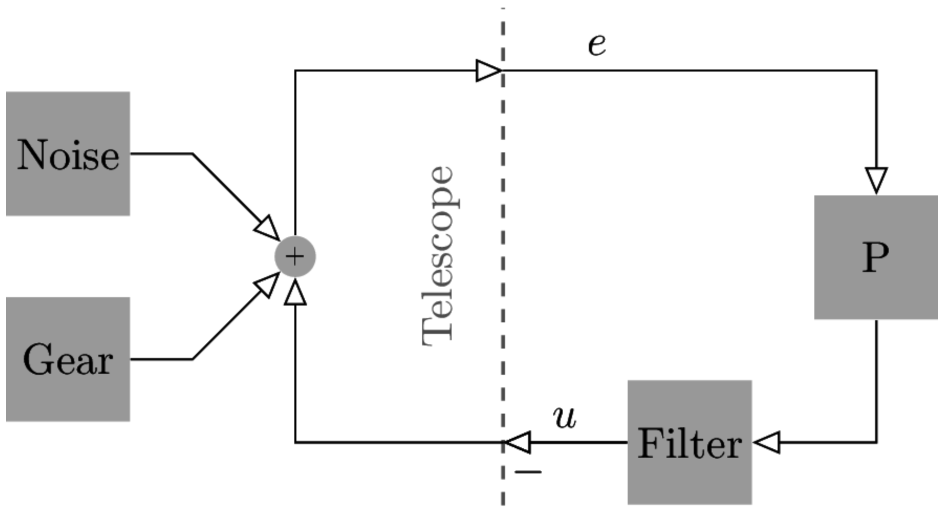
GP Guiding Algorithm Documentation

Introduction

This documentation is intended for developers working on PHD2 Guiding, enabling them to understand, maintain, and contribute to, the Gaussian process based guiding algorithm. The main concepts of telescope guiding and the PHD2 guiding framework are assumed to be known.

Feedback Control

The framework currently used for reducing the periodic error is *feedback* control, which is the main idea behind PHD guiding. The error is fed back to the telescope:



where `Noise` is disturbance from mechanical roughness, `Gear` is the periodic error and drift introduced by the gear, `P` is a proportional controller and `Filter` is used for post-processing the control signal. In the proportional controller, the control action u is proportional to the measured error e:

$$u = -g_p e,$$

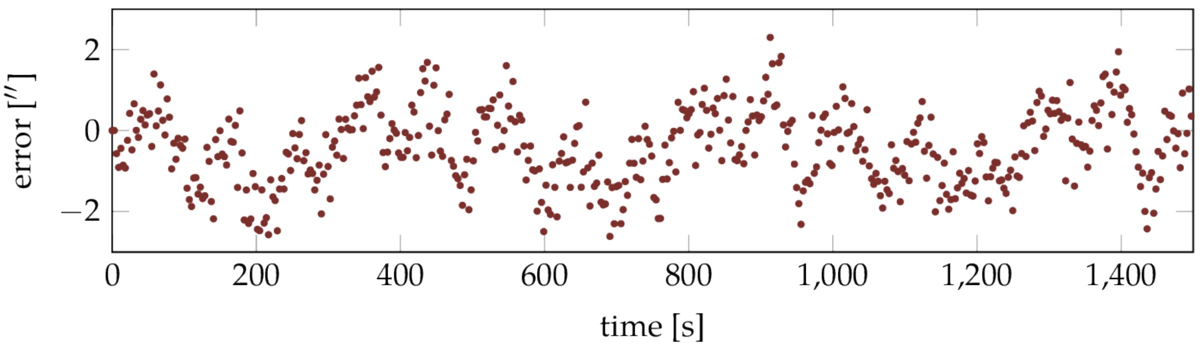
where g_p is the "aggressiveness", or "proportional gain" of the controller. This gain must be smaller than 1 and is usually set to a value around

0.5 to prevent overshoot.

Additionally, some of the guiding algorithms have filters to increase performance:

- low-pass filtering, to prevent oscillations ("LowPass", "LowPass2")
- hysteresis, to prevent direction changes ("Hysteresis")
- switching suppression, to prevent direction changes ("ResistSwitch")
- min-move filtering to suppress small movements

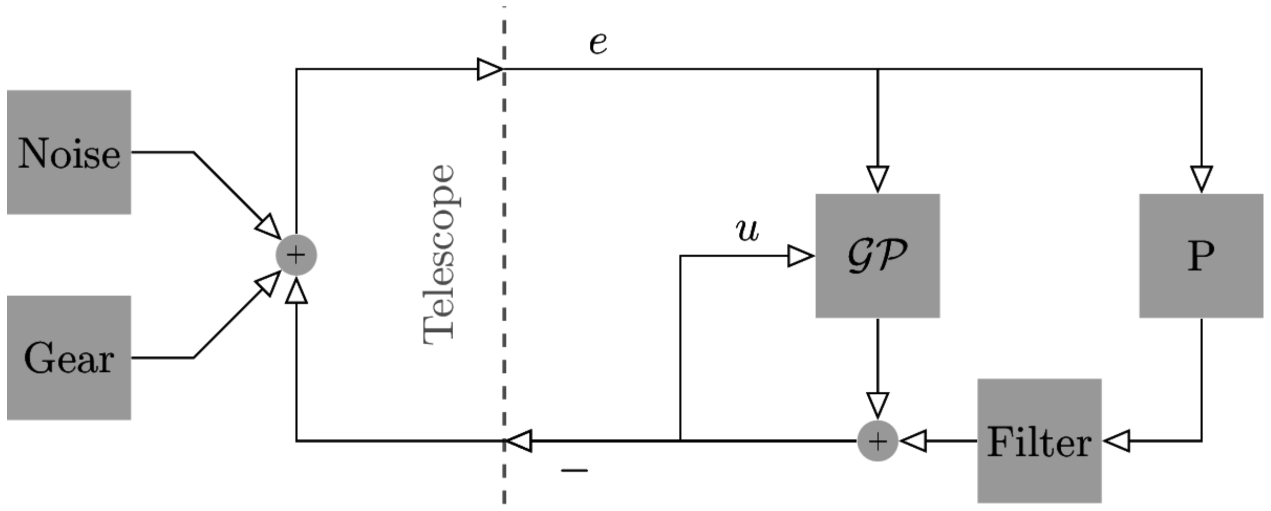
The existing algorithms in PHD2 guiding already provide good performance. However, the residual error still shows periodic structure, suggesting a potential performance gain. The following plot shows a typical measurement with the "Hysteresis" guiding algorithm.



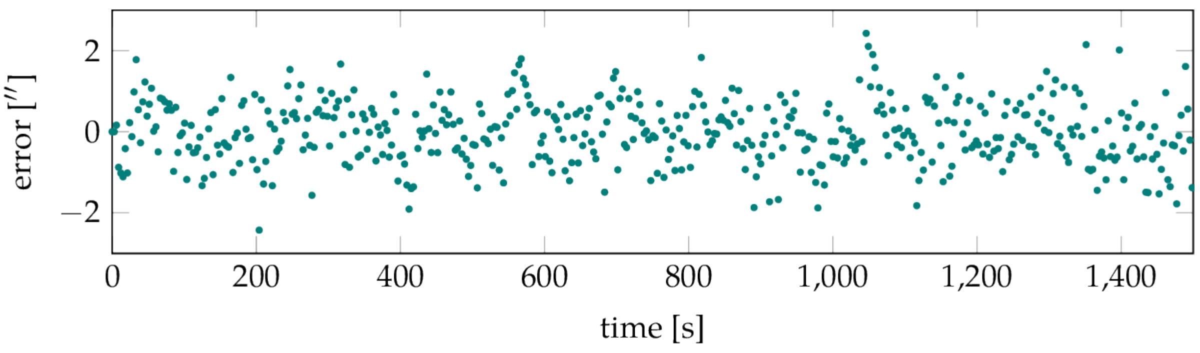
The most likely explanation for this residual structure a combination of periodic gear error and the delayed nature of feedback controllers: In order to correct an error, it needs to be measured first. If the measurement process is slow (as with telescope guiding), there is always a portion of the error that builds up before the measured error is corrected. This delay is responsible for the periodic structure in the residual error.

Predictive Control

The idea behind the predictive control framework is to predict the periodic error from the past data instead of waiting for the measurement process to pick up the error. In order to predict, a model that is constantly updated is used. This model called a regression model, and in our case we use a special type called Gaussian process (GP).



From the control signals u and the measured errors e, the Gaussian process $\mathcal{GP}$ infers a model for the gear error that is introduced by `Gear`. This gear error can be approximately predicted and corrected. Of course, there is also the non-periodic `Noise`, which cannot be predicted. These errors are compensated with a standard feedback controller (the P-block in the above diagram). The `Filter` block in this setup only contains a minimal move limit `min_move_` to prevent the tracking of atmospheric noise effects. Additionally, the prediction can be scaled down proportionally with the `prediction_gain_` parameter, if desired. The following plot shows a measurement with GP-based predictive control on the otherwise identical setup as before. It shows about 20 % less RMS error and much less residual structure, the periodic components are strongly attenuated.



Overall, predictive control offers a performance increase in the telescope guiding framework. The main benefit of using a learning method to identify the periodic error (compared to static periodic error correction in software or hardware) is that the periodic error curve is identified online, without the need for large training datasets, and without the need for synchronization between error curve and hardware.

In addition to the better performance with automatic identification of the periodic error, the predictive control algorithm is able to increase robustness. Since the maximal offset between the guide star and its original position (before loosing it) is only a few pixels, the guide star is often lost if the measurement is failing for several minutes, e.g., when a cloud obscures the guide star. Using predictive control with automatic drift and periodic error identification reduces the distance between guide start and its target position, increasing the probability of finding the guide star after a failure.

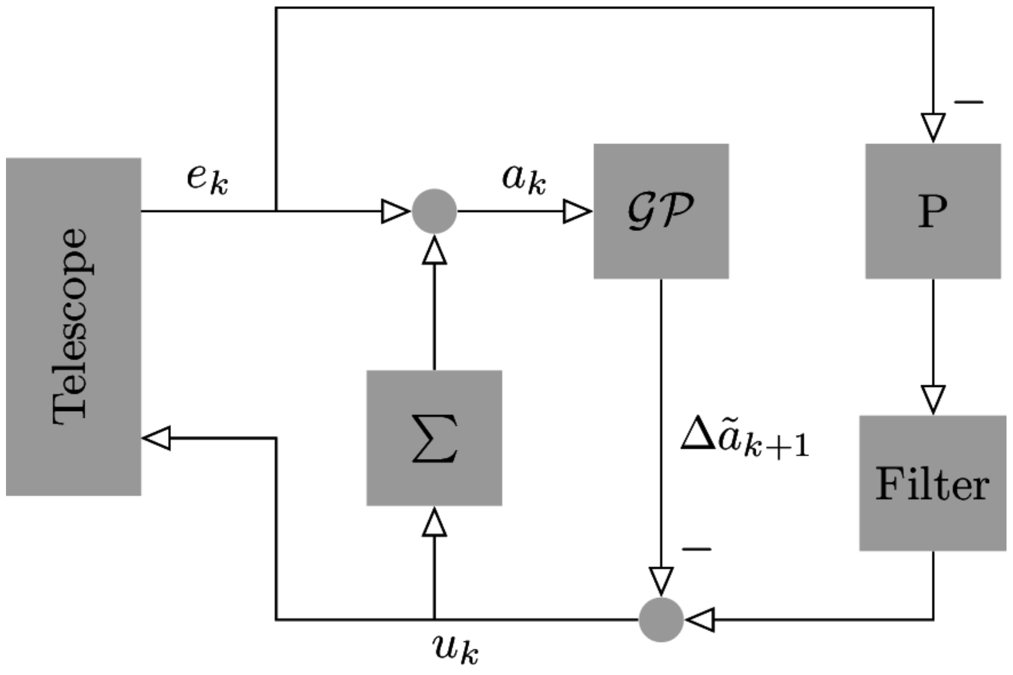
In summary, Gaussian process based predictive control offers:

- Performance increase through predictive PEC
- Robustness increase through predictive drift correction

Data Preprocessing

The identification of the underlying gear error function works much better for the accumulated gear error $a_k = e_k + \sum_{i=0}^{k} u_i$ instead of the absolute error (the control signal u_k is summed up and the current residual error e_k is added on top). This gives better results because the gear error sums up, while random effects like steps of a stepper motor or atmospheric motion cancel out. We train the GP on the accumulated gear error and predict the difference in accumulated gear error for the next time step. The prediction is fed back entirely (unit feedback gain), the current residual error is fed back with a proportional controller.

Overall, the controller structure is as follows



Gaussian Process Regression

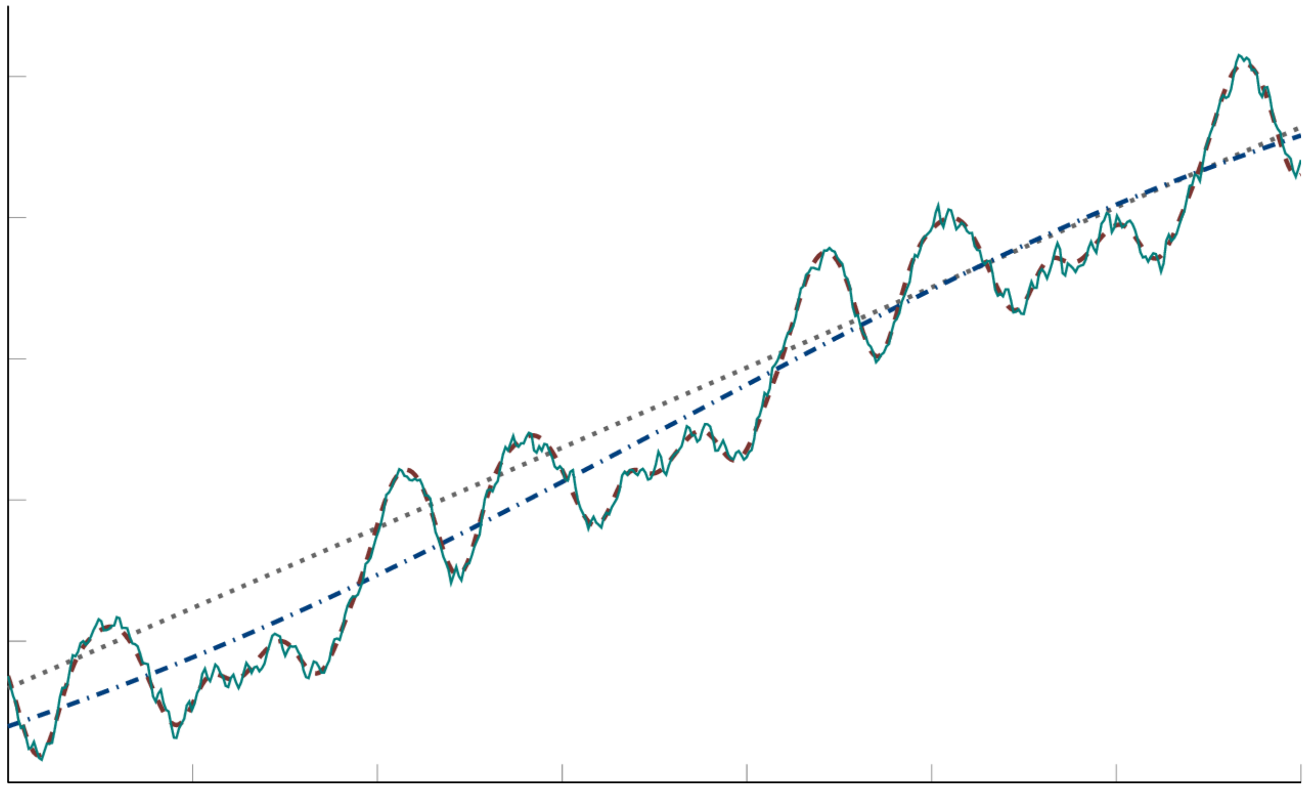
Regression is the task of finding a functional relationship between two variables. For example, the wall clock time and the gear error. If we have such a model for the gear error, we can also use it for predictions by evaluating the function for a time in the future.

Gaussian process regression is a standard technique in machine learning. It is similar to the well-known least-squares regression method, but offers more flexibility in the model. Of course, we cannot give a full introduction Gaussian processes here, and kindly refer to a freely available textbook for the mathematical background.

The model of a GP regression consists of a covariance function k (and, in our case, fixed features ϕ). While the fixed features are exactly used like in least squares regression, where we can define (e.g., polynomial) feature functions, the covariance function is a bit more advanced. The covariance function (also known as kernel function) is a function of two data sets and returns a covariance matrix ($K_{i,j} = k(t_i, t_j) \ \forall t \in T$). It defines pairwise similarities between the input points in the dataset. The shape of the covariance function defines the functions that can be learned by the GP.

Model

Overall, the model consists of four different components that, added together, resemble a typical gear error function:



- Linear drift (dotted, gray) due to drift that is caused by alignment mismatch.
- Slow variations (dash-dotted, blue) due to mechanical deformations from load shift.
- Periodic component (dashed, red) due to the periodic gear error from imprecise cogs.
- Quick variations (solid, green) due to mechanical roughness.

Drift Features

The drift component is as a linear function and outside of the GP, because it is easier to capture polynomial features with a standard linear regression instead of a Gaussian process. This means that we model the overall accumulated gear error as

$$a(t) = [\beta_0 \ \beta_1]^\intercal \begin{bmatrix} 1 \\ t \end{bmatrix} + f(t),$$

where β_0, β_1 are the weights of the linear regression and $f(t)$ is the periodic error, modeled with a Gaussian process.

Kernels

The kernel defines the shape of the functions modeled by the Gaussian process. Since we want to account for different effects, we take the sum over different kernels. This way, we can directly model the different components of the gear error.

The kernel is given by

(Equ. 1)
$$k_c\left(x, x'; \eta\right) = k_{SE,0}\left(x, x'; \eta\right) + k_P\left(x, x'; \eta\right) + k_{SE,1}\left(x, x'; \eta\right)$$

where η is the collection of all parameters and

- $k_{SE,0}$ is a slowly varying square exponential component (long length scale)

$$k_{SE,0}\left(x, x'; \theta_0, \ell_0\right) = \theta_0^2 \exp\left(\frac{\|x - x'\|^2}{2\ell_0^2}\right),$$

  accounting for long-term mechanical deviations (e.g., load shift in the mechanics)

- $k_P\left(x, x'; \theta_P, \ell_P, \lambda\right)$ is the periodic component

$$k_P\left(x, x'; \theta_P, \ell_P, \lambda\right) = \theta_P^2 \exp\left(-\frac{2\sin^2\left(\frac{\pi}{\lambda}\left(x - x'\right)\right)}{\delta(\ell_P)^2}\right),$$

  capturing the recurrent disturbances from the gear. $\delta(\ell_P) = 4\sin(\frac{\pi}{\lambda}\ell_P)$ converts the natural parameter ℓ_P to the hard-to-grasp parameter δ of the covariance function.

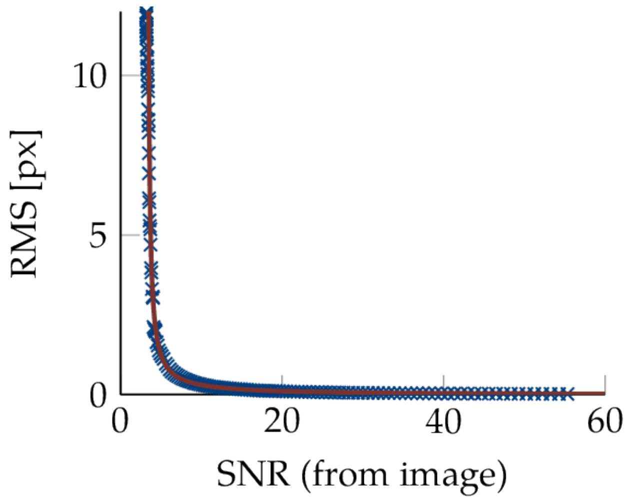
- $k_{SE,1}$ is a quickly varying square exponential component (short length scale)

$$k_{SE,1}\left(x, x'; \theta_1, \ell_1\right) = \theta_1^2 \exp\left(\frac{\|x - x'\|^2}{2\ell_1^2}\right)$$

  accounting for mechanical roughness and seeing noise

Measurement Noise

The noise level of the measurements is not constant. Depending on the signal-to-noise ratio (SNR) of the image, the RMS error of the centroid for the star center measurement varies. With simulated exposures we identified the following relationship between the SNR and the RMS error



which is modeled with linear regression

(Equ. 2)

$$\mathrm{RMS}(\mathrm{SNR}) = w_1 (\mathrm{SNR} - \mathrm{SNR}_0)^{-1} + w_0,$$

where SNR_0 is an offset, and the parameters w_0, w_1 were identified empirically. For each datapoint the estimated RMS error is stored, so that the measurement uncertainty can be considered in the subsequent regression algorithm.

Parameters

The usefulness of the regression framework strongly depends on the correct identification of the periodicity parameter λ (the frequency / period length of the periodic error). This parameter is identified by frequency analysis on the measurements. Since usual frequency analysis tools only work well when the data is sampled in a regular grid, we regularize the data by interpolation and averaging. Additionally, we remove the linear trend from the data so that there is no jump from the beginning to the end and apply a windowing function (Hamming window). This eliminates certain artifacts of the Fourier analysis.

After applying a fast Fourier transform (FFT) on the data, the strongest component is identified by sorting the Fourier coefficients. From the according frequency the period length is calculated. In order to increase precision, the data vector is zero-padded before the FFT and quadratic interpolation is applied after the FFT. In order to prevent jumps in the period length parameter, the updates work with a learning rate. When the guiding is stopped, the current value of the period length is stored to the configuration, so that the guiding always starts with the value that was identified in the last run.

All other parameters can be estimated in advance by physical reasoning and optimization of real-world data sets. They only need to be changed (if ever) if the user has a telescope setup which is very different to what is used on average. The method usually copes well with parameters that are a bit off, except for the period length.

Symbol	Variable	Position	Meaning	Explanation
ℓ_0	SE0KLengthScale_	`hyperparameters[0]`	length scale, long SE	Defines how quickly the long-term mechanical deviations change.
θ_0	SE0KSignalVariance_	`hyperparameters[1]`	signal variance, long SE	Defines how much the mechanics deviates due to load shift and other long term effects.
ℓ_p	PKLengthScale_	`hyperparameters[2]`	length scale, periodic	Defines how quickly the small structure within one period of the periodic error is allowed to change.
θ_p	PKSignalVariance_	`hyperparameters[3]`	signal variance, periodic	Defines the variance of the periodic gear error.
ℓ_1	SE1KLengthScale_	`hyperparameters[4]`	length scale, short SE	Defines how quickly the mechanical roughness changes.
θ_1	SE1KSignalVariance_	`hyperparameters[5]`	signal variance, short SE	Defines how much mechanical roughness we expect.
λ	PKPeriodLength_	`hyperparameters[6]`	period length	The period length of the periodic gear error.

Currently all these parameters are exposed to the user, but it should be possible to reduce the parameter number significantly after testing.

Inference

Standard Equations

In the standard case, the Gaussian process mean prediction y at a time t, based on a set of data points Y at times T, is done according to

$$y = f(t) = K_{tT}(K_{TT} + S)^{-1} Y,$$

where $K_{tT} = k(t, T)$ is the kernel function evaluated for the prediction time t and all data inputs T, $K_{TT} = k(T, T)$ is the kernel matrix obtained by evaluating the covariance function for all data input pairs, and $S = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_N^2)$ is a diagonal matrix of the measurement noise values.

In our particular case, we wish not to predict the small component $k_{SE,1}$; therefore the first part of the equation, k_{tT}, is evaluated for the other kernels only:

$$y = f(t) = K^{\star}_{tT}(K_{TT} + S)^{-1} Y,$$

where $K^{\star}_{tT}$ is calculated with the reduced kernel $k^{\star}$ consisting of $k_{SE,0}$ and k_P, while the regular K_{TT} is evaluated with the full kernel k cons isting of $k_{SE,0}$, k_P and $k_{SE,1}$.

Also, since we use fixed feature functions for offset and linear drift, the prediction needs to be modified to

(Equ. 3)
$$y = f(t) = K^{\star}_{tT}(K_{TT} + S)^{-1}Y + \bar{\beta}^{\intercal} R_t,$$

where $\bar{\beta} = (\Phi K^{\intercal}_{TT}\Phi^{\intercal})^{-1}\Phi K^{-1}_{TT} Y$, where $\Phi = \phi(T)$ is the linear regression feature matrix evaluated at the input points, and $R_t = \phi(t) - \Phi K^{-1}_{TT} K^{\star}_{Tt}$.

In code, the inference algorithm looks like the following, where the whole algorithm was collected from different functions in `gaussian_process.cpp`

```
// The data covariance matrix
Eigen::MatrixXd data_cov = covFunc_->evaluate(data_loc_, data_loc_); //
this computes kTT

// compute and store the Gram matrix
gram_matrix_.swap(data_cov); // store the covariance matrix in the GP
instance
gram_matrix_ += data_var_.asDiagonal(); // add the diagonal noise matrix S

// compute the Cholesky decomposition of the Gram matrix
chol_gram_matrix_ = gram_matrix_.ldlt(); // the Cholesky decomposition is
used for matrix inversion

// pre-compute the alpha, which is the solution of the chol to the data.
alpha_ = chol_gram_matrix_.solve(data_out_); // this is an intermediate
result

feature_vectors_ = Eigen::MatrixXd(2, data_loc_.rows()); // allocate
vectors for Phi
// precompute necessary matrices for the explicit trend function
feature_vectors_.row(0) = data_loc_.array().pow(0);
feature_vectors_.row(1) = data_loc_.array().pow(1);

feature_matrix_ = feature_vectors_ *
chol_gram_matrix_.solve(feature_vectors_.transpose());
chol_feature_matrix_ = feature_matrix_.ldlt();

beta_ = chol_feature_matrix_.solve(feature_vectors_) * alpha_; //
intermediate result

// The prior covariance matrix (evaluated on test points)
Eigen::MatrixXd prior_cov = covFunc->evaluate(locations, locations); //
this is ktt

// The mixed covariance matrix (test and data points)
Eigen::MatrixXd mixed_cov = covFunc->evaluate(locations, data_loc_); //
evaluated with the reduced kernel

Eigen::MatrixXd phi(2, locations.rows());
// calculate the feature vectors for linear regression
phi.row(0) = Eigen::MatrixXd::Ones(1,locations.rows()); // locations.pow(0)
phi.row(1) = locations.array(); // locations.pow(1)

Eigen::VectorXd m = mixed_cov * alpha_; // that's the standard prediction

Eigen::MatrixXd R = phi - feature_vectors_ *
chol_gram_matrix_.solve(mixed_cov.transpose());
m += R.transpose() * beta_; // here we add the effect of the linear trend
```

Subset-of-Data Approximation

It is neither feasible nor necessary to keep all measurements as data points. 2-3 revolutions of the gear is enough to have accurate predictions of the periodic error. Therefore, a two-fold approach is used to limit the amount of data points for the FFT parameter identification and for the Gaussian process inference scheme.

1. The data points are stored in a circular buffer, which is also known as first-in-first-out (FIFO) buffer. The circular buffer class is part of PHD2 guiding and is used to store the data objects for the GP guiding. The number of points can be configured in the code with the constant `CIRCULAR_BUFFER_SIZE` and should be chosen such that at least two periods of the periodic error are covered. Ideally this number should be a power of two, otherwise the FFT is zero-padded to the nearest power of two.
2. The GP inference is more demanding than the FFT, therefore it is necessary to reduce the number of data points further. A simple and computationally cheap method is the Subset-of-Data approximation: For the inference, only a portion of the data points is used. For deciding which data points to include, the covariance between the prediction point and all data points is evaluated. The covariance vector is sorted so that only the most important data points are included in the algorithm. The number of data points is currently exposed in the configuration window as `points_for_approximation`.

The sorting code is a bit tricky because it is important to know the indices of the selected elements and not only their value. The idea is to create a vector of integer indices and then sort them according to their associated value. This is achieved by implementing a functor:

```
// A functor for special orderings
struct covariance_ordering
{
    covariance_ordering(Eigen::VectorXd const& cov) : covariance_(cov){}
    bool operator()(int a, int b) const
    {
        return (covariance_[a] > covariance_[b]);
    }
    Eigen::VectorXd const& covariance_;
};
```

This is basically a "greater than" function that does not compare the function arguments (a and b) but instead the elements of the stored covariance vector. The actual sorting is done as follows:

```
// calculate covariance between data and prediction point for point
selection
covariance = covFunc_->evaluate(data_loc, prediction_loc);

// generate index vector
std::vector<int> index(covariance.size(), 0);
for (int i = 0 ; i != index.size() ; i++) {
    index[i] = i;
}

// sort indices with respect to covariance value
std::sort(index.begin(), index.end(),
    covariance_ordering(covariance)
);
```

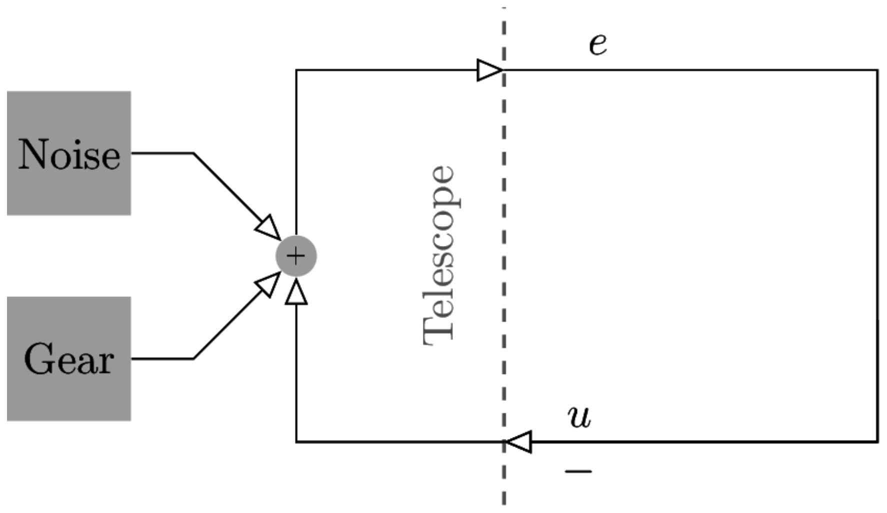
When we now use the first `M` elements of the index vector, we can select the `M` elements with the highest covariance from the dataset.

Feedback Control

Deviations from the target position are tackled by applying negative feedback control to the telescope hardware. In PHD2 guiding the measurements and actions are normalized in the calibration phase.

Deadbeat Control

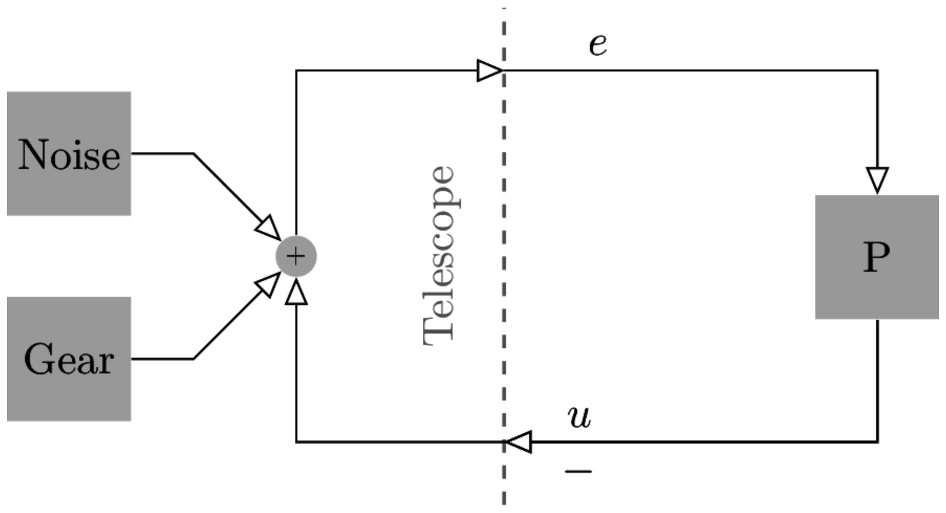
One control model is 1-step optimal control, also known as "deadbeat control" and is supposed to reduce the error to zero in one time step. In PHD2 guiding, because of the normalization, this means that 100 % of the error are fed back (100 % "aggressiveness"):



This controller is implemented in the `None` guiding method and often leads to overshoot and amplification of atmospheric noise.

Proportional Control

The common scheme for reducing the error in a sensible way, implementing in all guiding algorithms except `None` and `Gaussian Process` so far, is to use proportional control:



The error signal is multiplied with a proportional feedback gain. Since the control actions are normalized, this feedback gain ("aggressiveness") must be between 0 and 1. Usually a value of 0.5 to 0.7 is used; there is a tradeoff between disturbance rejection and noise amplification. Also, the P-controller cannot deal well with constant offsets: In the case of drift, the error never is reduced to zero in the long run.

Pausing and Dithering

Pausing

Every few minutes, the guiding is stopped for refocusing the telescope. Since the GP guider provides best performance when the history of collected data points is longer than two revolutions of the gear, it is necessary to keep the algorithm running during the times of pausing. Therefore, blind guiding is activated by calling `deduceResult` whenever the system is in pause state. Since the GP guider has a predictive model of the gear, even in the pausing phase control commands are issued. However, since no measurements are made during this time, the data points (that are generated from the control signal only) are considered to have very high noise, so that the inference algorithm does not use them.

Dithering

Every few minutes, the target position of the guiding is changed by an arbitrary amount of pixels to make sure pixels with high noise or wrong values (e.g., hot pixels) are at different positions of the target image when stacking multiple subframes. During the settling period, where only P-control is applied in order to move the telescope to the new target position, the telescope is operated in dark tracking mode. However, for the large dither steps, PHD2 applies direct mount movements which are not seen by the guiding algorithm. Therefore, there will be a rather large time step during dithering, but this should not disturb the algorithm, since the measurements have very high noise. The only important aspect is that the last dithering step should always be carried out by the PPEC algorithm so that the time difference between the last regular guide step and the last dithering step is covered.

Hysteresis Blending

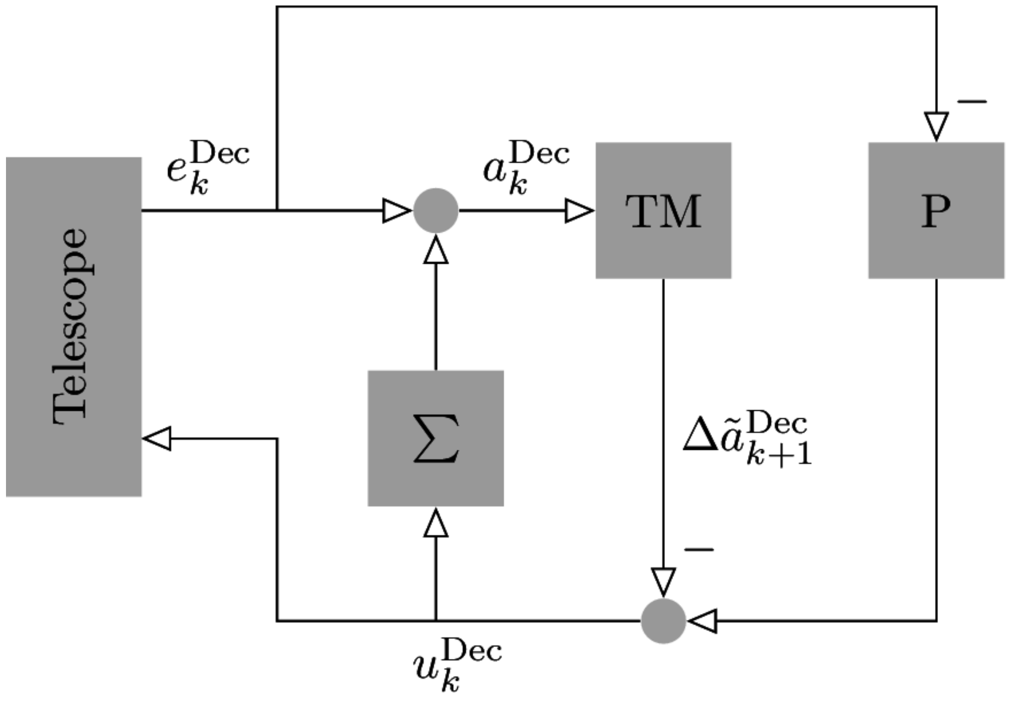
The Hysteresis guiding algorithm is very successful in providing good performance for a variety of mounts. We use this fact to start guiding with Hysteresis, where we use a fixed hysteresis parameter of 0.1 and the aggression and min move parameters from Predictive PEC. Over the course of two period lengths the system smoothly blends over to use the Predicitve PEC algorithm. This increases performance especially in the initial phase where there isn't enough information available to properly predict the gear error.

Declination Guiding

Note: Declination guiding ("Predictive Drift") was removed from the first release of the GP Guiding feature. Until it is added back to the code, this section is not relevant.

In order to increase robustness for finding the guide star after a passing cloud, it is important that the declination guider has a predictive control feature as well. However, standard approaches like linear regression or robust regression often fail when there are strong nonlinear effects like gear backlash. Therefore, we implemented a robust drift estimation technique that is resilient to even strong gear backlash effects. This comes at the price of slow controller convergence in the estimation phase. In order to alleviate the slow convergence, we implemented the Dec guiding with a PD controller instead of a pure P controller.

The overall controller structure is:



where Σ is a cumulative sum, TM is a trimmed mean estimator and P is a P-controller.

Algorithm

Similar to the RA axis, the accumulated gear error is calculated for the Declination. On the finite difference between the accumulated gear errors, a trimmed mean is evaluated. This means that the vector is sorted, the highest and lowest quantiles are dropped and a mean is calculated for the remaining values. This essentially is a compromise between the mean (not robust to outliers) and the median (no averaging). The trimmed mean is a good approximation of the true drift, but it requires many measurements.

With the prediction from the drift, a control scheme similar to the RA axis is applied: The predicted error is fully compensated, while a P controller is used to drive the residual error to zero.

In addition to the predictive and feedback control, gear backlash is prevented in a radical way: The controller is not allowed to issue control actions into the direction of the drift. It is assumed that small errors to this direction are compensated by the drift over time, and errors in the other direction can safely be corrected with a feedback controller (no switching can occur).