

# Kalman Filter Tuning with Bayesian Optimization

Zhaozhong Chen, Nisar Ahmed, Simon Julier, and Christoffer Heckman

**Abstract**—Many state estimation algorithms must be *tuned*: given the state space process and observation models, the process and observation noise parameters must be chosen. Conventional tuning approaches rely on heuristic hand-tuning or gradient-based optimization techniques to minimize a performance cost function. However, the relationship between tuned noise values and estimator performance is highly nonlinear and stochastic. Therefore, the tuning solutions can easily get trapped in local minima, which can lead to poor choices of noise parameters and suboptimal estimator performance. This paper describes how Bayesian Optimization (BO) can overcome these issues. BO poses optimization as a Bayesian search problem for a stochastic “black box” cost function, where the goal is to search the solution space to maximize the probability of improving the current best solution. As such, BO offers a principled approach to optimization-based estimator tuning in the presence of local minima and performance stochasticity. While extended Kalman filters (EKF) are the main focus of this work, BO can be similarly used to tune other related state space filters. The method presented here uses performance metrics derived from normalized innovation squared (NIS) filter residuals obtained via sensor data, which renders knowledge of ground-truth states unnecessary. The robustness, accuracy, and reliability of BO-based tuning is illustrated on practical nonlinear state estimation problems, closed-loop aero-robotic control.

**Index Terms**—Kalman filtering, filter tuning, Bayesian optimization, nonparametric regression, machine learning.

## I. INTRODUCTION

**M**ANY state estimation algorithms, including Kalman filters and particle filters, are recursive and model-based [1], [2]. They decompose the estimation problem into a cycle with two main steps: *state prediction* followed by *measurement update*. The state prediction step uses a process model to predict how the state evolves over time. The measurement update step uses an observation model to relate a measured quantity to the state estimate. Since both the process and observation models are imperfect, errors in these models are treated as random noise terms that are injected into the system. Most designs assume the noises are white, zero mean and uncorrelated. As a result, filter tuning consists of choosing the values of the process and observation noise covariances.

Many tuning procedures adopt a divide-and-conquer strategy. The first stage is to choose the observation covariance. This is normally carried using laboratory or bench testing. The sensor is placed in a condition in which the noise-free sensor values can be predicted. The observation noises are determined by statistically characterizing the difference between the predicted and actual values. In the second stage, the process noises are chosen. Since the process noises contain information about the state disturbances and dynamic model uncertainties, which often cannot be reproduced in laboratory settings, the covariance is often chosen by collecting data from an operational domain and quantifying the quality of

the estimates. Typically a performance cost is assigned, and the process noise covariance adjusted to minimize the value of that cost.

However, there are several problems with this two-stage approach. First, with laboratory testing, it is not always possible to model how sensors will react in operational environments. Changes in temperature, for example, can cause the biases in IMUs to change. As a result, the observation noises might not be properly characterized. Second, the interaction between noise levels and filter performance is not straightforward. Theoretical analysis has shown that even if the process and observation models are linear, the presence of modeling errors lead to noises which are state-dependent and correlated over time [3], [4]. As a result, many non-unique tuning solutions can appear. Finally, they tend to rely on statistics which require knowledge of the ground truth of the system to compute. Although this is possible to obtain in simulations or laboratory settings with precise reference measurement systems, such approaches are of limited use for many practical real-world applications where ground truth data is not available.

This paper makes three contributions, which significantly extends the preliminary work presented in [5]. The first is a general framework for Kalman filter noise parameter tuning based on Bayesian Optimization (BO). BO provides an attractive way to overcome the limitations of other gradient-based optimal filters auto-tuning strategies, which can easily get trapped in poor local minima. The BO framework developed here uses nonparametric surrogate models based on Student-t process regression, which offers better robustness and performance compared to Gaussian process surrogate models that are more typically used for Bayesian optimization and which were considered in [5].

The second contribution is the development and validation of novel stochastic cost functions for optimization-based auto-tuning. Most auto-tuning algorithms including our previous work [5] use the Normalized Estimation Error Squared (NEES) which requires the ground truth values of the state. We may not be able to obtain the ground truth easily in the real world, in this paper, our approach extends the previous work and uses Normalized Innovation Squared (NIS), which is computed from the difference between the predicted and actual sensor measurements and does not require ground truth. This makes our approach practical for many real-world applications, where only the sensor observations are available for filter validation.

Finally, the performance of the BO auto-tuning framework is demonstrated and evaluated in simulation for a challenging application: longitudinal state estimation for the Mars Science Laboratory (MSL) Skycrane platform. The results for this application shows that BO not only provides reliable and

computationally efficient estimates of unknown filter parameters, but can also provide useful probabilistic information about each parameter through the whole domain space, which existing state-of-the-art auto-tuning methods cannot do. While this work focuses mainly on auto-tuning of extended Kalman filters (EKFs), the BO framework can be readily extended to other related state space filtering algorithms.

The rest of this paper is structured as follows. Sections II and III formally introduces an overview and problem statement for filter auto-tuning. Section IV describes our Bayesian optimization framework using nonparametric Student-t process regression and how it is applied to Kalman filter parameter auto-tuning. Section V describes the set up, and analysis for the numerical simulation studies using the Bayesian optimization framework to auto-tune EKFs for the Mars Skycrane longitudinal state estimation problem. Conclusions are given in Section VI.

## II. PRELIMINARIES

### A. System Overview

The state of the system at time step  $k$  is  $\mathbf{x}_k \in \mathbb{R}^{n_x}$ , where  $n_x$  is the dimension of the state vector. The process model that propagates the state from  $k-1$  to  $k$  is

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{v}_k) \quad (1)$$

where  $\mathbf{u}_k \in \mathbb{R}^{n_u}$  is the control input vector and  $\mathbf{v}_k \in \mathbb{R}^{n_v}$  is the process noise.

The observation model is

$$\mathbf{z}_k = h(\mathbf{x}_k, \mathbf{w}_k) \quad (2)$$

where  $\mathbf{z}_k \in \mathbb{R}^{n_z}$  is the observation vector and  $\mathbf{w}_k \in \mathbb{R}^{n_w}$  is the measurement noise.

It is typically assumed that the process and observation models are sufficiently accurate that the process and observation noises are zero-mean, independent, Gaussian distributed random variables.

Given the structure of the system, the goal is to develop an estimation algorithm which takes in a sequence of observations and control inputs, and computes an estimate of the state. The errors in initial conditions, together with the process and observation noises, means that the state is not known perfectly. Therefore, some means of quantifying the uncertainty must be used. A common approach is to use the mean and covariance of the state estimate.

### B. Mean and Covariance Representation

Our goal is to estimate the state of a random variable  $\mathbf{x}_i$  at a discrete time  $i$  and quantify the uncertainty  $\mathbf{P}_{i|i}$  in that estimate. Let  $\hat{\mathbf{x}}_{i|j}$  be the estimate of  $\mathbf{x}_i$  using all observations up to time step  $j$ , and the covariance of this estimate be  $\mathbf{P}_{i|j}$ :

$$\hat{\mathbf{x}}_{i|j} = \mathbb{E}[\mathbf{x}_i | \mathbf{z}_{1:j}] \quad (3)$$

$$\mathbf{P}_{i|j} = \mathbb{E} \left[ (\mathbf{x}_i - \hat{\mathbf{x}}_{i|j}) (\mathbf{x}_i - \hat{\mathbf{x}}_{i|j})^\top | \mathbf{z}_{1:j} \right]. \quad (4)$$

However, computing an estimate which obeys this property in practice is difficult to achieve. Modelling errors, for example, can always lead to biased estimates. Therefore, most practical systems use a weaker condition called *covariance consistency* [6]. In this case, a valid estimate has the properties:

$$\hat{\mathbf{x}}_{i|j} \approx \mathbb{E}[\mathbf{x}_i | \mathbf{z}_{1:j}] \quad (5)$$

$$\mathbf{P}_{i|j} \geq \mathbb{E} \left[ (\mathbf{x}_i - \hat{\mathbf{x}}_{i|j}) (\mathbf{x}_i - \hat{\mathbf{x}}_{i|j})^\top | \mathbf{z}_{1:j} \right]. \quad (6)$$

where  $\approx$  is application-specific and  $\mathbf{A} \geq \mathbf{B}$  means that  $\mathbf{A} - \mathbf{B}$  is positive semidefinite. In other words, the estimate should be approximately unbiased, and the estimator should not over estimate its level of confidence. At the same time, we would like the difference between the predicted covariance and actual mean squared error to be as small as possible.

### C. Kalman Filters

The Kalman filter is one of the best known and most widely used algorithms for state estimation. It is derived from the fact that the correction applied to the estimate is a linear rule of the form

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \mathbf{e}_{\mathbf{z},k},$$

where  $\mathbf{K}_k$  is a gain matrix, and

$$\mathbf{e}_{\mathbf{z},k} = \mathbf{z}_k - \hat{\mathbf{z}}_{k|k-1},$$

which is the difference between the actual and predicted sensor measurement, is the innovation vector. It acts as an error signal in the filter, and provides a correction term for the state estimate. The Kalman filter chooses the value of  $\mathbf{K}_k$  to minimize the mean squared error in  $\hat{\mathbf{x}}_{k|k}$ .

The algorithm proceeds as follows [7]. The state is predicted according to the equation

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \quad (7)$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\top + \mathbf{Q}_k. \quad (8)$$

The update is calculated from

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \mathbf{e}_{\mathbf{z},k}, \quad (9)$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}, \quad (10)$$

$$\mathbf{S}_{k|k-1} = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k, \quad (11)$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_{k|k-1}^{-1}, \quad (12)$$

where  $\mathbf{F}_k$  and  $\mathbf{H}_k$  are the Jacobian matrices of the process and observation models.

$$\mathbf{F}_k = \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k} \quad (13)$$

$$\mathbf{H}_k = \left. \frac{\partial h}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k|k-1}} \quad (14)$$

Note that once Eq. (1) and (2) have been chosen, the only degree of freedom left is to choose  $\mathbf{Q}_k$  and  $\mathbf{R}_k$ . This process is known as tuning.

### III. TUNING

As explained in the introduction, tuning involves choosing  $\mathbf{Q}_k$  and  $\mathbf{R}_k$  to minimize some performance cost. Two widely used measures are the *normalized estimation error squared (NEES)* and the *normalized innovation error squared (NIS)*. These are computed from

$$\epsilon_{\mathbf{x},k} = \mathbf{e}_{\mathbf{x},k}^T \mathbf{P}_{k|k}^{-1} \mathbf{e}_{\mathbf{x},k} \quad (15)$$

$$\epsilon_{\mathbf{z},k} = \mathbf{e}_{\mathbf{z},k}^T \mathbf{S}_{k|k-1}^{-1} \mathbf{e}_{\mathbf{z},k} \quad (16)$$

where  $\mathbf{e}_{\mathbf{x},k} = \hat{\mathbf{x}}_{k|k} - \mathbf{x}_k$  and  $\mathbf{e}_{\mathbf{z},k} = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})$  is the innovation vector. If the dynamical consistency conditions are met, then

$$\mathbb{E}[\epsilon_{\mathbf{x},k}] \approx n_{\mathbf{x}} \quad (17)$$

$$\mathbb{E}[\epsilon_{\mathbf{z},k}] \approx n_{\mathbf{z}}. \quad (18)$$

It is often assumed that the prediction and observation errors are Gaussian. In this case,  $\epsilon_{\mathbf{x},k}$  and  $\epsilon_{\mathbf{z},k}$  will be  $\chi^2$ -distributed random variables with  $n_{\mathbf{x}}$  and  $n_{\mathbf{z}}$  degrees of freedom respectively [7]. Therefore,  $\chi^2$  hypothesis tests can be performed on calculated values for  $\epsilon_{\mathbf{x},k}$  (when ground truth data is available) and  $\epsilon_{\mathbf{z},k}$  to see if the consistency conditions hold at each time  $k$ .

#### A. Approaches to Tuning

In this paper we focus on the process noise tuning because it is the hardest to tune in the Kalman filter. In practice, NEES  $\chi^2$  tests are conducted using multiple offline Monte Carlo “truth model” simulations to obtain ground truth  $\mathbf{x}_k$  values. The truth model simulator represents a high-fidelity model of the “actual” system dynamics and sensor observations, which may contain nonlinearities and other non-ideal characteristics that must be compensated for via Kalman filter tuning. NIS  $\chi^2$  can be conducted offline using multiple Monte Carlo simulations (e.g. in parallel with NEES tests), but can also be conducted online using real-time sensor data.

Online/offline NIS tests are conducted as follows<sup>1</sup>: suppose  $N$  independent instances of the true state are randomly initialized according to  $\hat{\mathbf{x}}_{0|0}$  and  $\mathbf{P}_{0|0}$  (the initial state of the filter), and then propagated through the true stochastic dynamics (1) and measurement model (2) for  $T$  time steps, yielding sample ground truth sequences  $\mathbf{x}_1^i, \mathbf{x}_2^i, \dots, \mathbf{x}_T^i$  and measurement sequences  $\mathbf{z}_1^i, \mathbf{z}_2^i, \dots, \mathbf{z}_T^i$  for  $i = 1, \dots, N$ . If the resulting measurement sequences are then fed into a Kalman filter with tuning parameters  $(\mathbf{Q}_k, \mathbf{R}_k)$ , the resulting NEES and NIS statistics for each simulation run  $i$  at each time  $k$  can be averaged across problem instances to give the test statistics:

$$\bar{\epsilon}_{\mathbf{x},k} = \frac{1}{N} \sum_{i=1}^N \epsilon_{\mathbf{x},k}^i \quad (19)$$

$$\bar{\epsilon}_{\mathbf{z},k} = \frac{1}{N} \sum_{i=1}^N \epsilon_{\mathbf{z},k}^i. \quad (20)$$

<sup>1</sup>This is the same as the offline truth model tests conducted in [5]; here we still use ground truth in order to check if the filter is consistent but in practice it is not required.

Then, given some desired Type I error rate  $\alpha$ , the NEES and NIS  $\chi^2$  tests provide lower and upper tail bounds  $[l_{\mathbf{x}}(\alpha, N), u_{\mathbf{x}}(\alpha, N)]$  and  $[l_{\mathbf{z}}(\alpha, N), u_{\mathbf{z}}(\alpha, N)]$ , such that the Kalman filter tuning is declared to be consistent if, with probability  $100(1 - \alpha)$  at each time  $k$ ,

$$\bar{\epsilon}_{\mathbf{x},k} \in [l_{\mathbf{x}}(\alpha, N), u_{\mathbf{x}}(\alpha, N)],$$

$$\bar{\epsilon}_{\mathbf{z},k} \in [l_{\mathbf{z}}(\alpha, N), u_{\mathbf{z}}(\alpha, N)].$$

Otherwise, the filter is declared to be inconsistent. Specifically, if  $\bar{\epsilon}_{\mathbf{x},k} < l_{\mathbf{x}}(\alpha, N)$  or  $\bar{\epsilon}_{\mathbf{z},k} < l_{\mathbf{z}}(\alpha, N)$ , then the filter tuning is “pessimistic” (underconfident), since the filter-estimated state error/innovation covariances are too large relative to the true values. On the other hand, if  $\bar{\epsilon}_{\mathbf{x},k} > u_{\mathbf{x}}(\alpha, N)$  or  $\bar{\epsilon}_{\mathbf{z},k} > u_{\mathbf{z}}(\alpha, N)$ , then the filter tuning is “optimistic” (overconfident), since the filter-estimated state error/innovation covariance are too small relative to the true values.

The  $\chi^2$  consistency tests provide a very principled basis for validating Kalman filter performance in domain-agnostic way, and also provide a well-established means for guiding the tuning of noise parameters  $\mathbf{Q}_k$  and  $\mathbf{R}_k$  in practical applications. Tuning via the  $\chi^2$  tests is most often done manually, and thus requires repeated “guessing and checking” over multiple Monte Carlo simulation runs. However, this quickly becomes cumbersome and non-trivial for systems with several tunable noise terms. Heuristics for manual filter tuning have been developed in the linear-quadratic optimal control literature [8], e.g. to coarsely tune diagonals of  $\mathbf{Q}_k$  first, before fine-tuning the elements of  $\mathbf{Q}_k$  further. Such heuristics are useful for bounding the shape and magnitude of  $\mathbf{Q}_k$  in linear-Gaussian problems, but are of little help for tuning ‘fudge factor’ process noise parameters that are used to cope with model errors from state truncation, approximations of non-linearities, poorly modeled dynamics, etc. Given this, alternative optimization techniques are needed which are robust to stochastic variations in the cost function and which can explore nonlinear spaces while also satisfying the filter consistency requirements.

#### B. Previous Tuning Work

Much of the previous Kalman filter auto-tuning work is based on consistency checking ([9] and [10]).

Reference [9] uses a genetic algorithm to tune Kalman filter. This algorithm simulates the Darwin concept of “survival of the fittest” to choose a good parameter set. It treats each parameter set in the parameter space as an “individual.” The specific parameter value corresponding to that individual is coded into a string as a binary value and treated as a “chromosome,” which is the genetic information. The fittest individual is selected according to the numerical value of NEES and error covariance norm, which is the cost function. The genetic algorithm is implemented after some modifications. First, random parameter sets in the parameter space are selected as the initial “population.” They will spawn the next generation by pairing two individuals and exchanging parts of their chromosomes randomly. The population is believed to have converged once the population has a low cost. In this approach, a large number of Monte Carlo runs is not

used because of computation limits, which leads to a problem that some wrong individuals may also be able to pass the consistency test. They add one more option to the cost function besides the consistency test to solve that problem: when the consistency value is smaller than a threshold the cost function switches to a value based on the norm of the error covariance  $\mathbf{P}$ .

In their simulation experiment, they use a simple oscillator as an example, aiming at tuning the speed and position noise. The optimal value is not achieved because the structure of their cost function: the minimization routine will tend to have smaller state error covariance norm and instead of smaller consistency value.

Another previous work [10] uses downhill simplex numerical optimization algorithm to minimize the NEES based cost function. A simplex is a collection of  $N + 1$  points in an  $N$ -dimensional space and all their interconnecting line segments. The simplex algorithm attempts to locate a minimum of the function by a series of movements in the  $N$ -dimensional space. Those movements include reflection, expansion and contraction. Details of those movements can be seen in the paper [10].

However, the simplex algorithm can easily be stuck in a local minima so there may be cases that this method will fail. Although the algorithm's cost function is based on NEES, there are no plots showing the consistency check after getting a tuning result.

Our previous work [5] focus on Kalman filter tuning using NEES  $\chi^2$  tests too. Due to the hardware improvements these years, it is not that time consuming to implement a large number of Monte Carlo tests consisting of, say, several hundred runs. We simulated a car moving along a straight line and optimized a two dimensional process noise and a one dimensional measurement noise. We successfully showed that use Bayesian optimization to tune Kalman filter process noise covariance as well as measurement noise covariance and can yield good results. However, in our previous work, we did not perform formal post hoc consistency validation checks to confirm the readers that the error at each timestamp is small enough. Our previous work also only limited analysis and application to a linear dynamical system, and did not consider extensions to linearization-based approximations for non-linear filtering. At the same time, the above mentioned references and our prior work [5] use NEES based consistency check method, which makes it impossible to use when the ground truth is not available. In this paper, we also propose to use a NIS based consistency check method. NIS based tuning method makes it possible for us to tune the Kalman filter with just sensor measurements. We apply our methods on more complicated and practical nonlinear cases and also validate statistical consistency of the optimized result. Finally, in this paper we use an improved Bayesian optimization procedure which is based on nonparametric Student's-t regression models, which leads to significantly more robust surrogate models and tuning solutions than the Gaussian processes (GPs) regression models used in our prior work.

### C. Summary

Problems with existing approaches are that (a) they fall into local minima; (b) they often have to use NEES; (c) they run into issues with noise and stochastic variation from small finite number of MC runs. We use Bayesian optimization to avoid falling into local minima and we use NIS to avoid using groundtruth.

## IV. BAYESIAN OPTIMIZATION FOR AUTO TUNING

Many approaches for solving nonlinear optimization problems use gradient descent. However, the risk with these approaches is that they can fall into local minima. This issue is exacerbated for filter tuning problems defined by noisy nonlinear dynamical systems. Stochastic variations and nonlinear model characteristics can introduce many local minima into objective functions for tuning that can trap gradient descent methods. One principled way to handle such cases is to use *Bayesian optimization* [11], which poses optimization as a probabilistic search problem.

Bayesian optimization is first described for dealing with generic "black box" stochastic objective functions. Its novel application is then described for simulation-based Kalman filter auto-tuning.

### A. Bayesian Optimization Theory

Consider the minimization of an objective function  $y : \mathcal{Q} \rightarrow \mathbb{R}$ , where  $\mathcal{Q} \in \mathbb{R}^d$  is the search or solution space, and  $\mathbf{q}^* \in \mathcal{Q}$  is the minimizer, such that  $y(\mathbf{q}^*) \leq y(\mathbf{Q})$ ,  $\forall \mathbf{Q} \in \mathcal{Q}$ . Furthermore, we assume that each element of  $i$  of  $\mathbf{q}$  lies in the interval  $\mathcal{Q}(i) \in [\mathbf{q}(i)_l, \mathbf{q}(i)_u]$ .

The intuition behind BO arises from the following. First suppose that the entire solution space were densely sampled. Carrying out this process, the map  $y : \mathcal{Q} \rightarrow \mathbb{R}$  is entirely known and the minimizer can be read off directly. However, this dense sampling scheme is not possible in practice. Rather, the search algorithm samples a subset of the parameter space. Since the sampling is incomplete, the shape of the cost surface is not known but, rather, must be estimated from sparse and incomplete data. Therefore, the goal of Bayesian optimization is to find the minimizer of the noisy objective function  $y$  while at the same time learning about the mapping from  $\mathbf{q}$  to  $y$  via Bayesian inference. Bayesian optimization uses "black box" point evaluations of  $y$  to efficiently find  $\mathbf{q}^*$ . This is accomplished by maintaining beliefs about how  $y$  behaves over all  $\mathbf{q}$  in the form of a surrogate model  $\mathcal{S}$ , which statistically approximates  $y$  and is easier to evaluate (e.g. since evaluations of  $y$  might require expensive high-fidelity simulation). During optimization,  $\mathcal{S}$  is used to determine where the next design point sample evaluation of  $y$  should occur, in order to update beliefs over  $y$  and thus simultaneously improve  $\mathcal{S}$ , while finding the (expected) minimum of  $y$  as quickly as possible. The key idea is that, as more observations  $E$  are sampled at different  $\mathbf{q}$  locations, the  $\mathbf{q}$  samples themselves eventually converge to the expected minimizer  $\mathbf{q}^*$  of  $y$ . Since  $\mathcal{S}$  contains statistical information about the level of uncertainty in  $y$  (i.e. related to the posterior belief  $p(y|E)$ ), Bayesian optimization

effectively leverages probabilistic “explore-exploit” behavior to learn a probabilistic model of  $y$  while also minimizing it.

We next describe the two main components of the Bayesian optimization process: (1) the surrogate model  $\mathcal{S}$ , which encodes statistical beliefs about  $y$ ; and (2) the acquisition function  $a(\mathbf{q})$ , which is used to intelligently guide the search for  $\mathbf{q}^*$  via  $\mathcal{S}$ .

1) *Surrogate Model*: The surrogate model is the model used to approximate the objective function. In the BO literature, nonparametric regression models based on stochastic processes are widely used [12] because they naturally model probability distributions over uncertain functions and can be evaluated at arbitrary query points given some finite set of sample observation points. Although Gaussian process (GP) are frequently used, in this work we use Student-t processes (TP) instead, following the recommendation of Shah, et al. [13].

A Student-t process is a stochastic process such that every finite collection of samples from the process has a multivariate Student-t joint distribution. The mean function  $\Phi(\mathbf{q})$ , the kernel function  $k(\mathbf{q})$  and the parameter  $v$  are the main characteristics of TP. It can be written as

$$y(\mathbf{q}) \sim \mathcal{TP}(v, \Phi(\mathbf{q}), k(\mathbf{q}, \mathbf{q}')). \quad (21)$$

The real-valued parameter  $v > 2$  controls how “heavy-tailed” the process is. The heavier the tail (i.e. the smaller the  $v$ ), the more likely it is that the TP will produce a value that is far from the mean value. The TP tends to a GP as  $v \rightarrow \infty$ . The TP is attractive because it provides some extra benefits over GP, without incurring more computational cost. For example, the predictive covariance for the TP explicitly depends on observed  $y$  data values; this is a useful property which the Gaussian process lacks. Furthermore, distributions over the cost function  $y$  may in general be heavy-tailed, so it is better to use TP to “safely” model their behaviors [14]. Similarly, every finite collection of TP samples  $\mathbf{q}_{1:n} = (\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n)$  has a multivariate Student-t distribution, which can be written as

$$y(\mathbf{q}_{1:n}) = \frac{\Gamma(\frac{v+n}{2})}{\Gamma(v/2)((v-2)\pi)^{n/2}|K|^{1/2}} \\ \times (1 + \frac{1}{v-2}(\mathbf{q}_{1:n} - \Phi(\mathbf{q}_{1:n}))^T K^{-1}(\mathbf{q}_{1:n} - \Phi(\mathbf{q}_{1:n})))^{-\frac{v+n}{2}} \quad (22)$$

where  $\Gamma(n)$  is the gamma function for  $n \in \mathbb{R}$ , and  $v > 2$ .  $K$  is the covariance matrix consisting of kernel function  $k$  evaluations,

$$K = \begin{bmatrix} k(\mathbf{q}_1, \mathbf{q}_1) & k(\mathbf{q}_1, \mathbf{q}_2) & \cdots & k(\mathbf{q}_1, \mathbf{q}_n) \\ k(\mathbf{q}_2, \mathbf{q}_1) & k(\mathbf{q}_2, \mathbf{q}_2) & \cdots & k(\mathbf{q}_2, \mathbf{q}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{q}_n, \mathbf{q}_1) & k(\mathbf{q}_n, \mathbf{q}_2) & \cdots & k(\mathbf{q}_n, \mathbf{q}_n) \end{bmatrix}. \quad (23)$$

Equation (22) is written as the following for simplicity,

$$y(\mathbf{q}_{1:n}) \sim MVT_n(v, \Phi(\mathbf{q}_{1:n}), K). \quad (24)$$

For Bayesian optimization, newly sampled  $\mathbf{q}$  and  $y$  values are added to the vector  $\mathbf{q}_{1:n}$  and  $y(\mathbf{q}_{1:n})$  to construct a surrogate

model of the underlying objective function according to Eq. (24). In most implementations of Bayesian optimization, as the new sample values are added, the hyperparameters for the kernel function  $k$  are also re-estimated from the available data and updated accordingly. The updated surrogate model is then used to compute the acquisition function, which is used to select the next sample  $\mathbf{q}$  for evaluation of  $y$ .

2) *Acquisition Function*: The expected improvement function is one of many well-known acquisition functions; other possible and popular choices for the acquisition function include the *Lower Confidence Bound (LCB)*. We use expected improvement function in our implementations and we’ll discuss how this acquisition function is generated next. Suppose that  $n$  points  $\mathbf{q}_{1:n}$  and  $y(\mathbf{q}_{1:n})$  have been sampled and have been incorporated into the surrogate model. The current minimizer is  $\mathbf{q}_{1:n}^* = \min_{m \leq n} y(\mathbf{q}_m)$ . This will be one of these points, since observations of  $y$  are not available for other points in  $\mathcal{Q}$ . The algorithm needs to choose the next point  $\mathbf{q}_{n+1}$  to be sampled. We seek a new location which will yield a new, lower, minimum. In other words,  $y(\mathbf{q}_{n+1}) < y(\mathbf{q}_{1:n}^*)$ . One way to evaluate the new sample point is to evaluate its *improvement* with respect to  $\mathbf{q}_{1:n}^*$

$$g(\mathbf{q}_{n+1}, \mathbf{q}_{1:n}^*) = \max(0, y_n(\mathbf{q}_{1:n}^*) - y(\mathbf{q}_{n+1})). \quad (25)$$

This only assigns a non-zero value if  $\mathbf{q}_{n+1} < \mathbf{q}_{1:n}^*$ . Therefore, the idea is to chose  $\mathbf{q}_{n+1}$  which maximizes the improvement. Since we only have access to the surrogate function, the improvement is stochastic. Therefore, we choose the next sample point based on the *expected improvement*

$$E_n[g(\mathbf{q}_{n+1}, \mathbf{q}_{1:n}^*) \mid \mathbf{q}_{1:n}, \mathbf{y}(\mathbf{q}_{1:n})] \quad (26)$$

$E_n[\cdot]$  is the expectation based on current posterior distribution, given by the current *MVT* surrogate model. We need maximize Eq. 26 to find the next point to sample.

$$\mathbf{q}_{n+1} = \operatorname{argmax}_{\mathbf{q}_{n+1}} E_n[g(\mathbf{q}_{n+1}, \mathbf{q}_{1:n}^*) \mid \mathbf{q}_{1:n}, \mathbf{y}(\mathbf{q}_{1:n})] \quad (27)$$

The closed form solution of the EI acquisition function using the TP surrogate is [14], [15],

$$EI_n(\mathbf{q}) = (y_n(\mathbf{q}_{1:n}^*) - u)\Psi(z) + \frac{v}{v-1}(1 + \frac{z^2}{v})\sigma\psi(z) \quad (28)$$

where  $u$  and  $\sigma$  are the mean and variance of the conditional Student’s-t distribution of  $\mathbf{q}_{n+1}$ , which is presented below in (31).  $\Psi(\cdot)$  and  $\psi(\cdot)$  are the CDF and PDF of the standard Student-t distribution  $MVT_1(v, 0, 1)$ . The conditional *MVT* distribution is similar to the conditional multivariate Gaussian distribution: if we have  $\mathbf{q}_{1:n+1}$  and  $y(\mathbf{q}_{1:n+1})$  described by a multivariate *MVT* pdf, then

$$\begin{bmatrix} y(\mathbf{q}_{1:n}) \\ y(\mathbf{q}_{n+1}) \end{bmatrix} \sim MVT_{n+1}(v+1, \begin{bmatrix} \Phi(\mathbf{q}_{1:n}) \\ \Phi(\mathbf{q}_{n+1}) \end{bmatrix}, \begin{bmatrix} K(\mathbf{q}_{1:n}, \mathbf{q}_{1:n}) & K(\mathbf{q}_{1:n}, \mathbf{q}_{n+1}) \\ K(\mathbf{q}_{n+1}, \mathbf{q}_{1:n}) & K(\mathbf{q}_{n+1}, \mathbf{q}_{n+1}) \end{bmatrix}) \quad (29)$$

where  $K(\mathbf{q}_{1:n}, \mathbf{q}_{1:n})$  is the same as eq. (23),  $K(\mathbf{q}_{1:n}, \mathbf{q}_{n+1}) = [k(\mathbf{q}_1, \mathbf{q}_{n+1}), \dots, k(\mathbf{q}_n, \mathbf{q}_{n+1})]^T$ , and  $K(\mathbf{q}_{n+1}, \mathbf{q}_{n+1}) = k(\mathbf{q}_{n+1}, \mathbf{q}_{n+1})$ . (29) can be written more simply as

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \sim MVT_{n+1}(v+1, \begin{bmatrix} \Phi_1 \\ \Phi_2 \end{bmatrix}, \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix}) \quad (30)$$

The conditional Student-t distribution of  $y(\mathbf{q}_{n+1})$  is then given by [16]

$$\begin{aligned} y(\mathbf{q}_{n+1}|\mathbf{q}_{1:n}, y(\mathbf{q}_{1:n})) &\sim MVT_1(v+n, u, \sigma) \\ u &= \Phi_2 + K_{21}K_{11}^{-1}(y_1 - \Phi_1) \\ \sigma &= \frac{v+d}{v+n}K_{22} - K_{21}K_{11}^{-1}K_{12} \\ d &= (y_1 - \Phi_1)^T K_{11}^{-1}(y_1 - \Phi_1) \end{aligned} \quad (31)$$

The prior mean function for the surrogate model in Bayesian optimization can be set as a constant without changing the final result [17], so let  $\Phi_1 = \mathbf{0}, \Phi_2 = 0$ . Substituting Eq. (31) into Eq. (28), the only unknown variable will be  $\mathbf{q}_{n+1}$ . To find the  $\mathbf{q}_{n+1}$  that maximizes Eq. (28) for the largest improvement, another inner optimization problem must be solved within Bayesian optimization. Luckily, Eq. (28) is known, there are several ways to maximize the EI function, such as *DIRECT* [18], which is a derivative free and deterministic nonlinear global optimization algorithm that is widely used for Bayesian optimization via nonparametric surrogate model regression. Once the next point to sample is selected by the inner loop optimization, the Bayesian optimization loop can continue until the termination criterion (maximum iteration or minimum observation change between two iterations) is met.

The resulting algorithm for Bayesian optimization is referred to here as TPBO (Student-t processes Bayesian optimization). EI is used in this work because it has been shown to yield better or equal performance to other acquisition functions for a wide variety of applications [19], [20], [21].

### B. Stochastic Costs for Consistency-based Filter Auto-tuning

Now consider how  $y(\mathbf{q})$  can be defined via NEES and NIS consistency test statistics for Kalman filter tuning. As such, let  $\mathcal{Q}$  be some space of configurable Kalman filter parameters (e.g. the set of all parameters defining some positive definite symmetric process noise covariance  $\mathbf{Q}_k$ ) and let  $\mathbf{q} \in \mathcal{Q}$  be a design point.

Consider first the case of tuning based on assessment of NEES statistics obtained via Monte Carlo ground truth simulation models. If  $N$  Monte Carlo simulations are performed for  $T$  time steps at any given design point  $\mathbf{q}$ , starting from the initial conditions  $\hat{\mathbf{x}}_{0|0}$  and  $\mathbf{P}_{0|0}$ , then the average NEES statistic  $\bar{\epsilon}_{\mathbf{x},k}$  can be computed using Eq. (19) for each time  $k = 1, \dots, T$ . To summarize how “well-behaved”  $\bar{\epsilon}_{\mathbf{x},k}$  is across all time steps, we can leverage the fact that the expected value of  $\bar{\epsilon}_{\mathbf{x},k}$  for a consistent Kalman filter should be  $n_{\mathbf{x}}$ . Therefore, we use the following cost function to evaluate how much  $\bar{\epsilon}_{\mathbf{x},k}$  deviates from this ideal expected value

$$y(\mathbf{q}) = J_{NEES}(\mathbf{q}) = \left| \log \left( \frac{\sum_{k=1}^T \bar{\epsilon}_{\mathbf{x},k}/T}{n_{\mathbf{x}}} \right) \right| \quad (32)$$

The reason why we use the log of the cost is that the NEES value itself is bounded from below (by 0) but is not bounded above. Taking the log ensures that the cost will space from negative to positive infinity.

By a similar reasoning,

$$y(\mathbf{q}) = J_{NIS}(\mathbf{q}) = \left| \log \left( \frac{\sum_{k=1}^T \bar{\epsilon}_{\mathbf{z},k}/T}{n_{\mathbf{z}}} \right) \right| \quad (33)$$

where  $\bar{\epsilon}_{\mathbf{z},k}$  is the NIS outcome which could either be obtained from a ground truth simulation, or from a set of real data logs. One can also use negative log likelihood cost function, which will yield similar optimization result after our test.

Algorithm 1 summarizes the TPBO procedure for Kalman filter tuning. An attractive feature of TPBO is that it naturally provides uncertainty quantification on the shape of the objective function at both sampled and unsampled locations. This allows TPBO to cope with multiple local minima in the parameter space  $\mathcal{Q}$ .

---

#### Algorithm 1 TPBO for Kalman filter tuning

---

- 1: Initialize TP with seed data  $\{\mathbf{q}_s, y_s\}_{s=1}^{N_{seed}}$  and hyper-parameters  $\Theta$
  - 2: **while** termination criteria not met **do**
  - 3:      $\mathbf{q}_j = \operatorname{argmax}_{\mathcal{Q}} a(\mathbf{q})$
  - 4:     Evaluate  $y(\mathbf{q}_j)$ , e.g. using  $J_{NEES}(\mathbf{q})$  or  $J_{NIS}(\mathbf{q})$ .
  - 5:     Add  $y(\mathbf{q}_j)$  to  $\mathbf{f}(\mathcal{Q})$ ,  $\mathbf{q}_j$  to  $\mathcal{Q}$ , and update  $\Theta$
  - 6: **end while**
  - 7: **return**  $\mathbf{q}^* = \operatorname{argmin}_{\mathbf{q}_j \in \mathcal{Q}} \mathbf{f}(\mathbf{q}_j)$
- 

Based on samples over the system input and corresponding outputs from the objective function, Bayesian optimization fits a surrogate model of the “true” objective function. The optimization method then repeats this process to find a minimum of the current surrogate model, update this surrogate model, and find the minimum of the new surrogate model until pre-set termination conditions are met. The key steps for the EKF-Bayesian optimization tuning procedure are shown in Figure 1.

After the iteration starts, we can see in the flow chart that we maximize the acquisition function. The new point that can maximize the acquisition function will be chosen as the next set of process noise to run the EKF system. After running the EKF system with the new set of process noise we can get a new cost, which will be used to update the surrogate model.

## V. APPLICATION TO EXTENDED KALMAN FILTERS

We evaluated our Bayesian optimization auto-tuning algorithm on a nonlinear state estimation application that uses the Extended Kalman filter (EKF). This application is a closed-loop control system for the aero-robotic Mars Science Lab (MSL) Skycrane landing system. In this case, TPBO is used to automatically tune the assumed process noise covariance matrix, which is one of the main “tuning knobs” for the EKF.

The “Skycrane maneuver” was used as a deployment method for the Mars Science Laboratory (MSL) Curiosity rover upon its arrival and descent near the surface of Mars, as an alternative to the air bag method used in previous missions. Thrusters were used to stabilize the MSL Descent Stage System (a robotic aircraft) to zero horizontal velocity and to slowly guide the system to 20m above the ground

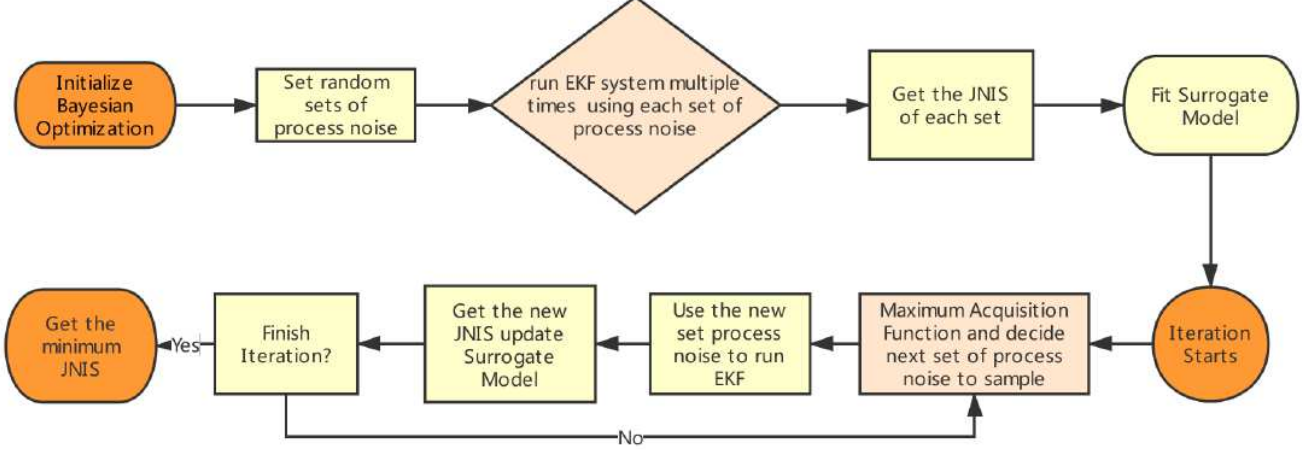


Fig. 1: System Flowchart: Bayesian optimization will fit a surrogate model based on some sampled input parameters and associated costs according to the “black box” objective function, and then starts iterating until a minimum cost is found.

to deploy the rover. A simplified model of this latter stages longitudinal dynamics will be used to simulate vehicle state estimation just prior to the rover deployment phase. Figure 2 depicts a simplified 2D longitudinal dynamics model of the MSL Descent Stage aircraft. More detailed descriptions of the MSL platform are given in [22], [23].

#### A. System description

The system is modeled as a rectangular box with two thrusters, one each on the bottom corners of the aircraft mounted at angle  $\beta$  to the vehicle  $z$  axis. The simplified vehicle states consist of the inertial translation  $\xi$  (m), altitude above surface  $z$  (m), pitch angle  $\theta$  (radian), and rates  $\dot{\xi}$  (m/s),  $\dot{z}$  (m/s), and  $\dot{\theta}$  (rad/s). The control inputs are defined in terms of the thrusts  $T_i$  (Newtons) produced by the  $i^{th}$  thruster. The state and control input are therefore

$$\begin{aligned} \mathbf{x}(t) &= (\xi, \dot{\xi}, z, \dot{z}, \theta, \dot{\theta})^T \\ \mathbf{u}(t) &= (T_1, T_2)^T. \end{aligned} \quad (34)$$

The equations of motion are derived here by considering only gravity, thrust, and drag forces (the vehicle is assumed not to generate significant lift in this phase). Drag will be modeled as  $F_{drag} = 1/2 C_D \rho A v^2$ , where  $C_D$  is the drag coefficient,  $\rho$  is the atmosphere density,  $v$  is the magnitude of the velocity, and  $A$  is the approximate cross-sectional area of the vehicle in the direction of motion. Let  $m_b$  be the mass of the Skycrane aircraft and payload,  $m_f$  be the mass of the fuel,  $\omega_b$  and  $h_b$  the width and height dimensions of the Skycrane body as shown in Fig. 2,  $\omega_f$  and  $h_f$  the dimensions of the propellant housing, and  $h_{cm}$  and  $\omega_{cm}$  the dimensions for the vehicle center of mass. The motion equation are written as

$$\begin{aligned} \ddot{\xi} &= \frac{T_1(\sin(\theta + \beta)) + T_2(\sin(\theta - \beta)) - F_{D,\xi}}{m_b + m_f} + \tilde{\omega}_1 \\ \ddot{z} &= \frac{T_1(\cos(\theta + \beta)) + T_2(\cos(\theta - \beta)) - F_{D,z}}{m_b + m_f} - g + \tilde{\omega}_2 \\ \ddot{\theta} &= \frac{1}{I_\eta} ((T_1 - T_2)(\cos\beta \frac{\omega_b}{2} - \sin\beta h_{cm})) + \tilde{\omega}_3 \\ \frac{1}{I_\eta} &= \frac{1}{12} (m_b(\omega_b^2 + h_b^2) + m_f(\omega_f^2 + h_f^2)) \\ F_{D,\xi} &= \frac{1}{2} C_D \rho (A_s \cos(\theta - \alpha) + A_b \sin(\theta - \alpha)) \dot{\xi} \sqrt{\dot{\xi}^2 + \dot{z}^2} \\ F_{D,z} &= \frac{1}{2} C_D \rho (A_s \cos(\theta - \alpha) + A_b \sin(\theta - \alpha)) \dot{z} \sqrt{\dot{\xi}^2 + \dot{z}^2} \\ \alpha &= \tan^{-1} \frac{\dot{z}}{\dot{\xi}} \end{aligned} \quad (35)$$

To simplify the model further, changes in  $m_f$  will be ignored. Values for these constants are provided in the appendix.

Sensors for state estimation consist of a simplified ideal single-axis IMU, i.e. an accelerometer and rate gyro pair which provide noisy measurements of inertial  $\xi$  accelerations and pitch rotations about the inertial  $\zeta$  axis.

The sensor data also include on-board barometer readings to gauge altitude. Image-based tracking measurements from an overhead passing satellite are also converted into noisy  $\xi$  platform position reports. The measurement vector can be written as

$$\mathbf{y} = \begin{bmatrix} \xi \\ z \\ \dot{\theta} \\ \dot{\xi} \end{bmatrix} + \tilde{\mathbf{v}}(t) \quad (36)$$

where  $\tilde{\mathbf{v}}(t) \in \mathbb{R}^4$  is the sensor error vector. The process

disturbance and measurement noise vectors are

$$\tilde{\mathbf{w}}(t) = (\tilde{\omega}_1, \tilde{\omega}_2, \tilde{\omega}_3)^T, \quad (37)$$

$$\tilde{\mathbf{v}}(t) = (\tilde{v}_1, \tilde{v}_2, \tilde{v}_3, \tilde{v}_4)^T, \quad (38)$$

all of which are modeled as additive white Gaussian noise. To obtain the appropriate matrices for the EKF, the discrete time state transition matrix is approximated by taking the Jacobian of the Euler-intergrated continuous time motion model  $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$  (with sample period  $\delta t = 0.1s$ ). The Jacobian of measurement model is obtained from Eq. (36). One important thing we need to notice is that our system now is in continuous time, to implement it we need convert it into discrete time, which needs some extra work. The details for obtaining the corresponding Jacobian matrices and discretization are provided in the Appendix. Note that the vehicle must maintain a desired nominal trim state of  $\mathbf{x}_{ref} = (0, 0, 20, 0, 0, 0)^T$  (steady hover 20 m above the surface). Linearization about the trim state reveals that the continuous time perturbation dynamics are unstable but controllable and observable. Hence, to maintain the platform at the desired state using estimated full-state state feedback, a Linear Quadratic Regulator (LQR) controller is also used to define the control inputs  $\mathbf{u}(t)$  at each discrete time step according to the control law,

$$\mathbf{u}_k = \mathbf{u}_{nom} - \mathbf{K}_{lin}(\mathbf{x}_k - \mathbf{x}_{ref}) \quad (39)$$

where the  $\mathbf{K}_{lin}$  is a pre-calculated LQR gain, which can be obtained offline using the separation principle assuming ideal full-state feedback for the linearized dynamics about trim (values given in Appendix). The same closed-loop control law is used throughout the Bayesian optimization auto-tuning procedure and the thrust values are made available to the EKF. The nominal thrusts  $\mathbf{u}_{nom}$  correspond to when the aircraft stabilizes to the desired state without process noise, and is given by

$$T_{1,nom} = T_{2,nom} = 0.5g \frac{m_b + m_f}{\cos \beta}. \quad (40)$$

An example of running the EKF for the Skycrane system can be seen in Figure 3, which shows the EKF's estimated state values over time with the help of LQR controller. Each element's variance of the process noise is  $(0.01, 0.01, 0.001)$  for  $\tilde{\mathbf{w}}(t)$  and variance of measurement noise is  $(1.0, 0.5, 0.025, 0.0225)$  for  $\tilde{\mathbf{v}}(t)$ . They all have zero mean.

### B. Discrete EKF From Continuous Time Model

The EKF prediction stage's formula from the continuous time will be different from the general discrete time EKF. The prediction stage is

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k), \quad (41)$$

$$\mathbf{P}_{k|k-1} = \tilde{\mathbf{F}}_k \mathbf{P}_{k-1|k-1} \tilde{\mathbf{F}}_k^T + \mathbf{\Omega}_k \mathbf{Q}_k \mathbf{\Omega}_k^T, \quad (42)$$

We cannot directly obtain  $f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k)$  using the continuous time formula. There are some other ways. The first method is that we can use the first order linearized form of  $f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k)$  to estimate it, which means

$$f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \approx \hat{\mathbf{x}}_{k-1|k-1} + \delta t f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \quad (43)$$

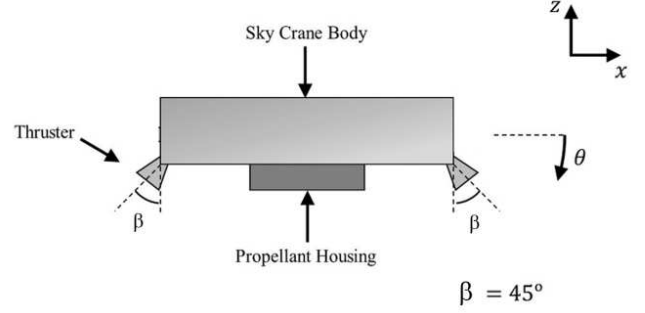


Fig. 2: The Skycrane aircraft has two thrusters angled at 45 degrees, which nominally try to keep the platform 20 meters above the ground with zero translational and rotational motion.

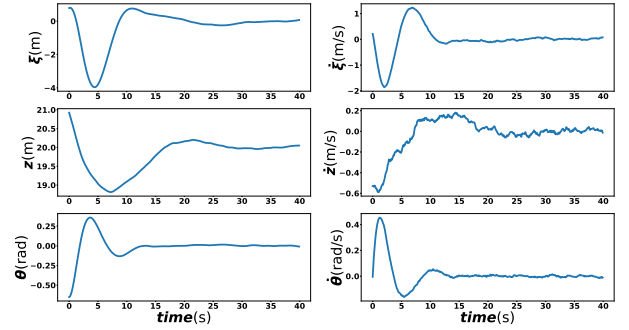


Fig. 3: Sample Skycrane simulation showing the LQR controller maintains the desired reference state using EKF-estimated full-state feedback.

The second method is that we can numerically solve the ordinary differential equations (ODE)  $f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k)$ , which can yield a more precise result. In our implementation we use the ODE integration library [24] to estimate the Eq. (41). In equation (42)  $\tilde{\mathbf{F}}_k \approx \mathbf{I} + \delta t \mathbf{F}_k$ ,  $\mathbf{F}_k$  can be computed from Eq. (13).  $\mathbf{\Omega}_k \approx \delta t \mathbf{\Gamma}_k$ .  $\mathbf{\Gamma}_k$  is a mapping matrix from the 3 dimension noise to the 6 dimension noise, which can be seen in the Appendix. The process noise covariance is

$$\mathbf{Q}_k = \begin{bmatrix} \mathbf{Q}_{\xi} & 0 & 0 \\ 0 & \mathbf{Q}_{\dot{\xi}} & 0 \\ 0 & 0 & \mathbf{Q}_{\dot{\theta}} \end{bmatrix} \quad (44)$$

The update stage will remain the same as Eq. 9 to 12. The measurement noise and its covariance are still fixed and written in the Appendix.

### C. Optimization results

As a first simple experiment, TPBO is used to perform a 1D parameter search for  $\mathbf{Q}_k$  defined as a constrained diagonal matrix, where the process noise covariances are such that  $\mathbf{Q}_{\xi} = \mathbf{Q}_{\dot{\xi}} = 10 * \mathbf{Q}_{\dot{\theta}}$ .

For the 1D parameter optimization, TPBO was applied over the range  $\mathbf{Q}_{\xi} \in [1 \times 10^{-2}, 1]$ , using 10 initial surrogate model



seed samples, 50 total iterations and 200 Monte Carlo run. The surrogate model and samples points for different sample iterations are shown in Figure 5. Table I also shows the numerical values for the final best minimizer found.

For the 2D parameter optimization, the parameter  $\mathbf{Q}_{\ddot{z}} = 0.1$  is held fixed, while  $\mathbf{Q}_{\ddot{\xi}}$  and  $\mathbf{Q}_{\ddot{\theta}}$  are optimized. The lower bound and upper bound are set as  $[1 \times 10^{-2}, 1 \times 10^{-3}]$  to  $[1, 1]$  respectively. We have 20 initial samples, 80 iterations and 200 Monte Carlo run. Again, the mean value of the surrogate model and the sample points at different iterations the TPBO found are shown in Figure 6. From 1D optimization result we can clearly see the result converge to points around 0.1 with high confidence and from the 2D optimization result we can see the result converge to points around  $[0.1, 0.01]$ . However, from the 2D result Table I we can see after 80 iterations, the cost does not change significantly as the  $\mathbf{Q}_{\ddot{\xi}}$  change when the  $\mathbf{Q}_{\ddot{\theta}}$  is around 0.1. This phenomenon may lead to a non-optimal result from TPBO. In Bayesian optimization, this happens when certain dimensions do not have a great impact on the cost, which encourages the addition of weights on other dimensions.

For the 3D optimization result, the boundaries for  $\mathbf{Q}_{\ddot{\xi}}$ ,  $\mathbf{Q}_{\ddot{z}}$ ,  $\mathbf{Q}_{\ddot{\theta}}$  are  $[1 \times 10^{-2}, 1 \times 10^{-2}, 1 \times 10^{-3}]$  to  $[1, 1, 1]$  respectively. We have 30 initial samples, 100 iterations and 200 Monte Carlo run for each sample. The value of  $\mathbf{Q}_{\ddot{z}}$  is far away from the optimal and it suffers from the same reason as the 2D optimization, which yield a relative larger error when we check the RMSE of  $z$  in Figure 7.

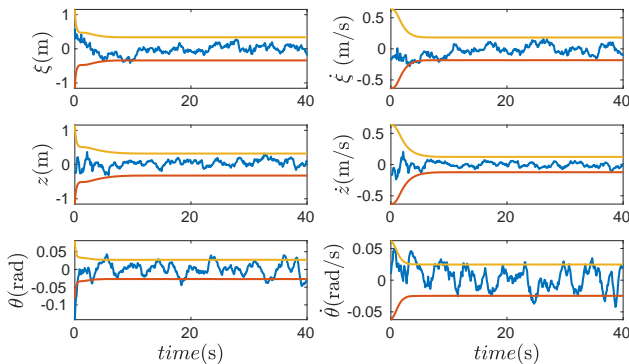


Fig. 4: An example time series error between Skycrane estimates using the BO-tuned optimal parameter values and ground truth states (orange lines:  $2\sigma$  bounds; blue line: error). Intuitively, the error should lie between the  $2\sigma$  bounds about 95% of the time if the filter is consistent.

To validate the results of TPBO, the RMS error of three states  $\xi, z, \theta$  between the EKF’s state estimation and the groundtruth (state from the simulator) are evaluated. We apply the 1D, 2D, 3D optimized parameters with a random set of process noise for 50 simulation runs of the Skycrane EKF and then obtain results in Figure 7. These results show that the 2D optimization results yield the best state estimates, since it has the minimum median error and smallest lower and upper error bounds. The 1D optimization result is similar to 2D optimization result. It is worth noting that all the errors are

Type \ Result	Cost	Optimal
1D opt	0.0206	(0.098, 0.098, 0.0098)
2D opt	0.0251	(0.0446, 0.1, 0.0119)
3D opt	0.0193	(0.0349838, 0.999984, 0.0152815)

TABLE I: Optimal means the optimal process noise for the EKF covariance. They stands for  $\mathbf{Q}_{\ddot{\xi}}$ ,  $\mathbf{Q}_{\ddot{z}}$ ,  $\mathbf{Q}_{\ddot{\theta}}$  respectively.

in fact small; for example, even though the RMSE of  $\theta$  is visually larger than the others, its median value is  $3 \times 10^{-3}$  rads. This is because measurement noise is set to fit the model precisely, so that its behavior will be robust most of the time. Under this condition, TPBO’s optimization ability is assessed over a relatively small range of NIS cost values. Figure 4 shows a typical trace for the state estimation error using the 3D optimized parameter estimates, indicating that consistent estimates are in fact obtained.

## VI. CONCLUSION

As a black box optimization method, TPBO simplifies what we need to know about a system in order to get the minimum cost. We used an example, Skycrane State estimation to show that this algorithm can be applied to complex nonlinear systems. This novel approach can also help the practitioners get the optimal process noise covariance much faster than tuning the EKF manually. In this paper we have shown results using an NIS-based cost function only. Although a NEES-based cost function can also work just as well, as shown in ref. [5], this requires the availability of ground truth state information, so NIS-based cost functions may often be more practical and are applicable with real sensor data. In this paper, we also have focused only on optimizing filter process noise parameters. However, the same auto-tuning process can be applied if the measurement noise also needs to be adjusted for a particular application [5]. In fact, if we don’t have confidence in either process noise or measurement noise covariances, it is possible to use TPBO to optimize these parameters simultaneously. The flexibility of the TPBO allows us to do more.

In the future, as this algorithm is robust to use, we aim to apply it to more realistic hardware-based system tuning problems. Another interesting direction for future research involves optimization of higher dimensional parameter spaces, where some dimensions may potentially have little/no noticeable effect on the NIS tuning cost. Possible strategies for handling this might include using TPBO to optimize those parameters which have a significant impact on the tuning cost, leaving the remaining parameters to be hand-tuned. Since the TPBO algorithm is able to support arbitrary “black box” cost function evaluations, modifications or alternatives to the NIS cost function could also be explored. Finally, the flexibility of our TPBO approach means that it can be applied to other optimal estimator tuning problems. Most notably, for example, we have already applied this approach to VI SLAM (Visual Inertial Simultaneous Localization and Mapping) extrinsic

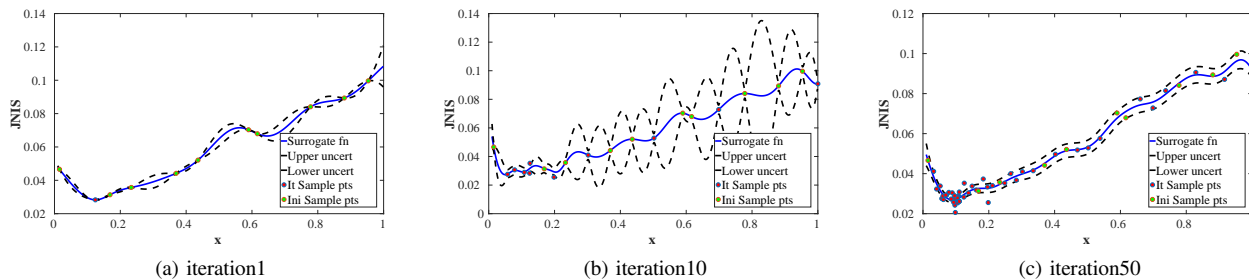


Fig. 5: 1D plots of surrogate functions and uncertainty bounds for the Skycrane problem, as well as sample points chosen by Bayesian optimization. The black dashed line represents the lower and upper uncertainty bounds for a 95% confidence interval. The green dots are the initial sample points. The red dots are the sample points after the iteration starts. The blue line is the mean of the surrogate model

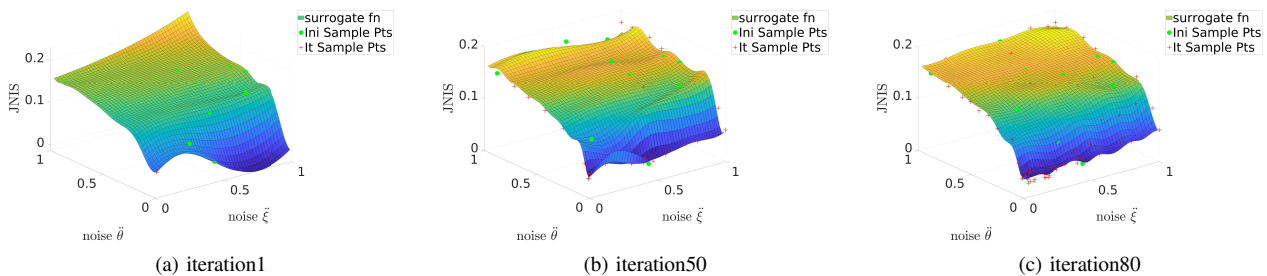


Fig. 6: 2D surface plots of surrogate functions for the Skycrane problem, as well as sample points. The green circles are the initial sample points and the red cross is the sample points after the iterations starts. Upper and lower bound surfaces are not plotted here or the surfaces may cover each other.

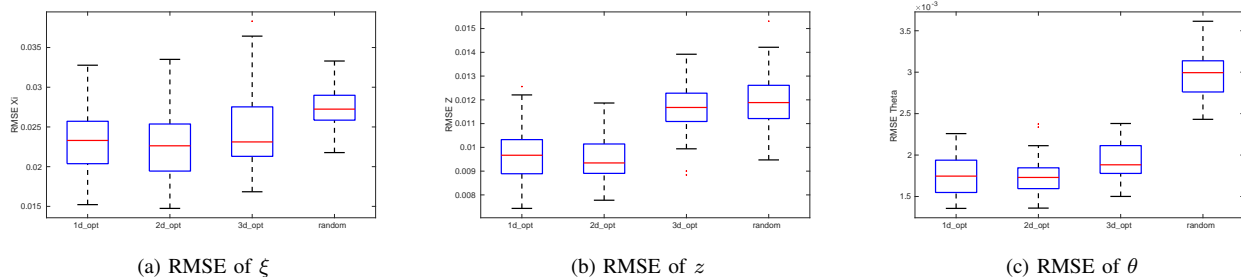


Fig. 7:  $Nd\_opt$  means we use the optimization result from the  $ND$  search and run Skycrane system. The left one shows the RMSE of translation and the right one shows the RMSE of rotation.

parameter calibration [25], where the “extrinsic parameter” means the relative pose between the camera and other sensors.

Substituting for Eq. (35) and taking derivatives, the Jacobian of the process model is

## APPENDIX A

### ADDITIONAL INFORMATION OF SKY-CRANE MODEL

#### A. Jacobian and Parameters

The process model for the Skycrane has the form  $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) + \tilde{\mathbf{w}}$ , where

$$\begin{bmatrix} \dot{\xi} \\ \dot{\zeta} \\ \dot{z} \\ \dot{\theta} \end{bmatrix} = f \left( \begin{bmatrix} \xi \\ \zeta \\ z \\ \theta \end{bmatrix}, \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} \right) + \begin{bmatrix} 0 \\ \tilde{\omega}_1 \\ 0 \\ \tilde{\omega}_2 \\ 0 \\ \tilde{\omega}_3 \\ 0 \end{bmatrix} \quad (45)$$

$$\mathbf{F}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{F}_{11} & 0 & \mathbf{F}_{13} & \mathbf{F}_{14} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \mathbf{F}_{31} & 0 & \mathbf{F}_{33} & \mathbf{F}_{34} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (46)$$

where

$$\begin{aligned}
\mathbf{F}_{11} &= nc((A_{sc} + A_{bs})(2\dot{\xi}^2 + \dot{z}^2)/V_t + \dot{\xi}\dot{z}(A_{bc} - A_{ss})/V_t) \\
\mathbf{F}_{13} &= nc((A_{sc} + A_{bs})(\dot{\xi} + \dot{z})/V_t + \dot{\xi}^2(A_{ss} - A_{bc})/V_t) \\
\mathbf{F}_{14} &= l(T_1 \cos(\beta + \theta) + T_2 \cos(\beta - \theta) + cd\dot{\xi}V_t(A_{ss} - A_{bc})) \\
\mathbf{F}_{31} &= nc((A_{sc} + A_{bs})(\dot{\xi} + \dot{z})/V_t) + \dot{z}^2(A_{bc} - A_{ss})/V_t \\
\mathbf{F}_{33} &= nc((A_{sc} + A_{bs})(2\dot{\xi}^2 + \dot{z}^2)/V_t + \dot{\xi}\dot{z}(A_{ss} - A_{bc})/V_t) \\
\mathbf{F}_{34} &= l(T_2 \sin(\beta - \theta) - T_1 \sin(\beta + \theta) + cd\dot{\xi}V_t(A_{ss} - A_{bc}))
\end{aligned} \tag{47}$$

Taking derivates of (36),

$$\mathbf{H}(t) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \mathbf{H}_{31} & 0 & \mathbf{H}_{33} & \mathbf{H}_{34} & 0 \end{bmatrix} \tag{48}$$

where

$$\begin{aligned}
\mathbf{H}_{31} &= nc((A_{sc} + A_{bs})(2\dot{\xi}^2 + \dot{z}^2)/V_t + \dot{\xi}\dot{z}(A_{bc} - A_{ss})/V_t) \\
\mathbf{H}_{33} &= nc((A_{sc} + A_{bs})(\dot{\xi} + \dot{z})/V_t + \dot{\xi}^2(A_{ss} - A_{bc})/V_t) \\
\mathbf{H}_{34} &= l(T_1 \cos(\beta + \theta) + T_2 \cos(\beta - \theta) + cd\dot{\xi}V_t(A_{ss} - A_{bc}))
\end{aligned} \tag{49}$$

The symbols in (47) and (49) are defined as follows

$$\begin{aligned}
cd &= 0.5\rho C_D \\
l &= \frac{1}{m_f + m_b} \\
nc &= -0.5\rho l C_D \\
\omega_{cm} &= \frac{\omega_b}{2} \\
\alpha &= \tan^{-1}(\dot{z}/\dot{\xi}) \\
V_t &= \sqrt{\dot{\xi}^2 + \dot{z}^2} \\
A_s &= (h_b d_b) + (h_f d_f) \\
A_b &= (\omega_b d_b) + (\omega_f d_f) \\
A_{sc} &= A_s \cos(\theta - \alpha) \\
A_{ss} &= A_s \sin(\theta - \alpha) \\
A_{bc} &= A_b \cos(\theta - \alpha) \\
A_{bs} &= A_b \sin(\theta - \alpha)
\end{aligned} \tag{50}$$

All the basic constants value are written here

$$\begin{aligned}
\rho &= 0.02 \text{ kg m}^{-3} \\
g &= 3.711 \text{ m s}^{-2} \\
\beta &= \frac{\pi}{4} \text{ rad} \\
C_D &= 0.2 \\
m_f &= 390 \text{ kg} \\
\omega_f &= 1 \text{ m} \\
h_f &= 0.5 \text{ m} \\
d_f &= 1 \text{ m} \\
m_b &= 1510 \text{ kg} \\
\omega_b &= 3.2 \text{ m} \\
h_b &= 2.5 \text{ m} \\
d_b &= 2.9 \text{ m} \\
h_{cm} &= 0.9421 \text{ m}
\end{aligned} \tag{51}$$

Mapping between 3 dimensional process noise and 6 dimensional measurement noise  $\mathbf{\Gamma}_k$  is

$$\mathbf{\Gamma}_k = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{52}$$

The fixed measurement noise is

$$\tilde{\mathbf{v}}(t) = (1.0, 0.5, 0.025, 0.0225)^T \tag{53}$$

The measurement noise covariance is set as

$$\mathbf{R}_k = \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.025 & 0 \\ 0 & 0 & 0 & 0.0025 \end{bmatrix} \tag{54}$$

### B. Feedback Law of LQR controller

We need linearize the motion model in order to use LQR controller, e.g. calculating the Jacobian of motion model. We have calculated the Jacobian of motion model with respect to state  $\mathbf{x}$ . We also need the Jacobian with respect to control  $\mathbf{u}$  and noise  $\mathbf{w}$  respectively. The Jacobian with respect to control input is

$$\mathbf{U}(t) = \begin{bmatrix} 0 & 0 \\ \sin(\theta + \beta)l & \sin(\theta - \beta)l \\ 0 & 0 \\ \cos(\theta + \beta)l & \cos(\theta - \beta)l \\ 0 & 0 \\ \frac{1}{l_\eta}(0.5 \cos(\beta)\omega_b - \sin(\beta)h_{cm}) & -\mathbf{U}_{50} \end{bmatrix} \tag{55}$$

The Jacobian with respect to the noise is

$$\mathbf{W}(t) = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{56}$$

Substitute  $\mathbf{x}_{ref}$ ,  $\mathbf{u}_{nom}$  into equation 56 and 55 we can get the linearized value of jacobian at the desired point  $\mathbf{F}_{\mathbf{x}_{ref}}$  and  $\mathbf{U}_{\mathbf{u}_{nom}}$ . Then the linearized state space model around the desired state can be written as

$$\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{F}_{\mathbf{x}_{ref}} \mathbf{x} + \mathbf{U}_{\mathbf{u}_{nom}} \mathbf{u} \\
\mathbf{y} &= \mathbf{x}
\end{aligned} \tag{57}$$

Then we can get the optimal gain matrix  $K_{lin}$  by

$$\mathbf{K}_{lin} = \mathbf{R}_{con}^{-1} \mathbf{U}_{\mathbf{u}_{nom}}^T \mathbf{S}_{con} \tag{58}$$

where  $\mathbf{S}_{con}$  is the solution of of the associated Riccati equation

$$\begin{aligned}
\mathbf{F}_{\mathbf{x}_{ref}}^T \mathbf{S}_{con} + \mathbf{S}_{con} \mathbf{F}_{\mathbf{x}_{ref}} - \mathbf{S}_{con} \mathbf{U}_{\mathbf{u}_{nom}} \mathbf{R}_{con}^{-1} \mathbf{U}_{\mathbf{u}_{nom}}^T \\
+ \mathbf{Q}_{con} = \mathbf{0}
\end{aligned} \tag{59}$$

the  $\mathbf{R}_{con}$  is a 2 by 2 diagonal matrix with its diagonal element (0.01, 0.01) and the  $\mathbf{Q}_{con}$  is a 6 by 6 diagonal matrix with its

diagonal elements (200 15 200 15 10000 15). Finally we get our  $\mathbf{K}_{lin}$

$$\mathbf{K}_{lin} = \begin{bmatrix} 100.0 & -100.0 \\ 406.575 & -406.575 \\ 100.0 & 100.0 \\ 519.086 & 519.086 \\ 3053.285 & -3053.285 \\ 3140.470 & -3140.470 \end{bmatrix}^T \quad (60)$$

#### REFERENCES

- [1] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*. Ieee, 2000, pp. 153–158.
- [2] X. R. Li, Z. Zhao, and V. P. Jilkov, "Practical measures and test for credibility of an estimator," in *Proc. Workshop on Estimation, Tracking, and Fusion A Tribute to Yaakov Bar-Shalom*, 2001, pp. 481–495.
- [3] H. Dette, A. Pepelyshev, and A. Zhigljavsky, "Optimal designs in regression with correlated errors," *Annals of statistics*, vol. 44, no. 1, p. 113, 2016.
- [4] L. Wanninger, "Real-time differential gps error modelling in regional reference station networks," in *Advances in Positioning and Reference Frames*. Springer, 1998, pp. 86–92.
- [5] Z. Chen, C. Heckman, S. Julier, and N. Ahmed, "Weak in the needs?: Auto-tuning kalman filters with bayesian optimization," in *2018 21st International Conference on Information Fusion (FUSION)*. IEEE, 2018, pp. 1072–1079.
- [6] J. K. Uhlmann, "Covariance consistency methods for fault-tolerant distributed data fusion," *Information Fusion*, vol. 4, no. 3, pp. 201–215, 2003.
- [7] Y. Bar-Shalom, X. Li, and T. Kirubarajan, *Estimation with Applications to Navigation and Tracking*. New York: Wiley, 2001.
- [8] R. F. Stengel, *Optimal control and estimation*. Courier Corporation, 1986.
- [9] Y. Oshman and I. Shaviv, "Optimal tuning of a kalman filter using genetic algorithms," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2000, p. 4558.
- [10] T. D. Powell, "Automated tuning of an extended kalman filter using the downhill simplex algorithm," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 5, pp. 901–908, 2002.
- [11] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "Boa: The bayesian optimization algorithm," in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*. Morgan Kaufmann Publishers Inc., 1999, pp. 525–532.
- [12] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 63–71.
- [13] A. Shah, A. Wilson, and Z. Ghahramani, "Student-t processes as alternatives to gaussian processes," in *Artificial Intelligence and Statistics*, 2014, pp. 877–885.
- [14] A. Shah, A. G. Wilson, and Z. Ghahramani, "Bayesian optimization using student-t processes," in *NIPS Workshop on Bayesian Optimisation*, 2013.
- [15] B. D. Tracey and D. Wolpert, "Upgrading from gaussian processes to studentst processes," in *2018 AIAA Non-Deterministic Approaches Conference*, 2018, p. 1659.
- [16] P. Ding, "On the conditional distribution of the multivariate t distribution," *The American Statistician*, vol. 70, no. 3, pp. 293–295, 2016.
- [17] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.
- [18] D. E. Finkel, "Direct optimization algorithm user guide," *Center for Research in Scientific Computation, North Carolina State University*, vol. 2, pp. 1–14, 2003.
- [19] J. R. Gardner, M. J. Kusner, Z. E. Xu, K. Q. Weinberger, and J. P. Cunningham, "Bayesian optimization with inequality constraints," in *ICML*, 2014, pp. 937–945.
- [20] M. A. Gelbart, J. Snoek, and R. P. Adams, "Bayesian optimization with unknown constraints," *arXiv preprint arXiv:1403.5607*, 2014.
- [21] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. De Freitas, "Bayesian optimization in high dimensions via random embeddings," in *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.
- [22] A. Steltzner, D. Kipp, A. Chen, D. Burkhart, C. Guernsey, G. Mendeck, R. Mitcheltree, R. Powell, T. Rivellini, M. San Martin *et al.*, "Mars science laboratory entry, descent, and landing system," in *2006 IEEE Aerospace Conference*. IEEE, 2006, pp. 15–pp.
- [23] R. Mitcheltree, A. Steltzner, A. Chen, M. SanMartin, and T. Rivellini, "Mars science laboratory entry, descent and landing system verification and validation program," in *2006 IEEE Aerospace Conference*. IEEE, 2006, pp. 6–pp.
- [24] K. Ahnert and M. Mulansky, "Odeint-solving ordinary differential equations in c++," in *AIP Conference Proceedings*, vol. 1389, no. 1. AIP, 2011, pp. 1586–1589.
- [25] Z. Chen, "Visual-inertial slam extrinsic parameter calibration based on bayesian optimization," 2018.