Pose Estimation Simulation Study and Sensitivity Analysis: Beyond Simple Point Associations and PnP

Romeo Valentin (romeov@stanford.edu)

Summer 2023

Abstract

This repository contains the libraries, experiments, and reports developed during the 2023 Summer internship by Romeo Valentin (romeov@stanford.edu or romeo.valentin.int@airbus-sv.com). This report summarizes the findings made during my 2023 Summer internship. We investigate the VNV pipeline wrt to the question whether the current runway representation is sensible and leads to numerically stable position estimation results. We use a simulation study as well as theoretical analysis to discuss the merits and drawbacks of the current representation. In summary, we find that if our pixel prediction error does not exceed a standard deviation of 1 pixel, we are generally within spec, and are robust to attitude changes and small amounts of noise in the attitude estimation. We provide evidence that currently the system does stay within this 1 pixel limit the majority of the time; however it remains to be seen whether this performance also holds in more difficult conditions like night scenarios or given occlusion.

1 Introduction

The goal of the vision pipeline is to provide a pose estimate of the airplane when approaching a runway. To this end, the system must first try to find the runway, which may only occupy a small fraction of the total image. Then, the system must extract useful image features of the runway, which may be matched up with the runway width, length, etc retrieved from a database. Finally, the image features are combined to produce a pose estimate at each frame, which is then fused together with other sensor and predictive information.

In this report we will focus on the second and third step of this process. When designing these system, many design decisions come up, including how to design a computer vision pipeline to extract image features, what image features are useful, and how to robustify the system against small errors.

The code and experiments are all provided at `https://github.com/airbus-wayfinder/PNPSolve.jl`.

1.1 Overview of the current approach

First, let's briefly review how the current implementation functions.

In order to simplify the problem and reduce the solution space, we assume that the orientation (attitude) of the airplane is well-known – indeed for the current analysis it is assumed that the orientation estimate is error-free.[1] Then, stage two of the system takes an image crop containing the runway, and predicts a pair of pixel indices (i_x, i_y) for the two near runway corners (specifically, the ends of the runway threshold). These indices are converted to spatial coordinates and matched up with the projections of known runway corner locations. The position estimate is then computed by using the OpenCV PnP implementation, which essentially solves a quadratic optimization problem minimizing the reprojection error of the known corner locations compared against the measured coordinates. It is solved iteratively with the Levenberg-Marquardt algorithm.

1.1.1 Service volume

In general, the PnP problem has many possible solutions if only a few correspondence points are given. Additionally, it may become ill-posed when the perspective becomes "extreme", e.g. when the camera is very close to the runway. However, the MPVS specifies a "service volume" which restricts the space in which the algorithm has to function. Specifically, the service volume is currently defined as

- alongtrack distance between 300 m and 6000 m,
- crosstrack angle between $-3°$ and $3°$, and
- vertical angle between $1.2°$ and $6°$.

[1] The question how to integrate uncertainty or errors in the orientation is an interesting one but will not be discussed in this report.

1.2 Contributions of this work

Given the described approach, several questions arise naturally, which may include

- How does our pose estimate change in the presence of prediction errors? How does this relate to the "allowed" pose estimate errors?
- What is the distribution of the "real" prediction errors? Are we already "precise enough"? How much room for improvment is there?
- Can we reduce the impact of prediction errors by using different or more features?
- Can we improve our estimate by using information from multiple runways?
- Can we improve on the OpenCV PNP formulation?

Throughout this report we will give insights to all of these questions and will develop a mathematically well-founded approach to solving the pose-estimation problem in a robust way.

2 Measurement Error Distribution and Correlations

In order to estimate the effects of measurement errors on the downstream performance, we first want to analyze how large the measurement errors are in practice. In theory, this is straightforward – we run the ML model on a bunch of samples, and compare the ML predictions with the labels.

Unfortunately, the results look very poor upon first inspection, with a standard error of about 10 pixels for the various predictions. However, we get significantly better results when comparing the ML predictions only to the hand-labeled labels.

The procedure is therefore as follows:

1. We collect the most recent VNV results;
2. we filter only for samples that have hand-labeled labels;
3. we filter for only samples that are in the service volume specified by the MPVS. We may optionally filter for "extreme" samples too (see below).

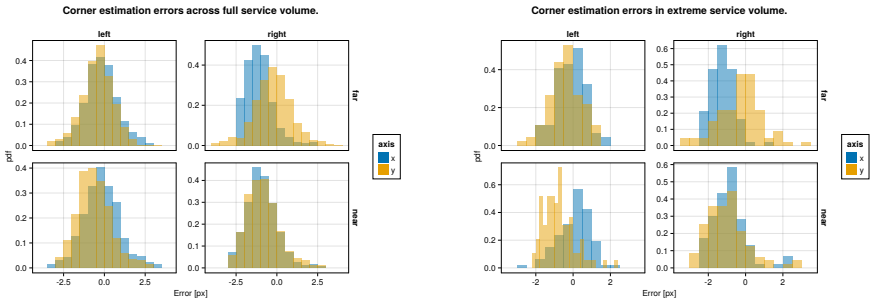


Figure 1: Empirical error distributions of prediction location of the different runway corners.

4. we plot the distributions and correlations of prediction errors (in pixel space).

We define the "extreme" service volume as follows:

```
in_extreme_service_volume(row) =
  row.gt_translation_x in (-6000m .. -5500m) ||
  abs(row.horizontal_angle) in 2°..3° ||
  row.vertical_angle in 1.2°..2.0°
```

Filtering for hand-labeled data reduces us to about 5\Further filtering for the service volume leaves us with 1332 samples. Finally, further filtering for the extreme service volume (where "extreme" only needs to be satisfied by one property at a time) leaves us with only 105 samples. Keep this in mind when interpreting the following results.

Fig. 1 shows the distribution of measurement errors for the four runway corners, in x and y direction respectively. We can see that the error distribution is relatively small, and can be decently approximated by a standard Normal distribution. We note however that it will be important to generate similar plots for data that has worse conditions, e.g. in the night, cloudy weather, etc.

2.1 Correlations

We can also wonder if the errors are correlated. For example, when we are predicting the near left corner too far to the right, do we also predict the far left corner too far to the right? What about relations between left-right and up-down? In Fig. 2 we see that there is indeed a strong correlation between

all for corners, such that all x predictions are correlated, and all y predictions are correlated; however x and y do not seem to be correlated.

The full correlation matrix and the correlation plot for data on the extremes of the service volume are provided in Section 6.2.

2.2 Conclusion

For the further studies, we will proceed with the assumption that errors are sampled from a zero-mean Gaussian with one pixel of standard deviation, which seems approximately justified. In general we will consider the uncorrelated case (although somewhat misspecified given the above results), however we will also briefly consider the correlated case.

We do note, however, that a Gaussian distribution may underestimate the "heavy tails". In other words, the following results may be overly optimistic.

3 Simulation study

In this section we will introduce details and results of our empirical simulation study. The effects of various forms of misalignment, measurement errors, and modeling difficulties are hard to analyze holistically (although we try in). However, it is relatively straightforward to set up a simulation that processes hypothetical scenarios by

1. providing runway and aircraft position,
2. simulating measurement noise,
3. retrieving the pose estimate by solving the PnP problem, and
4. repeating this experiment numerically many times.

Note that here "measurement" refers to the Neural Network that predicts the location of features in the image plane.

In the next sections, we will first briefly explore how the simulation is set up, and then investigate the effects of

- using near corners, near&far corners, corners from other runways;
- using sideline angles as an additional feature;
- considering correlated noise in the measurements;
- considering a non-straight approach attitude;

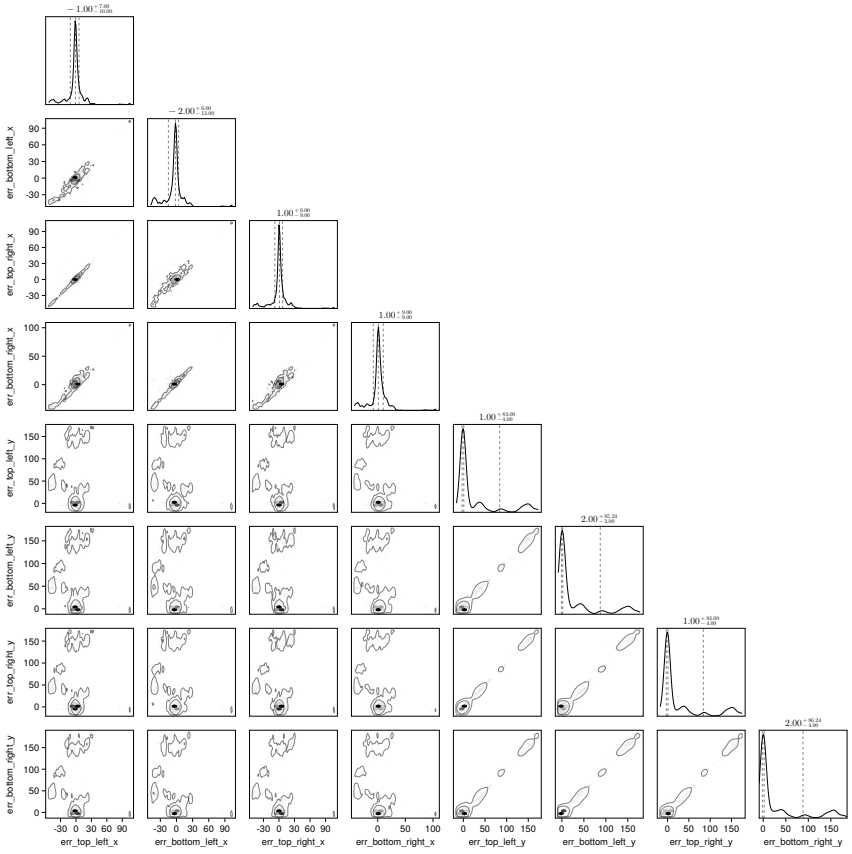


Figure 2: Error correlations for all samples in the service volume. We see that x values and y values are strongly correlated, however there's no strong correlation between x and y.

- the effects of error in attitude estimation; and
- consistency across different runways (KABQ, KSFO, ...).

A note on reproducibility: *We put significant effort into providing reproducible results and figures. The code to reproduce the figures should be fully contained in the* `PNPSolve.jl` *project on the Wayfinder github project. Instructions to run the code are provided in the Readme.*

3.1 Baseline setup:

For the simulation, we use the spatial data of real runways, which we retrieve from a database (`2307 A3 Reference Data_v2.xlsx`). Our baseline case will be the runway KABQ (Albuquerque), with the aircraft positioned at an alongtrack distance of (negative) 6000 m, horizontally centered (i.e. no crosstrack error), and with a vertical angle of 1.2°. A picture of KABQ is provided in . Note that the crosstrack and height values are chosen such that they lie on the extreme of the service boundary. The attitude is facing straight forward at the runway (no pitch), and is assumed to be precisely known.

We start by using only near two corners for the position estimation, with equal weighting in x/y direction. Measurement errors are sampled by adding zero-mean Gaussian noise with one pixel of standard deviation. (This is roughly consistent with the real error distribution in decent conditions, see Section 2.)

The optimization problem is initialized with an initial guess that is set to the true location plus three samples from a zero-mean Gaussian with 50 meters of standard deviation.[2] We solve the problem similar to how opencv solves it: given known 3D datapoints, and assuming a pose, we project the 3D datapoints onto the screen and compare with the actual measurements. Then we use an optimization method to minimize the squared sum of errors in x and y direcetion (screen coordinates). Specifically, we use the Levenberg-Marquardt algorithm (same as OpenCV) provided by LsqFit.jl. We also tried other algorithms, but generally observed worse performance, specifically when other measurements like angles are also taken into account.

In the next sections, we will first explore the numerical results that we receive when running this setup. Then, we will change most of the assumptions, one-by-one, and investigate the results on the estimation error.

3.2 Error distribution for different alongtrack distances.

We start by investigating the error distribution of the pose estimate resulting from randomly sampled measurement noise, and evaluated at different alongtrack distances.

Fig. 3 shows the resulting error distributions (median, 25th and 75th percentile, approximate 99th percentile) for different alongtrack distances given the baseline setup described above. (Note that we additionally report the error requirements specifiified in the MPVS.)

We can observe that the y (crosstrack) and z (height) directions are indeed well within spec, and will likely still easily be in spec even given significantly larger pixel errors. However, the x (alongtrack) direction does not have such a large margin for error, although the requirements are just about satisfied at the current level.

In order to interpret these results, let's recall the following caveats:

1. We assume both near corners are perfectly visible;

2. Despite the runway being fully visible, in some situations we may have larger pixel errors than assumed here, which will increase these error distributions (approximately linearly, see Section 3.6);

3. We assume a dead-straight attitude and no attitude error (we explore violating these assumptions in Section 3.7).

[2] We find that the basic optimization is relatively robust to initialization, but becomes more sensitive when more measurements, like angles, are added.

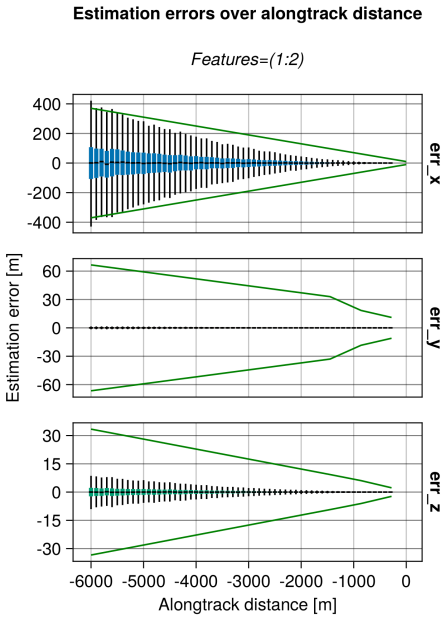


Figure 3: We plot the distribution of position estimate errors using the two near corners (i.e. features 1:2) for different alongtrack distances, and report median, quartiles, and approximate 99th percentiles.

Next, we will explore the effects of adding other runway and angular features, and then consider what happens when some of our assumptions are violated.

3.3 The case of correlated noise

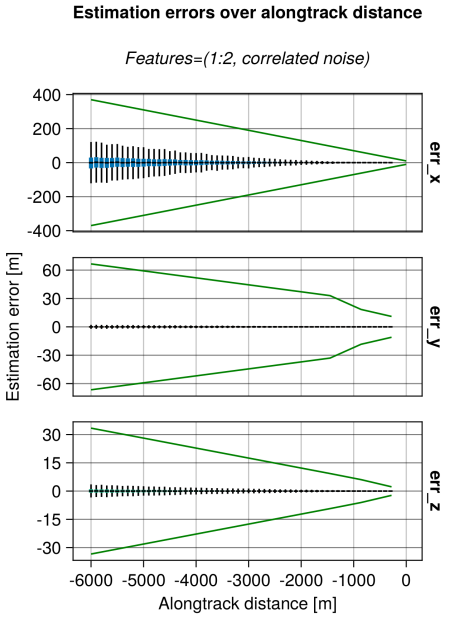
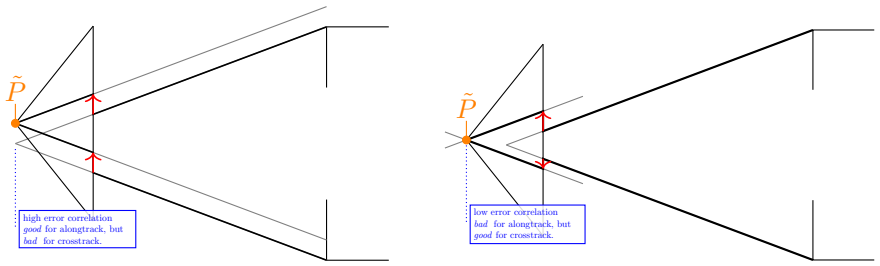


Figure 4: This shows the results with the same setup as above (near corners only, one pixel of standard measurement error), but now the errors are sampled in a correlated fashion according to the correlations measured from real predictions.

As we have seen in Section 2.1, the measurement errors are not uncorrelated as assumed in our simulation setup. Indeed, the errors within the x and y predictions, respectively, are highly correlated. Fig. 4 presents the results when rerunning the simulation study, but this time simulating correlated noise with correlation values given by the measured values (see Section 6.2 for the full values).

We find that while the y and z errors are mostly the same, the alongtrack error has improved drastically.



(a) The case of correlated noise. (b) The case of uncorrelated noise.

Figure 5: A schematic illustration of why uncorrelated noise leads to large alongtrack errors.

3.4 Beyond near corners.

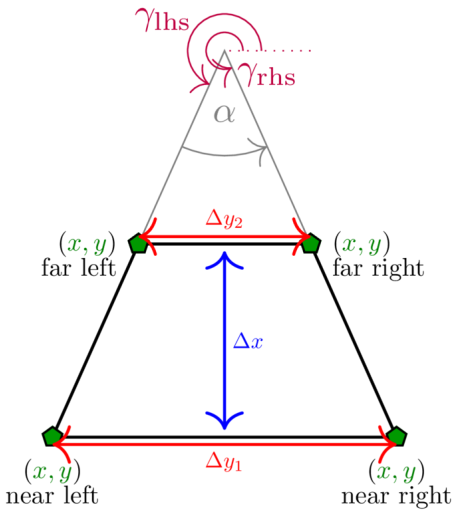


Figure 6: Overview over possible zeroth and first order image features.

We will now consider what features beyond the near corner one could consider adding to the system. For example, we can add additional markings of current or other runways, features of the environment, and also consider "derived" features like the approach angle.

We start by classifying possible features into three categories, moving from features directly in pixel-space to more abstract features, ultimately ending up at directly predicting the position.

Zeroth order: Pixel-space predictions.

- feature locations or pixel indices, e.g. the location of the runway corners in image/pixel space
- notice that depending on whether we predict a location or a pixel index, we can phrase the problem as a *classification* or a *regression* problem.

First order: Image-space derivatives.

- features which can be "drawn into" the image space, e.g. projected threshold width, projected runway length, (enclosed) sideline angles
- notice that all of these predictions are now *regression* problems.

Second order: Beyond the image space.

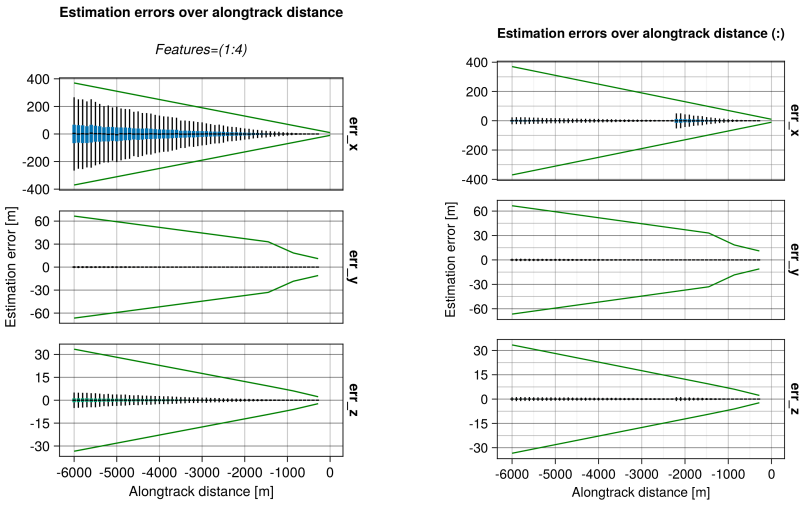
- Direct position prediction
- Another "orthogonal basis", e.g. Nima's angular representation
- Other angles

In Fig. 6 we provide a basic overview over zeroth and first order features.

For now, let us stick to "zeroth order" features, and consider adding the far runway corners, and additionally measuring runway corners from other runways. Fig. 7 presents the results for that setup. First, we notice an improvement by using the far corners of approximately $(33\%, 15\%, 40\%)$ for (x, y, z), respectively. The improvements in x and z directions can be rationalized by realizing that using these points, we can measure "vertical projection length" in the image plane (i.e. Δx in Fig. 6), which strongly correlates with alongtrack position and height. However, we notice that the errors in x-direction still have a fairly wide spread.

The reason for the significantly worse precision in x-direction is easy to rationalize. The position estimation from image correspondences roughly corresponds to finding the intersection (or closest point) of two almost parallel rays which pass through the camera plane and the 3d correspondences. A small error in the specifics of the rays corresponds in the intersection being moved drastically along the ray's directions – which corresponds to our alongtrack estimate.[3]

[3]Notice that this problem is directly related to the condition number of a 2x2 matrix, which is roughly speaking poorly conditioned when the column vectors (i.e. ray directions) are almost parallel and singular if they are exactly parallel.



(a) Results using all four corners of the approaching runway.

(b) Results using all four corners of all runways.

Figure 7: Using near&far features (lhs) and using features from all visible runways (rhs).

It is therefore natural to consider also landmarks that lie in a direction different to the alongtrack direction. The results of this are pictured on the right side in Fig. 7, and indeed we see that the alongtrack performance is massively improved! Interestingly, we also see that the performance drops again closer to the runway, presumably because some threshold corners go out of sight. Notice that we still assume the same error distribution for those detections – however due to the sharp angle, the real error distributions might be larger, and these results may be overly optimistic.

It is also important to note that while these results are promising, if we design the system such that it requires relying on other runways to be in sight, we are severely limiting the systems applicability to airports with multiple runways, and assume all of them to be clearly in sight (no occlusion etc). However, this technique may be used to further boost performance of an already certifiable system.

3.5 First order features: Including angular measurements

In the previous section we have seen the effect of including different pixel features, i.e. runway corners from the approaching or other runways.

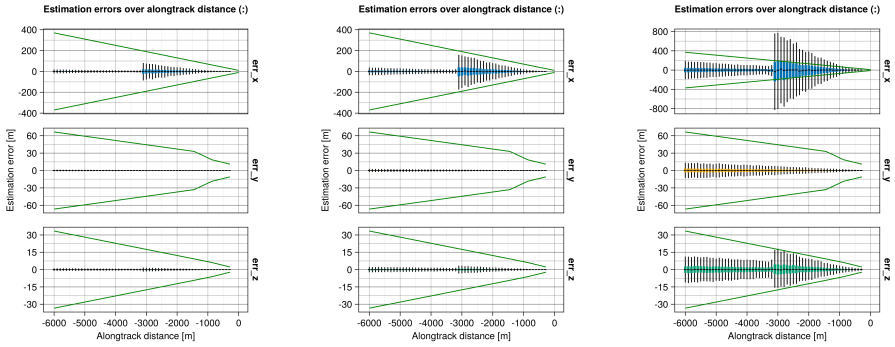
Now also consider adding additional information: the left and right sidelines angles of the approaching runway (i.e. γ_{lhs} and γ_{rhs} in Fig. 6).

We first note that adding these features seems to make the numerical optimization much more unstable, requiring a more sophisticated solving optimizer and having convergence problems if the initializations are not great (especially if the optimization is initialized with a lesser height than the true solution).

In Table 1 we report the prediction performances at different noise levels. We notice that already one degree of angular noise results in worse prediction performance than using no angular measurements at all. However, the results can be improved somewhat if there is very little noise – however even then the improvements are not great, as the optimization still needs to take the noise near corners into account. A re-weighting scheme based on the angle sensitivities and error distributions could, improve the results – however we also found that larger weights for the angular terms resulted in more numerical instability.

σ_{angle}	0.0°	0.01°	0.1°	0.3°	0.5°	1.0°	no angles
σ_{x}	101.22	104.61	105.89	110.58	122.37	159.51	156.54
σ_{y}	0.3863	0.3967	0.3986	0.4267	0.4869	0.5968	0.5678
σ_{z}	2.1189	2.1929	2.2021	2.3073	2.5945	3.4132	3.3251

Table 1: Prediction errors when using sideline angle measurements with different levels of noise, and compared against a baseline (last column).



(a) One pixel of standard error.

(b) Two pixels of standard error.

(c) Ten pixels of standard error.

Figure 8: Linear correlation between measurement errors and prediction errors. Left-to-right: 1pxl, 2pxl, and 10pxl of standard pixel error.

σ_{angle}	0.0°	5.0°	10.0°	15.0°	20.0°	25.0°	30.0°	35.0°	40.0°	45.0°
σ_{x}	3.622	4.034	15.41	43.16	62.25	71.97	77.04	89.72	81.32	86.64
σ_{y}	0.319	0.350	4.641	32.24	46.76	55.61	63.67	68.68	71.67	74.78
σ_{z}	0.327	0.333	1.242	14.29	21.11	27.26	29.89	31.37	35.46	36.69

Table 2: Position estimation errors given misaligned aircraft attitude. Using features from all visible runways.

3.6 Further assumptions: Linear error scaling and different runways

It seems empirically correct that the position estimation errors scale linearly with the feature location errors, see e.g. the comparison in Fig. 8. However, we do note that it's not clear whether this also holds for more "nonstandard" setups, for example when the attitude wrt the runway is misaligned, and all features are located at a "sharp angle".

Another related question that is whether our results actually hold for the majority of the runways. To answer this question, we have rerun the basic setup (with near corners and all visible corners) for every runway listed in the datasheet, which includes about 2000 runways. Indeed we find significant variance, however more data is required.

3.7 Further assumptions: Attitude misalignment

Here, we explore if the results also hold when we're not facing the runway straight on. Note that we still assume that we know the precise aircraft orientation/attitude, but we now sample that attitude as follows: We start with a straight heading, 6000 m alongtrack distance away (as before). Then, we sample a random vector in the Unit sphere, and sample an angle from a zero-mean Gaussian with a given standard deviation. The orientation is then rotated around the sampled vector by the sampled orientation.

The results when taking features from all runways are presented in Table 2. Tables for only near- and near-far corners are found in Section 6.3.

Studying this table, we notice that we receive significant prediction problems when the camera is even only 15° tilted wrt the runway direction.

3.8 Further assumptions: Attitude prediction error

In the previous section we explored the effects of having an attitude that is precisely known, but not pointing straight at the runway. In this section we

σ_{angle}	0.0°	0.01°	0.1°	0.3°	0.5°	1.0°
σ_{x}	146.39	148.52	144.93	149.27	150.55	150.22
σ_{y}	0.5724	0.8271	6.1943	17.536	30.030	58.797
σ_{z}	3.1291	3.1956	6.8751	16.831	29.688	57.163

Table 3: Standard position estimation error given a wrong belief about our attitude (which is actually dead straight), using only near features.

consider the question what happens if we are facing the runway precisely, but falsely believe that we are rotated by a small amount.

Table 3 provides the results for this setup. As we can see, around 0.5° attitude belief error already leads to position estimation errors close to the requirements. Keep in mind, this experiment also includes pixel position measurement noise (see the first column where there no attitude noise yet). Notice also that the alongtrack error stays nearly unaffected, unlike the crosstrack and height error.

newpage

4 Theoretical analysis

We begin the discussion by analyzing the PnP algorithm for a specific runway and the simplest choice of feature representation: predicting the pixel coordinates of the near runway corners. We will explore how we can analyze the algorithms sensitivity to errors, and what happens when we introduce additional features to the representation, namely the far corners, and corners from other runways. We will see that by default, the algorithm performs somewhat poorly in particular in alongtrack direction, and also does not improve by adding the features. We will discuss what can be done for this specific case, and then discuss a more general setting where we consider a range of different parametrizations and their sensitivities.

4.1 Localization on the Albuquerque runway.

Let's start by considering the localization problem on the Albuquerque runway (KABQ). KABQ has three different runways, which are oriented towards north-east, south-east, and east respectively. For now we will assume that we are located south-west of the airport, approaching the north-east facing runway. This runway has length 3000 m and width 65 m. We define the coordinate system as in the MSVP: it is centered in the middle of the runway's

threshold line, with the x-axis aligned with the alongtrack direction, the y-axis pointing along the threshold line to the left, and the z-axis pointing up.

In order to understand the safety-properties of our algorithm, we want to explore some worst-case properties. Therefore, we will position our aircraft/camera as far and low as "possible" (as given by the service volume); however, we will center it and align it perfectly for now. Let θ denote the position and R the rotation matrix (Euler notation), then we have

$$\theta = \begin{bmatrix} -6000\,\mathrm{m} \\ 0\,\mathrm{m} \\ 126\,\mathrm{m} \end{bmatrix} \text{ and } R = I_3.$$

We will now consider the real locations and projections of the front two corners. For this, let

$$p_{n,l}(\theta) = \begin{bmatrix} x_{n,l}(\theta) \\ y_{n,l}(\theta) \\ z_{n,l}(\theta) \end{bmatrix} = \begin{bmatrix} 6000\,\mathrm{m} \\ 0\,\mathrm{m} \\ -61\,\mathrm{m} \end{bmatrix}$$

denote the *near, left* (index (n, l)) threshold corner in camera perspective, and let

$$p'_{n,l} = \begin{bmatrix} x'_{n,l} \\ y'_{n,l} \end{bmatrix}$$

denote the projections of the corner onto the image plane, using the standard pinhole camera model with focal length $f = 35\,\mathrm{mm}$. (Here, x' points up in the camera image, and y' points left.) Specifically, let's consider $\tilde{p}'_{n,l}$ as the measured (and possibly erroneous) corner positions estimates in the image as determined by the computer vision algorithm, and $p'_{n,l}(\theta)$ as the reprojection of the known corner location given a camera position estimate θ. Then we can try to find our pose estimate by solving

$$\theta = \arg\min_{\theta} \sum_{\rho \in \{l,r\}} \left(\tilde{x}'_{n,\rho} - x'_{n,\rho}(\theta)\right)^2 + \left(\tilde{y}'_{n,\rho} - y'_{n,\rho}(\theta)\right)^2 \tag{1}$$

using an iterative solver and a reasonable initial guess. Notice that Eq. (1) contains four data points $(\tilde{x}'_{n,l}, \tilde{x}'_{n,r}, \tilde{y}'_{n,l}, \tilde{y}'_{n,r})$ for three variables $(\theta_1, \theta_2, \theta_3)$, i.e. we already have an overdetermined problem[4].

[4]Even though the problem is overdetermined, there still may be (and are) multiple local minima, due to the nonlinearity of the projection and squared loss. However, if sufficient care is taken with the optimization algorithm and initial guess, we have reasonable hopes to converge to the "correct" θ.

In order to analyze this equation further, we denote the loss function as $l(\theta)$ and set the derivative to zero, i.e. $\nabla_\theta l(\theta) \stackrel{!}{=} \vec{0}$. Then we get

$$\nabla_\theta l(\theta) = \sum_{\rho \in \{l,r\}} \left(\tilde{x}'_{n,\rho} - x'_{n,\rho}(\theta)\right) \left(-\frac{\partial x'_{n,\rho}(\theta)}{\partial \theta}\right) + \left(\tilde{y}'_{n,\rho} - y'_{n,\rho}(\theta)\right) \left(-\frac{\partial y'_{n,\rho}(\theta)}{\partial \theta}\right) \stackrel{!}{=} 0. \tag{2}$$

Interestingly, we can see that the quantities q (where $q = x'_{n,\rho}$ or $q = y'_{n,\rho}$) are minimized "more strongly" if $\frac{\partial q(\theta)}{\partial \theta}$ is large. Another perspective on this observation is that $\frac{\partial q(\theta)}{\partial \theta_i}$ determines how much the difference $(\tilde{q} - q(\theta))$ influences the pose coordinate θ_i. (Recall that $\frac{\partial q(\theta)}{\partial \theta}$ is a three dimensional vector with one entry for each θ_i.)

Further, we notice that if q is very sensitive to θ, minimizing $(\tilde{q} - q(\theta)$ will have a larger effect on θ than if q is not very sensitive – again directly linked through $\frac{\partial q(\theta)}{\partial \theta}$. Therefore, it turns out that we are placing an implicit "importance weights" of how much influence a given quantity has on θ where the importance weight is equal to $\left(\frac{\partial q(\theta)}{\partial \theta}\right)^2$.

It seems therefore natural to consider the partial derivatives directly to understand each components influence on the final solution. This will also help us understand what happens when we add more image features.

4.2 Computing partial derivatives.

We wish to compute the partial derivatives $\frac{\partial q_i}{\partial \theta_j}$ for the quantities and position coordinates introduced above. Unfortunately, the relation between q_i and θ_j can become quite complicated, especially when considering the general case with arbitrary rotations and positions. However, we are able to establish analytic relationships without too much struggle when considering positions that have no crosstrack offset and straight orientation. (We will discuss later what happens when these assumptions are violated.)

TODO (Romeo): I can probably move this part to the appendix or so...
Before we get started, let's recall one important mathematical theorem – The Inverse Function Theorem – which roughly states that for bijective functions $q(\theta)$ we have $\frac{\partial q(\theta)}{\partial \theta} = \frac{1}{\frac{\partial \theta}{\partial q(\theta)}}$ (see e.g. Wikipedia).

Let us again consider the position and orientation introduced in Section 4.1. In this setting, we first establish relations between x' and the different $\theta_i \in \{x, y, z\}$ using

$$\frac{x'}{f} = \frac{z}{x}.$$

Using this simple relation, we can find

$$\frac{\partial x'}{\partial x} = -\frac{fz}{x^2}$$

and

$$\frac{\partial x'}{\partial z} = -\frac{f}{x}.$$

We further notice that x' and y are independent.

We can do something similar for y', although the equations are more complicated (see Section 6.4). All together, we get the following Jacobi matrix:

$$\left(\frac{\partial q_i}{\partial \theta_j}\right)_{ij} = \begin{bmatrix} -\frac{fz}{x^2} & 0 & -\frac{f}{x} \\ -\frac{\sqrt{\left[\frac{\Delta y}{\Delta y'}\right] f^2 - z^2}}{[x'^2+f^2]} \frac{y'^3}{y^2} & \sqrt{\frac{x'^2+f^2}{x^2+z^2}} & -\frac{\sqrt{\left[\frac{\Delta y}{\Delta y'}\right] f^2 - x^2}}{[x'^2+f^2]} \frac{y'^3}{y^2} \end{bmatrix} \tag{3}$$

$$= \begin{bmatrix} -1.225 \times 10^{-7} & 0 & 5.833 \times 10^{-6} \\ -2.963 \times 10^{-8} & 5.835 \times 10^{-6} & -6.224 \times 10^{-10} \end{bmatrix}.$$

(Note that $\frac{\partial x_{n,l}}{\partial \theta_j} = \frac{\partial x_{n,r}}{\partial \theta_j}$ etc, therefore the full matrix would be just twice the printed matrix stacked).

If we recall again that these terms build an implicit weight on how strongly each term is minimized, we can see that $\theta_1 = x$ is mostly minimized through x', and similar for $\theta_3 = z$. Interestingly though, $\frac{\partial z}{\partial x'}$ is roughly five times larger than $\frac{\partial x}{\partial x'}$. This means in cases where x and z are not consistent, the optimization algorithm places a much higher weight on minimizing the consistency error for z. In Section 5 we will go into more detail how such effects can be circumvented.

Using the inverse of the derived partial derivatives, we can also make some statements about the magnitude of the errors that may be introduced given some pixel error. Recall that by the Inverse Function Theorem we have $\frac{\partial \theta_j}{\partial q_i} = \frac{1}{\frac{\partial q_i}{\partial \theta_j}}$ and that $\Delta\theta_j \approx \frac{\partial \theta_j}{\partial q_i} \cdot \Delta q_i$. For now, let us assume that we have one pixel of error, and let the pixel size be 0.003 45 mm. Then, e.g. the error introduced to our estimate for x through y' can be up to 126 m. TODO (Romeo): Have to think a bit more about how the potential errors interact with the implicit weights...

Note that this is roughly consistent with our experimental numerical results, that show that for one pixel of measurement error we get about 120 m of alongtrack estimation error.

4.3 Beyond near corners.

Now that we have seen how to analyze the scenario when measuring the locations of the near corners, let's consider what would happen if we take into account the far corners as well. Indeed we can simply compute the same Jacobian as in Eq. (4) and replace x by $x + \Delta x$. This yields

$$\left(\frac{\partial q_i}{\partial \theta_j}\right)_{ij} = \begin{bmatrix} -5.444 \times 10^{-8} & 0 & 3.888 \times 10^{-6} \\ \\ -1.317 \times 10^{-8} & 3.889 \times 10^{-6} & -9.822 \times 10^{-9} \end{bmatrix}. \tag{4}$$

TODO (Romeo): I have to think a bit more about what this implies and how this matches up with the observation I've made in the numerical simulations.

5 The PnP Problem and beyond the OpenCV implementation

In the previous chapter we have seen different possibilities for representing image features and how they relate to our current position estimate. However, it is not a trivial problem to recover a position estimate given the image features.

Indeed, in general no exact solution exists, and instead a least-squares problem has to be solved iteratively to find the position that is the most consistent with the observations. Additional care has to be taken when selecting the specific solver, and when setting up the objective function. In the next pages, we will discuss how to set up the objective function, how to weight each term, how to deal with correlated noise, and which solver to choose.

5.1 Related work: OpenCV

Before we dive in to the detail, let's first consider why simply using the OpenCV solver may not be sufficient. OpenCV offers a range of algorithms solving the "PnP Problem", i.e. the Perspective-from-n-Points, in particular

- an iterative solver based on general point reprojections,

- algebraic solvers building an exact correspondence,
- several specialized iterative solvers when certain assumptions hold.

However, all these solvers are not ideal for our scenario:

1. We are trying to solve a simpler problem than the general PnP problem, as we already have the rotation given.
2. We may have more points than the minimum required, which eliminates algebraic solutions.
3. The PnP solver may only work with point correspondences, i.e. zeroth order features as described in .
4. The OpenCV implementation weighs all terms equally. I.e., the error in x-direction of the front left corners has the same weight as the error in y-direction of the back right corner.

In the geometric analysis, we have held many factors constant and were able to compute analytic equations that way. However, when not all factors are controlled, this type of analysis can fall apart. This further becomes more difficult if we try to solve an over-determined problem.

In the following we will consider the following thoughts:

- solving an overdetermined system in practice.
- sensitivity of the solution in an overdetermined system

5.2 Solving an overdetermined system in practice.

Recovering the pose from N known points is generally called the PnP problem (Perspective-n-Point). Basically, given a pose guess, the known points are projected onto the camera and then compared to the measured points (assuming known association), typically using a simple squared loss in x- and y-direction. More advanced techniques exist, but the typical solution uses the Levenberg-Marquardt algorithm (a mix of first- and second-order optimization), or sometimes a damped Gauss-Newton method (only second order), to solve the resulting problem.[5] Initial guesses may be provided or computed with the DLT algorithm (as OpenCV does).

Therefore, a simple implementation may look like this:

[5] The OpenCV website has a good overview of other methods.

```
function pnp_LM(world_pts::Vector{XYZ{Meters}},
                pixel_locations::Vector{Point2{Pixels}},
                pred_angles::Vector{Angle},
                cam_rotation::Rotation{3};
                initial_guess::XYZ{Meters} = XYZ(-100.0m, 0m, 30m),
                )
    N, M = length(pixel_locations), length(pred_angles)
    w_points, w_angles = ones(N)/(N+M), ones(M)/(N+M)
    loss(cam_position::XYZ{Meters}) = let
        P = make_projector(cam_position, cam_rotation)
        projected_points = P.(world_pts)
        projected_angles = compute_angles(projected_points)
        # compute possible derived properties, like enclosing angle, ...
        return ( sum(w_points.*(projected_points .- pixel_locations).^2)
               + sum(w_angles.*(projected_angles .- pred_angles).^2)     )
               ↪  #      (loss-eq)
    end
    sol = optimize(loss, initial_guess,
                   LevenbergMarquadt())
    return minimizer(sol)
end
```

where

```
function make_projector(cam_position::XYZ{Meters},
↪   cam_rotation::Rotation{3};
                      focal_length=25.0mm, pixel_size=0.00345mm/1px)
    scale = focal_length / pixel_size
    projector(pt::XYZ{Meters}) =
        rel_pt::Point3{Meters} = cam_rotation'*(pt - cam_position)
        proj::Point2{Pixels} = scale * 1/rel_pt[1] * Point2(rel_pt[2],
        ↪  rel_pt[3])  #      (proj-eq)
        return proj
    end
    return projector
end
```

(Note the auto-vectorization syntax `f.(vec)` which applies `f` elementwise to each element of `vec`.)

Notice that we may include extra terms and/or weights in loss-eq, e.g. weights on x vs y, angular terms, etc. The problem seems almost linear – notice however the division in proj-eq. [6] For the LM algorithm, we need the gradient and Hessian. Fortunately, if we use the right tools we can

[6] I believe there might be some tricks similar to the DLT algorithm that let us solve the problem using Moore-Penrose pseudoinverse. However, this doesn't work anymore if we e.g. also consider angles.

automatically construct them, e.g. using `ForwardDiff.jl` or similar.

5.3 Sensitivity of the solution of an overdetermined system.

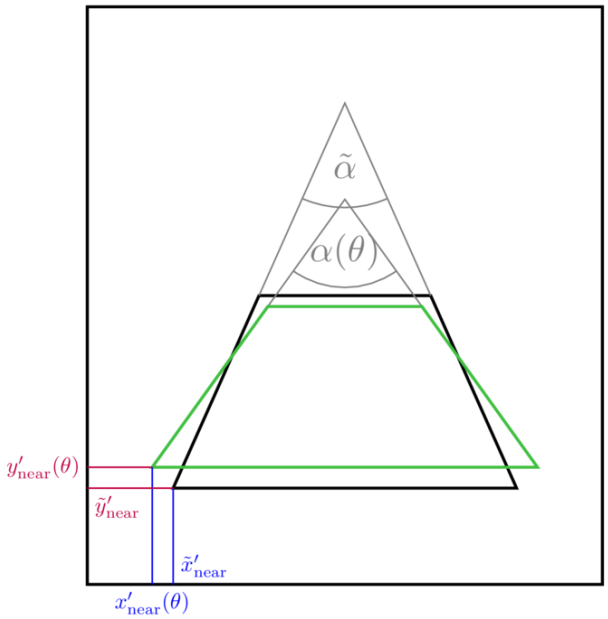


Figure 9: A simple example of runway observations $\tilde{\alpha}, \tilde{x}'$ and $\tilde{y}'$ and predictions $\alpha(\theta), x'(\theta)$ and $\tilde{y}'$) given the position estimate θ.

Next we will investigate the question what happens when we aggregate solutions to sub-problems in a least-squares fashion. The motivation is as follows: Suppose that part of our pose estimation is done through an angular representation, and part through the near and far left runway corners. Let θ denote our variable of interest, i.e. the position $\theta = (x, y, z)$. Then, our objective function might look like

$$\min_{\theta} f(\theta) \tag{5}$$

with

$$f(\theta) = w_1 \left(\alpha(\theta) - \tilde{\alpha}\right)^2 + w_2 \left(x'_{\text{near}}(\theta) - \tilde{x}'_{\text{near}}\right)^2 + w_3 \left(x'_{\text{far}}(\theta) - \tilde{x}'_{\text{far}}\right)^2 \tag{6}$$

where α denotes the angle, $x_i' = project(x_i)$ the projected world coordinates, $\tilde{q}$ the estimate of any quantity q, and $q(\theta)$ the simulated quantity q given the pose θ. An illustration is provided in Fig. 9.

In order to solve this system, we choose weights w_i for each component. Should we set them equally? Do we need to tune them by hand? Or can we come up with a mathematical suggestion? To further motivate this question, we start by considering a few reasons why we can not set all weights equally. Then we derive

5.3.1 Motivation

We will briefly discuss three different perspectives on why weights need to be chosen carefully, and can in general not be set to be equal.

First, an easy way to see that we can not choose all weights to be equal in Eq. (6) is to consider the case where the angle α is measured in radians, and one where it is measured in degrees. To illustrate, let $(\alpha(\theta) - \tilde{\alpha}) = 0.1\text{rad} \approx 5.7°$. It is now easy to see that, despite both values representing the same measurement, if we ignoring the units when evaluating $f(\theta)$ the representation in degrees has approximately $57^2 = 3249$ times more relative weight then the first, and may thus dominate the the terms with w_2 and w_3.

The observant reader will notice that this is an obvious problem given the lack of dimensional analysis. Indeed, when considering Eq. (6) again, it seems clear that we should not add an angle to a distance in the first place – as you would also not add 100 millimeters to 100 meters and expect a result with quantity 200. It therefore falls onto the factors w_i to contain the correct "conversion" such that all terms can be added sensibly. We will see that by setting the weights to the inverse squared derivative $w_i = 1/\left(\partial_\theta q(\theta)\right)^2$ we can stay consistent in dimensional analysis.

A second argument can be made around uncertainties. Suppose we can predict one measurement with high certainty, say 0.1 units, and another one only with very low certainty, say 10 units (measured for example in pixel space, in degrees, etc). It seems clear that we should place a higher weight on the first measurement than on the second. But how much higher?[7] We will see that under the Gaussian assumption, the weight will be exactly the inverse variance, which is directly related to the previous result (more on

[7] There is an interesting duality when measuring the location of things in pixel space: Objects that are closer are also larger, and may have therefore have a larger uncertainty measured in pixels than objects far away, even though they are easier to detect! The connection then comes through the sensitivity analysis, given that the sensitivity of the pose to errors in pixel space is typically also lower for large objects, so it cancels out.

that later).

A final, somewhat related argument considers the sensitivities of each term directly. Suppose we try to estimate θ by measuring two related quantities – $q_1(\theta)$ and $q_2(\theta)$ – but $q_1(\theta)$ is very sensitive to error in θ, i.e. $\partial_\theta q(\theta)$ is large. If we then try to solve $\min_\theta (q_1(\theta) - x_1)^2 + (q_2(\theta) - x_2)^2$ directly, the result will be dominated by $q_1(\theta)$, even though it may not contain any more information than $q_2(\theta)$. Yet again, the key lies in choosing weights as the inverse of the squared derivatives.

Next, we present some mathematical analysis to underline each point.

5.3.2 Sensitivity analysis

Suppose that the predicted quantity $q(\theta)$ is very sensitive to θ, i.e. $\partial_\theta q(\theta)$ is large. Consider

$$\min_\theta f(\theta) \text{ with } w_1(q_1(\theta) - x_1)^2 + w_2(q_2(\theta) - x_2)^2 \tag{7}$$

and let $\theta_1 = [q_1]^{-1}(x_1)$ and similarly θ_2, where $[q_i]^{-1}$ denotes the function inverse, assuming it exists. Solving $\partial_\theta f(\theta) = 0$ yields

$$\begin{aligned}\partial_\theta f(\theta) &= 2\left[w_1 \partial_\theta q_1(\theta)\left(q_1(\theta) - x_1\right) + w_2 \delta_\theta q_2(\theta)\left(q_2(\theta) - x_2\right)\right] = 0 \\ &\Rightarrow -\frac{w_1 \partial_\theta q_1(\theta)}{w_2 \partial_\theta q_2(\theta)} \frac{\left(q_1(\theta) - x_1\right)}{\left(q_2(\theta) - x_2\right)} = 1.\end{aligned}$$

If we now assume $w_1 = w_1$ and $\partial_\theta q_1(\theta) \gg \partial_\theta q_2(\theta) = C \cdot \partial_\theta q_1(\theta)$ with $C = 100$ we get

$$\frac{C}{1} \frac{\left(q_1(\theta) - x_1\right)}{\left(q_2(\theta) - x_2\right)} = 1, \tag{8}$$

i.e. error in $q_1(\theta)$ is weighted 100 times as much as error in $q_2(\theta)$. Further, since $q_1(\theta)$ grows faster, we actually end up with something like C^2. To see this, consider simply $q_1(\theta) = 100\ \theta$ and $q_2(\theta) = \theta$. Inserting this into Eq. (8) yields

$$\begin{aligned}\frac{100}{1} \frac{\left(100\ \theta - x_1\right)}{\left(\quad 1\ \theta - x_2\right)} &= 1 \\ \Rightarrow \theta &= \frac{100^2 \frac{x_1}{100} + 1^2 \frac{x_2}{1}}{100^2 + 1^2}\end{aligned}$$

i.e. we interpolate between the two solutions $x_1/100$ and $x_2/1$ with weights $w_1 = 100^2/(100^2 + 1^2)$ and $w_2 = 1/(100^2 + 1^2)$. (Note: We can easily obtain the same result by inserting our choices for $q_1(\theta)$ and $q_2(\theta)$ into Eq. (7), rewriting the equation as $\min_\theta w_1 100^2(\theta - \frac{x_1}{100})^2 + w_2 1^2(\theta - \frac{x_2}{1})^2$, and recalling our result from .)

A Appendix

A.1 Reproducing the results

A.1.1 Correlations

The results in this section may be reproduced by the notebook located at *https://github.com/airbus-wayfinder/PNPSolve.jl/blob/main/notebooks/error_distribution.jl* *using input data located at* *login2:/home/romeo.valentin.int/vnv_proces*

A.1.2 Barplots for estimation errors

A.1.3 Different features

```
julia> plot_alongtrack_distance_errors(; distances=(300:100:6000).*1m, features=(;featu
julia> plot_alongtrack_distance_errors(; distances=(300:100:6000).*1m, features=(;featu

julia> plot_alongtrack_distance_errors(; distances=(300:100:6000).*1m, features=(;featu

julia> plot_alongtrack_distance_errors(; distances=(300:100:6000).*1m, features=(;featu
```

A.1.4 Attitude misalignment

```
julia> let feature_mask=(1:2),
           sigma_rot=1.0°
         df = make_alongtrack_distance_df(; feature_mask, sigma_rot, sample_rotations=t
                                            N_measurements=1000, distances=(6000:6000)
         std.(eachcol(df)[[:err_x, :err_y, :err_z]])
       end
```

A.1.5 Attitude prediction errors

Reproduce by running *experiment_with_attitude_noise()*.

σ_{angle}	0.0°	5.0°	10.0°	15.0°	20.0°	25.0°	30.0°	35.0°	40.0°	45.0°
σ_{x}	153.3	151.4	199.2	193.4	217.4	211.4	205.0	181.9	206.6	173.9
σ_{y}	0.581	3.484	27.31	45.62	56.73	66.35	70.51	75.84	79.31	81.20
σ_{z}	3.267	3.742	14.06	22.87	28.98	33.28	34.73	37.42	38.68	40.70

Table 4: Features: (1:2)

σ_{angle}	0.0°	5.0°	10.0°	15.0°	20.0°	25.0°	30.0°	35.0°	40.0°	45.0°
σ_{x}	100.4	100.3	111.9	138.3	117.9	124.3	147.5	123.1	132.4	131.6
σ_{y}	0.495	4.875	27.89	44.95	56.70	66.60	70.57	74.87	78.44	80.46
σ_{z}	1.941	3.371	13.94	22.91	27.96	32.68	35.86	37.57	39.17	40.17

Table 5: Features: (1:4)

A.2 Correlations

Correlation values (in service volume):

$$corr(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} x \\ y \end{bmatrix}) = \left[\begin{array}{cccc|cccc} 1.0 & 0.92 & 0.98 & 0.93 & -0.05 & -0.05 & -0.04 & -0.04 \\ 0.92 & 1.0 & 0.91 & 0.95 & -0.18 & -0.19 & -0.18 & -0.19 \\ 0.98 & 0.91 & 1.0 & 0.93 & -0.04 & -0.05 & -0.03 & -0.03 \\ 0.93 & 0.95 & 0.93 & 1.0 & -0.2 & -0.2 & -0.19 & -0.2 \\ \hline -0.05 & -0.18 & -0.04 & -0.2 & 1.0 & 1.0 & 1.0 & 1.0 \\ -0.05 & -0.19 & -0.05 & -0.2 & 1.0 & 1.0 & 0.99 & 1.0 \\ -0.04 & -0.18 & -0.03 & -0.19 & 1.0 & 0.99 & 1.0 & 1.0 \\ -0.04 & -0.19 & -0.03 & -0.2 & 1.0 & 1.0 & 1.0 & 1.0 \end{array}\right]$$

Notice that all x and all y values are extermely correlated, with a factor of almost 1 (!), and the x/y correlations are almost uncorrelated, although they do seem to have a persistent small negative correlation.

Fig. 10 shows a graphic of the error correlations of the extreme service volume only. Note that it looks similar to the one provided in the report above, even though comparitively few samples were used for creation.

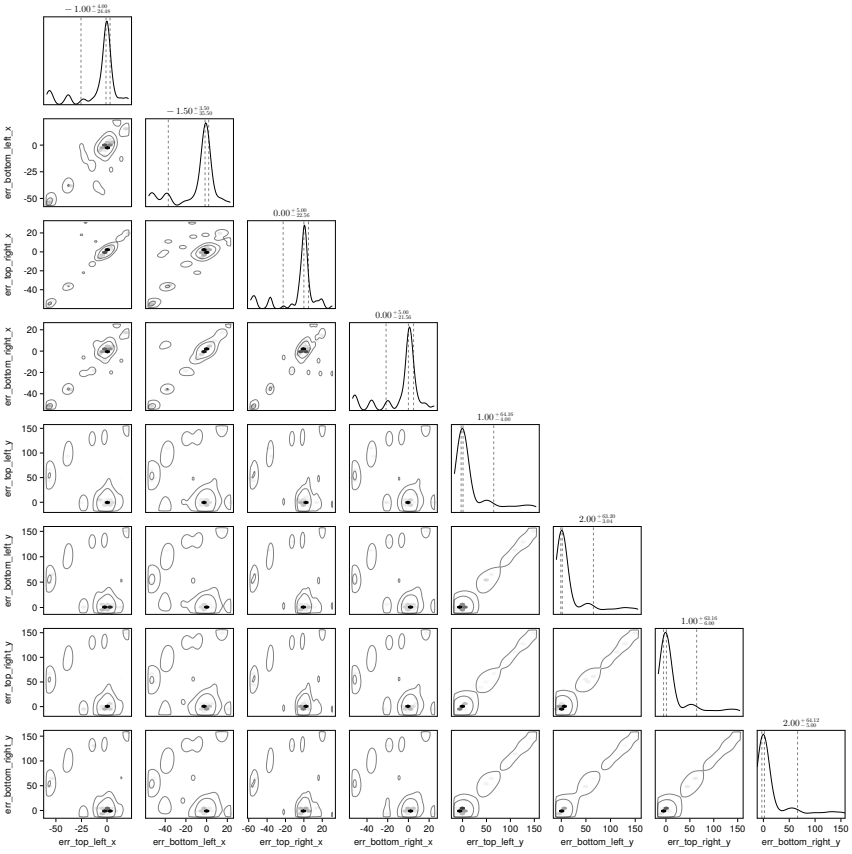


Figure 10: Error correlations only for samples in the extreme service volume.

A.3 Misaligned Attitude, more tables

A.4 Derivation of $\frac{\partial x'}{\partial x}$

By looking from the top onto the situation, we can establish

$$\frac{x^2 + z^2}{y^2} = \frac{f^2 + x'^2}{y'^2}$$
$$\Rightarrow x^2 = \frac{y^2}{y'^2}(f^2 + x'^2) - z^2.$$

Taking the square root, differentiating by x' and inverting the fraction yields the result. The result for $\frac{\partial x'}{\partial z}$ follows similarly.