Probabilistic Pose Estimation from Image Features

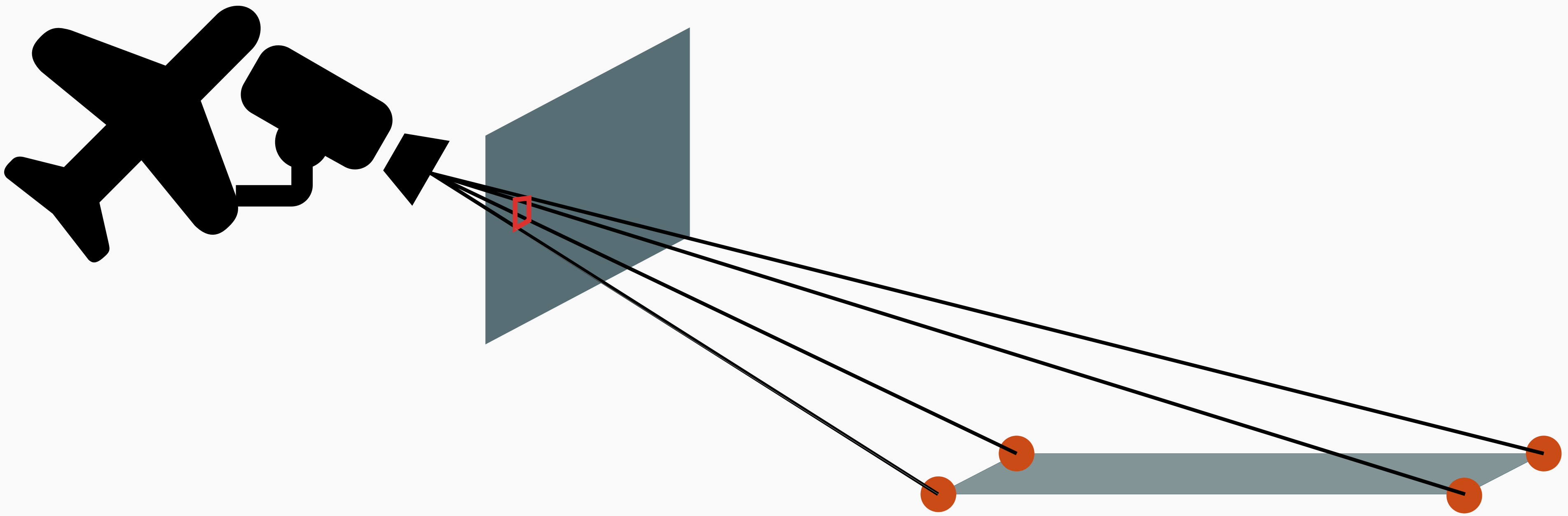
Feeding the ~~beast~~ Kalman filter with well-calibrated estimates

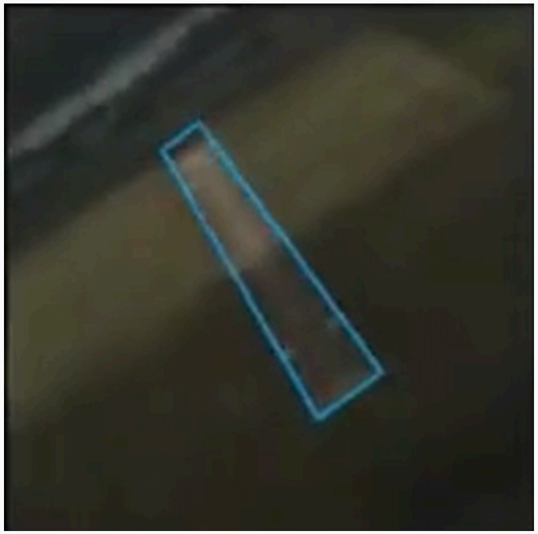
Romeo Valentin (romeov@stanford.edu)

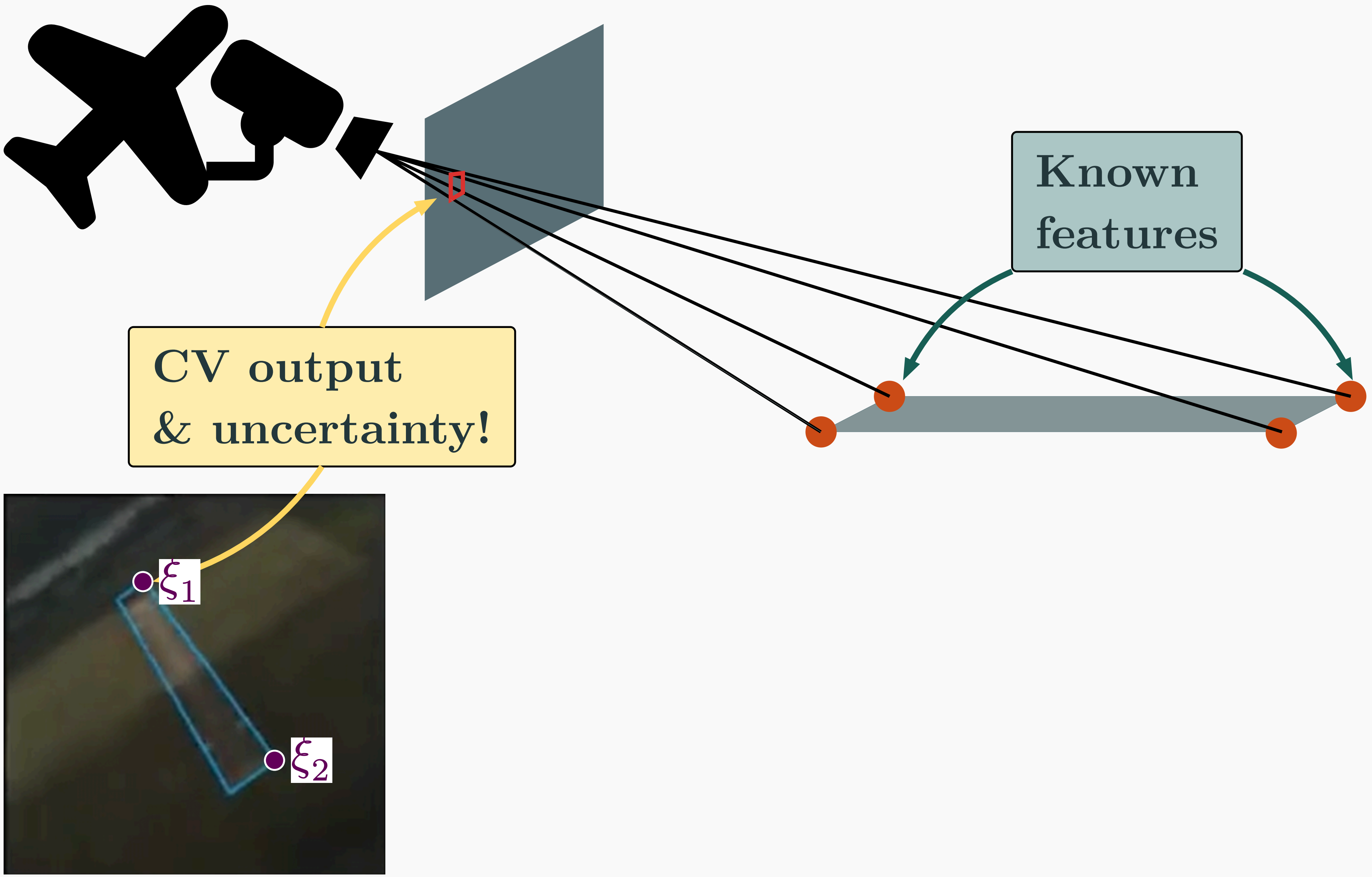
September 2024 • DASC

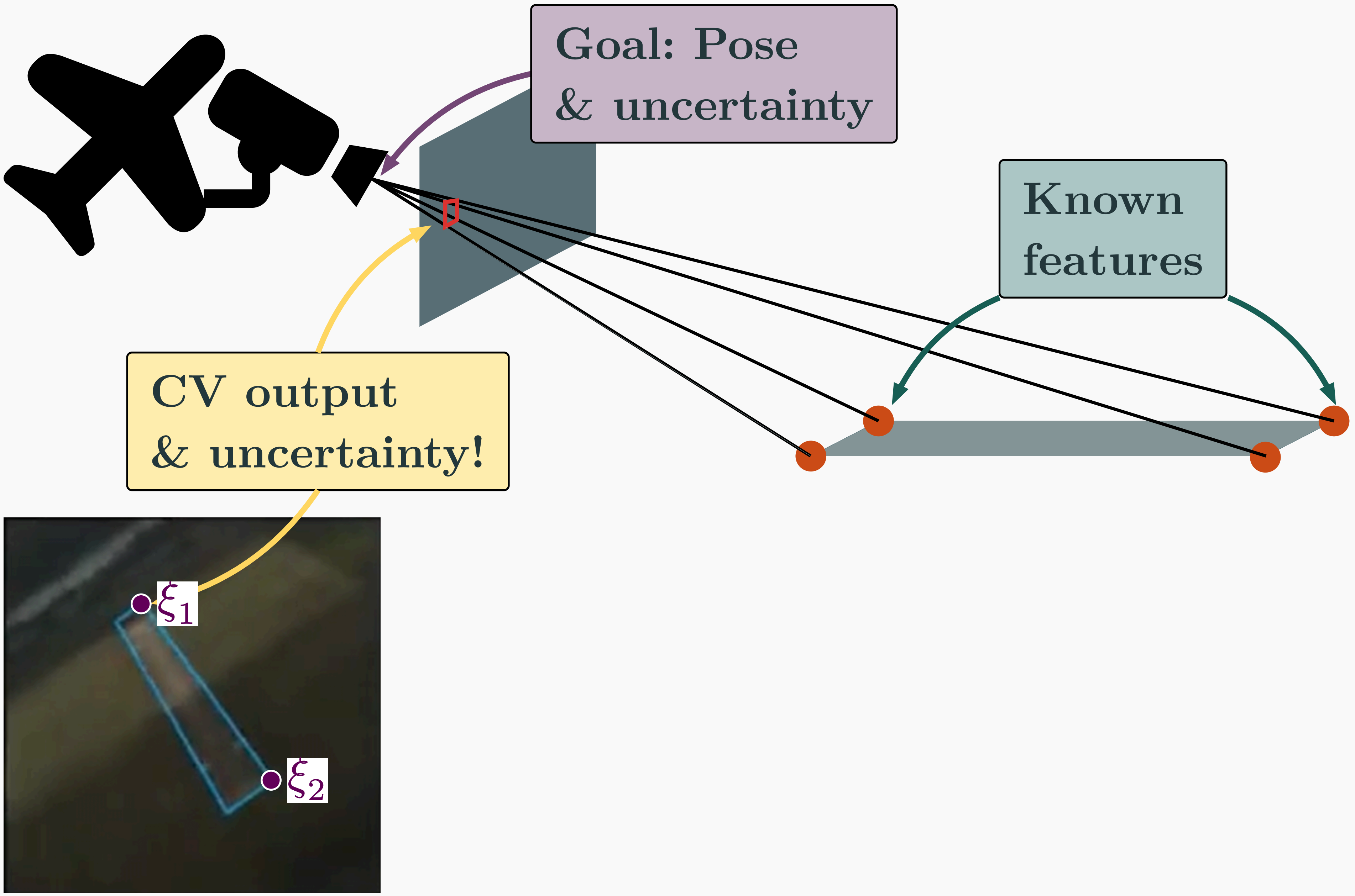
Collaborators:

- Sydney Katz
- Joon Lee
- Mykel Kochenderfer
- Don Walker
- Matthew Sorgenfrei



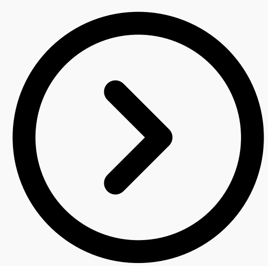



Known features
CV output & uncertainty!
ξ_1
ξ_2

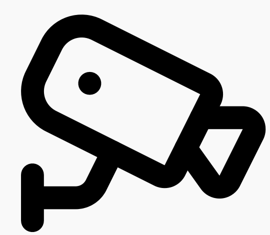

Goal: Pose
& uncertainty
Known
features
CV output
& uncertainty!
ξ_1
ξ_2

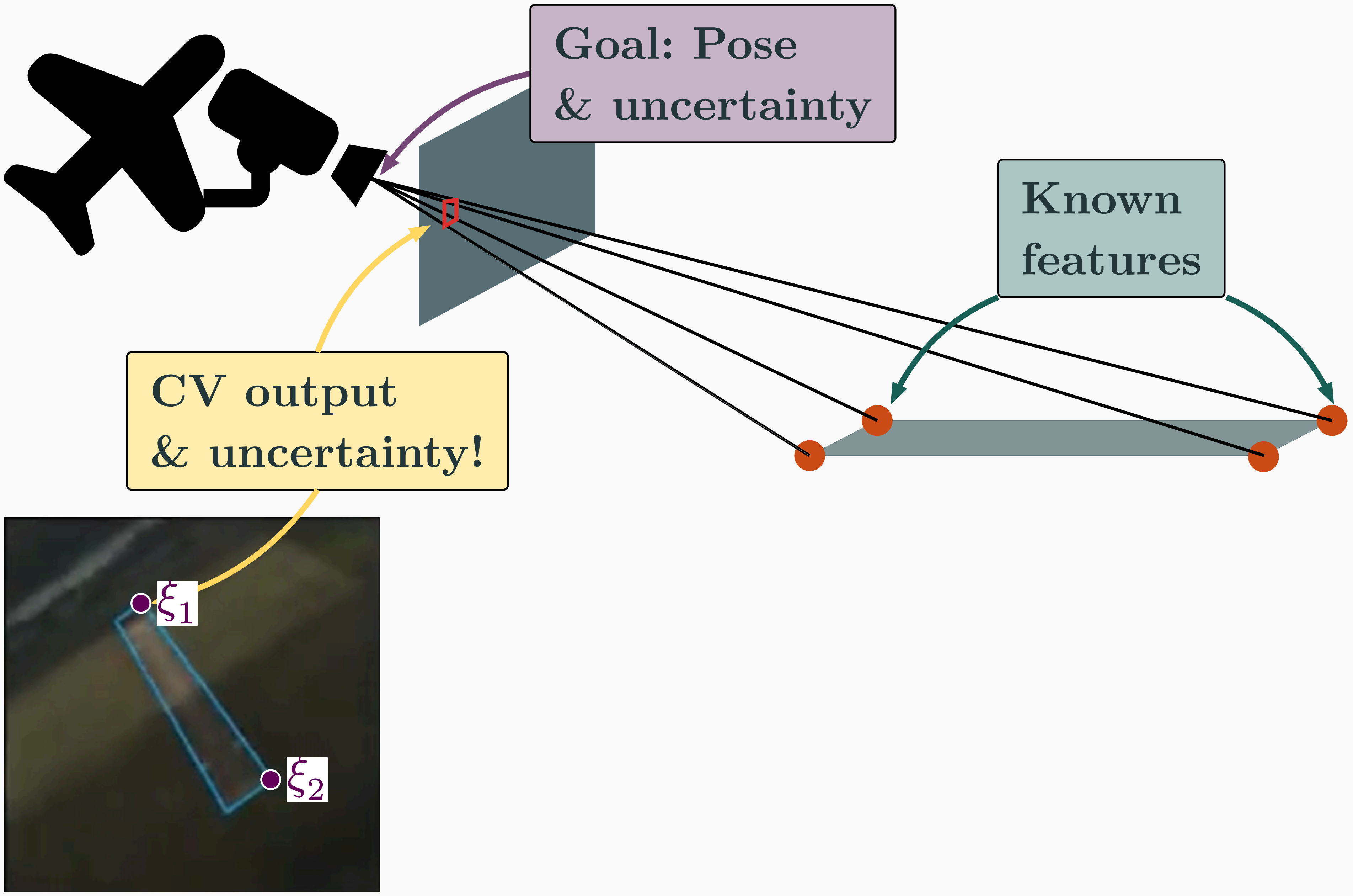
GPS:

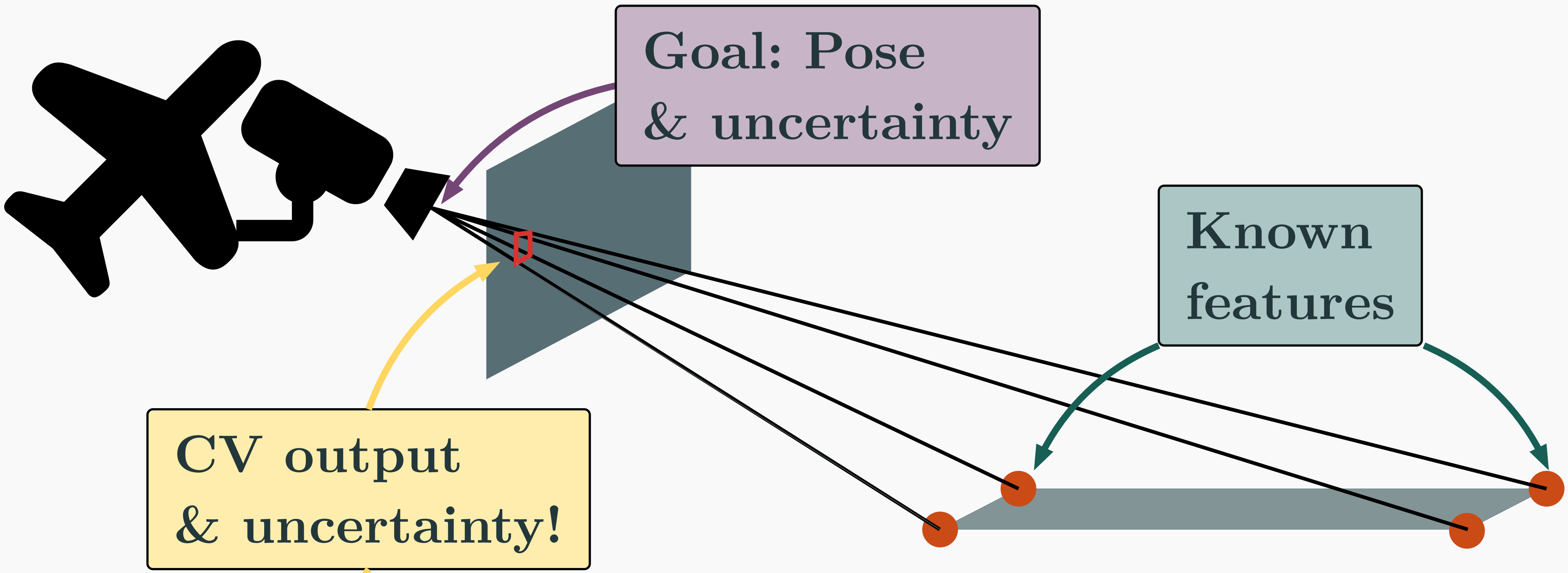
IMU:

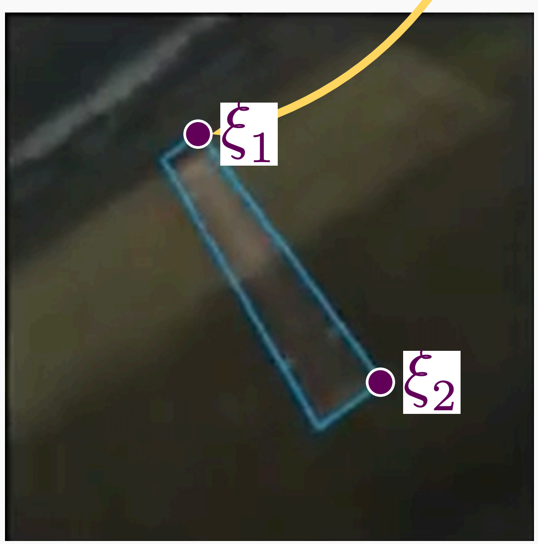


Camera?




Goal: Pose
& uncertainty
Known
features
CV output
& uncertainty!
ξ_1
ξ_2



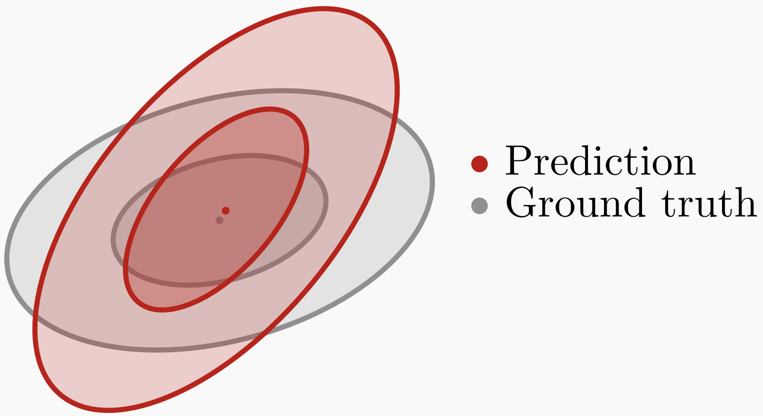


$$\text{pose } \boldsymbol{\beta} \to \text{pose } p(\boldsymbol{\beta} \mid \ldots)$$

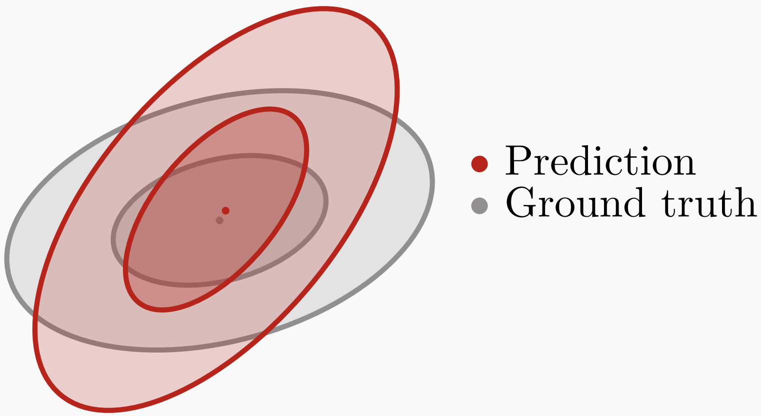
$$\text{eval } \boldsymbol{\beta} \to \text{eval } p(\boldsymbol{\beta} \mid \ldots)$$

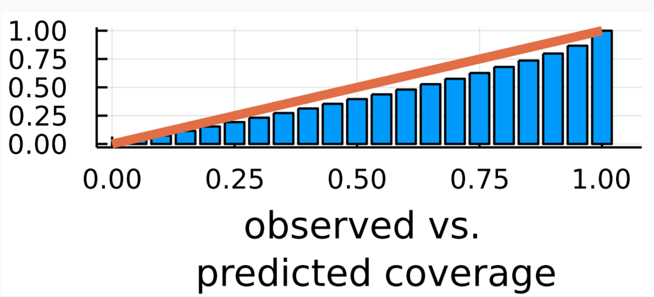
What makes a good probability prediction?

Calibration curve:

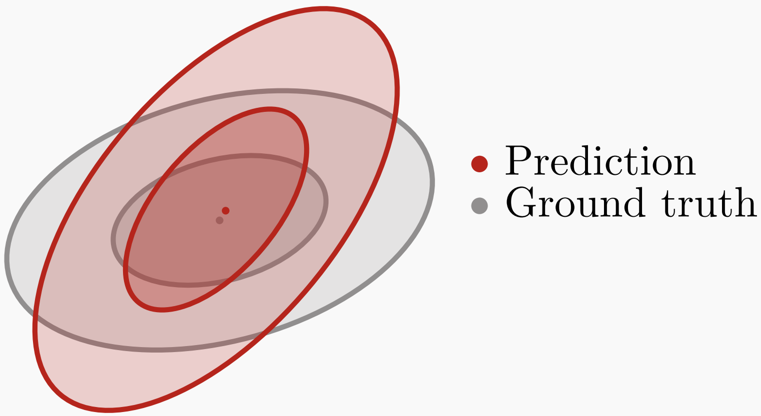


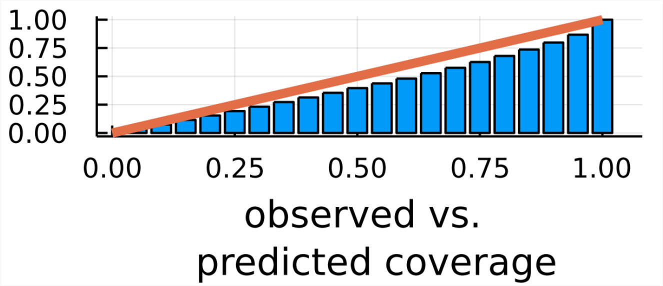
Calibration curve:



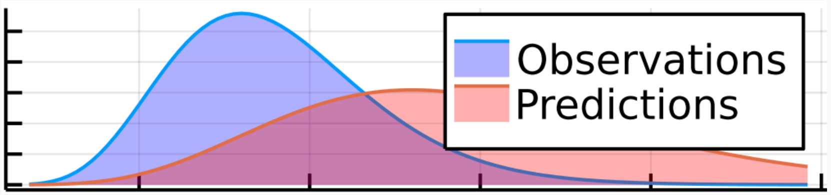


Calibration curve:

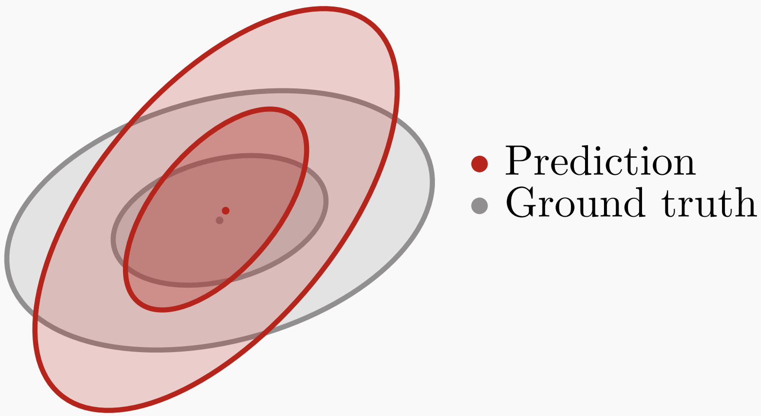


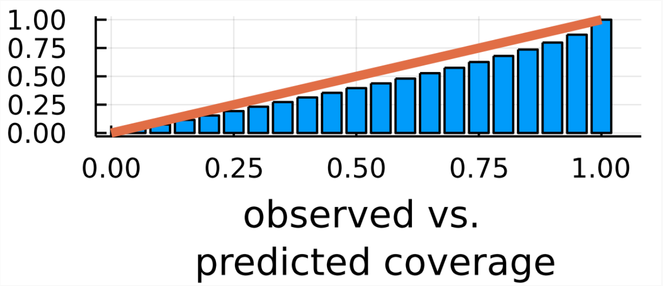


Sharpness:

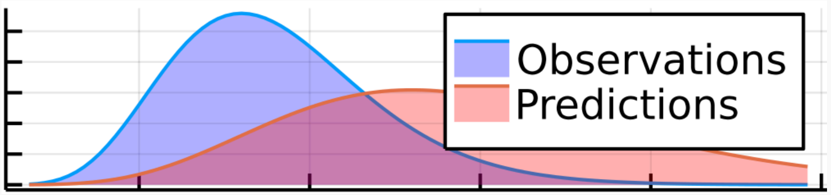


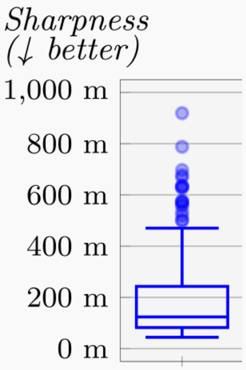
Calibration curve:





Sharpness:





Three Probabilistic Estimators

`LSQEstimator`

Sample noise, repeatedly solve least squares problem

$$\boldsymbol{\beta} = \arg\min_{\theta} \sum w_i \cdot \left(\mathrm{proj}_{\boldsymbol{\beta}}(x_i) - y_i\right)^2.$$

`LinearApproxEstimator`

Assume everything Gaussian & linear

$$\mu_\beta = \arg\min_{\beta} \boldsymbol{r}(\beta)^T \boldsymbol{r}(\beta)$$

$$\Sigma_\beta = \left(J^T \Sigma_\varepsilon^{-1} J\right)^{-1} J^T \Sigma_\varepsilon^{-1} J \left(J^T \Sigma_\varepsilon^{-1} J\right)^{-1}.$$

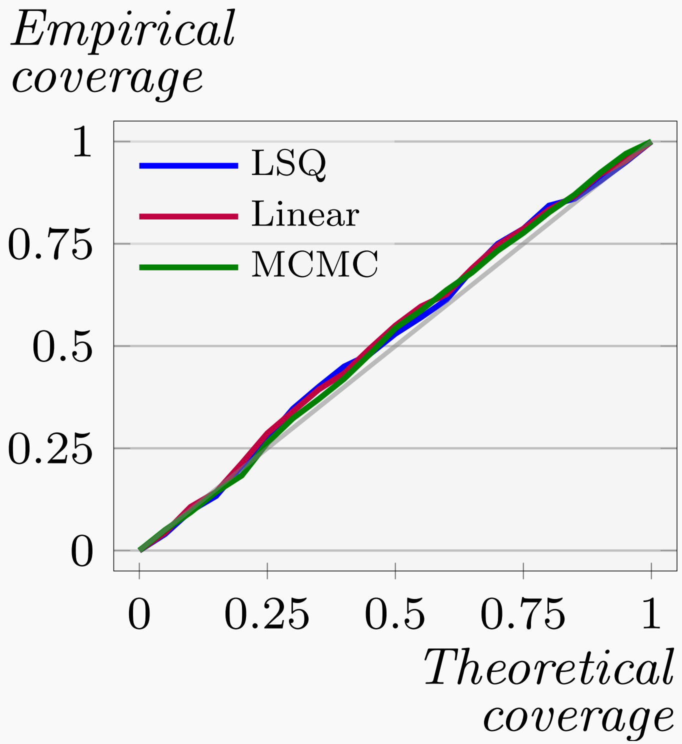
`MCMCEstimator`

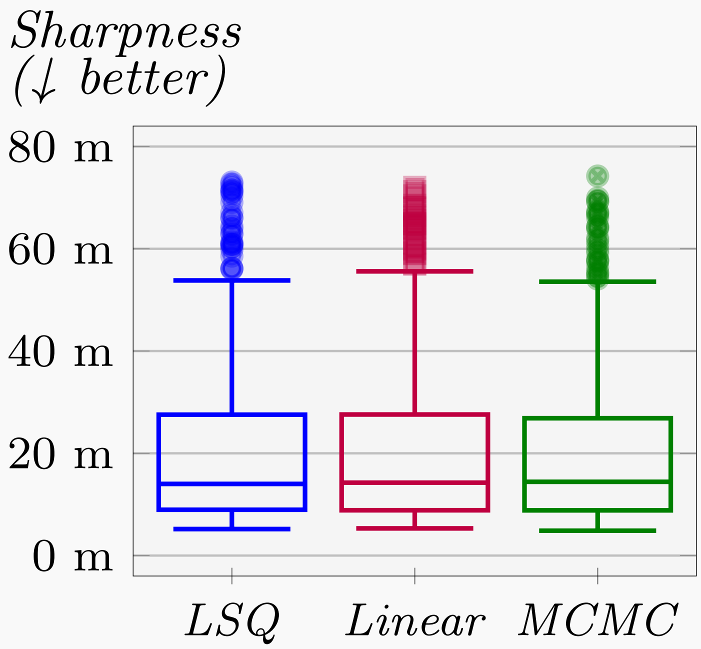
Automagically sample directly from pose distribution

$$\beta \sim p(\beta \mid \left\{\mathrm{proj}_{\boldsymbol{\beta}}(x_i), y_i\right\}_i).$$

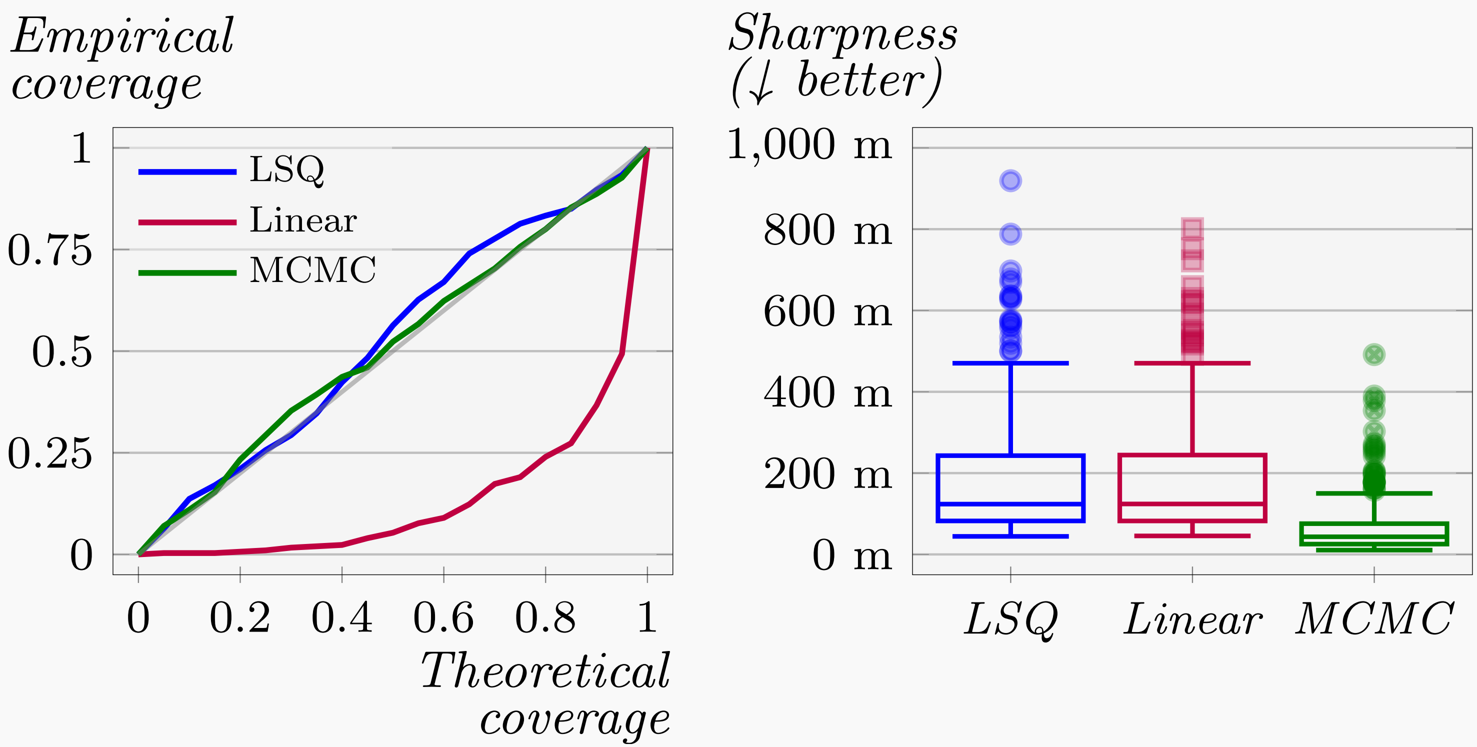
Putting it together!

Uncorrelated normal noise:

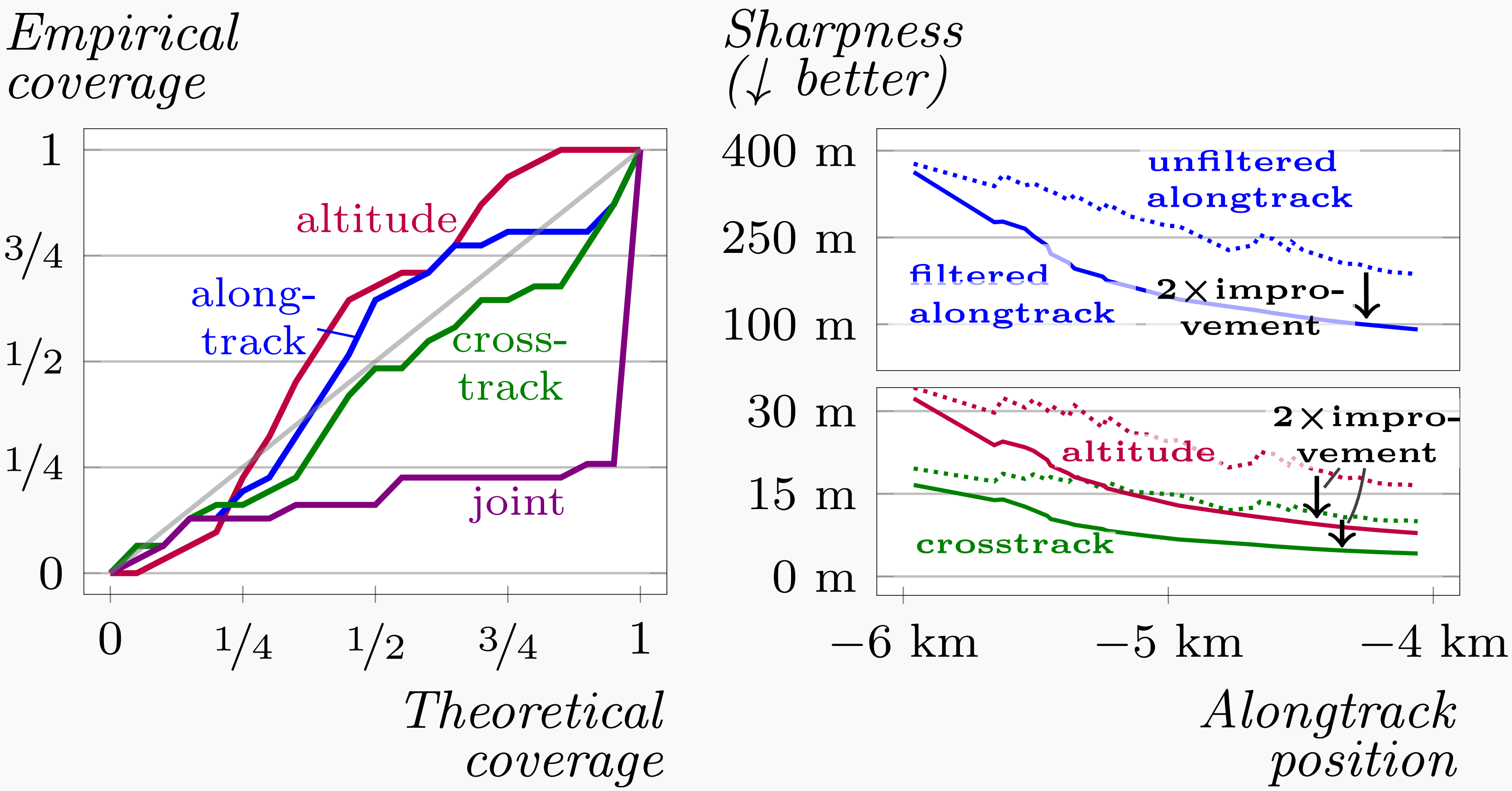




Long tail noise:



Results after Kalman filter:



Runtime costs:

Noise sampling	311 ms
Lin. Approx	0.4 ms
MCMC	183 ms

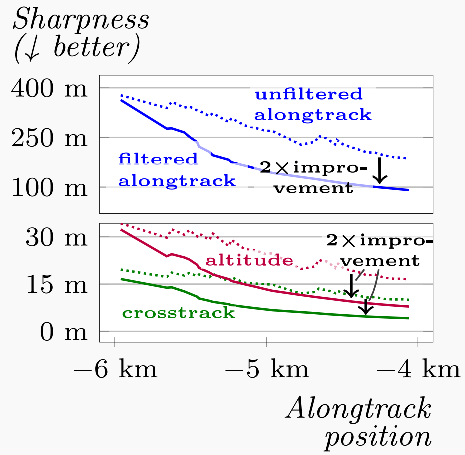
ProbabilisticParameterEstimators.jl

```julia
using ProbabilisticParameterEstimators, RunwayLib, GeodesyXYZExt
runway_corners = [
  XYZ(0.0m, -25m, 0m), XYZ(0.0m, 25m, 0m),
  XYZ(3000.0m, -25m, 0), XYZ(3000.0m, 25m, 0m),
]
projection_measurements = mymodel(...)
paramprior = MvNormal(...); noisemodel=UncorrGaussianNoiseModel(...)

pose_dist = predictdist(LSQEstimator(), projection_fn,
    runway_corners, projection_measurements, paramprior, noisemodel)
```

Implements three estimators:

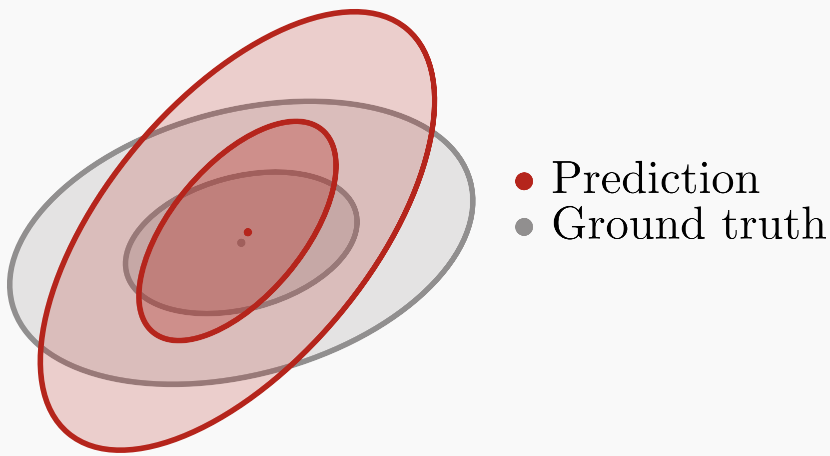
- `LSQEstimator`
- `LinearApproxEstimator`
- `MCMCEstimator`

→ read more in the paper.



Experimental C library

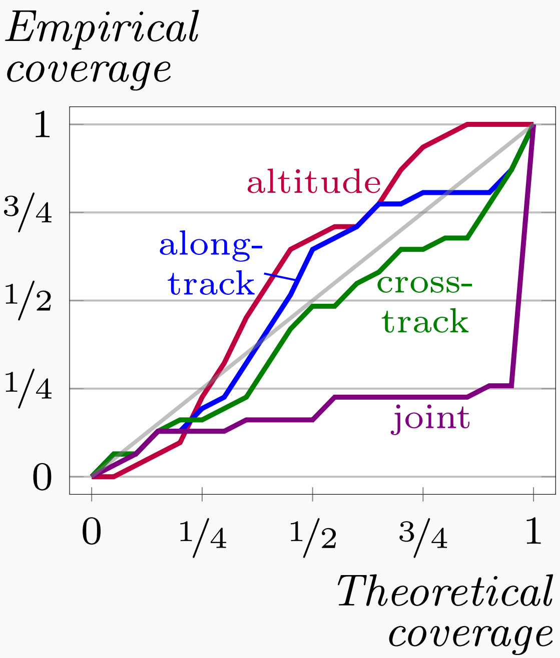
```c
#include <stdio.h>
#include <stdlib.h>

// Julia headers (for initialization and gc commands)
#include "julia_init.h"
#include "runwaypnpsolve.h"

int main(int argc, char *argv[])
{
    init_julia(argc, argv);

    int ret;
    double dst_pos[3] = {0., 0., 0.};
    double dst_cov[9] = {0., 0., 0., 0., 0., 0., 0., 0., 0.};


    double rwylength = 3500.0;  // m
    double rwywidth = 61.0;  // m
    double rwycorners[3*4] = { /* insert runway geometry */ };

    double measuredprojs[2*4] = { /* get from sensor */ };

    int n_rwycorners = 4;
    ret = predict_pose_c_interface(dst_pos, dst_cov,
                                   rwycorners, n_rwycorners,
                                   measuredprojs);
}
```

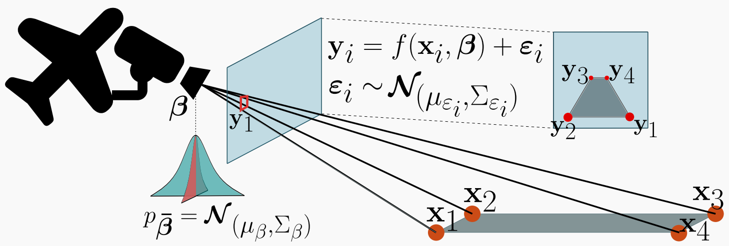


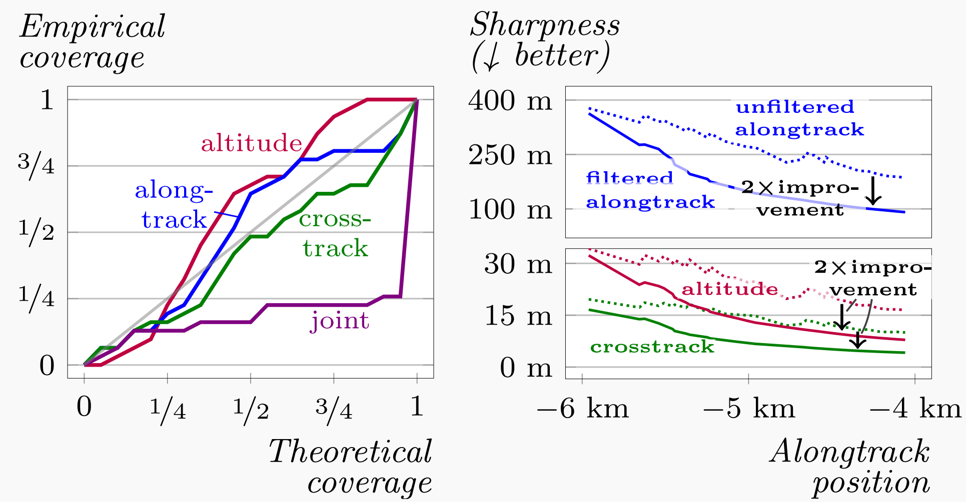
How can we quantify "quality" of predictions?
→ read more in the paper.

Usage Example (Julia):

```
using MvNormalCalibration
preds = [predictdist(...) for t in timesteps]
truevals = # ...

est = computecalibration(preds, truevals)
plot(est...)
```







Goal: Estimate $p(\beta \mid \ldots)$ from probabilistic measurements.

Three estimators:

i) `LSQEstimator`

ii) `LinearApproxEstimator`

iii) `MCMCEstimator`

& different **noise models**.

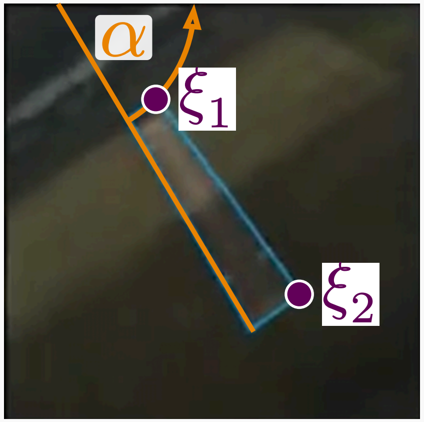
Calibration & filtering:

- New calibration metric for $\mathcal{N}(\mu, \Sigma)$
- Demonstrate calibration after filtering.
- Filtering despite bias (WIP)

What is the new approach?

Consider **probabilistic inputs / outputs**:

$\alpha \in \mathbb{R}$ $\xi \in \mathbb{R}^2$	$\longrightarrow$ $\longrightarrow$	$\alpha \in \mathbb{P}(\mathbb{R})$ $\xi \in \mathbb{P}(\mathbb{R}^2)$	e.g., $\mathcal{N}(\mu_\alpha, \sigma_\alpha^2)$ e.g., $\mathcal{N}(\mu_\xi, \Sigma_\xi)$
$\beta \in \mathbb{R}^6$	$\longrightarrow$	$\beta \in \mathbb{P}(\mathbb{R}^6)$	
$\arg\min_\beta \sum_i (\cdot)_i^2$ metric $:= \lVert \cdot \rVert_2$	$\longrightarrow$ $\longrightarrow$	$p(\beta \mid \alpha, \xi_1, \xi_2, \ldots)$ metric $:= \begin{pmatrix} \text{calibration} \\ \text{\& sharpness} \end{pmatrix}$	e.g., $\mathcal{N}(\mu_\beta, \Sigma_\beta)$



Three estimators:

i) Noise Sampling + least squares
ii) Linear Approximation for Σ_β
iii) Markov chain Monte Carlo

The `RunwayPNPSolve.jl` ecosystem

Paper:

See resources – soon `arxiv`, then DASC.

Registered:

`ProbabilisticPoseEstimators.jl`

`MvNormalCalibration.jl`

Unregistered:

`GeodesyXYZExt.jl`

`RunwayLib.jl`

`RunwayPNPSolve.jl`

`RunwayPNPSolveBinary.jl` (unpublished)

`GeodesyXYZExt.jl` (unregistered)

Experimental feature: strongly typed unitful type restrictions.

Usage Example (Julia):

```
using GeodesyXYZExt
camera_pos = XYZ(-2500.0m, 20m, 300m)
camera_rot = RotXYZ(roll=10.0°, pitch=3°, yaw=-1.5°)

function project(pos::XYZ{<:WithDims(m)}, rot::Rot3, world_pt::LLA)
  # ...
  return ImgProj(...)
end
```

Restrict:

- Coordinate system
- Unit

But flexible with:

- Number type, such as `Float64` or `Dual{Float64}`

RunwayLib.jl

Some utility functions, such as point and line projections.

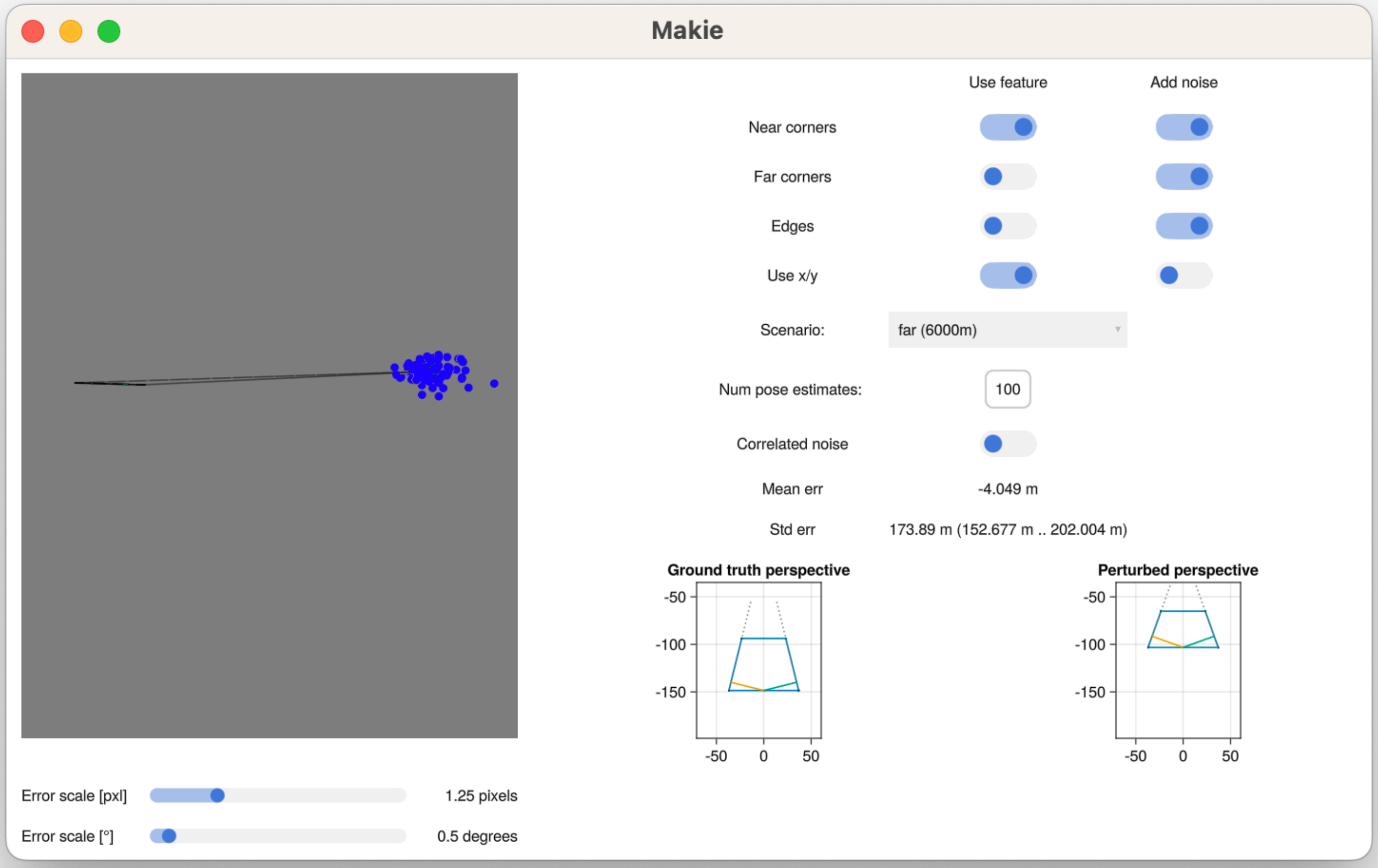
Usage Example (Julia):

```
cam_pose = CamTransform(camera_rot, cam_pos)
world_pt = XYZ(300.0m, 20.0m, 0.0m)

projection::ImgProj = project(cam_pose, world_pt)
rho_theta = project_line(camera_pose, ImgProj(0.0pxl, 0.0pxl),
                         (world_pt, world_pt + XYZ(100.0m, 0m, 0m)))
```

RunwayPNPSolveBinary.jl (unpublished)

- binary generation using PackageCompiler.jl
- can run without Julia installation, e.g. onboard
- 0.5ms to 250ms for one parameter estimate
  - fast enough for real-time!

Experimental visualizations with `Makie.jl`



The RunwayPNPSolve.jl ecosystem.

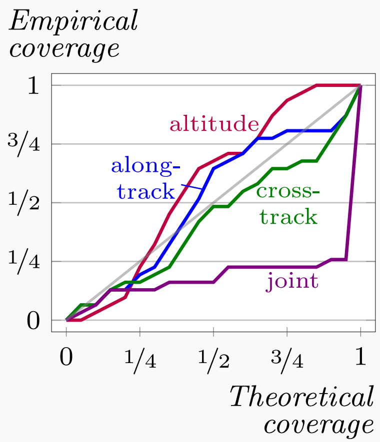
Packages:

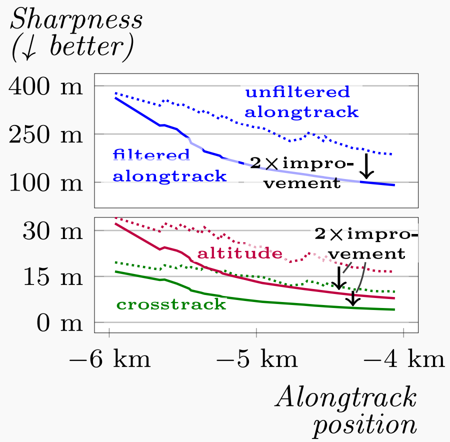
- ProbabilisticPoseEstimators.jl
- MvNormalCalibration.jl
- GeodesyXYZExt.jl
- RunwayLib.jl
- RunwayPNPSolveBinary.jl

Acknowledgements:

- NonlinearSolve.jl and the SciML ecosystem
- Makie.jl ecosystem
- CoordinateTransformations.jl, Rotations.jl, and Geodesy.jl
- Unitful.jl and UnitfulAngles.jl

→ Check out the paper in the talk resources to learn more about these figures!





... and my co-authors

- Sydney Katz, Joon Lee, Mykel Kochenderfer
- Don Walker, Matt Sorgenfrei (A^3 by Airbus)

Also, come see my poster on ThreadedDenseSparseMul.jl at 7pm in Method (1.5)!

Why camera based?

We already have

- GPS
- IMU / INS
- radio-based (VOR)

so why **camera-based**?

$\Rightarrow$ **independent failure mode**