Lecture 13, Oct 17, 2025

Incremental SfM and Visual Odometry

- Suppose we're processing images as they arrive, can we build up the SfM problem incrementally instead of all at once as in bundle adjustment?
- This problem is known as *incremental SfM*, or *visual odometry* (VO) and *visual SLAM*
  - Visual odometry only computes the positions, while in visual SLAM we also build up a scene representation and potentially incorporates loop closures
  - Visual odometry methods compute the *egomotion* between frames, i.e. the motion of the camera/robot platform relative to the world
  - We will discuss stereo visual odometry, since we need depth information when doing frame-to-frame motion estimation
- Visual odometry can be used in environments where normal wheel odometry is infeasible/inaccurate, e.g. high-slip environments like sand, aerial vehicles, or where we require higher short-term accuracy than wheel odometry can offer
- VO algorithms can be grouped into 4 categories, depending on whether they use monocular or stereo images, and whether they use image features or directly use pixel intensities
  - Feature-based VO algorithms run feature extraction and matching, while intensity based methods directly try to match pixel intensities from one image to the next
  - Importantly, stereo measurements have an invertible observation model, so we can take points in one frame and use the inverse model to calculate where we should expect it in the next frame
- Define the combined stereo camera model (note the origin is defined as the middle of the cameras, so the cameras centres are at $x = \pm b/2$
  - Assuming same intrinsics for both cameras, $\boldsymbol{M} = \begin{bmatrix} f_u & 0 & c_u & f_u\frac{b}{2} \\ 0 & f_v & c_v & 0 \\ f_u & 0 & c_u & -f_u\frac{b}{2} \\ 0 & f_v & c_v & 0 \end{bmatrix}, \boldsymbol{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$
  - The forward model is $\begin{bmatrix} u_l & v_l & u_r & v_r \end{bmatrix}^T = \boldsymbol{f}(\boldsymbol{p}) = \boldsymbol{M}\frac{1}{z}\boldsymbol{p}$ where (u_l, v_l) and (u_r, v_r) are the rectified pixel coordinates
    * Jacobian given by $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}} = \boldsymbol{M}\frac{1}{p_3}\begin{bmatrix} 1 & 0 & -p_1/p_3 & 0 \\ 0 & 1 & -p_2/p_3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -p_4/p_3 & 1 \end{bmatrix}$
  - Inverse model: $\boldsymbol{\rho} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \boldsymbol{g}(y) = \frac{b}{u_l - u_r}\begin{bmatrix} \frac{1}{2}(u_l + u_r) - c_u \\ \frac{f_u}{f_v}\left(\frac{1}{2}(v_l + v_r) - c_v\right) \\ f_u \end{bmatrix}$
    * This is only possible to do for a stereo camera since we have depth information
    * Jacobian: $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{y}} = \frac{b}{(u_l - u_r)^2}\begin{bmatrix} -u_r + c_u & 0 & u_l - c_u & 0 \\ -\frac{f_u}{f_v}\left(\frac{1}{2}(v_l + v_r) - c_v\right) & \frac{f_u}{2f_v} & \frac{f_u}{f_v}\left(\frac{1}{2}(v_l + v_r) - c_v\right) & \\ \frac{f_u}{2f_v} & -f_u & 0 & f_u \quad 0 \end{bmatrix}$
- The VO problem involves finding the relative transformation between frames $\boldsymbol{T}_{ba}$, given an image captured in frame a and an image in frame b
- In a feature-based stereo VO pipeline, we run the standard stereo matching, then feature detection and matching of features between frames; then we can do outlier rejection (very important) and then solve for the pose transformation, potentially incorporating other sensors into the mix
- To compute the pose transformation between frames, we can try to align the point clouds we get from the two frames (since the stereo model is invertible)

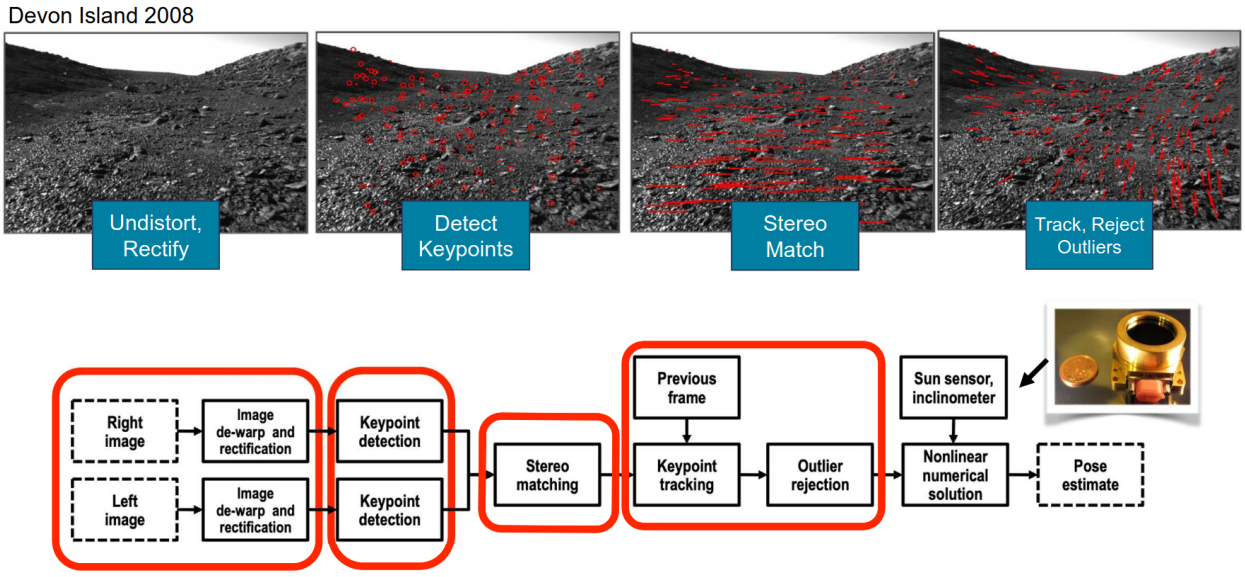

Figure 1: Feature-based stereo VO pipeline.

  - Assuming we know correspondences perfectly, there is a closed-form solution
  - Let $\boldsymbol{p}_a^j, \boldsymbol{p}_b^j$ be the points in frames $\vec{\mathcal{F}}_a$ and $\vec{\mathcal{F}}_b$ respectively; we wish to minimize a squared cost function with respect to the rotation $\boldsymbol{C}_{ba}$ and translation $\boldsymbol{r}_a$
  - $E(\boldsymbol{C}_{ba}, \boldsymbol{r}_a) = \frac{1}{2}\sum_{j=1}^{J} w_j \left(\boldsymbol{p}_b^j - \boldsymbol{C}_{ba}(\boldsymbol{p}_a^j - \boldsymbol{r}_a)\right)^T \left(\boldsymbol{p}_b^j - \boldsymbol{C}_{ba}(\boldsymbol{p}_a^j - \boldsymbol{r}_a)\right)$
  - Each data point can have a weight, usually based on our confidence of that feature; this can be a scalar or matrix weight
- Scalar weighted point cloud alignment has an exact solution without iteration:
  1. Compute centroids $\boldsymbol{p}_a = \frac{\sum_{j=1}^{J} w^j \boldsymbol{p}_a^j}{\sum_{j=1}^{J} w^j}, \boldsymbol{p}_b = \frac{\sum_{j=1}^{J} w^j \boldsymbol{p}_b^j}{\sum_{j=1}^{J} w^j}$
     - The idea is to align the two point cloud centroids, so we only have an alignment (rotation) to solve for
  2. Compute the outer product matrix $\boldsymbol{W}_{ba} = \frac{1}{\sum_{j-1}^{J} w^j} \sum_{j=1}^{J} w^j (\boldsymbol{p}_b^j - \boldsymbol{p}_b)(\boldsymbol{p}_a^j - \boldsymbol{p}_a)^T$
  3. Compute the SVD of the outer product matrix, $\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T = \boldsymbol{W}_{ba}$
  4. Compute the rotation using the SVD, and then use the inverse of the rotation to compute the correct translation vector
     - $\boldsymbol{C}_{ba} = \boldsymbol{V} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\boldsymbol{U})\det(\boldsymbol{V}) \end{bmatrix} \boldsymbol{U}^T$
     - $\boldsymbol{r}_a = -\boldsymbol{C}_{ba}^T \boldsymbol{p}_b + \boldsymbol{p}_a$
     - Notice the translation is just subtracting the centroids (in the correct frame)
- Bonus: this can be used to align point clouds without registration by iterating, which is the ICP algorithm
- Due to the stereo geometry, we typically have much higher depth uncertainty than in x and y, and further points have more uncertainty; using simple scalar weights does not capture this complexity, so it's often inaccurate
  - Consider a Gaussian noise model (ignoring for now the long tail we discussed before)
  - $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{p}) + \boldsymbol{n}, \boldsymbol{n} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R})$ where $\boldsymbol{R} \in \mathbb{R}^{4\times4}$ is the pixel measurement noise covariance (in both cameras)

  - Linearize the inverse model: $\hat{\boldsymbol{\rho}} = \boldsymbol{g}(\boldsymbol{y}) = \boldsymbol{g}(\boldsymbol{f}(\boldsymbol{\rho}) + \boldsymbol{n}) \approx \boldsymbol{\rho} + \dfrac{\partial \boldsymbol{g}}{\partial \boldsymbol{y}} \boldsymbol{n}$
  - Therefore $\hat{\boldsymbol{\rho}} \sim \mathcal{N}(\boldsymbol{\rho}, \boldsymbol{G}\boldsymbol{R}\boldsymbol{G}^T)$ where $\boldsymbol{G} = \left.\dfrac{\partial \boldsymbol{g}}{\partial \boldsymbol{y}}\right|_{\boldsymbol{y}}$
- Now we can use the landmark covariances in the optimization as matrix weights
  - $E(\boldsymbol{C}_{ba}, \boldsymbol{r}_a) = \dfrac{1}{2} \sum_{j=1}^{J} \left(\boldsymbol{p}_b^j - \boldsymbol{C}_{ba}(\boldsymbol{p}_a^j - \boldsymbol{r}_a)\right)^T \boldsymbol{\Sigma}^j \left(\boldsymbol{p}_b^j - \boldsymbol{C}_{ba}(\boldsymbol{p}_a^j - \boldsymbol{r}_a)\right)$
  - $\boldsymbol{\Sigma}^j = \left(\boldsymbol{G}_b^j \boldsymbol{R}_b^j \boldsymbol{G}_b^{j\,T} + \boldsymbol{C}_{ba} \boldsymbol{G}_a^j \boldsymbol{R}_a^j \boldsymbol{G}_a^{j\,T} \boldsymbol{C}_{ba}^T\right)^{-1}$ where $\boldsymbol{G}_b^j = \left.\dfrac{\partial \boldsymbol{g}}{\partial \boldsymbol{y}}\right|_{\boldsymbol{f}(\boldsymbol{p}_b^j)}, \boldsymbol{G}_a^j = \left.\dfrac{\partial \boldsymbol{g}}{\partial \boldsymbol{y}}\right|_{\boldsymbol{f}(\boldsymbol{p}_a^j)}$ and $\boldsymbol{R}_b^j, \boldsymbol{R}_a^j$ are the pixel measurement covariances in the two cameras for each point
    * Note now we have covariances resulting from measurement errors in both frames, so we need to transform the uncertainty in frame a to frame b using $\boldsymbol{C}_{ba}$
  - With matrix weights, this no longer has a closed-form solution, so we use Gauss-Newton to iterate as usual
- For outlier rejection, we can first do RANSAC with scalar-weighted point cloud alignment to find the inlier set, then use matrix-weighted alignment to calculate the precise transformation
  - If we have other sensors, we can add another term to the cost function for point cloud alignment to incorporate it