory according to a cost functional, modifying the nominal path to account for obstacles
    * The cost functional would contain terms for collision avoidance and robot dynamics
    * This is like MPC, but using the path from the higher level planner to initialize
  - The lookahead point for planning is determined dynamically based on vehicle velocity and other factors – the faster we're going, the further we need to look ahead for obstacles and the less we can deviate from the original path
- The Autonomoose uses a conformal lattice planner:
  - To create the lattice we first need to connect points using a suitable parameterization with boundary conditions $(x_0, y_0, \theta_0, \kappa_0)$ and $(x_f, y_f, \theta_f, \kappa_f)$, with maximum curvature and minimum speed constraints
    * We've already seen the cubic spline: $\boldsymbol{r}(u) = (x(u), y(u)), u \in [0,1], x(u) = \alpha_3 u^3 + \alpha_2 u^2 + \alpha_1 u + \alpha_0, y(u) = \beta_3 u^3 + \beta_2 u^2 + \beta_1 u + \beta_0$
      - Quintic splines can also be used, but constraining curvature is more difficult and can have discontinuities
    * Instead we can use polynomial spirals, where we start with the parametrized curvature $\kappa(s) = a_3 s^3 + a_2 s^2 + a_1 s + a_0$, and integrate it to recover θ, x, y
      - This allows us to directly specify the start and end curvatures and ensures curvature continuity
      - However this has no closed-form solution, so we integrate numerically (e.g. Simpson's rule)
      - Curvature constraints can be imposed by checking only the 1/3 and 2/3 points on $\kappa(s)$ due to the cubic
  - The objective function uses the bending energy $\int_0^{s_f} \|\kappa(s)\|^2 \, \mathrm{d}s$ (encouraging paths that turn as little as possible)
    * Using a bending energy cost function encourages distributing the curvature along the path, avoiding sharper turns
    * The gradient for the bending energy also has a closed-form solution
  - It's often difficult for solvers to satisfy exact hard constraints, so we can convert some of them to soft constraints (penalize deviations heavily) to solve the optimization faster
    * For cubic spirals we can remap the parameters into the positions of 4 control points and an arc length parameter; the first and last points are already determined by the boundary conditions,

so we only need to optimize over 3 variables
  - To generate the conformal lattice (i.e. possible trajectories set), we sample points laterally on the road at the lookahead point, and compute cubic spirals to each goal point before checking for obstacles and discarding infeasible paths
    * We integrate numerically using a trapezoidal rule to recover a discrete path (faster than Simpson's rule if we're generating the entire path)
    * Collision checking is done using bounding volume hierarchies, approximating the car as 3 circles
    * Objective function for path selection is a design choice; we can do it based on deviation from the centre path or the path from the previous iteration
  - Doing this repeatedly for each planning step results in a form of MPC
- The next step is to define a velocity profile and taking into account dynamic obstacle in the environment
  - The behavioural planner generates a reference velocity, tracked by minimizing $\int_0^{s_f} \|v(s) - v_{ref}(s)\| \, \mathrm{d}s$
    * Hinge loss can be more effective since it penalizes going over the reference speed more than going under
  - For smoothness, we penalize jerk $\int_0^{s_f} \|\ddot{x}(s)\|^2 \, \mathrm{d}s$
  - We also want to match the velocity of the lead vehicle while going to a fixed distance behind it
  - The acceleration is constrained to stay within the friction ellipse, which contains the maximum magnitude of accelerations before tires lose traction, or a smaller range for comfort
    * Note lateral acceleration is $\frac{v^2}{r} = v^2\kappa$
  - Using these velocity limits, we can derive a desired velocity v_f based on the minimum of all of them and connect it to our current velocity v_0 to form the velocity profile
    * Linear ramp: useful for acceleration
    * Trapezoidal profile: linear deceleration to a slower speed, constant velocity coast to get close to target, and then decelerate to stop again