CIS 5800

Machine Perception

Instructor: Lingjie Liu

Lec 20: April 16, 2025

Q1.2 (a)

for the parallel lines on the ground plane

a. All the vanishing points you can draw in the photo (i.e., excluding those at infinity) are on the same line. True

Otherwise, the answer is “False”

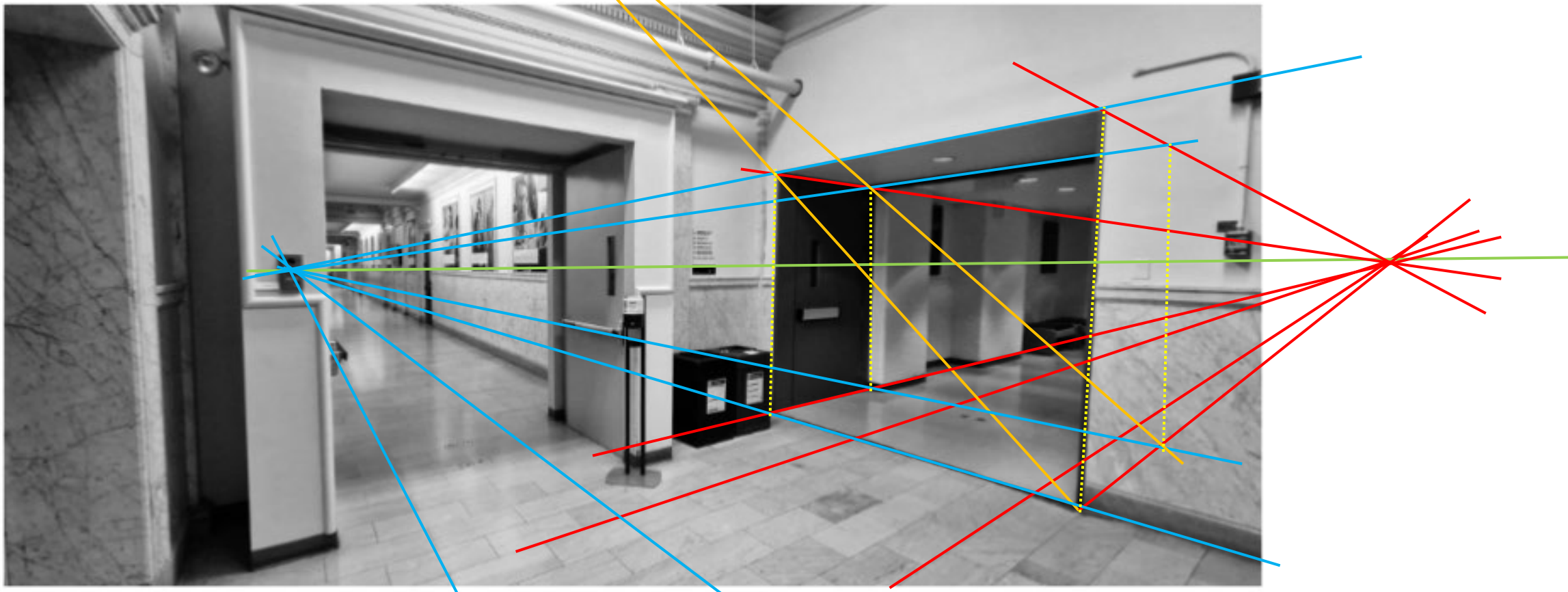
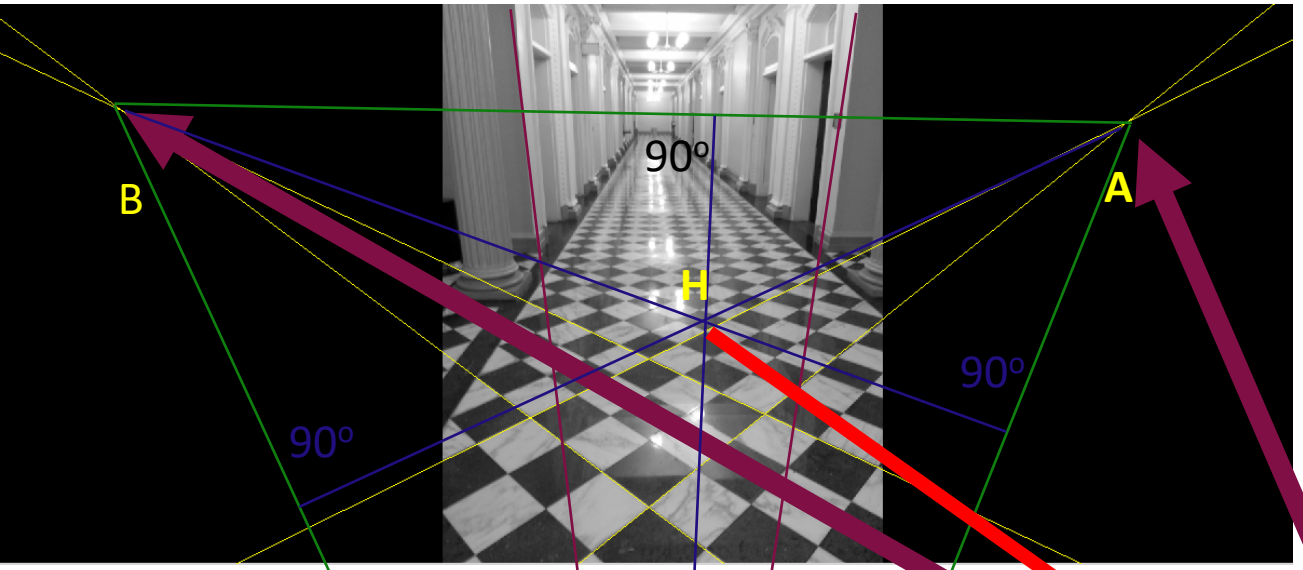


Figure 1: Hallway intersection at Penn

Q1.2 (a)



Online localization approaches

Input: a continuous visual observation stream, in real-time

- **"VO" (visual odometry):** mainly concerned with localizing and tracking the robot w.r.t its start position in an unknown environment (so we do not have a map of the environment), traditionally over small time windows ("short-term").
- **"Visual SLAM" (simultaneous localization and mapping):** jointly estimating 3D models of scenes and localizing the robot (camera) w.r.t. that scene, maintaining long-term consistency.
  - Short-term VO as defined above is often a module within Visual SLAM

We will focus on long-term VO through a widely used state-of-the-art system from 2016: ORB-SLAM

Basic pipeline for long-term *consistent* VO

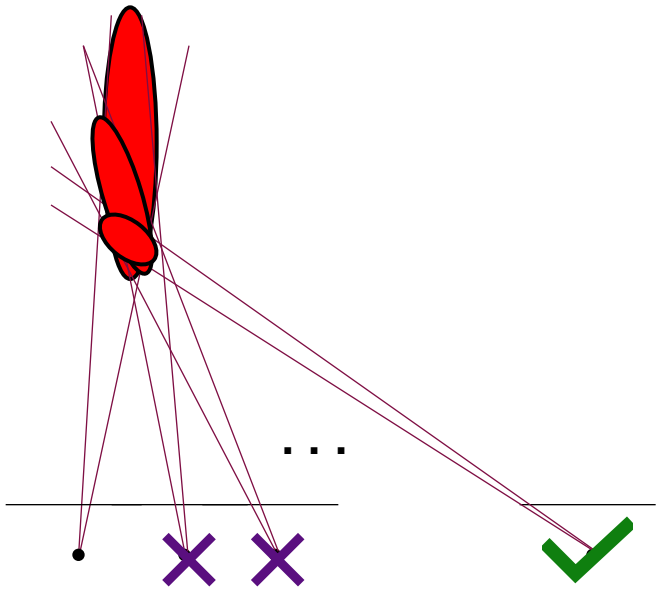
- **Step 1:** First do **2-view SfM** between first 2 frames from video (or any other method of estimating motion)
  - Outputs:
    - R_2, T_2 of 2nd camera w.r.t 1st
    - A 3D map with the 3D positions $\boldsymbol{X}_i$ of various points
- **Step 2:** For next frame k, identify 2D-2D correspondences with the previous frame.
  - These yield 2D-3D correspondences between this frame and the current map!
  - Then compute R_k, T_k with **PnP**. This is the VO output at time k.
  - Thus, the map serves as a memory of all past frames, enforcing consistency and reducing drift.
- **Step 3:** Improve **triangulated** 3D map $\boldsymbol{X}_i$ (using the likely bigger "baseline") and expand it, using correspondences that aren't yet in the map.
- Go back to Step 2 to process the next frame.

?

Good long-term visual odometry goes with map-building: **SLAM**

Note: Larger baselines => smaller triangulation error

Assuming some lack of precision in the 2D pixel locations of pixel correspondences, larger baselines lead to less propagation of that uncertainty into the 3D triangulated locations.



ORB-SLAM Overview

- Over the next few slides, we will walk through ORB-SLAM, a widely used algorithm that was proposed in 2016.

- This is not an exhaustive description, and it is not intended for you to be able to implement it from scratch.

- Instead, it is intended to give you a sense for what a modern SLAM system looks like.

ORB-SLAM Demo

ORB-SLAM

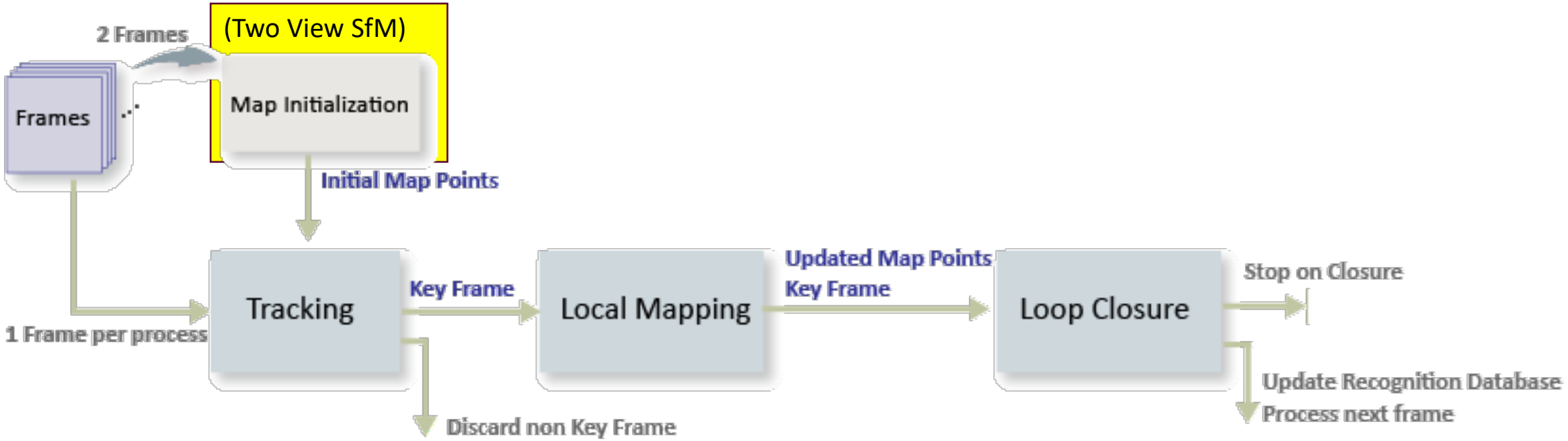
Raúl Mur-Artal, J. M. M. Montiel and Juan D. Tardós

{raulmur, josemari, tardos} @unizar.es

Instituto Universitario de Investigación
en Ingeniería de Aragón
Universidad Zaragoza

Universidad
Zaragoza

ORB-SLAM Pipeline



- **Map Points:** A list of 3-D points that represent the map of the environment reconstructed from the key frames.
- **“Key Frames:”** A subset of video frames that contain cues for localization and tracking. Two consecutive key frames should **overlap but also have sufficient visual change/long enough baseline**.

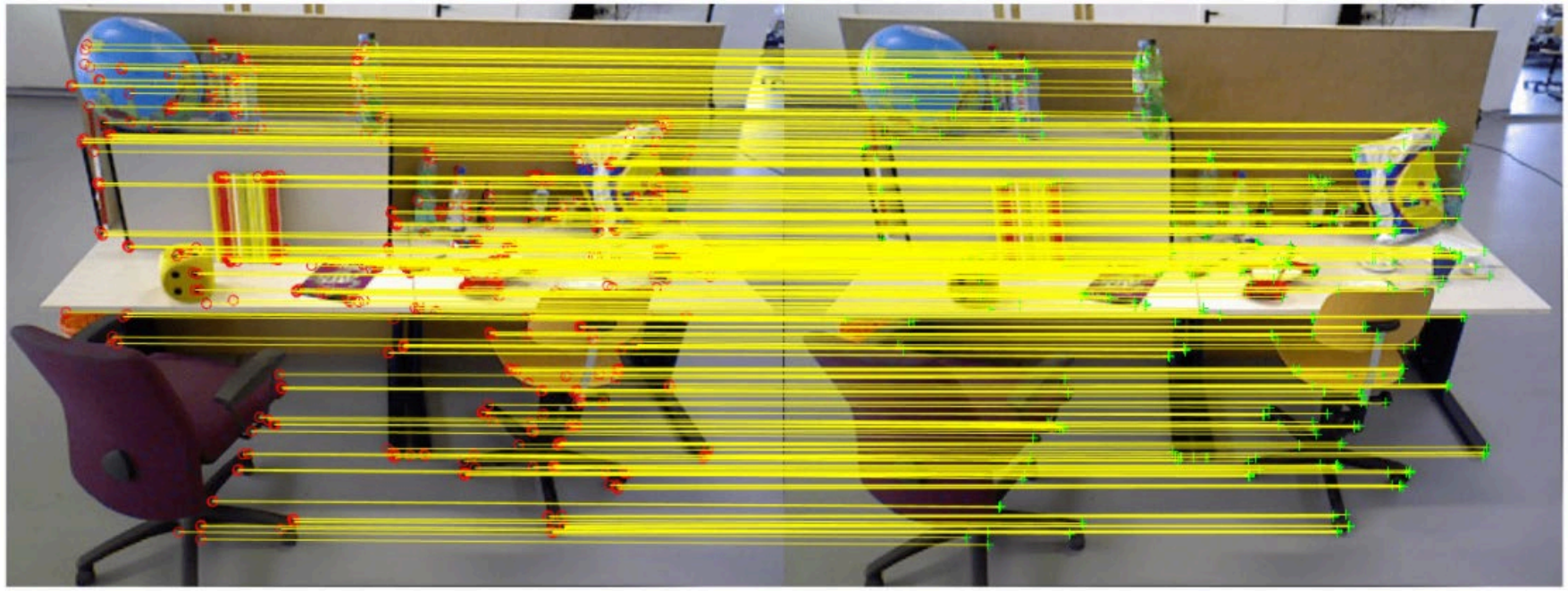
Map Initialization In ORB-SLAM: Overview

- **Wait for a good keyframe:**
  - Keep comparing between 1st frame (by default a keyframe) and kth frame until none of the following happens:
    - Homographic frames (rank-deficient system for E matrix estimation):
      - The scene is planar
      - There is no translation
    - Insufficient inliers for an estimate E (e.g. when using RANSAC)
  - Declare this as the next (2nd) keyframe
- Store keyframe features and poses computed with **2-view SfM**
- **Triangulate** and initialize a 3D map with these localized points.
- (Optional) **Refine** initial reconstruction using "2-frame bundle adjustment"
  - Non-linear joint maximization of the consistency between camera poses and 3D structures (more next class)

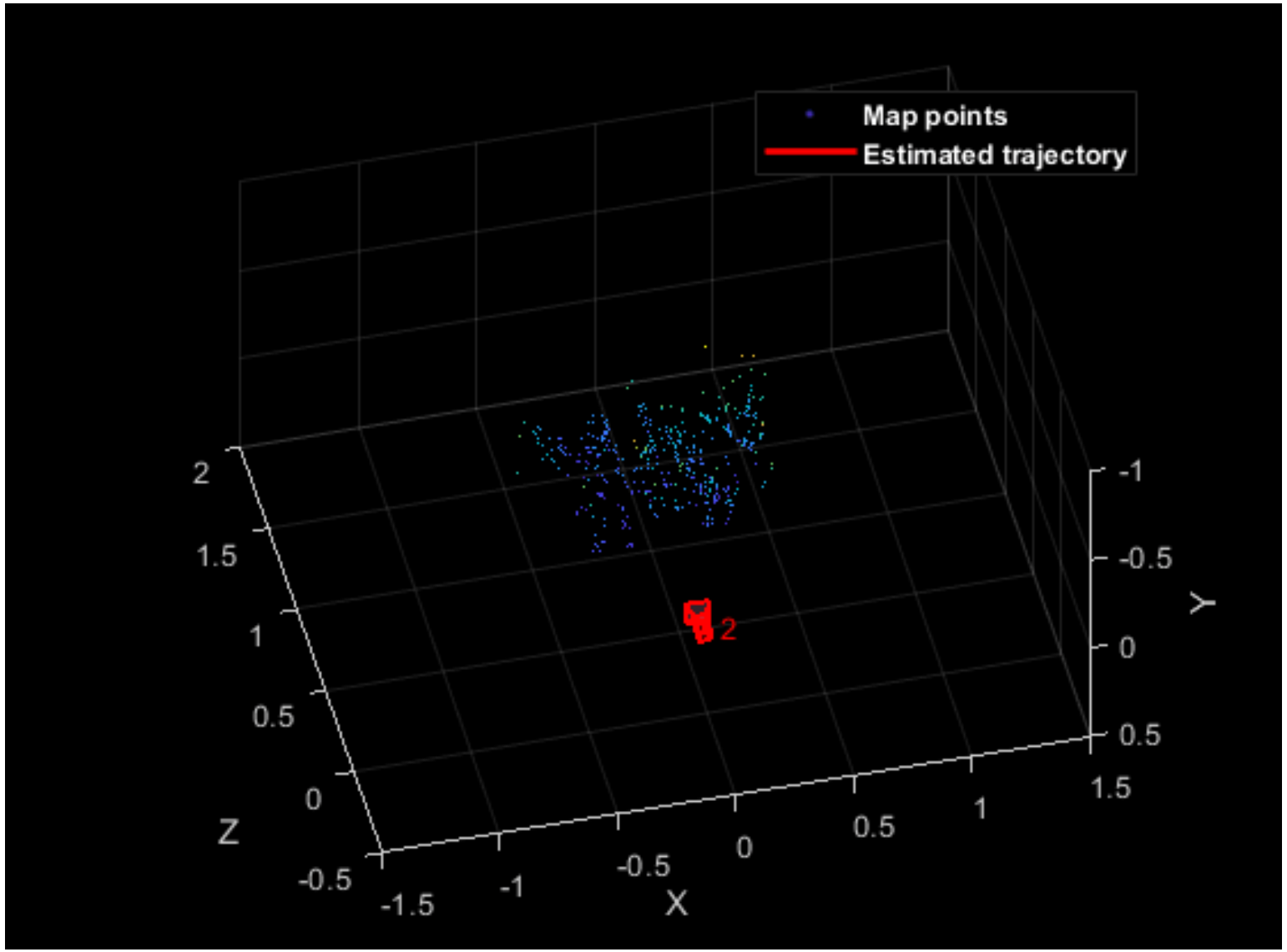
Example of a Detected Keyframe

An example from the default sequence in the ORB-SLAM package

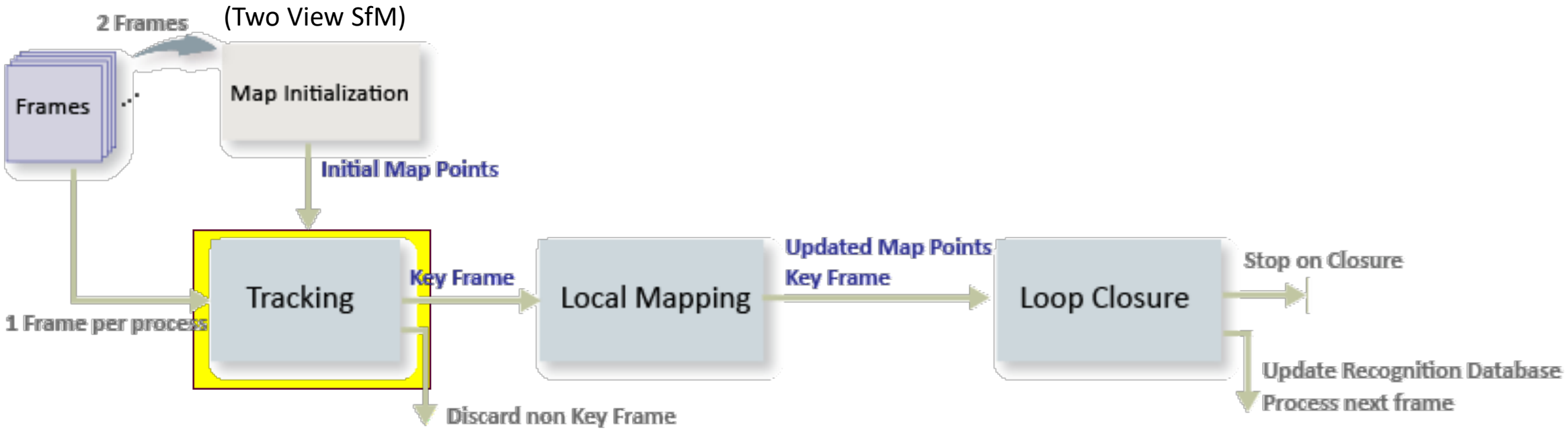
Frame 1 Frame 26



Example of Map Initialization Through Triangulation



ORB-SLAM Pipeline

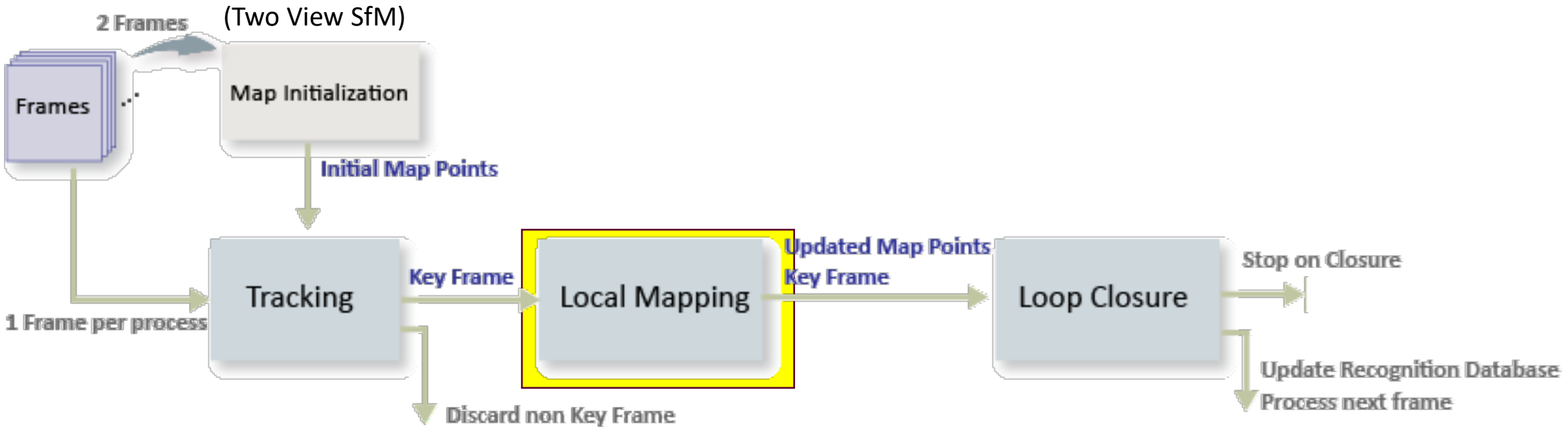


Tracking in ORB-SLAM: overview

1. **Appearance-based correspondences with previous keyframe:** Extract ORB features (like SIFT), match with previous keyframe features that are already incorporated into the 3-D map.
2. **PnP:** Estimate the camera pose wrt previous keyframe with PnP + RANSAC. (similar to one of the steps in map initialization)
3. **Geometry-based correspondences with previous keyframe:** Then given the camera pose, project the (unmatched) map points observed by the previous keyframe into the current frame and search for more feature correspondences. Not just appearance-based correspondences now.
4. **Refining PnP:** Refine camera pose by **performing a "pose-only bundle adjustment"**. (solving PnP non-linearly, details out of our scope)
5. This frame is a key frame if both of the following conditions are satisfied:
    1. At least 20 frames have passed since the previous keyframe or the current frame tracks fewer than 100 map points
    2. Fewer than 90% of the map points tracked by the current frame are also tracked by the reference key frame (i.e., there is significant change from the previous keyframe) (Keyframes should have overlap, but not be redundant.)

red are parameters that you can set

ORB-SLAM Pipeline

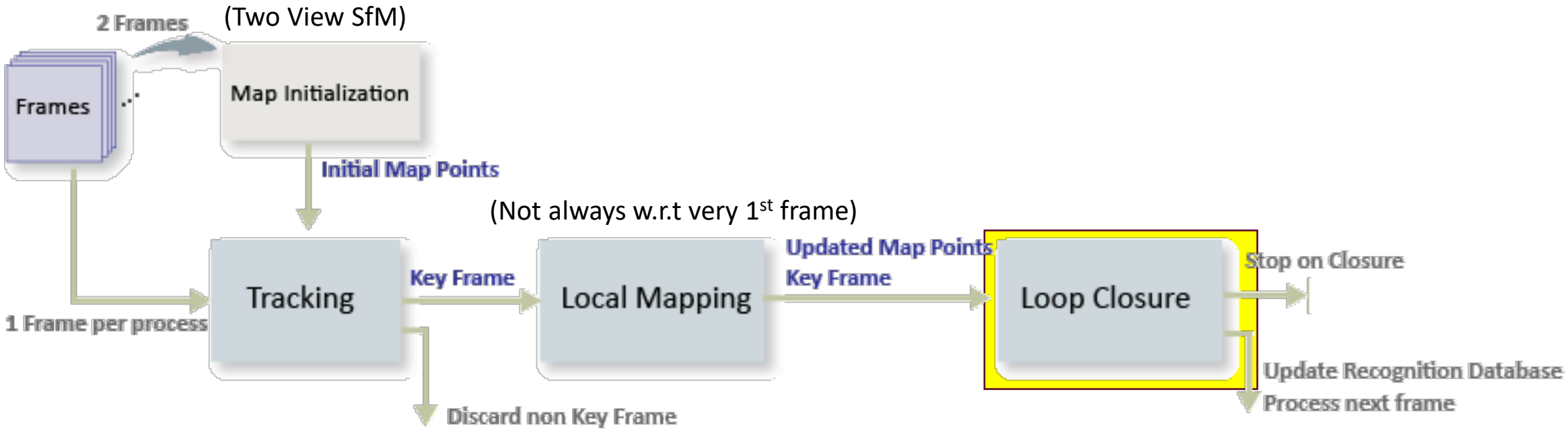


Local Mapping

- **Geometry-based correspondences with full map:** Project "local" map points visible in several "neighboring" keyframes into the current frame to search for more feature correspondences:
  - refine the camera pose even further
  - triangulation of new correspondences to grow the map further
- Local bundle adjustment (non-linear SfM) to refine these and achieve optimal reconstruction in the neighborhood of the current camera pose.

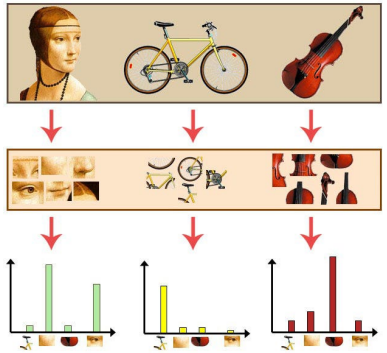
Lots of quality checking to avoid redundant keyframes (careful use of memory), and low-quality correspondences.

ORB-SLAM Pipeline



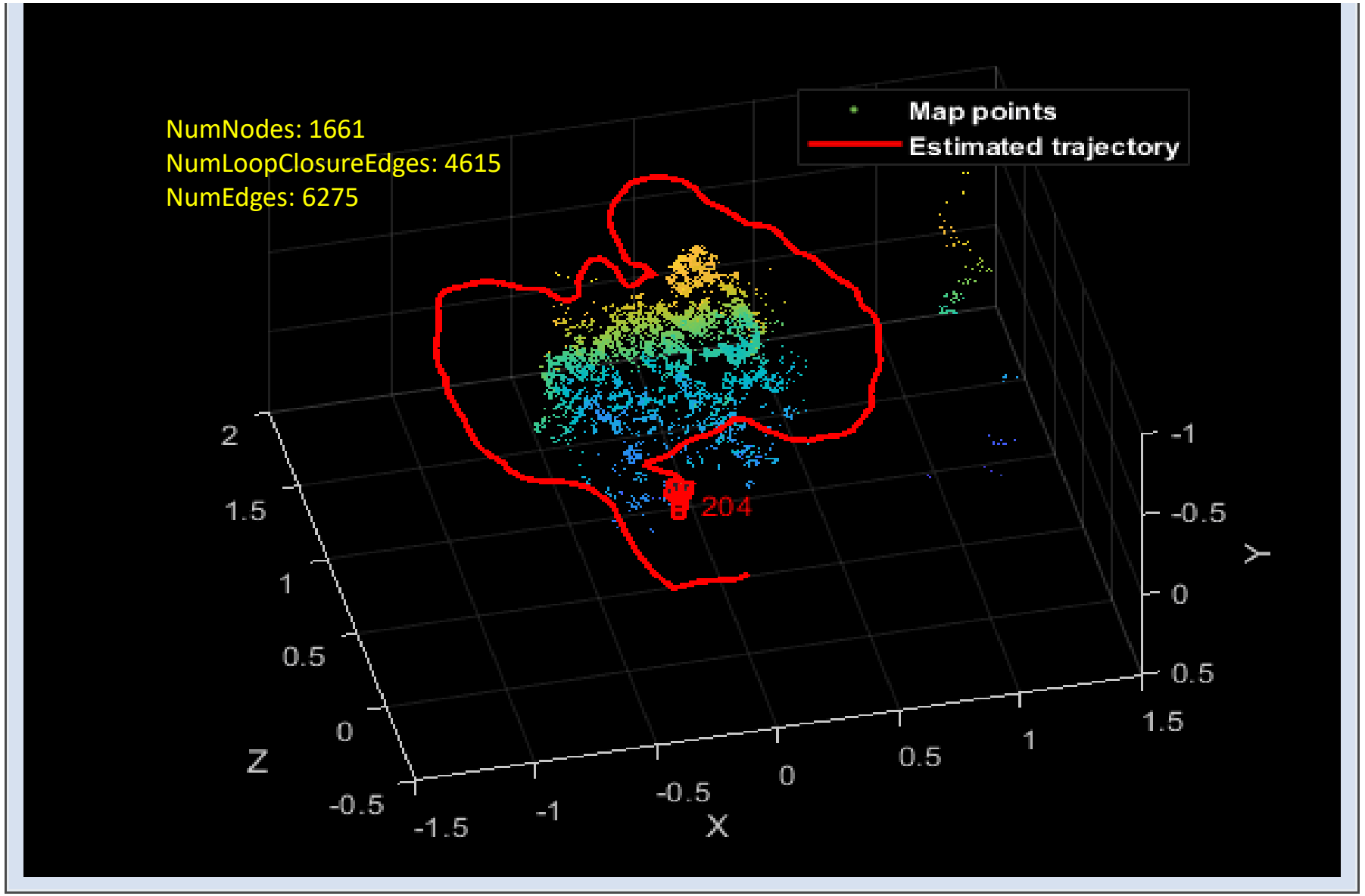
- **Covisibility Graph**: A graph consisting of key frame as nodes. Two key frames are connected by an edge if they share common map points. The weight of an edge is the number of shared map points.
- **Recognition Database:** A database used to recognize whether a place has been visited in the past. The database stores the visual word-to-image mapping based on the input bag of features. It is used to search for an image that is visually similar to a query image.

Loop Closure



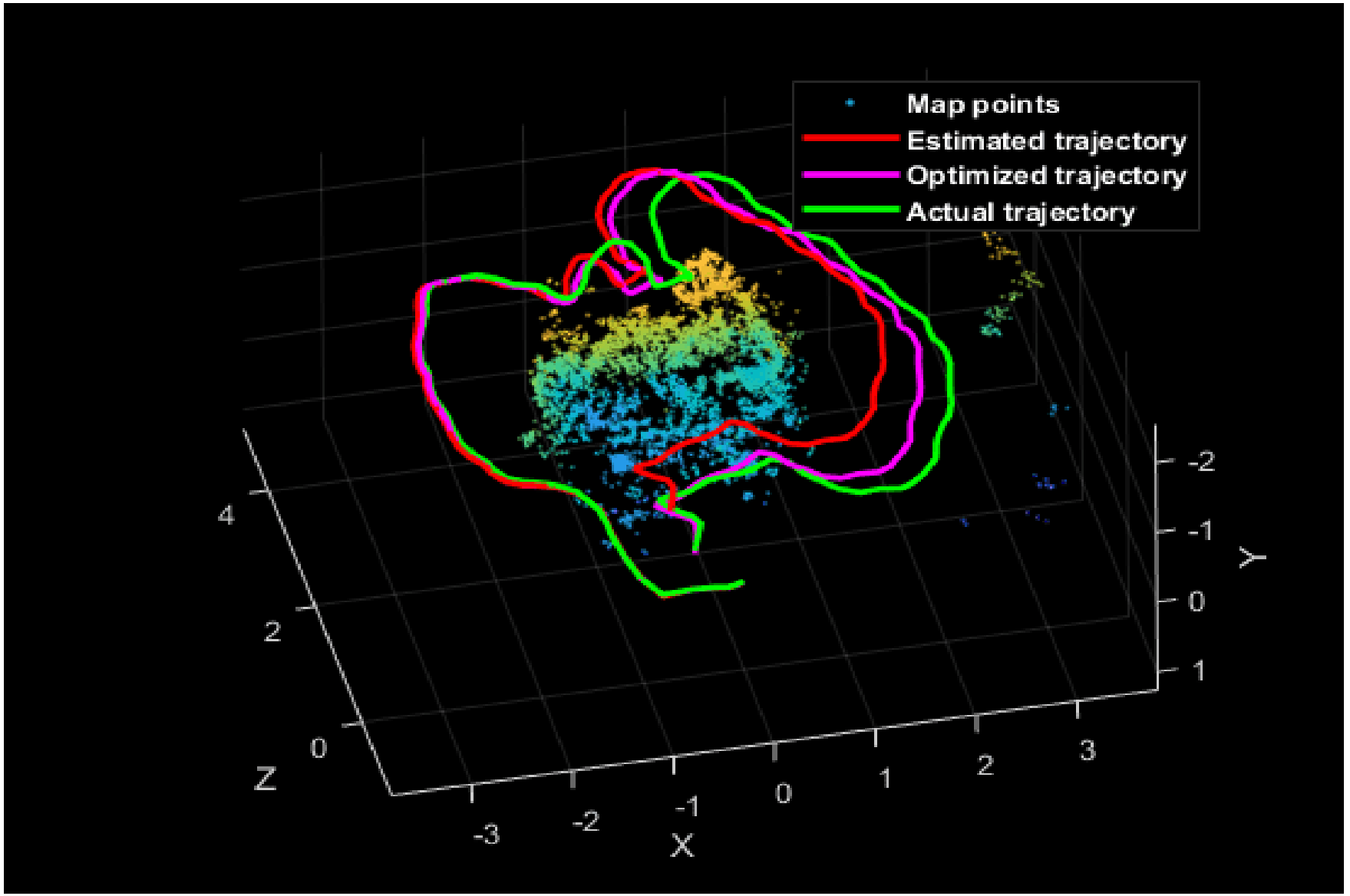
- How do I know that I come back at a visited place?
  - Place recognition: With deep learning, or other more classical recognition techniques like “Bag of Visual Words”, find past keyframes that look similar.
  - If visual similarity candidate is found, run a **geometric consistency check:** align the 3D points from current frame, and this keyframe, and check that they follow a similarity transformation.
- If loop closure declared
  - Run bundle adjustment (**graph pose optimization g2o**) to update all poses in essential graph (A subgraph of covisibility graph containing only edges with high weight, i.e. more shared map points).
    - Optimize for most consistent rotations and translations over the full graph

Loop closure before graph pose optimization

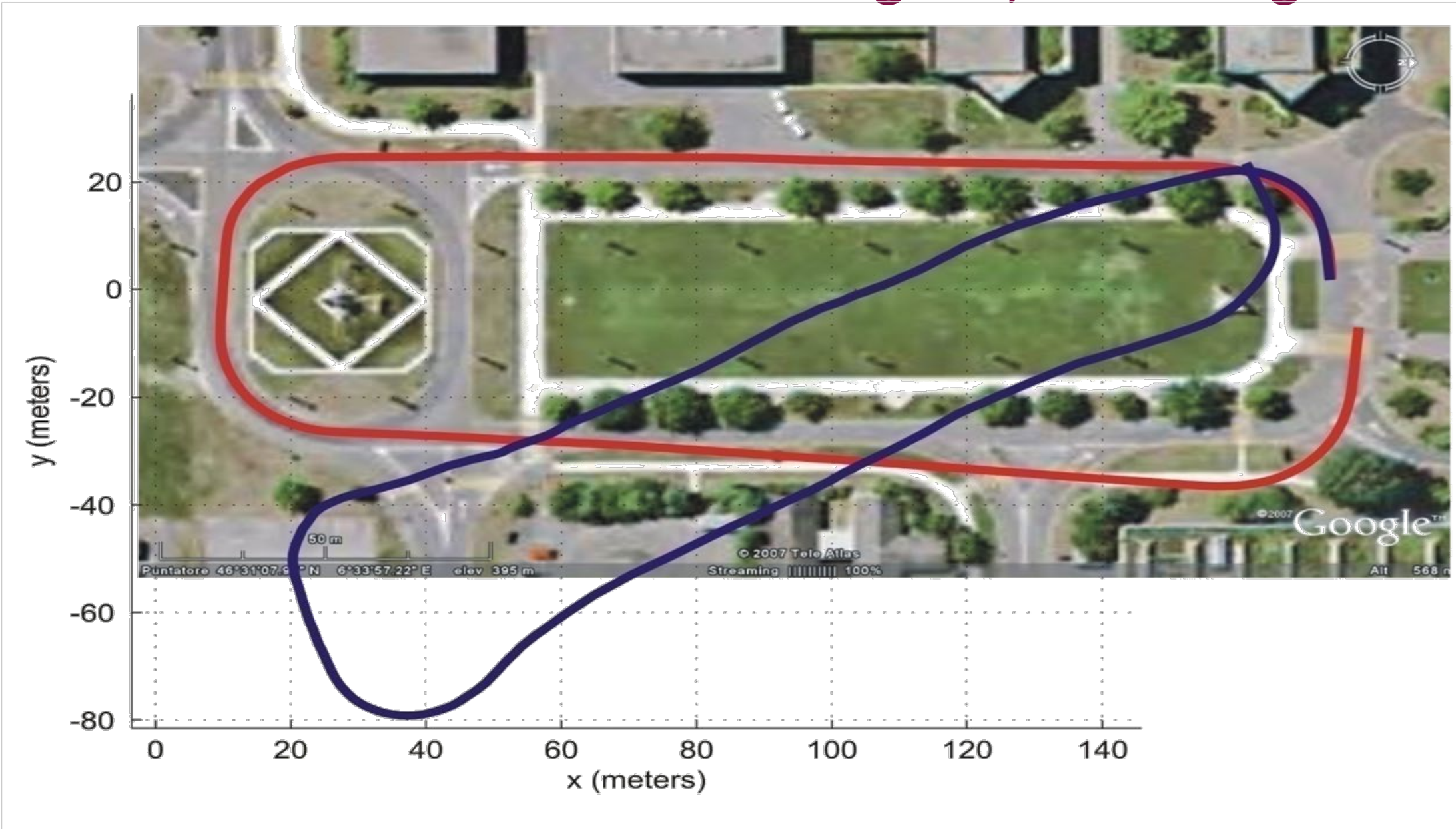



```
Loop edge added between keyframe: 6 and 204
```

Loop closure after graph pose optimization



Footnote 1: RANSAC is a workhorse throughout, makes a big difference!



Footnote 2: Visual Odometry >> IMU Odometry on Mars

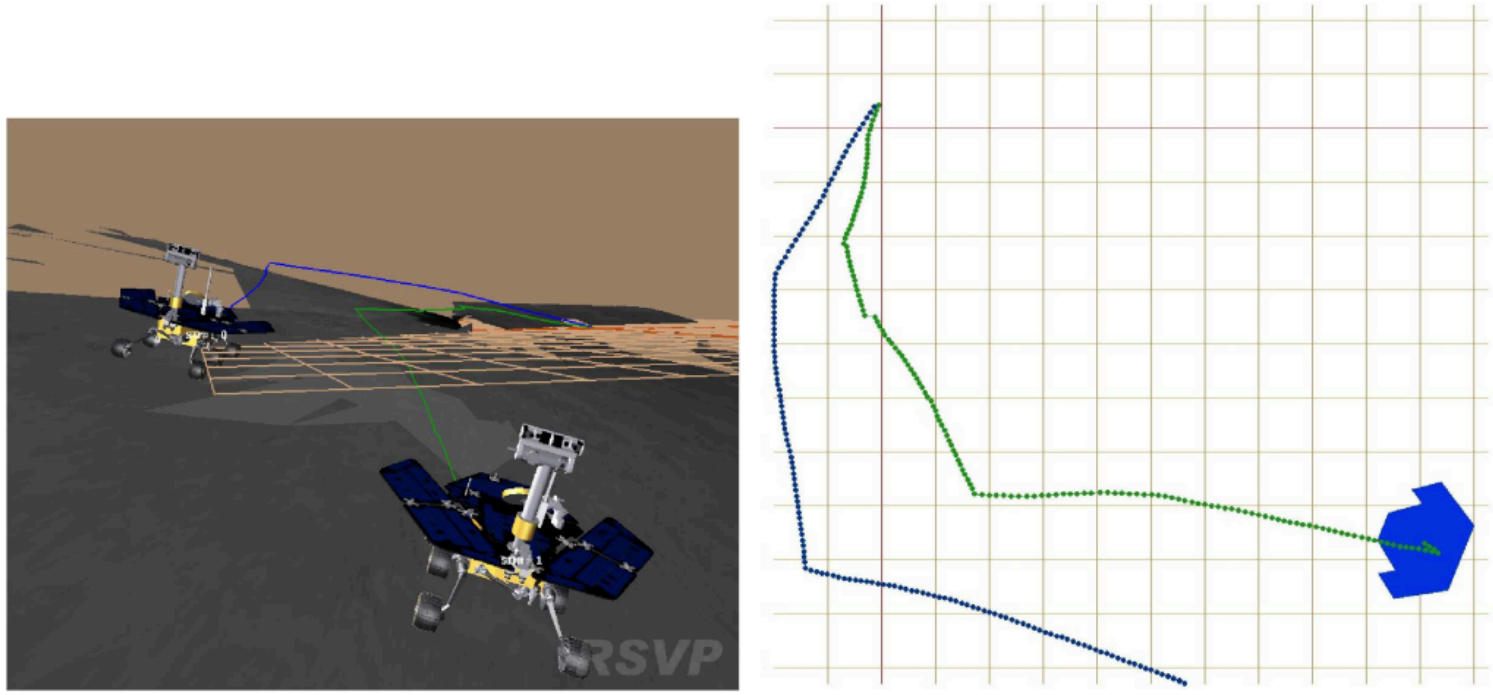


Figure 4: Views of Opportunity's 19 meter drive from Sol 188 through Sol 191. The inside path shows the correct, Visual Odometry updated location. The outside path shows how its path *would* have been estimated from the IMU and wheel encoders alone. Each cell represents one square meter.

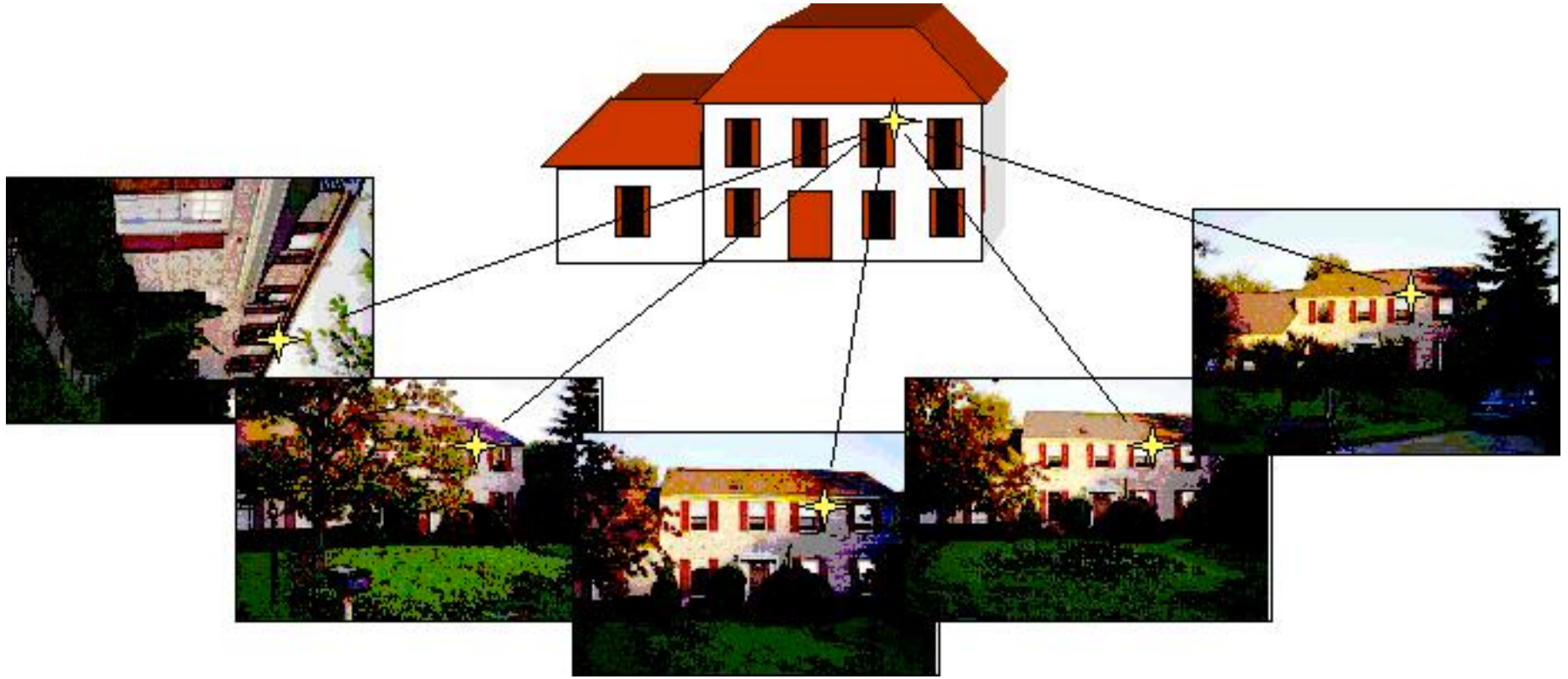
3D Motion and Structure from Multiple Views or Bundle Adjustment

Structure from Motion for Unordered Image Collections

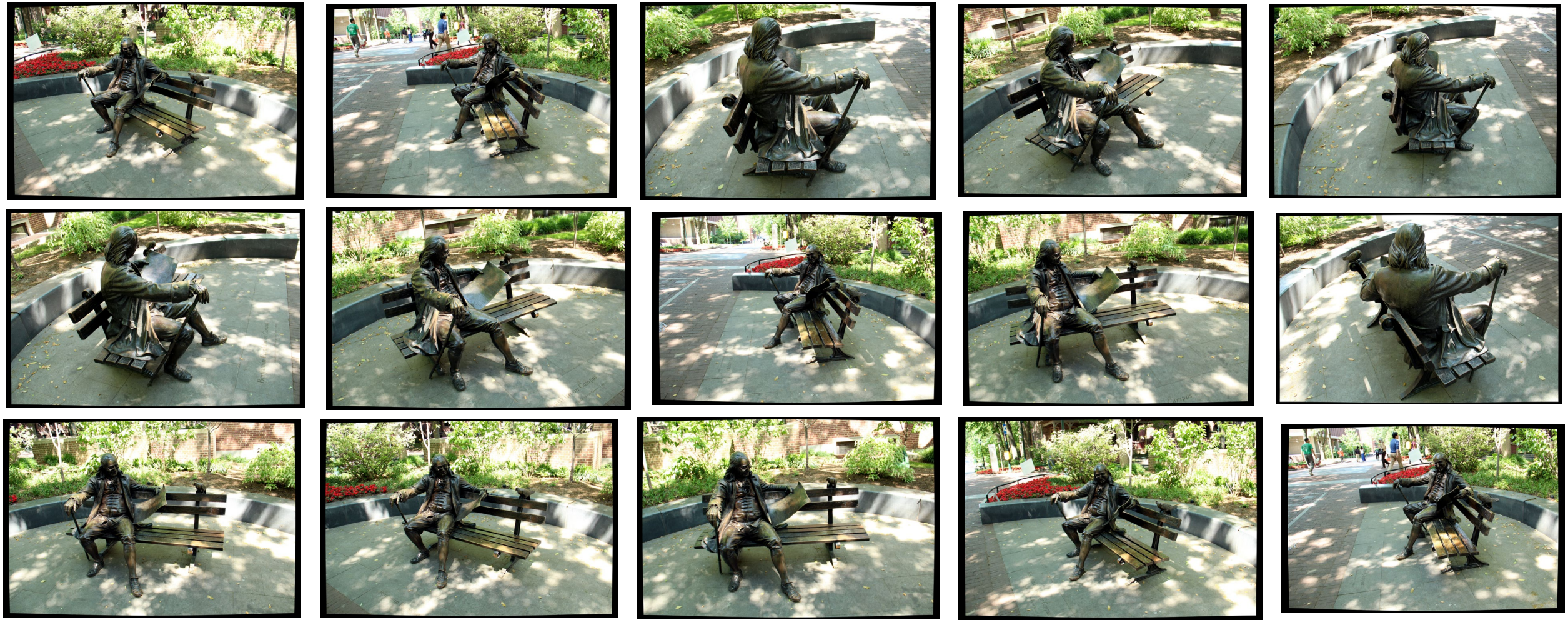
Popular open-source software packages for multi-view SfM:

- Bundler
- COLMAP

Multiple views ⇒ Camera Poses + Scene 3D structure



.. and an example closer to us



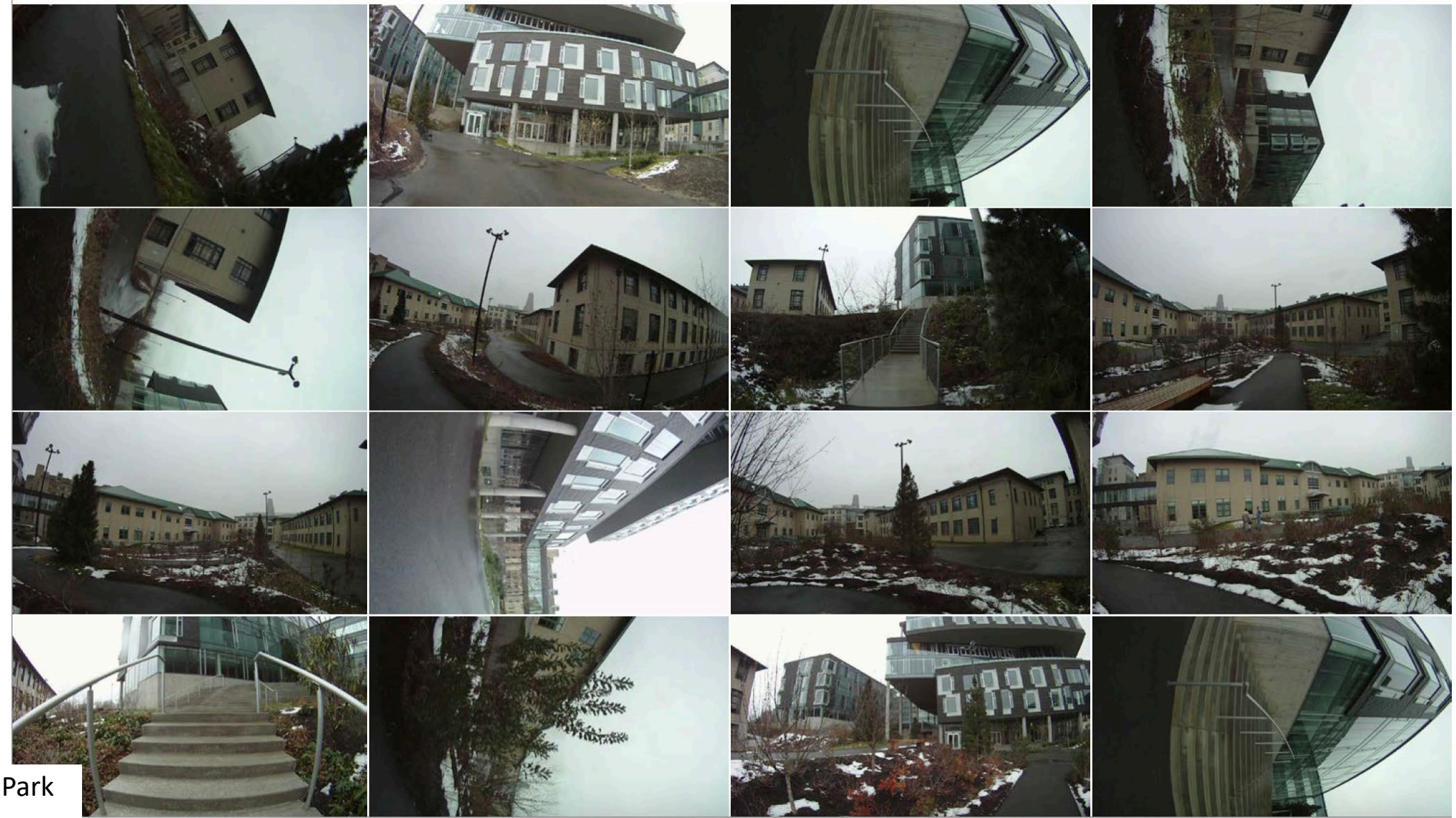
3D reconstruction

Bundler

Urbanscape project 2006

Multiple Images

Let’s walk through a multi-view SfM pipeline till we get to BA

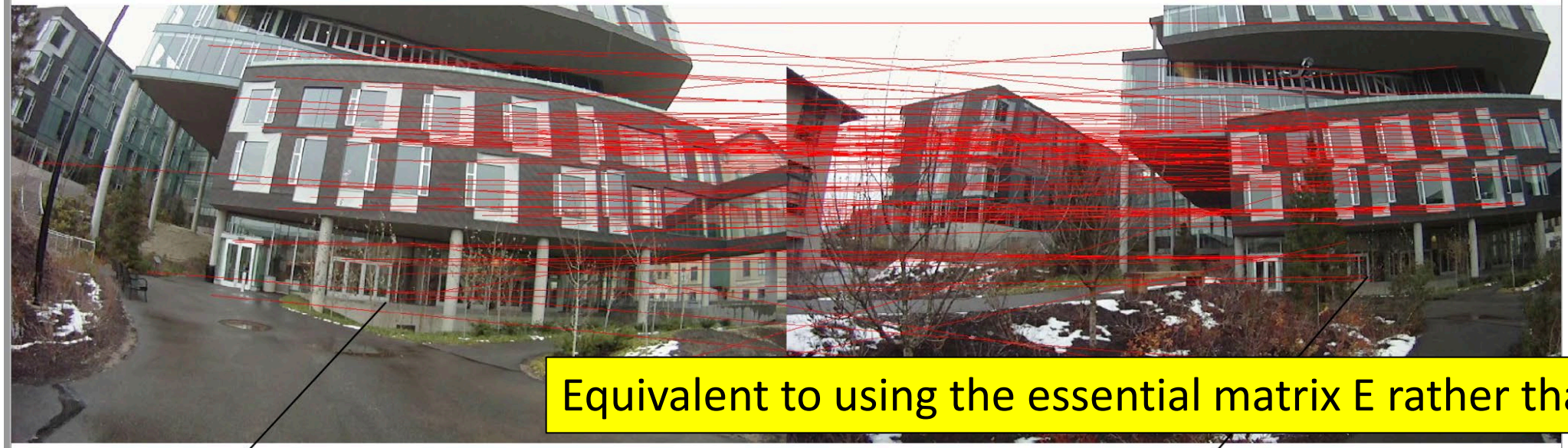


Feature Matching

For a practical intro to SIFT correspondences, see: https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

Feature Matching

Outlier Rejection via RANSAC



Equivalent to using the essential matrix E rather than F

$\mathbf{x}_1$ $\mathbf{x}_2$

$\mathbf{x}_2^T \mathbf{F} \mathbf{x}_1 = 0$ (1)

F: Fundamental matrix

1. Randomly choose 8 correspondences.
2. Build the fundamental matrix for 8 correspondences.
3. Evaluate Equation (1) for all correspondences.
4. Include correspondences that satisfies Equation (1) in the inlier set.
5. Iterate 1-4 and find the matches that produces the maximum inlier set.

Relative Transform

Fundamental Matrix

Equivalent to finding R, T, as we discussed in class.

$\mathbf{x}_1$ $\mathbf{x}_2$

$\mathbf{x}_2^T \mathbf{F} \mathbf{x}_1 = 0$

F: Fundamental matrix

$\mathbf{F} = \mathbf{F}(\mathbf{R}, \mathbf{C})$

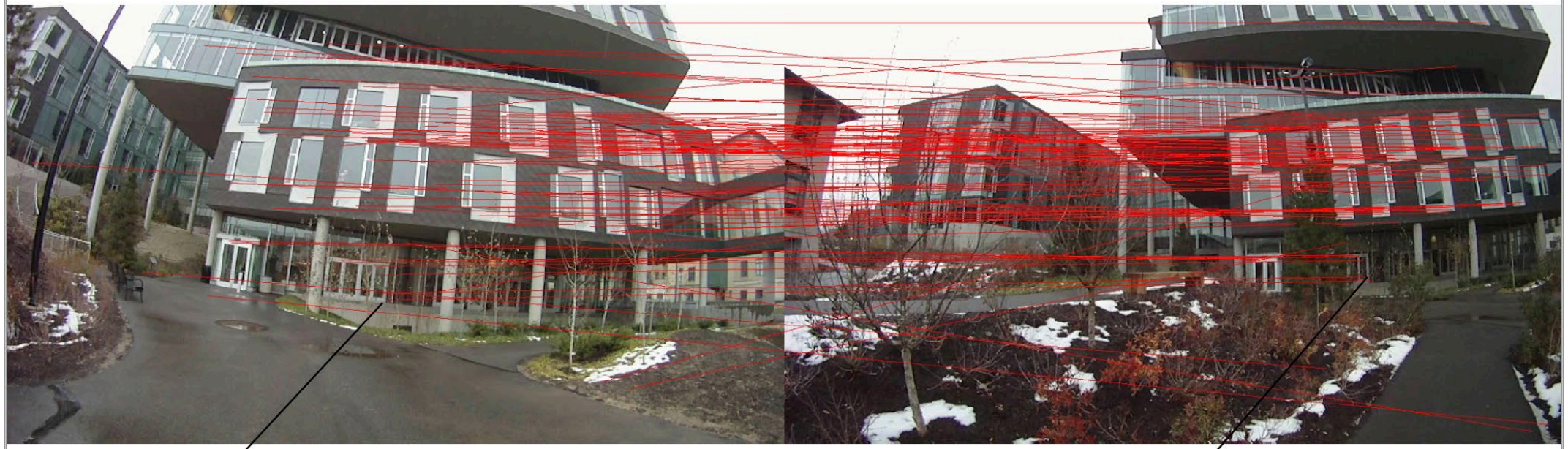
Relative camera transform

R, **C**

C: Camera center position

R: Camera orientation

Point Triangulation



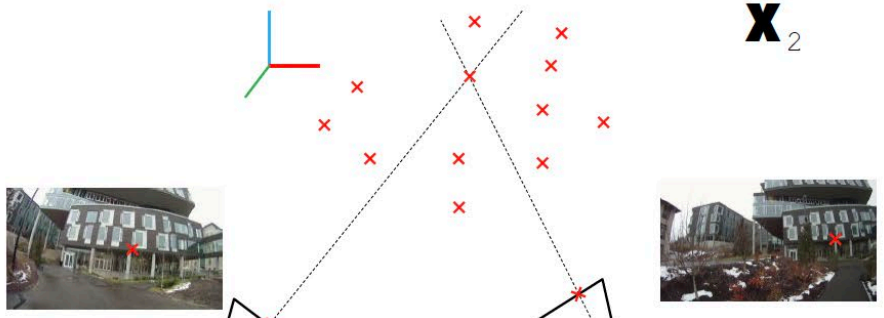
$\mathbf{x}_1$ $\mathbf{x}_2$

$\mathbf{x}_2^T \mathbf{F} \mathbf{x}_1 = 0$

F: Fundamental matrix

$\mathbf{F} = \mathbf{F}(\underline{\mathbf{R}, \mathbf{C}})$

Relative camera transform



$\mathbf{R}, \mathbf{C}$

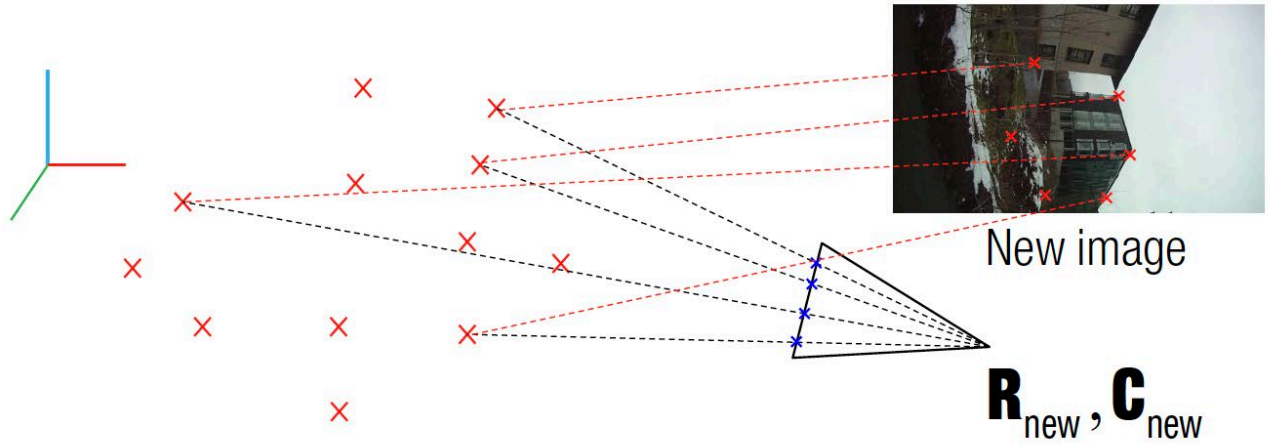
C: Camera center position

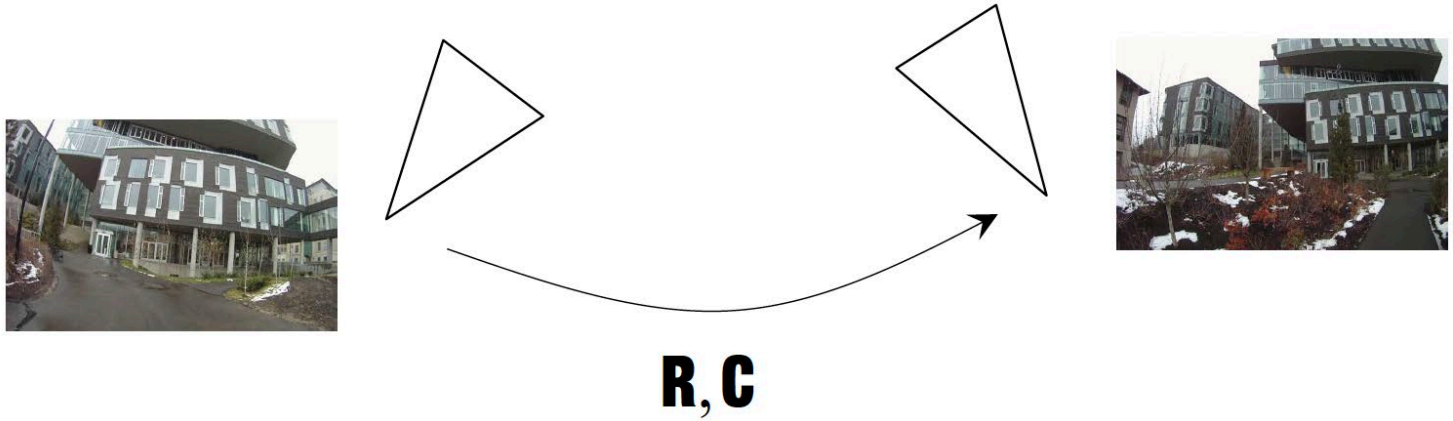
R: Camera orientation

Hyun Soo Park

New Camera Registration

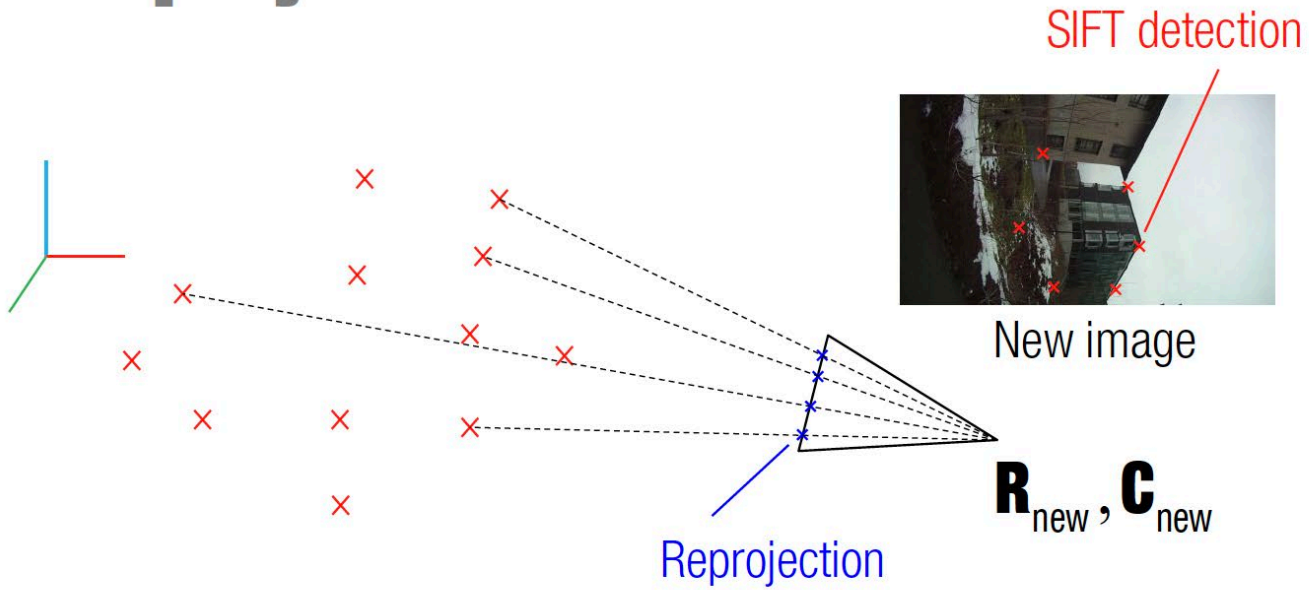
Perspective-n-point





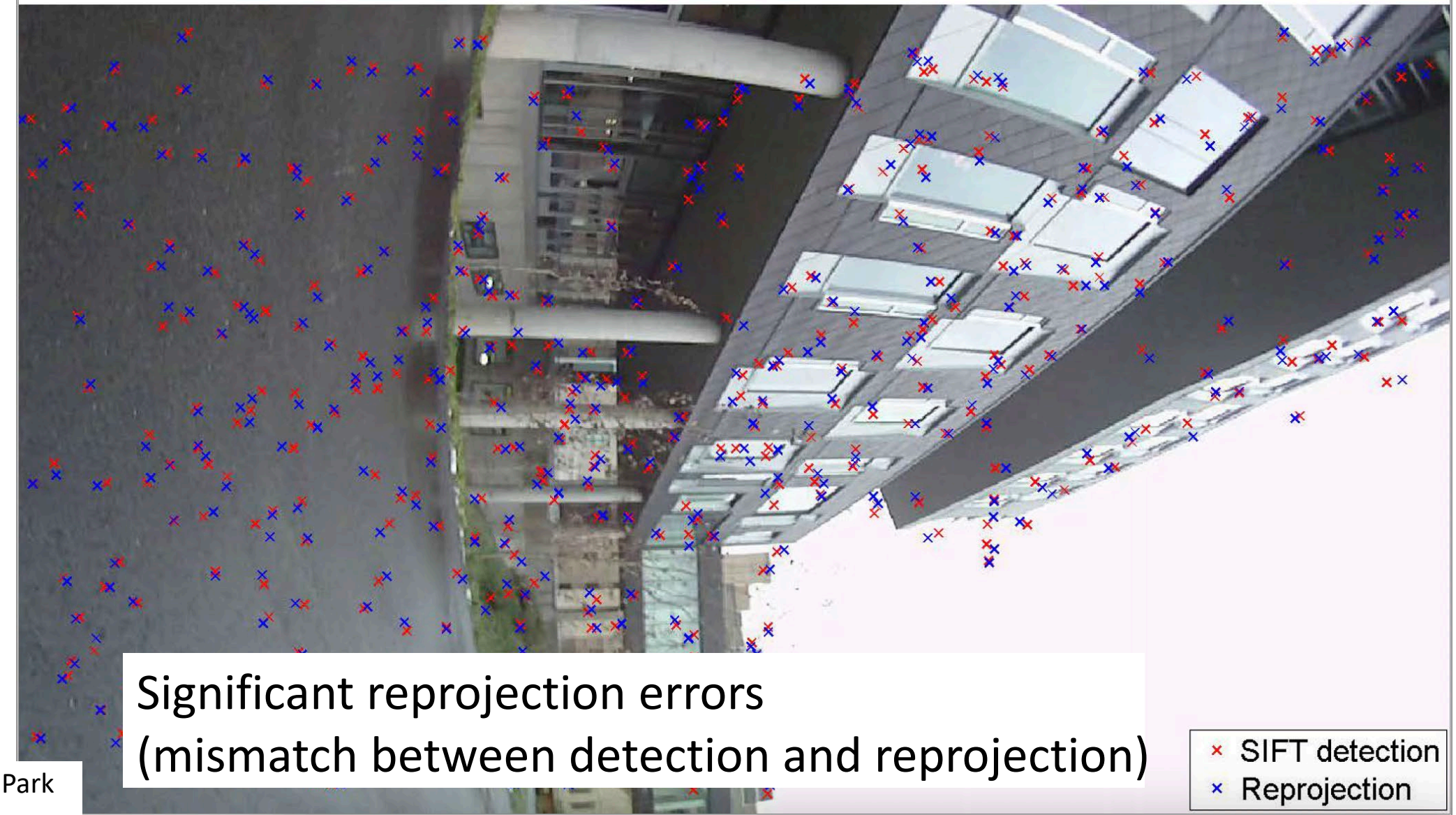
Geometric Analysis

Reprojection Error



Geometric Refinement

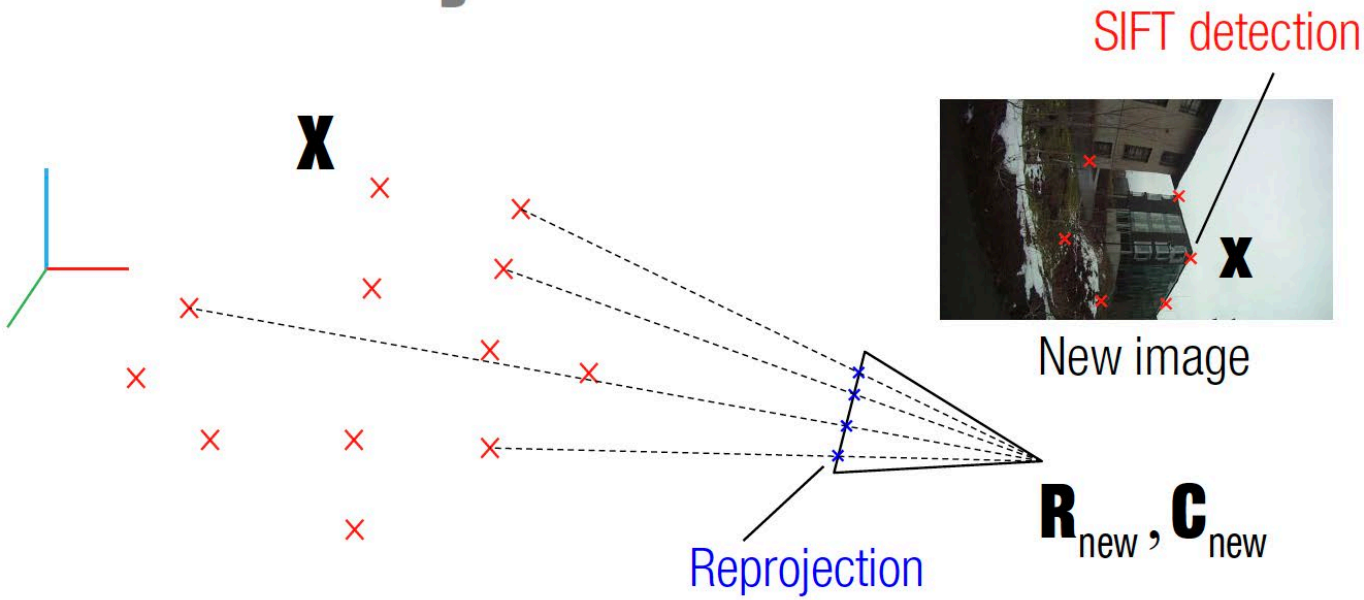
Before Bundle Adjustment

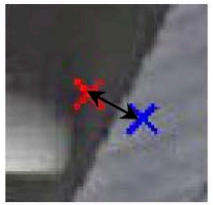


Significant reprojection errors
(mismatch between detection and reprojection)

Geometric Refinement

Bundle Adjustment





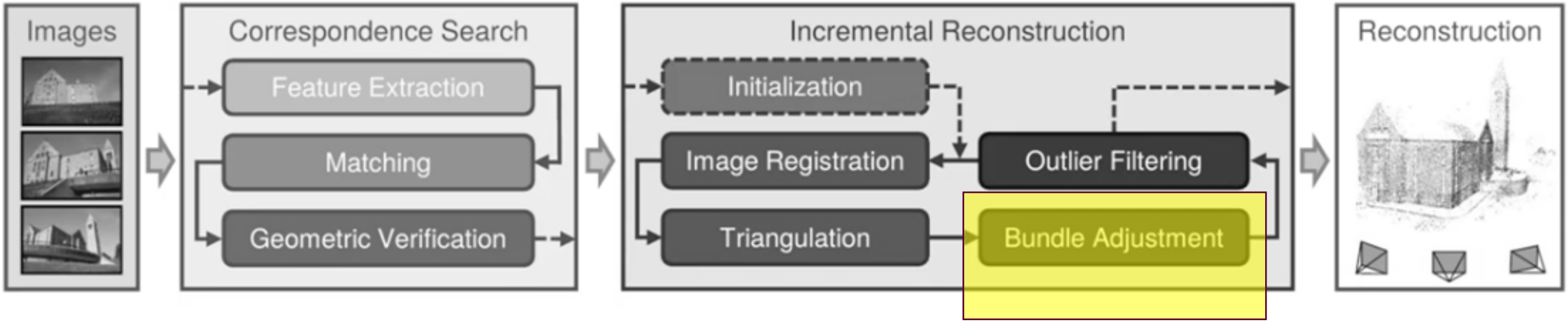
$$\{\mathbf{P}, \mathbf{X}\} = \underset{\mathbf{P},\mathbf{X}}{\operatorname{argmin}} \sum_{j} \sum_{i} \left\| \mathbf{P}_j(\mathbf{X}_i) - \mathbf{x}_{ij} \right\|^2$$

For all points

For all cameras

Reprojection Detection

Bundle Adjustment (BA) in SfM



Schonberger and Frahm, COLMAP, 2016. See also Worksheet on Canvas.

BA is often the final step in SfM, used to *refine* camera poses and 3D points starting from some (pretty good) initialization.

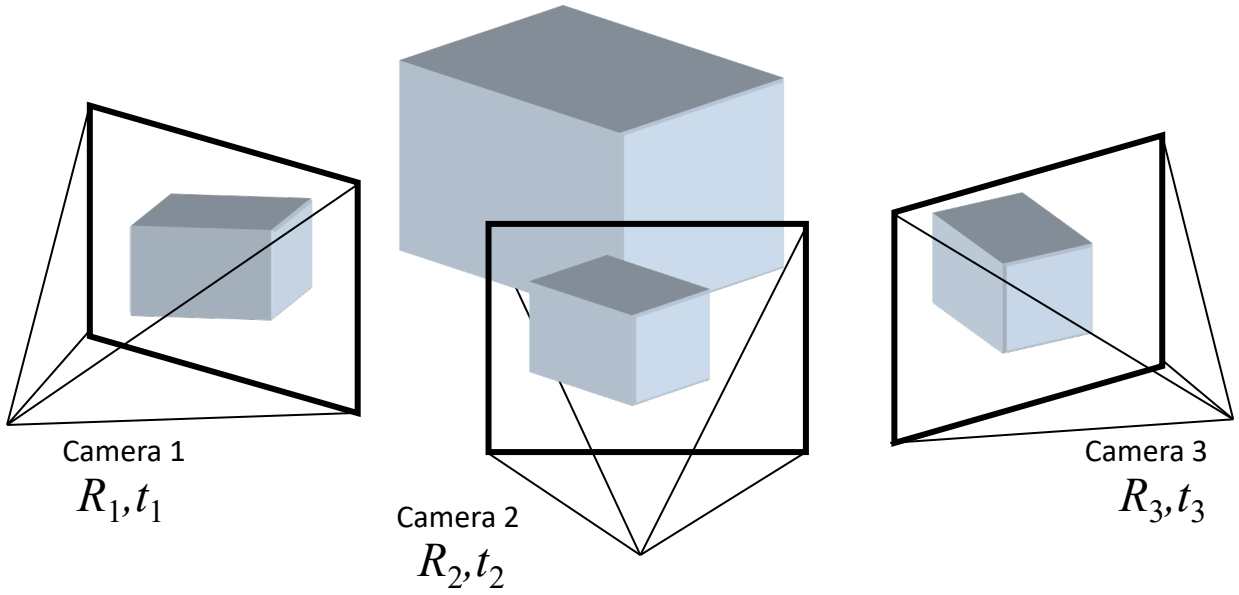
It achieves this refinement through joint non-linear minimization of a **multi-view SfM objective function.**

Bundle Adjustment Objective Function (Recap)

- So what is the error that bundle adjustment tries to minimize?
- **“Reprojection error”**: The sum of errors between 2D observations and the “re-projected” 2D points.

$$\underset{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}}{\operatorname{argmin}} \sum_{n,f} d(\boldsymbol{x}_{nf}, \quad K_f[R_f|T_f]\boldsymbol{X_n})$$

$d(.)$ is usually a simple 2D mean squared error after normalizing to homogeneous coordinate w=1.



“Bundle Adjustment” (BA)

Bundle Adjustment is the process of jointly optimizing for camera poses and object structure from some number of camera images.

- **Q:Isn’t this what we did in 2-view SfM?**
  - A: No, we now want to handle >2 views
- **Q: Isn’t this like what we did within ORB-SLAM by solving 2-view SfM, and registering new views to the 3D map with PnP, and re-triangulating?**
  - A: No. This solves for the first 2 cameras, then for the third, and so on in an *ad hoc* way. Recall that we used BA there for *joint* optimization to make sure all the poses are consistent and minimize some overall error.
    - E.g. Let pose of jth camera w.r.t ith be $(R_{i\to j}, T_{i\to j})$. We might have computed $(R_{1\to 2}, T_{1\to 2})$ and $(R_{1\to 3}, T_{1\to 3})$. But what if: $R_{1\to 3} \neq R_{2\to 3}R_{1\to 2}$ and $T_{1\to 3} \neq T_{1\to 2} + T_{2\to 3}$?

So Why Not Just Do Bundle Adjustment Directly?

- BA = direct non-linear minimization of the reprojection errors, which is a good global objective for SfM.
- But:
  - It requires good initializations, and
  - It becomes computationally expensive as the number of cameras and points grows

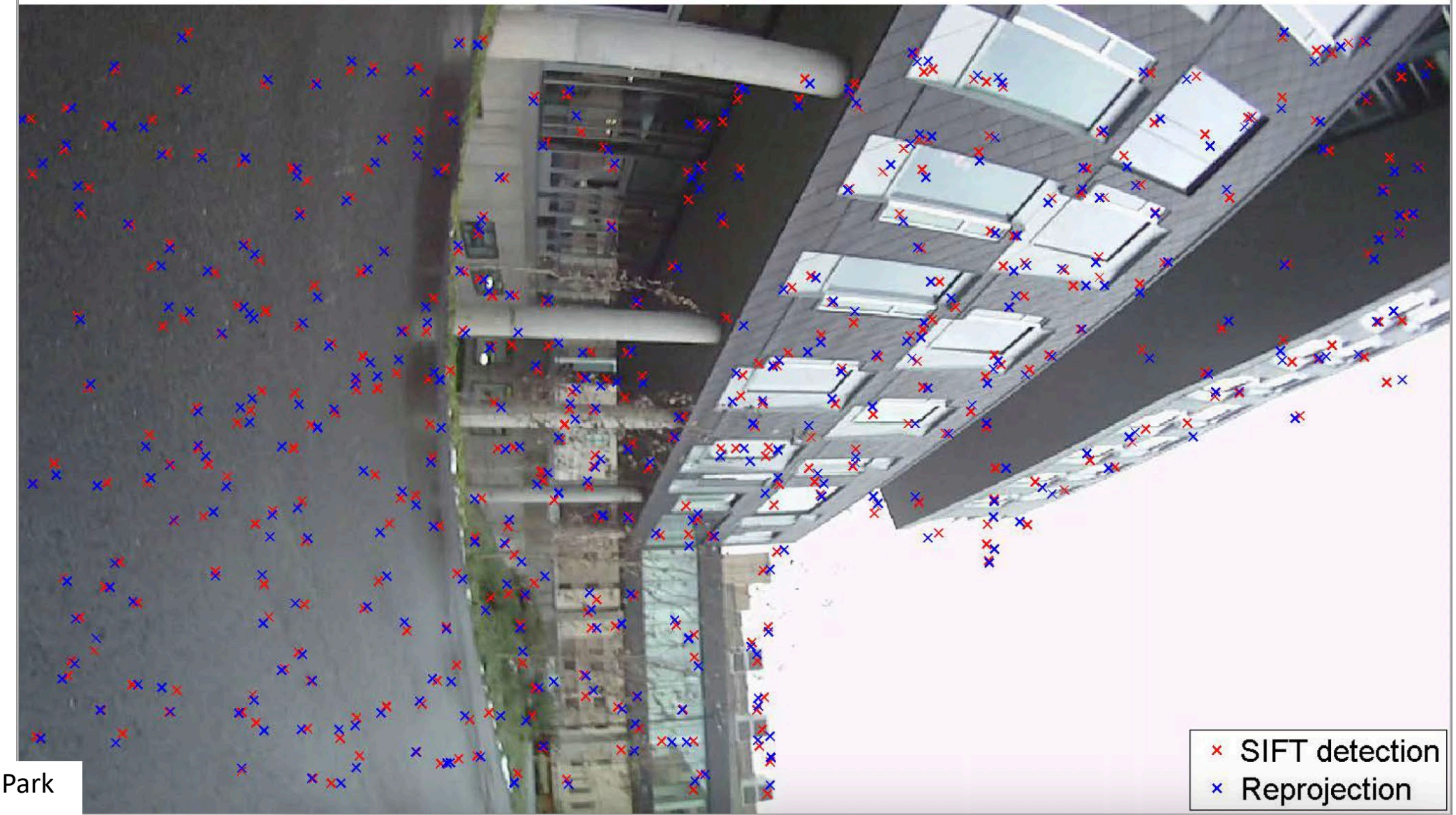
- Often used judiciously as a “refinement” step in any SfM system, after initializing with other techniques. e.g. Incremental 2-view SfM as in ORB-SLAM

How We Normally Use BA

- Start with an initial guess of 3D points and camera poses
- Project estimated 3D points through estimated camera matrices
- Compare locations of projected 3D points with measured 2D points to compute the reprojection error
- Adjust everything jointly to minimize the reprojection error in the images.

Geometric Refinement

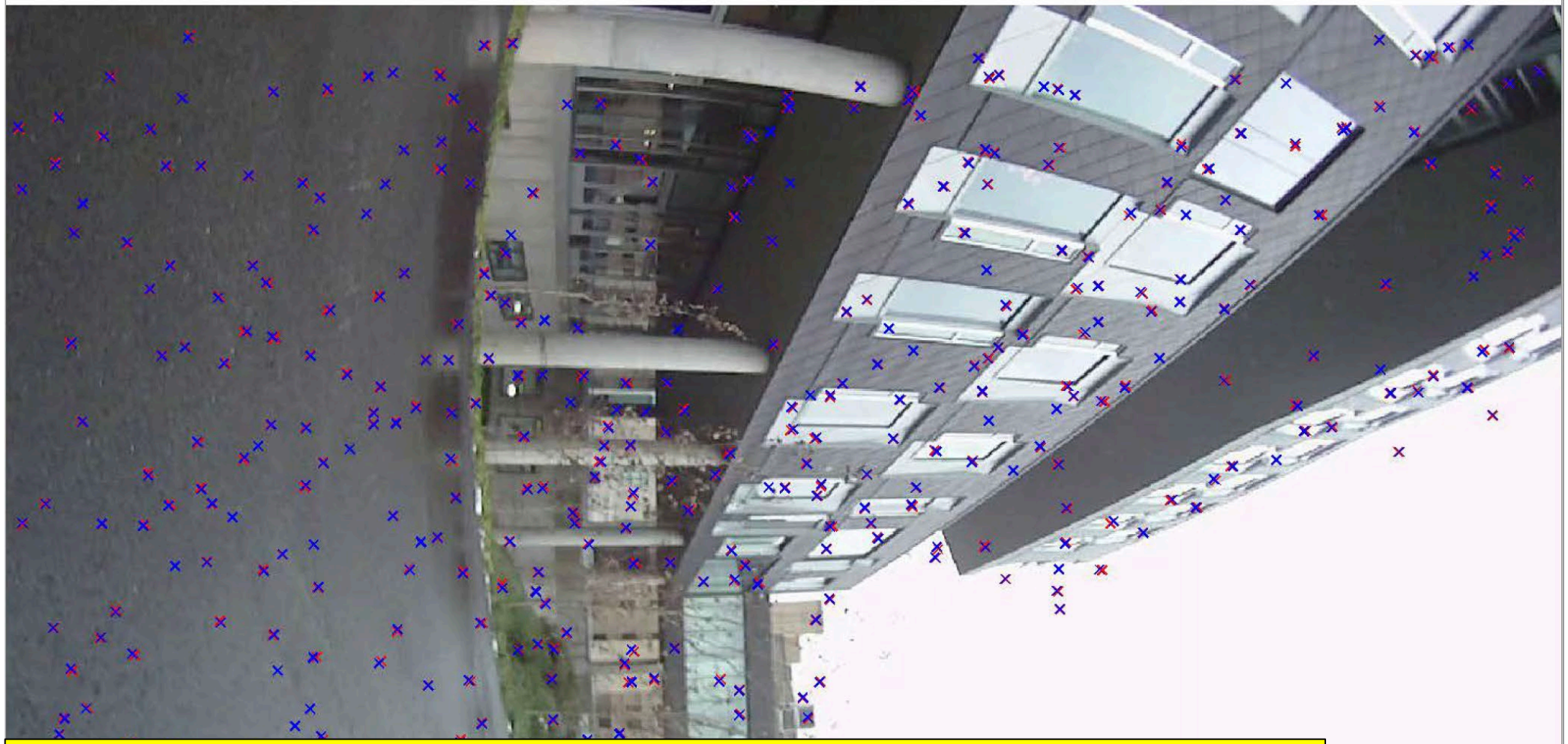
Before Bundle Adjustment



Hyun Soo Park

Geometric Refinement

After Bundle Adjustment



Near-zero reprojection errors
(perfect alignment between detection and reprojection)

3D model from F frames, N points

Reference frame ambiguity hence we fix the first frame to be the world frame:

$$R_1 = I \qquad \text{and} \qquad T_1 = 0$$

Even with fixing the first frame, a global scale factor is still present. If we multiply all 3D points and T with the same scale measurements do not change.

Hence we have $6(F-1)+3N-1$ independent unknowns

+6(F-1) camera poses
+3N 3D point
-1 scale unknown

and $2NF$ equations:

Homogeneous equations
$\boldsymbol{x}^f \sim [R|T]\boldsymbol{X}$

$$x_p^f = \frac{R_{11}^f X_p + R_{12}^f Y_p + R_{13}^f Z_p + T_x}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z}$$

$$y_p^f = \frac{R_{21}^f X_p + R_{22}^f Y_p + R_{23}^f Z_p + T_y}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z}$$

Best case, where every point is visible in every frame.

How many equations do we need?

If equations are independent (not always) then

$$2NF \geq 6F + 3N - 7$$

For two frames, it was already known that $N \geq 5$.

For three frames, $N \geq 4$.

N: point correspondences

F: frames

Example: how many unknowns and equations?

- E.g. F=1k images, and N=100k points in total
  - 6x1k + 3x100k -7 = ~306k unknowns!
- How many equations?
  - If all points are visible everywhere, then:
    - 2NF = 2*1k*100k = 200M equations!
    - If at least 306k of these equations are independent, we should be able to solve for the unknowns.

Note: These are all "back of the envelope" calculations that hide lots of assumptions. E.g. all points observed in all images etc. But still useful.

As a rule of thumb, a point should be visible in at least 4-6 views to enable robustly finding its 3D pose (structure)

Expanding the reprojection error

$$\underset{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}}{\operatorname{argmin}} \quad d(\boldsymbol{x}_{nf}, \quad K_f[R_f|T_f]\boldsymbol{X_n})$$

The usual thing we do for overdetermined systems … least squares

$$\underset{u=\{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}\}}{\operatorname{argmin}} \quad ||\epsilon(u)||_2^2$$

Also ways to introduce uncertainties here, but we skip this.

$$\epsilon^T = \left(\dots \quad x_p^f - \frac{R_{11}^f X_p + R_{12}^f Y_p + R_{13}^f Z_p + T_x}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z} \quad y_p^f - \frac{R_{21}^f X_p + R_{22}^f Y_p + R_{23}^f Z_p + T_y}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z} \quad \dots \right.$$

So, BA is a non-linear least squares problem. How to solve?

Side Notes: Suppl. Reading Materials

A Mini-Course on Optimization

First-order optimization methods: gradient descent

To minimize a differentiable function $f(x)$ over parameters x

1. Start from some initialization x
2. Compute the gradient $g = \frac{df}{dx}(x)$, the direction of local descent
3. Descend along that direction by some step-length α i.e. $\delta x = -\alpha g$
4. Set $x \leftarrow x + \delta x$, then go back to step 2 and repeat

Q: How to set step-length α?

A: Often set to a small constant. Many other options, like Adaptive Gradients.

Gradient Descent Illustration

$\mathcal{L}(\boldsymbol{\theta}_0, \boldsymbol{\theta}_1)$

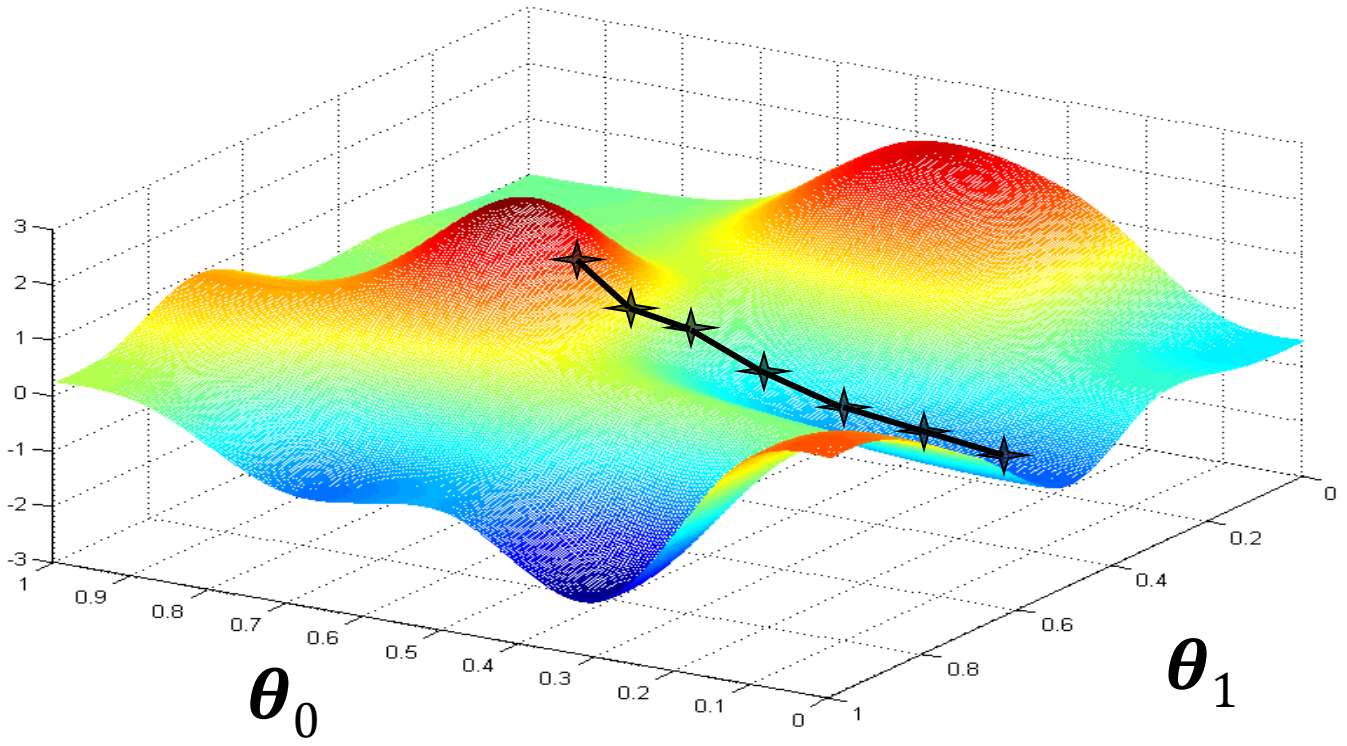
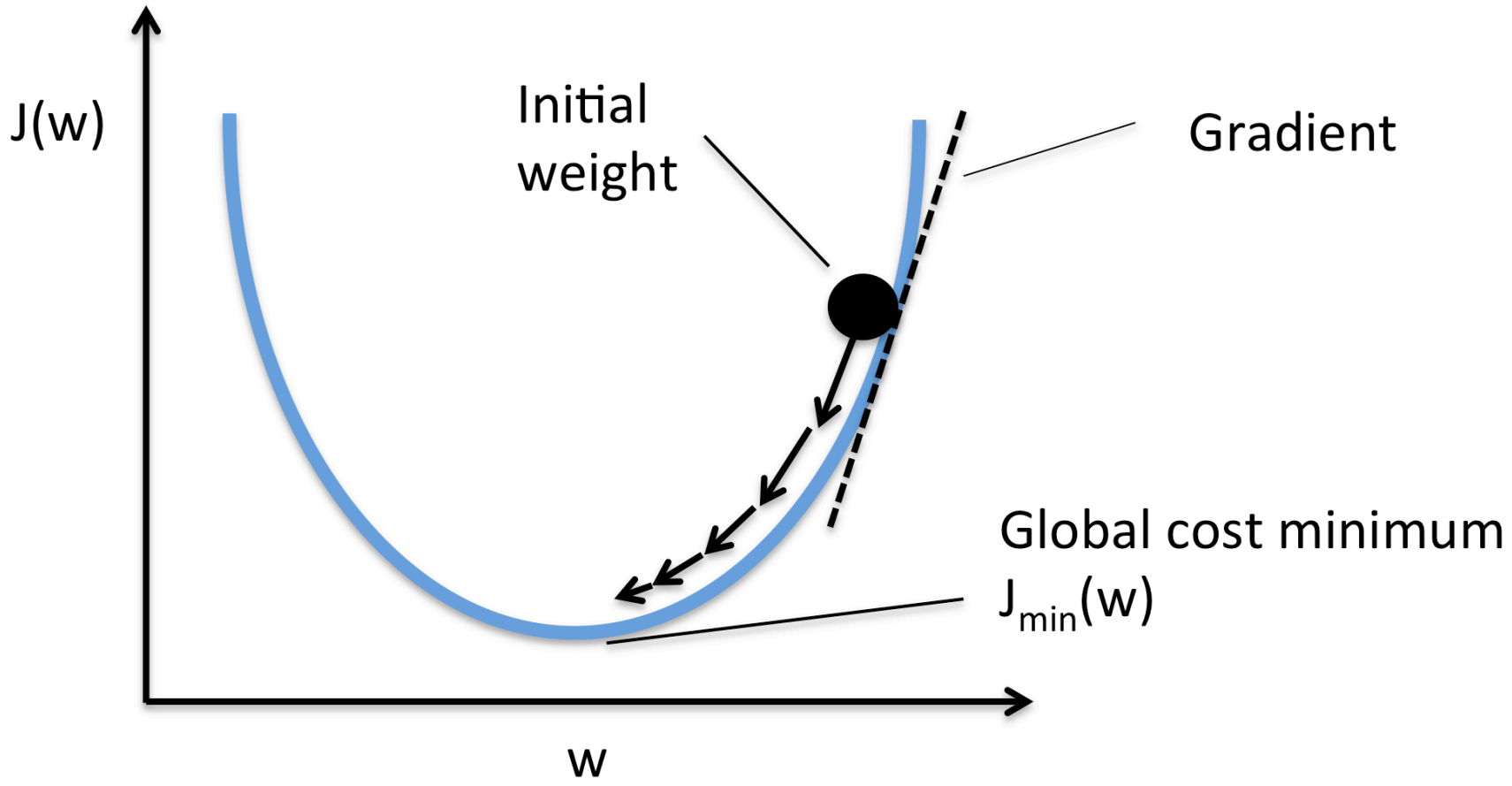


Figure by Andrew Ng

Gradient Descent Illustration



Slow convergence due to small gradients near the minimum.

https://sebastianraschka.com/faq/docs/gradient-optimization.html

Second-order optimization: a bird's eye view of the strategy

To minimize a (twice-) differentiable function $f(x)$ over parameters x

- Start with an initial estimate for x
- Locally approximate $f(x)$ using e.g. a second-order Taylor expansion.

$$f(x+\delta x) \approx f(x) + g^T \delta x + \frac{1}{2}\delta x^T H \delta x,$$

where: $g = \frac{df}{dx}(x)$, and $H = \frac{d^2 f}{dx^2}(x)$

Hessian

- Try to find a displacement $x \to x + \delta x$ that locally minimizes the quadratic local approximation of $f(x)$
- This does not usually give the exact minimum of $f(x)$, but with luck it will improve over the initial estimate, and allow us to iterate to convergence.

Newton/Newton-Raphson's method (second-order)

- Locally approximate $f(x)$ using e.g. a Taylor expansion.

$$f(x+\delta x) \approx f(x) + g^T \delta x + \frac{1}{2}\delta x^T H \delta x,$$

where: $g = \frac{df}{dx}(x)$, and $H = \frac{d^2 f}{dx^2}(x)$. Assume the "Hessian" H is positive semi-definite for now, so that you can find the minimum.

$$\frac{df}{dx}(x+\delta x) \approx H\delta x + g = 0 \Rightarrow$$

$$\delta x = -H^{-1} g$$

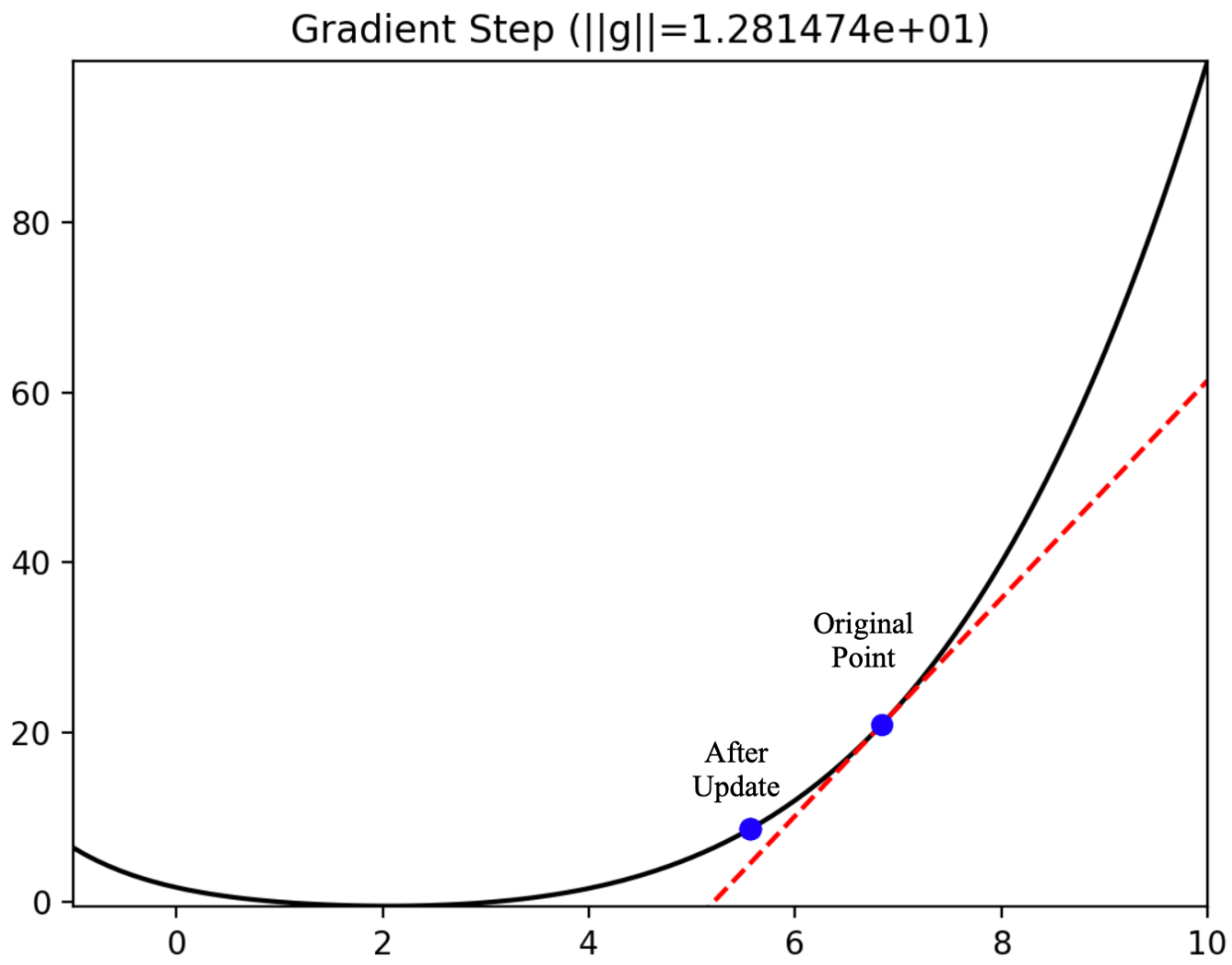
Goes directly to the minimum of the local quadratic approximation of f

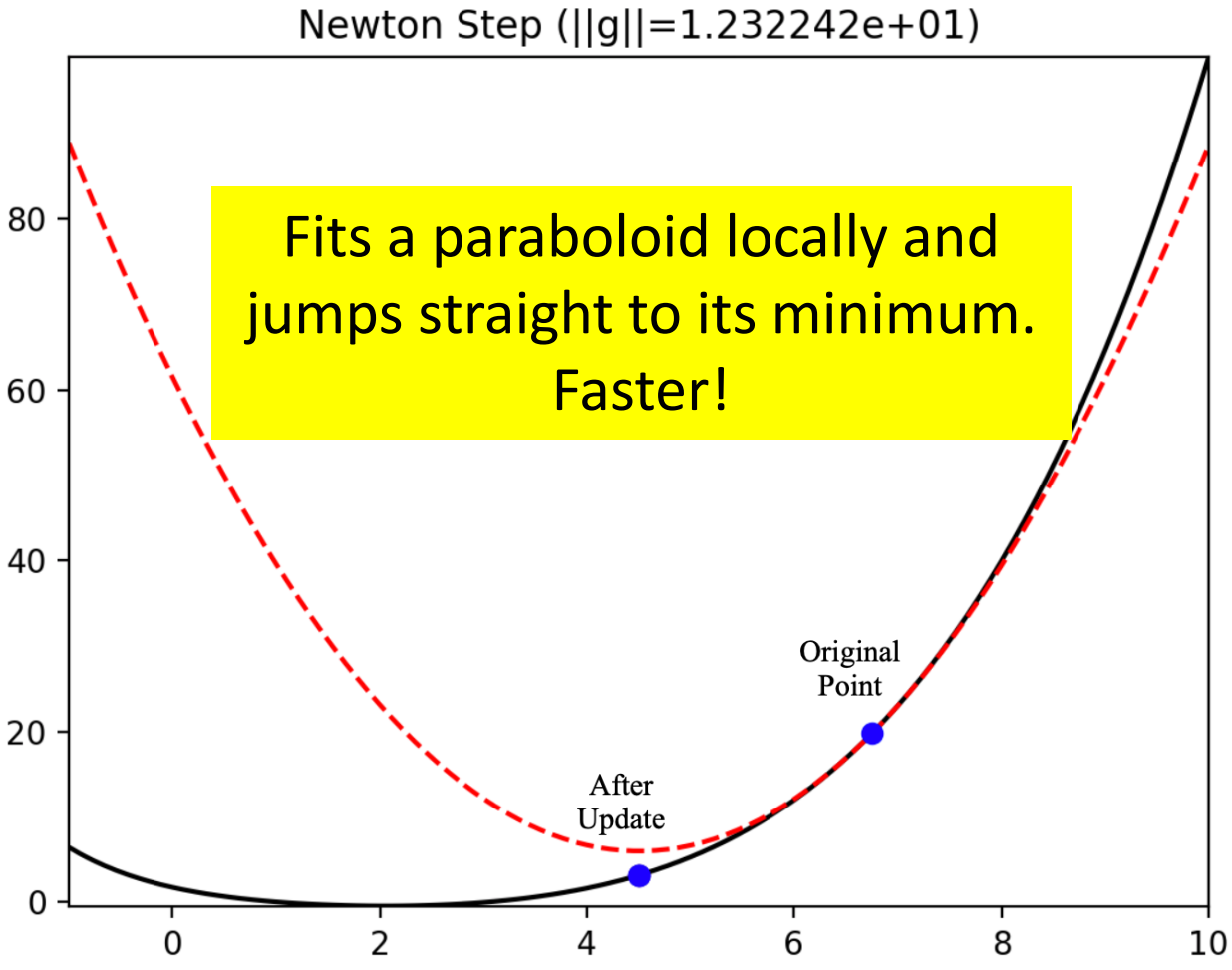
Iterating over this update = "Newton's method"

The most basic "second-order" non-linear optimization method

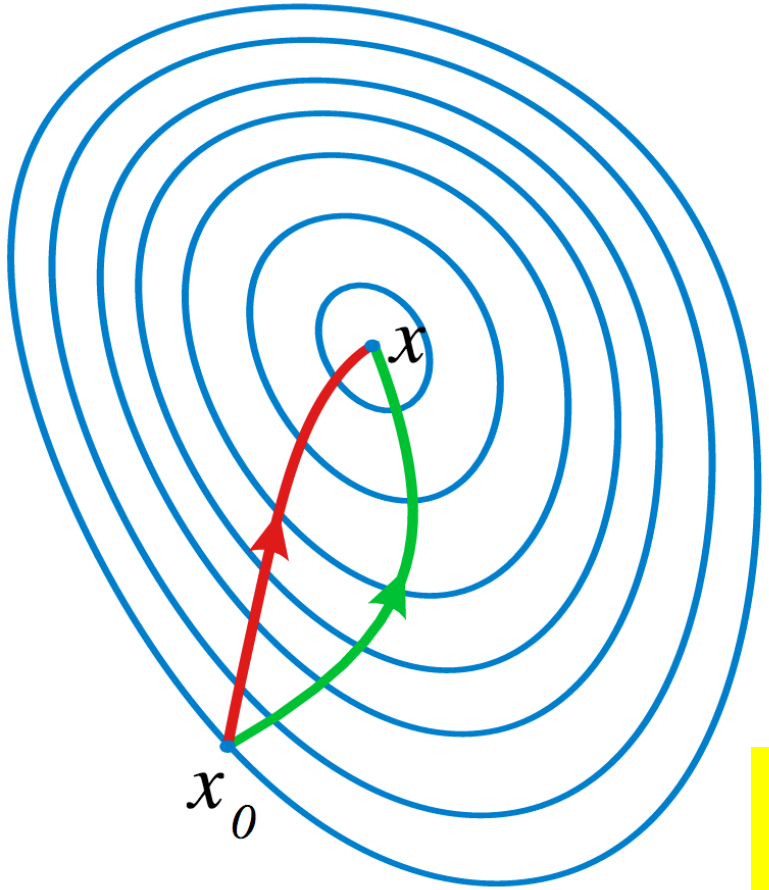
(Asymptotic quadratic convergence = error approx. squared at each iteration)

Gradient Descent Vs. Newton's Method





Gradient Descent Vs. Newton's Method



Red – Newton's Method
Green - gradient descent iteration.

When it works, Newton's method often finds faster, more direct routes!

Newton's method uses curvature information (i.e. the second derivative) to take a more direct route.

Problem: Computing the Hessian $H = \frac{d^2 f}{dx^2}$ is expensive!

Wikipedia

Gauss-Newton Method for Least Squares part 1/2

An approximate version of Newton's method for least squares:

$$\underset{u=\left\{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}\right\}}{\operatorname{argmin}} \quad f(u) = ||\epsilon(u)||_2^2$$

Gradient $g = \nabla_u f = 2\sum_i \epsilon_i(u)\nabla_u \epsilon_i(u) = 2\,J(u)^T \boldsymbol{\epsilon}(u),$ A

Where $J_{ij} = \frac{\partial \epsilon_i}{\partial u_j}$ Jacobian

Hessian reads (computing the gradient of g):

$$H(u) = 2\sum_i \nabla_u \epsilon_i(u)\nabla_u \epsilon_i(u) + \epsilon_i(u)\frac{\partial^2 \epsilon_i}{\partial^2 u^2}$$

$$= 2J(u)^T J(u) + 2\sum_i \epsilon_i(u)\frac{\partial^2 \epsilon_i}{\partial u^2}$$

Ignoring the hard-to-compute quadratic terms, $H(u) \approx 2\,J(u)^T J(u)$ B

Saves computation, and is approx. true when error ϵ_i is small, or the function is ~ linear.

Gauss-Newton Method for Least Squares part 2/2

We saw earlier, Newton's update for general non-linear optimization:

$$\delta u = -H^{-1} g$$

For least squares problems $\underset{u}{\operatorname{argmin}} f(u) = ||\epsilon(u)||_2^2$, we have seen:

- By A on the last slide, $g = 2\, J(u)^T \epsilon(u)$
- By B on the last slide, $H(u) \approx 2 J(u)^T J(u)$

Directly leads to Gauss-Newton update, often called "Normal Equation":

$$\delta u = -\left(J(u)^T J(u)\right)^{-1} J(u)^T \epsilon(u)$$

Q: Have you seen an expression like this before when solving linear equations?

Hint: If you were minimizing $\left|\left|Ax - b\right|\right|_2^2$, what would the solution be?

Gauss-Newton for the BA Least Squares Problem?

Recall that the BA problem looked like:

$$\underset{u=\left\{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}\right\}}{\operatorname{argmin}} \quad f(u) = ||\epsilon(u)||_2^2$$

So the new update at each iteration would look like:

$$\delta u = -(J^T J)^{-1} J^T \epsilon$$

(J and ϵ are both functions of the parameters u)

Computational Nightmares!

$$\underset{u=\{\{X_n\}_{n=1}^{N},\{R_f,T_f\}_{f=1}^{F}\}}{\operatorname{argmin}} \|\epsilon(u)\|_2^2$$

$$\epsilon^T = \left(\dots \quad x_p^f - \frac{R_{11}^f X_p + R_{12}^f Y_p + R_{13}^f Z_p + T_x}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z} \quad y_p^f - \frac{R_{21}^f X_p +}{R_{31}^f X_p +} \right.$$

We've decided we want to do something like:

$$\delta x = -(J^T J)^{-1} J^T \epsilon$$

What is the size of $J = \left[\frac{\partial \epsilon_i}{\partial u_j}\right]_{ij}$?

- E.g. F=1k images, and N=100k points in total
  - 6x1k + 3x100k -7 = ~306k unknowns!
- How many equations?
  - If all points are visible everywhere, then:
    - 2NF = 2*1k*100k = 200M equations!

Recall:

- The dimension of the reprojection error vector $\boldsymbol{\epsilon}$ is 2NF (F frames, N points in each frame, 2 dimensions per point in the reprojection error vector)
- The number of unknown parameters $\boldsymbol{u}$ is $\mathrm{M} = 6F + 3N - 7$

The size of J is $2NF \times M$. (e.g. 200e6 x 306e3)

$J^T J$ is $M \times M$ e.g. (306e3 x 306e3).

For general matrices, inverse scales as $O(M^3) \approx 2.8e16$.

This is all bad news!

BA is all about linear algebra implementation tricks!

- The modern bundle adjustment literature is all about how to deal with this massive computational complexity cleverly.

- Some useful properties to simplify huge linear equation systems:
  - Try to avoid inverses of *large general matrices* to the extent possible, and instead reduce to "simpler" matrix inverses.
    - Block diagonal matrices can be inverted block-by-block.
  - Try to set up equations $A\boldsymbol{x} = \boldsymbol{b}$ with triangular matrices A, much easier to solve. ("forward/backward substitutions")
  - Avoid matrix multiplications of large general matrices:
    - Sparse matrices, including non-diagonal ones, are much easier to multiply