Lecture 8, Sep 26, 2025

Camera Pose Estimation (Perspective-n-Point)

- The *pose estimation problem* is the estimation of the pose of an object (or camera) from 3D-2D correspondences
  - Typically we know where the points lie on the object in 3D, and we also know where those points appear in our image
  - This is known as the *perspective-n-point* or PnP problem
    - For 3 points, this is known as P3P
    - 3 points is the minimum number of points we need to get a solution, but often we want to use more points to remove ambiguity and reject noise
  - This process can be used to estimate the camera pose for extrinsic calibration; intrinsic parameters can be estimated at the same time
    - This is how camera calibration with a checkerboard works
- There are linear and nonlinear algorithms for this
  - The linear case locally linearizes the problem and may not be very accurate if the initial guess is bad
  - The nonlinear algorithm uses an initial guess and iterates to find the solution
  - We often start from the linear algorithm and then use nonlinear algorithm to refine
- Solving for $\boldsymbol{P} = \boldsymbol{K}\begin{bmatrix}\boldsymbol{C} & \boldsymbol{t}\end{bmatrix}$ requires at least 6 correspondences (since there are 6 degrees of freedom), but if we have intrinsics already we only need 3
- The linear algorithm for solving PnP is the *direct linear transform* (DLT), which stacks a system of equations
  - We have $\boldsymbol{P} \in \mathbb{R}^{3\times 4}$ (the combination of the extrinsic and intrinsic matrices) with 12 unknowns
  - For each correspondence we can construct 2 equations from it; with 6 correspondences we can solve the whole system
    - $x_i = \dfrac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$
    - $y_i = \dfrac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$
    - Note the division due to normalization
  - Because the intrinsics matrix $\boldsymbol{K}$ is upper-triangular, we can do a QR factorization on $\boldsymbol{P}$ to recover the separate intrinsic and extrinsic matrices
- However, DLT does not impose constraints on the structure of the resulting matrices, namely the structure of the rotation matrix, so after QR factorization we often end up with a result that does not fit into our camera models, forcing us to make an approximation; this is why DLT is not an exact solution

Nonlinear Least Squares

- Nonlinear least squares (Gauss-Newton optimization) is an optimization approach we can use to solve this system
  - For nonlinear least squares, we wish to optimize $E(\boldsymbol{x}) = \dfrac{1}{2}\boldsymbol{e}(\boldsymbol{x})^T\boldsymbol{e}(\boldsymbol{x})$ where $\boldsymbol{e}(\boldsymbol{x})$ is some nonlinear function
    - Note $\boldsymbol{e}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y}$, i.e. the prediction minus the observation; the order is important, otherwise we end up maximizing the error instead!
    - In the linear case we can substitute $\boldsymbol{e}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ and expand the error function, then take a derivative to obtain the normal equation
  - For the nonlinear case we linearize around an initial guess (the operating point) $\boldsymbol{x}_{op}$
  - $\boldsymbol{e}(\boldsymbol{x}) \approx \boldsymbol{e}(\boldsymbol{x}_{op}) + \boldsymbol{J}_e\delta\boldsymbol{x}$ where $\boldsymbol{J}_e = \left.\dfrac{\partial\boldsymbol{e}}{\partial\boldsymbol{x}}\right|_{\boldsymbol{x}_{op}}$ is the Jacobian and $\delta\boldsymbol{x}$ is a small deviation
    - Now substitute this back into the error function and notice we get an expression very similar to the linear form, so we can use the same techniques to solve this

  - We can solve the linearized system, and add the $\delta \boldsymbol{x}$ to our initial operating point $\boldsymbol{x}_{op}$ to get the next linearization point
  - Do this until our Jacobian becomes sufficiently small which means we have converged
  - We are essentially approximating the cost function as a quadratic at each step and optimizing the quadratic for a local solution
- Important notes for nonlinear least squares:
  - Choosing good initial guesses is important, otherwise the optimization process can get trapped in a local minimum
  - The states we are solving for must exist in a vector space (i.e. we cannot apply constraints, since then the vector space is no longer closed)
- For multiple errors, we sum over all the errors, so in the normal equation we sum over $\boldsymbol{J}^T\boldsymbol{J}$ and $\boldsymbol{J}^T\boldsymbol{e}$ and multiply in the end
  - $E(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{N} \boldsymbol{e}_i(\boldsymbol{x})^T \boldsymbol{e}_i(\boldsymbol{x})$
  - $\delta\boldsymbol{x}^* = -\left(\sum_{i=1}^{N} \boldsymbol{J}_{e_i}^T \boldsymbol{J}_{e_i}\right)^{-1}\left(\sum_{i=1}^{N} \boldsymbol{J}_{e_i}^T \boldsymbol{e}_i(\boldsymbol{x}_{op})\right)$
  - Note we need to reevaluate the Jacobian for each measurement i, since the linearization point is all different!
- Often we have associated uncertainties for each error, so we can do a weighted version of least squares, so $E(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{N} \boldsymbol{e}_i(\boldsymbol{x})^T \boldsymbol{W}_i \boldsymbol{e}_i(\boldsymbol{x})$
  - Now we have $\delta\boldsymbol{x}^* = -\left(\sum_{i=1}^{N} \boldsymbol{J}_{e_i}^T \boldsymbol{W}_i \boldsymbol{J}_{e_i}\right)^{-1}\left(\sum_{i=1}^{N} \boldsymbol{J}_{e_i}^T \boldsymbol{W}_i \boldsymbol{e}_i(\boldsymbol{x}_{op})\right)$
  - $\boldsymbol{W}_i$ are symmetric matrices, one describing the weight for each measurement
  - Often we want scalar weights so $\boldsymbol{W}_i$ for each measurement i is just a multiple of the identity
  - For vector-valued observations, we can use the inverse of the measurement covariance matrix $\boldsymbol{\Sigma}$, known as the *information matrix*
- Note nonlinear least squares is equivalent to a maximum likelihood estimate if we assume our data has additive noise drawn from IID multivariate zero-mean Gaussians; weights are the information matrices of each noise Gaussian in this case
- To use NLS for pose estimation, we optimize the reprojection error $\boldsymbol{e}_i = \boldsymbol{x}_i - f(\boldsymbol{p}_i; \boldsymbol{C}, \boldsymbol{t}, \boldsymbol{K})$
  - f is our projection function which takes p_i, converts it to the camera frame using the extrinsic calibration, then to pixel space using the intrinsics and normalizes it
  - But, rotation matrices have constraints, so we can't use Gauss-Newton as-is; we need to either express $\boldsymbol{C}$ in a way that ensures it is a rotation matrix, or modify our update so that $\boldsymbol{C}$ remains orthogonal
- This leads to the *Wahba problem*, which involves identifying a rotation matrix $\boldsymbol{C}$ between frames given corresponding unit vector measurements $\boldsymbol{u}_i, \boldsymbol{v}_i$ in two frames
  - The cost function is $E(\boldsymbol{C}) = \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{C}\boldsymbol{u}_i - \boldsymbol{v}_i)^T(\boldsymbol{C}\boldsymbol{u}_i - \boldsymbol{v}_i) = \frac{1}{2}\sum_{i=1}^{N} \boldsymbol{e}_i(\boldsymbol{C})^T \boldsymbol{e}_i(\boldsymbol{C})$
  - Approach 1: Euler angles
    * Optimize over 3 Euler angles instead
    * However, this runs the risk of Gimbal lock – if our solution ends up near points with Gimbal lock, we run into numerical sensitivity issues or our solution may collapse entirely
      - This can work if we have good initial guesses
    * Let $\boldsymbol{C}(\boldsymbol{\theta}) = \boldsymbol{C}_3(\theta_3)\boldsymbol{C}_2(\theta_2)\boldsymbol{C}_1(\theta_1)$, then $\boldsymbol{e}_i(\boldsymbol{\theta}_{op} + \delta\boldsymbol{\theta}) = \boldsymbol{e}_i(\boldsymbol{\theta}_{op}) + \boldsymbol{J}_{e_i}\delta\boldsymbol{\theta}$
    * The Jacobian $\boldsymbol{J}_{e_i} = \frac{\partial \boldsymbol{C}\boldsymbol{u}_i}{\partial \boldsymbol{\theta}}$ can be computed in each column as follows:
      - $\frac{\partial \boldsymbol{C}\boldsymbol{u}_i}{\partial \theta_3} = [\boldsymbol{1}_3]_\times \boldsymbol{C}_3(\theta_3)\boldsymbol{C}_2(\theta_2)\boldsymbol{C}_1(\theta_1)\boldsymbol{u}_i$

- $\dfrac{\partial \boldsymbol{C}\boldsymbol{u}_i}{\partial \theta_2} = \boldsymbol{C}_3(\theta_3)\left[\mathbf{1}_2\right]_\times \boldsymbol{C}_2(\theta_2)\boldsymbol{C}_1(\theta_1)\boldsymbol{u}_i$
- $\dfrac{\partial \boldsymbol{C}\boldsymbol{u}_i}{\partial \theta_1} = \boldsymbol{C}_3(\theta_3)\boldsymbol{C}_2(\theta_2)\left[\mathbf{1}_1\right]_\times \boldsymbol{C}_1(\theta_1)\boldsymbol{u}_i$
- Note $\mathbf{1}_i$ is a zero vector with 1 in the ith spot

– Approach 2: use axis-angle

    * To avoid gimbal lock we keep the operating point $\boldsymbol{C}_{op}$ in matrix form, and consider a perturbation $\boldsymbol{C}(\delta\boldsymbol{\phi})$, so the update becomes $\boldsymbol{C}_{op} \leftarrow \boldsymbol{C}(\delta\boldsymbol{\phi})\boldsymbol{C}_{op}$

    * Recall the Rodrigues formula: $\boldsymbol{C}(\boldsymbol{\phi}) = 1 + \sin\phi\left[\hat{\boldsymbol{n}}\right]_\times + (1-\cos\phi)\left[\hat{\boldsymbol{n}}\right]_\times^2$

- For small angles $\boldsymbol{\phi}$ we approximate $\sin\phi = 0, \cos\phi = 1$
- $\boldsymbol{C}(\delta\boldsymbol{\phi}) \approx 1 + \delta\phi\left[\hat{\boldsymbol{n}}\right]_\times = 1 + \left[\delta\boldsymbol{\phi}\right]_\times$

    * Substitute the approximation for $\boldsymbol{C}(\delta\boldsymbol{\phi})$:

- $$\begin{aligned}\boldsymbol{e}_i(\delta\boldsymbol{\phi}) &= \boldsymbol{C}\boldsymbol{u}_i - \boldsymbol{v}_i \\ &= \boldsymbol{C}(\delta\boldsymbol{\phi})\boldsymbol{C}_{op}\boldsymbol{u}_i - \boldsymbol{v}_i \\ &= (1 + \left[\delta\boldsymbol{\phi}\right]_\times)\boldsymbol{C}_{op}\boldsymbol{u}_i - \boldsymbol{v}_i \\ &= \boldsymbol{C}_{op}\boldsymbol{u}_i - \boldsymbol{v}_i + \left[\delta\boldsymbol{\phi}\right]_\times \boldsymbol{C}_{op}\boldsymbol{u}_i \\ &= \boldsymbol{e}_i(\boldsymbol{C}_{op}) - \left[\boldsymbol{C}_{op}\boldsymbol{u}_i\right]_\times \delta\boldsymbol{\phi} \\ &= \boldsymbol{e}_i(\boldsymbol{C}_{op}) + \boldsymbol{J}_{e_i}\delta\boldsymbol{\phi}\end{aligned}$$

    * Therefore the Jacobian is $\boldsymbol{J}_{e_i} = -\left[\boldsymbol{C}_{op}\boldsymbol{u}_i\right]_\times$, and with this we can use the normal equation to compute the update for $\delta\boldsymbol{\phi}^*$ and update the operating point