Lecture 22, Mar 4, 2026

LiDAR Scan Registration

- Using scan alignment with LiDAR, we can get a much better version of dead reckoning compared to wheel odometry
- Consider a model (reference) point set $M = \{ \boldsymbol{p}_M^1, \ldots, \boldsymbol{p}_M^{N_M} \}$ and data (target) point set $D = \{ \boldsymbol{p}_D^1, \ldots, \boldsymbol{p}_D^{N_D} \}$, and assume that each point in D corresponds to a single point in M, i.e. $N_M = N_D$, but unlike in feature-based odometry we no longer have correspondences
  - We wish to find $\boldsymbol{T}_{MD} = \{ \boldsymbol{C}_{MD}, \boldsymbol{t}_D^{MD} \}$, the transformation between two scans
  - The goal is to minimize $\Delta = \boldsymbol{p}_M^i - \boldsymbol{C}_{MD}(\boldsymbol{p}_D^j - \boldsymbol{t}_D^{MD})$
  - Form a loss $\mathcal{J}$, then we have $T^* = \underset{T \in SE(3)}{\arg\min}\, \mathcal{J}(M, T(D))$
- Many variations are possible in the problem formulation, including parameterization of the transformation, point dimensionality (e.g. intensity, colour, learned features), and point-level filtering, cost metrics, association (e.g. using semantic information)
- In the basic *iterative closest point* (ICP) algorithm:
  1. Transform data points $\boldsymbol{p}_M^j = \boldsymbol{C}_k(\boldsymbol{p}_D^j - \boldsymbol{t}_k)$ to form $\tilde{D}$
  2. Find nearest neighbour correspondences $\{ I, J \} = \mathrm{NN}(M, \tilde{D})$ between all points in M and transformed points in D
  3. Form quadratic error: $\mathcal{J} = \frac{1}{2} \sum_{j=1}^{J} w^j \left(\boldsymbol{p}_M^i - \boldsymbol{C}_k \left(\boldsymbol{p}_D^j - \boldsymbol{t}_k \right) \right)^T \left(\boldsymbol{p}_M^i - \boldsymbol{C}_k \left(\boldsymbol{p}_D^j - \boldsymbol{t}_k \right) \right)$
     - Each w^j is a scalar weight
     - Similar to visual odometry, we can also use matrix weightings here; however usually for LiDAR we have similar uncertainties in all dimensions, unlike in stereo vision, so we usually use scalar weights
  4. Gradient descent: $\boldsymbol{T}_{k+1} = \boldsymbol{T}_k + \alpha \nabla \mathcal{J}$
  5. Iterate until convergence
- To control the size of point clouds, we can subsample points (from one or both sets)
  - The basic method uses every point
  - Uniform (voxel-based) sub-sampling preserves the structure of the point cloud
  - Random sampling preserves the density but may lose structure
  - Feature-based sampling tries to identify structural features to preserve traits of the environment
  - Normal-space sampling tries to sample a subset with surface normals distributed as uniformly as possible, to preserve important information
    * This is better for mostly smooth areas with sparse features

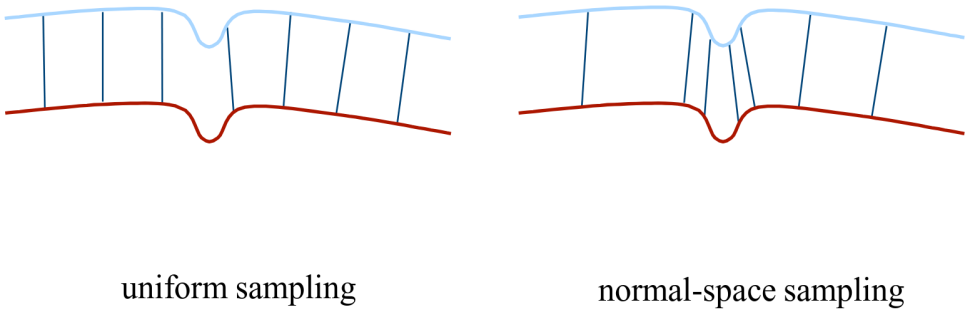

Figure 1: Comparison of uniform/voxel-based subsampling vs. normal-space sampling.

- We can use many other methods to weight point correspondences in our cost function
  - The basic method uses point-to-point correspondences
  - Point-to-plane (or point-to-line in 2D) can be performed by calculating surface normals, and only penalize errors in the direction of the normal; effectively lets planes slide past each other

    * This is especially useful since LiDAR has density varying with distance
    * $\mathcal{J} = \frac{1}{2}\sum_{j=1}^{J} w^j \left(\boldsymbol{p}_M^i - \boldsymbol{C}_k\left(\boldsymbol{p}_D^j - \boldsymbol{t}_k\right)\right)^T \boldsymbol{n}^j \boldsymbol{n}^{jT} \left(\boldsymbol{p}_M^i - \boldsymbol{C}_k\left(\boldsymbol{p}_D^j - \boldsymbol{t}_k\right)\right)$ where $\boldsymbol{n}$ are the surface normals
    * Slower to compute, but often faster convergence
  - Plane-to-plane follows a similar idea but calculates surface normals in both sets
    * Use covariances $\boldsymbol{\Sigma}_\varepsilon = \begin{bmatrix} \varepsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ for some small ε
      - This makes the error weighted much more in the direction of the surface normal
    * Transform each covariance: $\boldsymbol{\Sigma}_i^M = \boldsymbol{C}(\boldsymbol{n}_i^M)\boldsymbol{\Sigma}_\varepsilon \boldsymbol{C}(\boldsymbol{n}_i^M)^T, \boldsymbol{\Sigma}_j^D = \boldsymbol{C}(\boldsymbol{n}_j^D)\boldsymbol{\Sigma}_\varepsilon \boldsymbol{C}(\boldsymbol{n}_j^D)^T$
      - $\boldsymbol{C}$ is a rotation matrix that aligns $\boldsymbol{n}$ with the x axis (we don't care about the other directions)
    * $\mathcal{J} = \frac{1}{2}\sum_{j=1}^{J} \left(\boldsymbol{p}_M^i - \boldsymbol{C}_k\left(\boldsymbol{p}_D^j - \boldsymbol{t}_k\right)\right)^T (\boldsymbol{\Sigma}_i^M + \boldsymbol{C}_k \boldsymbol{\Sigma}_j^D \boldsymbol{C}_k^T)^{-1} \left(\boldsymbol{p}_M^i - \boldsymbol{C}_k\left(\boldsymbol{p}_D^j - \boldsymbol{t}_k\right)\right)$
      - Note that we had to transform $\boldsymbol{\Sigma}_j^D$ further, since it's in a different frame
  - This can be extended to curves but is less common
  - The *normal distribution transform* (NDT) tries to match clusters to clusters
    * Divide the scan into cells and model the surface within each cell as a Gaussian
      - $\rho_{c_i}(\boldsymbol{p}) = \exp\left(-\frac{(\boldsymbol{p}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{p}-\boldsymbol{\mu}_i)}{2}\right)$
    * Each point in the data point cloud is scored based on the probability density of the Gaussian in its cell
      - $\Lambda(\boldsymbol{T}) = -\sum_{\boldsymbol{p}_D^j \in D} \rho_c(\boldsymbol{T}(\boldsymbol{p}_D^j))$ where ρ_c is taken for the grid that $\boldsymbol{p}_D^j$ falls into
    * In highly concentrated surfaces, e.g. walls, we get narrow Gaussians, which helps a lot with alignment; however corners get blurred
      - We can use clustering to dynamically allocate the cell sizes to reduce the blurring issue

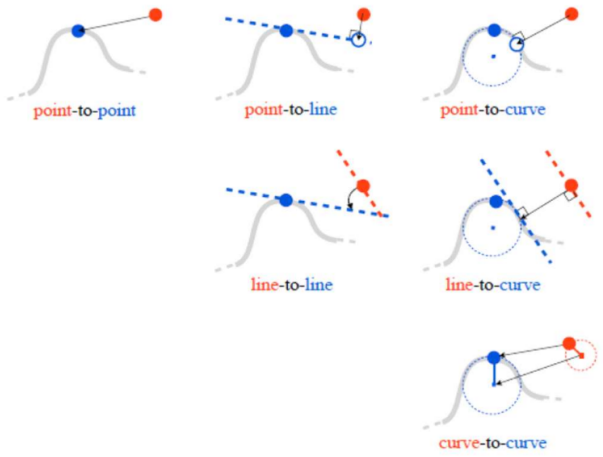

Figure 2: Different point correspondence methods.

- To compute surface normals, we find a neighbourhood of local points around each point, and identify principal components
  - Assume a linear model with best fit hyperplane $a^T p = c$ where a, c are parameters, p are the points
  - Minimize $\frac{1}{k}\sum_{i=1}^{k}(\boldsymbol{a}^T \boldsymbol{p}_i - c)^2$

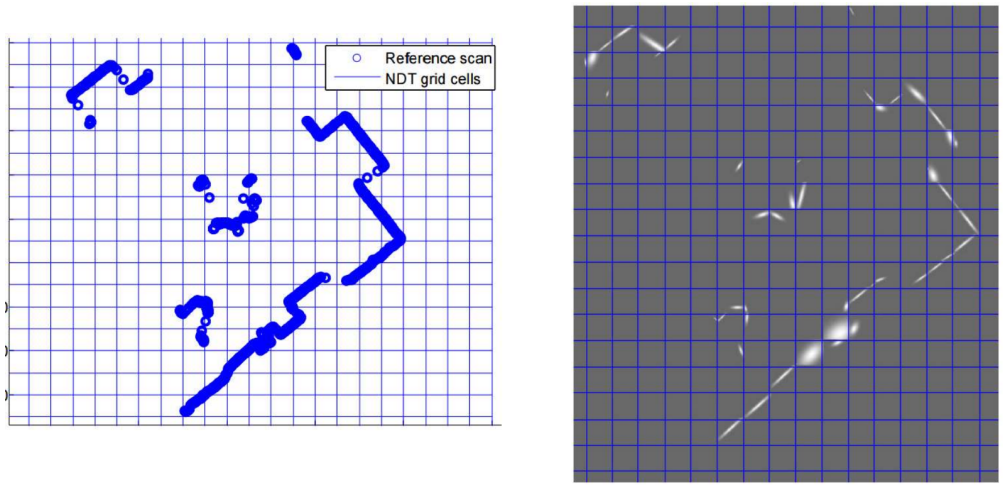

Figure 3: NDT visualization.

  - This reduces to PCA of the outer product $M = \frac{1}{k}\sum_{i=1}^{k}(\boldsymbol{p}_i - \bar{\boldsymbol{p}})(\boldsymbol{p}_i - \bar{\boldsymbol{p}})^T$
    * The eigenvectors indicate the principal directions of variance
    * The eigenvector associated with the smallest eigenvalue is the direction of least variance, which is the surface normal direction
- For data association, we also have options:
  - Closest point (basic option)
  - Closest compatible point (e.g. checking that normals/colours/curvature/etc are within range)
  - Normal shooting travels along the surface normal and selects the first point (i.e. closest point along the surface normal)
  - Outlier rejection with e.g. RANSAC, or checking with the previous transformation
    * We can reject points that have distances that are not consistent with neighbours, but must be tuned to avoid getting stuck in local minima
- Practical considerations:
  - Point cloud distortion from movement of the vehicle during a scan
    * We can usually get a motion estimate during the data capture to undistort it
  - Anisotropic filtering to remove points that don't provide much information
    * Filtering based on distance
    * Same idea as subsampling the point cloud
  - Optimizations for real-time constraint
    * Coarse-to-fine alignment (start with downsampled point cloud that can be quickly registered, progressively use more points to refine)
  - Quality estimation to detect incorrect registrations
    * Using covariance of the final point associations
    * Using overlap between consecutive scans or features
- Note that all methods we've discussed are local solutions; for global point cloud registration we usually rely on geometric feature matching