Robust optimal sensor placement for optimal estimation

A framework and application to adaptive optics

S.C.A. de Groot

Master of Science Thesis

Delft University of Technology

Delft Center for Systems and Control

Robust optimal sensor placement for optimal estimation

A framework and application to adaptive optics

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

S.C.A. de Groot

October 8, 2018

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Robust optimal sensor placement for optimal estimation

by

S.C.A. de Groot

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: October 8, 2018

Supervisor(s):

prof. dr. ir. M. Verhaegen

ir. R. Doelman

Reader(s):

dr. R. Ferrari

Abstract

The goal of this thesis is to present a framework which can be used to optimize the observer and controller performance by placement of sensors and actuators on a continuous domain. The framework provides choices of model structure, cost functions and optimization methods to perform the placement. Systems which would be suitable for the use of the framework are for example adaptive optics systems, active dampening of windmill blades or climate control systems in buildings.

To demonstrate the use of the framework, two applications are presented. Both applications are sensor placement for an adaptive optics system. The first implementation uses a Linear Time Invariant (LTI) state space model and uses a Vector Autoregressive (VAR) approach for identification. The second implementation uses a Linear Parameter Varying (LPV) state space model with VAR identification approach. The second implementation is meant to demonstrate how the placement can be made more robust to changing conditions.

Finally we show how parametrization of the sensor locations can be used to gain insight into what properties an optimal placement exhibits.

Table of Contents

Acknowledgements

This thesis was written as the final part of my Master of Science degree in Systems & Control at the DCSC department of the TU Delft. I would like to thank everyone who was involved in the research and writing of this thesis.

First off all, I would like to thank my supervisors prof. dr. ir. M. Verhaegen and ir. R. Doelman for their many useful insights and support during the writing of this thesis. Without their help during these last eight months, writing this thesis would not have been possible. However, any mistakes that remain are my own.

Furthermore I would like to thank my parents, who always supported me during my school and academic career, and always helped me to be the best version of myself.

Lastly I would like to thank my fellow students and friends Ruben van Beelen and Floris van Elteren for the many discussions, ideas, tips, tricks and cups of coffee.

Delft, University of Technology
October 8, 2018

S.C.A. de Groot

Chapter 1

Introduction

1-1 Motivation

In general, the focus in control engineering literature is placed on designing the best controller for a given system. It is assumed that the system to be controlled is a given, with fixed inputs and outputs. In practice however, placement and selection of sensors and actuators is an integral part of the design process. Choosing the best locations and combinations of sensors and actuators can be a very difficult task which greatly influences the controllability and observability of a system. To this end, it is useful to be able to quantify the usefulness of a sensor or actuator at a specific location, and use this information to find optimal locations to improve the performance of the system. Furthermore, placing sensors and actuators in a strategic way can reduce the amount required to achieve design goals, making placement a useful tool in reducing overall cost.

1-2 State of the art

When considering the state of the art of these techniques and their applicability in industry, we can note an interesting trend. When you wanted to own a car before the 1900's, you would contact a manufacturing company who would build you a rolling frame with an engine and steering wheel. You would then take this frame to coach builder, who would then fabricate custom bodywork and seating much like you would see on a horse drawn carriage. This was all very expensive and time consuming. At the start of the 20th century new manufacturing techniques led to a massive increase in mass production and in 1908 the first model T Ford rolled of the production line. Illustrated by Ford's famous quote: "you can order the model T in any color you like, as long as it's black." almost all production started to become mass production, ending up in the 21st century, where almost all goods are mass produced. In recent years new production techniques have become mainstream which can reverse this trend. For example 3d printing in the medical field, the automotive and aerospace industry. Or for example companies like the Delft start-up Mapper, which strives to make custom lithography for semi conductor manufacturing affordable. To this end we could also use the placement algorithms to provide custom solutions for specific applications. These custom solutions are interesting because they allow us to make our optimization goals much more specific and produce a product which performs as good as possible in it's intended application, instead of a product which performs well in many applications which it will never be used for.

Most publications regarding sensor and actuator optimization have been done on sensor and actuator selection. Many approaches make use of relaxations and LMI's to solve the selection problem[1, 2, 3, 4]. Comparing older papers with current ones, we see that improved computing power has changed the approach which is taken, from manually checking combinations[5, 6] to purely using computers. When it comes to placement, most publications note that the problem is not convex in general. Due to this authors choose optimization algorithms which make less use of the shape of the objective function like genetic algorithms[7, 8, 9]. Because of the importance of the underlying dynamics, systems which translate well into state space form are often used, like Ginzberg-Landau equations. In most cases these systems can be described using partial differential equations which are then transformed into state space form using spatial discretization, and/or some method of linearization[10]. The optimizations of sensor and actuator location are often based on the same cost functions used in optimal control. Others have based cost functions of observability and controllability Gramians.[11] which carry the caveat in MIMO applications that each individual sensor and actuator must render the system observable and controllable respectively, as the Gramians are undefined for unobservable and uncontrollable systems.

1-3 Contribution

The contribution that this thesis tries to make is a framework which can be used to place sensors and actuators for systems where these can be placed on a domain, e.g. along the length of a windturbine blade or airplane wing, in a climate controlled room, in the aperture of an adaptive optics system. Using the framework proposed in this thesis, it will be possible to place sensors and actuators using different cost functions and model forms, compare the performance of these models and cost functions and evaluate how robust the placements are compared to each other.

1-4 Thesis outline

This thesis is divided into two parts. The first part is the "Framework" for solving sensor and actuator placement problems. It is divided into three chapters. The first chapter contains information on model and observer structure for different types of systems, and cost functions relevant to each model. The second chapter contains optimization methods for solving the placement problem. The third chapter contains a method to compare the performance of different implementations. These elements are meant to be used in "mix and match" fashion to be best suited for the specific placement problem at hand. The second part contains two applications of the framework. Both applications describe solving the placement problem for an adaptive optics system using different model structures to demonstrate how the framework is meant to be used, and how the placement can be made robust to changing conditions.

Part I

Framework

Chapter 2

Model structures

2-1 LTI model

This section describes the first model which can be used. The Linear Time Invariant (LTI) model in state space form is one of the most used model structures and applicable for many systems. It can be used for many systems, linear systems, but non-linear systems linearized around an operating point as well. The model structure allows for relatively easy identification, modeling and calculation of stability, controllability, observability and calculation of (optimal) state estimators.

2-1-1 LTI Model structure

We consider a discrete time LTI model of order N in state-space form as in Equation (2-1) where $x \in \mathbb{R}^m$ is the state. $u \in \mathbb{R}^n$ is the input, $y \in \mathbb{R}^o$ is the output, $w \in \mathbb{R}^m$ is the disturbance, $v \in \mathbb{R}^o$ the measurement noise and q_a and q_s are the actuator and sensor positions respectively. It is assumed that the pair $(A, B_1(q_a))$ is stabilizable and the pair $(A, C(q_s))$ is detectable.

$$\begin{aligned} x[k+1] =& Ax[k] + B_1(q_a)u[k] + B_2w[k], \\ y[k] =& C(q_s)x[k] + v[k], \end{aligned} \tag{2-1}$$

If we now implement state feedback $u = -Fx$ the resulting system will become

$$\begin{aligned} x[k+1] =& (A - B_1(q_a)F)x[k] + B_2w[k], \\ y[k] =& C(q_s)x[k] + v[k], \end{aligned} \tag{2-2}$$

2-1-2 Observer

To be able to perform state feedback, an optimal observer or static Kalman filter is implemented. The observer is of the form in Equation (2-3), where $A_o = A$ in open loop and

$A_o = (A - B_1(q_a)F)$ in the closed loop case. Because our system is LTI, we can separate the controller and observer problem.

$$\begin{aligned} \hat{x}[k+1] =& (A_o - LC(q_s))\hat{x}[k] + B_1(q_a)u[k] + Ly[k], \\ \hat{y}[k] =& C(q_s)\hat{x}[k], \end{aligned} \tag{2-3}$$

where L is the observer gain. We will implement a static Kalman observer. The static Kalman observer gain for a discrete time LTI system is[12]

$$L = (A_o P C(q_s)^T)(C(q_s) P C(q_s)^T + R)^{-1}, \tag{2-4}$$

where P is the solution to the Discrete Algebraic Ricatti Equation (DARE)

$$P = A_o^T P A_o - (A_o^T P C(q_s)^T)(R + C(q_s) P C(q_s)^T)^{-1}(C(q_s) P A_o) + Q. \tag{2-5}$$

2-1-3 Performance

To be able to quantify the performance of the placement we need a cost function for the placement of the sensors and actuators. The observer performance can be based on the solution to the DARE. This solution, P, is the estimation error covariance. Hence, if we reduce the amplitude of the elements of P we will improve the estimation of the system state. This procedure can be mirrored when placing actuators, where reducing the amplitude of the DARE solution elements will reduce the relative influence of the disturbance on the system states. In general, the cost function of an optimization problem needs to be scalar valued. As the solution to DARE is a matrix, we cannot use it directly. There are several scalar performance metrics based on the P matrix, each with its own influence on the behaviour of the system, so called matrix norms. Two common and relatively easy to calculate norms are the trace function and the maximum eigenvalue. There exist several optimality types which can be used to categorize a cost function. These types originally relate to the Fisher information matrix, which elements quantify the amount of information obtained about a variable through a measurement[13], but can be applied to the error covariance in a similar way[14]. We will make use of two of these types. The first one, A-optimality, suppresses the average variance of the estimation error, and its corresponding cost function is defined as:

$$J_A = trace(P). \tag{2-6}$$

Optimization with the maximum eigenvalue cost function produces E-optimality which minimizes the largest axis in the uncertainty ellipsoid spanned by the eigenvectors of the error covariance matrix, which can be interpreted for example as a "worst case" error of a single state variable, making it useful if we want to optimize an upper bound on the estimation error.

$$J_E = \overline{\lambda}(P). \tag{2-7}$$

To achieve a better understanding of these two concepts, they are illustrated for a 2×2 covariance matrix for normally distributed random variables x and y in Figure 2-1. In this figure you can see 350 randomly generated points with a certain covariance. The blue ellipse is called the confidence ellipse, which shows the area in which a certain percentage of points will be located(in this case 95%). Using the A-optimality cost function reduces the surface area of the blue ellipse. The green and pink lines represent the eigenvectors of the covariance matrix, and optimizing using the E-optimality cost function will reduce the length of the longest eigenvector, changing the shape of the ellipse.

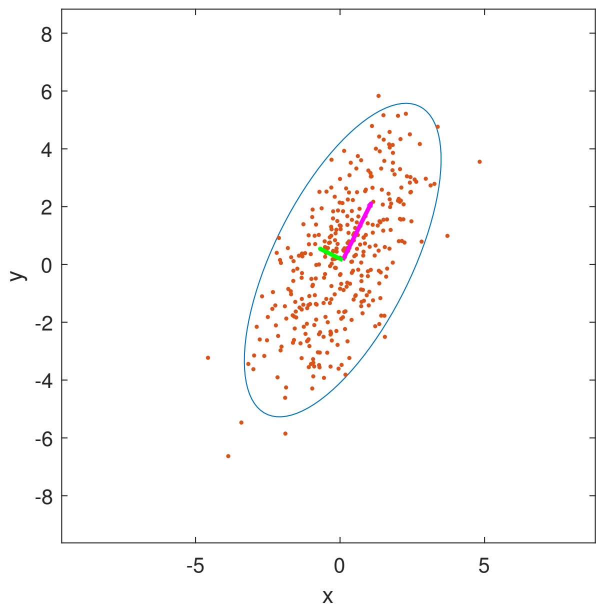


Figure 2-1: 350 normally distributed points, their 95% confidence ellipse and eigenvectors of the covariance matrix. Based on code from [15]

2-2 LPV model

As useful and widely applicable as LTI models are, there are many systems where their use is limited to a certain range of operation, or operating point. To broaden the usefulness of the framework, a model form which can improve on the applicability of the LTI model, whilst still maintaining some of its useful properties, and introduce robustness to the placement is presented here. The model form is called the LPV model, where the values of the system matrices can vary linearly with one or more parameters, for example an airplane model whose A matrix will change with altitude, or a wind turbine model which varies with wind speed.

2-2-1 LPV model structure

We consider an Linear Parameter Varying (LPV) model with parameter independent output of the following form[16]:

$$\begin{aligned} x[k+1] =& \sum_{i=1}^{n} \mu[k](A^{(i)}x[k] + B^{(i)}(q_a)u[k] + K^{(i)}e[k]), \\ y[k] =& C(q_s)x[k] + Du[k] \end{aligned} \tag{2-8}$$

Where $\mu[k] \in \mathbb{R}^p$ with its elements bounded by an operating domain D is a scheduling parameter independent of the other system variables.

2-2-2 Observer

Depending on the parameter set μ there can be multiple approaches to calculate the observer gain. Firstly, we can calculate the observer gain on-line for each step as if the system was LTI. Secondly, we can implement an observer with dynamic observer gain based on the transient DARE, which is guaranteed to be stable for all μ dynamics.[17, 12] In the case where the dynamics of μ are significantly slower than the system dynamics we can safely implement the static online calculations without the possibility of the observer becoming unstable.[18] The main advantage for our application is the fact that we can calculate all possible DARE solutions(and therefore all possible feedback gains) simply by knowing the range of the parameters, which would not be possible when using the dynamic gain.

2-2-3 Performance

For the LTI model we described the use of the estimation error covariance, or solution to the DARE to obtain a cost function for optimizing the sensor and actuator locations. In the LPV case, P becomes dependent on μ, and if a transient DARE based observer is implemented, on the behavior of μ and system state as well. Building on the operating point approach, we can remedy the problem by basing our cost function on the value of P in these operating points denoted by μ_i. Similar as for the LTI model, we provide an E and A optimality cost function. If we look at the influence of the placement on all variables, or A-optimality, the cost function will become:

$$J_A = \frac{1}{N}\sum_{i=1}^{N} trace(P(\mu_i)). \tag{2-9}$$

Where we take the average of the trace at each operating point to keep in line with the A-optimality. For the worst case variable, or E-optimality, the cost function will become:

$$J_E = \max \overline{\lambda}(P(\mu_i)). \tag{2-10}$$

Where we take the worst case of all operating points to keep in line with the E-optimality.

Chapter 3

Placement optimization methods

After choosing a model and cost function from the previous section, we can optimize with respect to the sensor positions. It is assumed that the number of sensors to be used is determined a priori, to avoid making the problem overly complex. In the literature, the terms selection and placement are often used interchangeably, whilst we consider these terms to fundamentally different processes in terms of sensor/actuator optimization. Sensor/actuator selection implies that we select sensors/actuators which would mean that we have predetermined set to choose from. This would mean that the problem is a combinatorial search problem. Placement is considered as placing sensors/actuators on a continuous domain. When the number of sensors/actuators is kept constant the optimization variables, which are the sensor and actuator coordinates, are all continuous and real.

3-1 Methods

The choice of optimization method heavily depends on the cost function. Due to the dependency of the discussed cost functions on the DARE, the cost function will never be convex. How the sensor and actuator locations map to the cost function is not constrained. Because of this we might have to deal with many local minima. To remedy this problem, we can choose an optimization method which can escape local minima. Avoiding problems with local minima can be avoided by employing a global optimization algorithm like Grid Search (GS), or by using algorithms which can "jump" out of them like the Data-based Online Nonlinear Extremumseeker (DONE).

3-1-1 Grid Search

The easiest way to account for many of the problems encountered in non convex optimization is to check all the possibilities. In an integer problem this can be done very straightforward. However, when we deal with continuous variables like in the placement problem, there are infinitely many possible solutions, taking an infinite amount of time to check all solutions. A way around this, is to discretize the number of solutions with the use of a grid. This method does not provide an optimal solution, but if we can assume that the solutions in the space between each point on the grid transition relatively smoothly from one solution to the next, we can assume that the solution is very close to optimal. If this is not the case, the spacing between points must become smaller. A large downside to this method is that it requires many calculations and does not take into account any previous information which makes it very inefficient. Take for example a problem with two variables with constraint $x^2 + y^2 \leq 1$ then the grid search would entail evaluating the points as seen in Figure 3-1. If the cost function takes a lot of processing to evaluate, and we would like to use such an algorithm to place many sensors, it is clear that this method is not very favorable.

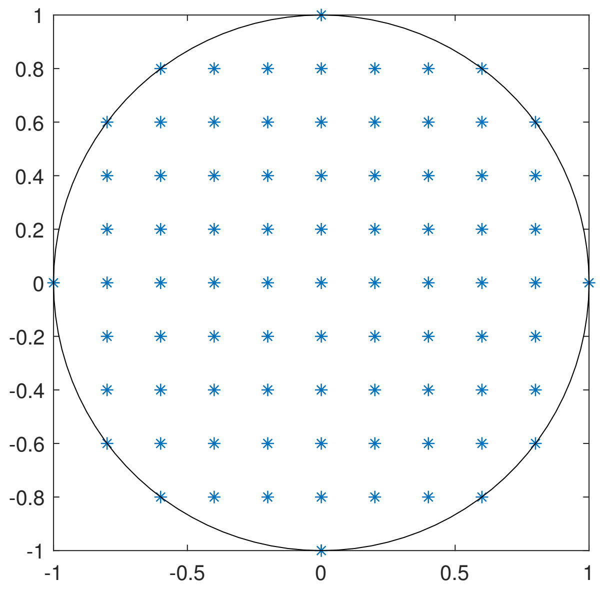


Figure 3-1: Grid with finite number of possible values on a circular domain bounded by $x^2+y^2 \leq 1$ and spacing of 0.2

3-1-2 Random Walk

The Random Walk (RW) algorithm is one of the simplest optimization algorithms. In short, the process is to randomly perturb the current solution until a better one is found and repeat this process with this new solution. Important choices for this algorithm are the type of random noise and its distribution. In general a good choice is normally distributed zero mean white noise where the variance will effectively determine the step size of the algorithm. Some of the disadvantages of the RW algorithm are its strong dependency on starting position, the requirement of many cost function calculations and not being able to escape local minima easily. Several adaptations can be made to the algorithm to improve its performance. For example, The random step can be applied negatively to the current solution if the step does not improve the current performance. Another possibility is to use a variable step size, which can be based on the iteration number or how much the previous step improved the solution. The algorithm written as pseudo code can be seen in Algorithm 1. Here the x is the state and $\hat{x}$ is the perturbed state, c is the step size, k is the number of iterations, $J()$ is the cost function.

Algorithm 1 Random Walk

```
1: procedure RANDOM WALK($x, c, k$)
2:     while $i \leq k$ do                          ▷ run for k iterations
3:         $\hat{x} = x + cN$                        ▷ where $N \in \mathcal{N}\{1, 0\}$
4:         if $J(\hat{x}) < J(x)$ then
5:             $x = \hat{x}$
6:     return $x$
```

3-1-3 DONE

The DONE algorithm was first proposed in [19] and later described in its own paper [20]. The DONE algorithm can be devided into three main steps. First, it uses a set of measurements from the cost function which must be optimized to generate a surrogate function, called the Random Fourier Expansions (RFE) function. The second step is to find the minimum of this surrogate function with a gradient optimization algorithm. The starting point for this algorithm is determined around the last added datapoint by adding some noise. The third step is to add the outcome of the second step to the data set. after this the algorithm is repeated. A more detailed technical description of the algorithm can be found in [21]. The DONE algorithm is very useful in situations where calculating the value of the cost function is very costly, or the result contains noise. Furthermore the computational complexity does not depend on the number of previous measurements, as adding a point to the surrogate function is an iterative process. An important part of setting up the DONE algorithm is making sure the cost function has a negative minimum. Because the error covariance matrix P is positive semi-definite, the trace and maximum eigenvalue will allways be positive. To remedy this, a negative offset is added. A rule of thumb provided by the author of [20] for this offset is twice the value of the starting position. For the sake of recollection, we provide the DONE algoritm as presented in [21] again, including the pseudocode in Algorithm 3:

Determining the RFE function

The RFE function is of the form

$$g(x) = \sum_{k=1}^{D} c_k \cos(\omega_k^T x + b_k), \tag{3-1}$$

with $D \in \mathbb{N}$ the number of functions,b_k being realizations of independent and identically distributed, uniformly distributed random variables B_k on $[0, 2\pi]$ and with the $\omega_k \in \mathbb{R}^d$ being realizations of i.i.d. random vectors Ω_k with an arbitrary continuous pdf. The ideal weights $\bar{c}$ depend on the Fourier transform of the unknown function f that we wish to approximate. Now assume a finite set of measurement points $x_1, x_2, \ldots, x_N$ that have been drawn independently on a pdf p_X that is defined on a compact set $\mathcal{X} \in \mathbb{R}^d$ and corresponding measurements $y_1, \ldots, y_N$, with $y_n = f(x_n) + \eta_n$. The weights of c_k are determined by minimizing the mean squared error.

$$J_N(c) = \sum_{n=1}^{N} \left(y_n - \sum_{k=1}^{D} c_k \cos(\omega_k^T x_n + b_k) \right)^2 + \lambda \sum_{k=1}^{D} c_k^2, \tag{3-2}$$

$$= ||y_N - A_N c||_2^2, \tag{3-3}$$

with λ a regularization parameter to deal with noise, overfitting and ill-conditioning, and

$$y_n = [y_1, \cdots, y_N]^T \tag{3-4}$$

$$A_N = \begin{bmatrix} \cos(\omega_1^T x_1 + b_1) & \cdots & \cos(\omega_D^T x_1 + b_D), \\ \vdots & \ddots & \vdots \\ \cos(\omega_1^T x_N + b_1) & \cdots & \cos(\omega_D^T x_N + b_D) \end{bmatrix}. \tag{3-5}$$

The solution for c_N for this least squares problem is

$$c_N = (A_N^T A_N + \lambda I_{D\times D})^{-1} A_N^T y_N. \tag{3-6}$$

As the problem is expanded by a single point each iteration, recalculating the entire problem is inefficient. A more efficient approach is to update the RFE with the new measurement. This is done by employing a recursive least squares algorithm. Each iteration a new measurement y_n at x_n is taken which is used to update the RFE. Let $a_n = [\cos(\omega_1^T x_n + b_1) \cdots \cos(\omega_D^T x_n + b_D)]$ then the weights of the RFE can be found by minimizing regularized MSE

$$J_n(c) = \sum_{i=1}^{n} (y_i - a_c)^2 + \lambda ||c||_2^2. \tag{3-7}$$

Let c_n be the optimum of iteration n, then the goal is to calculate c_{n+1} without finding the optimum for Equation (3-7) again. The recursive least squares solution for this problem is defined as

$$c_0 = 0, \quad P_0 = \lambda^{-1} I_{D\times D}, \tag{3-8}$$

$$\gamma_n = 1/(1 + a_n P_{n-1} a_n^T), \tag{3-9}$$

$$g_n = \gamma_n P_{n-1} a_n^T, \tag{3-10}$$

$$c_n = c_{n-1} + g_n(y_n - a_n c_{n-1}), \tag{3-11}$$

$$P_n = Pn - 1 - g_n g_n^T / \gamma_n. \tag{3-12}$$

Instead of performing these update rules explicitly, a rotation matrix Θ_n is found to solve the following equation.

$$\begin{bmatrix} 1 & a_n P_{n-1}^{1/2} \\ 0 & P_{n-1}^{1/2} \end{bmatrix} \Theta_n = \begin{bmatrix} \gamma_n^{-1/2} & 0 \\ g_n \gamma_n^{-1/2} & P_n^{1/2} \end{bmatrix}. \tag{3-13}$$

The rotation matrix can be found by QR factorization of the transpose of the left hand side of Equation (3-13). The RFE update step is written in pseudocode in Algorithm 2.

Algorithm 2 updateRFE

```
1: procedure UPDATERFE($c_{n-1}, P_{n-1}^{1/2}, a_n, y_n)$)
2:     Retrieve $g_n {\gamma_n}^{-1/2}, {\gamma_n}^{-1/2}$and $P_n^{1/2}$ from the QR factorization
3:     $c_n = c_{n-1} + g_n(y_n - a_n c_{n-1})$
4:     $g(x) = [\cos(\omega_1^T x + b_1) \ldots \cos(\omega_D^T x + b_D)]c_n$
5:     return $g(x)$
```

Calculating the optimum

Now that the surrogate function $g(x)$ is determined, the optimum of this function is determined. The method used is the L-BFGS method, which is a low memory using variation on the BFGS method described in [21]. This is a gradient based algorithm, which is a logical choice as $g(x)$ is differentiable. The starting point for the optimization is determined by

$$x_{init} = P_{\mathcal{X}}(x_n + \zeta_n), \tag{3-14}$$

$$\zeta_n \in \mathcal{N}(0, \sigma_\zeta^2 I_{d\times d}), \tag{3-15}$$

$$\mathcal{X} = \begin{bmatrix} ub \\ lb \end{bmatrix}. \tag{3-16}$$

Where $P_{\mathcal{X}}$ is the projection operator onto $\mathcal{X}$. This is done to guarantee the starting point is within the solution domain. ζ_n is zero mean white noise with variance σ_ζ which is called the exploration parameter. The addition of noise to the starting point is meant to prevent the algorithm from starting in the point where the model was trained.

Determining the next measurement point

Now that the optimum of $g(x)$, $\hat{x}_n$, is found, the next step is to determine where the next measurement will be taken. Similarly to determining the starting point of the algorithm, noise is added to the optimum and it is projected on $\mathcal{X}$.

$$x_{n+1} = P_{\mathcal{X}}(\hat{x}_n + \xi_n), \tag{3-17}$$

$$\zeta_n \in \mathcal{N}(0, \sigma_\xi^2 I_{d\times d}), \tag{3-18}$$

$$\tag{3-19}$$

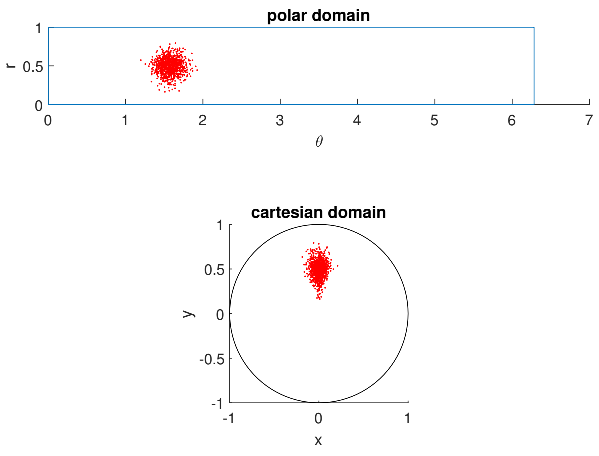
where σ_ξ is also an exploration parameter. The noise is meant to avoid the algorithm getting stuck in local minima or saddle points. Increasing σ_ξ will increase the "exploration" capabilities of the algorithm but might lead to slow convergence.

Algorithm 3 DONE algorithm

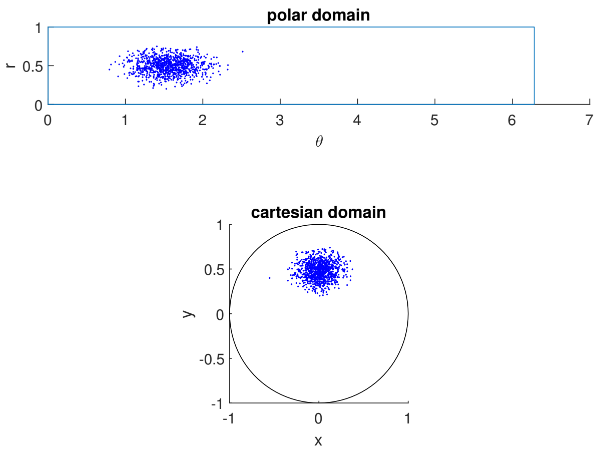
```
1: procedure DONE($f, x_1, N, lb, ub, D, \lambda, \sigma_\zeta, \sigma_\xi$)
2:     Draw $\omega_1 \dots \omega_D$ from $p_\omega$ independently
3:     Draw $b_1 \dots b_D$ from Uniform(0,2$\pi$) independently
4:     $P_0^{1/2} = \lambda^{1/2} I_{D\times D}$
5:     $c_0 = [0 \dots 0]^T$
6:     $\hat{x}_0 = x_1$
7:     for $n = 1, 2, 3, \dots, N$ do
8:         $a_n = [\cos(\omega_1^T x_n + b_1) \dots \cos(\omega_D^T x_n + b_D)]$
9:         $y_n = f(x_n) + \eta_n$
10:        Draw $\zeta_n$ from $\mathcal{N}(0, \sigma_\zeta^2 I_{d\times d})$
11:        $g(x) = \text{updateRFE}(c_{n-1}, P_{n-1}^{1/2}, a_n, y_n - 2y(x_{init}))$
12:        $x_{init} = \max(\min(x_n + \zeta_n, ub), lb)$
13:        $[\hat{x}_n, \hat{g}_n] = L - BFGS(g(x), x_{init}, lb, ub)$
14:        Draw $\xi_n$ from $\mathcal{N}(0, \sigma_\xi^2 I_{d\times d}$
15:        $x_{n+1} = \max(\min(\hat{x}_n + \xi_n, ub), lb)$
16:    return $\hat{x}_n$
```

3-2 Parametrization of optimization variables

In some cases it can be beneficial to reduce the number of variables to simplify the optimization problem. Another advantage is the fact that a lower number of optimization variables leads to a more intuitive problem. When parametrizing variables, a large number of variables can become dependent on small number of parameters without issue. There are some cases where the parametrization does lead to issues. When the variables depend on the parameters in a non-linear way, like when using polar coordinates for the parameter, and Cartesian coordinates for the variables, the performance of optimization methods which make use of random noise perturbations like the RW and DONE algorithms are affected negatively. An example of this can be seen in Figure 3-2a. A parameter set with values $r = 0.5, \theta = 0.5\pi$ is randomly perturbed by a zero mean normally distributed stochastic variable with a variance of 0.1. If these perturbed points are translated to the Cartesian domain, we see that the scattering of the random points is distorted, and that when these points would be used as-is in an algorithm, the exploration by DONE of the candidate points in RW would be placed unfairly, limiting the performance of the optimization algorithm. To obtain the perfectly 'fair' distributed random points we would need to perform relatively complex calculations for each point. An easy solution which performs relatively well is to simply scale the random variable along the θ axis by increasing the standard deviation for θ The result of scaling the random variables by a factor of 2 can be seen in Figure 3-2b. Performing such a scaling does not lead to perfect results for translating the distribution, but does improve performance of the optimization algorithms.



(a) Unscaled perturbations of a coordinate in polar vs Cartesian domain



(b) Scaled perturbations of a coordinate in polar vs Cartesian domain (scaling factor 2)

Figure 3-2: 500 random perturbations of a coordinate in the polar domain, scaled and unscaled, which are then translated to the cartesian domain.

3-3 Summary

In this chapter we described three optimization methods, each with their own advantages and disadvantages. In this section we provide an overview of the different methods and their tuning parameters.

DONE

The DONE algorithm constructs a surrogate function using measurements from the cost function. Optimizing this surrogate function is more efficient in the case where the measurements take a long time, or when they are noisy. There are several tuning parameters which must be chosen in such a way that the surrogate function represents the original cost function well enough to find the optimum. These parameters can be seen in Table 3-1. The number of iterations N determines how many measurements are taken. Increasing this number will increase computation time, but will also improve the surrogate function fit. The number of random variables, D, determines how well the surrogate function can be fitted to the cost function. Increasing this number will increase the computation time strongly. The distribution of these variables is in general chosen as a zero mean normal distribution. The choice of relaxation coefficient depends on how much noise is present in a measurement and the confidence in the choice of the other parameters. The exploration coefficients must be chosen in such a way that the algorithm does converge, but does not get stuck in local minima.

parameter	use
N	Number of iterations
D	Number of random variables
$p(x)$	Distribution of random variables
λ	Fitting relaxation coefficient
σ_ζ	Exploration coefficient for optimization start
σ_ξ	Exploration coefficient for addition to the cost function
x_1	Starting point

Table 3-1: DONE tuning parameters and their use

Random Walk

The RW algorithm is much simpler than the DONE algorithm, with only three tuning parameters, which can be seen in Table 3-2. When the variance is chosen we must consider two things. A very low variance will lead to slow convergence, requiring many iterations. A high variance will result in a high chance of the algorithm terminating far from the optimum.

parameter	use
x	Starting point
c	Variance of the random addition
k	Number of iterations

Table 3-2: RW tuning parameters and their use

Grid Search

For the standard grid search we only have one tuning parameter, which is the grid size. The grid size much be chosen in such a way that the resulting data represents the original cost function relatively well. decreasing the grid size will result in a quadratic increase in computation time. A simple addition to the algorithm would be to have different spacing in different directions.

parameter	use
s	Grid size

Table 3-3: GS tuning parameters and their use

Chapter 4

Observer and control performance

In the previous chapters, we saw that we can predict the performance of an observer or closed loop system with the use of DARE solutions. These solutions give information on the performance in the case that the model is perfect. In practice, this will never be the case. Furthermore, if we would like to compare the performance of systems with different model structures and different cost functions, we need a general method to signify the performance. A method to signify the performance of a model or observer is Variance Accounted For (VAF). The VAF is a score between 0 and 100 percent based on the ration between an error signal, and the original signal. The formula for the VAF is as follows:

$$\mathrm{VAF}_i = \left(1 - \frac{\mathrm{var}(\alpha_i - \hat{\alpha}_i)}{\mathrm{var}(\alpha_i)}\right) * 100. \tag{4-1}$$

Where α_i is the ith element of original signal and $\hat{\alpha}_i$ is the the ith element of the estimated signal. In systems with many outputs or states to estimate, it can be useful to have performance measure which depends on all variables. In that case the mean VAF of all the signals does not represent the performance fairly in all cases, as estimating high amplitude outputs or variables can be more important for the total performance than estimating low amplitude signals. In that case it can be useful to implement a weighted VAF. A VAF which is weighted for the mean amplitude is defined as follows:

$$\mathrm{VAFw} = \frac{\sum\limits_{i=1}^{n} \sqrt{\mathrm{var}(\alpha_i)} \left(1 - \frac{\mathrm{var}(\alpha_i - \hat{\alpha}_i)}{\mathrm{var}(\alpha_i)}\right) * 100}{\sum\limits_{i=1}^{n} \sqrt{\mathrm{var}(\alpha_i)}} \tag{4-2}$$

4-1 Overview

In this section we present an overview of the cost functions discussed in Chapter 2 for the two different optimality types. The A-optimality cost functions are used to optimize all variables at the same time. This cost function for the Linear Time Invariant (LTI) case is:

$$J_A = trace(P). \tag{4-3}$$

Where P is the estimation error covariance. The cost function for the Linear Parameter Varying (LPV) case is:

$$J_A = \frac{1}{N}\sum_{i=1}^{N} trace(P(\mu_i)). \tag{4-4}$$

Where $P(\mu_i)$ is the error covariance at the ith operating point.

The E-optimality cost functions are used when we want to optimize the "worst case" variable, or optimize for a lower bound on performance. The E-optimality cost function for the LTI model is:

$$J_E = \overline{\lambda}(P). \tag{4-5}$$

Where P is again the estimation error covariance. The E-optimality cost function for the LPV case is:

$$J_E = \max \overline{\lambda}(P(\mu_i)). \tag{4-6}$$

Where $P(\mu_i)$ again is the error covariance at the ith operating point.

Part II

Application to adaptive optics

Chapter 5

Adaptive optics setup and simulation

To demonstrate how the placement framework is meant to be used, we need a real life problem to implement. Adaptive optics is a good example, because the disturbances which must be controlled can be measured on a continuous domain, which is a requirement for placement. Furthermore, Adaptive optics systems employ many sensors and actuators, making the placement problem more interesting due to the many variables. In this chapter, we first give an overview of the adaptive optics setup, followed by a detailed description of each of its parts, and show how the system is simulated.

5-1 Schematic AO setup

Using a telescope, we would like to produce sharp images of an object of interest which is far away in outer space, which we will call the source. These images are taken by a scientific camera. The light from the source is often quite dim, which requires long shutter times. Due to turbulence in the atmosphere the light from the source is dynamically disturbed, causing our images to become blurry. This turbulence is something which is visible with the naked eye in some situations as well. When you look at the stars on a clear night, they can be seen flickering slightly, or during a hot sunny day it is possible to see the distorted light above asphalt streets, or building rooftops. These disturbances to the light are often called wavefront aberrations. To remedy the blurry space images caused by the distorted wavefront, we can employ an Adaptive Optics (AO) setup. Such a setup can be seen in Figure 5-1. As the light from the source passes through the atmosphere, it is distorted by turbulence. As the light passes through the telescope, it is reflected by a deformable mirror. This mirror can be used to counteract the aberrations leading to a completely undisturbed image in the ideal case. After reflecting off the deformable mirror, the light passes through a beam splitter, which sends the light to wavefront sensor and the imaging camera. Speaking in classical control terms for this setup, the deformable mirror contains the actuators, the wavefront(WF) sensor contains the sensors and the state is the residual wavefront which exists between the actuators and sensors. To demonstrate the placement framework, we will completely focus on adapting the wavefront sensor to optimize the wavefront reconstruction, as for the sensors we can reconstruct the wavefront at locations we do not measure but we cannot influence the wavefront at locations we do not actuate.

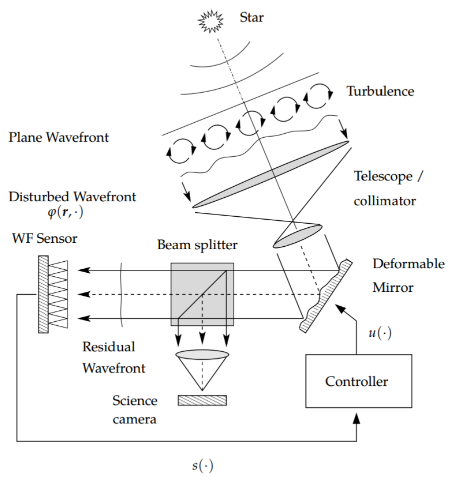


Figure 5-1: (From [22]) The adaptive optics setup for wavefront correction

5-2 Atmosphere model and simulation

We consider a "Frozen flow" atmosphere model[23] with two layers at different height with the same direction and speed. The distortions caused by this model and observed at the telescope are called the wavefront, denoted by ϕ. This wavefront is simulated using the Object-Oriented Matlab Adaptive Optics (OOMAO) toolbox[24]. The toolbox can simulate all the parts in an adaptive optics setup, but for our purposes we will only use the wavefront simulation part of the toolbox. There are two approaches to describing the wavefront, zonal and modal. With the zonal approach, the wavefront is described as values at locations, like an image. The modal approach uses a set of orthogonal base functions, which can be used to reduce the number of variables. We will take a look at two sets of basis functions, the Zernike base, which is the standard in adaptive optics, and a more general approach which could also be used for different systems. The coefficients of the Zernike basis are generated by OOMAO. The second base is obtained through performing a Singular Value Decomposition (SVD) on a time series of wavefront images. This technique is called Principal Component Analysis (PCA) and has many (statistical) applications, from finance to facial recognition.

5-2-1 Zernike base

The Zernike polynomals are a set of radially orthogonal basis functions. The functions have an azimuthal and radial degree. To be able to order these polynomials, a sequential index was introduced by Robert Noll[25]. Several other indices exist, but in this thesis all indices are assumed to be the Noll index. Another assumption is that all polynomials are normalized.

The Zernike polynomials can be used to describe the wavefront as follows:

$$\phi = \sum_{i=2}^{\infty} \alpha_i Z_i, \tag{5-1}$$

Note that the first index is skipped, as it has no influence on wavefront distortion. In practice, a truncated set of Zernike polynomials is used to describe a wavefront. The number of polynomials to be used is normally determined from the properties of the deformable mirror. Using the OOMAO toolbox we can obtain a time series of $\alpha \in \mathbb{R}^n$, where n is the chosen number of polynomials, on which we can perform system identification. In Figure 5-2 the 2nd to 17th zernike functions are shown.

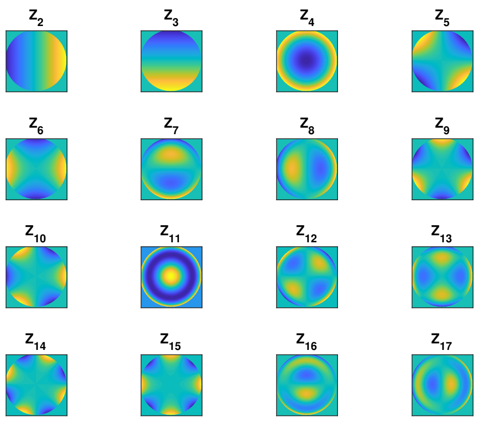


Figure 5-2: Zernike function 2 through 17

5-2-2 Principle Component Analysis base

The principal component basis is obtained by performing an SVD on simulation data. First a time series of ϕ is obtained, with $\phi[k] \in \mathbb{R}^{n \times n}$ where n is the resolution. Each instance of $\phi[k]$ is then restructured into a vector and is placed into a matrix A. Now we perform the SVD on the matrix A, which results in matrices U, Σ and V. U is the orthogonal base which contains the basis functions with which the wavefront can be reconstructed. Σ contains the singular values. These values signify how much each base function is represented in the wavefront. V contains the temporal data of the time series $\phi[k]$.

$$A = \begin{bmatrix} | & | & | & | \\ \mathrm{vec}(\phi[1]) & \mathrm{vec}(\phi[2]) & \dots & \mathrm{vec}(\phi[N]) \\ | & | & | & | \end{bmatrix} \tag{5-2}$$

$$A = U\Sigma V^T \tag{5-3}$$

After obtaining U, we can calculate coefficients similar to those of the Zernike Polynomials by projecting each time step of $\phi[k]$ onto U. If we restructure the columns of U into $n \times n$ matrices, we obtain the base images with which we can reconstruct the wavefront image. The first 16 images can be seen in Figure 5-3. In Figure 5-4 the singular values corresponding to the modes are plotted. It can be seen that the values stay quite high even for very high order modes, finally dropping off after the 811th value, which are the number of pixels within the circular aperture when we take $n = 32$.

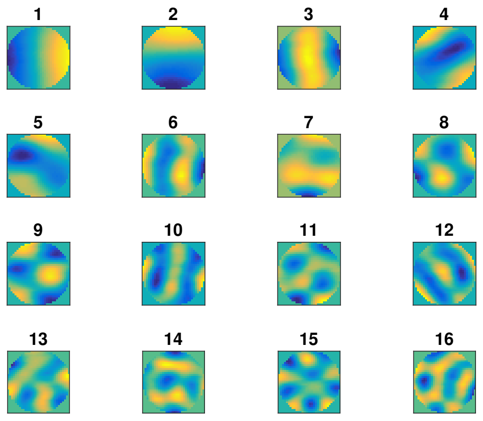


Figure 5-3: First 16 base images obtained by SVD of the wavefront images

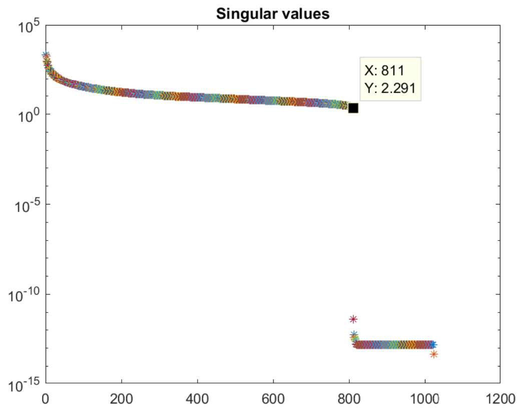
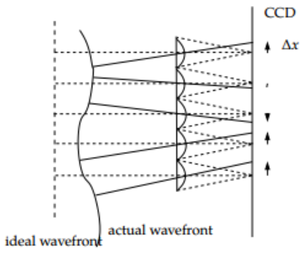


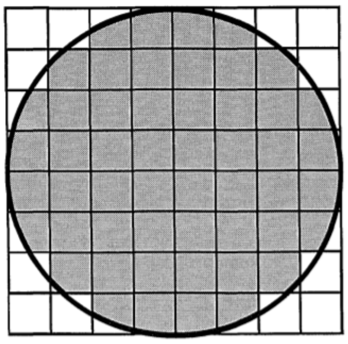
Figure 5-4: Singular values obtained by SVD of the wavefront images

5-3 The Shack-Hartman sensor

The sensors employed by an adaptive optics system are within the Shack-Hartmann (SH). The SH sensor consists of a CCD or CMOS sensor superimposed by an array of lenslets, which focus the light as spots on the sensor. The wavefront distortion causes the focused light to deviate from the center of the lenslet, causing the absorbed flux to deviate for adjacent pixels, see Figure 5-5a. The output of the SH sensor thus can be seen as having an output related to the angle or derivative of the wavefront at that location. In this thesis we consider a sensor to be a group of sensors under a single lenslet with a differential output. As such, each placed sensor will have two outputs which are related to the wavefront slope in perpendicular directions. The measurement equations for a sensor can be seen in Equations (5-4) and (5-5). Here we use the values of the Zernike polynomials at the sensor location to calculate the measurement.



(a) (From [22]) Sideview of the SH sensor



(b) (From [26]) A frontal view of a SH sensor with 8x8 lenslets superimposed on a circular aperture.

Figure 5-5: Schematic frontal and side view of the SH sensors

$$s_{ix}(x_i, y_i) = \sum_{j=2}^{N} (Z_j(x_i - \frac{1}{2}d, y_i) - Z_j(x_i + \frac{1}{2}d, y_i))\alpha_j + v_i, \tag{5-4}$$

$$s_{iy}(x_i, y_i) = \sum_{j=2}^{N} (Z_j(x_i, y_i - \frac{1}{2}d) - Z_j(x_i, y_i + \frac{1}{2}d))\alpha_j + v_i, \tag{5-5}$$

Where d is the pixel pitch. To be able to use these measurement equations in the model structures of Chapter 2, we must rewrite them in matrix form. For brevity, we define a matrix with Zernike values at a set of sensor locations as

$$Zc(x, y) = \begin{bmatrix} Z_2(x_1, y_1) & Z_3(x_1, y_1) & \dots & Z_N(x_1, y_1) \\ Z_2(x_2, y_2) & Z_3(x_2, y_2) & \dots & Z_N(x_2, y_2) \\ \vdots & \vdots & \ddots & \vdots \\ Z_2(x_M, y_M) & Z_3(x_M, y_M) & \dots & Z_N(x_M, y_M) \end{bmatrix}, \tag{5-6}$$

$$\begin{bmatrix} s_{1x}(x_1, y_1) \\ s_{2x}(x_2, y_2) \\ \vdots \\ s_{Mx}(x_M, y_M) \end{bmatrix} = (Zc(x + \frac{1}{2}d, y) - Zc(x - \frac{1}{2}d, y))\alpha + v, \tag{5-7}$$

$$\begin{bmatrix} s_{1y}(x_1, y_1) \\ s_{2y}(x_2, y_2) \\ \vdots \\ s_{My}(x_M, y_M) \end{bmatrix} = (Zc(x, y + \frac{1}{2}d) - Zc(x, y - \frac{1}{2}d))\alpha + v, \tag{5-8}$$

where x, y are vectors containing the coordinates of the sensors, d is the pixel pitch, and v is the measurement noise.

5-4 Actuators

For the adaptive optics example, we cannot perform placement of actuators in a meaningful way. With the sensors we can recreate the wavefront in locations where no measurements are taken, but it is not possible to actuate the deformable mirror in locations where there is no actuator. A system where actuator placement would be possible is for example a resonance damping system on an airplane wing or windmill blade.

5-5 Simulation and validation using OOMAO

The OOMAO toolbox contains a set of standard constants which can be used when simulating, in addition to these we used some custom settings for the speed, height and direction of the two simulated disturbance layers. The size and view angle of the telescope were kept as the standard values in the toolbox. The values can be found in the code in Appendix A-1 and in Table 5-1. Most parameters are taken from [27], which describes the atmospheric parameters at Mauna Kea Hawaii, where the Keck observatory is located. The telescope diameter is chosen to be smaller than that of the telescopes at the Keck observatory, to be applicable to more telescopes which are currently in use.

variable	value
layer speed	30 m/s
height layer 1	500 m
height layer 2	5000 m
telescope diameter	8 m
Fried parameter	20^{-2} m
Outer scale	30 m
fractional R_0	0.2
sampling frequency	100 Hz

Table 5-1: Parameters for OOMAO

Using this setup with source, atmosphere and telescope a set of Zernike coefficients is generated which is used for validation. An example resulting simulated Zernike coefficients can be seen in Figure 5-6. One of the shortcomings of the OOMAO toolbox is that the way the atmospheric layers are generated prohibits a change of wind angle during simulation. As a workaround the dataset used for the Linear Parameter Varying (LPV) implementation consists of several simulations with constant μ. This results in a μ function which has a stair shape, which can be seen in Figure 5-7 $\mu = -0.5$ corresponds with $\theta = -\pi$ and $\mu = 0.5$ corresponds with $\theta = \pi$. $\theta = \pi$ is not part of the simulation data due to phase wrapping interfering with the LPV model fit. In Chapter 8 we will explain further why $\theta = \pi$ is not part of the identification data.

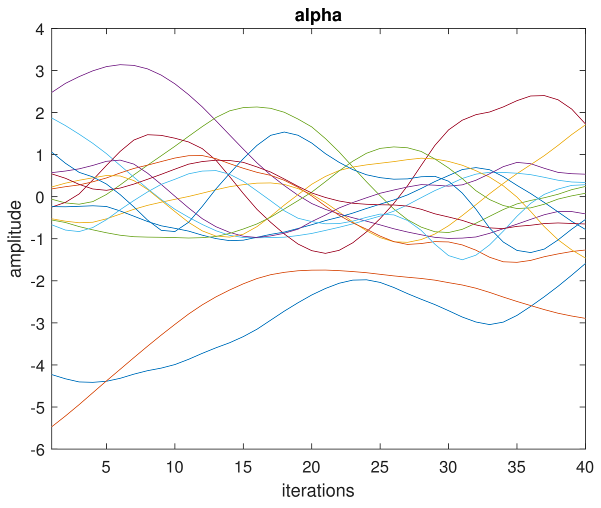


Figure 5-6: Zernike coefficients for 30 samples, which corresponds with 0.3 seconds

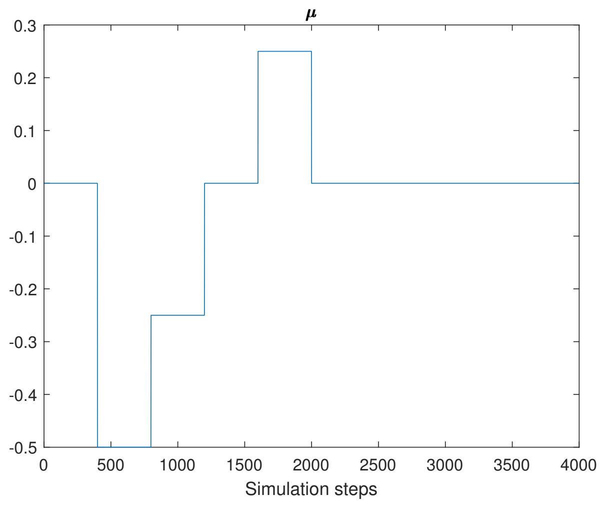


Figure 5-7: Values for μ during the simulation for the LPV identification data.

Chapter 6

Implementation using LTI model

The first implementation example will use an Linear Time Invariant (LTI) Vector Autoregressive (VAR) model. The optimization will be done using all provided algorithms and cost functions to compare their performance. After this we will parametrize the sensor locations to be dependent on only two variables to be able to interpret the solutions in a more intuitive way, and hope to be able to explain what the properties of a "good" solution are.

6-1 Identification

Due to the modal approach and the fact that we will construct our C output matrix separate from the identification, we have direct access to the state data. Because of this it makes sense to apply a VAR approach to the identification. The VAR model can be of any order, but in this case we choose a second order model as this produces the best fit, which was determined experimentally. The VAR function is written in Equation(6-1),

$$\underbrace{[\alpha_{k+3} \quad \alpha_{k+4} \quad \dots]}_{\alpha_y} = [A_1 \, A_2] \underbrace{\begin{bmatrix} \alpha_{k+2} & \alpha_{k+3} & \dots \\ \alpha_{k+1} & \alpha_{k+2} & \dots \end{bmatrix}}_{\alpha_x} + e, \tag{6-1}$$

where e is the error signal, α is the state sequence and A_1 and A_2 are are the coefficient matrices which must be determined through the optimization problem as defined in Equation (6-2),

$$\min_{A_1 A_2} ||\alpha_y - [A_1 \ A_2]\alpha_x||_F^2 + c||[A_1 \ A_2]||_F^2, \tag{6-2}$$

where c is a relaxation coefficient for the additional part to the right of the relaxation coefficient which promotes stable solutions. This problem is a convex least squares problem, and is solved using MOSEK[28]. After optimization we can construct our state space model using the A_1 and A_2 coefficient matrices, the measurement matrices from Section 5-3 for a specific set of sensor locations and$\sqrt{C_e}$, which is the Cholesky decomposition of the covariance matrix of error signal e. During the sensor location optmization process, we will augment our state space model simply by changing the C matrix. Our model now looks as follows:

$$\begin{bmatrix}\alpha_{k+1}\\ \alpha_k\end{bmatrix} = \begin{bmatrix}A_1 & A_2\\ I & 0\end{bmatrix}\begin{bmatrix}\alpha_k\\ \alpha_{k-1}\end{bmatrix} + \begin{bmatrix}\sqrt{C_e}\\ 0\end{bmatrix} w, \tag{6-3}$$

$$C(x,y) = \begin{bmatrix}Zc(x+\frac{1}{2}d,y) - Zc(x-\frac{1}{2}d,y)\\ Zc(x,y+\frac{1}{2}d) - Zc(x,y-\frac{1}{2}d)\end{bmatrix}, \tag{6-4}$$

$$s = C(x,y)[I \;\; 0]\begin{bmatrix}\alpha_k\\ \alpha_{k-1}\end{bmatrix} + v. \tag{6-5}$$

Where $C(x,y)$ is the measurement matrix for an arbitrary sensor set as determined in Chapter 5, w is the stochastic disturbance signal and α_k are the zernike Coefficients.

6-2 Observer

Now that we have our LTI model, we can design the static gain Kalman filter. There are two things we have not determined yet, the feedback gain L and the measurement noise R, which we need to solve Equation (2-5). We assume the measurement noise to be uncorrelated and equal for all sensors. σ is used to scale the measurement noise variance.

$$A_o = \begin{bmatrix}A_1 & A_2\\ I & 0\end{bmatrix} \tag{6-6}$$

$$C_o(x,y) = C(x,y)[I \;\; 0] \tag{6-7}$$

$$R = \sigma I \tag{6-8}$$

$$Q = C_e \tag{6-9}$$

Solving Equation (2-5) with the values above leads to the following state estimator:

$$\begin{aligned}\hat{\alpha}[k+1] &= (A_o - L(x,y)C_o(x,y))\hat{\alpha}[k] + L(x,y)s[k],\\ \hat{s}[k] &= C_o(x,y)\hat{\alpha}[k],\end{aligned} \tag{6-10}$$

6-3 Placement

For the placement of the sensors we consider a set of 9 sensors, using Cartesian coordinates x_i and y_i for each sensor this leads to 18 optimization variables. The number of sensors was chosen experimentally, as it is the lowest number of sensors which can be placed on a $n \times n$ grid which makes the system observable. This starting solution can be seen in Figure 6-1. The sensor locations are constrained to a circle around the origin with the convex optimization constraint

$$x_i^2 + y_i^2 \leq 1. \tag{6-11}$$

The reason for this constraint is the fact that in a conventional telescope this area is outside of the aperture.

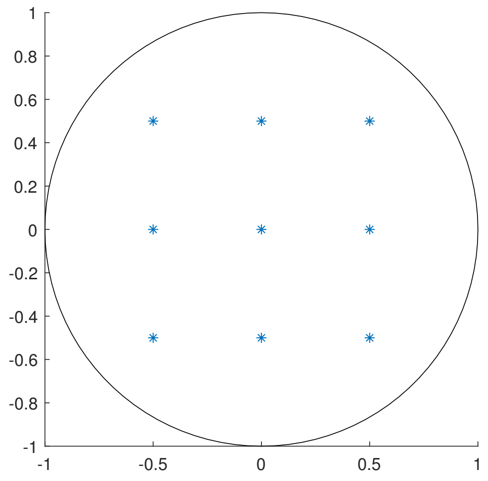


Figure 6-1: starting positions of the 9 sensors on a grid

6-4 Optimization

Now we can optimize the sensor locations. to illustrate the relative performance and differences between the methods provided in Chapter 3, we will implement all of them and compare their performance. The Data-based Online Nonlinear Extremumseeker (DONE) and Random Walk (RW) algorithm can be implemented directly as described. The grid search algorithm however, must be modified to be able to work for our cost functions. As the system is not observable for low amounts of sensors and it is impossible to calculate a Kalman filter for an unobservable system, we will not look for the best location to place each sensor consecutively, but start out with the configuration as depicted in Figure 6-2(blue dots), and look for the best place to relocate each sensor, starting with the top left and moving right and down, along a finer grid than the original.This finer grid is depicted by the red dots. This relocating of sensors is performed twice to save time, but could could be repeated until no sensor can be improved.

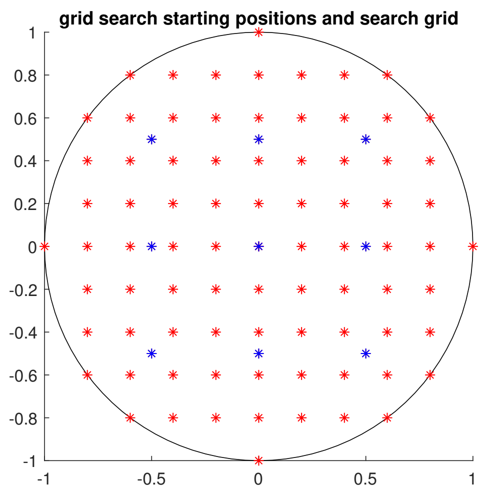


Figure 6-2: starting positions of the 9 sensors on a grid(blue) and search grid locations(red)

6-4-1 Behavior of the algorithms

Now we will take a look at the behavior and performance of the three implemented optimization algorithms. The results shown are for the placement problem with a trace cost function. All conditions will be kept the same for the different methods.

DONE

First we will take a look at the DONE algorithm. In Figure 6-3 the value of the cost function and the the DONE surrogate function at the evaluated point for that step is plotted for 200 steps. It can be seen that even after about 70-80 steps the algorithm is very close to its final value, but we chose 200 steps to increase the chance of finding the optimum. The value of the current point fluctuates as it randomly checks around the current optimum, making it a well suited algorithm for problems like ours which contain many local minima. These 200 iterations took 40.2 seconds, but the algorithm could have been terminated earlier.

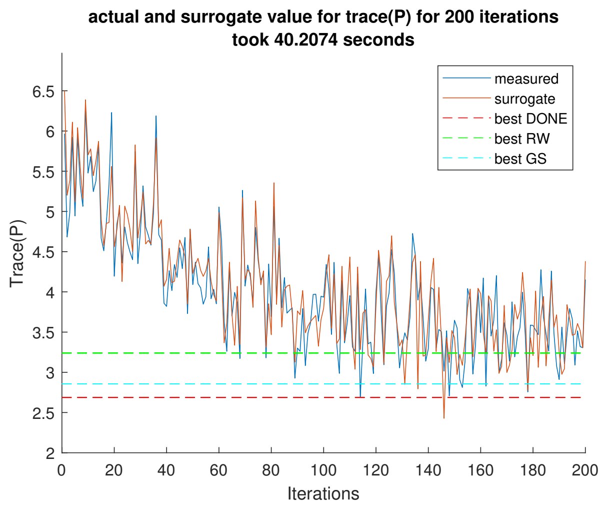


Figure 6-3: Value of trace(P) at points evaluated by DONE (measured and value of the surrogate function). The dotted lines represent the optimal values found by all methods

The values of the tuning parameters for DONE as described in Chapter 3 for the LTI model can be seen in Table 6-1. The distribution $p(x)$and number of functions D were kept the same as suggested by the author. The other parameters were determined experimentally. The Matlab code for the DONE algorithm can be seen in Appendix A-2.

parameter	value
N	200
D	1000
$p(x)$	$\mathcal{N}\{0,1\}$
λ	0.1
σ_ζ	0.05
σ_ξ	0.015
x_1	the starting configuration

Table 6-1: DONE tuning parameters for the LTI model

Grid Search

For the Grid Search (GS) algorithm the value of trace(P) is plotted for each move. After all the sensors were moved by searching top-down, the sensors were moved again, searching from the bottom up. Interesting to note is that this algorithm is the only one which places sensors on top of each other. Such a result could be interpreted as a single sensor with half the measurement error instead of two sensors. In our case this is not possible because each sensor will be exactly the same, and would require a contraint that limits the placement of sensors on top of each other. The other algorithms do not have a tendency to do this and do not need a constraint to keep the sensors apart. Looking at Figure 6-4, it can be seen that the algorithm can find a better position for each sensor at almost every step, even though not moving the sensor is a possibility as well. The grid size was chosen as 0.2, as this was the highest value whith reasonable computing time.

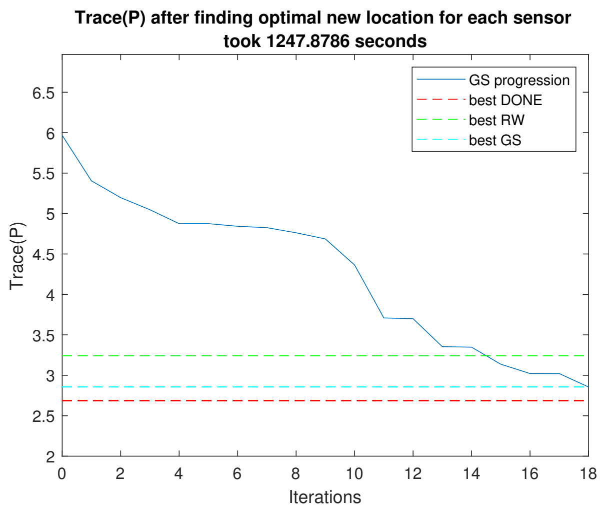


Figure 6-4: Value of trace(P) for grid search. The dotted lines represent the optimal values found by all methods

Random walk

The final algorithm, RW, was executed for 500 iterations. It can be seen in Figure 6-5 that after only 200 iterations the algorithm is very close to the final value (around 3.3). The RW algorithm performed the worst from the three algorithms. The tuning parameters for the RW algorithm can be seen in Table 6-2. Again, these parameters were chosen experimentally.

parameter	value
x	Starting point
c	0.01
k	500

Table 6-2: RW tuning parameters

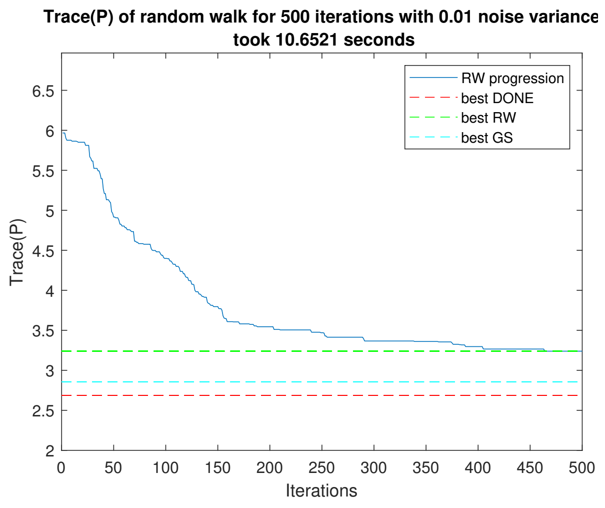


Figure 6-5: Value of trace(P) for random walk The dotted lines represent the optimal values found by all methods

The results produced by the algorithms seem to terminate at similar values, but the solutions found by the algorithms are not similar at all, take for example the solutions shown in Figure 8-2. These solutions correspond with the results in this section. From this it seems that the cost function has many local minima.

Chapter 7

Implementation using LPV model

For the second implementation we will use the Linear Parameter Varying (LPV) model to account for changing wind direction. The wind direction is chosen as a value between $-\pi$ and π. Other than the addition of different possible wind angles we will keep the aproach the same as the one for the Linear Time Invariant (LTI) model.

7-1 Identification

For the same reasons as stated in the LTI approach and to keep this implementation in line with the previous chapter, we will implement a second order Vector Autoregressive (VAR) model. The main difference with the previous chapter is that we will have to estimate four matrices in stead of two. We will denote the coefficient matrices with a subscript number for the VAR order and a superscript number for the LPV order. Our VAR will then be as follows:

$$\underbrace{\begin{bmatrix}\alpha_{k+3} & \alpha_{k+4} & \cdots\end{bmatrix}}_{\alpha_y} = \begin{bmatrix}A_1^0 + \mu_{k+2}A_1^1 & A_2^0 + \mu_{k+1}A_2^1\end{bmatrix} \underbrace{\begin{bmatrix}\alpha_{k+2} & \alpha_{k+3} & \cdots \\ \alpha_{k+1} & \alpha_{k+2} & \cdots\end{bmatrix}}_{\alpha_x} + e, \tag{7-1}$$

where μ_k is the normalized wind angle. The optimization problem is a convex quadratic problem as μ_k is known for all k. The optimization problem is then defined as:

$$\min_{A_1^0 A_1^1 A_2^0 A_2^1} \left|\left|\alpha_y - [A_1^0 + \mu A_1^1 \;\; A_2^0 + \mu A_2^1]\alpha_x\right|\right|_F^2 + c\left|\left|[A_1^0 + \mu A_1^1 \;\; A_2^0 + \mu A_2^1]\right|\right|_F^2. \tag{7-2}$$

After determining the cost function we can determine part of our LPV state space model. Similar to the LTI implementation, we now have all the necessary information except for the noise input matrix, that we call K, which will now become parameter dependent as well. Because of this, we will need to perform an additional optimization to determine the two parameters. Unfortunately, this process is much more difficult, as there is a Cholesky decomposition involved which makes the optimization non-convex. The model so far can be seen in Equations (7-3) to (7-5)

$$\begin{bmatrix}\alpha_{k+1}\\ \alpha_k\end{bmatrix} = \begin{bmatrix}A_1^0+\mu A_1^1 & A_2^0+\mu A_2^1\\ I & 0\end{bmatrix}\begin{bmatrix}\alpha_k\\ \alpha_{k-1}\end{bmatrix} + \begin{bmatrix}K\\ 0\end{bmatrix} w \tag{7-3}$$

$$C(x,y) = \begin{bmatrix}Zc(x+\frac{1}{2}d,y) - Zc(x-\frac{1}{2}d,y)\\ Zc(x,y+\frac{1}{2}d) - Zc(x,y-\frac{1}{2}d)\end{bmatrix}, \tag{7-4}$$

$$s = C(x,y)[I \;\; 0]\begin{bmatrix}\alpha_k\\ \alpha_{k-1}\end{bmatrix} + v. \tag{7-5}$$

The K matrix is assumed to be linearly dependent on μ. We rewrite Equation (7-3) to represent this, which can be seen in Equation (7-6). As we use a step function to vary the μ parameter, we can calculate the error covariance with the error signal during each μ step, which we will call $C_{e\mu}$. We can then minimize the cost function in Equation (7-7) to obtain the two LPV parameters K^0 and K^1.

$$\begin{bmatrix}\alpha_{k+1}\\ \alpha_k\end{bmatrix} = \begin{bmatrix}A_1^0+\mu A_1^1 & A_2^0+\mu A_2^1\\ I & 0\end{bmatrix}\begin{bmatrix}\alpha_k\\ \alpha_{k-1}\end{bmatrix} + \begin{bmatrix}K^0+\mu K^1\\ 0\end{bmatrix} w \tag{7-6}$$

$$\min_{K^0 K^1} \left|\left| K^0 + \mu K^1 - \sqrt{C_{e\mu}} \right|\right| \tag{7-7}$$

7-2 Observer

We implement an observer with online calculated feedback gain. The feedback gain is calculated using Equations (2-4) and (2-5), and the following time dependent parameters:

$$A_o[k] = \begin{bmatrix}A_1^0+\mu[k+2]A_1^1 & A_2^0+\mu[k+1]A_2^1\\ I & 0\end{bmatrix} \tag{7-8}$$

$$C_o(x,y) = C(x,y)[I \;\; 0] \tag{7-9}$$

$$R = \sigma I \tag{7-10}$$

$$Q[k] = (K^0+\mu[k]K^1)(K^0+\mu[k]K^1)^T \tag{7-11}$$

The time dependent state estimator then becomes:

$$\begin{aligned}\hat{\alpha}[k+1] &= (A_o[k] - L(x,y)[k]C_o(x,y))\hat{\alpha}[k] + L(x,y)[k]s[k],\\ \hat{s}[k] &= C_o(x,y)\hat{\alpha}[k],\end{aligned} \tag{7-12}$$

7-3 Placement

The placement for the LPV implementation is done the same way as for the LTI implementation.

7-4 Optimization

As the solution to the DARE is dependent on μ, we use the method described in Section 2-2-3 to calculate the Discrete Algebraic Ricatti Equation (DARE) solution offline. The Optimization is performed using the same algorithms as the LTI case. The parameters for the Random Walk (RW) and Grid Search (GS) case are kept the same. The conditions for the Data-based Online Nonlinear Extremumseeker (DONE) case differ slightly and can be seen in Table 7-1.

parameter	value
N	200
D	1000
$p(x)$	$\mathcal{N}\{0, 1\}$
λ	0.1
σ_ζ	0.5
σ_ξ	0.1
x_1	the starting configuration

Table 7-1: DONE tuning parameters for the LPV model

Chapter 8

Results

This chapter provides the results from implementing the previous two chapters in Matlab. The most important Matlab code can be found in Appendix A, other files can be provided by the author on request until 24-10-2019. The results for the implementations discussed in the previous two chapters will be discussed in the order they were introduced. We will look at the performance obtained using the solution found with each cost function and optimization method. Afterwards we will investigate the placement properties by parametrizing the sensor locations.

8-1 LTI Results

This section presents the results obtained by implementing Chapter 6. We will compare the performance of the system when optimized when using different cost functions, and investigate the performance under conditions which differ from the conditions when the system was identified. We will make use of the Variance Accounted For (VAF) and weighted VAF to compare different cost functions and give a more realistic comparison in regards of real life performance. We generate measurement data using the measurement equations from Chapter 5. These simulated measurements are perturbed with noise, to simulate measurement error.

8-1-1 Trace cost function

In Table 8-1 the VAF results for the different optimization methods are shown. The observer implemented with the functions in Chapter 6-2 was tested with a new dataset generated using OOMAO containing 2000 samples, equal to the size of the identification set. It is clear that all optimization methods perform better than the non optimized configuration at estimating the Zernike coefficients, and that in each case the results are more favorable after weighting although the difference is negligible, around 0.5 percent. The Data-based Online Nonlinear Extremumseeker (DONE) version performed the best, which was expected as it also had the lowest cost function value. The weighting has the most influence on the non optimized VAF.

method	mean VAF	weighted VAF
base	86.87	90.16
RW	95.22	95.78
DONE	95.25	95.94
GS	93.26	94.04

Table 8-1: VAF for solutions found using trace(P) cost function

8-1-2 Maximum eigenvalue cost function

The same optimization and simulation as performed for the trace cost function was performed using maximum eigenvalue cost function. The results can be seen in Table 8-2. The results are similar to the ones obtained with the trace cost function, however, it is clear that the trace results performed better for each method both in the mean and weighted case. Similar to the trace function the DONE method seems to give the best results for finding an optimum.

method	mean VAF	weighted VAF
base	86.87	90.16
RW	92.53	94.14
DONE	93.79	94.63
GS	91.72	93.29

Table 8-2: VAF for solutions found using max(eig(P)) cost function

8-1-3 Robustness to changing conditions

In the previous sections we assumed the conditions of the validation data to be exactly the same as the data used for the identification and optimization of the model. In this section we will look at the performance of the estimator under different wind angles. The data used for this optimization was generated with a constant wind angle for each evaluated point. We consider the wind angle at which the model was optimized to be 0 radians, and the clockwise deviation to be positive. The optimization was done using the trace cost function. The results can be seen in Figure 8-1. From this figure we can clearly conclude that the Linear Time Invariant (LTI) model does not perform well at all when the identification parameters change just even a little. Furthermore, the optimized locations quickly drop below the performance of the sensor starting positions. Because of this we cannot claim that optimal sensor locations found by this method are very robust.

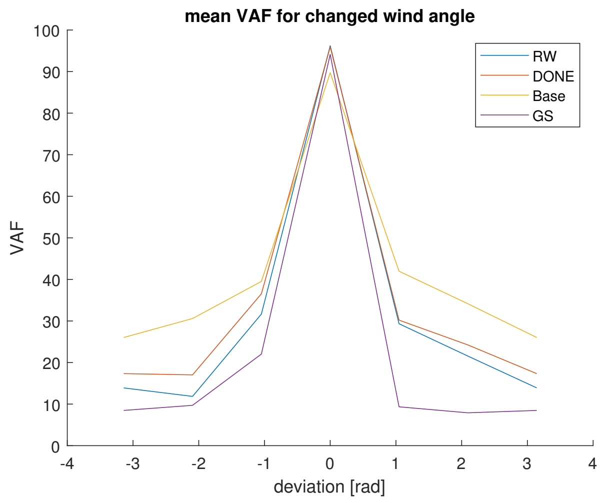


Figure 8-1: VAF for changed wind angle using trace(P) cost function for the LTI model

8-1-4 Parametrization of sensor locations

It would be interesting to know what kind of properties a good sensor placement has so we can intuitively decide what will likely become a good sensor placement and adjust the starting positions accordingly. The solutions which provided the results in Section 8-1-1 can be seen in Figure 8-2. For this simulation the wind moved from left to right. If we look at the solutions found by the different methods, we see that they have some things in common. Firstly it seems that sensors at the edge of the domain are preferred to sensors in the middle. Furthermore, it seems that sensors have a tendency to move towards the left side of the domain, where the wind originates.

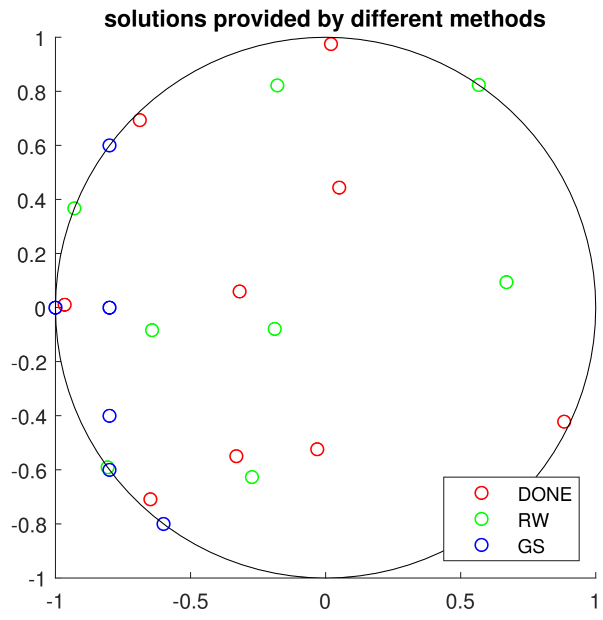


Figure 8-2: Solutions for different methods using the trace(P) cost function and LTI model.

To substantiate these claims and demonstrate these tendencies, we will parametrize the sensor locations. The sensors will be parametrized using two variables, one related to the radial position of each sensor, and one related to the angle of the sensor position to the negative x axis. An example of parametrically placed sensors can be seen in Figure 8-3. We use two groups of 5 sensors, one group placed at R and one at $\frac{1}{2}R$, for a total of 10 sensors. Optimization with the first set fixed at $R = 1$ results in an optimal location for the second set around these proportions. The sensors are divided evenly between θ and $-\theta$. This specific parametrization was chosen to accentuate the edge seeking properties of the placement. Other parametrizations could be for example scaling or shifting the starting positions from Figure 6-1. Another advantage to parametrizing the sensor locations is the fact that by reducing the number of variables a grid search algorithm becomes much more viable. We can check the entire domain of the optimization variables much more easily allowing us to determine if the minimum found by the algorithms is indeed the global minimum.

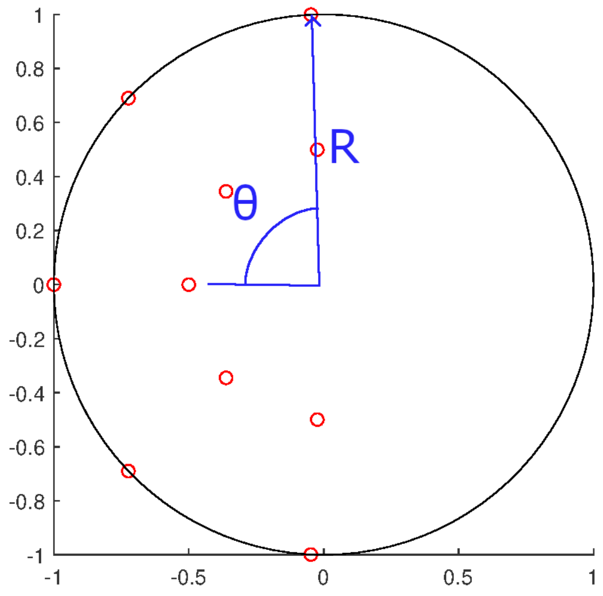


Figure 8-3: example sensor parametrization using two parameters, angle of the outer sensors θ with the negative x-axis and R the distance from the outer sensors from the centre.

The parametrized sensor locations were optimized for both cost functions in the same way as with the free placement. The VAF results for the trace cost function can be seen in Table 8-3, and the results for the maximum eigenvalue cost function can be seen in Table 8-4. Comparing these results to the results obtained by optimizing with free variables as shown in Tables 8-1 and 8-2, we see that the parametrized solutions perform better than the original solutions. This could partly be explained by the fact that the parametrized solutions employs 10 sensors while the original solutions use 9, even though this parametrization excludes most orientations. In the same way as with the non-parametrized solution, we can see that the trace cost function provides better results than the maximum eigenvalue cost function. Another interesting thing to note is that the performance of the found solutions with the parametrized sensor locations have VAF scores which are much closer together than with the free placement.

method	mean VAF	weighted VAF
RW	95.40	95.86
DONE	95.47	95.90
GS	95.52	95.91

Table 8-3: VAF for solutions found using trace(P) cost function

method	mean VAF	weighted VAF
RW	91.88	93.91
DONE	93.48	94.73
GS	92.10	94.05

Table 8-4: VAF for solutions found using max(eig(P)) cost function

To further investigate the solutions provided by the parametrized optimization process, we look at the solutions provided by the algorithms. The solutions found using the trace cost function that produced the results in Table 8-3 can be seen in Figure 8-4. "surface min" in the legend refers to the result obtained using the Grid Search (GS) algorithm. It can be seen that the solutions are very close together, which explains the performance of these solutions to be so similar.

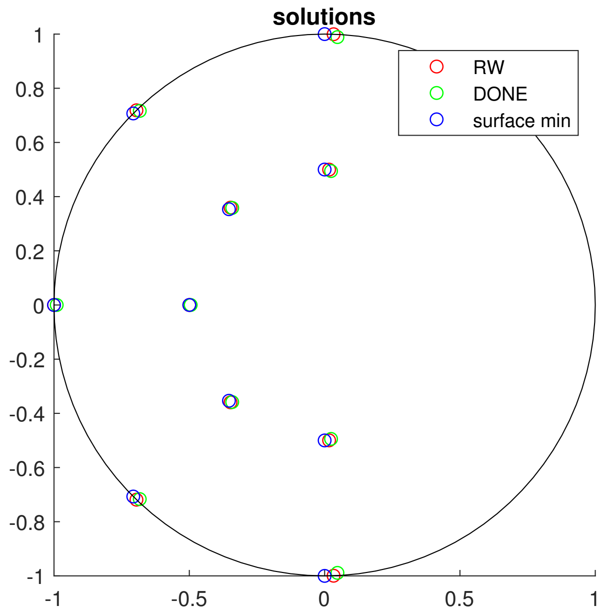


Figure 8-4: solutions for different methods using parametrization, trace(P) cost function

Because we reduced the number of variables to two, it has become much easier for the GS algorithm to check all values in the solution domain. Another advantage is that we now can plot the solutions for the domain in a way that is visually intuitive. Using the data obtained by the GS Algorithm, we can plot the surface of the cost function to better understand the algorithm behavior and to demonstrate that sensor locations at the outside and close to the wind source are indeed preferred over all other possible placements. The surface for the trace cost function can be seen in Figure 8-5. We can see there exists a clear global minimum to the cost function, and that higher values for R are preferred for all θ, and the angle at which the sensors are spread out matters less in comparison. The optimum angle places the sensors in such a way that they cover the y direction as much as possible, whilst keeping all of the sensors close to the wind source. We can use the surface plot together with a surface plot of the DONE surrogate function to see if the chosen parameters for DONE are able to produce a surrogate function which can sufficiently follow the original cost function. The surrogate surface is represented in Figure 8-5 as the semi translucent surface. As our original function is fairly smooth and does not have very steep slopes, the parameters do not require to be set very accurately. As visible in Figure 8-5, the surrogate function follows the function relatively well.

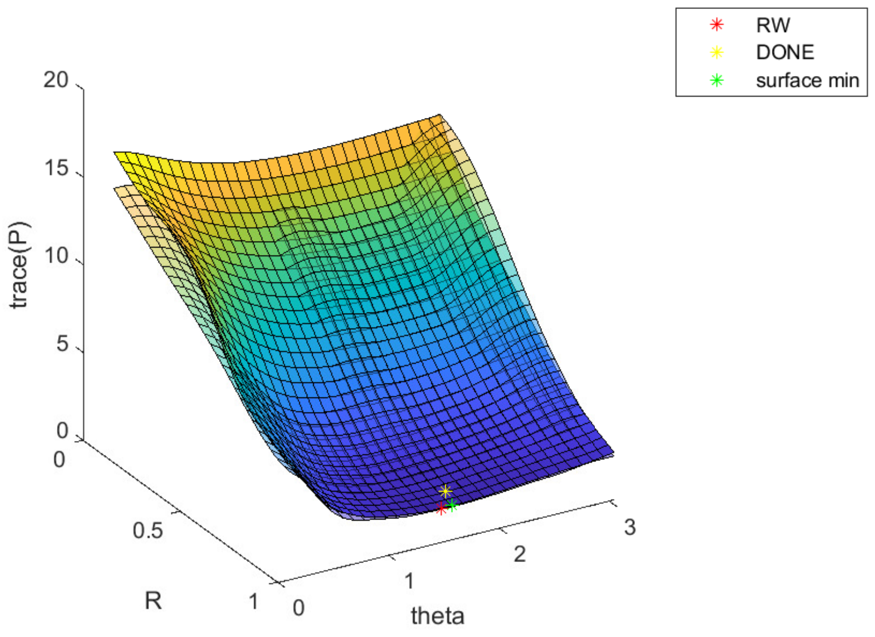


Figure 8-5: Trace(P) with parametrized solutions and DONE surrogate function(translucent surface).

We will check the robustness to changing wind conditions the same way as we have done for the free placement in Section 8-1-3, again varying the angle of the incoming wind between $-\pi$ and π, with the angle being zero at the direction where the identification was performed. The results can be seen in Figure 8-6. If we compare this figure with Figure 8-1, we see that the results are very similar, but in this case the different solutions perform almost identically, which is expected as the found solutions are very similar as well. We can draw the same conclusion, we sacrifice performance outside of the conditions during optimization to improve the performance under these conditions.

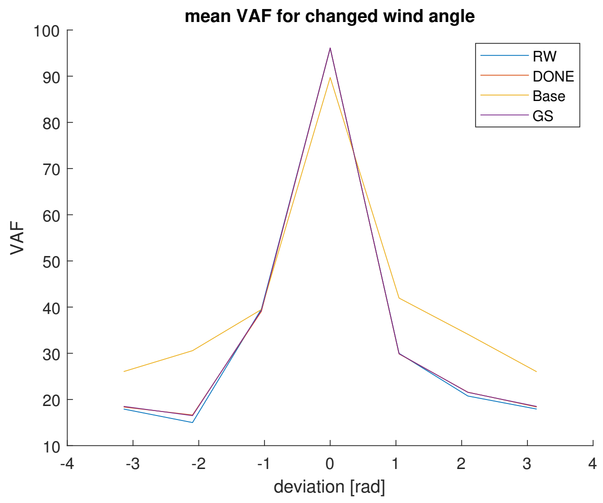


Figure 8-6: VAF for different methods and wind angle deviation, trace(P) cost function and parametrized solutions

The previous three images were all produced using the trace cost function, similar to the previous sections we now investigate the results obtained using the maximum eigenvalue cost function. The solutions found using this cost function can be seen in Figure 8-7. If we compare this figure to Figure 8-4, we see some differences. Both the optimal θ and R are lower than in the case of the trace cost function.

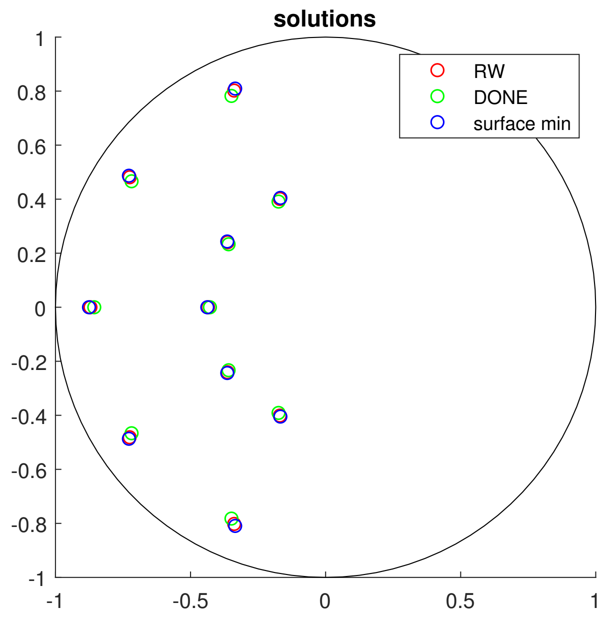


Figure 8-7: solutions for different methods with max(eig(P)) cost function and parametrized solutions

The surface found using the GS algorithm and the DONE surrogate function for the maximum eigenvalue cost function can be seen in Figure 8-8. Comparing this figure to Figure 8-5 we can see that the shape op the functions are very similar, both are smooth and have lower values for high R. If we look at the DONE surrogate function, we see that it differs much more from the actual cost functions in most of the plot except around the optimum. This happens in the case where DONE finds the optimum after a low number of iterations and does not move around much for the rest of the optimization.

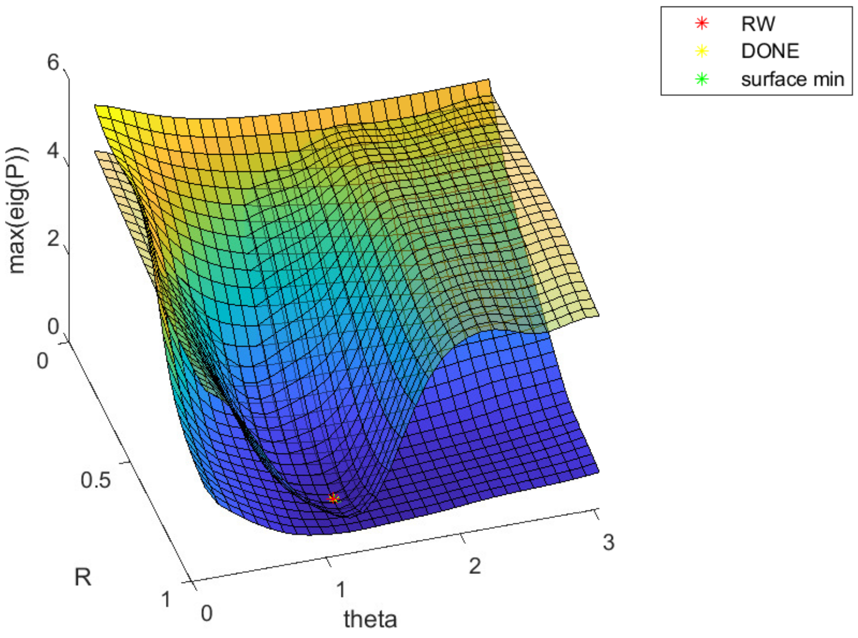


Figure 8-8: max(eig(P)) cost function with solutions parametrized and DONE surrogate function(translucent surface)

the VAF plot for deviating wind angle using the solutions from Figure 8-7 can be seen in Figure 8-9. Comparing this figure to the one obtained using the trace cost function, we draw the same conclusions which can be drawn by comparing Tables 8-3 and 8-4 or Tables 8-1 and 8-2. The trace cost function leads to a better VAF score than using the maximum eigenvalue. Overall, this could be explained by the fact that the VAF score is similar to the trace in a sense that both signify an average score instead of the worst case score the maximum eigenvalue score provides.

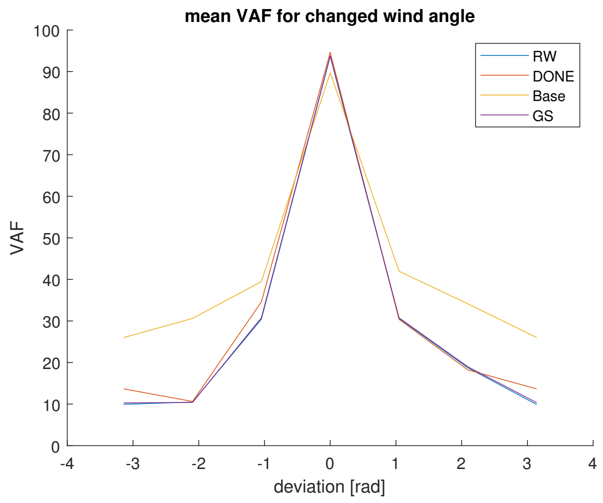


Figure 8-9: VAF for different methods and wind angle deviation, with max(eig(P)) cost function and parametrized solutions

8-2 LPV results

From the previous section with the results from the LTI implementation and especially Figures 8-1, 8-6 and 8-9, we can conclude that implementing this system in an actual telescope would require some work to be practical, because it would only work at times when the wind comes from the same direction as during identification. The LPV implementation uses the wind direction as a parameter, and therefore work under a larger set of circumstances. In this section we will investigate the effect of the use of the Linear Parameter Varying (LPV) model on the optimal sensor locations. We will not show any free solutions, as we would draw the same conclusions as in the previous chapter. The Resulting VAF for the entire wind angle range for the three optimization methods can be seen in Figure 8-10. We can see that the solutions found by DONE and GS perform best. The performance starts to drop fast for angles larger than 0.5π for all methods. This area is also the only area where the performance of the placed sensors does not improve. This drop can be explained by the fact that the identification data does not contain information for this wind angle, as can be seen in Figure 5-7. Unfortunately, omitting either data for $\theta = \pi$ or $\theta = -\pi$ is necessary to avoid problems introduced by wrapping. As the system is physically the same for $\theta = \pi$ and $\theta = -\pi$ the LPV identification would result in an impossible fit to the data.

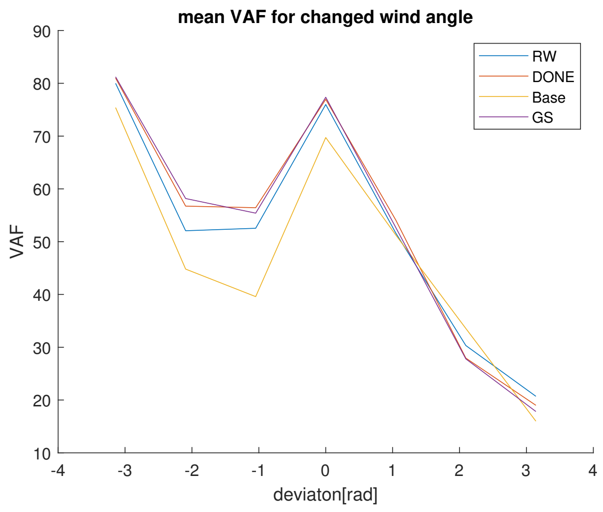


Figure 8-10: VAF for different wind angles with mean(trace(P)) cost function, free variables

8-2-1 Parametrization

We will now look at the parametrized results for the LPV model using the cost functions described in Section 2-2-3. First we will look at the mean trace, after that we will look at the results obtained using the maximum maximum eigenvalue function. We will present the same three figures per cost function as in the last section. the found solutions, the cost and surrogate functions, and the VAF performance for all wind angles. The solutions found using the mean trace cost function for all three optimization methods can be seen in Figure 8-11. We see that similar to the LTI case, the algorithms settle very close to each other. The sensor locations share one property with the LTI implementation, the optimal solution tends to be located at the edges of the aperture. The solution differs in the optimal angle, which now distributes the sensors almost equally over aperture, instead of focusing them close to the side of the wind direction.

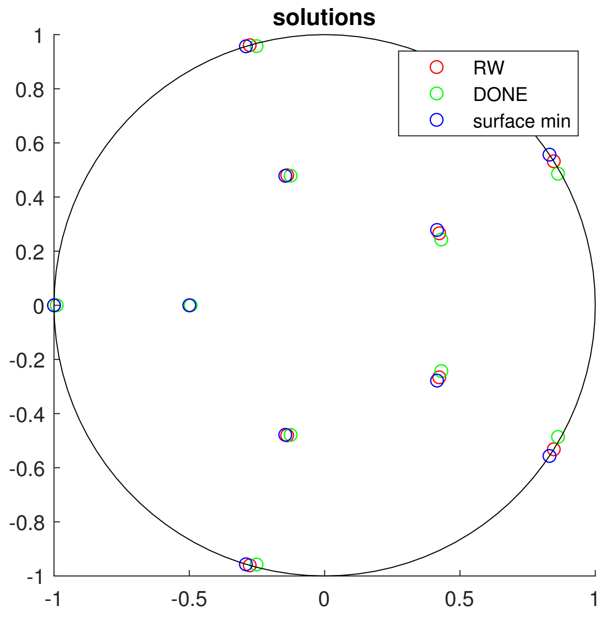


Figure 8-11: solutions for different methods, mean(trace(P)) cost function and parametrized solutions

The cost function for the LPV model with the mean trace cost function can be seen in Figure 8-12. As expected, it is similar to the cost functions found for the LTI model, with the difference being the optimal location. Again we see that the DONE surrogate function follows the cost function quite well.

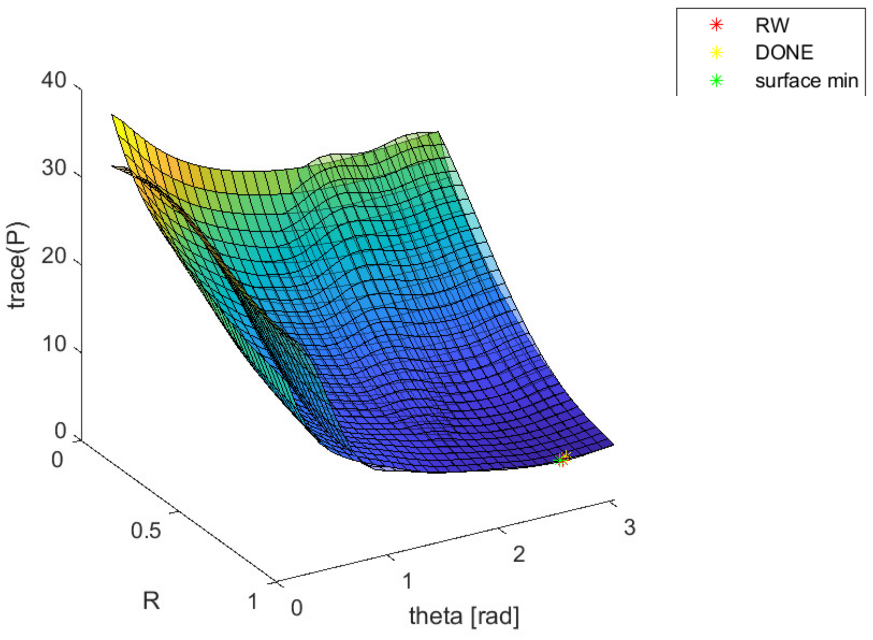


Figure 8-12: mean(trace(P)) cost function, parametric solutions, translucent surface is the DONE surrogate function.

The VAF for simulation of all wind angles can be seen in Figure 8-13. Again we see an increase for all wind angles. The system performs better than the freely placed sensors, which is not expected as we have less degrees of freedom. As we are comparing a setup with 10 sensors to a setup with 9, some of this improvement can be explained by the extra sensor.

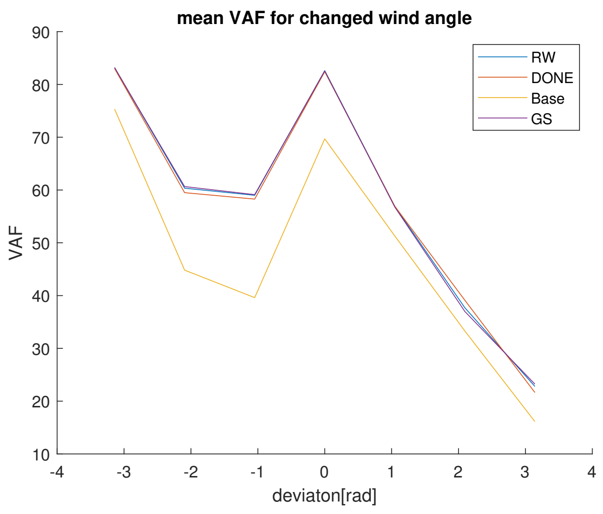


Figure 8-13: solutions for different methods, mean(trace(P)) cost function and parametrized solutions

8-2-2 Maximum eigenvalue cost function

We will now look at the placement results when using the maximum maximum eigenvalue cost function. We will look at the same three images, the found locations, the cost function and the VAF performance. The results found by the algorithms using the maximum maximum eigenvalue cost function can be seen in Figure 8-14. The results provided by the algorithms differ much more in this case.

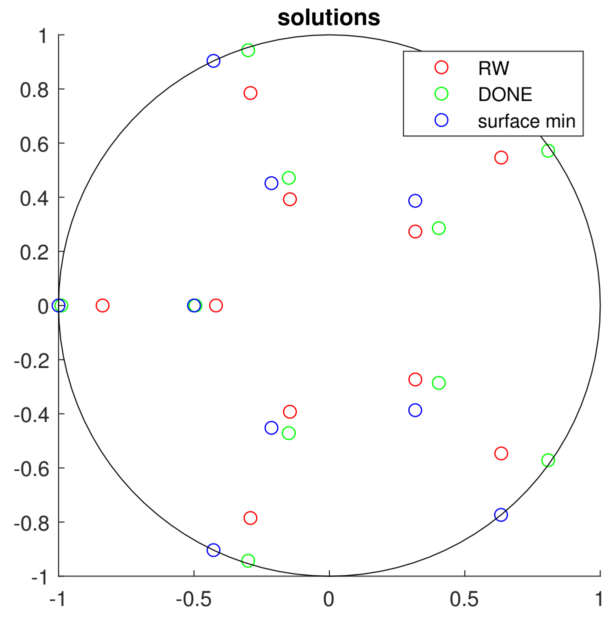


Figure 8-14: solutions for different methods, max(max(eig(P))) cost function and parametrized solutions

The maximum maximum eigenvalue cost function and its DONE surrogate function can be seen in Figure 8-15. Again, the surrogate function follows the original function quite well. If we compare the final positions, we see that the area around them is very flat. Comparing this area to the trace cost function in Figure 8-12, we see that it is much more flat. This could be the reason why the optimization algorithms could not find the same optimum.

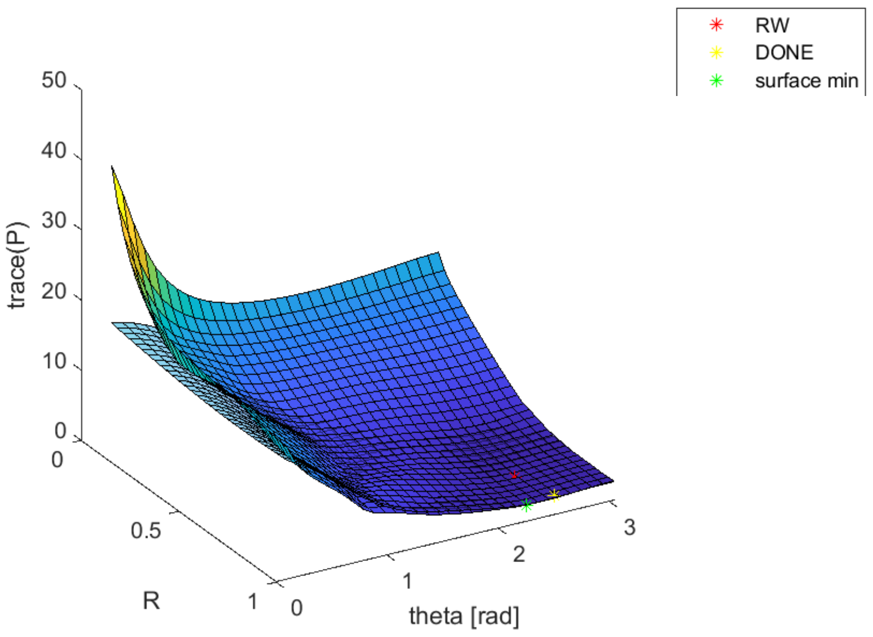


Figure 8-15: max(max(eig(P))) cost function, parametric solutions, translucent surface is the DONE surrogate function.

The VAF results for the maximum maximum eigenvalue cost function can bee seen in Figure 8-16. We see that the performance is similar to the performance of solution found by the mean trace cost function. The performance of the solutions is slightly lower than that of the solutions found by the mean trace cost function. Interestingly, the solutions from the DONE and GS algorithms perform almost the same.

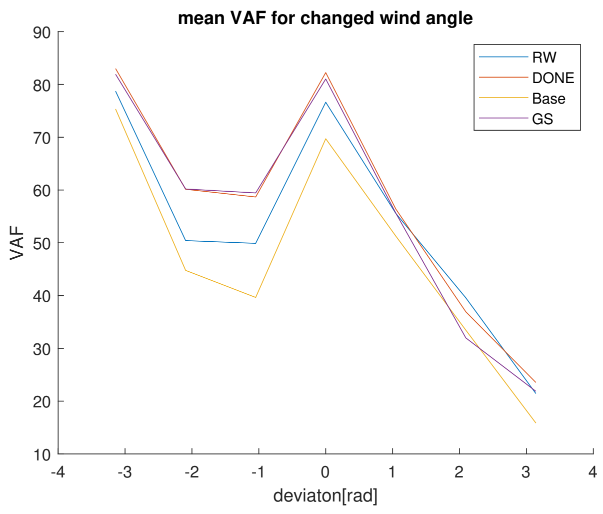


Figure 8-16: VAF for different methods, max(max(eig(P))) cost function and parametrized solutions

8-3 Comparison between LTI and LPV results

We will now compare the the results obtained by the LTI and LPV models. For all model forms and cost functions, The parametrized optima were positioned at the maximum R value. So in comparing the LTI and LPV models, we will consider the trace cost functions at $R = 1$. Figure 8-17 shows the trace cost functions for the LTI and LPV models. The functions are similar in shape, but have different minima as we saw in the previous section. The LPV minimum is at 2.55 radians, which means that the angle between the sensors is around $\frac{2}{5}\pi$ which distributes the sensors evenly over the aperture.

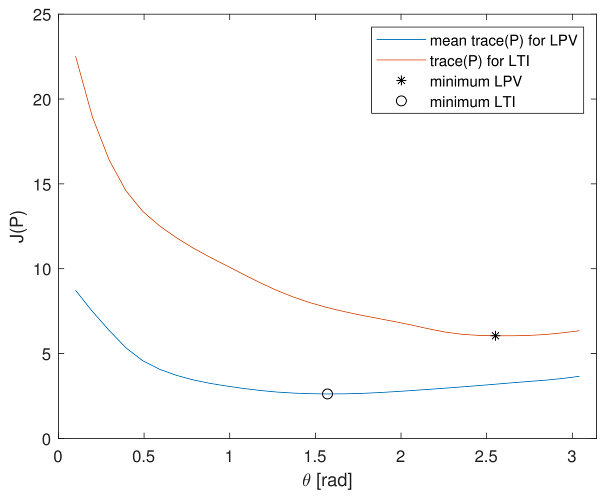


Figure 8-17: Parametrized cost functions for the trace cost functions for the LTI and LPV model at $R = 1$

If we now use the fact that the LPV placement creates a situation where we have sensors close to the edge for all wind directions. By running an LTI identification for for all wind angles and using this LTI model in combination with the LPV placement, we obtain the VAF performance in Figure 8-18. For this figure we used freely placed sensors and the trace cost function.

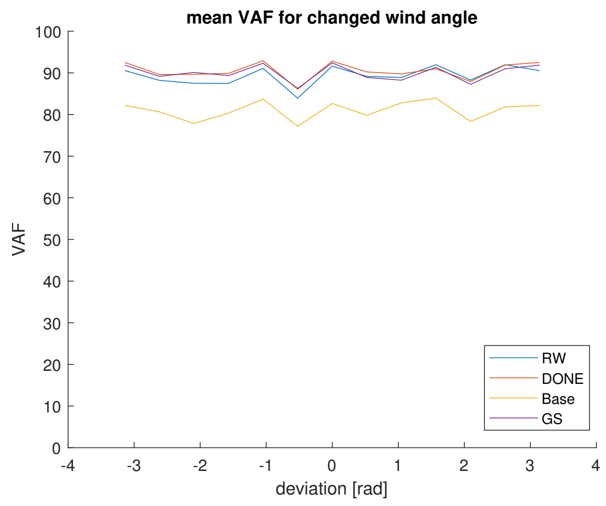


Figure 8-18: VAF performance when using the locations found for the LPV model with free placement and mean(trace(P)) cost function. The estimator uses an identified LTI model specific for each angle.

Chapter 9

Conclusion and recommendations

Conclusion

The objective of this thesis was to develop a framework for robust optimal placement of sensors and actuators. The goal of these placements is to improve the performance of the observers and controllers without increasing the number of sensors and actuators. To this end, we described how model structures should be chosen, which optimization methods and what cost functions can be used to reach this goal. In the second part of the thesis, this framework was applied to wavefront estimation for an adaptive optics setup using two different models.

First, we provided two model structures in Chapter 2, a Linear Time Invariant (LTI) and Linear Parameter Varying (LPV) model structure. For each structure an observer was described, with the required parameters for implementing them. Furthermore we provided cost functions which can be used to optimize the observers. Chapter 3 describes three optimization methods which can be used to optimize the models provided in Chapter 2. Chapter 4 provides a performance measure which can used to compare the performance of different implementations.

In the Second part of the thesis, the framework that was presented in the first part of the thesis was used to optimize the sensor placement in a Shack-Hartmann (SH) sensor which is used in an adaptive optics setup. The implemented observers must estimate atmospheric turbulence using a modal approach which was simulated using the Object-Oriented Matlab Adaptive Optics (OOMAO) toolbox. The conditions for the simulation were chosen to be realistic for a medium sized telescope at a prime observation location at high altitude.

The first implementation using the LTI model with 9 sensors placed in a square pattern achieved a mean Variance Accounted For (VAF) score of 86.9%, and using a trace cost function and the Data-based Online Nonlinear Extremumseeker (DONE) algorithm to optimize these locations performed best and yielded a VAF score of 95.3%, which is an improvement of 8.5% points. The downside to this method is that under different environmental conditions like like wind angle, the performance of the system drops drastically. From this we can draw the following conclusions:

- Optimizing for a single set of conditions will not result in a robust placement.
- The DONE algorithm is best suited for sensor placement within this framework.

The implementation using the LPV model with the same measurement and simulation conditions as the LTI model is meant to improve the robustness of the observer to changing wind conditions. The performance of the observer was improved for a large part of the operating range, but choosing the wind angle as parameter led to problems around the wrapping point. Optimizing the sensor locations led to improvement of performance for almost all wind angles, with an average VAF improvement of around 10% points. From this we can draw the following conclusion:

- The placement of sensors is only robust for conditions considered in the model.

To try and better understand what sensor locations are good sensor locations, and to show how the optimization process can be simplified, the sensor locations were parametrized using two parameters in the polar domain. With the use of only two parameters it is now possible to plot the cost functions of the models. With these plots we can conclude that good sensor locations have three main properties.

- Sensors must not be placed on top of each other.
- Sensors at the edge of the aperture domain are preferred.
- Sensors close to the wind source direction are preferred.

Recommendations

The results presented in this thesis show that significant improvements in observer performance can be made using the techniques in the placement framework. There are, however, several opportunities left to address shortcomings and improve on the current work. The first shortcoming stems from a limitation in the OOMAO toolbox. When simulating the disturbance, it is not possible to change the angle of the wind, which makes generating identification and validation data difficult for the LPV system. A good opportunity would be to test the framework with data with a more dynamic parameter set.

The part of the thesis that could benefit from the largest amount of improvement is the LPV implementation. The major problem with the LPV model is caused by the use of the wind angle as the parameter. Using the wind angle causes problems with identification around the wrapping point. The wind angle parameter could be replaced by a directional vector, which could eliminate this problem. Another opportunity would be to expand the system with more parameters like wind speed, although preliminary investigation shows that the effect of changing wind speed is much less pronounced in the performance of the observers.

In the last part of the results we showed that using the sensor positions found using the LPV model with an LTI model performs well for all wind angles if we allow for identification of these LTI models before the simulation. Such an implementation could be considered unfair, but could be imitated by using a look-up table with LTI models, and the framework could be expanded by adding such a method and corresponding cost functions.

We looked at what defines good placement of sensors for our adaptive optics implementation. It could be worth looking into what defines a good placement in a more fundamental sense. Some work has already been done in this direction[29]. This publication uses controllability and observability Gramians to rate sensor and actuator positions. Such a method using Gramians could be added to the framework.

Finally, in this thesis we focused on the placement of a set number of sensors. This number was chosen experimentally. Adding a method to determine the best number of sensors considering cost and performance could be an interesting direction.

Appendix A

Matlab code

A-1 Data generation

```
function [alpha,mu,phase,angleSet]=modelgen(nSimSteps,nPolynomials,genAng
    ,samplingFreq,useLpv)
% generate identification data using OOMAO.
%
%
%
warning off
%close all




%originally atmosphere(photometry.V,20e-2,30,...
%'fractionnalR0',0.5,'altitude',5e3,...
%'windSpeed',2,'windDirection',0);
atm = atmosphere(photometry.V,20e-2,30,...
     'fractionnalR0',[0.2 0.2],'altitude',[5e3 5e2],...
     'windSpeed',[30 35],'windDirection',[genAng genAng]);

nL = 8;
nPx = 4;
nRes = nL*nPx;
D = 6;
d = D/nL; % lenslet pitch


tel = telescope(D,'resolution',nRes,...
     'fieldOfViewInArcMin',2.5,'samplingTime',1/samplingFreq);

tel = tel + atm;
```

```

%%
ngs = source('wavelength',photometry.J);


ngs=ngs.*tel;
alpha=zeros(nSimSteps,nPolynomials-1);

zern=zernike(2:nPolynomials,D,'resolution',nRes,'unitNorm',true);
%%
printf('Calculating wavefront for %d steps...',nSimSteps);
figure;
phase=[];
anglerange=pi;
angleSet=-anglerange:anglerange/2:anglerange;
%angleSet=[angleSet ];
iAngleStep=1;
for iSimStep=1:nSimSteps
    +ngs;
    +tel;
    phase(:,:,iSimStep)=ngs.meanRmPhase;
    zern2=zern.\(ngs.meanRmPhase);
    alpha(iSimStep,:)=zern2.c';
    printf(iSimStep , nSimSteps);
%      imagesc(ngs.meanRmPhase)
%      drawnow
    if mod(iSimStep,nSimSteps/(2*(length(angleSet))))==0&&iSimStep<useLpv
        *(nSimSteps/2)+1
        delete(atm);
        reset(ngs);
        atm = atmosphere(photometry.V,20e-2,30,...
            'fractionnalR0',[0.2 0.2],'altitude',[5e3 5e2],...
            'windSpeed',[30 35],'windDirection',[0 0],'randStream',
                RandStream('mt19937ar','Seed',abs(sin(now))));
        atm.layer(1).windDirection=angleSet(iAngleStep);
        atm.layer(2).windDirection=angleSet(iAngleStep);
        iAngleStep=iAngleStep+1;
        tel2 = tel + atm;

        %%
        ngs = source('wavelength',photometry.J);


        ngs=ngs.*tel;

    end
    if iSimStep==nSimSteps/2&&useLpv==1
        delete(atm);
        reset(ngs);
        atm = atmosphere(photometry.V,20e-2,30,...
            'fractionnalR0',[0.2 0.2],'altitude',[5e3 5e2],...
            'windSpeed',[30 35],'windDirection',[0 0],'randStream',
                RandStream('mt19937ar','Seed',abs(sin(now))));
        atm.layer(1).windDirection=genAng%(min(angleSet)+(max(angleSet)-
            min(angleSet)).*rand(1,1));
        atm.layer(2).windDirection=atm.layer(1).windDirection;

        tel2 = tel + atm;

        %%
        ngs = source('wavelength',photometry.J);


        ngs=ngs.*tel;

    end
    mu(iSimStep)=(atm.layer(1).windDirection)/(2*pi);
end
close(gcf);
figure,plot(mu),title('\mu');
% for k=1:s/2
%     +ngs;
%     +tel;
%     phase(:,:,k+s)=ngs.meanRmPhase;
%   zern2=zern.\(ngs.meanRmPhase);
%     alpha(k+s/2,:)=zern2.c';
%     printf(k , s);
%
% %          atm.layer(1).windDirection=pi;
% %          atm.layer(2).windDirection=pi;
% %
%
%     imagesc(ngs.meanRmPhase)
%     drawnow
% end
printf;

figure
plot(alpha)
title('alpha')
warning on
end
```

A-2 DONE

```
%% done algorithm for determining sensor locations


xn=x0;%[xmin(1:size(xmin,1)/2,I2);xmin(size(xmin,1)/2+1:end,I2)]; %set
    the startingpoint
xd=[x0 zeros(length(x0),doneSteps-1)];
yd=zeros(1,doneSteps);
mnoise=0.1;% measurement noise(not used)
r=1;% circle diameter
if useLpv==0
```

```
    %LTI settings
    sig=1;%frequency range
    lam=1e-1;%relaxation coeff
    explore1=0.05;%random addition to optimization start
    explore2=0.15;%random addition to the new point that is added to the surrogate function
    [~,pOffset]=estimator(xn(1:length(xn)/2),xn(length(xn)/2+1:end),eo,Ato,nPolynomials,samplingFreq,arOrder,U,useSvd,dSens);
    offsett = 2*pOffset;
else
    %LPV settings
    sig=1;%frequency range
    lam=1e-1;%relaxation coeff
    explore1=0.5;%random addition to optimization start
    explore2=0.1;%random addition to the new point that is added to the surrogate function
     if useOpt==1
        offsett =2*tracePlpvopt(xn,nPolynomials,arOrder,Ato0,Ato1,Br0,Br1,eo,mu,U,useSvd,dSens);
    else
        offsett = 2*tracePlpv(xn,nPolynomials,ModelOptimized,mu,U,useSvd);
    end
end

nFunctions=500;
C=zeros(nFunctions,1);
B=2*pi*rand(1,nFunctions)';
P12=1/sqrt(lam)*eye(nFunctions);

% random functions
omega=sqrt(2*sig)*randn(length(x0),nFunctions)';

xmin=[];
Y=[];
Ymeasurement=[];
Ysurrogate=[];
options = optimset('algorithm','interior-point','display','off','MaxITer',50,'GradObj','on','Hessian','lbfgs');
Zd=zeros(nFunctions,doneSteps);
printf('finding minimum with %d steps...', doneSteps);
Ymeasurement = zeros(1,doneSteps);
% filename = 'optgif.gif';
Z=@(x) sqrt(2/nFunctions)*cos(omega*x+B);
figure
tic
for iDoneStep=1:doneSteps


    % measure
    if useLpv==0
        [~, Y(iDoneStep)]=estimator(xn(1:length(xn)/2),xn(length(xn)/2+1:end),eo,Ato,nPolynomials,samplingFreq,arOrder,U,useSvd,dSens);

    else
       % [Y(iDoneStep)] =tracePlpv(xn,nPolynomials,ModelOptimized,mu,U,
           useSvd);
         if useOpt==1
        [Y(iDoneStep)]  =tracePlpvopt(xn,nPolynomials,arOrder,Ato0,Ato1,
            Br0,Br1,eo,mu,U,useSvd,dSens);
    else
         [Y(iDoneStep)]   = tracePlpv(xn,nPolynomials,ModelOptimized,mu,U,
             useSvd);
    end
    end
    Y(iDoneStep)=Y(iDoneStep)-offsett;

    Zd(:,iDoneStep)=Z(xn);

    %qr update
    A =  [ 1,                Zd(:,iDoneStep)'*P12;    ...
         zeros(nFunctions,1),P12          ];

    R = qr(A');
    R = R';
    gamma_12 = R(1,1);
    ggamma_12 = R(2:end,1);
    P12= R(2:end,2:end);
    C=C+ (ggamma_12/gamma_12)*(Y(iDoneStep)-Zd(:,iDoneStep)'*C);

    % update the surrogate function
    G=@(x) C'*Z(x);
    % update the derivative surrogate function
    Gd=@(x) -omega'*diag(sin(omega*x+B))*C;
    % find minimum of cost function
    xmin(:,iDoneStep) = fmincon({G,Gd},(xn+explore1*randn(length(xn),1)),
         [], [],[],[],-1*ones(1,2*length(XcordVector))',1*ones(1,2*length(
        XcordVector))',@unitdisk2,options);


    Ysurrogate(iDoneStep)=G(xn)+offsett;

    [~, I2]=min(Y(1:iDoneStep));
    % determine addition point
    xn=xmin(:,iDoneStep) +explore2*randn(length(xn),1);
    printf(iDoneStep , doneSteps);

    circle(r);

    plot(xmin(1:size(xmin,1)/2,end),xmin(size(xmin,1)/2+1:end,end),'o')
    hold off
    drawnow
end
a3=toc;
printf;
%% plot things
```

```matlab
if useLpv==1
    %[lpvvaf,lpvvafw]=reconstructLPV(ModelOptimized,ModelIdentified,nPolynomials,xmin(:,I2),alpha(nSimSteps/2+1:end,:),nPolynomials-1,mu(:,nSimSteps/2+1:end),0) ;
   % [lpvvaf,lpvvafw]=reconstructLPV(ModelOptimized,ModelIdentified,nPolynomials,xmin(:,I2),alpha(1:nSimSteps/2,:),nPolynomials-1,mu(:,1:nSimSteps/2),0);
end
[M1 I1]=max(Y(1:end-1));
[M2 I2]=min(Y(1:end-1));

figDone1=figure;
     hold on
     plot(x0(1:length(x0)/2),x0(length(x0)/2+1:end),'*')%start
     plot(xmin(1:size(xmin,1)/2,end),xmin(size(xmin,1)/2+1:end,end),'o')%eind

     plot(xmin(1:size(xmin,1)/2,I1),xmin(size(xmin,1)/2+1:end,I1),'sg')%worst
     plot(xmin(1:size(xmin,1)/2,I2),xmin(size(xmin,1)/2+1:end,I2),'oc')%best
     plot(mean(xmin(1:size(xmin,1)/2,end)),mean(xmin(size(xmin,1)/2+1:end,end)),'^b')%eindgem
     plot(mean(xmin(1:size(xmin,1)/2,I2)),mean(xmin(size(xmin,1)/2+1:end,I2)),'vr')%bestgem
%      plot(xmin(1:9,:),xmin(10:18,:))

     circle(1);
     pbaspect([1 1 1])
     title('starting positions, best, worst, and final DONE solution')
     legend('start','final','worst','best')

%%
figDone2= figure;

     hold on
     semilogy(Y+offsett)
     semilogy(Ysurrogate)
     legend('measured','surrogate')
     axis([ 0 doneSteps 2 (offsett/2)+1])
     title({['actual and surrogate value for trace(P) for ' num2str(doneSteps) ' iterations'],['took ' num2str(a3) ' seconds']})
     %line([0 10000],[min(Y)+offsett min(Y)+offsett],'Color','black','LineStyle','--')

     % saveas(gcf,'done.eps','epsc')
%resprint;
```

A-3 Heuristics

```matlab
%% heuristic grid search and random walk algorithms for determining sensor position\
% heuristic grid search
gridSearchSpacing=-1:0.2:1; %grid spacing
[XcoordinatesSensor,YcoordinatesSensor]=meshgrid(sensorSpacing,
    sensorSpacing);
[XcoordinatesGridSearch,YcoordinatesGridSearch]=meshgrid(
    gridSearchSpacing,gridSearchSpacing);
XcoordinatesSensorVec=vec(XcoordinatesSensor);
YcoordinatesSensorVec=vec(YcoordinatesSensor);
XcoordinatesGridSearchVec=vec(XcoordinatesGridSearch);
YcoordinatesGridSearchVec=vec(YcoordinatesGridSearch);

[theta,r]=cart2pol(XcoordinatesGridSearchVec(1:end),
    YcoordinatesGridSearchVec(1:end));
circleMask = r<=1;
[XcoordinatesGridSearchVec,YcoordinatesGridSearchVec]=pol2cart(theta(
    circleMask),r(circleMask));

x0=[XcoordinatesSensorVec;YcoordinatesSensorVec];


xn=x0;%
xbestGS=xn;% Set starting postion
figure
kmax=2; %amount of runs
Yhb=ones(1,(length(sensorSpacing)^2)*kmax);
if useLpv==0
    [~,   tracePStart]=estimator(xn(1:length(xn)/2),xn(length(xn)/2+1:end
        ),eo,Ato,nPolynomials,samplingFreq,arOrder,U,useSvd,dSens);
else
    %[tracePStart] =tracePlpv(xn,nPolynomials,ModelOptimized,mu,U,useSvd)
        ;
    if useOpt==1
        tracePStart =tracePlpvopt(xn,nPolynomials,arOrder,Ato0,Ato1,Br0,
            Br1,eo,mu,U,useSvd,dSens);
    else
        tracePStart = tracePlpv(xn,nPolynomials,ModelOptimized,mu,U,
            useSvd);
    end
end
Yhb=Yhb.*tracePStart;
tic
for k=1:kmax;
    printf('finding minimum with %d steps...', length(sensorSpacing)^2);
    for iSensor=1:length(XcoordinatesSensorVec)
        %printf('finding minimum with %d steps...', length(Xcv2));
        for iGrid=1:length(XcoordinatesGridSearchVec)
            xn(iSensor)=XcoordinatesGridSearchVec(iGrid);
            xn(iSensor+(length(xn)/2))=YcoordinatesGridSearchVec(iGrid);
            if useLpv==0
                [~,   Yh(iGrid)]=estimator(xn(1:length(xn)/2),xn(length(
                    xn)/2+1:end),eo,Ato,nPolynomials,samplingFreq,arOrder,
                    U,useSvd,dSens);
            else
```

```

                if useOpt==1
                    Yh(iGrid) =tracePlpvopt(xn,nPolynomials,arOrder,Ato0,Ato1,Br0,Br1,eo,mu,U,useSvd,dSens);
                else
                    Yh(iGrid) = tracePlpv(xn,nPolynomials,ModelOptimized,mu,U,useSvd);
                end
            end

            if iSensor>1
                if Yh(iGrid) <= Yhb(iSensor-1 +(k-1)*(length(sensorSpacing)^2)) %|| Yhb(i +(k-1)*(length(spc3)^2))==0
                    Yhb(iSensor+(k-1)*(length(sensorSpacing)^2):end)=Yh(iGrid);
                    xbestGS(iSensor)=XcoordinatesGridSearchVec(iGrid);
                    xbestGS(iSensor+(length(xn)/2))=YcoordinatesGridSearchVec(iGrid);
                end
            else
                if Yh(iGrid) <= Yhb(iSensor +(k-1)*(length(sensorSpacing)^2)) %|| Yhb(i +(k-1)*(length(spc3)^2))==0
                    Yhb(iSensor+(k-1)*(length(sensorSpacing)^2):end)=Yh(iGrid);
                    xbestGS(iSensor)=XcoordinatesGridSearchVec(iGrid);
                    xbestGS(iSensor+(length(xn)/2))=YcoordinatesGridSearchVec(iGrid);
                end
            end
            hold off
            plot(xn(1:size(xbestGS,1)/2,end),xn(size(xbestGS,1)/2+1:end,end),'o')
            circle(1);
            drawnow
            printf(iGrid , length(XcoordinatesGridSearchVec));
        end
        xn(iSensor)=xbestGS(iSensor);
        xn(iSensor+(length(xn)/2))=xbestGS(iSensor+(length(xn)/2));
        % printf;

        disp(' ');
        printf('!clear')

        printf(iSensor , length(sensorSpacing)^2);

    end
    XcoordinatesGridSearchVec=flip(XcoordinatesGridSearchVec);
    YcoordinatesGridSearchVec=flip(YcoordinatesGridSearchVec);
end
%%
a=toc;
printf;
```

```
printf;
close gcf
figure;
hold on
plot(xbestGS(1:size(xbestGS,1)/2,end),xbestGS(size(xbestGS,1)/2+1:end,end
    ),'o')
plot(mean(xbestGS(1:size(xbestGS,1)/2,end)),mean(xbestGS(size(xbestGS,1)
    /2+1:end,end)),'^b')%eindgem
title('Optimal locations for grid search')
circle(1);
figHeur1=figure;
plot(0:1:18,[offsett/2 Yhb])
axis([ 0 18 2 (offsett/2)+1])
title({['Trace(P) after finding optimal new location for each sensor '],[
    'took ' num2str(a) ' seconds']})
%% random walk
rnoise=0.01; % noise addition for the optimizaton step
nRandomWalk=500;
xn=[vec(XcoordinatesSensor(1:end)); vec(YcoordinatesSensor(1:end))]; %
    starting positions

figure
if useLpv==0
    [~, Yh2]=estimator(xn(1:length(xn)/2),xn(length(xn)/2+1:end),eo,Ato,
        nPolynomials,samplingFreq,arOrder,U,useSvd,dSens);
else

    if useOpt==1
            Yh2 =tracePlpvopt(xn,nPolynomials,arOrder,Ato0,Ato1,Br0,Br1,
                eo,mu,U,useSvd,dSens);
        else
            Yh2 = tracePlpv(xn,nPolynomials,ModelOptimized,mu,U,useSvd);
        end
end
tic
disp(' ');
disp(' ');
printf('walking for %d steps...', nRandomWalk);
Yh3=zeros(1,nRandomWalk);
Yh2=ones(1,nRandomWalk).*Yh2;
for iWalkStep=1:nRandomWalk
    xn2=xn+rnoise*randn(1,length(xn))';
    if useLpv==0
        [~,  Yh3(iWalkStep)]=estimator(xn2(1:length(xn)/2),xn2(length(xn)
            /2+1:end),eo,Ato,nPolynomials,samplingFreq,arOrder,U,useSvd,
            dSens);
    else

        if useOpt==1
            Yh3(iWalkStep) =tracePlpvopt(xn2,nPolynomials,arOrder,Ato0,
                Ato1,Br0,Br1,eo,mu,U,useSvd,dSens);
        else
```

```
                Yh3(iWalkStep) = tracePlpv(xn2,nPolynomials,ModelOptimized,mu
                    ,U,useSvd);
            end
        end
        [theta,r]=cart2pol(xn2(1:length(xn)/2),xn2(length(xn)/2+1:end));
        for iGrid=1:length(r)
            if(r(iGrid)>1)
                r(iGrid)=1;
            end
        end
        [xn2(1:length(xn)/2),xn2(length(xn)/2+1:end)]=pol2cart(theta,r);
        if iWalkStep>1
            if Yh3(iWalkStep)<Yh2(iWalkStep-1) %&& max(r)<1
                xn=xn2;
                Yh2(iWalkStep:end)=Yh3(iWalkStep);
            end
        else
            if Yh3(iWalkStep)<Yh2(iWalkStep) %&& max(r)<1
                xn=xn2;
                Yh2(iWalkStep:end)=Yh3(iWalkStep);
            end
        end
        hold off
        plot(xn(1:size(xbestGS,1)/2,end),xn(size(xbestGS,1)/2+1:end,end),'o')
        circle(1);
        drawnow
        printf(iWalkStep , nRandomWalk);

end

a2=toc;
%%
xbestRW=xn;
printf;
figure;
hold on
plot(xn(1:size(xbestRW,1)/2,end),xn(size(xbestRW,1)/2+1:end,end),'o')
plot(mean(xbestRW(1:size(xbestGS,1)/2,end)),mean(xbestRW(size(xbestGS,1)
    /2+1:end,end)),'^b')%eindgem
title('optimal locations found by random walk')
circle(1);
figHeur2=figure;
plot(Yh2)
title({['Trace(P) of random walk for ' num2str(nRandomWalk) ' iterations
    with ' num2str(rnoise) ' noise variance'],[' took ' num2str(a2) '
    seconds']})
axis([ 0 nRandomWalk 2 (offsett/2)+1])
```

Bibliography

[1] F. Lin, M. Fardad, and M. R. Jovanović, "Design of optimal sparse feedback gains via the alternating direction method of multipliers," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2426–2431, 2013.

[2] F. Lin, M. Fardad, and M. R. Jovanović, "Synthesis of H2 optimal static structured controllers: Primal and dual formulations," in *2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 340–346, 9 2009.

[3] N. K. Dhingra, M. R. Jovanović, and Z. Q. Luo, "An ADMM algorithm for optimal sensor and actuator selection," in *53rd IEEE Conference on Decision and Control*, pp. 4039–4044, 12 2014.

[4] S. Joshi and S. Boyd, "Sensor selection via convex optimization," *IEEE Transactions on Signal Processing*, vol. 57, no. 2, pp. 451–462, 2009.

[5] M. van de Wal and B. de Jager, "A review of methods for input/output selection," *Automatica*, vol. 37, no. 4, pp. 487–510, 2001.

[6] M. V. D. Wal, P. Philips, and B. D. Jager, "Actuator and sensor selection for an active vehicle suspension aimed at robust performance," *International Journal of Control*, vol. 70, no. 5, pp. 703–720, 1998.

[7] N. Darivandi, K. Morris, and A. Khajepour, "An algorithm for LQ optimal actuator location," *Smart Materials and Structures*, vol. 22, no. 3, p. 35001, 2013.

[8] K. Morris, M. A. Demetriou, and S. D. Yang, "Using H2-Control Performance Metrics for the Optimal Actuator Location of Distributed Parameter Systems," *IEEE Transactions on Automatic Control*, vol. 60, pp. 450–462, 2 2015.

[9] D. Kasinathan and K. Morris, "H-infinity-Optimal Actuator Location," *IEEE Transactions on Automatic Control*, vol. 58, no. 10, pp. 2522–2535, 2013.

[10] C. Antoniades and P. D. Christofides, “Integrated optimal actuator/sensor placement and robust control of uncertain transport-reaction processes,” *Computers and Chemical Engineering*, vol. 26, no. 2, pp. 187–203, 2002.

[11] T. Summers and I. Shames, “Convex relaxations and Gramian rank constraints for sensor and actuator selection in networks,” in *2016 IEEE International Symposium on Intelligent Control (ISIC)*, pp. 1–6, 9 2016.

[12] M. Verhaegen and V. Verdult, *Filtering and system identification: A least squares approach.* 2007.

[13] D. Ucinski, “Measurement Optimization for Parameter Estimation in Distributed Systems,” *Proceedings of 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Berlin, Germany, August 1997*, vol. 5, 1999.

[14] C. Colburn, *Estimation techniques for large-scale turbulent fluid systems.* PhD thesis, UNIVERSITY OF CALIFORNIA, SAN DIEGO, 2011.

[15] V. Spruyt, “Website - How to draw a covariance error ellipse,” 2014.

[16] J.-W. van Wingerden and M. Verhaegen, “Subspace identification of Bilinear and LPV systems for open- and closed-loop data,” *Automatica*, vol. 45, pp. 372–381, 2 2009.

[17] S. H. Cha, M. Rotkowitz, and B. D. Anderson, “Gain Scheduling using Time-varying Kalman Filter for a class of LPV Systems,” *IFAC Proceedings Volumes*, vol. 41, pp. 4934–4939, 1 2008.

[18] J. Shamma and M. Athans, “Gain scheduling: potential hazards and possible remedies,” *IEEE Control Systems*, vol. 12, pp. 101–107, 6 1992.

[19] H. R. G. W. Verstraete, S. Wahls, J. Kalkman, and M. Verhaegen, “Model-based sensorless wavefront aberration correction in optical coherence tomography,” *Opt. Lett.*, vol. 40, pp. 5722–5725, 12 2015.

[20] L. Bliek, H. R. G. W. Verstraete, M. Verhaegen, and S. Wahls, “Online Optimization With Costly and Noisy Measurements Using Random Fourier Expansions,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, pp. 167–182, 1 2018.

[21] S. de Groot, “Robust optimal sensor and actuator placement for adaptive optics - Literature Survey,” tech. rep., TU Delft, 2018.

[22] M. Verhaegen, P. Pozzi, O. Soloviev, G. Vdovin, and D. Wilding, “Control for High Resolution Imaging,” 2017.

[23] I. B. Putnam, “Modeling a Temporally Evolving Atmosphere with Zernike Polynomials,” *AMOS technical conference*, 2012.

[24] R. Conan and C. Correia, “Object-oriented Matlab adaptive optics toolbox,” vol. 9148, p. 91486C, International Society for Optics and Photonics, 8 2014.

[25] R. J. Noll, “Zernike polynomials and atmospheric turbulence,” *Journal of the optical society of America.*, vol. 66, no. 3, pp. 2007–2011, 1976.

[26] R. G. Lane and M. Tallon, “Wave-front reconstruction using a Shack-Hartmann sensor,” *Applied Optics*, vol. 31, no. 32, pp. 6902–6908, 1992.

[27] C. Neyman, “Atmospheric Parameters for Mauna Kea,” tech. rep., 2004.

[28] “The {MOSEK} optimization software.”

[29] T. H. Summers and J. Lygeros, “Optimal Sensor and Actuator Placement in Complex Dynamical Networks,” *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 3784–3789, 2014.

Glossary

List of Acronyms

AO	Adaptive Optics
DONE	Data-based Online Nonlinear Extremumseeker
RFE	Random Fourier Expansions
GS	Grid Search
RW	Random Walk
DARE	Discrete Algebraic Ricatti Equation
LTI	Linear Time Invariant
LPV	Linear Parameter Varying
VAF	Variance Accounted For
SVD	Singular Value Decomposition
PCA	Principal Component Analysis
SH	Shack-Hartmann
VAR	Vector Autoregressive
OOMAO	Object-Oriented Matlab Adaptive Optics