2020 embedded VISION summit

Introduction To Simultaneous Localization and Mapping (SLAM)

Gareth Cross
Skydio
September, 2020

Skydio

What is SLAM?

Simultaneous
Localization and *Mapping*

Recover state of a vehicle or sensor platform, usually over multiple time-steps.

Recover location of landmarks in some common reference frame.

Simultaneous: We must do these tasks at the same time, as both quantities are initially unknown.

An age-old practice

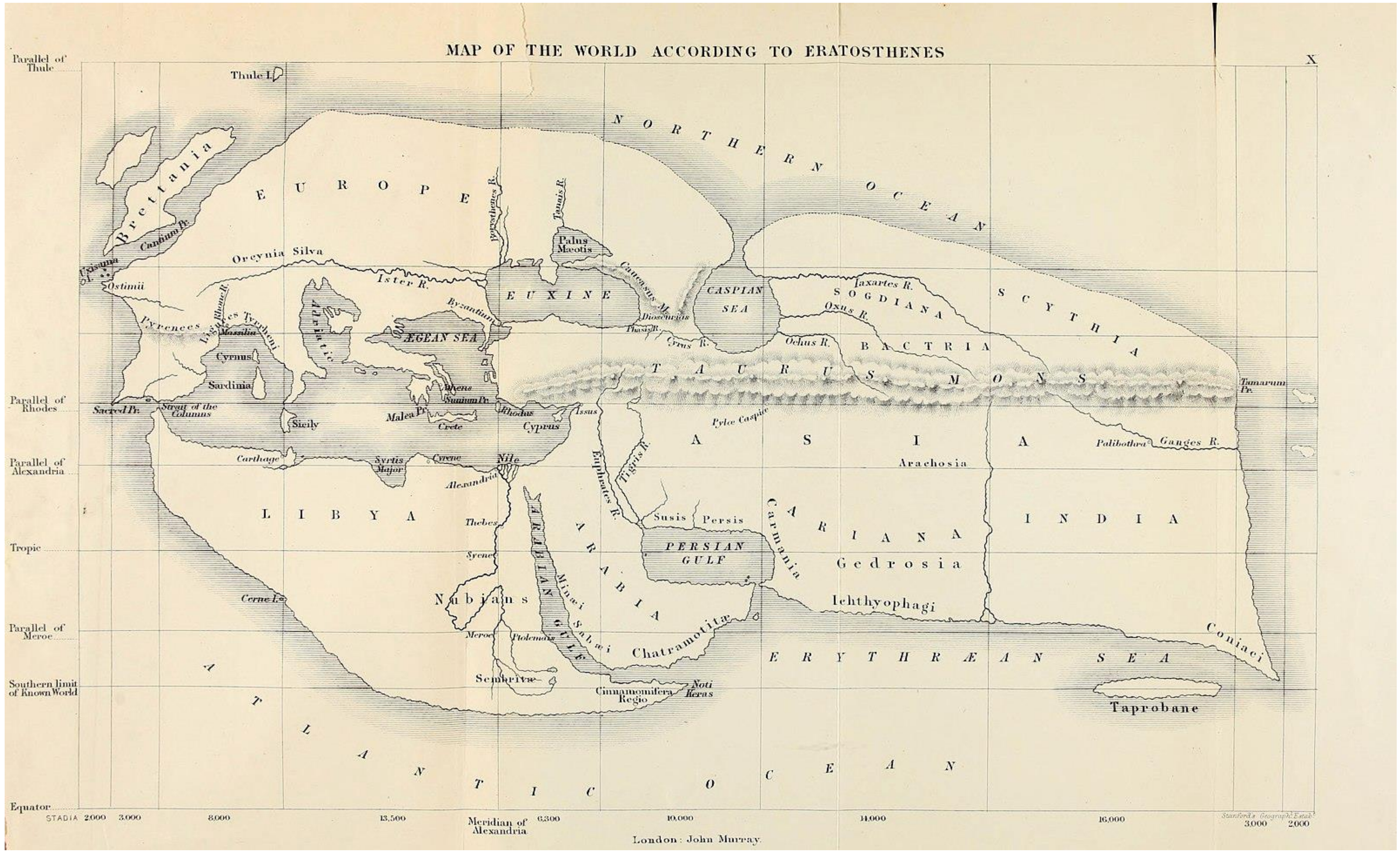
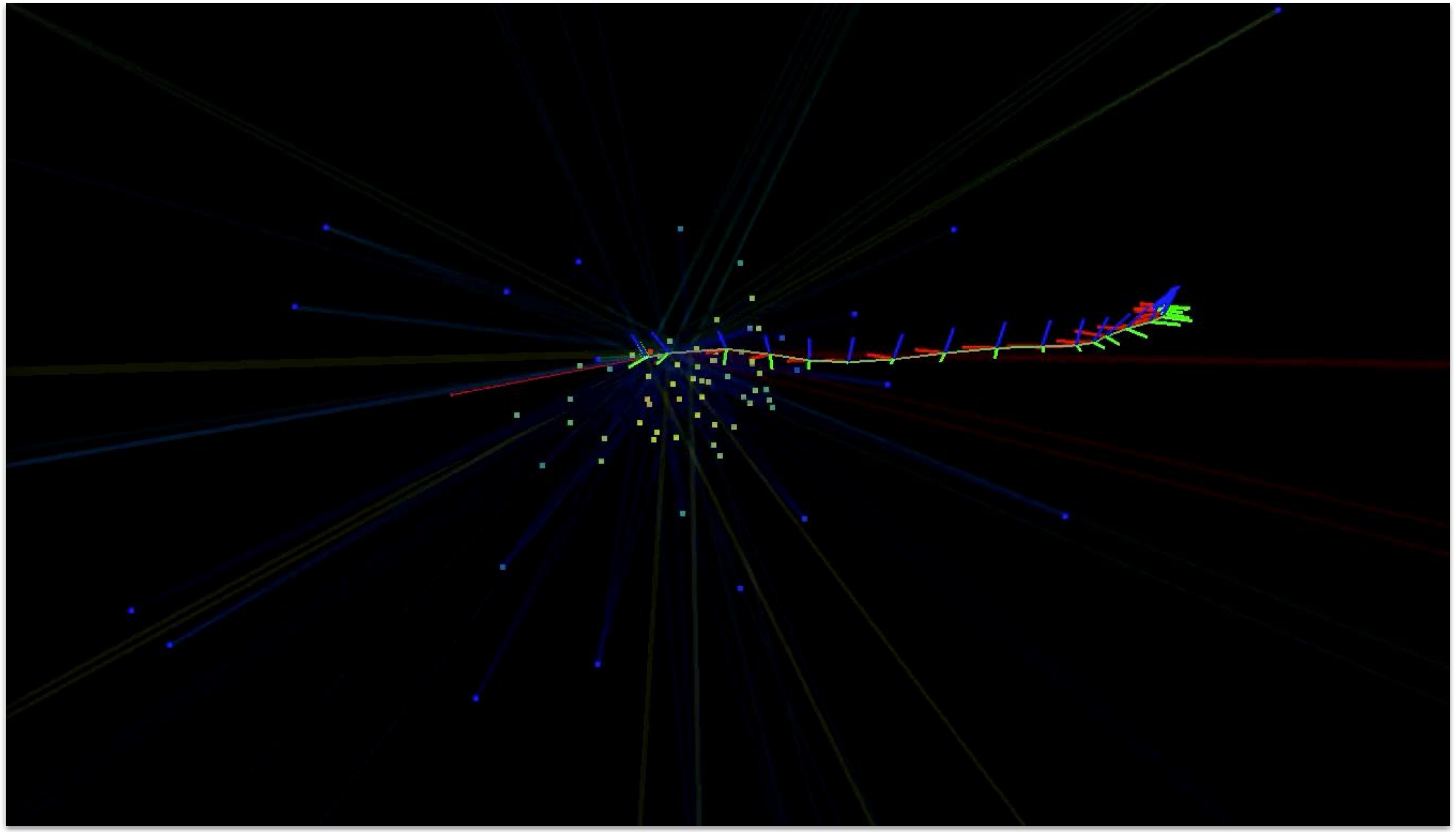
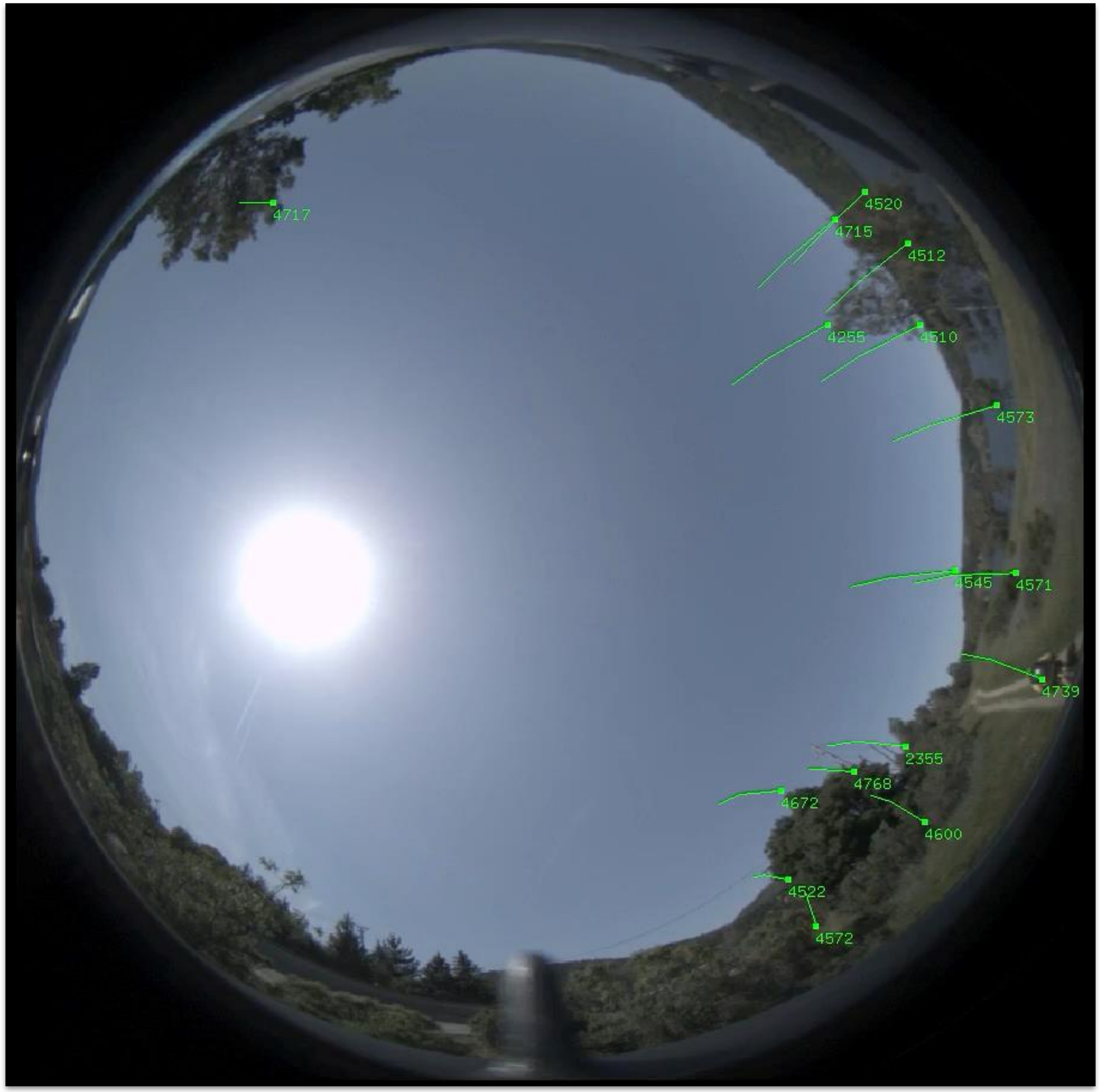


Image Source: A History of Ancient Geography among the Greeks and Romans from the Earliest Ages till the Fall of the Roman Empire via Wikipedia

Image Source: COLMAP / Schönberger, Johannes Lutz and Frahm, Jan-Michael, “Structure From Motion Revisited”, CVPR 2016

SLAM at Skydio

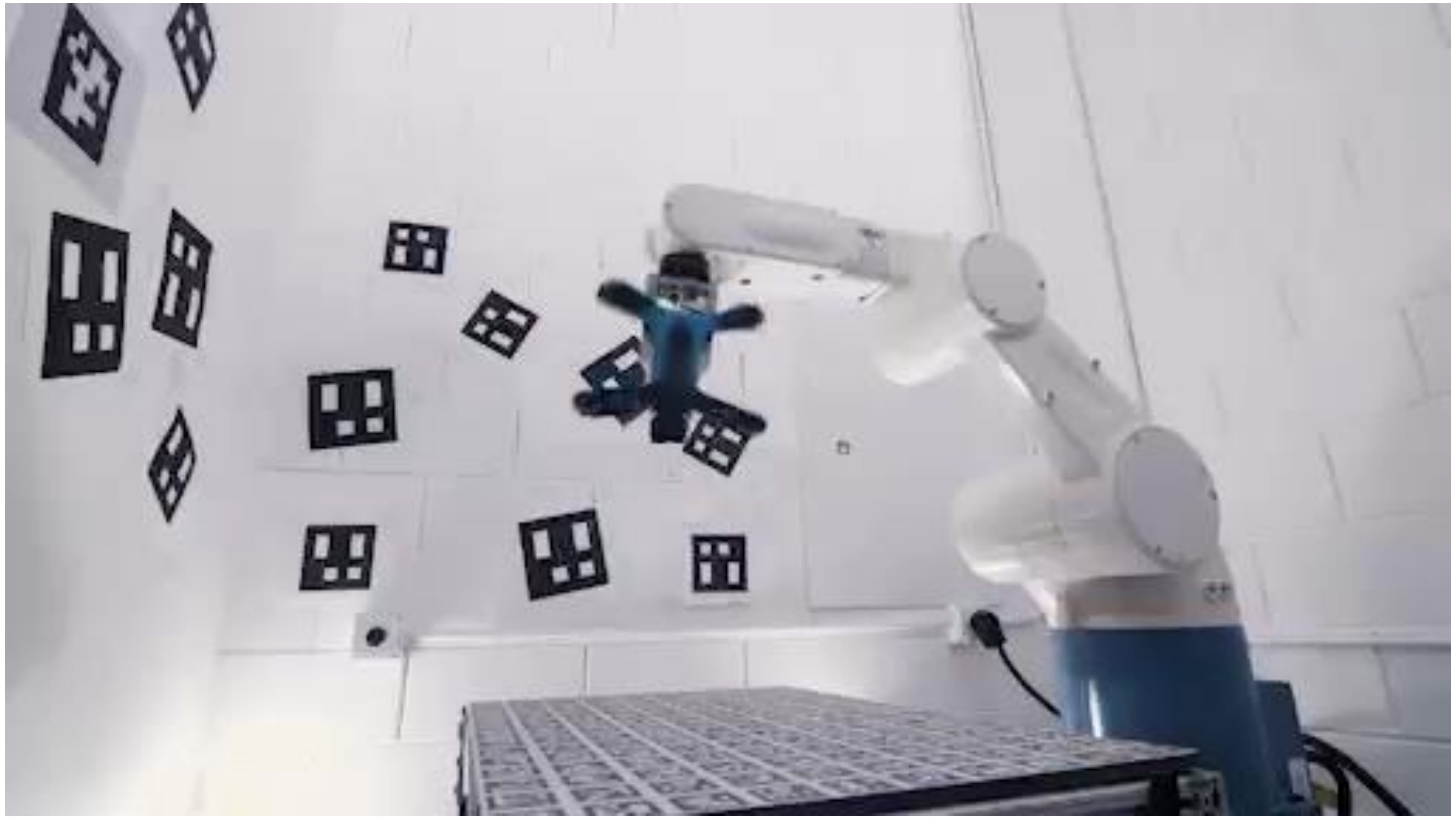




Visual Inertial Odometry (VIO) on the Skydio drone, an embedded system.

SLAM vs. Localization

Image Source: E. Kaplan, C. Hergarty, *Understanding GPS Principles and Applications*, 2005



Video Source: Skydio

For every SLAM problem, we have two key ingredients:

1) One or more *sensors*:

Cameras

Source: MatrixVision

Inertial Measurement Unit

Source: Lord MicroStrain

LiDAR/Range-finders

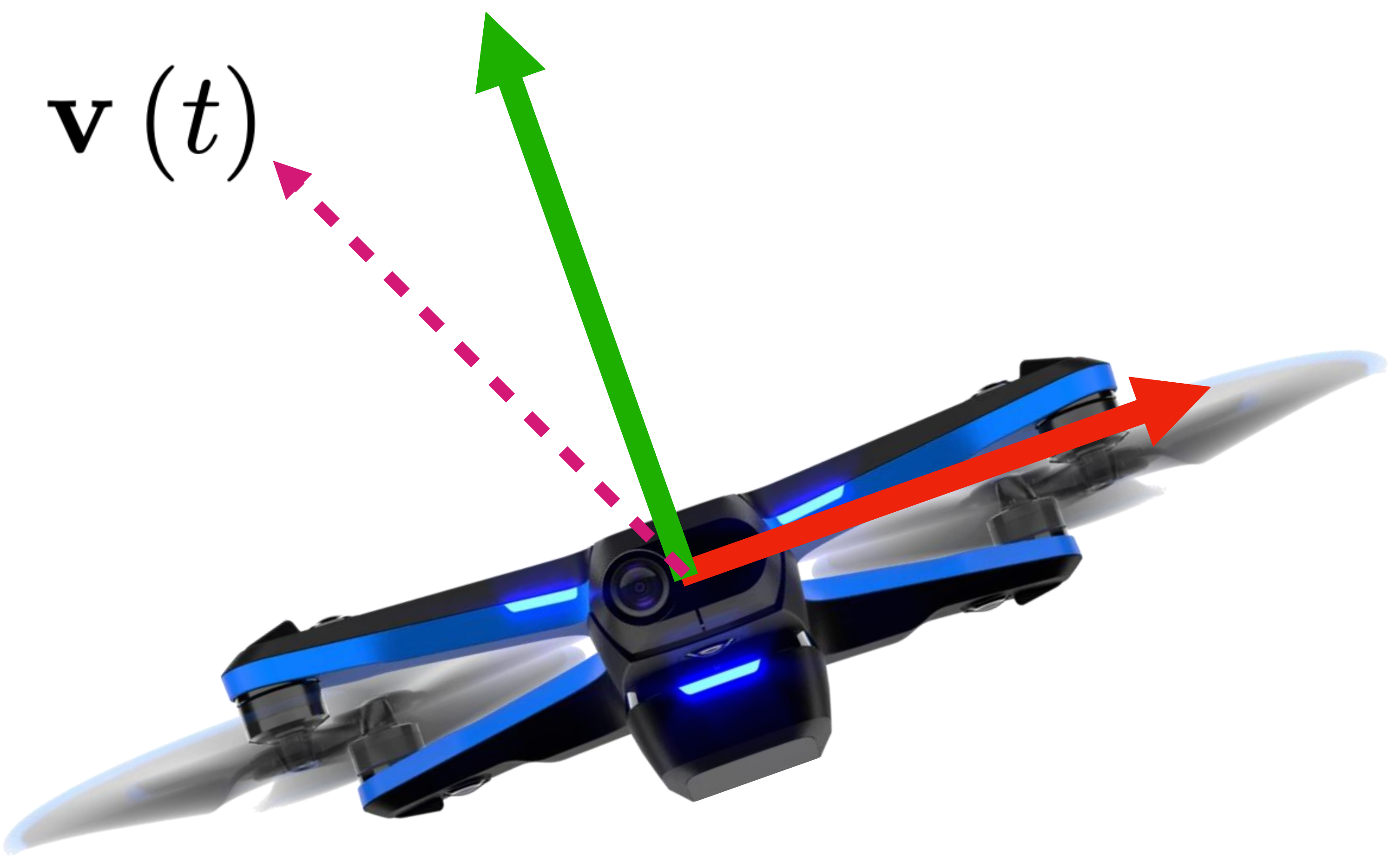
Source: Velodyne

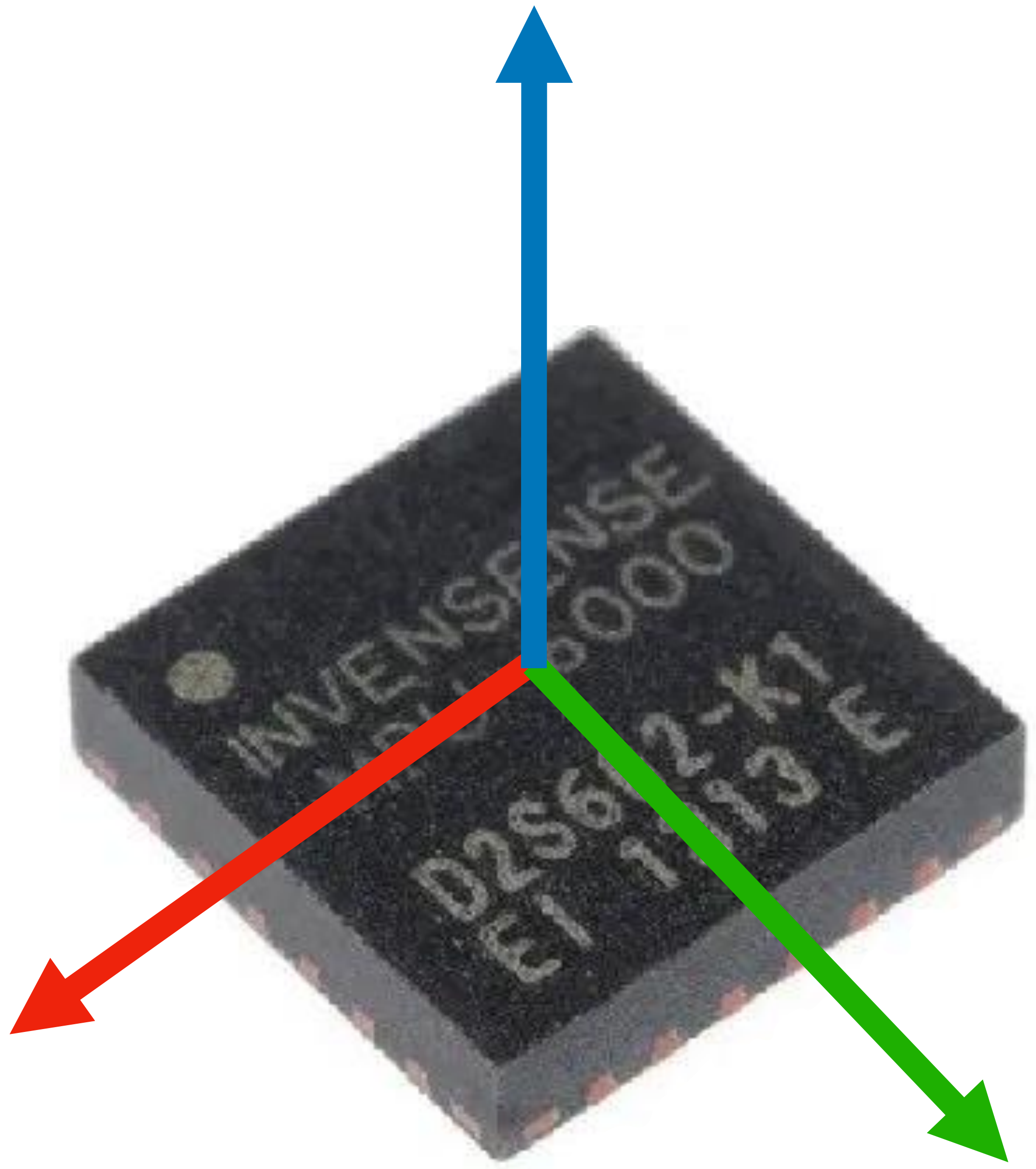
RGB-D/Structured Light

Source: Occipital

Formulating a SLAM Problem

2) A set of *states* we wish to recover.





Ego-motion: Rotation, position, velocity

World Structure (Map)

Calibration Parameters

Image Source: [DroneTest]

Skydio

Sensor Selection

- Choice of sensor will drive many downstream design considerations.
- Consider the *sensor measurement model*:

Ego-motion　　　　Calibration parameters

$$\underbrace{\mathbf{z}}_{\text{Sensor output}} = h\left(\overbrace{\mathbf{x}}^{\text{Ego-motion}}, \underbrace{\mathbf{m}}_{\text{Map}}, \overbrace{\kappa}^{\text{Calibration parameters}}\right) + \underbrace{\epsilon_z}_{\text{Noise}}$$

Sensor output　　　　Map　　　　Noise

High Level Goal

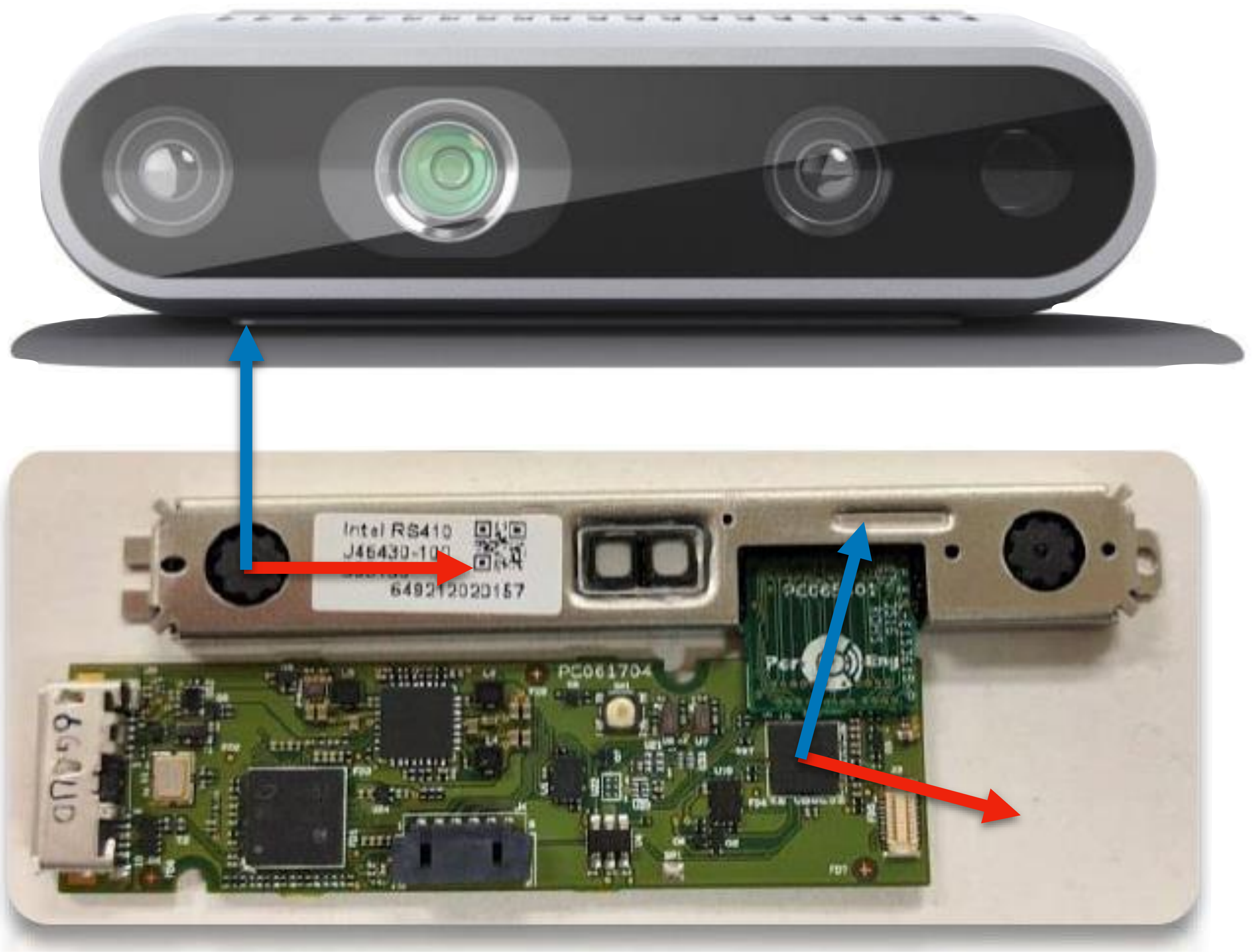
Take many measurements (possibly from many sensors), and recover the ego-motion, map, and calibration parameters.

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_i \\ \vdots \\ z_N \end{bmatrix} = \begin{bmatrix} h_1\left(\mathbf{x}, \mathbf{m}, \kappa\right) + \epsilon_{z_1} \\ h_2\left(\mathbf{x}, \mathbf{m}, \kappa\right) + \epsilon_{z_2} \\ \vdots \\ h_i\left(\mathbf{x}, \mathbf{m}, \kappa\right) + \epsilon_{z_i} \\ \vdots \\ h_N\left(\mathbf{x}, \mathbf{m}, \kappa\right) + \epsilon_{z_N} \end{bmatrix} \longrightarrow \boxed{\text{SLAM System}} \longrightarrow \tilde{\mathbf{x}}, \tilde{\mathbf{m}}, \tilde{\kappa}$$

Example - Calibration

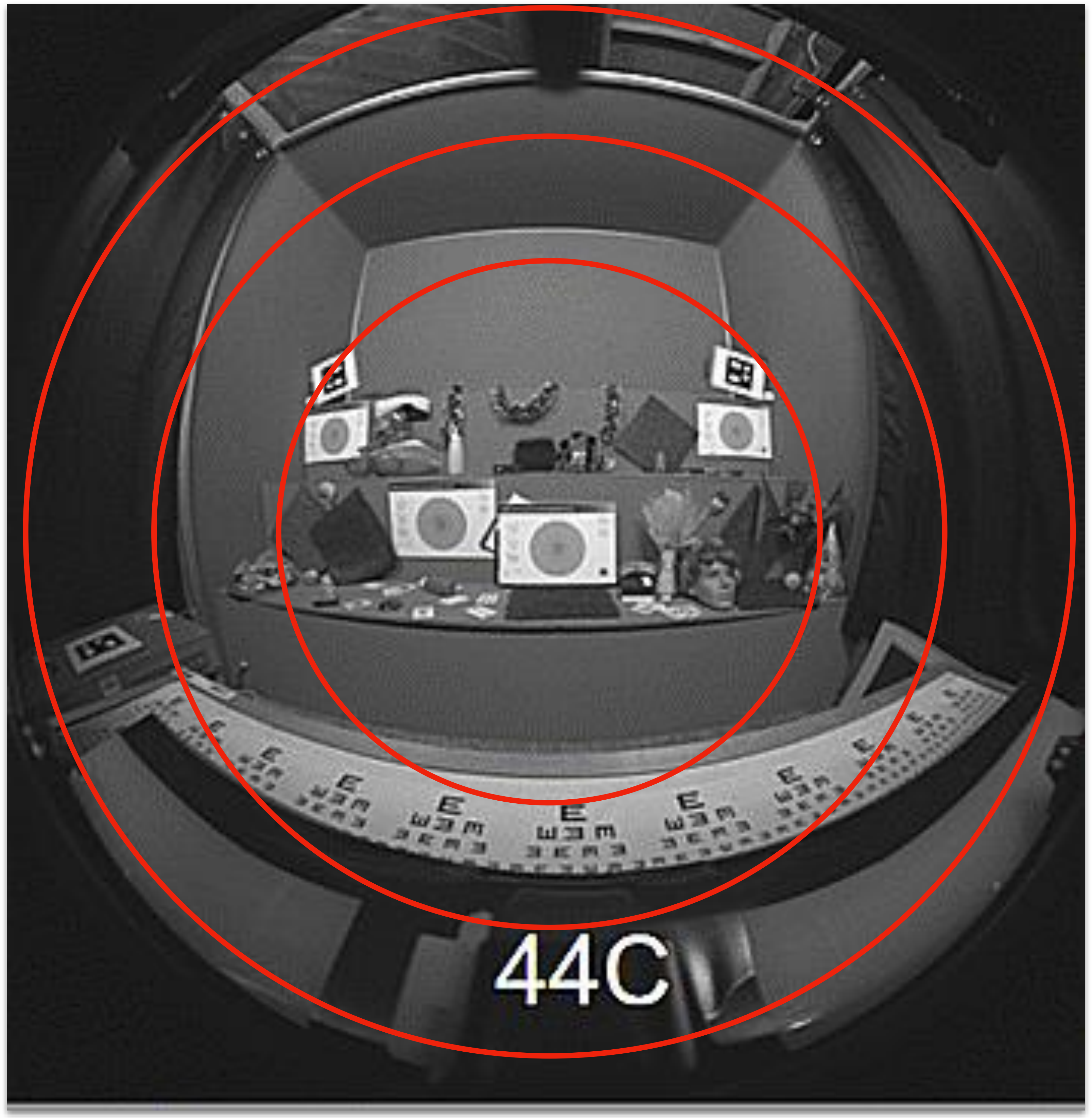
Tire inflation will affect the scale of wheel odometry, as could slippage between the tire and the road surface.

Image source: [MotorTrend.com]



Un-modelled extrinsic rotation between IMU and camera may cause increased drift in a visual SLAM pipeline.

Image source: [MWee RF Microwave]



Intrinsic temperature distortion may also introduce unexpected errors into vision estimates.

Image Source: Skydio

Example - Rolling Shutter

When selecting a camera sensor for your platform, you have the choice of *global* or *rolling* shutter.

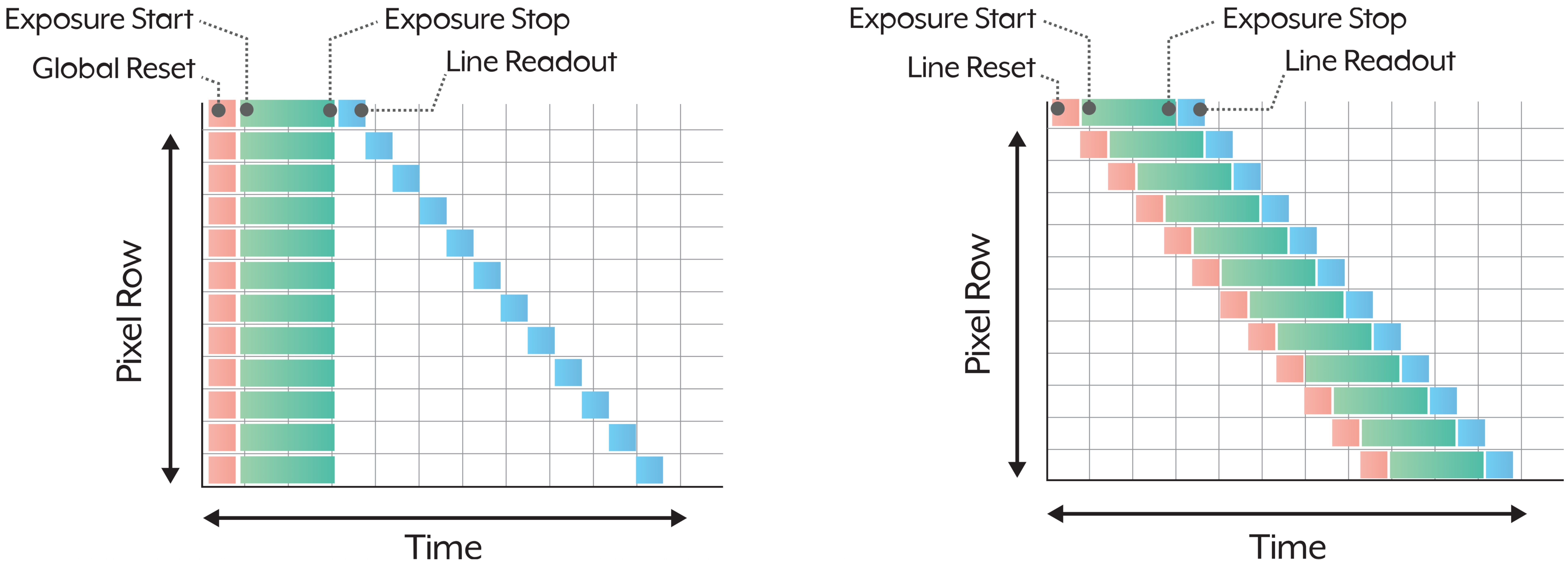


Image credit: [LucidVision]

Example - Rolling Shutter

Rolling shutter deforms rigid objects like the horizon line and the vehicle itself.

Global-shutter model:

$$\mathbf{z} = \pi \left(T_w^c \left(t_0 \right) \oplus p_w \right)$$

Camera projection

Point in the world.

Measured location in image.

Pose of the camera at time t_0

Transformation of world points into sensor frame.

Rolling-shutter model:

$$\mathbf{z} = \pi \left(T_w^c \left(t_0 \right) \int_{t_0}^{\tau} \dot{T}_w^c \left(t \right) dt \oplus p_w \right)$$

Inclusion of higher-order derivatives in the measurement model increases computational cost.

What about noise?

$$\mathbf{z} = h\left(\mathbf{x}, \mathbf{m}, \kappa\right) \boxed{+\, \epsilon_z}$$

- All sensors exhibit some minimum amount of *noise*.
- We distinguish between *noise* and *model error*.

A random error that can only be modeled via statistical means.

Example: thermal electrical noise.

Errors resulting from a limitation in our sensor model.

Example: failure to include a calibration parameter.

Uncertainty

- Owing to noise in the sensor inputs, SLAM is an inherently uncertain process.
- We can never recover the “true” states, only uncertain estimates of them.
  - More measurements *usually* means reduced uncertainty…
  - … But it also means increased computational cost.

The Map

Choice of sensor may also influence map parameterization.

Collection of photographs?

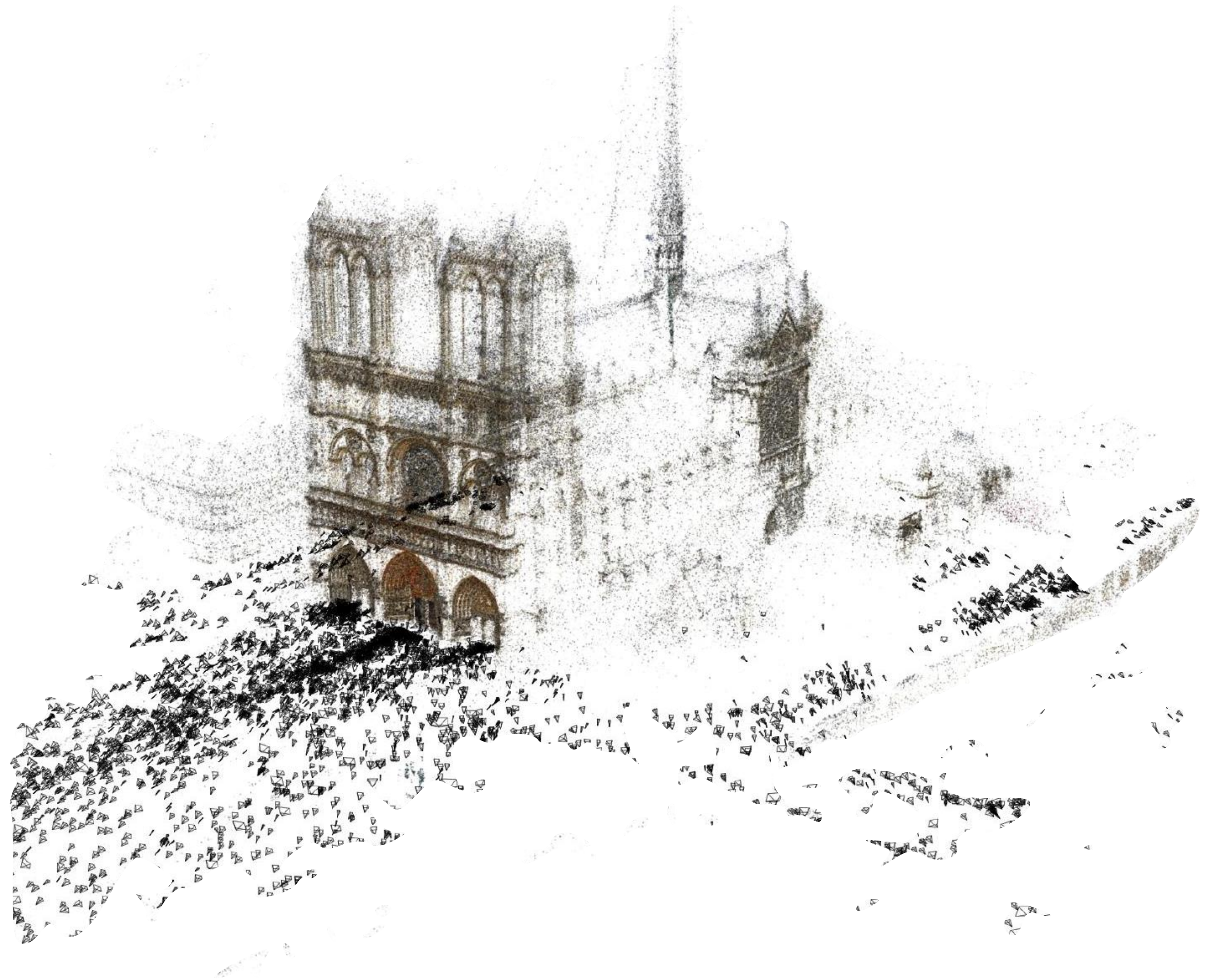


Image Source: [Noah Snavely]

2D LiDAR scans?

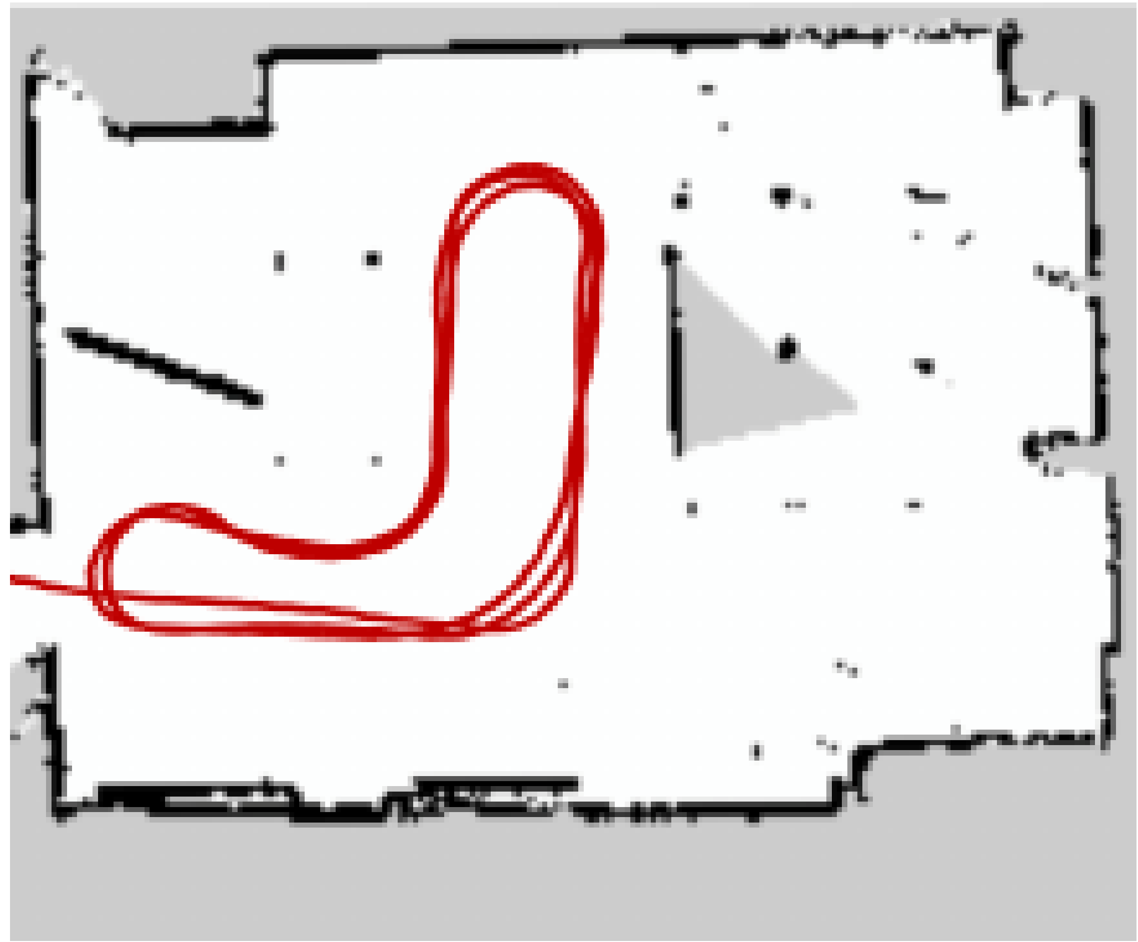


Image Source: B. Bellekens et al., A Benchmark Survey of Rigid 3D Point Cloud Registration Algorithms, 2015

3D range images?

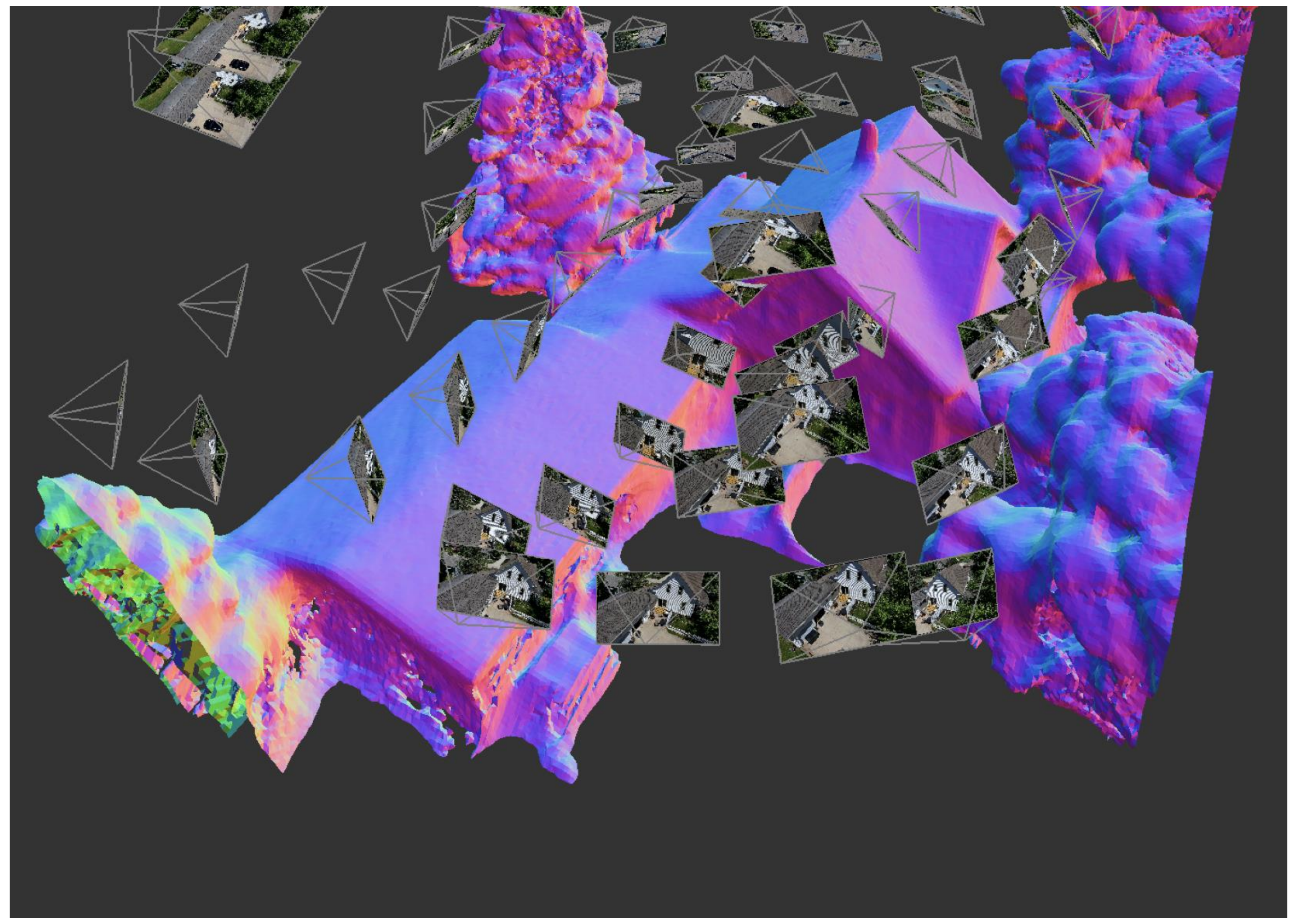
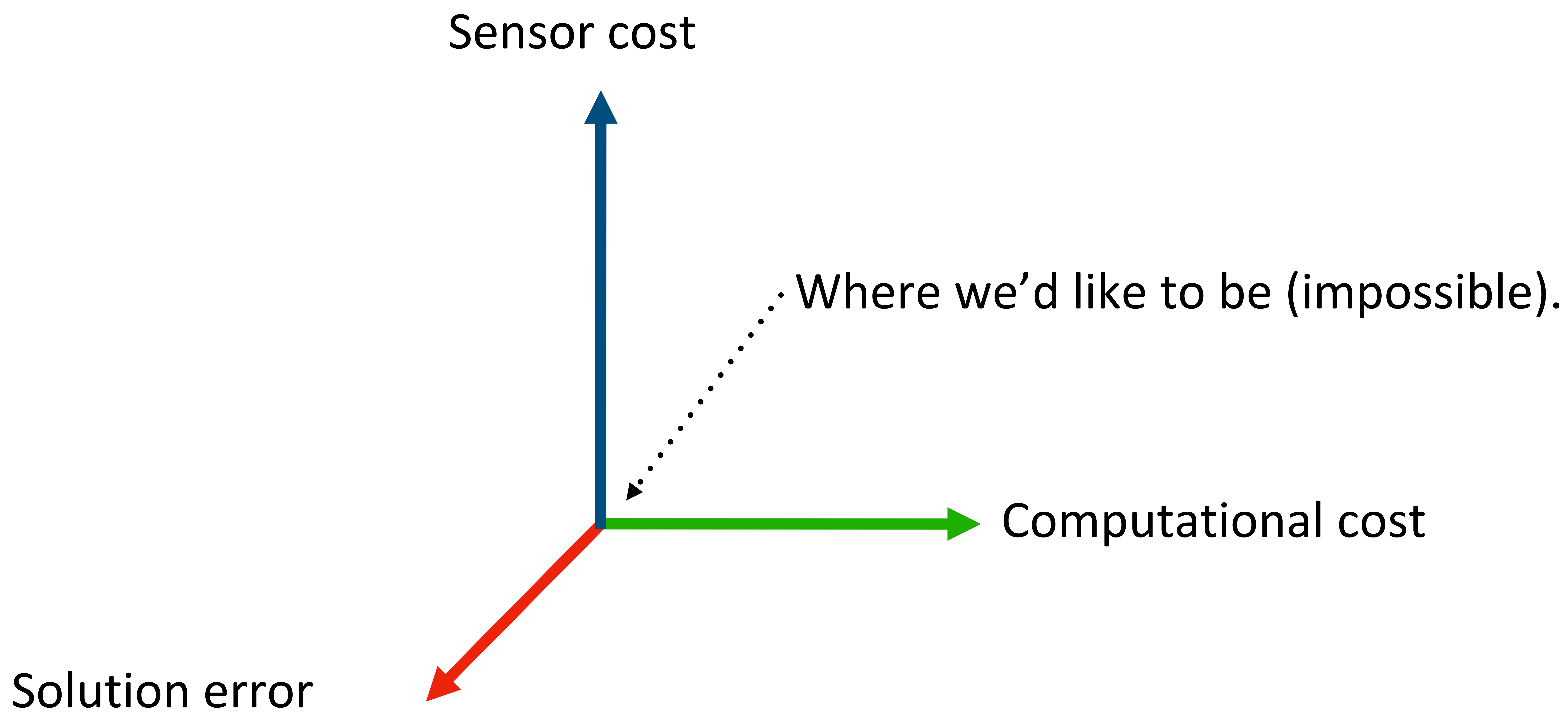


Image Source: Skydio

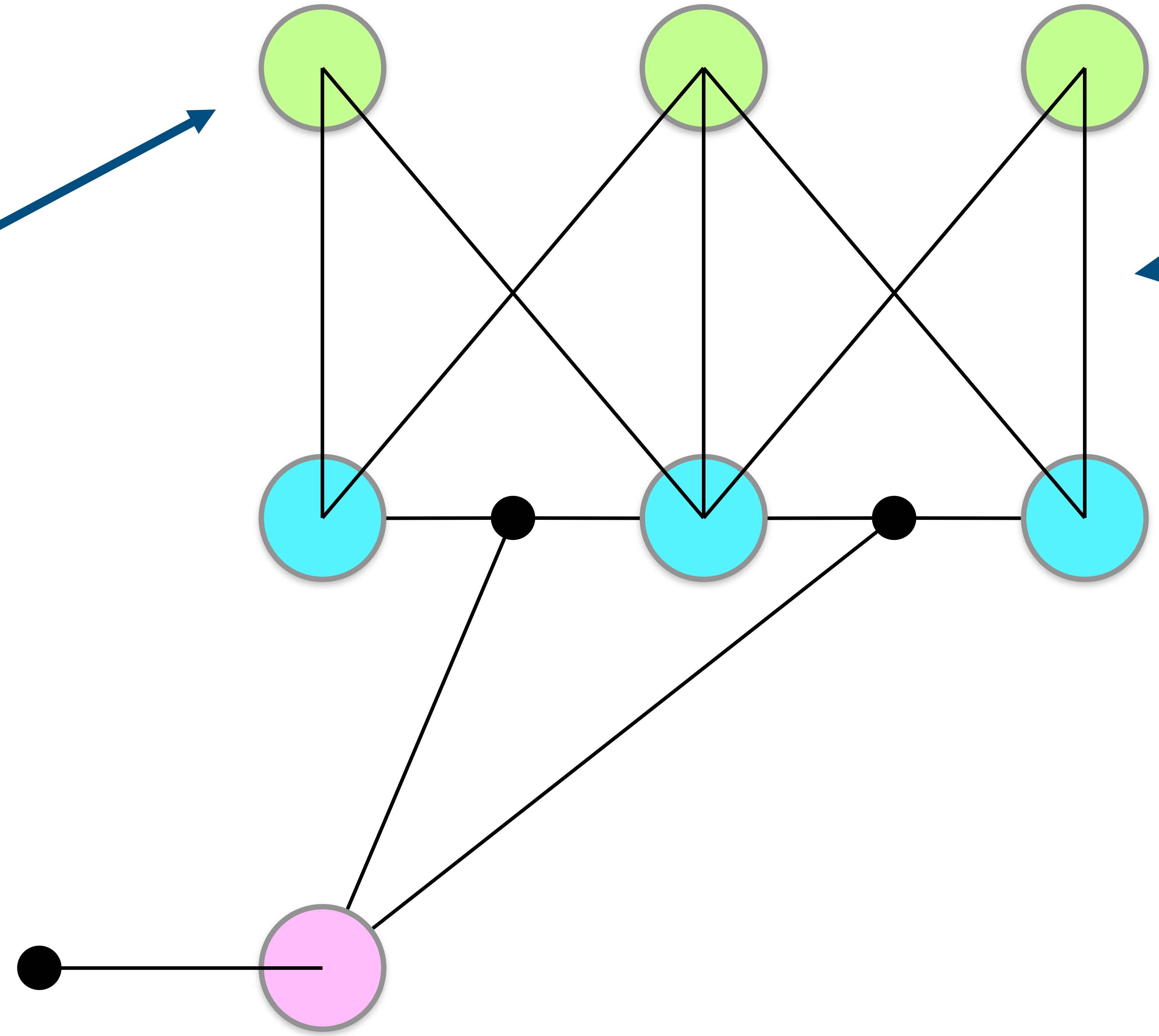
Design Trade-offs



Factor Graphs

Factor Graphs are a convenient method of graphically representing a SLAM problem.

Nodes represent states



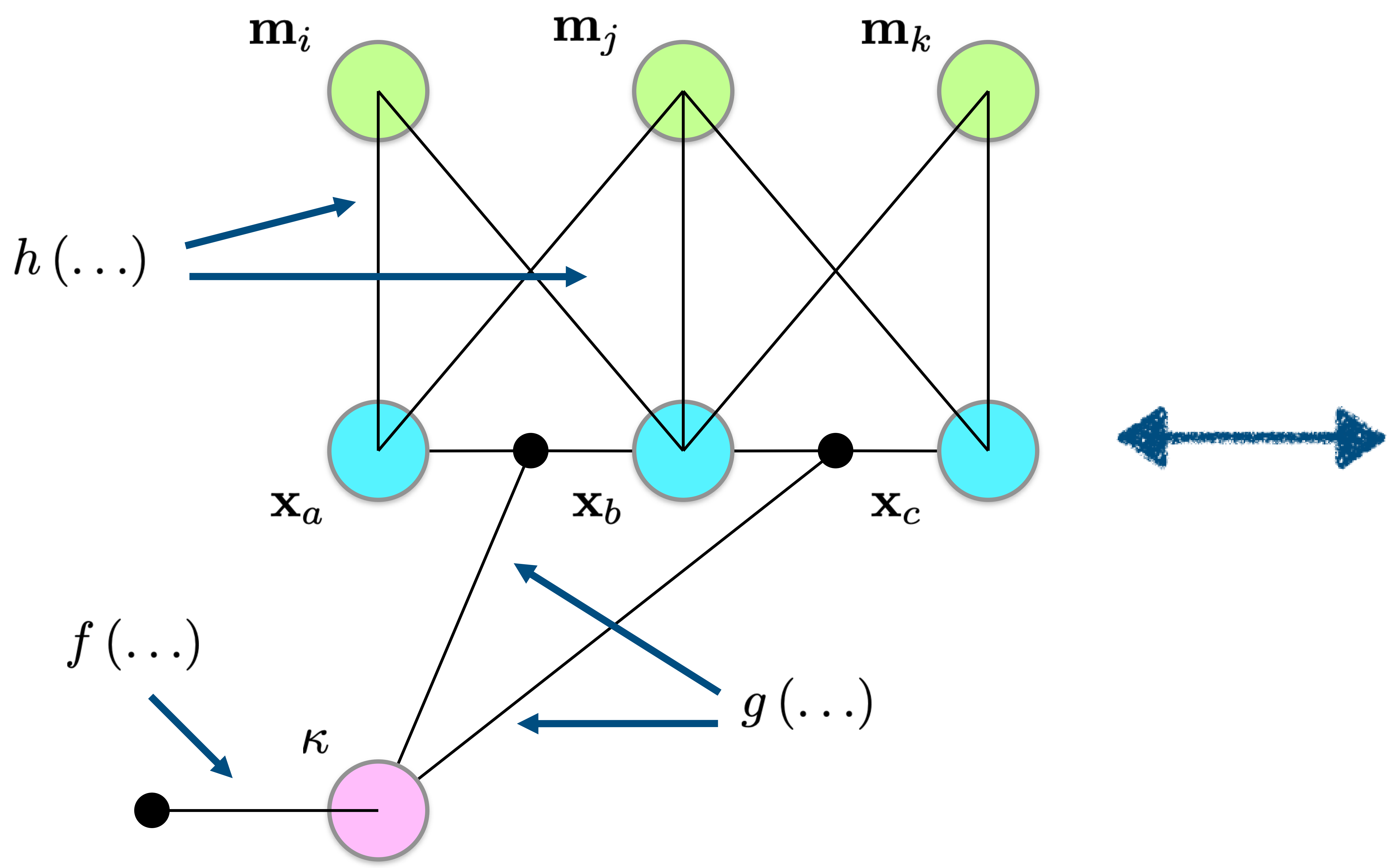
Edges (*factors*) represent information about the states, in the form of measurements or priors.

Factors may be unary, binary, ternary, etc…

In a SLAM problem, we will typically have nodes for our ego-motion, map, and calibration parameters.

Skydio

There is a mapping from the factor graph to our sensor measurements:



$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \\ z_8 \\ z_9 \\ z_{10} \end{bmatrix} = \begin{bmatrix} h(\mathbf{x}_a, \mathbf{m}_i) + \epsilon_{z_1} \\ h(\mathbf{x}_a, \mathbf{m}_j) + \epsilon_{z_2} \\ h(\mathbf{x}_b, \mathbf{m}_i) + \epsilon_{z_3} \\ h(\mathbf{x}_b, \mathbf{m}_j) + \epsilon_{z_4} \\ h(\mathbf{x}_b, \mathbf{m}_k) + \epsilon_{z_5} \\ h(\mathbf{x}_c, \mathbf{m}_j) + \epsilon_{z_6} \\ h(\mathbf{x}_c, \mathbf{m}_k) + \epsilon_{z_7} \\ g(\mathbf{x}_a, \mathbf{x}_b, \kappa) + \epsilon_{z_8} \\ g(\mathbf{x}_b, \mathbf{x}_c, \kappa) + \epsilon_{z_9} \\ f(\kappa) + \epsilon_{z_{10}} \end{bmatrix}$$

Skydio

Real Example: Bundle Adjustment (BA)

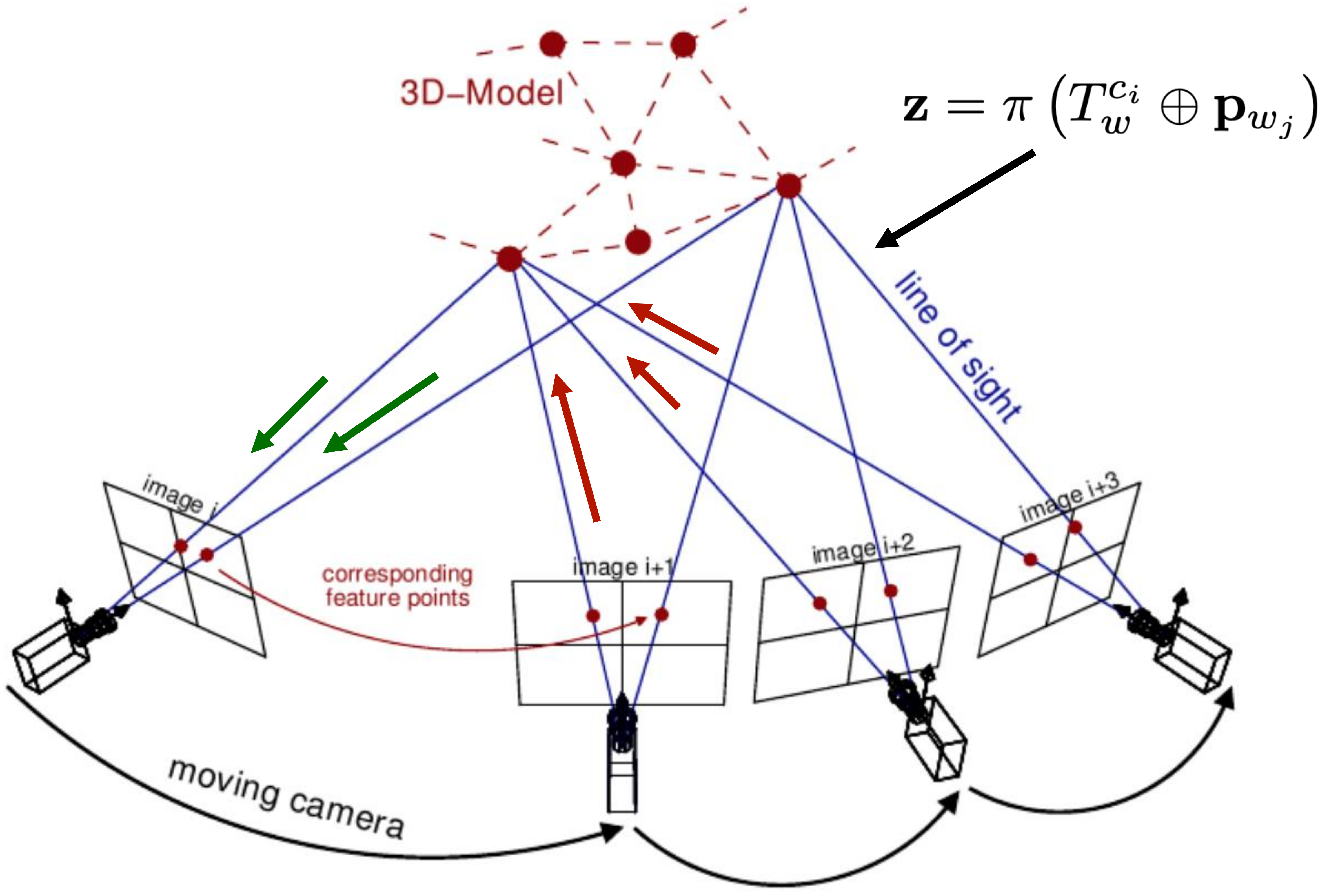


Image source: [Theia SFM]

- A form of *Structure from Motion (SFM)*.
- Leverage *projective geometry* to recover 3D landmarks and poses from 2D feature associations.
- Highly scalable and can be quite accurate.
- Using *marginalization* the compute cost can be bounded.

For more details on SFM, see Richard Szeliski's book as a jumping off point.

Structure From Motion (SFM)

Simultaneous

Localization and *Mapping*

Recover state of a vehicle or sensor platform, usually over multiple time-steps.

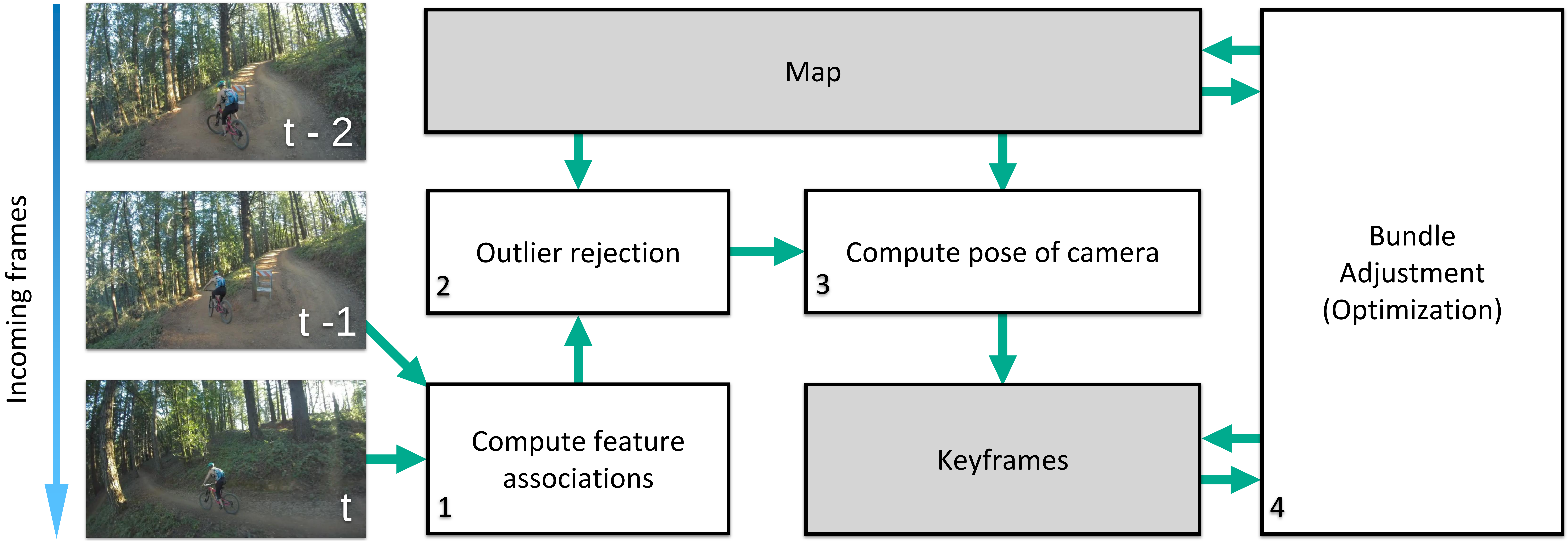
Recover camera pose with respect to map points

Recover location of landmarks in some common reference frame.

Triangulate map points using camera poses.

Bundle Adjustment is a form of optimization that does these steps *jointly*.

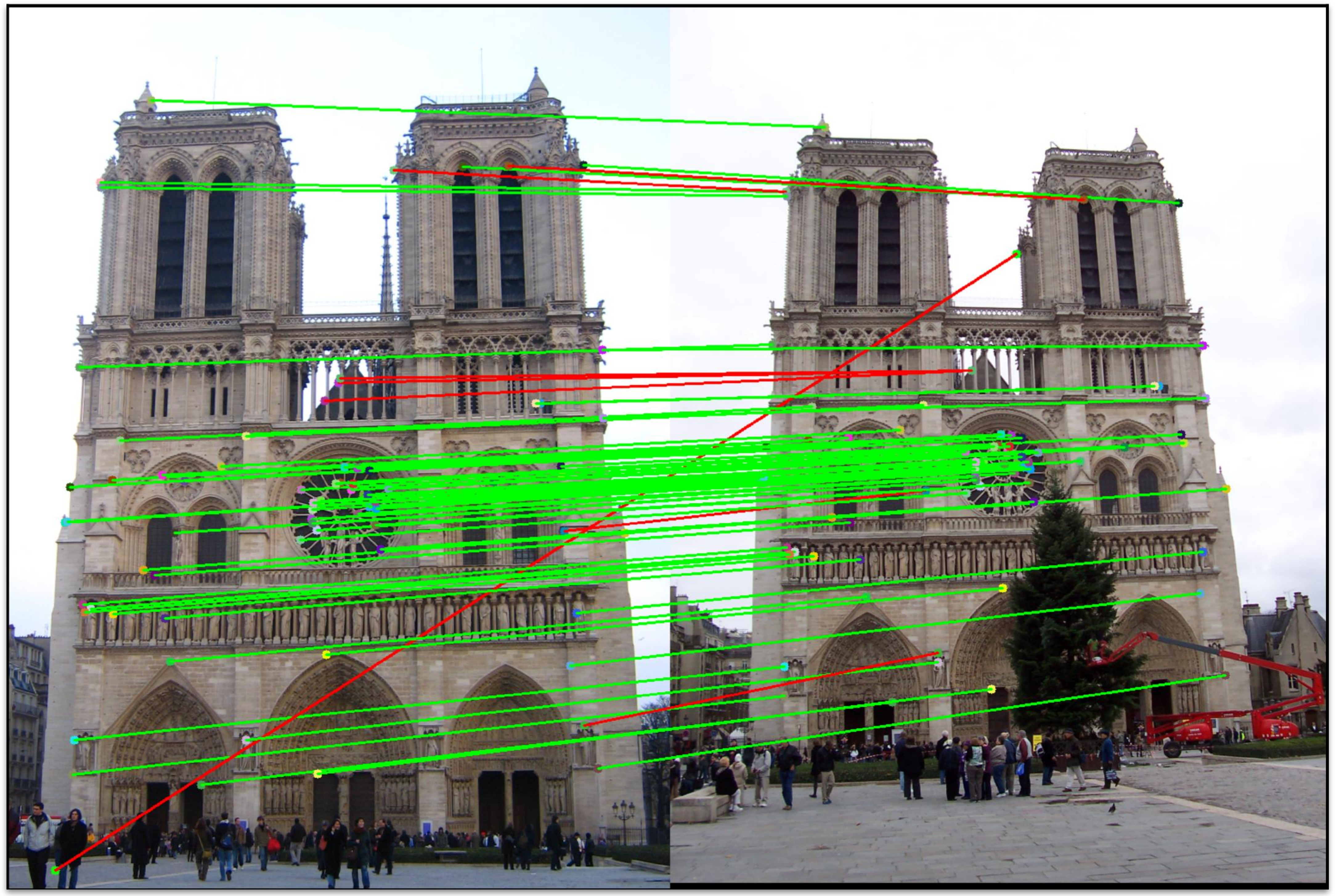
Typical SLAM Pipeline w/ BA



Keyframes store our estimates of the ego-motion.

How do we get feature associations?

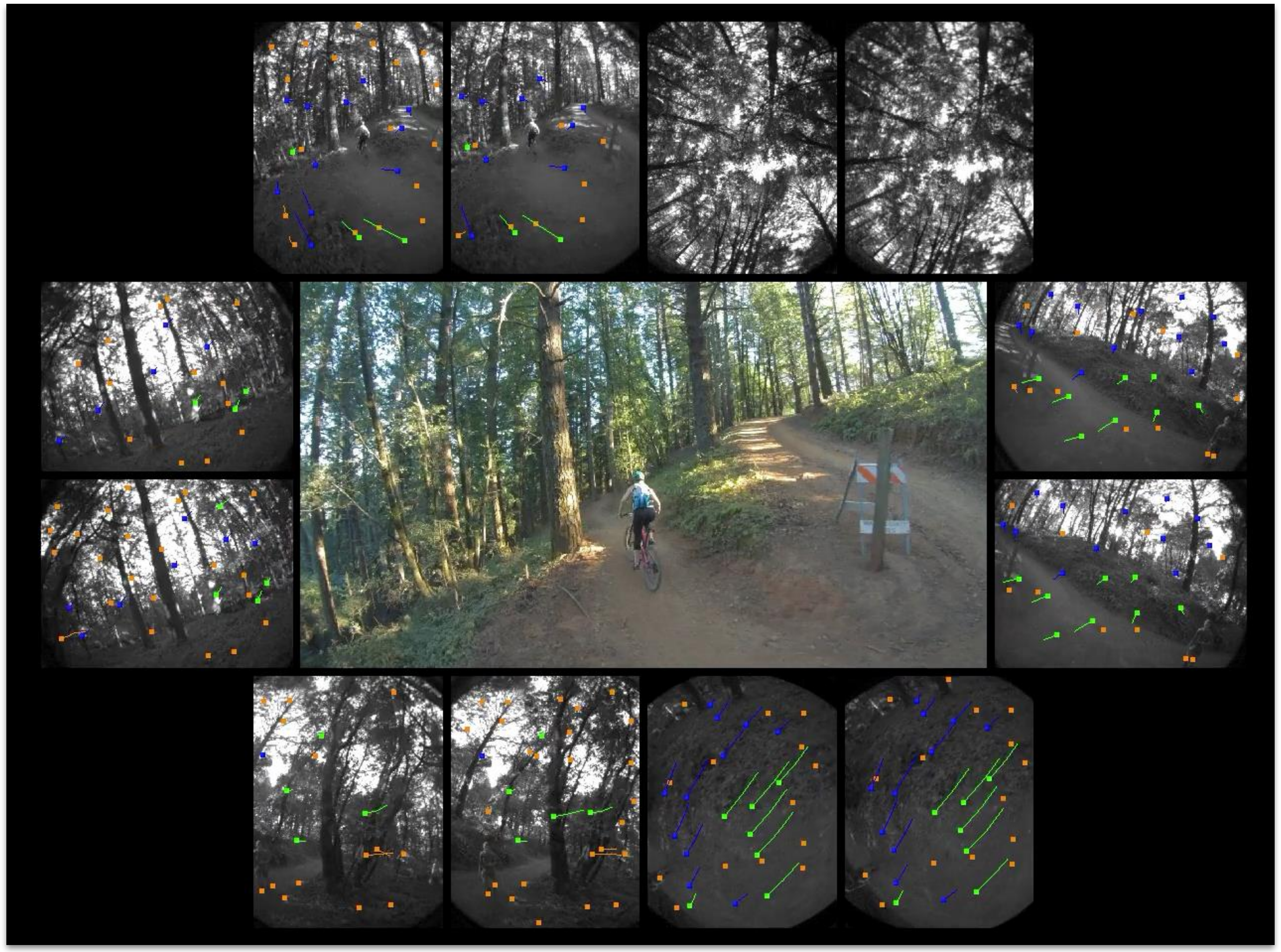
Descriptor Matching



Examples: [SIFT](), [KAZE](), [ORB](), [SuperPoint]()

Image Source: [Georgia Tech]()

Feature Tracking / Flow

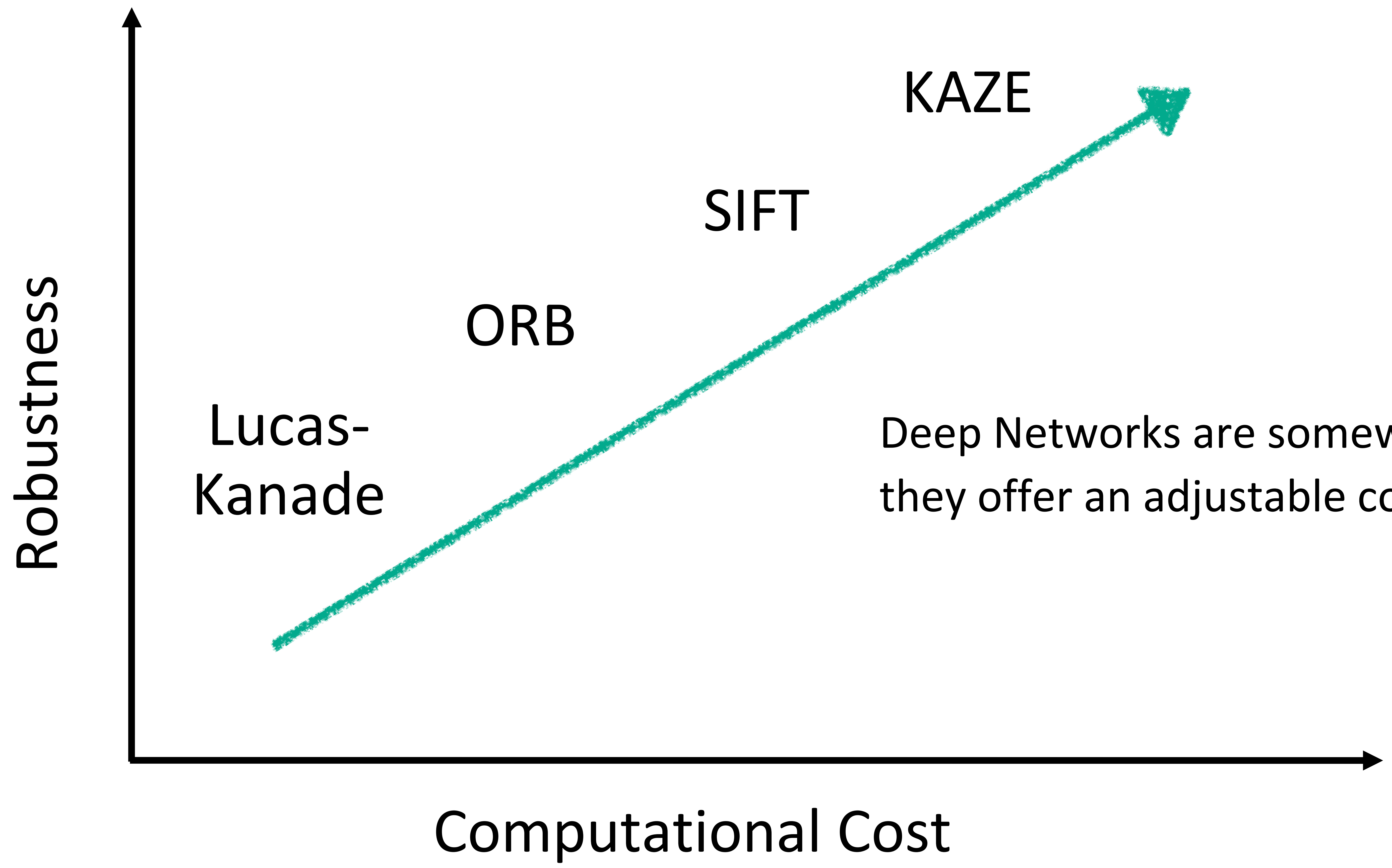


Examples: [Optical Flow](), [Lucas Kanade Tracking](), [FlowNet]()

Skydio

Design Trade-offs

A “rule of thumb” principle to consider in selecting features (axes not to scale):



Deep Networks are somewhat difficult to place since they offer an adjustable cost-robustness trade-off.

Outliers in Feature Association

Outliers

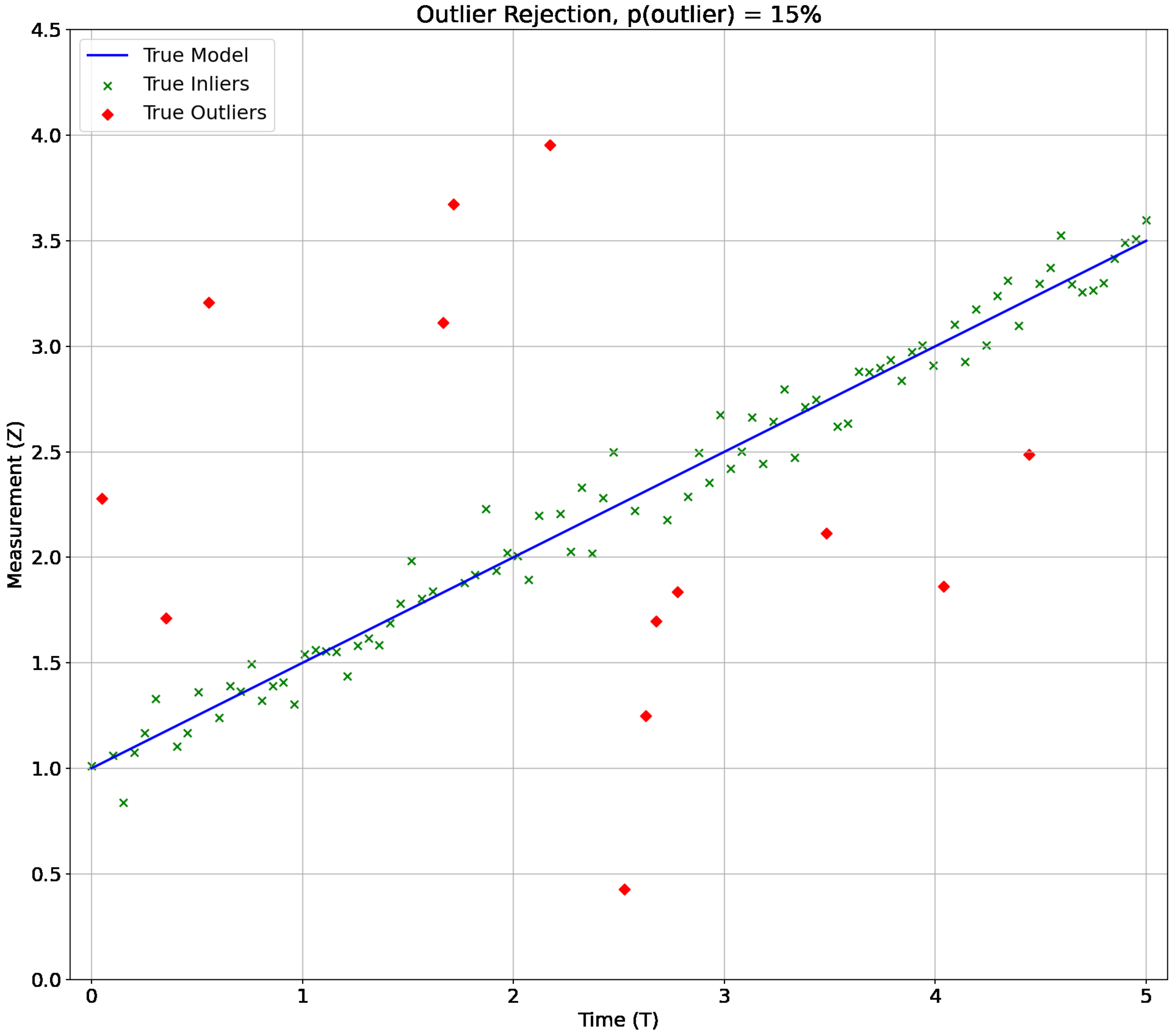
- *Outliers*: Data that does not agree with our sensor model.
- How do we deal with them?
- Let's review a (very simple) toy problem:

State: *alpha* and *beta*

$$\mathbf{z} = h\left(\alpha, \beta\right) = \alpha t + \beta + \epsilon_z$$

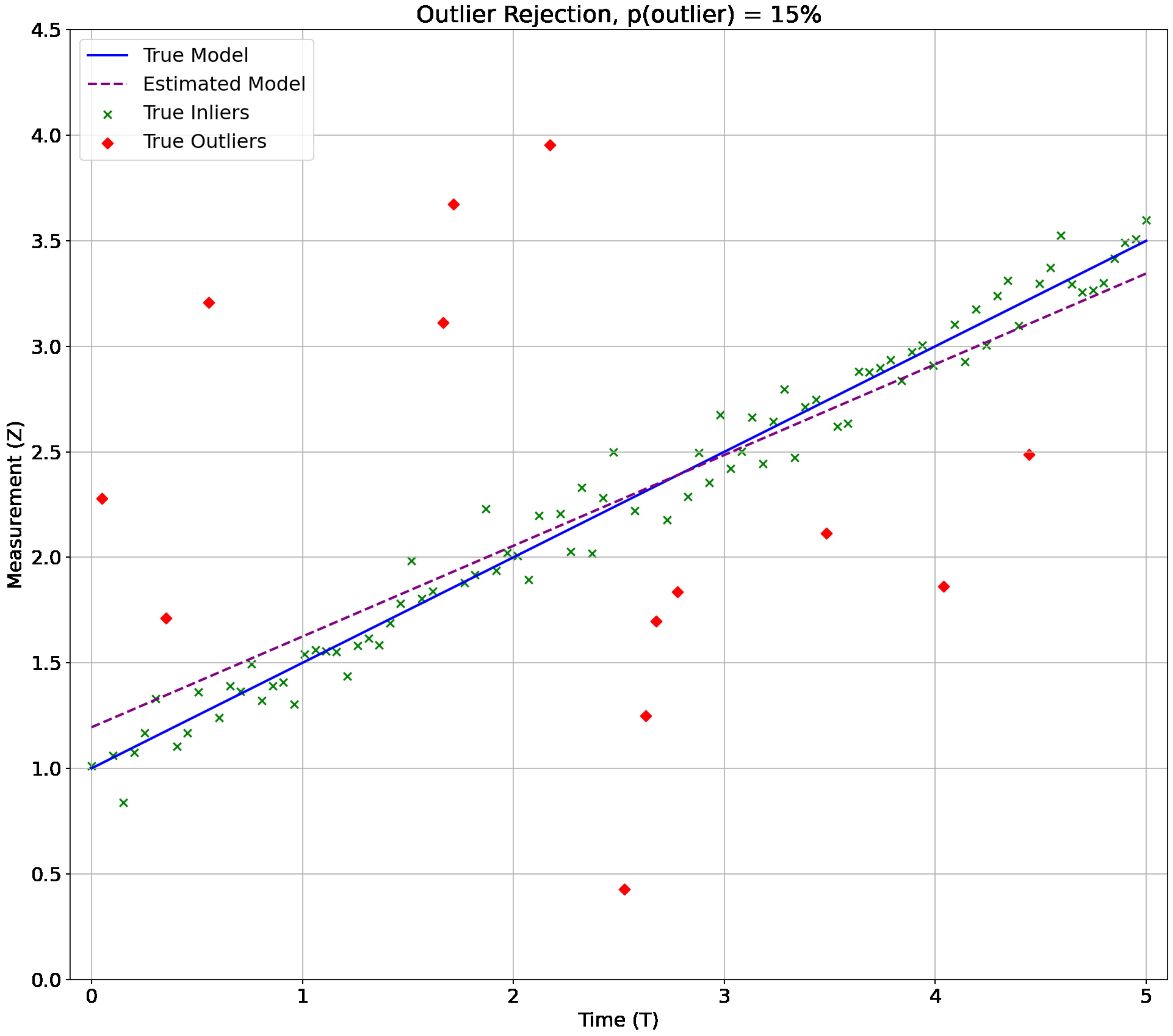
Measurement

Toy Problem



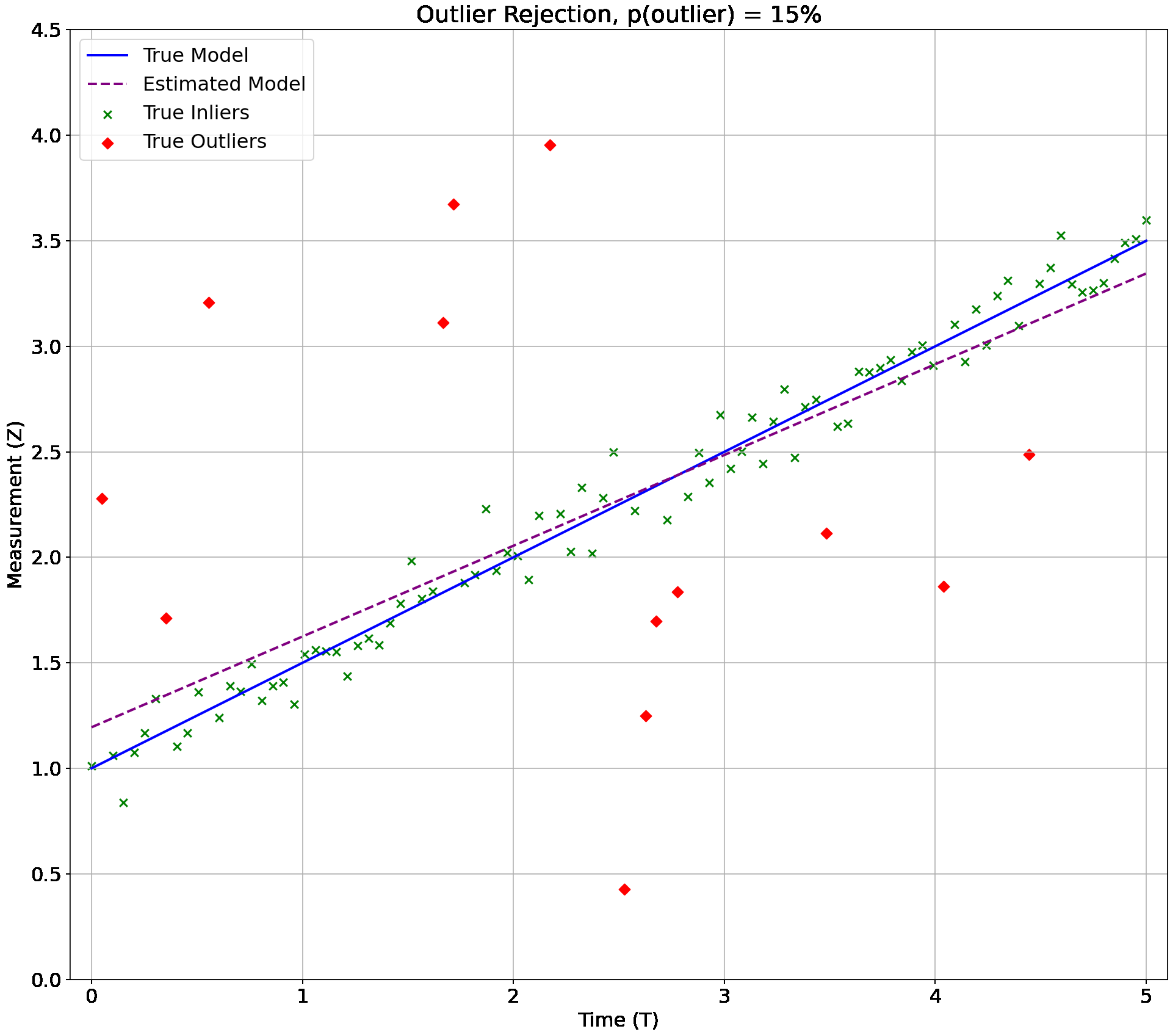
Outlier Rejection, p(outlier) = 15%
True Model
True Inliers
True Outliers
Measurement (Z)
Time (T)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
0
1
2
3
4
5

Toy Problem



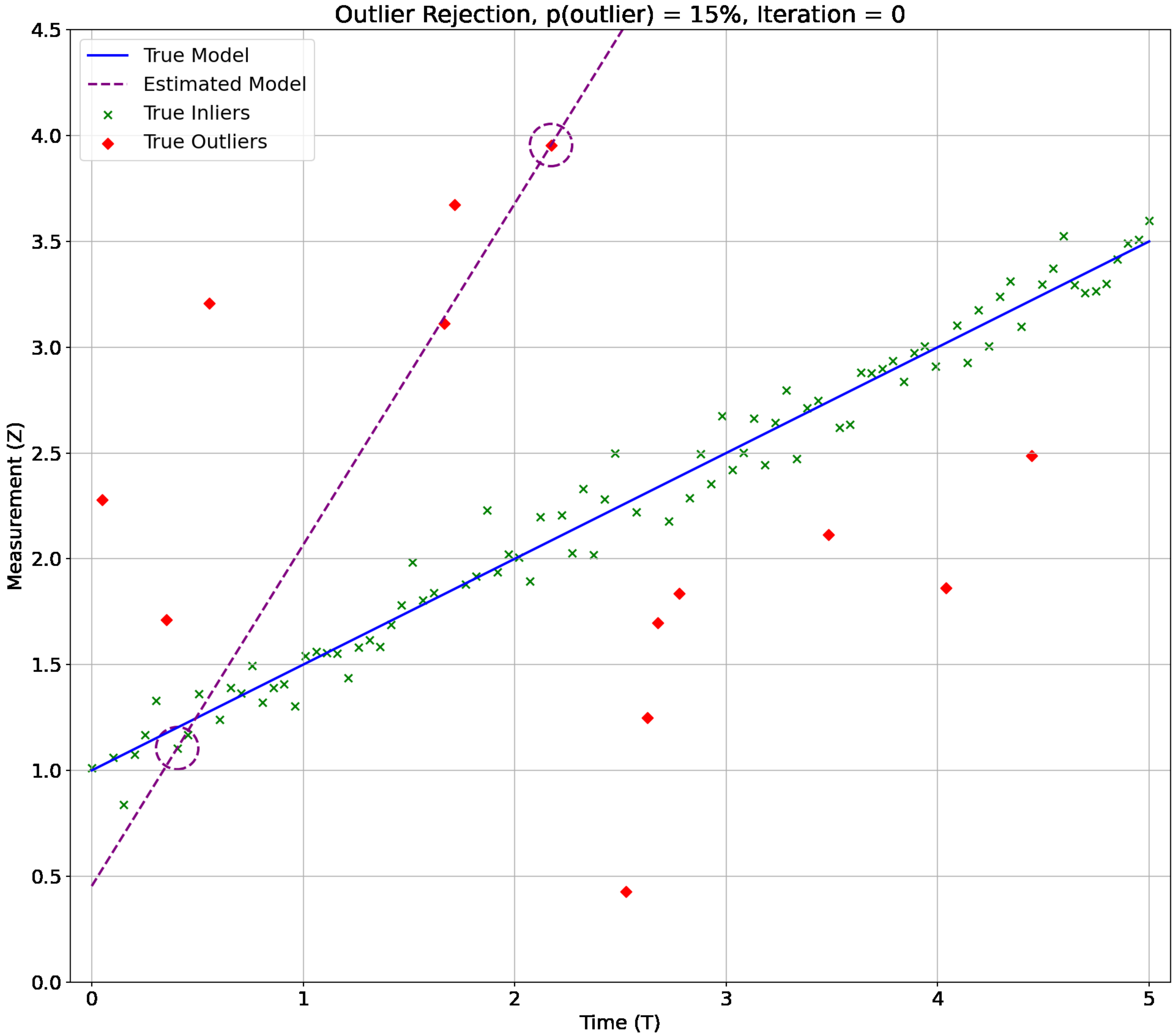
Outlier Rejection, p(outlier) = 15%
True Model
Estimated Model
True Inliers
True Outliers
Measurement (Z)
Time (T)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
0
1
2
3
4
5

RANSAC (Random Sample Consensus)



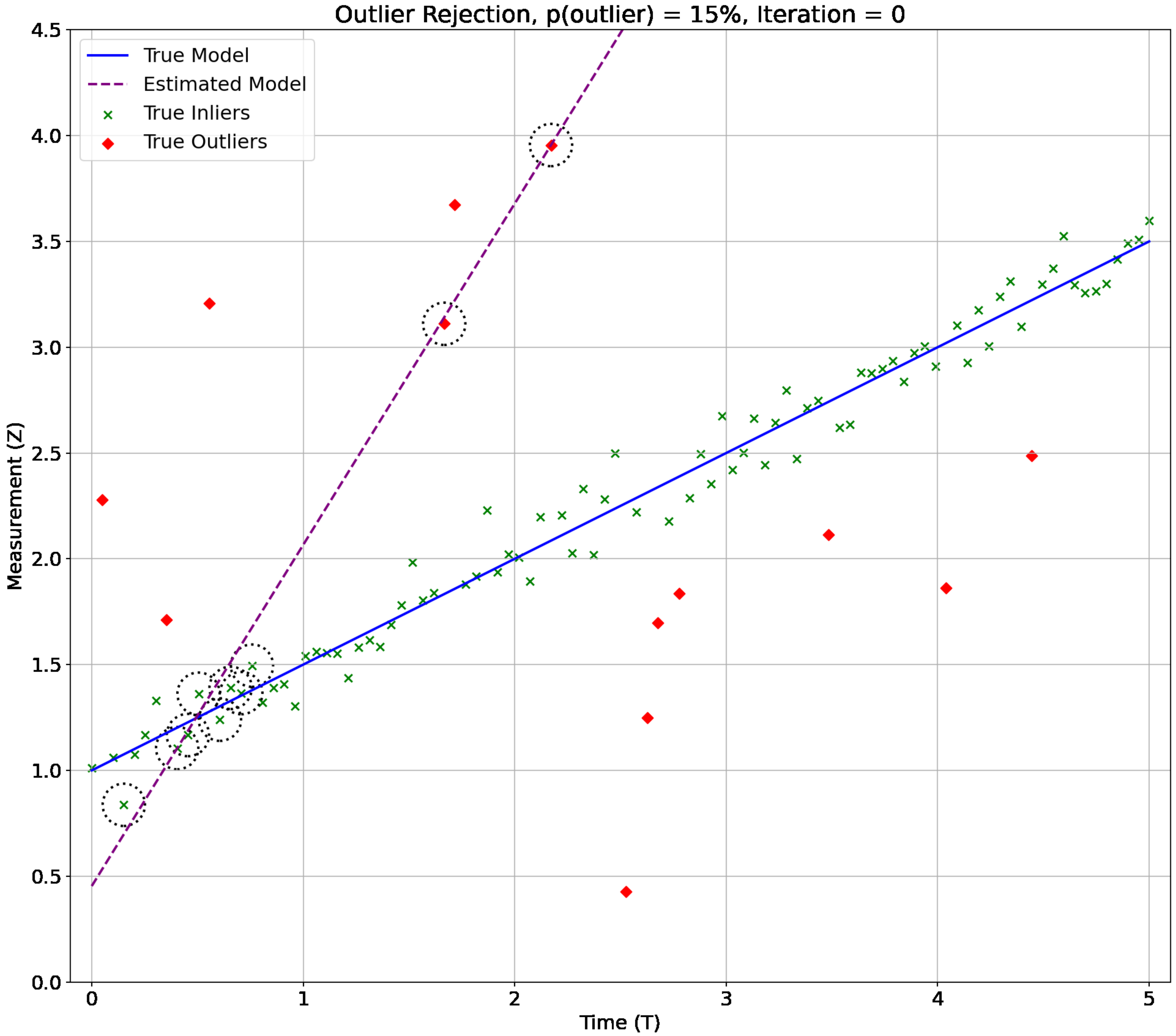
Skydio

RANSAC

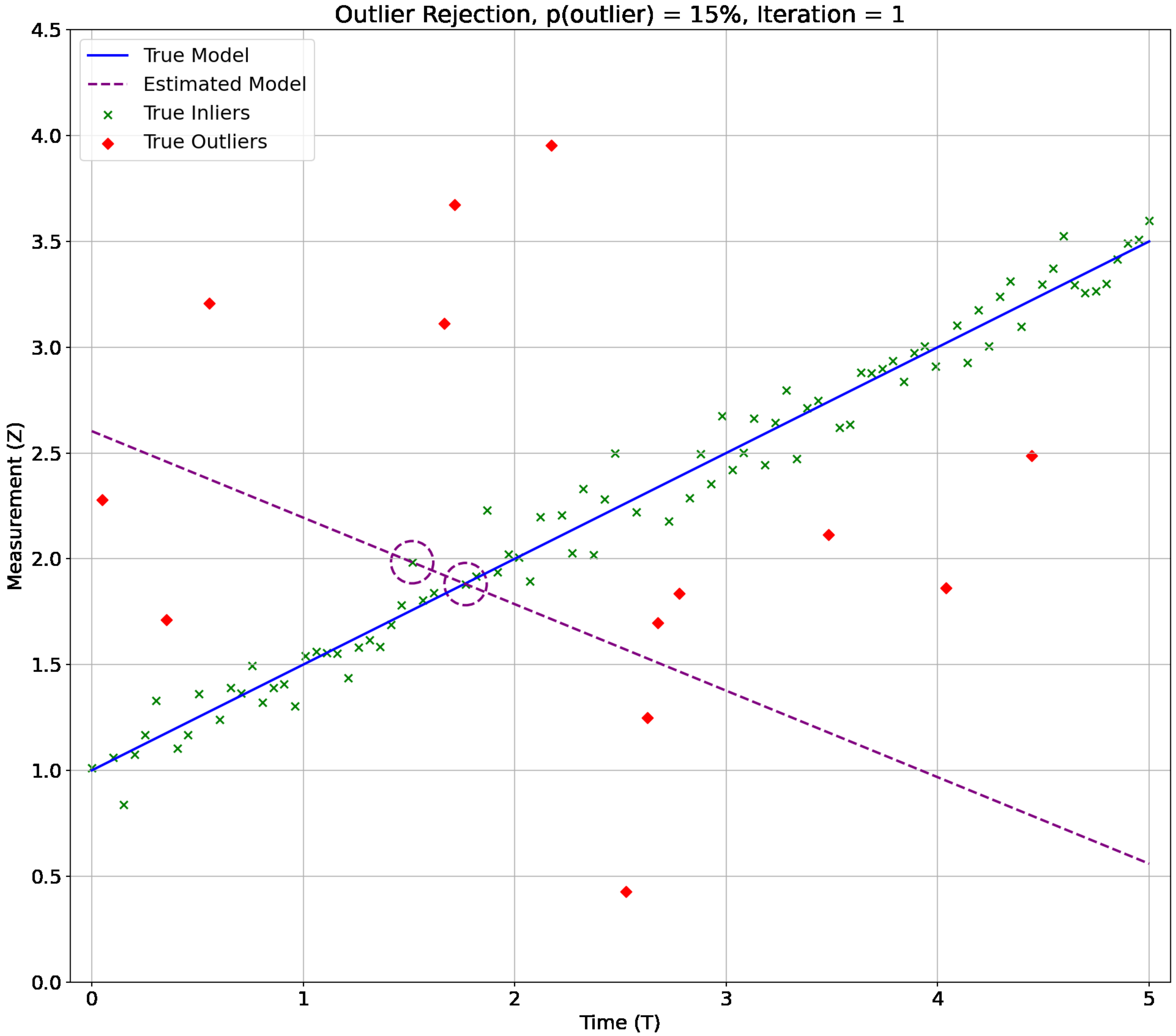


Outlier Rejection, p(outlier) = 15%, Iteration = 0
True Model
Estimated Model
True Inliers
True Outliers
Measurement (Z)
Time (T)
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5

RANSAC

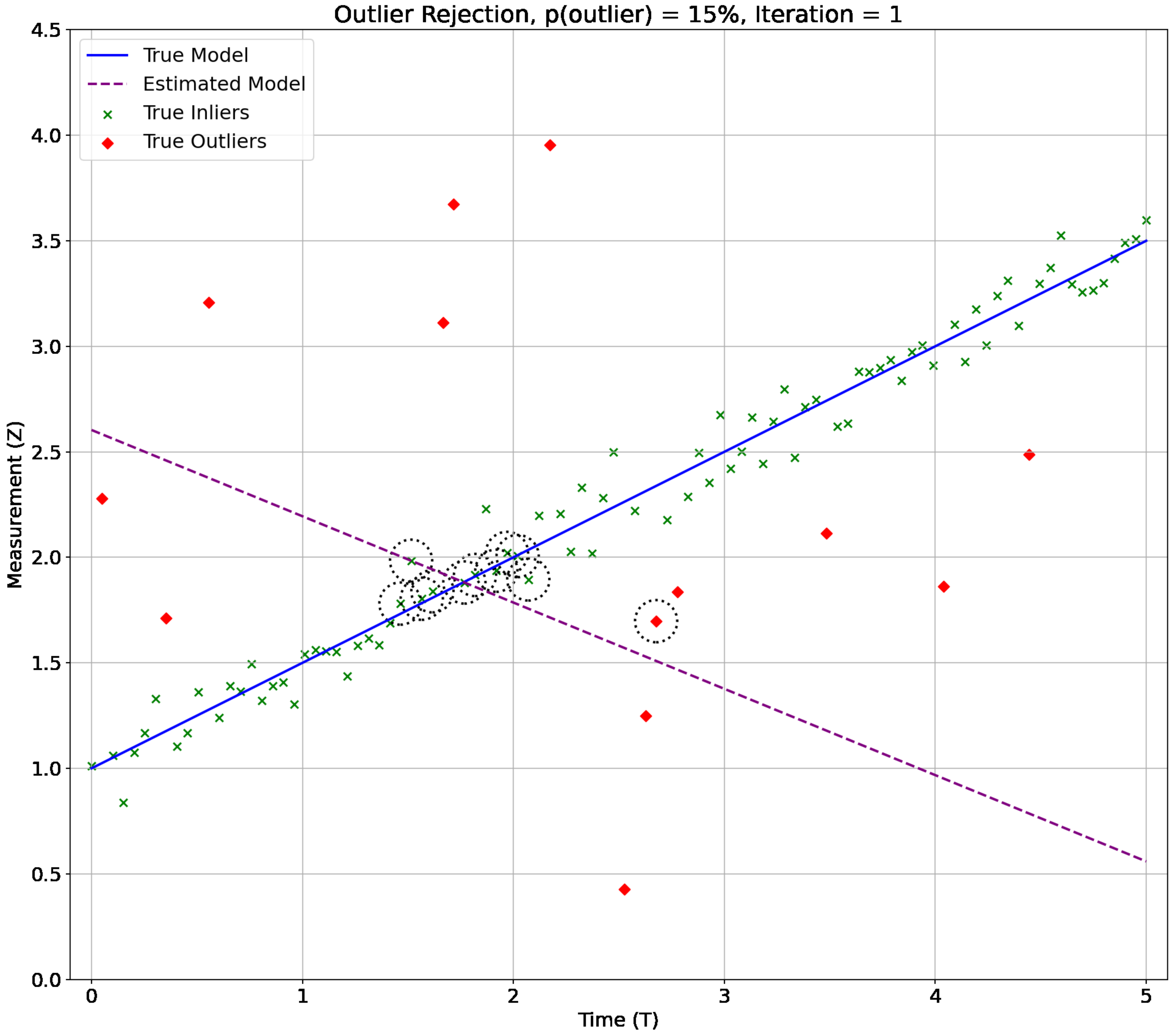

Outlier Rejection, p(outlier) = 15%, Iteration = 0
True Model
Estimated Model
True Inliers
True Outliers
Measurement (Z)
Time (T)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
0
1
2
3
4
5

RANSAC



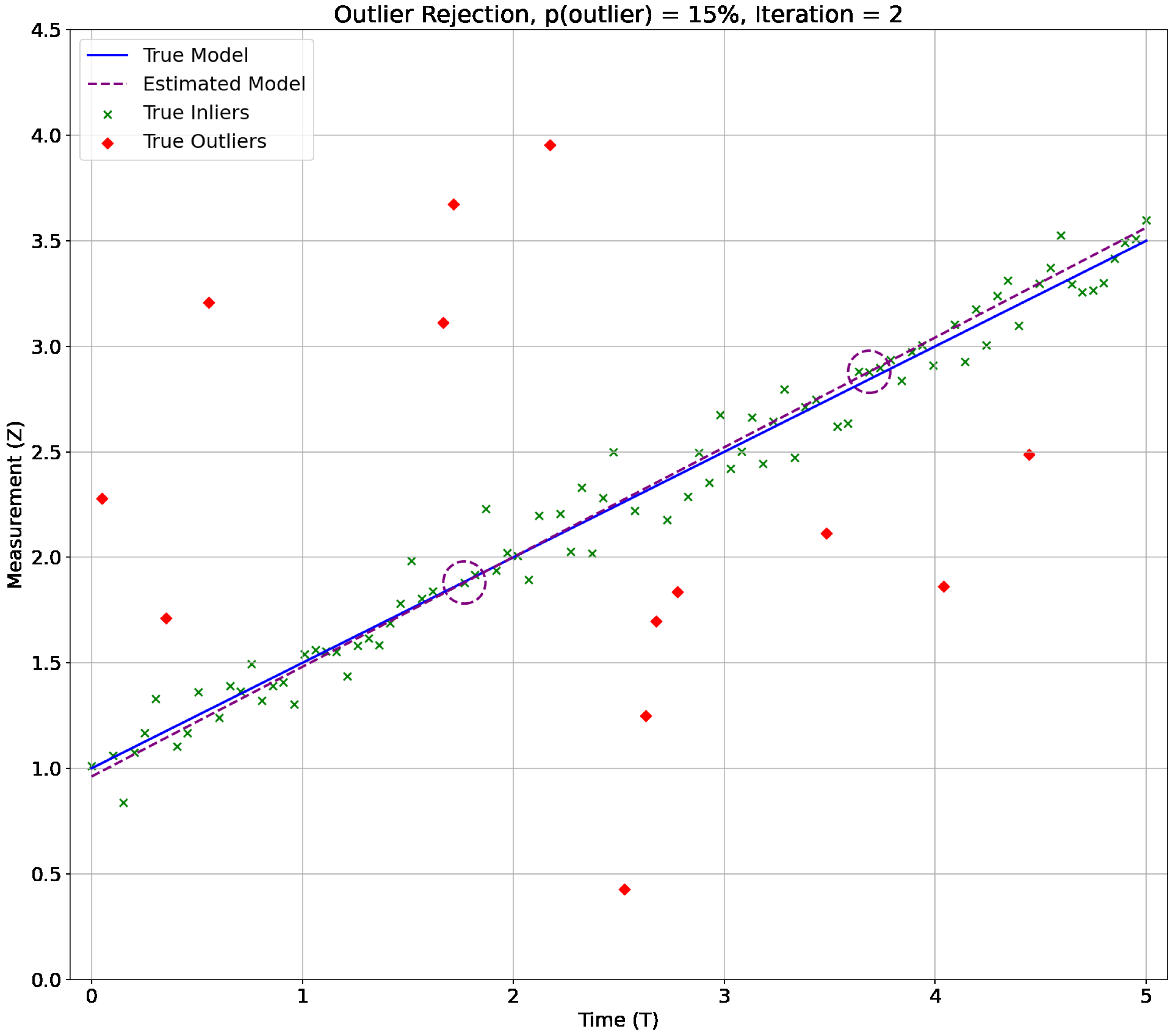
Outlier Rejection, p(outlier) = 15%, Iteration = 1
True Model
Estimated Model
True Inliers
True Outliers
Measurement (Z)
Time (T)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
0
1
2
3
4
5

RANSAC



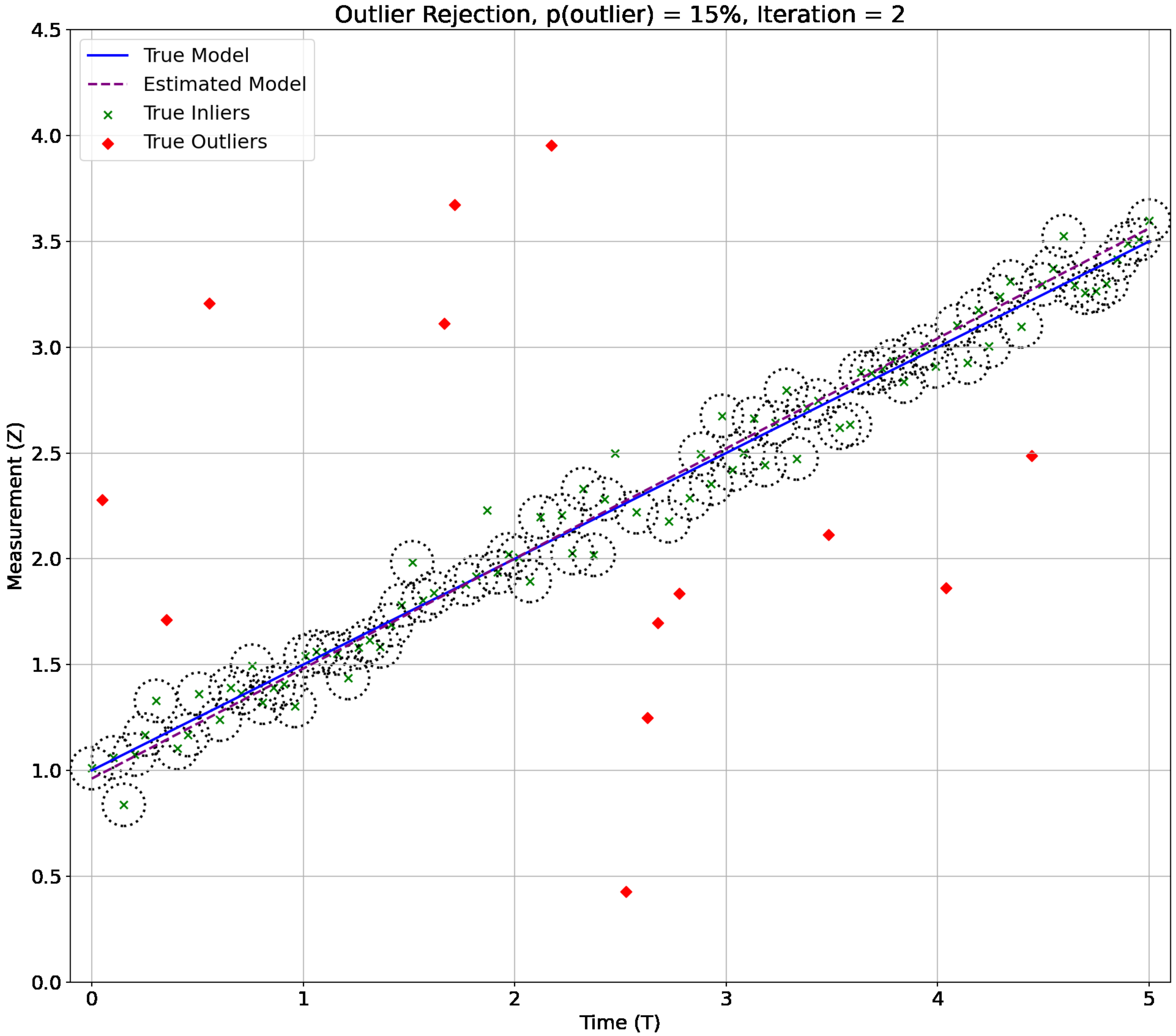
Outlier Rejection, p(outlier) = 15%, Iteration = 1
True Model
Estimated Model
True Inliers
True Outliers
Measurement (Z)
Time (T)
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5

RANSAC



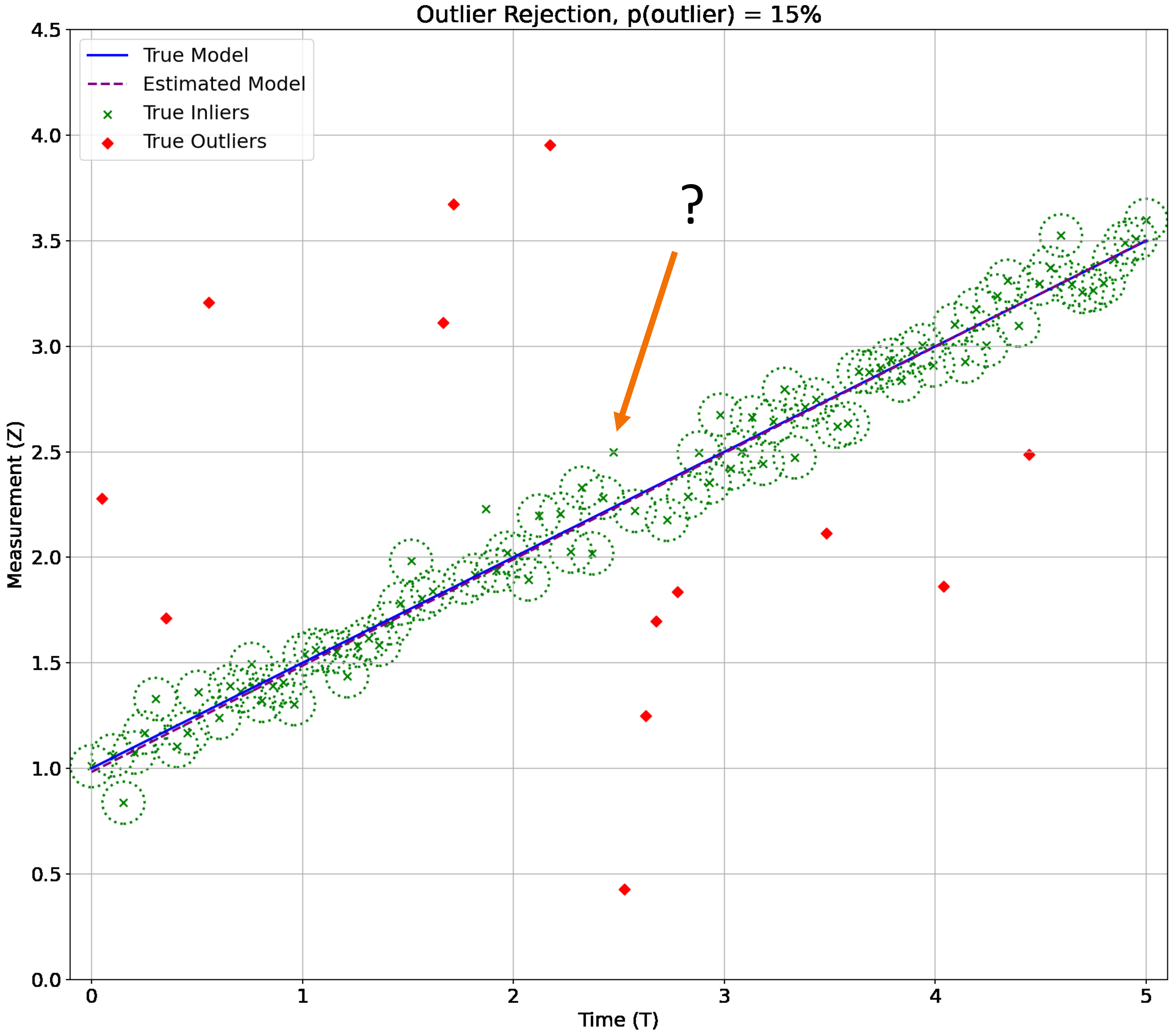
Outlier Rejection, p(outlier) = 15%, Iteration = 2
True Model
Estimated Model
True Inliers
True Outliers
Measurement (Z)
Time (T)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
0
1
2
3
4
5

RANSAC



Outlier Rejection, p(outlier) = 15%, Iteration = 2
True Model
Estimated Model
True Inliers
True Outliers
Measurement (Z)
Time (T)
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5

RANSAC



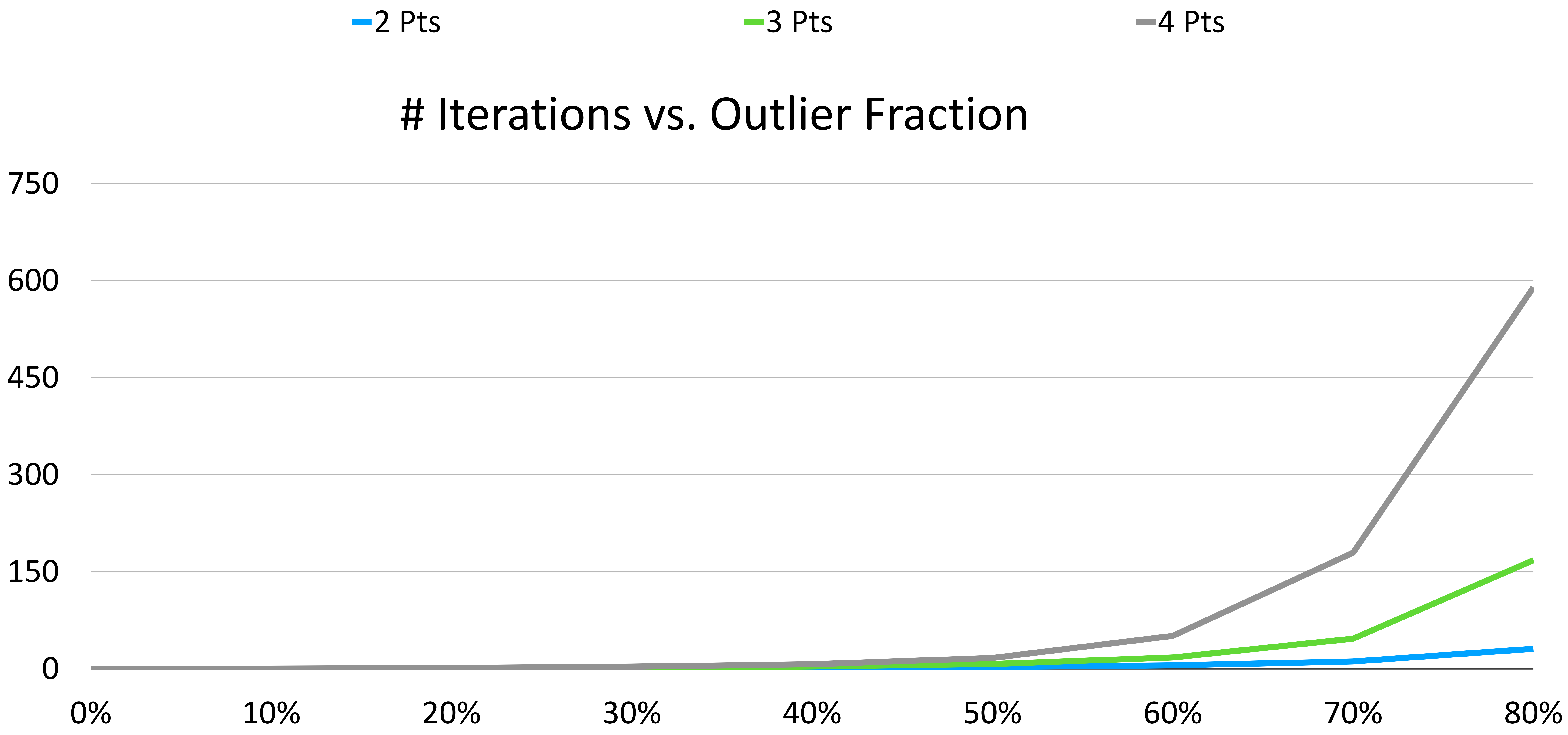
Skydio

RANSAC

- Pros:
  - Dead simple to implement: Draw *K* examples, solve, count, repeat.
  - Easily wrap around an existing method.
  - Trivially parallelized. Have more CPU time? Sample more.
- Cons:
  - Relatively weak guarantees.
  - Can require a lot of iterations for high outlier fractions or models with a large *K*.
  - Hyper-parameters need tuning.

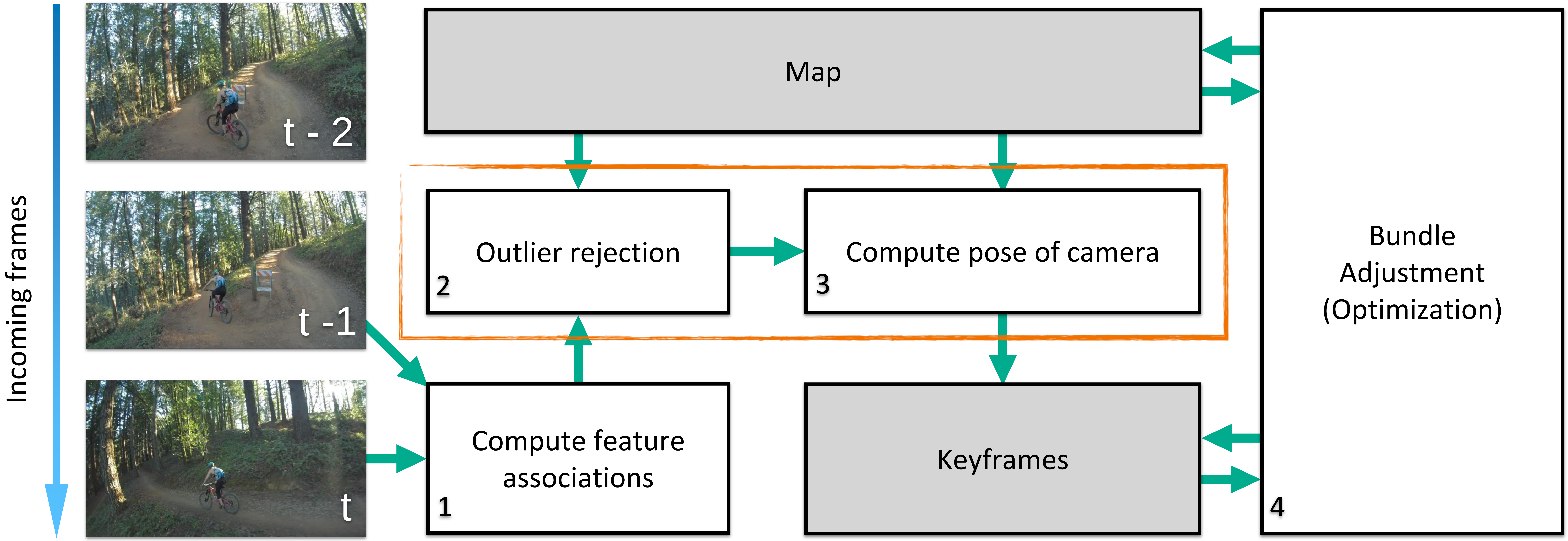
… *but*, still quite useful in practice.

RANSAC

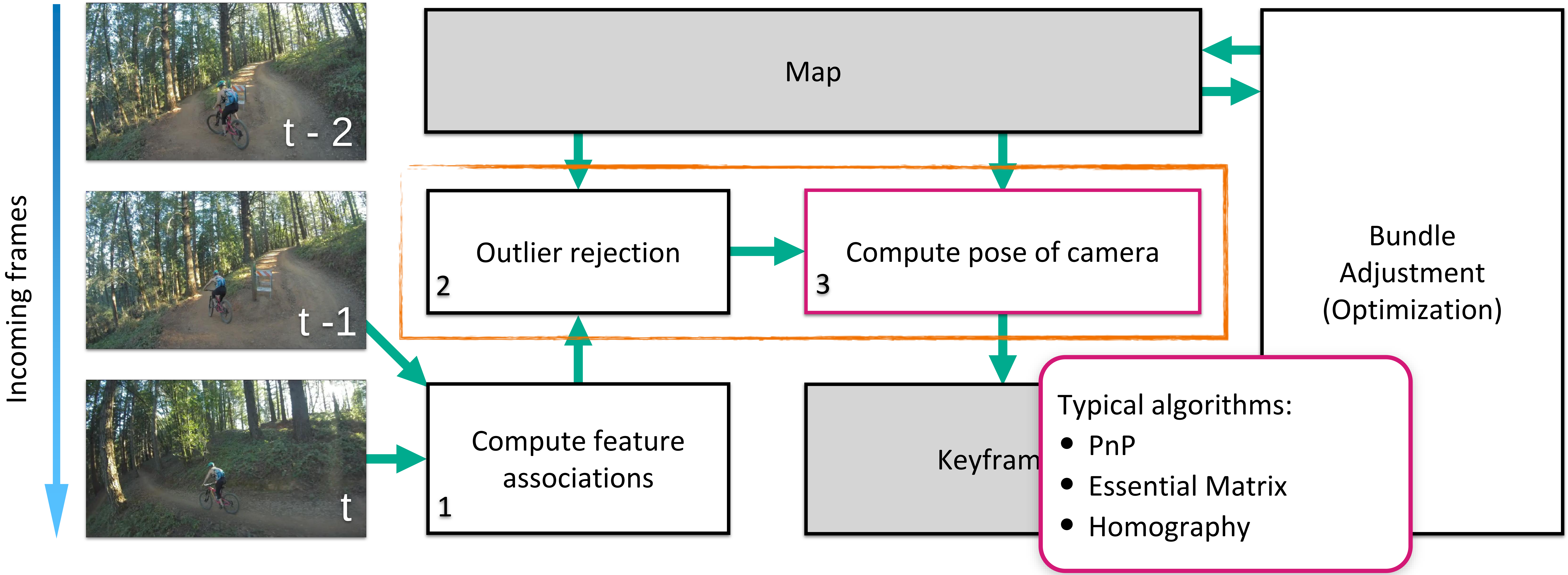


2 Pts
3 Pts
4 Pts
Iterations vs. Outlier Fraction
750
600
450
300
150
0
0%
10%
20%
30%
40%
50%
60%
70%
80%

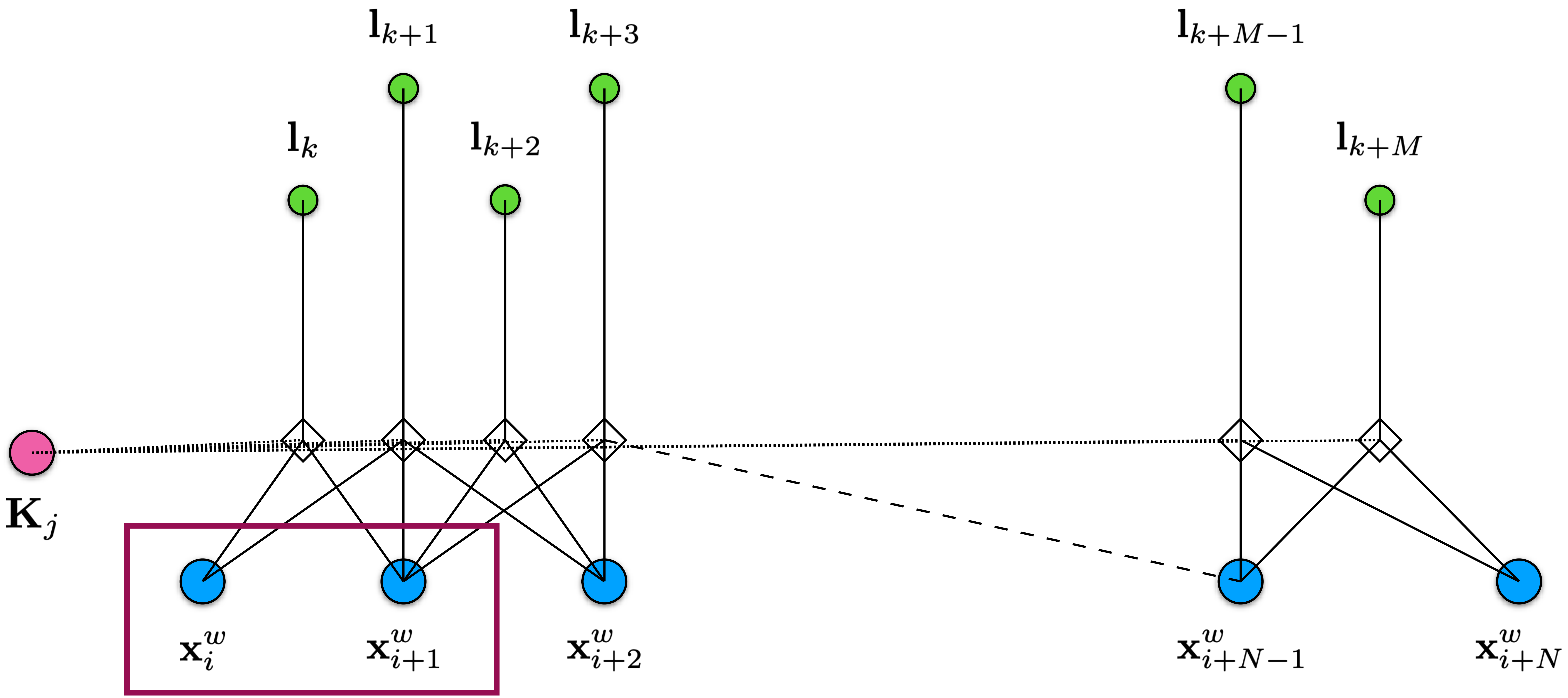
Typical SLAM Pipeline w/ BA



Typical SLAM Pipeline w/ BA



BA as a Factor Graph



Position, orientation

Landmarks

Camera calibrations

Bundle of rays

Skydio

BA as a SLAM Problem

- How do we actually recover the states, given the measurements and our model?

Camera poses, landmark positions, calibration parameters.

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_i \\ \vdots \\ z_N \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}, \mathbf{m}, \kappa) + \epsilon_{z_1} \\ h_2(\mathbf{x}, \mathbf{m}, \kappa) + \epsilon_{z_2} \\ \vdots \\ h_i(\mathbf{x}, \mathbf{m}, \kappa) + \epsilon_{z_i} \\ \vdots \\ h_N(\mathbf{x}, \mathbf{m}, \kappa) + \epsilon_{z_N} \end{bmatrix} \rightarrow \boxed{\quad ? \quad} \rightarrow \tilde{\mathbf{x}}, \tilde{\mathbf{m}}, \tilde{\kappa}$$

We need to fill in this box.

Feature tracks form a 'sensor' measurement.

Measurement model is given by the projective geometry of the problem.

Solving the Problem

- We can use a technique called *Nonlinear Least Squares* to do this.
- There are many ways to formulate SLAM problems generally, and we cannot review them all in the time allotted.
- *However*, this method is widely applicable, typically fast, and is straightforward to implement.
- For a much more comprehensive review, I highly recommend: *State Estimation for Robotics*, Tim Barfoot, 2015 (Free online)

Assumptions

- We will convert our measurement models into a system of equations.
- Prior to that, we will make an additional assumption - that the measurement noise is drawn from a zero-mean gaussian.

$$\epsilon_z \propto N\left(\mu = \mathbf{0}, \Sigma_z\right)$$

- We will also assume we have an *initial guess* for our states. In a time recursive system, this could come from the previous frame.

Nonlinear Least Squares

We re-write our measurements as a residual functions:

$$\mathbf{z}_{ij} = h\left(\mathbf{x}_i^w, \mathbf{l}_j, \mathbf{K}\right) + \epsilon_{z_{ij}} \longrightarrow \mathbf{f}_{ij} = h\left(\mathbf{x}_i^w, \mathbf{l}_j, \mathbf{K}\right) - \mathbf{z}_{ij}$$

The *i'th* camera pose, observing the *j'th* landmark.

And concatenate these into a large vector:

$$\mathbf{f}\left(\mathbf{x}^w, \mathbf{l}, \mathbf{K}\right) = \begin{bmatrix} \mathbf{f}_{ij} \\ \vdots \\ \mathbf{f}_{i+m,j+n} \\ \vdots \\ \mathbf{f}_{i+M,j+N} \end{bmatrix} \longrightarrow \boxed{||\mathbf{f}\left(\mathbf{x}^w, \mathbf{l}, \mathbf{K}\right)||_{\Sigma}^2 = \sum_i^M \sum_j^N ||\mathbf{f}_{ij}||_{\Sigma_{z_{ij}}}^2 \delta_{ij}}$$

We take the squared Mahalanobis norm, weighting by our assumed measurement uncertainty.

Nonlinear Least Squares

Our 'best estimate' will occur when the objective function is minimized:

$$\mathbf{y} := (\mathbf{x}^w, \mathbf{l}, \mathbf{K})$$
$$\tilde{\mathbf{y}} = \arg\min_{\mathbf{y}} ||\mathbf{f}\left(\mathbf{y}\right)||^2_{\Sigma}$$

Because f is usually going to be non-linear for most SLAM problems,

we end up *linearizing* the problem and taking a series of steps.

$$\mathbf{y}_{k+1} = \mathbf{y}_k + \delta y_k$$
$$\delta y_k = \arg\min_{\delta y} ||\mathbf{f}\left(\mathbf{y}_k\right) + \mathbf{J}\delta y||^2_{\Sigma}$$

Jacobian J is the linearization of f about our initial guess.

Nonlinear Least Squares

The solution at each iteration:

$$\delta \mathbf{y}_k = \left(\mathbf{J}^T \boldsymbol{\Sigma}_z^{-1} \mathbf{J}\right)^{-1} \mathbf{J}^T \boldsymbol{\Sigma}_z^{-1} \mathbf{f}\left(\mathbf{y}_k\right)$$

First order approximation of the Hessian. Inversion has complexity $O(|y|^3)$

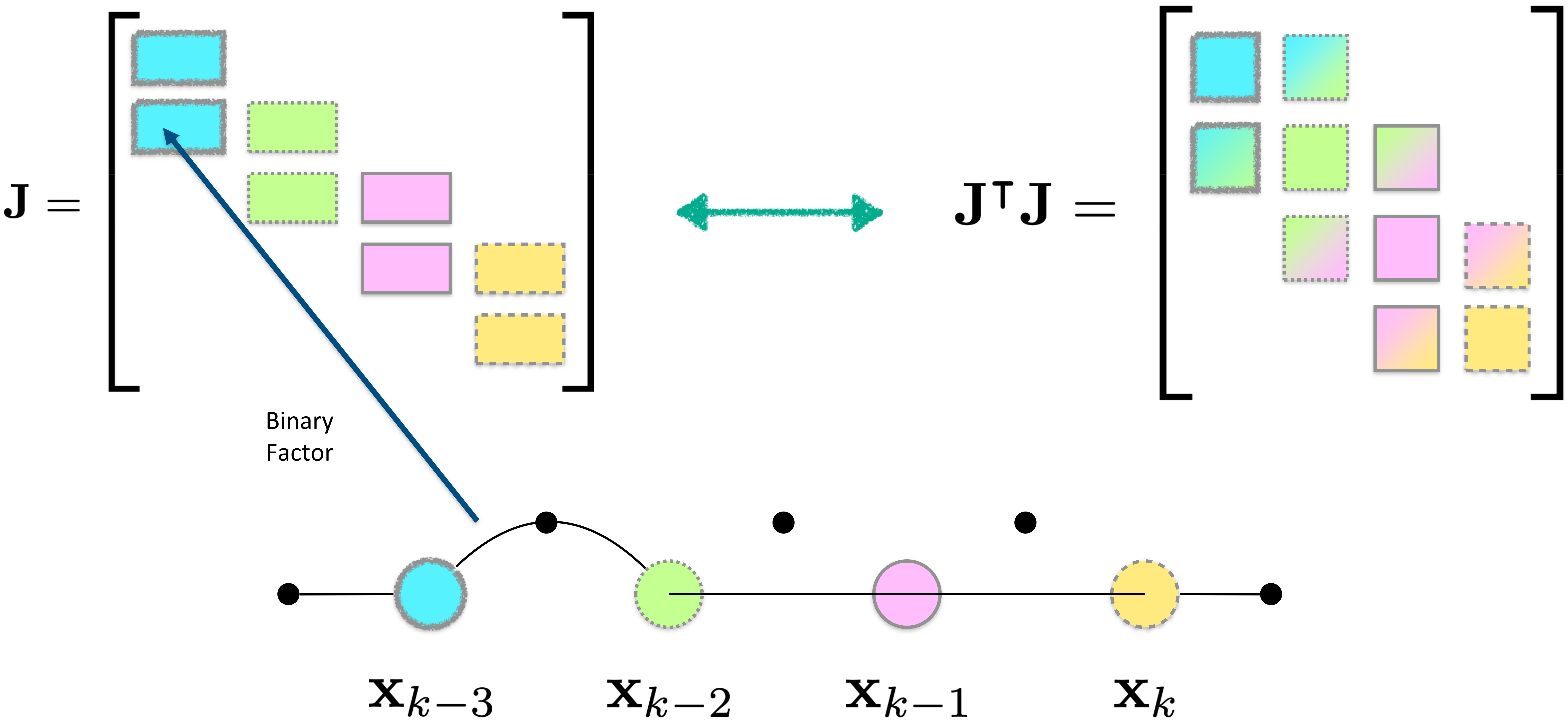
Each residual is weighted by its inverse uncertainty.

When linearized about the converged solution, the inverted Hessian doubles as a first order approximation of the *marginal covariance* of our estimate: *

$$\boldsymbol{\Sigma}_{\tilde{\mathbf{y}}} \approx \left(\mathbf{J}^T \boldsymbol{\Sigma}_z^{-1} \mathbf{J}\right)^{-1}$$

* See Barfoot, Chapters 3 and 4.

Sparsity



$\mathbf{J} =$
$\mathbf{J}^\intercal \mathbf{J} =$
Binary
Factor
$\mathbf{x}_{k-3}$
$\mathbf{x}_{k-2}$
$\mathbf{x}_{k-1}$
$\mathbf{x}_k$

Nonlinear Least Squares

- In the *linearized* form, the problem is 'easy' to solve.
  - Reduces to iterated application of *weighted least squares*.
  - Generally, cost of solving for updates is *cubic* in the number of states:
    - However, in some problems (like BA) there is sparsity we can leverage to improve this.
- Huge number of problems can be cast this way (given an initial guess).
- Can run in a fixed memory footprint → suitable for embedded use case.
- With the appropriate Σ weights we can show the NLS produces an *approximate* estimate of the uncertainty in our solution.

Caveats

- Remember our assumptions:
  - We needed an initial guess to linearize the system. If the guess is poor, the gradient used in the optimizer will steer our solution in the wrong direction.
  - Additionally, the covariance estimate we get out is only as good as the linearization point.
- We also assumed Gaussian noise on the measurements.
  - Outliers must be removed, or they will dominate the optimization.

Linearization

- It is worth considering the effect of linearization on our uncertainty estimate.
- For a Gaussian variable *u* and non-linear vector function *g*, we can approximate:

$$\mathbf{u} \propto N\left(\mu_{\mathbf{u}}, \boldsymbol{\Sigma}_{\mathbf{u}}\right)$$
$$\mathbf{v} = g\left(\mathbf{u}\right)$$

Because we linearized, the fidelity of our first-order Σ relies on this approximation.

$$\mathbf{v} \underset{\sim}{\propto} N\left(g\left(\mathbf{u}\right), \boxed{\frac{\delta g}{\delta \mathbf{u}} \boldsymbol{\Sigma}_{\mathbf{u}} \frac{\delta g}{\delta \mathbf{u}}^T}\right)$$

See Barfoot, Chapter 2.

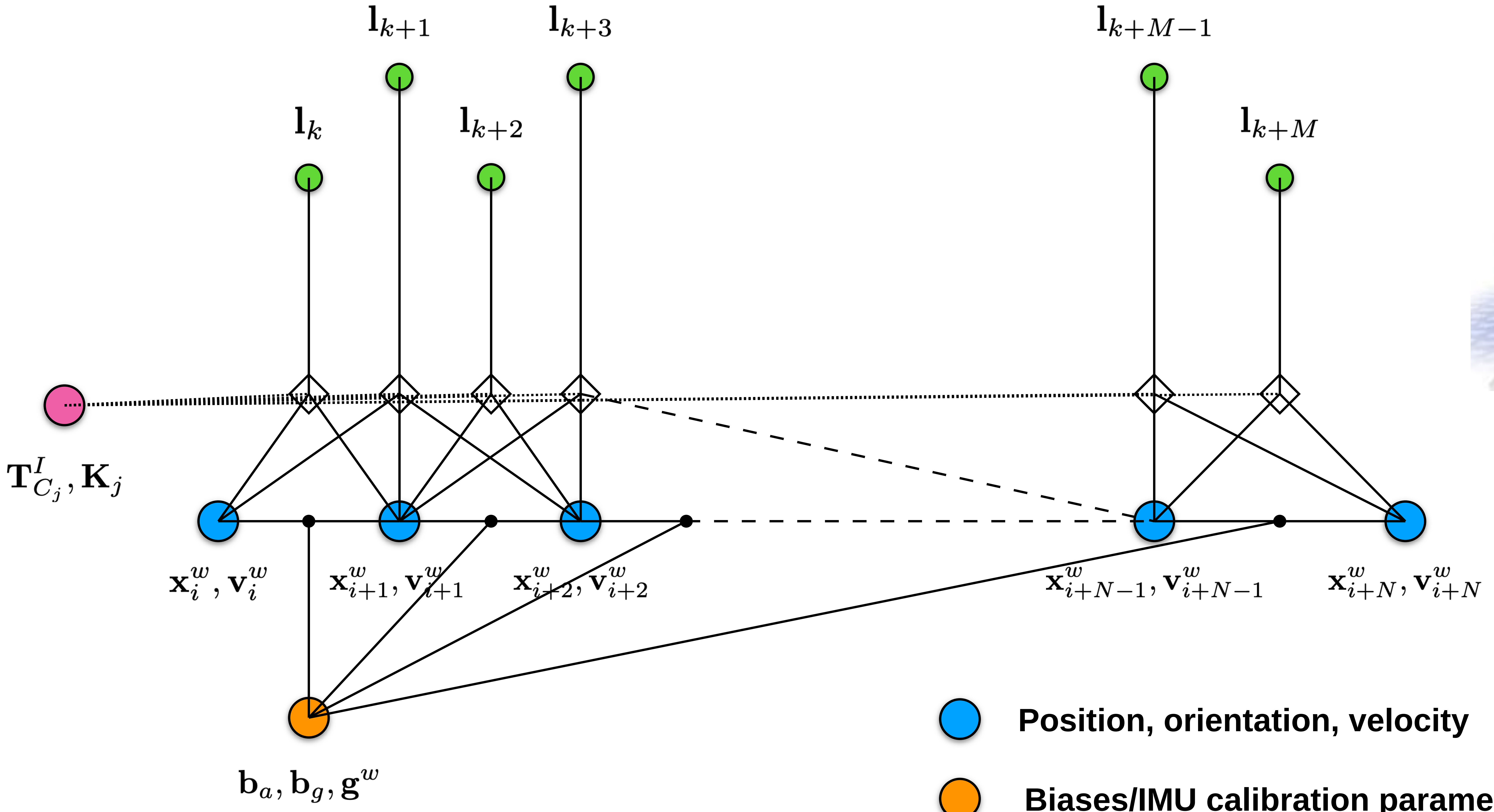
Tools

- Some relevant tools:
  - GTSAM, open source package created by Frank Dellaert et al.
    - Allows specification of problem in factor graph format, built for SLAM.
  - G2O
    - Includes solutions for SLAM and BA.
  - Ceres Solver, produced by Google
    - General non-linear least-squares optimizer.
  - Python
    - scipy.optimize.least_squares

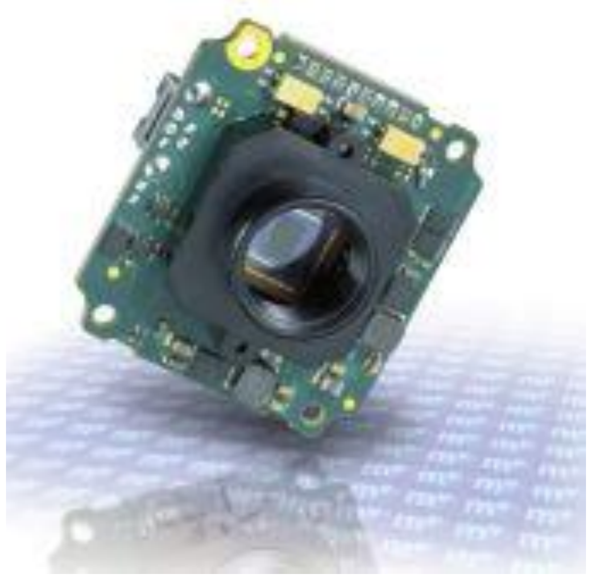
BA on Real-Time Systems

- BA can operate at small and large scale.
  - Small: A few image frames on a mobile phone.
  - Large: Tens of thousand of images at city-scale.
- Fairly straightforward to implement.
- But:
  - Robust association *may* require expensive descriptors.
  - After feature association, we must devote nontrivial compute to outlier rejection.
  - Update rate limited to camera frame rate (slow).

VIO as a Factor Graph: BA + IMU



Cameras



Source: MatrixVision

Inertial Measurement Unit

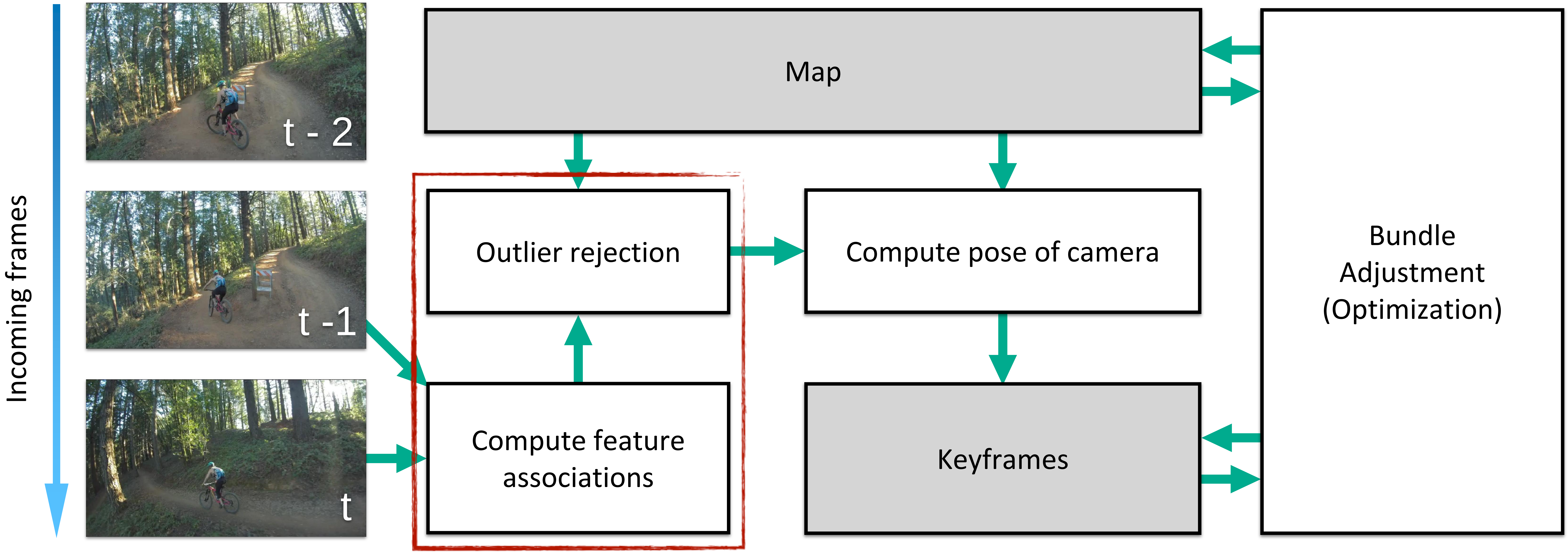
Source: Lord MicroStrain

- Position, orientation, velocity
- Landmarks
- Biases/IMU calibration parameters
- Bundle of rays
- Camera calibrations
- IMU, motion model

VIO

- One of the most successful adaptations of vision research to the market.
  - Present in smart phones, AR/VR headsets, drones, autonomous vehicles.
- Camera and IMU are highly complementary:
  - Camera:
    - Low update rate, high compute cost, subject to outlier data.
    - Able to relocalize accurately at large distances.
  - IMU:
    - High update rate, low compute cost, few outliers (maybe saturation).
    - Accurate over short intervals, but drifts over time.
    - Able to recover attitude with respect to global reference frame (gravity).

VIO



IMU can deliver substantial value here.

BA/VIO Implementations

- Existing open-source implementations (not exhaustive):
  - OpenMVG
  - COLMAP Offline SFM and Multi-view Stereo (MVS)
  - CMVS Multi-view Stereo
  - ORB-SLAM2 Real-time SLAM featuring BA optimization
  - PTAM One of the earliest functional visual-SLAM demos
  - VINS-Mono VIO, runs on a mobile device
  - Basalt VIO
  - ROVIO VIO, example of a *direct method*

Fin

- Additional Reading:
  - *State Estimation for Robotics* (Barfoot, 2015)
  - *Factor Graphs for Robot Perception* (Dellaert and Kaess, 2017)
  - *Visual Odometry*, (Scaramuzza and Fraundorfer, 2011)
  - *Probabilistic Robotics*, (Thrun, Burgard, and Fox, 2005)
  - GTSAM Software Library

Questions? Feel free to reach out: gareth@skydio.com