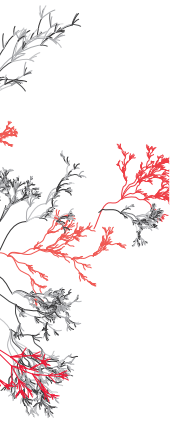
27-06-2017



MASTER THESIS

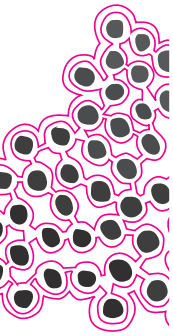
COMPRESSIVE SENSING IN DYNAMIC SCENES

Nick Doornekamp

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)

Programme: Applied Mathematics
Chair: Hybrid Systems

Exam committee:
Prof.dr. A.A. Stoorvogel (UT)
Dr. M. Bocquel (UT/Thales)
Dr. P.K. Mandal (UT)
Dr. D.J. Bekers (TNO)
Dr. J.C.W. van Ommeren (UT)
Dr. M. Podt (Thales)



UNIVERSITY OF TWENTE.

Abstract

In recent years, the Compressive Sensing (CS) framework has received considerable attention. Most of its applications are found in static problems, such as the reconstruction of images from seemingly incomplete data. In this report we ask ourselves whether the CS framework can also be of use in a dynamic scene. In particular, we consider the task of tracking multiple targets. For this task, the class of Bayesian filters is optimal, but in some cases computationally expensive. We consider two alternative approaches, one that uses only CS, the other combines CS with a Particle Filter. These are hoped to improve on Bayesian filters in a situation where computational resources are constrained. For both approaches we propose alterations to the algorithms from the literature. We provide numerical results of the comparison between the proposed algorithms and the algorithms from literature they are based on. Furthermore we provide directions towards a comparison of the proposed algorithms and algorithms from the class of Bayesian filters.

Contents

Abbreviations and nomenclature

Abbreviations

CS	Compressive Sensing
BP	Basis Pursuit
BPDN	Basis Pursuit Denoising
PF	Particle Filter
SNR	Signal-to-Noise ratio
IC	Information Criterion
RC	Rayleigh Cell
OSPA	Optimal Subpattern Assignment
YALL1	Your ALgorithms for L1 optimization
FAR	False Alarm Rate
HPFCS	Hybrid combination of PF and CS
SIR	Sequential Importance Resampling
MC	Monte Carlo
KLD	Kullback-Leibler Distance

Nomenclature

$f \in \mathbb{C}^n$	Original signal
$\Phi \in \mathbb{C}^{m \times n}$	Compression matrix
$f_c \in \mathbb{C}^m$	Compressed signal
$\Psi \in \mathbb{C}^{n \times n}$	Basis
$x \in \mathbb{C}^n$	State vector recovered in CS framework
$\hat{x} \in \mathbb{C}^n$	Estimated state in the context of CS
$A \in \mathbb{C}^{m \times n}$	Sensing matrix
$\rho \in \mathbb{R}$	Weighting parameter in BPDN
$y \in \mathbb{C}^m$	Noisy compressed signal (also measurement)
$N_{\text{iter}} \in \mathbb{N}$	Number of YALL1 iterations
$A^{(r)} \in \mathbb{R}$	Amplitude of component r
$\hat{A}^{(r)} \in \mathbb{C}$	Estimated amplitude and phase-shift of component r
$F^{(r)} \in \mathbb{R}$	Frequency of component r
$R \in \mathbb{N}$	Number of components
$\epsilon \in \mathbb{R}$	Stopping tolerance parameter in YALL1
N	Support of x
$x_{\hat{N}^C}$	Elements of x outside of the estimated support $\hat{N}$
$\omega_k \in \mathbb{C}$	Realization of the process noise at timestep k
σ_ω	Standard deviaton of the process noise
$\nu_k \in \mathbb{C}$	Realization of the measurement noise at timestep k
σ_ν	Standard deviaton of the measurement noise
s	State vector in the context of PF
$N_p \in \mathbb{N}$	Number of particles

1. Introduction

Compressive Sensing (CS) is a framework for the acquisition and processing of signals that are (approximately) sparse when expressed in a suitable basis. That is, when the signal is expressed in this basis only a few of the coefficients of its expansion are not (approximately) zero. This assumption holds for many signals encountered in practice and many classical compression techniques also rely on this. The classical approach to signal acquisition is to sample the signal at the Nyquist rate, which is guaranteed to be sufficient for a complete representation of the signal. However, in many situations a complete representation is not necessary. Instead, a compressed version of the signal is used, which is much smaller than the Nyquist-sampled data in many situations.

For example, when a picture is taken with a digital camera, dozens of megabytes of data are collected. However, it turns out that when this image is transformed to e.g. the wavelet domain (as is done by the well-known JPEG[1] compression method) only a relatively small number of wavelet coefficients are large; the others are approximately zero. In other words: much of the information can be captured using a small number of wavelets. Mathematically speaking, the image is approximately sparse in the wavelet domain. As a result, only the large coefficients have to be saved, while the quality of the image reconstructed from these coefficients is still close to the original. While this is useful, it also raises the question whether it is truly necessary to sample at a high rate if only a small amount of this data ends up being used in the final representation. With the CS framework the answer to this question is 'no': with CS, signals that are sparse in some domain can be recovered from a number of measurements that is small compared to what the Nyquist rate suggests. Instead of sampling the original signal directly, only a compressed version of it is acquired.

1.1 The CS framework

In the CS framework, the compressed signal is a linear function of the original signal $f \in \mathbb{C}^n$. The compressed signal, $f_c \in \mathbb{C}^m$, can therefore be expressed as the product of f and a matrix $\Phi \in \mathbb{C}^{m \times n}$: $f_c = \Phi f$. We will refer to Φ as the compression matrix. Given the compressed signal the CS framework recovers the state $x \in \mathbb{C}^n$, which is related to the original signal through basis $\Psi \in \mathbb{C}^{n \times n}$: $f = \Psi x$. E.g. if Ψ is the Fourier basis, then x contains the Fourier coefficients of f. In that case, if f is the sum of a few sine functions, only a few elements of x need to be nonzero: x is sparse.

With the CS framework it is possible to obtain f by first solving x from

$$\Phi\Psi x = f_c, \tag{1.1}$$

and then computing $f = \Psi x$. Whether this is actually done depends on the situation: in the context of this report it is not necessary to reconstruct f. Instead, the desired information is extracted from x directly. In the example where Ψ is the Fourier basis and f the sum of a few sines, this means the following: Instead of sampling f at or above the Nyquist rate, in the CS framework one would acquire only f_c. By solving $\Phi\Psi x = f_c$ for x, the x that underlies the original signal f is

[1] Joint Photographic Experts Group

obtained. With this x, it would be easy to obtain f. If instead of obtaining f the goal is for example to determine which frequencies the sines in f have, this is not necessary: this information can be extracted from x directly. In some areas, when the CS framework is applied without reconstructing f, this technique is referred to as sparse sensing instead of CS.

In the following we will denote the product of the compression matrix Φ and basis Ψ as $A \in \mathbb{C}^{m \times n}$ and refer to it as the sensing matrix. The CS framework is particularly interesting when m is much smaller than n. That is, the dimension of the acquired measurement is much smaller than the dimension of the original signal. In that case, denoted $m \ll n$, the sensing matrix will have many more columns than rows. Therefore system (1.1) is underdetermined, i.e. it has infinitely many solutions. Therefore, the key to CS lies in finding the right x from the under-determined system (1.1). Since the solution is assumed to be sparse, one way to proceed would be to select the sparsest solution that satisfies (1.1). This problem can be formalized as

$$\hat{x} = \underset{x \in \mathbb{C}^n}{\operatorname{argmin}} \{ \|x\|_0 \text{ s.t. } Ax = f_c \}, \tag{1.2}$$

where $\|x\|_0$ denotes the number of nonzero elements in x. In words, this problem is the minimization of the number of nonzero elements in x that can still describe f_c perfectly. This problem is combinatorial, since each combination of nonzero positions in the solution vector has to be considered, which makes finding its solution intractable. However, it has been proven that the ℓ^0-norm[2] in (1.2) can be replaced by the ℓ^1-norm in the sense that (1.2) and

$$\hat{x} = \underset{x \in \mathbb{C}^n}{\operatorname{argmin}} \{ \|x\|_1 \text{ s.t. } Ax = f_c \} \tag{1.3}$$

have the same solution under certain conditions. For details on these conditions, the reader is referred to the work of Candès *et al.* [10] or the work of Donoho [13]. In other words: the solution to (1.3) can be used to recover sparse vectors from (1.1). This is one of the fundamental results of CS. After all, since (1.3) is a convex problem, it is generally much easier to solve than (1.2). While solving such a problem for large inputs can still be expensive, much is known about how to do so and many efficient solvers are available. The problem in (1.3) is often referred to as Basis Pursuit (BP).

By demanding that $Ax = f_c$ it assumes noiseless measurements. If this assumption does not hold (i.e. $y = \Phi(f + \nu)$, where ν denotes the measurement noise) Basis Pursuit Denoising (BPDN) is more appropriate:

$$\hat{x} = \underset{x \in \mathbb{C}^n}{\operatorname{argmin}} \{ \|x\|_1 + \frac{1}{2\rho} \|Ax - y\|_2^2 \}. \tag{1.4}$$

This problem can be interpreted as a trade-off between sparsity (first term) and signal fidelity (second term), where the relative weights of these two objectives is determined by ρ.

Without any prior knowledge or assumptions about the signal of interest, classical signal processing theory like the Shannon-Nyquist theorem only guarantees perfect recovery when the number of measurements is more than twice the maximum frequency. Naturally, when prior information is available, as is the case with CS, this bound is pessimistic. In particular, it is shown that the number of compressive measurements m that is required to recover x perfectly in BP (1.3) is proportional to $S \log(n)$, where S denotes the number of nonzero entries of x. For the case of BPDN a bound on the reconstruction error (i.e. $\|x - \hat{x}\|$) can be provided. Again, for details of this proof and other theoretical results we refer to [10] and [13].

[2]Note that even though the notation suggests otherwise, $\|\cdot\|_0$ is, mathematically speaking, not a norm.

Applications for CS can be found in many areas. For example, Friedlander *et al.* [17] apply CS to reconstruct the original audio from compressive measurements. CS can also be applied to (radar-) estimation problems, such the work by Bekers *et al.* [4], who apply CS to determine the Direction Of Arrival of a single target. In this report we will consider only estimation problems: estimating certain parameter that underlie the measured signal.

However, most of the research into CS focuses on application in the area of imaging. A prominent example is magnetic resonance imaging (MRI), where the number of measurements required for such a scan, and therefore the amount of time a patient has to spend lying as still as possible, can be reduced by CS. An introduction to CS applied to MRI is provided by Lustig *et al.* [14]. Another interesting application of CS is the single-pixel camera project of Rice University. This project shows that useful (recognizable) images can be acquired with very limited hardware and a number of papers have been published on this topic, such as the work of Duarte *et al.* [16].

1.2 Dynamic CS

The CS theory and applications that have been discussed so far all consider static scenes. Measurements are collected once to estimate x or reconstruct f once. However, in many situations of interest the state x might change over time: a dynamic scene. The simplest way to deal with such a situation is to take a snapshot at certain points in time and treat these snapshots as if they were static scenes. But it is clear that there is something to gain if information could be compounded over time. Therefore, we ask ourselves:

How can we make use of CS in dynamic scenes?

One of the tasks that might come with a dynamic scene is the tracking of targets, which is the task that is considered in this report.

An existing and versatile solution for the problem of tracking targets is offered by Bayesian filters, such as the Kalman filter and the Particle filter (PF). PFs are known to be optimal[3] from a Bayesian perspective as the number of particles tends to infinity, as is also described later in this report. However, one usually needs a large numbers of particles to make the numerical approximation of a PF satisfactory, so that it requires a lot of computational resources.

We will explore two approaches to make use of CS in a dynamic scene. The first is a 'CS-only' approach, which aims to use information from the previous timestep to speed up or improve the accuracy of the CS algorithm at the current timestep. We will refer to such 'CS-only' solutions as Dynamic CS. The second approach is to combine CS with a Bayesian filtering algorithm, or more specifically: a PF. The choice for the PF is motivated by its asymptotic optimality for a very general class of situations. The following two sections introduce Dynamic CS and the combination of CS with Bayesian filters, after which an overview of the rest of the report is presented.

The intuition behind Dynamic CS is that the outcome of the algorithm at the previous timestep can help the algorithm at the current timestep. This could be interpreted as providing the CS algorithm with prior information, where the prior information was gathered during the previous timesteps. In this section we describe a number of ways to incorporate prior information into the CS framework. Once we can make use of prior information the introduction of Dynamic CS is a

[3]By 'optimal' we here mean that the estimated posterior distribution is as close to the truth as possible (i.e. no estimation method could provide an estimated posterior that is closer) not that the PF is the best solution in any situation. For example, as we will discuss later as well, in the case of a linear measurement model, linear dynamic model and Gaussian noise, a Kalman filter should be preferred over a PF.

matter of introducing a time-index.

One way to incorporate prior information into the CS framework is described in the work of Vaswani and lu [40]. They assume prior information on the support is available, where the support is defined as the positions where x is nonzero or above a certain threshold in the cases of sparse and approximately sparse states respectively. The support is denoted $N = \{i|x_i > \alpha\}$, where $\alpha = 0$ in the case of a sparse state. In other words, they assume that some of the possible target locations are in fact known target locations.
In the following, we always regard a situation with noisy measurements[4]. The prior information about the support of, in the form of an estimated support $\hat{N}$, is introduced into BPDN by slightly changing the problem:

$$\hat{x} = \underset{x \in \mathbb{C}^n}{\operatorname{argmin}} \{\|x_{\hat{N}^C}\|_1 + \frac{1}{2\rho}\|Ax - y\|_2^2\} \tag{1.5}$$

Here $x_{\hat{N}^C}$ is used to denote the elements of x outside the estimated support $\hat{N}$. Considering the ℓ^1-norm is used as a substitute for the ℓ^0-norm, this can be interpreted as follows: the number of nonzero elements *outside of areas where nonzero elements were found earlier* is penalized, instead of the number of nonzero elements in general. Vaswani and Lu [40] refer to the version of this problem without noise as Modified-CS or simply Mod-CS. Therefore (1.5) is referred to as Mod-BPDN. As noted by Friedlander *et al.* [17], the problem of sparse signal recovery with knowledge about the support was introduced in at least three papers: von Borries *et al.* [8], Vaswani and Lu [40] and Khajehnejad *et al.* [22]. All three make use of a weighted ℓ^1-norm to incorporate the support knowledge into the optimization problem. This norm is defined as:

$$\|x\|_{1,w} = \sum_i w_i |x_i|, \tag{1.6}$$

where w_i is the weight of entry i of the support. Vaswani and Lu [40] and von Borries *et al.* [8] set the weights of the estimated support entries to zero while the rest of the weights are one. Khajehnejad *et al.* [22] and Friedlander *et al.* [17] propose a slightly more general approach, where the weights of the estimated support entries are allowed to be nonzero. Both do require that there are at most two different groups of weights, so all weights are either $a \in [0, \infty)$ or $b \in [0, \infty)$.

CS algorithms that use prior information can be adapted to be dynamic by using the outcome of the previous timestep as the prior information of the current one. For example, Lu and Vaswani [26] describe 'Dynamic Mod-BPDN', where the Mod-BPDN problem is solved with the support estimated in the previous time-step.

$$\hat{x}_t = \min_{x_t \in \mathbb{C}^n} \{\|(x_t)_{\hat{N}_{t-1}^C}\|_1 + \frac{1}{2\rho}\|Ax_t - y_t\|_2^2\} \quad \text{where} \quad \hat{N}_{t-1} = \{i|\hat{x}_{t-1} > \alpha\}. \tag{1.7}$$

We will refer to (1.7) as Dynamic Mod-BPDN and CS algorithms that use information from the previous timestep(s) in general as Dynamic CS.

1.3 CS combined with Bayesian filters

Instead of trying to transfer information between pulses using only CS as in the previous section, another possibility is to deal with the dynamics of the scene with a Bayesian filter and let CS assist in some other way. In this section, a number of methods from the literature that use this approach

[4]To obtain an expression for the situation without noise, the term $\frac{1}{2\rho}\|Ax - y\|_2^2$ is replaced by $Ax = y$ as a constraint.

are discussed.

Ohlsson *et al.* [32] consider a situation where neither the measurement equation nor the dynamic model is required to be linear. Since the CS framework described earlier this chapter requires the measurement equation to be linear, Nonlinear Compressive Sensing (NLCS) - see e.g. [5] - is used in this situation. Nonlinear CS is combined with a particle filter, resulting in what they call the Nonlinear Compressive Particle Filter (NCPF). They show that the combination is necessary in their testcase, since (nonlinear) CS alone does not take the temporal relationship between measurements into account and their implementation of PF alone performs poorly due to the high state dimension. The combination they propose is a particle filter with a fixed cardinality that can be updated through Nonlinear CS. By default the particle filter with fixed cardinality is used at each timestep. If the likelihood of the signal estimated by this particle filter falls below a threshold, indicating that elements outside the currently assumed support are nonzero (i.e. there is a mismatch between the support used by the particle filter), the CS algorithm is used to detect which elements should be added to the support. Elements are removed from the current support when they fall below a threshold for a number of time steps. The idea of a PF with fixed cardinality that is updated through CS will be explored in section 4 and was implemented and tested during this project.

Ning *et al.* [30] combine a PF with CS for a Direction of Arrival problem where the targets are assumed to be on a grid. They use a CS algorithm to determine the initial locations of the targets and use this to create the (Gaussian) initial distribution for a PF. The idea of using CS to initialize a PF is discussed in section B.

Particle Filtered Modified Compressive Sensing (PaFiMoCS) by Das *et al.* [12] aims to deal with situations where the support does not necessarily change slowly over time, but does evolve according to a dynamic model. It does so by using a particle filter based on a dynamic model to provide the Mod-CS algorithm with a number of 'close enough' support-guesses (particles), sampled from the dynamic model. For each of these guesses the Mod-CS problem is solved to obtain an estimate of the signal value. They show that the PaFiMoCS performs better than a PF alone, single-snapshot CS at every timestep and Mod-CS alone. However, they do not consider the computational resources, which can be expected to be much higher for PaFiMoCS than for any of the other methods considered: Clearly solving the Mod-CS optimization problem for each of the PaFiMoCS particles will take much more computational resources than solving only the Mod-CS problem once or running a basic PF with the same number of particles.

1.4 Research goal and contributions

The goal of this research is to explore ways to make use of CS in dynamic scenes. Clearly there are many possible tasks in dynamic scenes where CS could be of use, but in this report we consider the task of tracking multiple targets. We discuss two approaches that use CS to perform this task: Dynamic CS and a combination of CS and a PF. For both approaches we propose adaptions of the algorithms described in the literature to make them more suitable for the situation that we consider.

In chapter 2 we start our treatise by describing the situation to which the approaches are applied. The applicability of the methods discussed is certainly not restricted to this situation, but the introduction of a concrete situation makes the discussion of the other methods more comprehensible. In this chapter we also describe how the performance of the methods will be measured. After that Dynamic CS and the combination of CS and PF are discussed in chapters 3 and 4, respectively. In particular, we discuss a way to include a dynamic model on the state in dynamic CS (in section 3.2.4) and a different trigger criterion for running CS in the combination of CS and PF (introduced in section 4.2.2). These adaptations are compared to the original algorithms in numerical simulations,

including their use of computational resources.

One way to proceed is to compare these methods to a multi-target PF, to see if these methods result in a gain, for example in terms of computational resources. However, as they are presented in this report, the assumptions and conditions of Dynamic CS and the combination of CS and PF differ too much from that of a multi-target PF to draw a sound conclusion from such a comparison. Therefore we provide directions towards a proper comparison of all three methods (section 5.4).

2. Simulation description

To make the methods comprehensible we first describe the situation to which they are applied. In the situation that we consider, at each timestep a signal pulse is generated. This pulse is the sum of an unknown number of frequency components. In section 2.1 this signal, denoted y_k at time k, is described in more detail. The objective of each of the methods is to determine an estimate of the state (which we will denote $\hat{s}_k$) of the system: the number of components, their amplitudes, and their frequencies. Finally these estimates are evaluated by a number of performance measures, which are described in section 2.2. These steps are illustrated by figure 2.1.

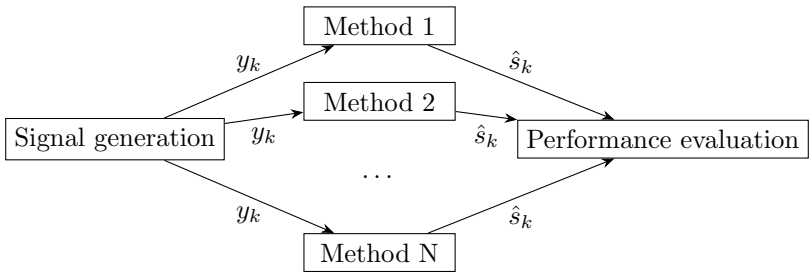


Figure 2.1: Schematic overview of the simulation

2.1 Signal description

The original signal f that is considered in this report is a sum of complex exponentials. Noise is added to f before it is compressed to obtain the compressed signal y. We will consider only compression matrices Φ whose rows are a subset of the rows of an identity matrix. So to obtain the compressed signal from the noisy signal, m of the n measurements in the noisy signal are selected. This signal generation is illustrated by figure 2.2.

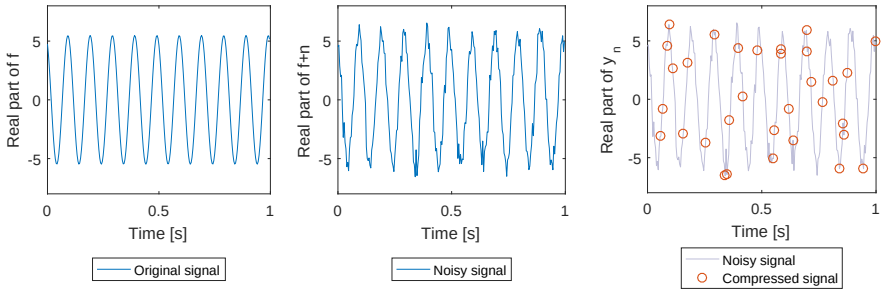


Figure 2.2: An illustration of the original, noisy and compressed signal consisting of a single frequency component

Note that these plots show only the real part of the respective signals while the signals considered

in this report also have an imaginary part.

A single pulse is described by the following expression:

$$y = \Phi\left(\sum_{r=1}^{R} A^{(r)} e^{2\pi i F^{(r)} t + i\xi^{(r)}} + n\right), \quad \text{where} \quad n \sim \mathcal{CN}(0, \sigma^2). \tag{2.1}$$

Here R denotes the number of components and each component r has amplitude $A^{(r)} \in \mathbb{R}$, frequency $F^{(r)} \in \mathbb{R}$ and phase-shift $\xi^{(r)} \in [0, 2\pi]$. The estimated amplitude is complex, so that it contains both the real amplitude and the phase shift. The notation $\mathcal{CN}(0, \sigma^2)$ is used to denote a complex Gaussian distribution with mean 0 and variance σ^2. The term $2\pi F^{(r)} t + i\xi^{(r)}$ will be referred to as the phase. When a series of pulses is considered, a time-index will be added. The amplitude, frequency, and phase-shift of component r of pulse k are then denoted $A_k^{(r)}$, $F_k^{(r)}$, and $\xi_k^{(r)}$, respectively. Unless it is necessary, the index k will dropped to improve readability. Throughout this report, 'component' and 'target' are used interchangeably and a timestep this refers to the index k. In the context of radar, what is referred to as a timestep or pulse in this report, is referred to as 'slow time'. The time within a pulse is then referred to as 'fast time'. In this report such a distinction is not necessary; a timestep refers to 'slow time'.

In this report, $\sigma = 1$ and the amplitudes of the components are determined by the specified Signal to Noise Ratio (SNR). The SNR is defined as the ratio between the power of the signal and the power of the noise. The SNR is usually denoted in dB. The amplitude A of a signal with an SNR of x dB is then $\sqrt{\sigma^2 \cdot 10^{\frac{x}{10}}}$.

The PF and the CS algorithms that are compared and combined in this report make use of the same input signal. To be able to compare them, they have to generate outputs that essentially contain the same information. Buts described in section 4.1.3, a PF will have an estimated posterior density as its output, while CS provides a single point estimate. Therefore, we extract a point estimate from the estimated posterior and determine the performance of the PF based on this point estimate.
This point estimate of the state contains the amplitudes and frequencies of the estimated components: $\hat{s} = [A^{(1)} F^{(1)} ... A^{(R)} F^{(R)}]$. The output $\hat{x}$ of the CS algorithms are related to the grid that is used, which could be interpreted as a number of frequency bins. For example, when a grid between 0.5 Hz and 10.5 Hz with a grid cell of 1 Hz is used, the information 'one component with amplitude 1.5 and frequency 5 Hz, one component with amplitude[1] 2.5 and frequency 7 Hz, both with phase-shift zero', would be presented as $\hat{x} = [0\ 0\ 0\ 0\ 1.5 + 0i\ 0\ 2.5 + 0i\ 0\ 0\ 0]$ in the output of CS. In words that is a response of 1.5 in the frequency bin 4.5 Hz-5.5 Hz and a response of 2.5 in the frequency bin 6.5 Hz - 7.5 Hz. The same information would be in the point estimate of the PF as $\hat{s} = [1.5 + 0i,\ 5,\ 2.5 + 0i,\ 7]$. In section 3.1.3 the conversion of $\hat{x}$ to the format of $\hat{s}$ is discussed.

During the internship that preceded this graduation project [15][2] the idea of running a PF on compressive measurements was explored. One of the challenges when working with compressive measurements is dealing with the correlations between measurements that the compression can cause. Note that it depends on the type of compression whether correlations are introduced at all. An example of a type of compression that introduces correlations is one that sums the signal values over an interval to obtain a compressed measurement. When two compressed measurements have overlapping intervals, they will be correlated. In [15] two methods of dealing with these correlations are investigated: updating the covariance matrix to include these correlations and using a prewhitening transformation.The latter transforms the noisy signal y so that it has uncorrelated noise again.

[1] In practical situations, the amplitude will have a unit. In a radar use-case this could be Volt, for example. However, since we assume the noise level to be known (see also 5.4.1), we will divide any signal by this noise level so that the noise level of the signal becomes one (dimensionless). Therefore we will not denote any unit for amplitude in this report.

[2] The report that is referred to here is not publicly available, but can be requested with the author

This is useful since, when applying CS methods, one often assumes uncorrelated measurement noise. While this is not necessary for running a CS algorithm, it is a common assumption in the derivation of many recovery guarantees. In this report we will use only types of compression that do not introduce correlations. As mentioned earlier, we will consider compression matrices whose rows are a subset of the rows of an identity matrix.

2.1.1 Relation to a radar use-case

As a motivation for considering the signal described in equation (2.1), we relate this signal to the range estimation in Frequency Modulated Continuous Wave (FMCW) radar. In this type of radar, an on-going sequence of pulses with a linear phase is transmitted. That is, the transmitted frequency, denoted f_{transmit}, linearly increases within each pulse. The difference between the highest and lowest frequency in a pulse is the bandwidth, denoted B and the average frequency is referred to as the carrier frequency, denoted f_c. The duration of a pulse is denoted τ. The signal that is received after it is reflected by an object is the same, only delayed by Δ_t. This is is illustrated by figure 2.3.

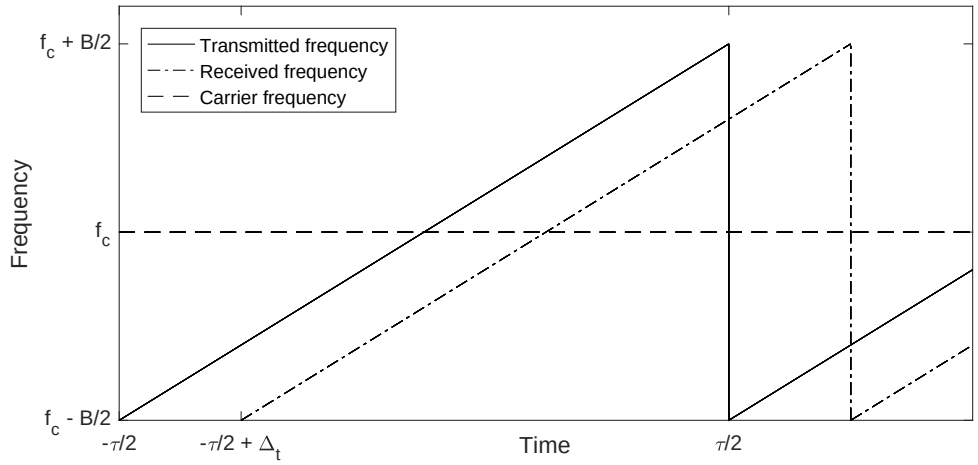


Figure 2.3: An illustration of the transmitted and received frequencies in an FMCW radar

$$f_{\text{transmit}}(t) = f_c + \frac{B}{\tau}t \quad \text{where} \quad -\frac{\tau}{2} < t < \frac{\tau}{2}. \tag{2.2}$$

Consequently, the phase of the transmitted signal, denoted ϕ_{transmit}, is quadratic over time:

$$\phi(t) = 2\pi \int f_{\text{transmit}}(t)dt = 2\pi f_c t + \pi \frac{B}{\tau}t^2. \tag{2.3}$$

That is, the phase at time t is the integral of frequency times 2π. Therefore the transmitted signal s_{transmit} is

$$s_{\text{transmit}}(t) = a \cdot e^{2\pi i f_c t + \pi i \frac{B}{\tau}t^2}. \tag{2.4}$$

This transmitted signal travels through the air until it is reflected by an object. Provided this object is stationary, the signal received by the radar receiver is then

$$s_{\text{receive}}(t) = a' \cdot e^{2\pi i f_c (t-\Delta_t) + \pi i \frac{B}{\tau}(t-\Delta_t)^2}. \tag{2.5}$$

Here a' denotes the amplitude of the received signal, which is determined by a number of factors such as the distance between the target and the radar (denoted r in this subsection). If the wave

travels at speed c - usually the speed of light in a radar context - then $\Delta_t = \frac{2r}{c}$. For the purpose of this section, the value of a' is not important, but in practice it is given by the radar equation, which can be found in any introduction to radar principles. If the goal is to determine r, the next step is to mix the two signals. This operation is defined for a pair of two signals s_1 and s_2 as $s_{\text{mix}} = s_2^* s_1$, where s_2^* is the complex conjugate of s_2. If the received signal is mixed with the transmitted signal, we obtain what is referred to as the beat signal (denoted s_{beat}):

$$s_{\text{beat}}(t) = a'' \cdot e^{\left(2\pi i f_c(t-\Delta_t)+\pi i \frac{B}{\tau}(t-\Delta_t)^2\right)-\left(2\pi i f_c t+\pi i \frac{B}{\tau}t^2\right)} = a''\ e^{2\pi i f_c \Delta_t + 2\pi i \frac{B\Delta_t}{\tau}t - \pi i \frac{B(\Delta_t)^2}{\tau}} = a'' \cdot e^{\phi(t)} \tag{2.6}$$

Now, all that is required to obtain r is to determine the (constant) frequency of the beat signal:

$$f_{\text{beat}}(t) = \frac{1}{2\pi}\frac{d\phi}{dt} = \frac{B\Delta_t}{\tau} = \frac{B}{\tau}\frac{2r}{c}. \tag{2.7}$$

If there are multiple objects that reflect the transmitted wave, the received signal will be the sum of these signals. In that case, the beat signal contains multiple frequencies, which results in a signal of the form described in equation 2.1. There are other examples of relationships between the frequency of the beat signal and distance to a target or its velocity in other radar types, but these will not be discussed here.

2.1.2 Scenario

The considered algorithms will be applied in three scenarios, which are illustrated in figure 2.4.

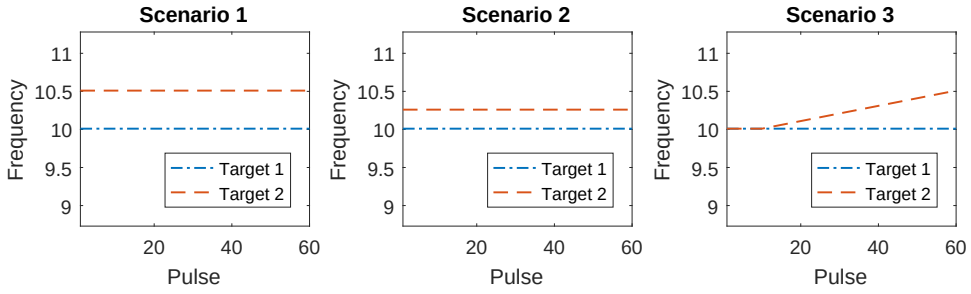


Figure 2.4: A graphical representation of the two scenarios considered

Scenario 1 considers two targets at a distance (i.e. frequency difference) of 0.5 Hz from each other, in scenario 2 this distance is 0.25 Hz. This distance can be varied more to determine how far targets need to be apart before they are distinguished consistently by an algorithm. Then these scenarios could be used to determine the resolution as the smallest distance where the algorithm can distinguish the two targets in at least a given fraction of the Monte Carlo (MC) runs.

In scenario 3 two targets first have the exact same frequency for 10 pulses, after which one moves away from the other at a rate of 0.01Hz per pulse. With this scenario at least two aspects of the algorithm performance can be evaluated at once. During the first 10 pulses the two targets are 'on top of each other', so that there is no way to distinguish them. Since the methods using CS prefer sparser solutions, they can be expected to estimate the number of targets to be one. The posterior distribution of a PF is expected to reflect that it is not possible to distinguish the targets: the hypotheses 'one target' and 'two targets' are then equally likely. If a point estimate would be extracted from this posterior distribution, one could include a preference for a sparser solution as well, for example by including some Information Criterion (IC), as described in section 4.1.2. From pulse 11 onward the targets steadily move apart, so that it intuitively becomes less

difficult to distinguish them. So, the sooner an algorithm can distinguish the two targets, the better.

In all scenarios the amplitude and phase-shift from equation (2.1), and therefore the complex estimate that is estimated as well, are constant but not known to the algorithm.

2.2 Performance evaluation

The state that is estimated by the different algorithms consists of an estimated number of targets and their locations. Since all of the numerical results in this report come from simulations, the ground truth is known and these estimates can be compared to that. Clearly, the closer the estimate is to the truth, the better the estimate. In a situation where there is always one true target and one estimate, defining what the distance between the estimate and the true target is, is fairly straightforward. However, since the true number of targets is not known a-priori, the estimated number of targets does not necessarily match the true number of targets. Therefore, in a situation where the number of targets is also estimated, the definition of the distance between the estimated state and the true state is not obvious. This is illustrated by figure 2.5: it is not intuitively clear which of the three estimated states is the 'closest' to the true state.

Figure 2.5: Different estimates of a situation with two true targets. Figure based on fig. 1 of [35]

In this section we first introduce the Rayleigh resolution criterion, which puts a handle on the maximum distance between a target and an estimate for which we still allow them to be associated to each other. After that, we discuss two approaches to measuring how close the estimated state is to the truth, which make use of this Rayleigh criterion. In the first a number of statistics such as the number of true targets that was correctly identified and the number of false alarms is considered. The second approach considers the distance between the estimated state and the true state to be a combination of the distance between the estimates and the true targets and the difference between the estimated number of targets and the true number of targets.

2.2.1 Rayleigh resolution criterion

To declare whether a true target is found by the algorithm, one needs to specify how close an estimate needs to be to a true target in order to be associated to that target. For this purpose, we will make use of the Rayleigh cell (RC), which has its origins in optics. It is defined as the distance between two point-sources of equal amplitude where the principal intensity maximum of one coincides with the first intensity minimum of the other [7]. When two true targets are at least one RC apart, they can be distinguished according to the Rayleigh resolution criterion. Therefore, it seems reasonable to relate the maximum distance where an estimate can be associated to a true target to this RC.

The Rayleigh resolution criterion can be applied to the context of this report (i.e. distinguishing different frequency components in a pulse) as well. In particular, the Fourier transform of a pulse with a constant frequency is a sinc-function, shown in figure 2.6. In this figure we also see that the

longer the pulse is, the narrower the principal peak of the Fourier transformed pulse is.

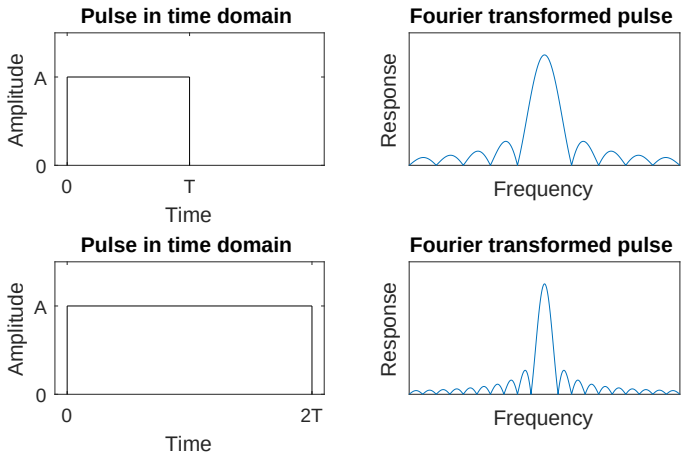


Figure 2.6: Pulses with a constant frequency of different lengths and their Fourier transform

This width is important because when two components are close to each other in terms of frequency, their corresponding sinc-functions will overlap too much, as is illustrated by figure 2.7. There we see that components close to each other results in one large peak (rightmost plot) instead of two distinguishable peaks (leftmost plot). The principal peak of the sinc has its first zero at a distance of $\frac{1}{T}$ from the middle, so that the Rayleigh criterion is at $\frac{1}{T}$ as well. If the targets are further away we declare them to be resolved, if they are closer together they are declared unresolved.

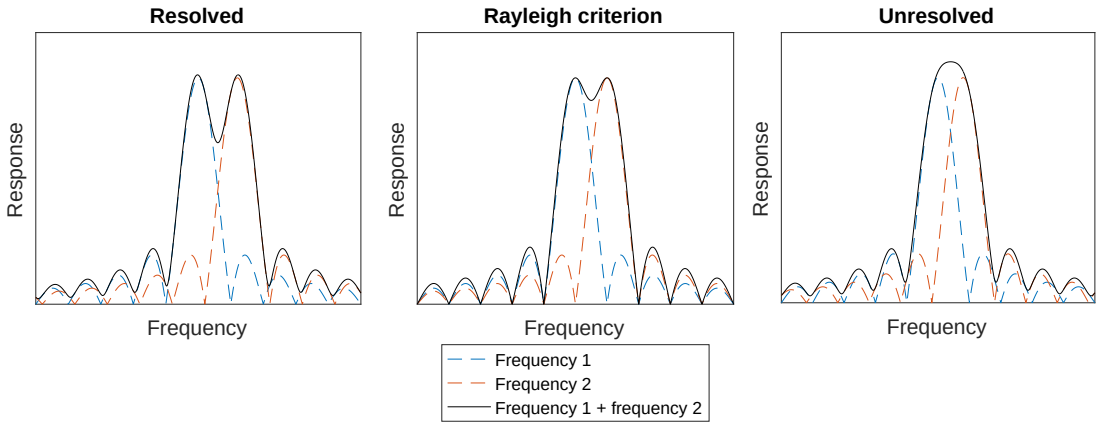


Figure 2.7: Sums of Fourier transformed pulses of different constant frequency components

2.2.2 Association of estimates to true targets

One way to characterize the performance is via the association of estimates to true targets. After that, a number of statistics (which will be introduced later in this section) can be extracted, which can be used to characterize the performance. In e.g. the work of Bekers *et al.* [4] and the work of Zhu [44].

With the definition of the RC from the previous subsection, the following rules are applied for the association of estimated targets to true targets:

- Estimates further than half a RC away from the true target are not associated to this target. So each true target has a window around it with the width of a RC and only estimates inside that window can be associated to this target.
- At most one estimate can be associated to each target.
- Estimates that are outside all windows are considered to be false alarms.
- Estimates are associated to true targets in the way that has the smallest total Euclidean distance.

After the estimates have been associated to the true targets, a number of statistics can be extracted:

- Number of true targets that was found (i.e. have an estimate associated to it);
- Number of true targets missed;
- Number of false alarms;
- Number of estimates inside at least one window but not associated to a true target.

These four statistics together can provide an impression of the performance. Depending on the situation, one statistic might be more important than the other. When analyzing the ability of the algorithm to distinguish two targets, the number of targets missed largely determines the performance. However, an algorithm that consistently finds both targets but produces many false alarms and estimates not associated to a target, is undesirable in many practical situations. The relative importance of these statistics depend on the situation at hand.

An exhaustive search over all possible associations of estimates to true targets is performed. The result of this search is the set of associations which has the smallest total Euclidean distance of those where the maximum number of estimates is associated to a true target. This procedure is detailed by the examples in section 2.2.4.

2.2.3 Optimal Subpattern Assignment Metric

The statistics described in the previous subsection do not provide much information about the accuracy of the different methods. In a sense, accuracy is embedded in the requirement that estimates more than half a RC away from the target cannot be associated to that target, but this requirement does not distinguish between the estimate being close to the target or just barely within the RC around the target. To get a more complete picture of the performance of the different methods, another metric will be included in the analysis: The Optimal Subpattern Assignment (OSPA) metric, proposed by Schuhmacher *et al.* [35].

The OSPA metric considers both the accuracy of the estimated target locations and the estimated number of targets. It makes use of a distance between a given estimate and a given true target similar to the one in the previous subsection. They define the distance with cut-off c between two frequencies $F^{(j)}$ and $F^{(i)}$ to be $d^{(c)}(F^{(j)}, F^{(i)}) = \min(c, \|F^{(i)} - F^{(j)}\|_1)$. Then, given the vector of estimated target locations $\hat{F} = [\hat{F}^{(1)}, ..., \hat{F}^{(n)}]$ and the vector of true target locations $F = [F^{(1)}, ..., F^{(m)}]$, the OSPA of order p with cut-off c is defined as

$$OSPA_p^{(c)}(F, \hat{F}) = \begin{cases} \frac{1}{R} \left\| \left(\min_{\pi \in \Pi_n} \Sigma_{i=1}^{m} d^{(c)}(\hat{F}^{(\pi(i))}, F^{(i)}), c^p(n-m) \right) \right\|_p & m \leq n \\ OSPA_p^{(c)}(\hat{F}, F) & m > n \end{cases} \tag{2.8}$$

Here Π_n is the set of permutations of $\{1, 2, ..., n\}$ and $\|\cdot\|_p$ is the L^p-norm: $\|(x, y)\|_p = (|x|^p + |y|^p)^{\frac{1}{p}}$. The first term considers the sum of the 'cut-off'-distances between the true targets and estimates, the second term considers the difference between the number of true targets and

the number of estimated targets. With this definition, c is the penalty that is given to estimates that are more than c away from any of the true targets.

The parameter c is interpreted as the maximum distance between an estimate and a true target at which the estimate can be assigned to that target. As suggested at the start of this section, we will take this to be half a RC, i.e. $c = \frac{1}{2T}$. The order parameter p determines how sensitive the metric is to estimates far from any of the true targets. The higher p, the more sensitive the metric is to such estimates. Schuhmacher *et al.* suggest using $p = 2$, which is what we will do here as well. An important advantage of taking $p = 2$ is that the association of targets within a RC of a true target in the previous subsection will correspond to the same pairs of estimates and true targets as the permutation π of $\min_{\pi \in \Pi_n} \Sigma_{i=1}^{m} d^{(c)}(\hat{F}^{(\pi(i))}, F^{(i)})$.

2.2.4 Examples

This subsection presents some examples that illustrate the procedure of associating estimates to true targets. It is recommended to look at these pictures in color rather than in grayscale.

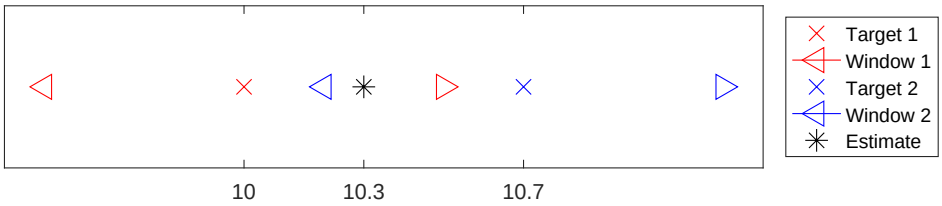


Figure 2.8: Two targets, one estimate

Figure 2.8 shows a situation where there are two true targets, but only one estimate. This estimate can be associated to both of the true targets, but will be associated to the closest true target: target 1 in this case. So in this case we have one missed target and one target found. The OSPA in this example is $\sqrt{0.3^2 + 0.5^2} \approx 0.58$.

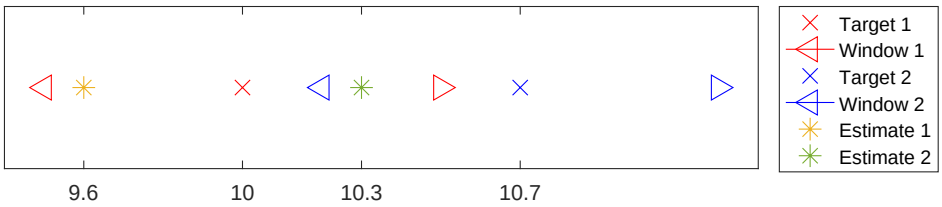


Figure 2.9: Two targets, two estimates

In figure 2.9 we have two estimates that are both inside at least one window. Since estimate 1 is outside the window of target 2 the distance between them is infinite by our definition. Therefore in the association with the smallest total distance estimate 2 is associated to estimate 1 while estimate 1 is associated to target 2, even though estimate 2 is closer to target 1 than to target 2. The OSPA in this example is $\sqrt{0.4^2 + 0.4^2} \approx 0.57$

In figure 2.10 a third estimate is added. Since this estimate is outside both windows, it is considered to be a false alarm. The two remaining estimates are associated as in the previous example. The OSPA in this example is $\sqrt{0.4^2 + 0.4^2 + 0.5^2} \approx 0.75$.

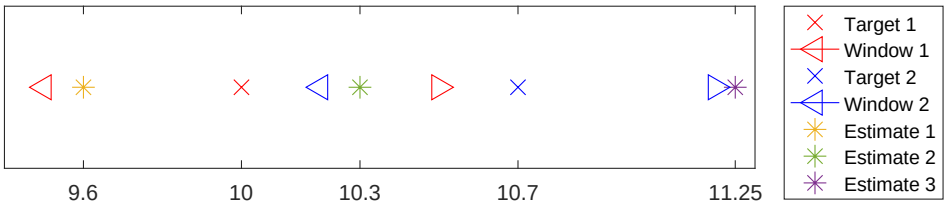


Figure 2.10: Two targets, three estimates

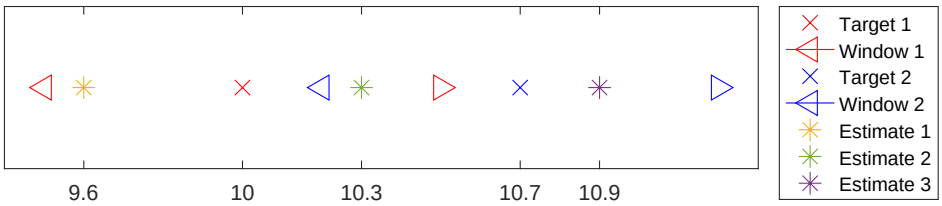


Figure 2.11: Two targets, three estimates

Figure 2.11 shows what happens if the third estimate is closer to target 2. In this case estimate 3 will be associated to target 2 and estimate 2 is associated to target 1. Estimate 1 cannot be associated to any of the targets, but since it is inside at least one window, it will not be classified as a false alarm. Such a situation might arise when one true target results in more than one estimate or in a cluttered scene where objects might be mistaken for targets. The OSPA in this example is $\sqrt{0.3^2 + 0.2^2 + 0.5^2} = 0.62$.

As a last example we consider the situation where two estimates are equally far from a true target: a tie. In the case of a tie, the entry that comes last in its row/column will be used. Since the vector will be ordered by frequency in the context of this report, this will be the one with the highest frequency. Ties might occur when grid-based methods like CS are used, while the probability of a tie is zero for methods like the PF, whose estimates can be anywhere in the continuous state space.

2.3 Algorithm efficiency

In many dynamic scenes it is not only of importance to obtain an accurate estimate of the state, but also to obtain it quickly. There is usually a trade-off between speed and accuracy. Both CS and a PF have parameters that can be changed to shift the balance between accuracy and computational resources, which are introduced in sections 3.1.2 and 4.1.1 respectively. If a better accuracy/resolution is desired, more computational resources will be required. These parameters can be tuned so that all algorithms have the same performance (e.g. in terms of ability to distinguish two targets), so that the amount of resources can be compared. This provides an answer to the question how efficient these algorithms are.

A practical measure of the efficiency is the empirical run-time, i.e., the time between handing the input to the algorithm and receiving the output. It should be noted that this measure depends strongly on factors that have nothing to do with the algorithm in principle. For example, the run-time strongly depends on the way an algorithm is implemented, whether the machine that the algorithm runs on, whether algorithm allow for parallel processing, what programming language is used, and how well it suits the algorithm. An alternative measure of algorithm efficiency is offered by the theoretical growth orders of memory and run-time. This measure is discussed in Appendix D.

3. Dynamic Compressive Sensing

In this chapter we investigate different variants of Dynamic CS. These variants are based on the same optimization problem as static CS, namely BPDN (as discussed in section 1.1). As an introduction to the different variants, we first describe in section 3.1 the specific convex optimization algorithm that we will use to solve BPDN. The main section of this chapter is section 3.2, in which we discuss Dynamic CS and the proposed variants. In particular, the variants of the Dynamic Mod-BPDN algorithm described in the literature are proposed in sections 3.2.3 and 3.2.4. The difference between these variants is in how they make use of information from the previous timesteps. But they are all motivated by the idea that information from previous timesteps can help to speed up the optimization algorithm.

3.1 Optimization

In this subsection, we discuss algorithms that can be used to solve the (L1 relaxation) optimization problems such as the (variants of) BPDN from equations (1.4) and (1.7). The focus of this section is on the solver that will be used: YALL1. Besides the algorithm that this solver uses, we also discuss its most important parameters and how they are tuned for our purpose.

3.1.1 YALL1 algorithm

The solver that was used during this project is 'Your ALgorithms for L1 optimization' (YALL1) [42, 43], which is a Matlab solver that can be used to solve a variety of ℓ^1-minimization problems. The algorithm is grid-based in the sense that the estimated state always lie on a pre-specified grid. It is assumed that the measurements are a linear function of the underlying state plus uncorrelated Gaussian noise (i.e. $y = \Psi x + \nu$).

To solve the optimization problem efficiently it relies on the Alternating Direction Method (ADM). In the ADM, problems of the following form are considered:

$$\min_{x,y}\{F_1(x) + F_2(y)|Ax + By = b\}. \tag{3.1}$$

where F_1 and F_2 are convex functions. The property of such problems that ADM exploits is that the variables x and y are only coupled through the constraint, but they are separated in the objective. A classic way to solve this problem is the augmented Lagrangian method. In this method, the augmented Lagrangian, denoted $\mathcal{L}$, of the problem would be minimized iteratively. In each iteration the augmented Lagrangian is minimized for a fixed value of the Lagrange multiplier λ, with respect to x and y simultaneously. This joint minimization is what makes the classic augmented Lagrangian method expensive and inefficient. The key to the efficiency of ADM is in this separability of the variables involved: the costly joint minimization of x and y is replaced by two simpler sub-problems. More specifically: in each iteration of ADM $\mathcal{L}$ is minimized first with respect to x, given the values of y and λ from the previous iteration. Then, with this value for x, $\mathcal{L}$ is minimized with respect

to y. And then finally, with the new values for x and y, also with respect to λ. A more elaborate description of the ADM applied to BPDN (equation (1.4)) can be found in the work of Yang and Zhang [41]. In this work, they also compare its performance to other ℓ^1-solvers, and conclude that YALL1 is "efficient and robust" and "competitive with other state-of-the-art algorithms" [41].

3.1.2 YALL1 parameters

Weight of signal fidelity

As mentioned in the introduction of this report, the problem that is solved by YALL1 in the context of this project - see equations (1.4) and (1.7) - aims to balance sparsity and signal fidelity. The relative importance of these two factors is determined by the parameter ρ, where the weight for signal fidelity is inversely proportional to ρ. A larger ρ means a smaller weight for signal fidelity and therefore a larger relative weight of sparsity. Therefore, the larger ρ, the sparser the solution.

Since different values of ρ usually lead to different solutions of the optimization problem, choosing a proper value for ρ is important. For the purpose of this project we will set ρ based on a noise-only simulation. In this simulation, a noise-only signal is fed into YALL1, for a range of values for ρ. For each of these values of ρ the fraction of MC runs where at least one target was found is used to approximate the false alarm rate (FAR) corresponding to this ρ.

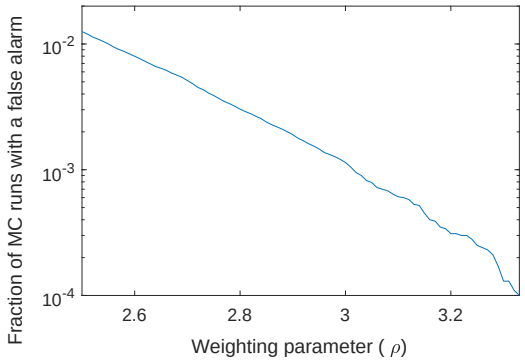


Figure 3.1: Fraction of MC runs with a false alarm for varying ρ

In this figure the ρ corresponding to the desired FAR can be found.

Table 3.1: ρ's corresponding to given false alarm rates

Noise-only FAR	10^{-2}	10^{-3}	10^{-4}
ρ	2.55	3.01	3.33

Stopping tolerance

The YALL1 algorithm has two stopping criteria, which are checked every two iterations. The first concerns the relative change:

$$\frac{\|x^i - x^{i-1}\|}{\|x^i\|} < (1-q)\epsilon. \tag{3.2}$$

Here x^i denotes the estimated state x after the i^{th} YALL1 iteration. In the YALL1 code q is hard-coded to be 0.1. If this first criterion is not satisfied, the second criterion is checked. This consists

of three inequalities and all three have to be satisfied. The first of these concerns the same relative residual as equation (3.2):

$$\frac{\|x^i - x^{i-1}\|}{\|x\|} < (1+q)\epsilon \tag{3.3}$$

The other two concern the difference between the primal and dual solutions (referred to as the duality gap) and the size of the norm of the residual relative to the norm of the current estimate of the state (referred to as the relative residual). For more details we refer to the work of Yang and Zhang [41]. The choice of ϵ affects the number of iterations that are required to meet the stopping criterion, where a smaller ϵ corresponds to more iterations.

Without going into too much detail we also mention here the parameter γ, which determines the step length in the iterations of YALL1. By default it is set to 1, but that turned out to cause some problems with convergence in the context of this report. In previous work with YALL1, TNO experienced the same issues. These problems did not arise when $\gamma = 0.9$, as was done during this project.

3.1.3 Post-processing

As mentioned in section 2.1 the output of YALL1 has a different way of presenting the information about the state than the PF. In particular, the output of YALL1 is related to a grid: the output corresponds to the intensity of the response for each of the grid-points. The post-processing procedure converts the YALL1 output to a format that can be compared to the point estimate produced by the PF. For on-grid targets this conversion is straightforward, since each nonzero corresponds to an estimated target. The only post-processing that is required is a low threshold to weed out nonzero elements caused by machine-accuracy issues. In this report we will only consider on-grid targets. The main reason for this assumption is the extra post-processing that will be needed for off-grid targets. This is illustrated by the figure below. In section 5.4 the case of off-grid targets is discussed in some more detail.

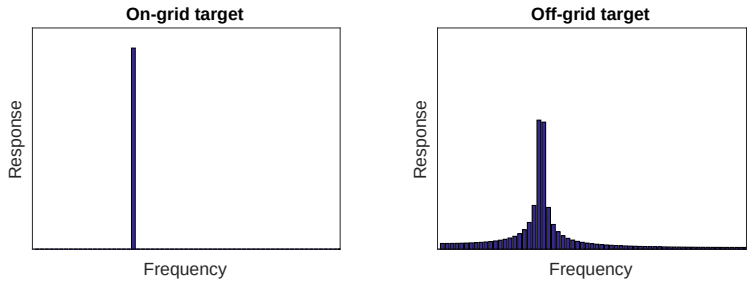


Figure 3.2: An illustration of what the true response of on-grid and off-grid targets might look like

3.2 Dynamic CS

In section 1.2 Dynamic CS was defined as a CS algorithm that uses information from previous pulses during the current pulse. Clearly there are many ways of doing so, four of which will be discussed in this section.

3.2.1 Initial condition

One way to provide prior information to CS is via its initial condition (i.e. the initial condition for YALL1). By default, the initial condition of YALL1 is $x_0 = A^*y$, where A^* denotes the conjugate transpose of the sensing matrix A. Alternatively, one could use the previous state estimate as initial condition. Intuitively that makes sense if the state does not change much between consecutive pulses. Then, unless the realization of the noise at this or the previous pulse was unfortunate, YALL1 already starts close to the minimum of the objective function, so the number of iterations that is required to converge is likely to be small.

3.2.2 Dynamic Mod-BPDN

As described in section 1.2, Vaswani and Lu [40] proposed to use the estimated state from the previous timestep to change the weights in the objective function at the current timestep, in what they called Dynamic Mod-BPDN. In their algorithm the weights in the weighted ℓ^1-norm (see equation (1.6)) are used to introduce the prior information. Their way of providing prior information to the next pulse is illustrated by the figure below. This and the other figures in this section are just illustrations, not examples of weights that were actually used in the numerical simulations.

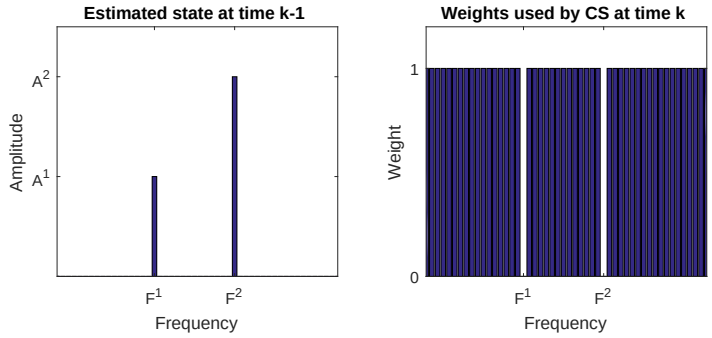


Figure 3.3: A graphical representation of the procedure determining the Dynamic Mod-CS weights

More precisely, the bins where targets were found in the previous pulses, are not penalized. This approach only makes sense if the locations where targets are present, are interpreted as 'known' target locations in the next timesteps, without any uncertainty. With this interpretation, Dynamic Mod-BPDN is expected to work well in situations where the targets do not move to other bins, but other (new) targets might pop up. However, the interpretation is not justified if the 'known' target locations were in fact false alarms or when targets may have moved to a different bin in the mean time.

3.2.3 Dynamic Mod-BPDN with nonzero weights: Dynamic Mod-BPDN+

When the interpretation of previously estimated as 'known' target locations is not justified, setting their weights to zero might lead to problems. Once a weight is set to zero - even if it originated from a false alarm or a target that has moved in the meantime - it offers the optimization algorithm an unpenalized degree of freedom to describe the measurements. As a result, the optimization algorithm is likely to include more targets than justified by the measurements. Therefore we propose Dynamic Mod-BPDN+, using the '+' to indicate that the weights at 'known' target locations are strictly positive, so not exactly zero as in Dynamic Mod-BPDN. The weights at 'known' target locations can now be interpreted as 'probable' target locations. With this interpretation, it makes sense that the weight at a location relates to the probability of a target being at that position. One way to proceed is to use the rule of thumb that targets with a high amplitude (i.e. a high SNR) are more

likely to be found. Then the weight could therefore be set inversely proportional to the estimated amplitude of the target at that location. This procedure is represented graphically by figure 3.4.

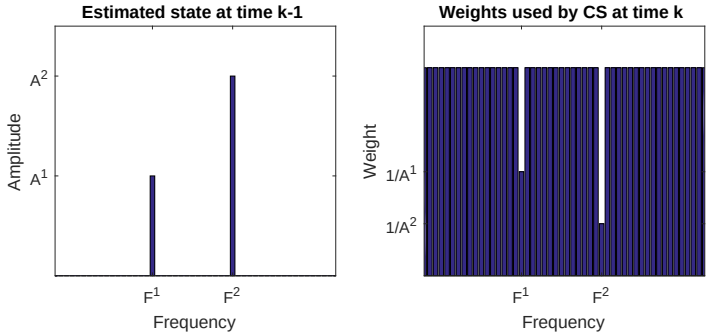


Figure 3.4: A graphical representation of the procedure determining the Dynamic BPDN+ weights

In words, if the state estimate at time $k-1$ was $\hat{s}_{k-1} = [A^1_{k-1}, F^1_{k-1}, A^2_{k-1}, F^2_{k-1}]$, the weights used at time k will be $\frac{1}{1+A^r_{k-1}}$ for the bin containing F^r_{k-1} (where $r = 1, \ ... \ , \hat{R}_{k-1}$) and one elsewhere. With this definition the weight tends to one as the estimated target amplitude tends to zero and the weight tends to zero as the estimated target amplitude tends to infinity.

Finally we mention that there are alternatives to the rule of thumb that is used in this subsection. One specific alternative is described in our recommendations in section 5.5.1.

3.2.4 Dynamic Mod-BPDN with weights from dynamic model: Dynamic Mod-BPDN*

While the methods described so far all make use of information from previous timesteps, none of them takes into account the dynamics of the targets. To do that, the PF described in section 4.1 makes use of a dynamic model for the state: $s_k = g(s_{k-1}) + \omega_k$, where $\omega_k \sim \mathcal{CN}(0, \sigma_\omega)$. We propose Dynamic Mod-BPDN*, which incorporates this same dynamic model in to the Dynamic Mod-BPDN procedures. The dynamic model can be integrated through the weights, similar to how the previous estimate determines the weights for Dynamic Mod-BPDN. A graphical representation of this procedure is shown in figure 3.5.

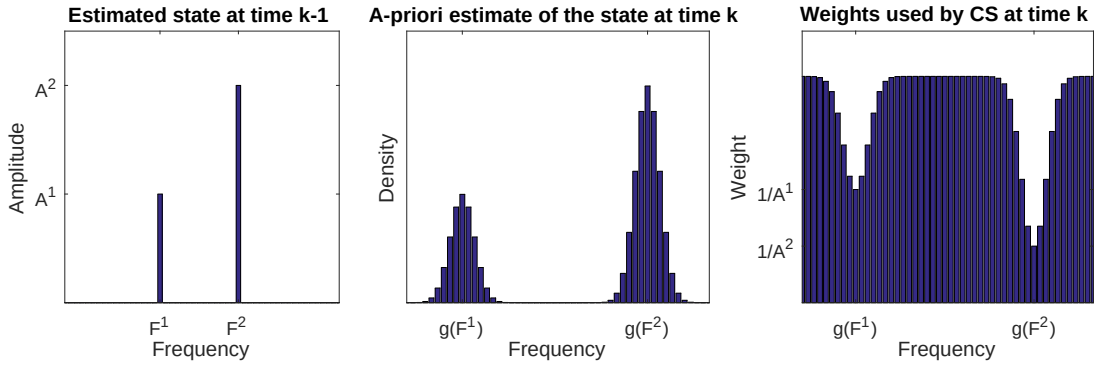


Figure 3.5: A graphical representation of the procedure determining the Dynamic Mod-BPDN* weights

In words, given the estimated state at time k, $\hat{s}_k$ (leftmost plot in figure 3.5), the dynamic model dictates that the prior distribution of the state (middle plot) at the next timestep should be

$\mathcal{CN}(g(\hat{s}_k), \sigma_\omega)$. As in Dynamic Mod-BPDN+, the distribution of the location of target r is then scaled between 0 and $\frac{1}{1+\hat{A}^r_{k-1}}$ and subtracted from an all-ones vector to obtain the weights used at time k (rightmost plot). Note that Dynamic Mod-BPDN+ is a special case of the procedure described in this paragraph. More specifically, Dynamic Mod-BPDN* reduces to Dynamic Mod-BPDN+ when $g(s_k) = s_{k-1}$ and $\omega_k = 0$.

4. Combination of PF and CS

Whereas the previous chapter considered a 'CS-only' approach to CS in dynamic scenes, this chapter considers a combination of CS and PF: the Hybrid combination of PF and CS (HPFCS). This combination is inspired by the Compressive Particle Filter proposed by Ohlsson *et al.* [32]. In the Compressive Particle Filter, a PF with a fixed cardinality is used to track a variable number of targets over time. Clearly, when the cardinality used by the PF does not match the true number of targets, there is a model mismatch. The HPFCS aims to detect this model mismatch by checking a trigger criterion at every timestep. When this criterion is met, CS is performed. Based on the output of CS, the cardinality of the PF can be (but is not necessarily) updated. At the next timestep the PF with the (possibly) new cardinality takes over again. Our implementation of HPFCS makes use of Sequential Importance Resampling (SIR) and YALL1, which are discussed in sections 4.1 and 3.1.1 respectively, but these could be replaced by other PF and CS algorithms respectively.

The motivation for using an algorithm like the HPFCS in this context is to reduce computational resources with respect to a 'PF-only' solution. The advantage of this method compared to a multi-target PF alone is that only one PF is run at all times, instead of multiple in parallel. In a situation where computational resources are restricted, running a number of PFs in parallel might not be feasible while running only one still is. This is especially true when each of the particle filters is required to have many particles, for example because the state dimension is high.
However, not gathering information over time for any cardinality hypothesis other than the one used at that moment is a clear disadvantage. This disadvantage is twofold. Firstly, by gathering information over time the preference for a different cardinality might be confirmed sooner than by waiting for the trigger criterion and then running CS. And secondly, when the cardinality of the PF is changed, a 'new' PF with the updated cardinality is started. Whereas each of the filters running in parallel would have already gathered the information from earlier pulses, this 'new' PF would have to start from nothing.

In this chapter we first provide a description of the PF and its mathematical background. This description also includes the multi-target PF that we would ultimately like to compare to Dynamic CS and the HPFCS. The aspect of the HPFCS that we will focus on in this report is the trigger criterion, which will be discussed in section 4.2. The Compressive PF by Ohlsson *et al.* [32] uses a trigger criterion that is based on likelihood. In subsection 4.2.2 we propose an alternative trigger criterion, which is based on a measure of autocorrelation within the residual.

4.1 Particle Filtering

The theoretical foundation of the PF is the Bayesian framework for solving the filtering problem. Therefore that problem will be the starting point for the introduction of the mathematical background of the PF. The step to a multi-target PF, as described in subsection 4.1.2 mostly concerns implementation of such an algorithm in the context of this report. An important aspect of using a multi-target PF in this context is the extraction of a point estimate, which will be discussed in subsection 4.1.3. We note that single- and multi-target PFs have a rich history in the literature, see for example [23], [2], [28] and references therein.

4.1.1 Bayesian framework for solving the filtering problem

In this section we provide a summary and restate the most important results of the mathematical description of particle filtering in the internship report [15], in which some preparatory work for this project was done. This part of the internship report was largely based on the work of Candy [11] and Arulampalam *et al.* [2]. The filtering problem is the problem of determining $p(s_k|z_{1:k})$, i.e., the distribution of the unknown state s_k given all measurements up to that time. This distribution is often referred to as the filtering distribution, posterior distribution, or simply posterior.

The state dynamics and measurements are described by:

$$\begin{aligned} &\text{Dynamic model:} \quad s_k = g(s_{k-1}) + \omega_k \\ &\text{Measurement model:} \quad z_k = h(s_k) + \nu_k \end{aligned} \tag{4.1}$$

Here g and h are (possibly nonlinear) functions, and ω_k and ν_k will be referred to as process- and measurement noise, respectively. In addition to these models, two more assumptions are made to derive a solution to the filtering problem:

$$\begin{aligned} &\text{The state is Markov: } p(s_k|s_{0:k-1}, z_{1:k-1}) = p(s_k|s_{k-1}) \\ &\text{Measurements only depend on the current state: } p(z_k|z_{1:k-1}, s_{1:k}) = p(z_k|s_k) \end{aligned} \tag{4.2}$$

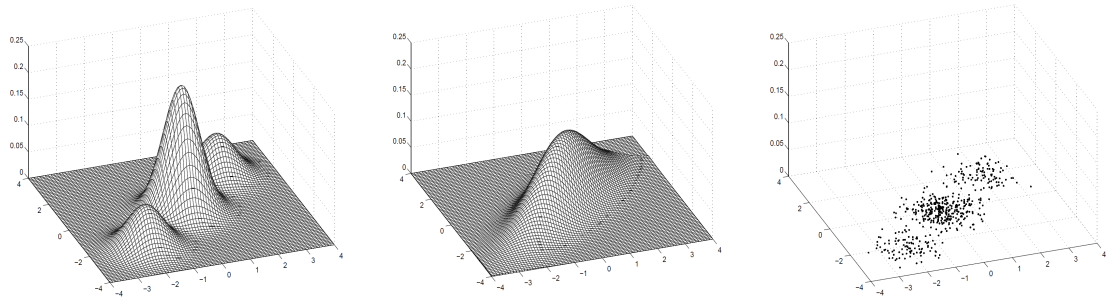
The filtering problem can then be solved by using the following equations sequentially:

$$\begin{aligned} \text{Prediction: } p(s_k|z_{1:k-1} &= \int p(s_k|s_{k-1})p(s_{k-1}|z_{1:k-1})ds_{k-1} \\ \text{Update: } p(s_k|z_{1:k}) &\propto p(z_k|s_k)p(s_k|z_{1:k-1}). \end{aligned} \tag{4.3}$$

Here $p(s_k|s_{k-1})$ is defined by the dynamic model in equation (4.1). These two equations offer an optimal solution to the filtering problem. However, using these equations to solve the filtering problem is only feasible in restricted cases. For example, the case where the dynamic- and measurement models are linear and Gaussian, in which case this sequential Bayesian framework results in the Kalman filter. Whenever the analytical solution is intractable, some kind of approximation is required. For example, even when the dynamic- and measurement models are not linear, one can still approximate them using a linear model, after which the Kalman filter can be applied to the linearized model. This approach is referred to as the Extended Kalman filter. Another kind of approximation is used by a particle filter, in which the posterior is approximated numerically. Instead of approximating the posterior using a pre-determined shape (in the case of an extended Kalman filter: a Gaussian distribution), a particle filter approximates it by a set of N_p particles:

$$p(s_k|z_{1:k}) \approx \sum_{i=1}^{N_p} w_k^i \delta(s_k - s_k^i). \tag{4.4}$$

Here s_k^i is the state of particle i at timestep k and w_k^i is its respective weight, N_p denotes the number of particles, and $\delta(x)$ is a Dirac delta. This Dirac delta can be thought of as an indicator 'function', which is zero everywhere except at x and integrates to unity[1]. In a particle filter the particles and weights are updated recursively. Figure 4.1 shows an example of a distribution and approximations by a Gaussian distribution (as is done in an extended Kalman Filter) and a set of particles. The approximation of the latter can be made arbitrarily good by simply increasing the number of particles. An infinite number of particles would describe the original distribution perfectly and therefore the approximation in equation (4.4) becomes arbitrarily good. In that case the framework described here is an optimal solution to the filtering problem. In practice however, the number of particles is restricted by computational resources and time, so that the approximation in equation (4.4) remains an approximation.



(a) Original distribution

(b) Distribution approximated by a Gaussian distribution (as in an extended Kalman Filter)

(c) Posterior approximated by a set of particles

Figure 4.1: Different approximations of a distribution (reprinted from [6])

The particle weights in equation (4.4) are determined using Importance Sampling (IS). IS is the principle of estimating a target distribution $p(s)$, which we cannot sample from, based on samples from some other distribution, $q(s)$, usually referred to as the importance distribution. To compensate for the discrepancy between $p(s)$ and $q(s)$, weights are introduced. This principle is also applied in the particle filter algorithm that we will discuss later. If IS is applied to the posterior we obtain (see e.g. [2] for the full derivation):

$$p(s_k|z_{1:k}) \approx \sum_{i=1}^{N_p} w_k^i \delta(s_k - s_k^i), \quad \text{where} \quad w_k^i \propto \frac{p(s_k^i|z_{1:k})}{q(s_k^i|z_{1:k})}. \tag{4.5}$$

Now if the importance density q is chosen so that $q(s_k|s_{0:k-1}, z_{1:k}) = q(s_k|s_{k-1}, z_k)$ and $q(s_k|z_{1:k}) = q(s_k|s_{k-1}, z_{1:k})q(s_{k-1}|z_{1:k-1})$, it can be shown (see e.g. [2] for the full derivation) that the weight update in (4.5) can be written as:

$$w_k^i \propto w_{k-1}^i \frac{p(z_k|s_k^i)p(s_k^i|s_{k-1}^i)}{q(s_k^i|s_{k-1}^i, z_k)}. \tag{4.6}$$

With this expression, it is possible to construct a recursive algorithm for approximating the posterior in only two steps:

1. Draw $s_k^i \sim q(s_k|s_{k-1}^i, z_k)$
2. Assign weights according to (4.6)

[1]Note that this is not a formal definition: the Dirac delta is not strictly speaking a function, since a function that is nonzero at only one point cannot have a nonzero integral. But for now this characterization will suffice.

In practice, this algorithm leads to a situation where only a few particles have a significant weight while all others have a very small weight. This problem is called (particle) depletion and a common solution is to include a resampling step. In resampling, a new set of particles is drawn from the discrete approximation of the filtering distribution. In other words, particles with a high weight are duplicated while particles with a low weight are discarded. After resampling, all particles are given weight $\frac{1}{N_p}$, since they are independent, identically distributed samples from the estimated posterior.

The particle filtering algorithm that is used during this project is the Sequential Importance Resampling (SIR) algorithm. The SIR algorithm follows from the aforementioned two-step procedure of approximating the posterior by choosing the following:

- The proposal distribution is taken to be $q(s_k|s_{k-1}^i, z_k) = p(s_k|s_{k-1})$
- Resampling is applied at every timestep

With these choices, equation (4.6) simplifies to $w_k^i \propto p(z_k|s_k)$. Therefore the SIR algorithm can be summarized as in Algorithm 1, where s_k^i is used to denote the state of particle i at time k. Many different resampling algorithms are available; the one used during this project is systematic resampling [2].

Algorithm 1 Sequential Importance Resampling (SIR)

- Initialization: Sample N_p particles from initial distribution $s_0^i \sim p(s_0)$
For time steps $k = 1, 2, ..., K$
- Prediction: draw N_p particles from $p(s_k|s_{k-1}^i)$
- Weight update: set $w_k^i \propto p(z_k|s_k^i)$
- Normalize weights: $w_k^i = \frac{w_k^i}{\sum_{j=1}^{N_p} w_k^j}$
- Resampling: draw N_p particles with repetition from $\sum_{i=1}^{N_p} w_k^i \delta(s_k - s_k^i)$
End For

The PF is just one of the many ways to implement the sequential Bayesian framework. In appendix C an alternative approach to this framework is discussed. In particular, sequential Maximum Likelihood Estimation (MLE) with a deterministic and with a stochastic dynamic model are considered.
Also, the SIR PF that is considered here is just one of the many ways to implement the PF. In this report we will not consider other PFs than the SIR PF, but the reader should be aware that there are many other types of PF, which might be better suited for a given situation. For example, the SIR PF is known to suffer from depletion when the SNR becomes high. In particular, the high SNR results in a high but narrow peak in the likelihood function, so that only a few particles in the SIR PF end up with a significant weight. However, other types of PFs can deal with high SNRs just fine.

4.1.2 Multi-target Particle Filtering

This subsection discusses the implementation of a multi-target PF. To deal with the (possibly) multi-target scene, a number of PFs with different cardinalities will be used in parallel. The number of targets is now one of the dimensions of the posterior distribution estimated by the multi-target PF. Since a point estimate is required in the context of this project, each PF determines its point estimate of the state given its cardinality, denoted R. Then some Information Criterion (IC) is used to determine which of these point estimates is the final output. Much like the BPDN problem considered in the context of CS, this IC tries to balance signal fidelity and sparsity. The cardinality corresponding to the minimum of the IC and its corresponding state estimate is the output of the filter. Therefore the IC is of great importance to the performance of the multi-target PF. Therefore, a selection of ICs that could be used is discussed in appendix A, including a recommendation of

which IC to use in the context of this report.

Figure 4.2 shows an overview of a multi-target filter, with the obtained signal y as input and the point estimate of the state $\hat{s}$ as output, which includes the estimated number of targets.

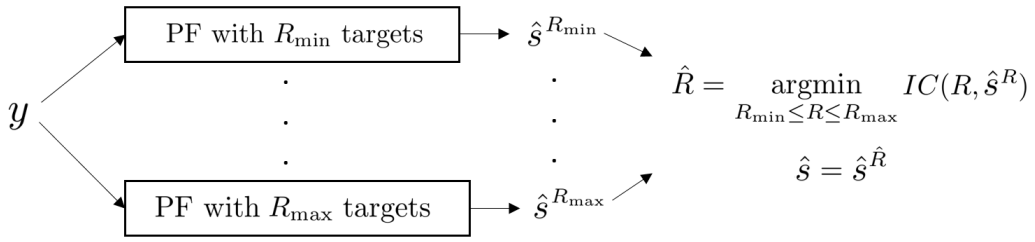


Figure 4.2: A graphical representation of the multi-target PF

The state that these filters estimate at timestep k consists of a frequency and an amplitude for each of the components: $s_k = [A_k^1, F_k^1, ..., A_k^{R_{\text{true}}}, F_k^{R_{\text{true}}}]$, where R_{true} is the true number of targets. It should be noted that setting the range of possible cardinalities a-priori provides the algorithm with prior information ('the number of targets is between $R_{\max}$ and $R_{\min}$'). Also, if the true number of targets is outside this range (i.e. $R_{\min} > R_{\text{true}}$ or $R_{\max} < R_{\text{true}}$), there is no way to get the correct answer.

In the SIR algorithm described in section 4.1.1, there are still three distributions that have to be specified specified: the initial distribution $p(s_0)$, the transition distribution $p(s_k|s_{k-1})$, and the likelihood $p(z_k|s_k)$. We will specify them as follows:

The initial distribution is a uniform distribution covering the frequency bandwidth. The transition distribution follows directly from the dynamic model in (4.1):

$$p(s_k|s_{k-1}) = \mathcal{N}(g(s_{k-1}), \sigma_\omega). \tag{4.7}$$

The likelihood of particle i follows from equation (2.1):

$$L(s_k^i|y) = e^{-ln(|\pi C|) - \frac{1}{2}(y_{\text{est}}(s_k^i)-y)^H C^{-1}(y_{\text{est}}(s_k^i)-y)}. \tag{4.8}$$

In this equation $|\pi C|$ denotes the determinant of the matrix πC, C is the covariance matrix of the noise and $y_{\text{est}}(s_k^i)$ is what the (noiseless) signal is supposed to look like if the state hypothesized by particle s_k^i was the true state. More precisely, if $s^i = [\hat{A}^{(1)}\hat{F}^{(1)}, ..., \hat{A}^{(\hat{R})}\hat{F}^{(\hat{R})}]$, then

$$y_{\text{est}}(s^i) = \Phi\left(\sum_{r=1}^{\hat{R}} \hat{A}^{(r)} e^{2\pi i \hat{F}^{(r)} t}\right). \tag{4.9}$$

We note here that we have considered also another multi-target PF. With this PF we investigated the effect of an improved initial distribution on the convergence properties in particular. Such an initial distribution can be obtained by e.g. a CS algorithm. The main finding is that a correct cardinality estimation is more important than a precise initial distribution for the target locations. The discussion of these findings can be found in appendix B.

4.1.3 Extracting a point estimate

The extraction of the point estimate is of great importance when the performance of a particle filter is evaluated using the performance measure that was defined in section 2.2. The full joint probability distribution is estimated by a PF. In the case of multiple targets this distribution is often

multi-modal, so that one should be careful when extracting a point estimate from it. For example, the mean of a bi-modal distribution is likely to be somewhere between the two modes, in an area where the posterior probability is not necessarily high (or in extreme situations perhaps even zero).

This problem is caused by the fact that the likelihood (see equation (4.10)) of a particle is permutation-invariant: $[A^1, F^1, A^2, F^2]$ has the exact same likelihood as $[A^2, F^2, A^1, F^1]$. In other words, the target corresponding to component j of a given particle is not necessarily the same target as component j of another particle. This is a well-known problem in multi-target particle filters, which is often referred to as the mixed-labeling problem. A possible way to deal with this problem is apply a type of clustering to 'order' the targets within particles. For example, Kreucher *et al.* [23] use k-means clustering. In this report, we will deal with this problem by sorting the components in a particle by frequency, which could be interpreted as one-dimensional clustering. While simple and cheap in terms of computational load, it has proven to be sufficient for our purpose. As a result, the posterior distribution of each of the components individually will be (at least close to) unimodal and symmetric, so that the weighted average can be used to extract a point estimate.

Instead of clustering the particles' states the mixed-labeling problem could be dealt with by using a different point estimate, such as the Maximum A Posteriori (MAP) estimate (as suggested in e.g. [34]). However, computing the MAP is a bit more involved than computing the mean of the estimated posterior and the mixed-labeling problem can be circumvented by using the simple one-dimensional clustering step. Therefore, the weighted average will be used in this project.

4.2 Trigger criterion

With the computational resources in mind, we would like to run CS only when we expect it may actually change the cardinality. As long as we suspect the PF has the correct cardinality, we wish the trigger criterion is not satisfied. In other words, we aim to detect whether there is a mismatch between the cardinality used by the PF and the true number of targets. This goal is illustrated by figures 4.3 and 4.4. In these figures, the residual is the difference between the noisy signal and the fitted signal.

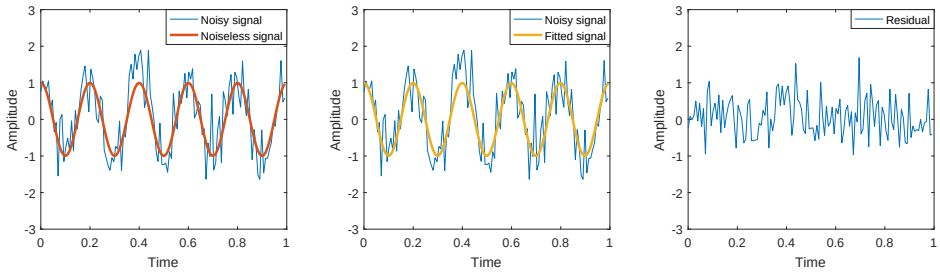


Figure 4.3: An illustration of the noisless signal and the noisy signal (leftmost plot), the same noisy signal and the fitted signal (middle plot) and the residual

Figure 4.3 shows an example of a situation where the cardinality of the PF matches the number of components of the true signal. Specifically, the true signal and the signal according to the state estimate of the PF, y_{est} in equation (4.9) (the fitted signal) both consist of one component. In figure 4.4 however, the true signal contains two components, while the fitted signal contains only one. We now discuss two trigger criteria, that aim to detect this mismatch between the true number of components and the cardinality of the PF based on the residual.

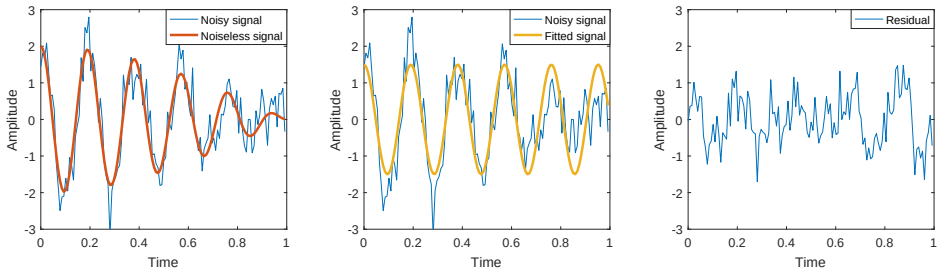


Figure 4.4: As figure 4.3, but for a mismatched model instead of a matched model

4.2.1 Likelihood

The criterion used by Ohlsson *et al.* in their Nonlinear Compressive PF is based on the likelihood. Given the measurements y, the likelihood of the point estimate of the PF at the current timestep, $\hat{s}_k$, is computed in the same manner as the likelihood of a particle. Specifically:

$$L(\hat{s}_k|y) = e^{-ln(|\pi n|)-\frac{1}{2}(y_{\text{est}}(s_k)-y)^H(y_{\text{est}}(s_k)-y)} \propto e^{-(y_{\text{est}}(s_k)-y)^H(y_{\text{est}}(s_k)-y)}. \tag{4.10}$$

The intuition behind this criterion is clear. A mismatched cardinality leads to a large residual $(y_{\text{est}}(s_k) - y)$ and therefore a low likelihood. So if the likelihood of $\hat{s}_k$ is low, that indicates that the current cardinality of the PF does not match the true number of targets. Therefore, when the likelihood is below a certain threshold, CS should be started. What this threshold should be depends on the situation at hand. One way to tune the threshold is to look at the FAR in the case of a perfectly matched model. Then we set the threshold so that in a certain (acceptable) fraction of the MC runs, using a PF with the correct cardinality, the criterion is met.

4.2.2 Autocorrelation

An alternative way to detect a model mismatch is via autocorrelation within the residual. If the cardinality of the PF is too low, the residual still contains a periodic signal, as illustrated by figure 4.4. Provided the frequency of this periodic signal is not too high with respect to the sampling rate, the residual will then have a high autocorrelation. When the cardinality of the PF matches the true number of components the residual is only noise, which should have no autocorrelation at all, as illustrated by figure 4.3.

A measure for sample autocorrelation can be borrowed from the Ljung-Box test [25], which tests the null-hypothesis that the data are independently distributed against the alternative that they exhibit correlation. The statistic that is used in this test is:

$$LB = m(m+2)\sum_{j=1}^{h}\frac{\hat{\rho}_j^2}{m-j}. \tag{4.11}$$

Here n is the sample size, h is the number of lags, and $\hat{\rho}_j$ denotes the j-lag sample autocorrelation, which is defined as:

$$\hat{\rho}_j = \sum_{t=1}^{m-j}(r_t - \bar{r})(r_{t+j} - \bar{r}), \tag{4.12}$$

where r_t is the t^{th} entry of the residual vector: $r = y - y_{\text{est}}$ and $\bar{r}$ is the average residual. The Ljung-Box test has its origins in time-series analysis, where the number of lags h that is incorporated in the computation of the statistic is normally determined according to seasonality in the data. In that

case the maximum lag is usually set to twice the length of the seasonal pattern. For the signals that are considered in this report this rule of thumb suggests setting $h \approx 10$. The threshold for this criterion can be tuned using the same procedure as the likelihood criterion in the previous paragraph.

5. Numerical results and analysis

In this chapter, we first present and analyze the numerical results concerning the variants of Dynamic CS and the HPFCS trigger criteria, in sections 5.2 and 5.3 respectively. The settings used to obtain these numerical results are discussed in section 5.1. The limitations to the extent to which these results may be generalized are discussed in subsection 5.4.1.
Then, in section we provide directions towards a proper comparison of Dynamic CS, the HPFCS and a multi-target PF. Finally we discuss alternatives and extensions of the research in this report in section 5.5.

5.1 Numerical settings

All simulations were performed under Windows 10 Enterprise 64-bit on a desktop PC with Intel® Core™ i7-6700 CPU @ 3.40Ghz and 8,00GB of Random Access Memory (RAM), running Matlab 2016b.

For the simulations concerning Dynamic CS (subsection 5.2) an SNR of 30 dB was used and 500 MC runs were performed. For the simulations concerning the HPFCS trigger criteria (subsection 5.3) the SNR was 10 dB and 10^4 MC runs were performed. For Dynamic CS a stopping tolerance (ϵ) of 10^{-4} and a step length (γ) of 0.9 was used. The weighting parameter ρ was set according to the FAR simulation results in table 3.1, which suggest to use $\rho = 3.01$ for a noise-only FAR of 10^{-3}. For the simulations concerning the HPFCS, 1000 particles were used.

The original signal contains $n = 256$ measurements, sampled at 256 Hz, so that the integration time T is 1 second. Therefore the RC has a width of 1 Hz, and the cut-off distance in the OSPA metric is 0.5 Hz. The compression matrix consisted of a random subset of 32 rows of an 256×256 identity matrix, which was the same for all simulations. The bandwidth regarded was 8.73 Hz to 11.28 Hz, so that one grid-cell for CS spans 0.01 Hz. In both scenarios Dynamic Mod-BPDN* and the HPFCS will use the following dynamic model:

$$s_k = s_{k-1} + \omega_k. \tag{5.1}$$

This model does not attempt to describe the targets' movements, so no assumptions have to be made about e.g. the shape of their trajectories. It is only assumed that they will not move too far with respect to their previous position. The difference between consecutive states should be small enough to be covered by the process noise ω_k. Therefore, the process noise on the frequency should at least by larger than the difference in frequency between consecutive pulses, which is 0.01 Hz in scenario 3. In the simulations $\sigma_\omega = 0.05$ for frequency and 0.5 for amplitude is used.

5.2 Dynamic CS

In this subsection, the numerical performance of a number of variants of Dynamic CS is compared. We regard the following variants of Dynamic CS:

- Dynamic BPDN, i.e., weight zero at 'known' target locations
- Dynamic BPDN+, i.e., weight inversely proportional to the estimated amplitude at 'known' target locations
- Dynamic BPDN*, i.e., weights following from the estimated state propagated through the dynamic model

In addition to these three, we include static CS (i.e. single-snapshot CS at every timestep) as a benchmark. If the other variants make use of the prior information effectively, they should outperform static CS. For each of these four we also look at the version 'with initial condition'. Which means that the initial condition for the optimization algorithm (YALL1) is taken to be the estimated state at the previous timestep. The other variants use $x = A^*y$ as their initial condition. Thus, a total of eight variants of (Dynamic) CS is considered.

In section 2.1.2, three scenarios were defined: scenarios 1 and 2, where two targets are at a constant distance from each other, and scenario 3 where they are 'on top of each other' during the first 10 pulses and then move apart for the next 50 pulses, at a rate of 0.01 Hz per pulse. In scenarios 1 and 2 we consider two distances between the two targets: 0.5 Hz and 0.25 Hz respectively.

5.2.1 Scenario 1

Figures 5.1 through 5.4 show the numerical results that were obtained from scenario 1, with a distance of 0.5 Hz between the two targets. In figure 5.1 we see the classifications of the association of estimates to true targets (as discussed in section 2.2.2) for each of the eight variants of (Dynamic) CS. Here we see that at this distance between targets, static CS (with or without initial condition) was able to distinguish the two targets (i.e. never miss one of them) in all the MC runs. The same holds for Dynamic BPDN*, but for none of the other five variants. After one pulse all variants are still equivalent, since there is no prior information yet. After that, Dynamic BPDN, Dynamic BPDN+, and their variants with initial condition, as well as Dynamic BPDN* with initial condition, start to miss true targets while at the same time introducing false alarms.

For Dynamic BPDN these false alarms may be explained by the fact that the weights of frequency bins where targets were found during previous pulses are set to zero. This setting of weights changes the optimization problem significantly, since the weight on sparsity for these locations becomes zero. That is, there is an unpenalized degree of freedom and even if this unpenalized frequency is not close to any of the true targets, the optimization algorithm can use it. Intuitively this should work fine if the 'known target locations' (i.e. the positions of the zero-weights) are at the correct position. However, in practice this is not always the case, especially when the grid is fine, as is the case in these simulations. Then, the estimated location might well be off by a couple of frequency cells (which are 0.01 Hz wide here). At the next pulse this location is then no longer penalized: the optimization algorithm can use this degree of freedom without increasing the ℓ^1-norm part of the objective function. However, the algorithm can also still add another location, at the original penalty. This can explain the extra targets, both the targets inside an RC but not associated and the false alarms, introduced by Dynamic BPDN after the first pulse. The behavior of the association classifications of Dynamic BPDN+ is (roughly) similar to that of Dynamic BPDN. This makes sense since the methods are indeed very similar. Especially when the SNR is high, as in the simulations here. With an SNR of 30 dB and the noise level normalized to one, the target amplitude is $\sqrt{10^{30/10}} \approx 31.6$. If the estimated amplitudes are roughly the same, the weights at the locations where targets were found, will be the reciprocal of this.

Whereas Dynamic BPDN and Dynamic BPDN+ are not able to consistently distinguish the two targets, Dynamic BPDN* was, and without introducing spurious estimates. The uncertainty about the precise target location that is incorporated by propagating the estimated state through

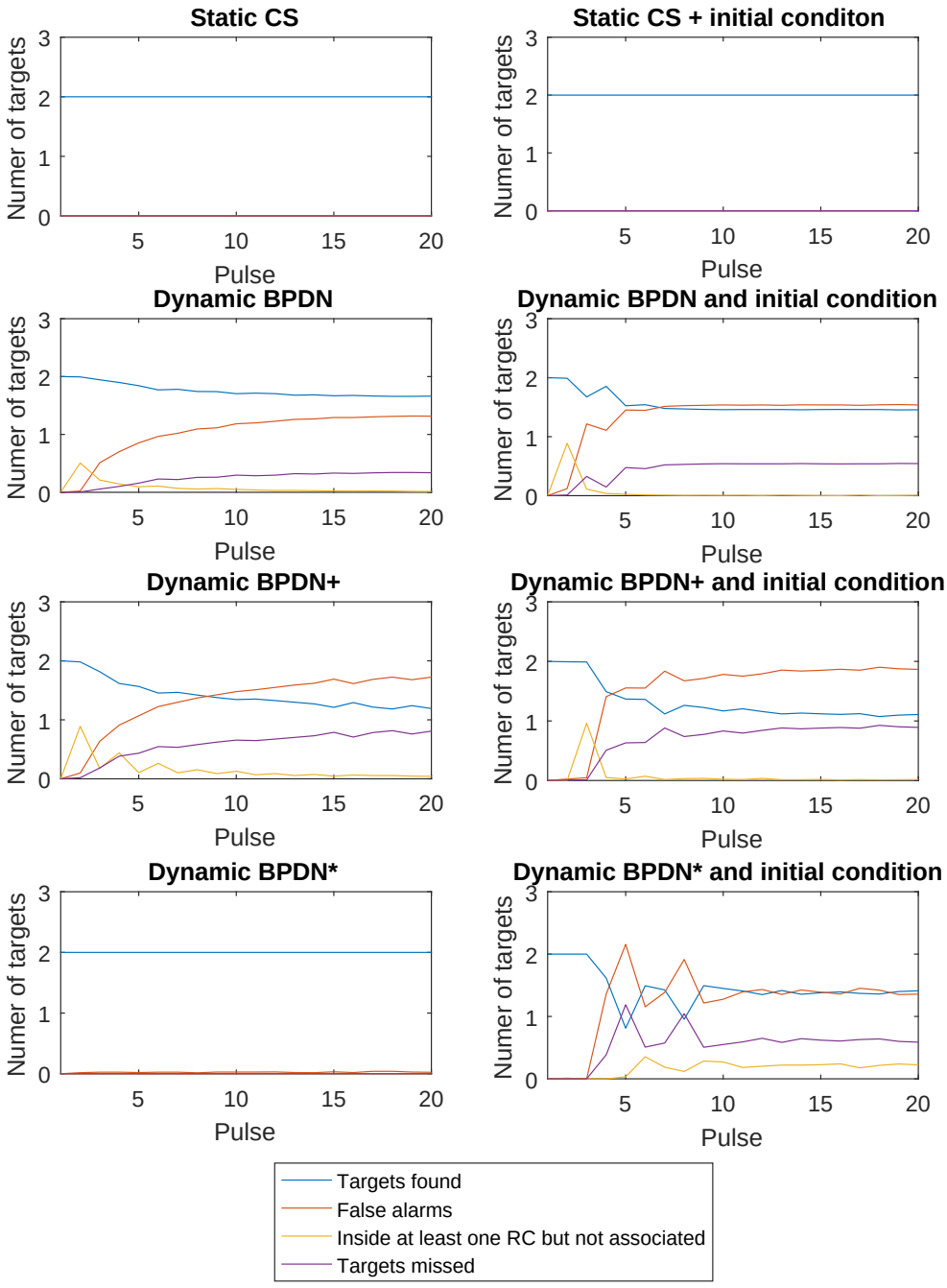


Figure 5.1: Results of the association of estimates to true targets in scenario 1, averaged over all MC runs

the model, apparently helps to solve the problems of Dynamic BPDN and Dynamic BPDN+ that were described above. However, this does not seem to hold when the estimated state is provided to Dynamic BPDN* as initial condtion. Since the only difference is the initial condition, this shows that the optimization problem does not have a unique minimum and that this initial condition leads the optimization algorithm to a significantly different (local) minimum. This is hard to explain,

because the state does not change in this scenario. Therefore, if the estimated state at the current timestep is close to the true state at the current timestep, it will also be close to the true state at the next timestep.

In figure 5.2, the eight 'number of targets found'-plots from figure 5.1 are plotted together in one figure. Here we see that indeed, static CS and Dynamic BPDN* consistently found both targets, while the other variants do not.

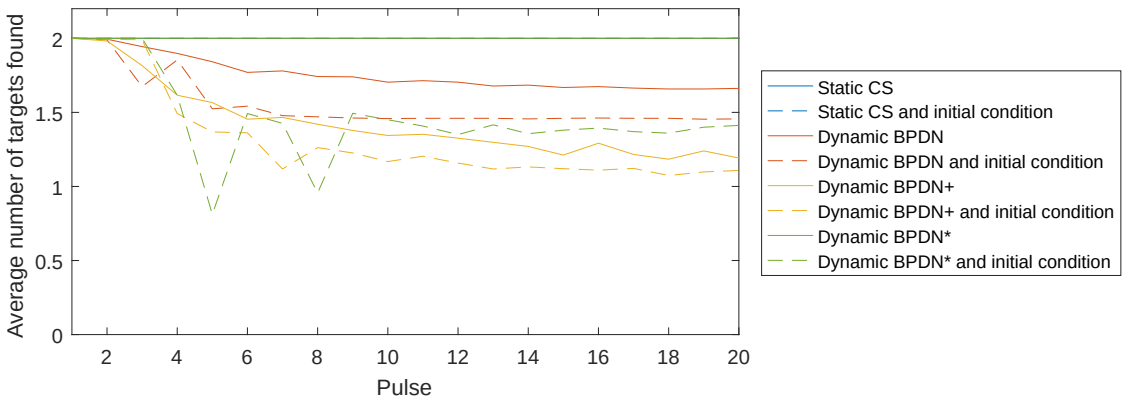


Figure 5.2: Average number of targets found in scenario 1. The plots for static CS are covered by the plot of Dynamic BPDN*.

Figure 5.3 shows the OSPA metric, which leads to largely the same conclusion. Static CS and Dynamic BPDN* do approximately equally well, while the others fall behind. Compared to the average number of targets found, we see here that while Dynamic BPDN (with or without initial condition) had a high number of targets found, while Dynamic BPDN+ (with or without initial condition) has a slightly better OSPA. This indicates that while Dynamic BPDN found more targets, the locations of the targets that Dynamic BPDN+ found were estimated more accurately. The only difference between these variants is the weight being either 0 or the reciprocal of the estimated amplitude, which therefore explains where the difference comes from. The zero weights in Dynamic BPDN lead to more, but less accurate estimates. This makes sense, since the algorithm is more likely to use the previously estimated target location since it is unpenalized, even when its location could be better.

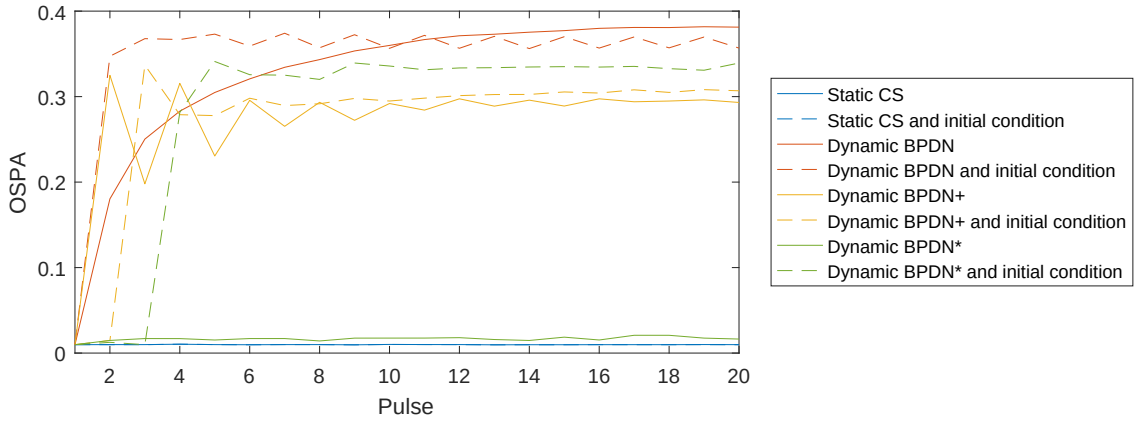


Figure 5.3: OSPA metric in scenario 1

The CPU-times, shown in figure 5.4, do lead to a different picture: Dynamic BPDN* requires less CPU-time than static CS. Since their performance in terms of number of targets found and OSPA are close, this indicates that Dynamic BPDN* makes use of the prior information that is provided. One of the striking features of this plot is the oscillating behavior of the OSPA of some of the variants. As is shown in figure 5.4, the average CPU-time also shows oscillating behavior. Specifically, when the OSPA is high for this timestep, the CPU-time is low at this timestep. This observation indicates that the optimization algorithm somehow terminated prematurely at these timesteps. Since the general behavior of these variants is not satisfactory, we will not further investigate this behavior.

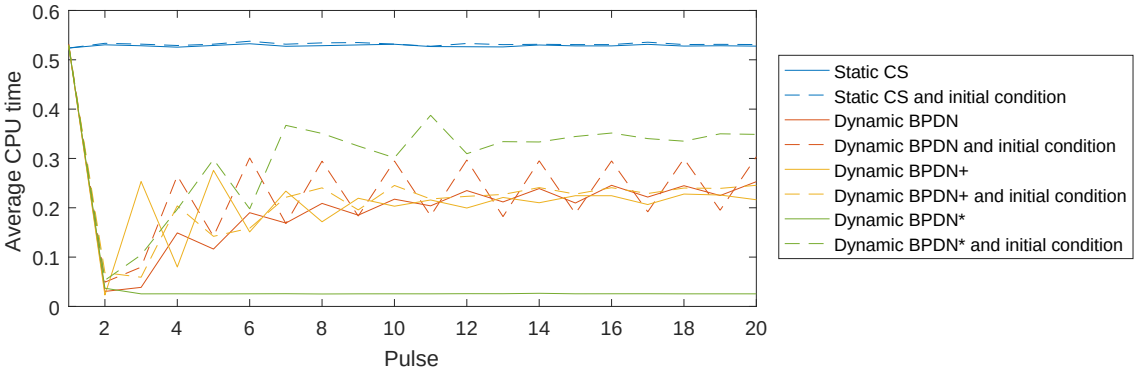


Figure 5.4: Average CPU times in scenario 1

Figures 5.1 through 5.4 show that both variants of Dynamic BPDN and Dynamic BPDN+ as well as Dynamic BPDN* with initial condition all show unsatisfactory results; they do not distinguish the two targets consistently. This scenario is the simplest of the three scenarios considered. The only variant of Dynamic CS that shows the desired behavior is Dynamic BPDN*, which shows what was hoped for: it can distinguish the two targets at lower computational costs than static CS, indicating that it makes use of prior information. Static CS shows the desired behavior, but at a higher computational cost.
To provide a more legible overview of the results, the numerical results for the next two scenarios will no longer show static CS with initial condition, Dynamic BPDN, Dynamic BPDN+ and Dynamic BPDN* with initial condition. Since the performance of static CS with initial condition is the same as with initial condition, we will only consider the variant without initial condition so that only

static CS and Dynamic BPDN* remain.

5.2.2 Scenario 2

Compared to the scenario in the previous subsection, the only difference is that the two targets are closer together. The distance between the two targets is 0.25 Hz instead of 0.5 Hz. This is expected to make it more difficult to distinguish them. Figures 5.5 and 5.6 show that that is indeed the case, since the number of targets found is lower. Specifically: the average number of targets found is roughly 1.8 for both static CS and Dynamic BPDN*, which can be interpreted as a detection probability of 90%. This was 100% in scenario 1. Another striking difference between the two scenarios are the false alarms that are produced by Dynamic BPDN*. These false alarms may be explained by the fact that the tuning of ρ was done in static CS, where all weights in the weighted ℓ^1-norm of equation (1.6) are one. In Dynamic BPDN* the weights depend on the estimated number of targets and many of them may differ from one. As a result the balance between sparsity and signal fidelity in the optimization problem is shifted.
This explanation also indicates possible directions to improve Dynamic BPDN*. One way to do so could be to adjust ρ for the reduced weights. Another way could be to scale the weights so that their sum is unchanged with respect to the weights in static CS.

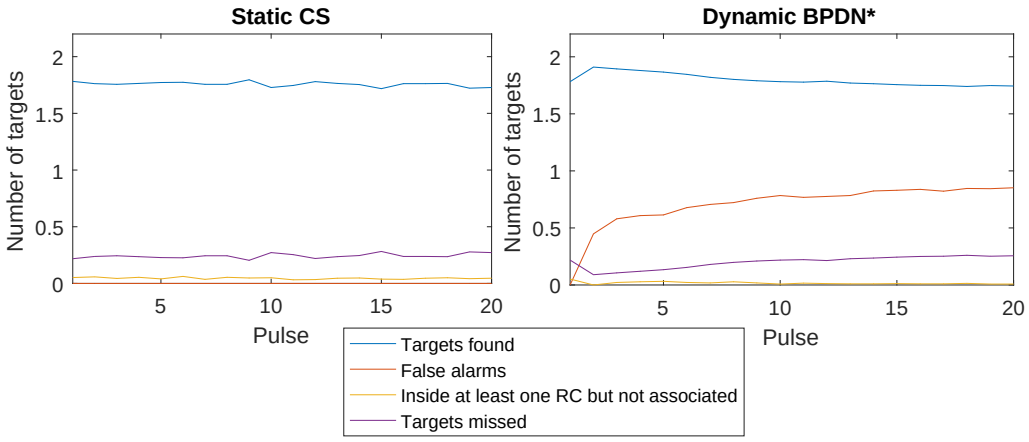


Figure 5.5: Results of the association of estimates to true targets in scenario 2, averaged over all MC runs

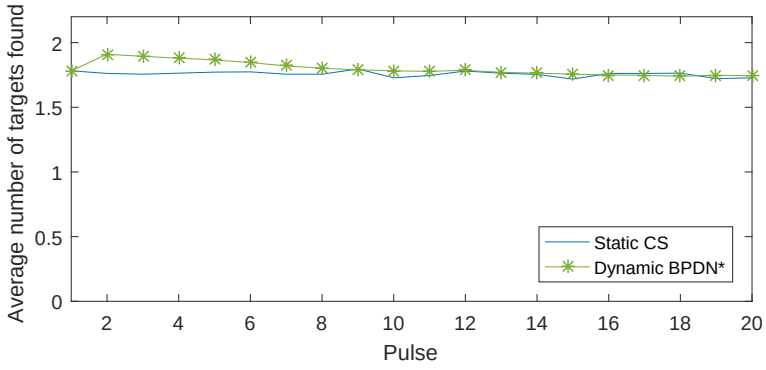


Figure 5.6: Average number of targets found in scenario 2

In figure 5.7 the OSPA metric for scenario 2 is shown. Here we see that while in scenario 1 the OSPA of Dynamic BPDN* and static CS was approximately the same, now the OSPA of static CS is lower. This difference is caused by the spurious estimates that Dynamic BPDN* has, while static CS has almost none.

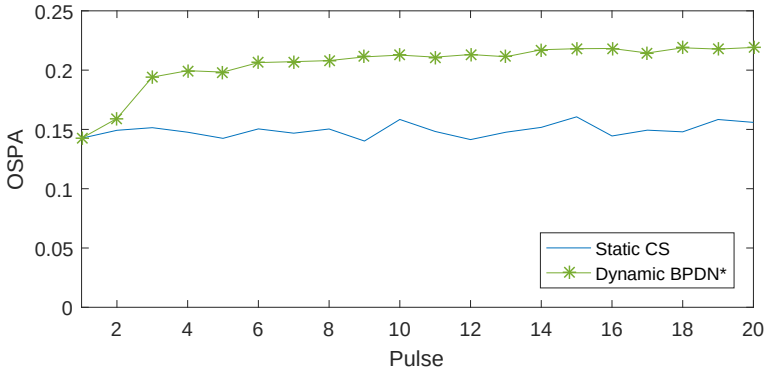


Figure 5.7: OSPA metric in scenario 2

Figure 5.8 shows that Dynamic BPDN* is still faster than static CS, but by a narrower margin than in the previous scenario. In particular, Dynamic BPDN*s CPU-time increased, while that of static CS has decreased compared to scenario 1. The decreased CPU-time of static CS is counter-intuitive, since we consider this scenario to be more difficult than the previous. On the other hand, the decrease in CPU-time does come together with a decrease in accuracy, which does agree with intuition. Another striking feature in this figure is the peak in CPU time at the second pulse for Dynamic BPDN*. This peak coincides with a slight increase in false alarms and number of targets found. We think that the cause for these behaviors is found in the stopping criterion. It seems that this scenario takes a different effect on the stopping criterion of the optimization algorithm than the previous. To indicate a more sound cause for this behavior more research into the stopping criterion is required.

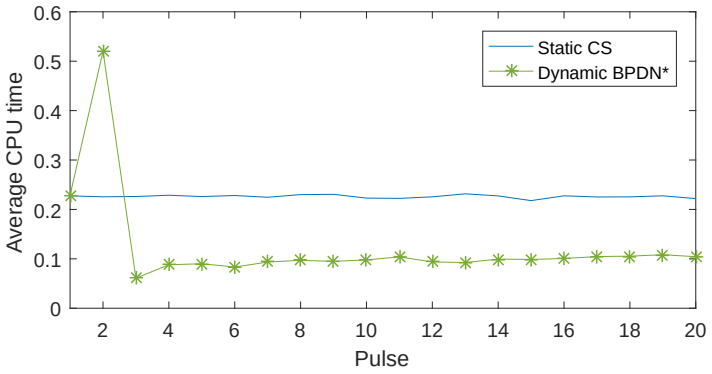


Figure 5.8: Average CPU times in scenario 2

5.2.3 Scenario 3

In the scenario considered in the previous two subsections, the scene was not actually dynamic yet: the target frequencies did not change over time. In this subsection we consider scenario 3, where the frequency of one of the targets does change, starting from pulse 11 onwards. Since one of the targets moves away from the other, it is expected that it becomes increasingly less difficult to distinguish the two.

Figures 5.9 and 5.10 show a lot of the expected behavior in static CS and Dynamic BPDN*. The number of targets found is one during the first 10 pulses, and after 30 pulses for Dynamic BPDN* and 40 pulses for static CS, the two targets are distinguished consistently.

The two features that stand out for static CS are the 'dip' in the number of targets found around pulse 30 and the spurious estimates (inside at least one RC but not associated) around pulse 45. We suspect that these features are caused by interference between the two frequencies, but without further research this suspicion is not easily confirmed. Once the distance between targets is larger than 0.35 Hz, static CS can distinguish the two targets consistently.
Dynamic BPDN* can distinguish the two targets consistently at a smaller distance, approximately 0.15 Hz. Apparently it suffers less from the interference that suspectedly caused the aforementioned 'dip' for static CS. However, as is shown by figure 5.10, it does take a couple of pulses longer before it reacts to the targets splitting up. Furthermore we see that Dynamic BPDN* produces more false alarms than static CS. After the targets start moving apart, immediately two false alarms appear (on average). After Dynamic BPDN* has started to distinguish the two targets consistently, one false alarm remains. As in scenario 2, these false alarms might be explained by the shifted balance between sparsity and signal fidelity caused by the reduced weights in the weighted ℓ^1-norm.

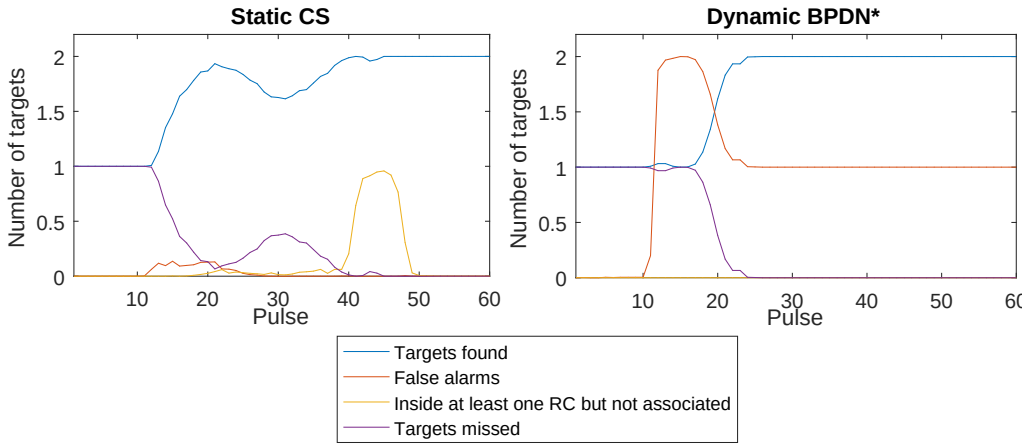


Figure 5.9: Results of the association of estimates to true targets in scenario 3, averaged over all MC runs

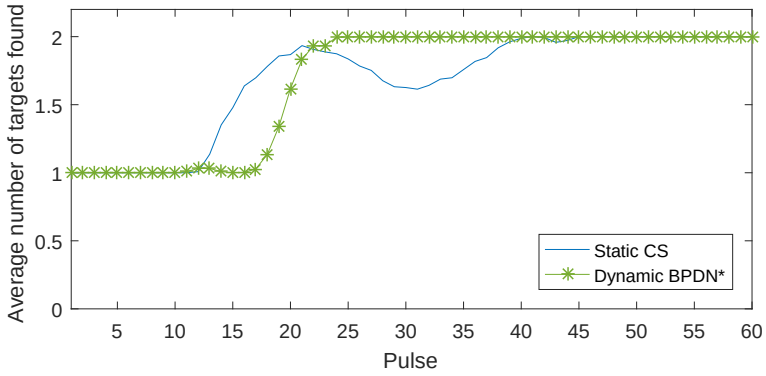


Figure 5.10: Average number of targets found in scenario 3

Figure 5.11 shows the OSPA for this scenario, in which we can recognize some of the striking behaviors from figures 5.9 and 5.10. For example, we recognize the peak of 'estimates inside at least one RC but not associated' of static CS between pulses 40 and 50 and the sudden increase of false alarms for Dynamic BPDN* at pulse 10 and then dropping again after the number of false alarms reduces to one after pulse 20. After that, the estimated number of targets of Dynamic BPDN* does not change anymore, so that its steadily decreasing OSPA can only be explained by its frequency estimate becoming more accurate. This makes sense, since the scenario becomes easier because the targets move apart.

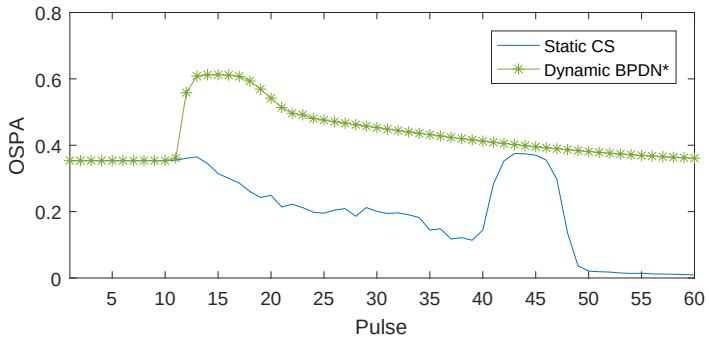


Figure 5.11: OSPA metric in scenario 3

The CPU-times in figure 5.12 show that although the difference is smaller than in the previous two scenarios, Dynamic BPDN* still takes less CPU-time than static CS. What is interesting to see is that between pulses 10 and 20, when the distance between the two targets has just started to increase, the CPU-time of Dynamic BPDN* has a short increase, while the CPU-time of static CS decreases strongly. A possible explanation for this is that for Dynamic BPDN* there is not much 'news' besides this area. At first, it perceives only one stationary target, after they only slowly move apart. There seems to be correlations between the peaks and drops in CPU-time and the number of false alarms starting and/or the number of estimates in a RC but not associated, but without further research we do not dare to claim that any of these relationships is causal.

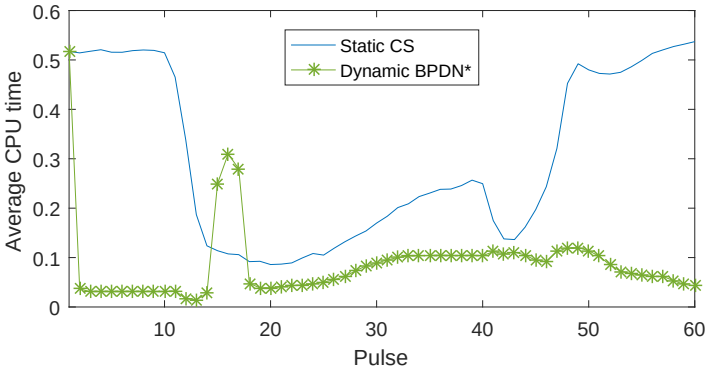


Figure 5.12: Average CPU times in scenario 3

5.3 HPFCS trigger criteria

In chapter 4 we introduced an algorithm that combines PF and CS, called HPFCS. In this section we zoom in on the trigger criterion that the HPFCS uses to determine whether it will run CS or not. Before the trigger criterion can be used in practice, a threshold has to be set in the algorithm. When setting this threshold, one has to find a suitable balance between single-target FAR (i.e. how often the criterion is met when there is in fact no mismatch) and sensitivity (i.e. how strong does the model mismatch need to be before the criterion is met). Note that the single-target FAR is not yet the FAR of the HPFCS. This will ultimately be determined by the CS algorithm. The single-target FAR only indicates how often CS is performed while there is in truth no model mismatch. Therefore, this single-target FAR can be allowed to be higher than the desired FAR of the HPFCS.

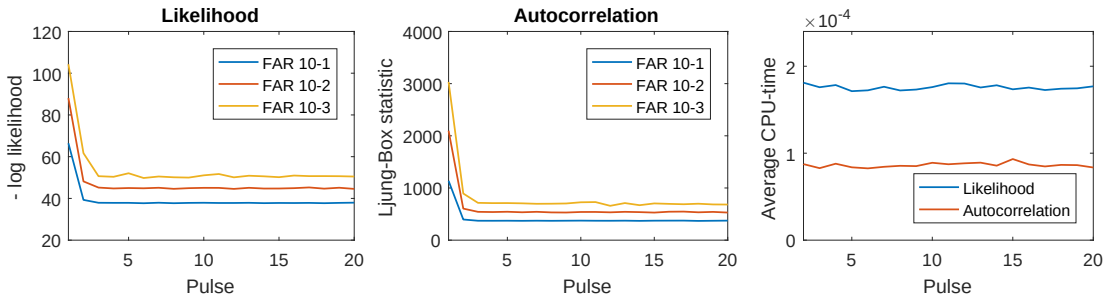


Figure 5.13: Output value and average CPU time of the trigger criteria in a single-target scenario with a single-target PF

Figure 5.13 shows the results of a single-target simulation with a single target PF. The two leftmost plots show the 90^{th}, 99^{th} and 99.9^{th} percentile of the statistics' values observed in this simulation. That is, in respectively 90%, 99% and 99.9% of the MC runs the statistic was lower than the value of the corresponding line in figure 5.13. Therefore, given a desired single-target FAR, this figure provides a threshold that will achieve it. The rightmost figure shows the average CPU-times of the computation of the two statistics.

In the two leftmost subplots of this figure we see that both statistics, the -log likelihood and the Ljung-Box statistic, start off high at the first pulse but then quickly converge to a constant level. This can be explained by the fact that the PF starts with an initial distribution that is relatively wide with respect to the likelihood. Table 5.1 shows the thresholds that are extracted that way.

Table 5.1: Thresholds corresponding to different single-target FARs

Single-target FAR	10^{-1}	10^{-2}
Likelihood threshold	37.8	44.7
Ljung-Box threshold	370	530

The rightmost plot of figure 5.13 shows that the computation of the Ljung-Box statistic required less CPU-time than the computation of the -log likelihood. More specifically, the computation of the Ljung-Box statistic is roughly two times as fast. Since both statistics use the residual, the difference between their CPU-times is made in the remaining calculation. For the likelihood criterion, this calculation consists of a multiplication of the complex conjugate of the residual, the covariance matrix and the residual itself. For the Ljung-Box statistic, a sum of m scalar products is calculated for each of the 10 lags. Since in this case m is significantly larger than 10, it makes sense that the computation of the Ljung-Box statistic is faster.
It should be noted that the covariance matrix is an identity matrix in this case, so that multiplication by it does not change anything and it can therefore be left out. If that is done, the computation of the likelihood criterion is likely to be faster than the numerical results suggest. Another thing to take into account is that the autocorrelation criterion requires the noise to be uncorrelated. Therefore, in the case of correlated measurements, these measurements will have to be prewhitened, which should be taken into account when analyzing the computational resources used by the autocorrelation criterion.

Next, the thresholds from table 5.1 are applied in scenario 3 (i.e. two targets on top of each other for 10 pulses, then one target moving away from the other for the remaining 50 pulses), with a single target PF. As the targets move further apart the model mismatch becomes stronger, since it becomes clearer that there are two targets. Figure 5.14 shows the fraction of MC runs where the trigger criterion exceeded the threshold per pulse.

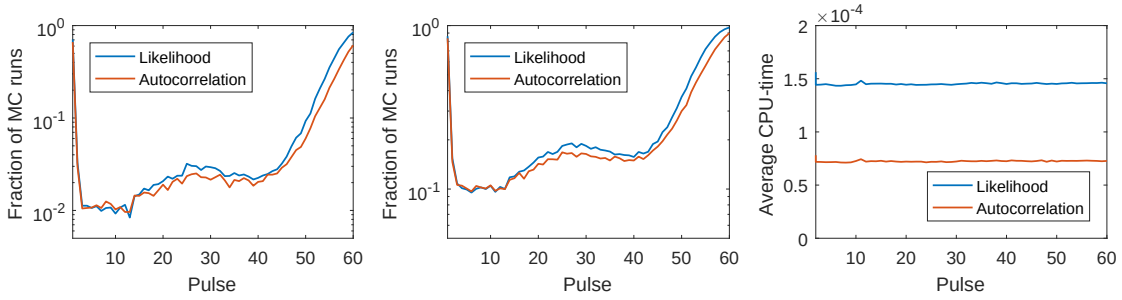


Figure 5.14: Fraction of MC runs where the trigger criterion was met for a single-target FAR of 10^{-2} and 10^{-1}

As expected, this fraction is approximately 1% and 10% during the first 10 pulses, corresponding to the single-target FARs. After that, the fraction increases for both statistics, where the likelihood criterion has a slightly higher fraction than the autocorrelation criterion. Since there is indeed a model mismatch after pulse 10, we would like to perform CS to (attempt to) fix that. Therefore, based on these fractions, the likelihood criterion is more sensitive than the autocorrelation. Therefore, if computational costs are not taken into account, the likelihood criterion is preferred over the autocorrelation criterion.

Which one of these two criteria is preferred in practice depends largely on the situation at hand. In the case of uncorrelated measurements, the computation of the likelihood criterion is lower than that of the current implementation of the autocorrelation criterion. However, in this case the multiplication by the covariance matrix in the computation of the likelihood could actually be left out altogether.
In the case of correlated measurements it depends on whether the signal has been prewhitened or not. If the signal has to be prewhitened specially for computing the Ljung-Box statistic, the computation costs of this statistic increase significantly. However, if CS is performed afterwards, the signal is preferably prewhitened for that anyway. On the other hand, if the signal has been prewhitened the covariance can be left out of the computation of the likelihood criterion. Therefore, if it is optimized for the situation at hand, the likelihood criterion will require less computational resources than the autocorrelation criterion.

To summarize: the likelihood criterion seems to be slightly more sensitive than the autocorrelation criterion. Moreover, while the numerical results that were presented here suggest the opposite, we expect the autocorrelation criterion to require more computational resources than the likelihood criterion if the implementation is optimized.

5.4 Towards a proper comparison of PF, HPFCS and dynamic CS

As mentioned in the introduction, the goal of this project was to determine whether there was something to gain by using CS in a tracking algorithm. While some steps towards this goal were presented in this report, it is not yet clear if there is indeed something to gain. As also mentioned earlier, we think that to determine whether there is, one should compare the algorithms introduced in this report to a multi-target PF. In this section we discuss a number of points of attention for future research towards that goal.

5.4.1 Limitations of presented numerical results

The numerical results discussed in sections 5.2 and 5.3 were obtained in simulations of a certain scenario (described in subsection 2.1.2). This scenario has some properties that limit the extent to which conclusions from these numerical results may be generalized. These properties are discussed in this subsection.

Targets of equal amplitude The scenarios that were used to compare the different types of Dynamic CS both have two targets with equal amplitudes. Provided each of the individual targets still has a sufficient SNR, there does not seem to be an intuitive reason why Dynamic CS would deal badly with targets with unequal amplitudes.

Known noise level In this report, the noise level was assumed to be known. The standard deviation of the noise is used to set the weight of signal fidelity in BPDN and in the likelihood function of the PF. The assumption that the noise level is known is common in the context of radar signal processing. A possible way to achieve this in practice is to collect (a large number of) measurements in a scene that has no targets.

Empirical run-time As discussed earlier in this report, the comparison of empirical run-times has its drawbacks. For the different types of Dynamic CS this is not an issue, since all are based on the same implementation of YALL1. For the trigger criteria however, their respective implementations might influence the run-time significantly. It is difficult to say if either of the implementations can be dramatically improved.

5.4.2 Assumptions

The next point of attention is the set of assumptions. In particular: the set of assumptions of the PF is less restrictive than that of Dynamic CS and the HPFCS in a number of aspects. Future researchers should be aware that every difference between the sets of assumptions will weaken the conclusion that can be drawn from the comparison.
For example, if Dynamic CS and a multi-target PF are compared in a scenario with linear measurements and white Gaussian noise, the conclusion could be that Dynamic CS is more efficient. However, Dynamic CS makes use of the assumptions of linearity and Gaussianity while the PF does not. If it would, the PF would reduce to the Kalman filter, which is optimal in the linear Gaussian case. Therefore it can be argued that the PF should not have been applied in this scenario in the first place. Vice versa, if the same two algorithms are compared in a nonlinear, non-Gaussian scenario, it can be argued that Dynamic CS should not have been used, since its assumptions do not hold. Therefore we discuss the differences in the sets of assumptions of the three methods in this section and indicate possibilities for dealing with these differences. For brevity we refer to the algorithms that use the CS framework (i.e. Dynamic CS and the HPFCS) as just 'CS' in this section.

Nonlinear measurements The CS algorithms that were used during this project all assume that the measurements are a linear function of the state and measurement noise. In a PF however, the measurement model may be nonlinear. While a large majority of papers related to CS focus on linear measurements, there are variations which do allow for nonlinear measurements, which are briefly introduced below.
Ohlsson *et al.* [31] discuss the case where the measurement model is quadratic and argue that a second-order Taylor approximation of the measurement model suffices in many (though not all) situations. Furthermore, they extend the classical CS framework for these quadratic measurements and develop an algorithm to solve the corresponding optimization problem: Quadratic Basis Pursuit. As its name suggests, this is closely related to the BP and BPDN problems that were introduced in section 1.2.

Blumensath [5] shows that the Iterative Hard Thresholding (IHT) can be used with nonlinear measurements. Furthermore, he introduces an adaption of the theory that underlies the guarantees that the CS framework offers for the case of nonlinear measurements. A comparison of three nonlinear optimization algorithms with a constraint on sparsity is presented by Beck and Eldar [3].

Uncorrelated noise The PF can deal with correlated noise through the likelihood: if the covariance matrix of the noise is known, the likelihood function can incorporate this without much effort. As discussed in section 1.3, the measurements can be prewhitened so that the assumption of uncorrelated noise holds once again. Doing so is not strictly necessary to run a CS algorithm, but the assumption of uncorrelated (Gaussian) noise is used in many derivations of recovery guarantees.

Off-grid targets As discussed in subsection 3.1.3, we assumed targets to be on grid during this project. A possible way to deal with off-grid targets is discussed in the work of Bekers *et al.* [4]. They use a combination of pruning, clustering and interpolation: the responses that exceed a certain threshold are clustered, where each cluster corresponds to a target. Then the largest response in the cluster and its maximum neighbor are interpolated to arrive at an estimate of the location of this target.

Prior knowledge on cardinality The multi-target PF described in this report uses prior knowledge about the number of targets (i.e. the cardinality is between $R_{\min}$ and $R_{\max}$) - prior information that is not incorporated in CS. One way to deal with this is to incorporate this constraint into CS, for example by replacing the BPDN problem (equation (1.4)) by

$$\hat{x} = \underset{x \in \mathbb{C}^n}{\operatorname{argmin}} \{ \frac{1}{2\rho} \|Ax - y\|_2^2 \text{ s.t. } R_{\min} \leq \|x\|_0 \leq R_{\max} \}. \tag{5.2}$$

However, it seems inevitable that the optimization algorithm would then always end up with $\hat{R} = R_{\max}$. Furthermore, this problem formulation is not easily translated to a convex relaxation such as BPDN. Another option would be to adjust the multi-target PF algorithm so that it does not require a range of cardinalities as an input. Which of these two options is favorable depends on the situation at hand (i.e. whether prior information on the cardinality is available).

Assuming that the original signal is sparse, as is done in CS, is also a form of providing prior information. In particular, in a situation where two hypotheses about the number of targets are equally likely according to the posterior distribution estimated with a PF, this prior information would lead us to prefer the smallest of the two. This prior information could be integrated into a PF via its prior distribution.

Prior knowledge on the state Both PF and (algorithms using) CS require prior knowledge on the state. For a PF the initial distribution should cover the truth and for CS the truth should be covered by the grid. In the context of the problem description of this project that means that they both require a bandwidth for frequency: CS to place its grid, PF to place its initial distribution. However, to guarantee that the initial distribution covers the truth, the PF also needs such a bandwidth for amplitude. A rough estimate of the amplitude could be enough to set such a bandwidth. But that estimation should then be included in the overall analysis.

5.4.3 Measures of algorithm efficiency

Another important point of attention is the measure of algorithm efficiency will be used. Such a measure will have to be taken into account in a comparison between CS and PF, since if it is not included it can be argued that the PF can be used with an arbitrarily large number of particles and will therefore be arbitrarily close to being optimal from the Bayesian perspective. The variants of Dynamic CS and the HPFCS are clearly further from optimal when it comes to integration of

information over time: Dynamic CS only passes on a small part of the available information on to the next timestep and the HPFCS does not integrate information over time for any of the hypotheses other than the one used at the current timestep. Therefore the only way to beat the PF is on algorithm efficiency: the PF is only asymptotically optimal. The choice of the measure of efficiency should be considered carefully, since an empirical measure will most likely depend on many external factors, as discussed in section 2.3. A possible way to avoid relying only on an empirical measure of algorithm efficiency and the issues that come with it, is discussed appendix D

5.5 Extensions and alternatives

Since the goal of finding ways to make use of CS in a dynamic scene was too broad for this project, we restricted ourselves to finding out whether there is something to gain by using CS in a tracking algorithm. That is still an open problem: in a situation where computational resources are limited, Dynamic CS or the HPFCS might be more efficient than a PF. However, this is not the only way to make use of CS in a dynamic scene. What is more, since the PF has been proven to be optimal with unrestricted resources one could expect it not to be too far from optimal in a situation with restricted (but still reasonable) amount of resources. If this is true, then if there is something to be gained by using CS in a tracking algorithm, it will not be much. Alternative ways to make use of CS in dynamic scenes were not explored much during this project but we advise future researchers to do so.

5.5.1 Interplay between PF and CS in the HPFCS

In the HPFCS described in section , after the trigger criterion is met, the output of CS is used to determine how the PF should continue at the next timestep. So far it was only mentioned that the PF will use the cardinality that is estimated by CS. If that is indeed the only information that is provided to the 'new' PF, it will have to start with only its initial distribution $p(s_0)$: an uniform distribution spanning the bandwidth in the context of this report. It seems there might be ways to do better, since there are at least two sources of information available: the estimated posterior from the PF and the state estimate of CS. In this section we discuss these sources of information and how they could be used in the HPFCS.

Estimated posterior from the PF In figure 5.15 we see two rows corresponding to two (hypothetical) situations where the estimated number of targets by the algorithm might change, from 1 on the left to 2 on the right.

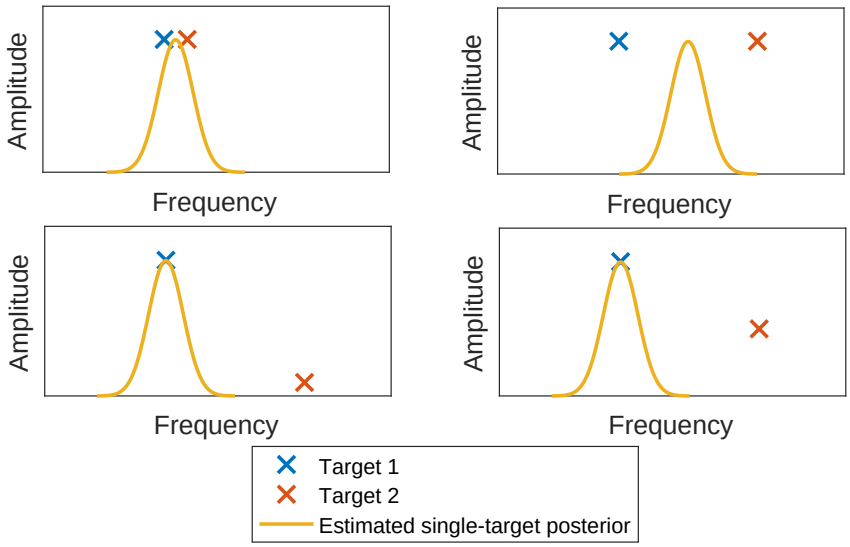


Figure 5.15: Two examples of situations where the trigger might go off.

In the situation at the top there are two targets of equal amplitude which are moving apart. In this case, the single-target PF is likely to place its estimate between the two targets. In the situation in the bottom row the single-target converged on the larger target while the other target increases in strength. In both situations the trigger criterion might be satisfied so that CS can identify that there are two targets instead of one. In the second situation there is useful information in the estimated posterior, since this has converged on the larger target. This estimated posterior contains information from previous pulses as well, so that it is likely to provide an accurate estimate of the state of this target. In the first situation however, it is probably better to ignore the estimated posterior altogether, since the information it contains is wrong.

State estimate of CS If CS is performed and a new PF is started, the initial distribution of the 'new' PF could be based on the state estimate of CS. To do so, the point estimate of CS has to be converted to a probability distribution. What kind of probability distribution that should be is not immediately clear. One way to proceed is to determine what type of probability distribution can be used to describe the error of the state estimated by CS. A distribution of this type can then be used as the initial distribution of the 'new' PF.

5.5.2 Alternative PF/Bayesian filter implementations

The main reason for using the SIR PF during this project was its simplicity. Future researchers should be aware that there are known issues with the SIR, which are alleviated by using a different implementation of a PF. For example, the SIR is known to have issues with depletion when the likelihood function is narrow compared to the prior distribution. That might happen e.g. when the SNR is high. There are other PF algorithms that deal with such situations more efficiently. The same holds for situations where the dimension of the state vector is high. These algorithms were outside the scope of this project, but we do mention here Markov Chain Monte Carlo (MCMC) algorithms, which offer a possible solution to the aforementioned issues with the SIR. Another aspect of the implementation of a multi-target PF is pruning the hypotheses. The multi-target PF algorithm described in this report has a fixed range of cardinalities, for each of which a PF is executed. An alternative to this is to prune the filters that have an unlikely cardinality and/or start up PFs with new cardinalities if necessary.

5.5.3 Alternatives to convex optimization

During this project, only the approach that uses convex relaxation (i.e. replacing the ℓ^0-norm by the ℓ^1-norm) is investigated. As stated before, this relaxation was necessary since the problem (1.2) is combinatorial. However, there are ways to solve that problem without relaxing the ℓ^1-norm. Here we briefly introduce some of these alternatives.

One of these approaches is to use greedy algorithms. Many of these are based on Matching Pursuit (MP), introduced by Mallat and Zhang in [27]. The intuition behind MP is to iteratively choose components from a dictionary to greedily reduce the approximation error. At each iteration we choose the component that allows for the greatest reduction in approximation error. This is achieved by choosing the dictionary entry with the largest inner product with the signal. The approximation of the signal using only this one dictionary entry is then subtracted from the signal. This process is repeated until some convergence criterion is satisfied.
A popular extension is Orthogonal Matching Pursuit (OMP) [33, 39], which can lead to better convergence than MP at the cost of increased computational complexity. This is achieved by projecting the signal onto the set of dictionary entries selected so far and updating all the coefficients extracted accordingly. As a result, the residual will be orthogonal to the approximation of the signal after each iteration, while this is not guaranteed in MP. The application of OMP to compressive measurements is discussed in e.g. [29]. Intuitively, a greedy approach might do well in situations like the one considered in this report (i.e. one or two on-grid targets in a small bandwidth with a high SNR), since the algorithm might find the two grid-points with a target quite quickly.

Ji *et al.* [20] consider the recovering of the original signal from compressive measurements from the Bayesian perspective. Whereas the other algorithms that were discussed here all provide a point-estimate of the sparse state x, Bayesian Compressive Sensing provides an estimated posterior density. With this, instead of providing only a point estimate of the reconstructed signal, BCS can also provide what they refer to as "error bars". As discussed in subsection 5.5.1 , one of the challenges in combining CS with a PF is that CS only provides a point-estimate, while a PF requires an initial distribution. Therefore, some kind of transition has to be included. With Bayesian CS however, its output could be used as the initial distribution for the PF directly. This would make the interplay between CS and a PF more natural.

5.5.4 Alternative procedures for determining Dynamic BPDN+ and Dynamic BPDN* weights

The procedures for determining the weights in Dynamic BPDN+ and Dynamic BPDN* are motivated by intuition and the rule of thumb that the locations of targets with a large estimated amplitude are found more accurately. While these procedures might work in certain situations, their current presentation lacks a mathematical foundation. Since we are dealing with uncertainties, an alternative procedure for setting the weights is to connect the weights of a frequency bin directly to the estimated probability that a target will be in this bin. In Dynamic BPDN* this is done to some extent, by using the shape of the distribution of the target locations based on the estimated target locations in the previous timestep. However, the value of the weight does not have a direct connection with the probability that a target is present in this bin, only with the estimated amplitude at the previous timestep.
A possible starting point for such a connection between probability and weights could be the work of Khajehnejad *et al.*, who describe a relationship between weights and a priori probabilistic information. They do so for the case where the weights can come from one of two sets, where the probability of an entry being nonzero is different for each set. They note that "while it is in principle possible to extend [their] techniques to models with more than two categories of [weights], the analysis becomes increasingly tedious ...". This suggests that their techniques could be extended to the case where the weights are linked to the estimated state of the previous timestep propagated through a dynamic

model.

5.5.5 Adaptive CS

In this report we only considered using a constant sensing matrix in the methods that use the CS framework. However, performance can be improved by adjusting the sensing matrix to the situation at hand. This is shown by e.g. Haupt *et al.* [18], who propose a sequential procedure for signal support recovery from noisy measurements. The signal of interest is assumed to be sparse and constant over time, but a different sensing matrix is used for each measurement. In this approach the dimension of the search space is reduced over time. In other words: the sensing effort is focused on dimensions where components are likely to be present. It is shown that with this adaptive approach, much weaker signals' support can be recovered than with non-adaptive approaches. While they only consider a signal of interest that is constant over time, it might be worthwhile to extend this approach to dynamic scenes.

6. Conclusions and recommendations

At the beginning of this report, we set out to explore ways to make use of CS in a dynamic scene. The specific task we considered was tracking multiple targets, which we hoped to do more efficiently than existing algorithms (Bayesian filters) in terms of computational resources, by making use of CS. To do so, we discussed, tested, and proposed variations for two approaches: Dynamic CS and HPFCS.

Of the variants of Dynamic CS, only Dynamic BPDN* did better than static CS in certain ways, which shows that not every method of transferring prior knowledge to the next timestep in a CS algorithm is suitable. Dynamic BPDN* did outperform static CS for at least one of the scenarios considered, where it achieved the same accuracy as static CS, but using less computational resources. In the other two scenarios the lower CPU-time of Dynamic BPDN* came at the cost of extra false alarms. From this we conclude that it is possible to gain something with respect to static CS by using (a variant of) Dynamic CS. There are still open questions and research opportunities for this approach, which indicated that devising an appropriate dynamic CS algorithm and proper tuning is no straightforward task.

The other approach that was considered, is to combine CS with a Bayesian filter, such as a PF. We described the HPFCS, which uses a PF to track the targets over time and CS to update the cardinality of this PF when a trigger criterion is met. The numerical results concerning this trigger criterion show that in the specific situation considered, the autocorrelation trigger criterion proposed in this report requires less computational resources than the likelihood criterion proposed in the literature. On the other hand, the likelihood criterion was also found to be slightly more sensitive, which could be an advantage. Furthermore, we have proposed a change in the implementation of the likelihood criterion that could improve its computational resources. Whether these conclusions will also be valid in practical situations needs verification with measurement data.

If the next goal is to determine whether there is something to gain by using an algorithm like HPFCS or a variant of Dynamic CS for tracking, a comparison to a suitable Bayesian filter is inevitable. In the case of nonlinear measurements and correlated, possibly non-Gaussian noise, this suitable filter would be a PF. Therefore we provide directions toward a proper comparison of the algorithms discussed in this report and a PF in sections 5.4 and 5.5. These directions show that there are many changes that need to be made to the algorithms presented in this report before such a comparison is proper. However, all of them concern techniques that are already known, so that performing these changes is a matter of implementation.

For now, the question whether there is something to gain in terms of computational resources with respect to a PF by using an algorithm that either combines CS with a PF or uses only CS, is still open. Considering the results in this report, we think HPFCS has the potential to improve on more classical PF implementations in terms of computational resources. Future developments in Dynamic CS need a thorough look at the choice of CS algorithm. As described in section 5.5, types of CS other than solving BPDN using YALL1, such as Bayesian CS, may shed another light on our

conclusions regarding Dynamic CS. Additionally, any conclusion about a comparison of algorithms based on computational resources will be weakened by the fact that these resources depend on many factors other than the algorithm itself. Furthermore, we think that if there is anything to gain, these gains will be small. After all, the class of Bayesian filters will always be (asymptotically) optimal if computational resources are not taken into account.

Bibliography

[1] H. Akaike. A New Look at the Statistical Model Identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[2] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[3] A. Beck and Y. C. Eldar. Sparsity Constrained Nonlinear Optimization: Optimality Conditions and Algorithms. *Society for Industrial and Applied Mathematics*, 23(3):1480–1509, 2013.

[4] D. Bekers, W. Rossum, B. Jacobs, M. Heiligers, L. Anitori, E. Stolp, L. Cifola, and M. Podt. On Pre-Whitening and Accuracy in DOA Estimation by Sparse Signal Processing on Beamformed Data. *4th Int. Workshop on Compressed Sensing on Radar, Sonar, and Remote Sensing (CoSeRa)*, pages 202–206, 2016.

[5] T. Blumensath. Compressed Sensing with Nonlinear Observation and Related Nonlinear Optimization Problems. *IEEE Transactions on Information Theory*, 59(6), 2013.

[6] Y. Boers. Particle Filters en Niet-Lineair Schatten. Technical report, Thales Nederland.

[7] M. Born and E. Wolf. *Principles of Optics*. Cambridge University Press, 1999.

[8] R. V. Borries, C. J. Miosso, C. Potes, and E. Paso. Compressed Sensing Using Prior Information. *2nd IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing*, pages 121–124, 2007.

[9] K. Burnham and D. Anderson. *Model Selection and Multimodel Inference (2nd edition)*. Springer, 2008.

[10] E. J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[11] J. V. Candy. *Bayesian Signal Processing*. John Wiley & Sons, Inc., 2009.

[12] S. Das and N. Vaswani. Particle Filtered Modified Compressive Sensing (PaFiMoCS) for tracking signal sequences. *Conference Record - Asilomar Conference on Signals, Systems and Computers*, 2(2):354–358, 2010.

[13] D. L. Donoho. For most large underdetermined systems of linear equations the minimal l1-norm solution is also the spartest solution. *Comm. on Pure and Applied Math*, 59(6):797–829, 2006.

[14] D. L. Donoho, J. M. Santos, and J. M. Pauly. Compressed Sensing MRI. *IEEE Signal Processing Magazine*, (March 2008):72–82, 2008.

[15] N. Doornekamp. Internship report 'Compressive Sensing in Dynamic Scenes'. Technical report, Thales Hengelo, 2016.

[16] M. F. Duarte, M. A. Davenport, D. Takhar, J. N. Laska, T. Sun, K. F. Kelly, and R. G. Baraniuk. Single-Pixel Imaging via Compressive Sampling. *IEEE Signal Processing Magazine*, 25(2):83–91, 2008.

[17] M. P. Friedlander, H. Mansour, R. Saab, and Ö. Yilmaz. Recovering compressively sampled signals using partial support information. *IEEE Transactions on Information Theory*, 58(2):1122–1134, 2012.

[18] J. Haupt, R. Baraniuk, R. Castro, and R. Nowak. Sequentially designed compressed sensing. *2012 IEEE Statistical Signal Processing Workshop, SSP 2012*, pages 401–404, 2012.

[19] C. M. Hurvich and C.-L. Tsai. Regression and Time Series Model Selection in Small Samples. *Biometrika*, 76(2):297, 1989.

[20] S. Ji, Y. Xue, and L. Carin. Bayesian Compressive Sensing papers. *IEEE Transactions on Signal Processing*, 56(6):2346–2356, 2008.

[21] S. Kay. *Fundamentals of Statistical Signal Processing, Volume 1: Estimation Theory*. Prentice Hall, 1993.

[22] M. A. Khajehnejad, W. Xu, A. S. Avestimehr, and B. Hassibi. Weighted l1 minimization for sparse recovery with prior information. *IEEE International Symposium on Information Theory - Proceedings*, pages 483–487, 2009.

[23] C. Kreucher, K. Kastella, and A. O. Hero III. Tracking Multiple Targets Using a Particle Filter Representation of the Joint Multitarget Probability Density. *IEEE Transations on Aerospace and Eletronic Systems (AES)*, 2004.

[24] S. Kullback and R. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[25] G. M. Ljung and G. E. P. Box. On a Measure of Lack of Fit in Time Series Models. *Biometrika*, 65(2):297, 1978.

[26] W. Lu and N. Vaswani. Regularized Modified BPDN for Noisy Sparse Reconstruction with Partial Erroneous Support and Signal Value Knowledge. *IEEE Transactions on Signal Processing*, 60(1):182–196, 2012.

[27] S. G. Mallat and Z. Zhang. Matching Pursuits With Time-Frequency Dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

[28] V. Maroulas and P. Stinis. Improved particle filters for multi-target tracking. *Journal of Computational Physics*, 231(2):602–611, 2012.

[29] D. Needell and J. A. Tropp. CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301–321, 2009.

[30] F. Ning, D. Gao, J. Niu, and J. Wei. Combining compressive sensing with particle filter for tracking moving wideband sound sources. *2015 IEEE International Conference on Signal Processing, Communications and Computing, ICSPCC 2015*, 2015.

[31] H. Ohlsson, S. S. Sastry, A. Y. Yang, R. Dong, and M. Verhaegen. Quadratic Basis Pursuit. Technical report, 2012.

[32] H. Ohlsson, M. Verhaegen, and S. S. Sastry. Nonlinear compressive particle filtering. *Proceedings of the IEEE Conference on Decision and Control*, (c):7054–7059, 2013.

[33] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pages 1–5, 1993.

[34] S. Saha, Y. Boers, H. Driessen, P. Mandal, and a. Bagchi. Particle based MAP state estimation: A comparison. *2009 12th International Conference on Information Fusion*, pages 278–283, 2009.

[35] D. Schuhmacher, B. T. Vo, and B. N. Vo. A consistent metric for performance evaluation of multi-object filters. *IEEE Transactions on Signal Processing*, 56(8 I):3447–3457, 2008.

[36] S. S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, 2008.

[37] D. J. Spiegelhalter, N. G. Best, B. P. Carlin, and A. van der Linde. Bayesian Measures of Model Complexity anf Fit. *Journal of the Royal Statistical Society Series B (Statistical Methodology)*, 64(4):583–639, 2002.

[38] D. J. Spiegelhalter, N. G. Best, B. P. Carlin, and A. Van der Linde. The deviance information criterion: 12 years on. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 76(3):485–493, 2014.

[39] J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, 2007.

[40] N. Vaswani and W. Lu. Modified-CS : Modifying Compressive Sensing for Problems with Partially Known Support. *IEEE Transactions on Signal Processing*, 58(9):1–12, 2010.

[41] J. Yang and Y. Zhang. Alternating direction algorithms for compressive sensing. *Society for Industrial and Applied Mathematics*, 33(1):250–278, 2011.

[42] Y. Zhang, W. Deng, J. Yang, and W. Yin. YALL1: Your ALgorithms for L1, http://yall1.blogs.rice.edu.

[43] Y. Zhang, J. Yang, and W. Yin. User's Guide for YALL1 : Your ALgorithms for L1 Optimization. *CAAM Technical report TR09-17*, 2009.

[44] X. Zhu. *Very High Resolution Tomographic SAR Inversion for Urban Infrastructure Monitoring*. PhD thesis, Technischen Universität München, 2011.

Appendices

A. Information Criteria

In the multi-target PF described in section 4.1.2, the task of the Information Criterion (IC) is to decide which of the PFs running in parallel is using the most plausible model (i.e. the most plausible number of components). Since the performance according to the measures described in chapter 2 is determined based on a point-estimate only and for a large part by the decision whether a single-target or a dual-target model is preferred, the task of the IC is of great importance. In this section we aim to first give an intuitive motivation of this task, after which the underlying theoretical basis, the Kullback-Leibler Distance [24] (denoted KLD, also sometimes referred to as Kullback-Leibler information, discrepancy, divergence or number instead of distance) is briefly introduced. A more elaborate discussion and the corresponding mathematical derivations of the 'quick and dirty' introduction of the KLD in this section can be found in e.g. chapter 2 of Burnham and Anderson [9].

The intuitive motivation for the use of an IC is that usually the fit of the model to the data becomes better as more parameters are estimated by a model. The more components may be used to describe the observed signal, the better the description will be. Therefore, without penalizing the introduction of extra parameters, the best model would in many cases be the one with the larger number of estimated parameters. This idea is easily understood in the context of regression: If we have n random data points from the two-dimensional plane, fitting a quadratic function is going to provide a better fit than a linear function and a cubic function will be better than a quadratic function, etc. And in the extreme, using a polynomial curve of degree $n-1$ can be found so that it fits the data perfectly. Therefore, when choosing the order of a model, looking only at how well it fits the data will always favor a model of higher order.

The theoretical basis for (most of the) the ICs that are discussed in this section is the KLD. The KLD of a candidate model with respect to the truth is a measure of 'how much information is lost when the model is used instead of the truth'. Naturally, models that minimize the loss of information are preferred: we want to select the candidate model with the minimal KLD to the truth, so that the lower the KLD of a candidate model, the better. The problem with this objective is that the truth is unknown in the situations of interest. It turns out however that the KLD can still be used to compare candidate models of the same truth used since, as Burnham and Anderson [9] phrase it "the truth drops out as a constant". In other words: it is possible to compute the KLD$-C$, where C is a constant that depends only on the truth. Since this truth is the same for every candidate model, the quantity KLD$-C$ can be computed and used to rank the models.

To compute the KLD of a given model, the parameters of the model must be set. These parameters are of course unknown, so that they have to be estimated. Therefore the KLD of the model is also an estimate and instead of minimizing the true KLD, the IC is actually minimizing *expected* KLD [9]. The challenge in estimating the expected KLD is in correcting for the bias that is introduced by evaluating the model using the data that were also used to fit its parameters. As is discussed in the next section, there are multiple ways of estimating this bias.

A.1 Comparison

As discussed in chapter 2, the performance of the multi-target PF will be measured by its ability to correctly identify whether it is dealing with a single-target or dual-target scene. Therefore the IC has a huge influence on the performance: the IC decides at whether the single-target or the dual-target model is preferred. In this section a number of criteria are listed, which all aim to estimate the KLD but differ in their estimate of the aforementioned bias term.

Akaike Information Criterion The Akaike[1] Information Criterion (AIC) [1] of a model is defined as

$$AIC = 2k - 2\log(\hat{L}), \tag{A.1}$$

where k is the number of parameters that are estimated by the model and $\hat{L}$ is the likelihood of this model with maximum likelihood parameters. The AIC has been shown to become biased towards more complex models as the sample size becomes large relative to the number of parameters [19]. In other words: the AIC tends to overfit when the sample size is small. Therefore a second-order bias correction is introduced in the corrected AIC, denoted AICc:

$$AICc = AIC + \frac{2k(k+1)}{n-k-1}. \tag{A.2}$$

As a rule of thumb, Burnham and Anderson [9] propose to use AICc instead of AIC when $\frac{n}{k} < 40$. Since the AICc converges to the AIC as n becomes large with respect to k, it could also be argued to always use AICc over AIC: the difference between the two is only significant in situations where you would prefer the AICc over the AIC according to this rule of thumb.

Deviance Information Criterion The Deviance Information Criterion (DIC) was introduced by Spiegelhalter *et al.* [37]. In the DIC the constant penalty on the number of parameters in AIC is replaced by a data-dependent bias correction. Intuitively this makes sense, since parameters that are heavily constrained by prior information do not constitute a 'degree of freedom'. Therefore instead of simply counting the number of parameters, the *effective* number of parameters is estimated. This effective number of parameters is then used as penalty term. Note that in the case of an uninformative prior or no prior information at all, the number parameters is equal to the effective number of parameters. While it is quite popular (over 8000 citations according to Google Scholar), there are some known issues with the DIC. This was also recognized by their original authors, who discuss the limitations in [38].

Bayesian Information Criterion Whereas the AIC, AICc and the DIC all aim to estimate the KLD, the Bayesian Information Criterion (BIC) is derived from a Bayesian framework instead [9]. The BIC is defined as

$$BIC = \log(n)k - 2\log(\hat{L}), \tag{A.3}$$

where n is the number of data points. The difference between AIC and BIC is in the weight of the penalty term k, which is 2 in the AIC and $\log(n)$ in the BIC.

[1] Akaike originally intended the 'A' in AIC to mean 'an', as in 'an information criterion' [1], but it is common practice to refer to the AIC as the Akaike Information Criterion.

A.2 Conclusion: Selecting an IC

With the properties of the ICs described in subsection A.1 in mind, one of the criteria has to be chosen. To make this choice we consider the following:

- The prior information in the problem described in chapter 2 is the bandwidth in which all targets are assumed to be. This is a uniform distribution, which is wide relative to the likelihood functions that were observed. Therefore, in the light of choosing an IC, we could assume this prior information to be uninformative. As a result, the number of parameters in a model is equal to the effective number of parameters, so that it does not need to be estimated. Therefore methods that do encompass such an estimation (such as DIC) will not be needed.
- The sample size is quite small with respect to the number of parameters. As discussed in chapter 5: $n = 32$, while k is either 2 or 4 and therefore AICc is preferred over AIC.

With these considerations in mind, the choice between AICc and BIC remains. There does not seem to be an obvious winner; the use of either can be justified. The difference between BIC and AICc is in the penalty term, which is larger for the BIC. Therefore, when the two prefer different answers, the model suggested by the BIC will have less parameters than the model suggested by AICc. In other words: when a low false-alarm rate is important the BIC will do better, when a high detection probability is important the AICc is. Because emphasis in the performance evaluation is on identifying the second target, and therefore less on avoiding false alarms, we choose to use the AICc.

It should be noted that besides the criteria discussed in this report, there is what Spiegelhalter *et al.* [38] describe as a "bewildering alphabet of information criteria: BPIC, CIC, EIC, FIC, NIC, TIC, ...". But since there do not seem to be any reasons not to use AICc or BIC in the context of this project, they will not be included in our comparison. One reason to look at other criteria would be that all the criteria discussed in this section use a point estimate to characterize the posterior estimated by the model. This is not a problem if the estimated posterior is symmetric and unimodal as in our example, but might cause problems otherwise. Criteria such as the Watanabe-Akaike IC (WAIC) can deal with such situations.

B. Influence of the quality of the initial distribution in a hybrid multi-target PF

During this project, a multi-target PF different from the one described in section 4.1.2 was investigated. In this multi-target PF the number of components hypothesized by a particle can change over time, according to a transition matrix. There are a number of fundamental issues with this approach, in particular with the transition matrix. The dynamic model (which includes this transition matrix in this case) is supposed to be a model of reality and therefore this transition matrix should have some kind of physical interpretation. With this transition matrix however, it is not clear what kind of probabilities should be in it. At the same time, these probabilities will have a significant impact on the overall performance. If one wishes to model a situation where targets might appear and/or disappear, a better way to do so is to include target births and deaths in the model. This approach is not discussed in this report.

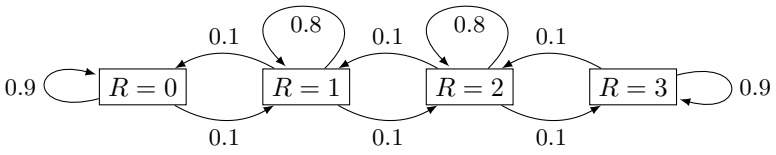
The motivation for investigating the multi-target PF described above was to find out whether it is useful to make an effort to obtain information that can be included in the PF during the initialization. In particular, if we make an extra effort to determine the number of targets and their positions at the first timestep, for example by using a CS algorithm, how much do we gain by that? Since the results that were obtained with the PF using a transition matrix might still offer some insights into (the initialization of) multi-target PFs, we will discuss this PF and some numerical results in this appendix. In particular, section B.1 describes the state and state transition model in this PF and section B.2 discusses the initial distribution that it uses. Then, before the results are presented in section B.4, the performance measure and the Cramer-Rao bound for this measure is discussed in section B.4.

B.1 Description of state and state transition distribution

The state of the system at time k is $s_k = [A_k^1, F_k^1, ..., A_k^R, F_k^R]$, where R is the true number of targets. Meanwhile the state of a particle $s_k^i = [A_k^{1,(i)}, F_k^{1,(i)}, ..., A_k^{\hat{R}_k^i,(i)}, F_k^{\hat{R}_k^i,(i)}]$, where $\hat{R}_k^i$ is number of targets hypothesized by this particle. In other words, a particle represents a hypothesis about the multi-target state, which consists of the number of targets and their respective amplitudes and frequencies. This estimated number of targets changes according to a Markov chain with a symmetric transition matrix and a maximum number of targets of 3:

The transition distribution, which in the SIR particle filter that was used for this project also serves as the importance distribution, can be described in words by the following rules:

- The number of targets hypothesized by particle i at time step k (denoted $\hat{R}_k^i$) is drawn from the Markov chain described in figure B.1.
- If $\hat{R}_{k-1}^i > \hat{R}_k^i$ one of the $\hat{R}_{k-1}^i$ targets is removed at random and the rest is propagate through the dynamic model (described in (4.1)). Here we distinguish two situations:



$$\begin{pmatrix} 0.9 & 0.1 & 0 & 0 \\ 0.1 & 0.8 & 0.1 & 0 \\ 0 & 0.1 & 0.8 & 0.1 \\ 0 & 0 & 0.1 & 0.9 \end{pmatrix} \tag{B.1}$$

Figure B.1: Transition model of the number of targets

  - If $\hat{R}^i_{k-1} = 2$ and $\hat{R}^i_k = 1$, the state of the particle at time k will be $\left[A^1_{k-1} + A^2_{k-1}, \frac{F^1_{k-1}+F^2_{k-1}}{2}\right]$ propagated through the dynamic model.
  - Otherwise the state of the particle at time k will be the state of the particle at time $k-1$ with the randomly selected target removed, propagated through the dynamic model

- If $\hat{R}^i_{k-1} = \hat{R}^i_k$ the number of targets remains unchanged and all targets are propagated through the dynamic model.
- If $\hat{R}^i_{k-1} < \hat{R}^i_k$ an extra target is drawn from $p(s_0)$. All targets, including the new one, are then propagated through the dynamic model.

B.2 Initial distribution

The initial distribution is defined as follows. For each particle $s(i)$ at time zero

$$s_0^{(i)} \sim \mathcal{U}(s_0 + \epsilon - 3\sigma_{s_0}, s_0 + \epsilon + 3\sigma_{s_0}), \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon). \tag{B.2}$$

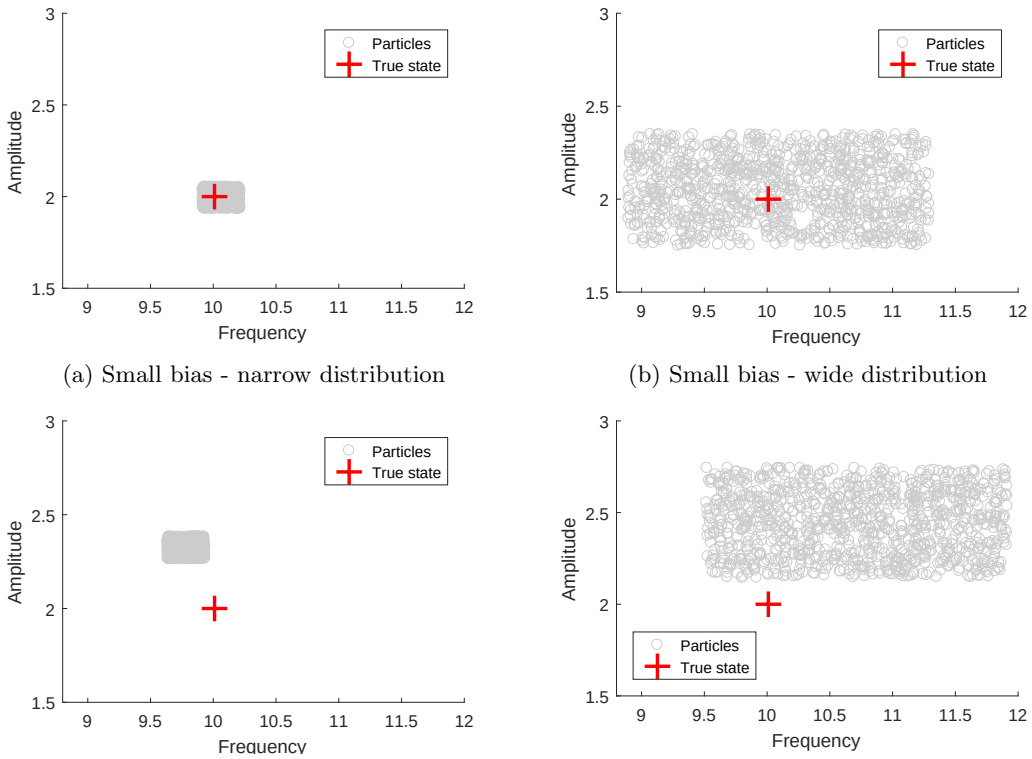
That is: the initial distribution is uniform around $s_0 + \epsilon$, where ϵ is a normally distributed bias, which is the same for all particles. This bias is introduced to simulate the error that an algorithm that is used to obtain an initial distribution inevitably makes. The larger the error of this algorithm, the larger the bias in the initial distribution. Note that in the following both s_0 and ϵ are usually multi-dimensional vectors. This artificial initial distriubtution is illustrated in figure B.2, where the magnitude of the bias corresponds to σ_ϵ and the width of the inital distribution corresponds to σ_{s_0}. Figure B.2 shows two situations (subfigures B.2c and B.2d) where the initial distribution does not cover the true state. This might occur when the error of the point estimate from the algorithm that is used to obtain the initial distribution is larger than expected, so that it is large with respect to the (chosen) width of the distribution. Even though the particle filter might still converge to the true state since the distance to the true state might eventually be covered by the process noise, this situation should be avoided.

In the multi-target case, the particles are first distributed over the hypotheses about the number of targets. In the following two ways to do this are discussed: uniformly over the possible hypotheses and 'clairvoyantly' (i.e. all particles are initialized with the correct hypothesis about the cardinality).

B.3 Performance evaluation

In this section we summarize the result from the internship report [15] concerning the Cramer-Rao bound (CRB). A more elaborate discussion of the CRB can be found in e.g. the work of Kay [21].

To evaluate the performance in the case of a single target we will look at the Mean Squared Error (MSE). The CRB is a lower bound on the variance of an unbiased estimator. Therefore, when comparing MSE to the CRB, it is assumed that the estimator is unbiased. In this subsection we state the CRB for the estimation of a vector of parameters with complex Gaussian noise. The CRB is defined via the Fisher Information matrix $I(\theta)$:



(a) Small bias - narrow distribution

(b) Small bias - wide distribution

(c) Larger bias - narrow distribution

(d) Large bias - wide distribution

Figure B.2: Examples illustrating the possible combinations of error magnitude and width of the initial distribution

$$[I(\theta)]_{i,j} = 2Re\left(\left[\frac{\partial f}{\partial \theta_i}\right]^H C^{-1}\left[\frac{\partial f}{\partial \theta_j}\right]\right) + Tr\left(C^{-1}\frac{\partial C}{\partial \theta_i}C^{-1}\frac{\partial C}{\partial \theta_j}\right). \tag{B.3}$$

Here $Re(z)$ denotes the real part of complex number z and $Tr(A)$ is the trace of matrix A. According to the CRB, the covariance matrix of the estimated θ satisfies

$$C_{\hat{\theta}} - I^{-1}(\theta) \geq 0. \tag{B.4}$$

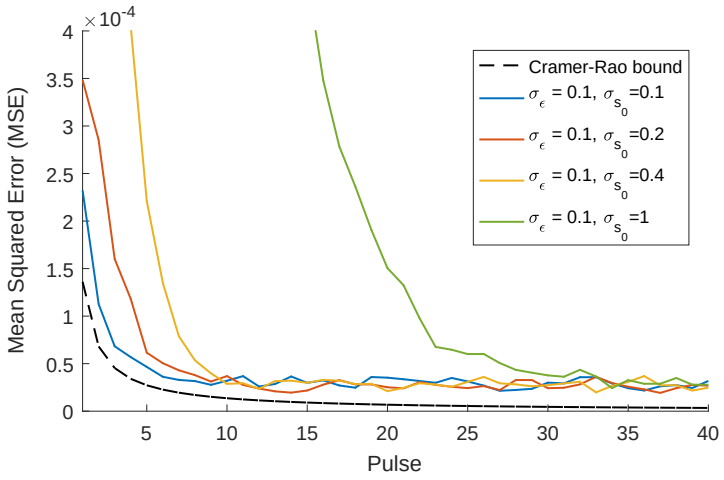
That is, the matrix $C_{\hat{\theta}} - I^{-1}(\theta)$ is positive semi-definite. This equation also provides lower bounds on the variances of the separate θ_i:

$$var(\theta_i) = [C_{\hat{\theta}}]_{i,i} \geq [I^{-1}(\theta)]_{i,i} \tag{B.5}$$

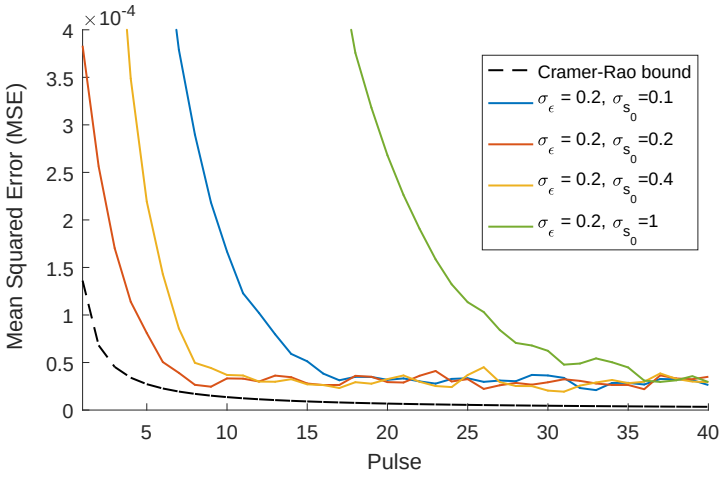
B.4 Numerical results and analysis

B.4.1 Single target

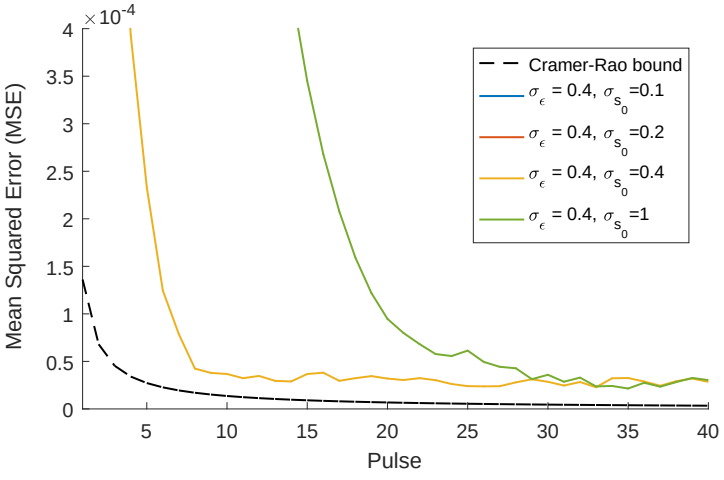
Figure B.3 shows the MSE for each number of pulses for different error standard deviations (denoted σ_ϵ, as in section B.2). In this figure, each line represents a width of the initial distribution (denoted σ_{s_0}).



(a) $\sigma_\epsilon = 0.1$



(b) $\sigma_\epsilon = 0.2$



(c) $\sigma_\epsilon = 0.4$

Figure B.3: MSE per number of pulses for varying σ_ϵ and σ_{s_0}

In figure B.3c the lines for $\sigma_{s_0} = 0.1$ and $\sigma_{s_0} = 0.2$ are outside the chosen window. Based on this figure we can make the following observations:

- For a large σ_ϵ in combination with a small σ_{s_0}, the MSE does not (always) converge. In particular: if the initial distribution is
- When the MSE does converge, it always converges to the same level. That is, the quality of the initial distribution, provided it is good enough to let the filter converge at all, does not affect the precision after convergence. It only determines the number of timesteps required to get to that level.
- The fastest convergence is achieved when σ_{s_0} is equal to σ_ϵ. If $\sigma_{s_0} > \sigma_\epsilon$ the initial distribution will cover the true state, so that the PF will converge. But the larger σ_{s_0}, the longer this takes. If $\sigma_{s_0} < \sigma_\epsilon$ however, the true state could be outside the initial distribution. If the distance to the true state is not too large, the PF will still converge, but if it is too far all particles would have a likelihood of practically zero. In that case the PF will not converge at all.

B.4.2 Multiple targets

In all the multi-target PF results, σ_{s_0} is proportional to σ_ϵ.

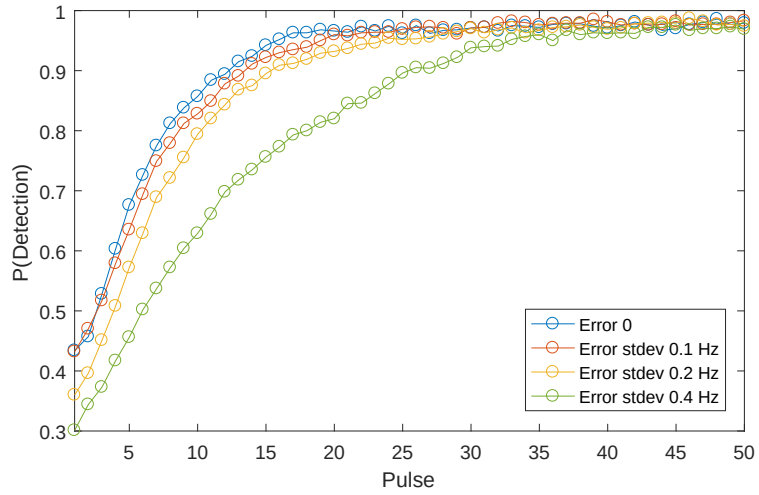
In figure B.4 we see the detection probabilities for two targets, 0.07 Hz apart. For figure B.4a a uniform initial distribution over the number of targets was used, for figure B.4b all particles started with the (correct) hypothesis that there were two targets. Each line represents a different error standard deviation (σ_ϵ in equation (B.2)). Based on this figure B.4 we can make the following observations:

- Since we have coupled σ_{s_0} to σ_ϵ, the larger σ_ϵ, the lower the detection probability in the first pulses.
- Like for the single-target case, the error in the initial distribution does not affect the level to which the detection probability converges over time.
- With a perfect initial distribution, the detection probability starts off high but drops a little in the first pulses. That is caused by the perfect initial distribution: all particles start with the (correct) hypothesis of two targets.

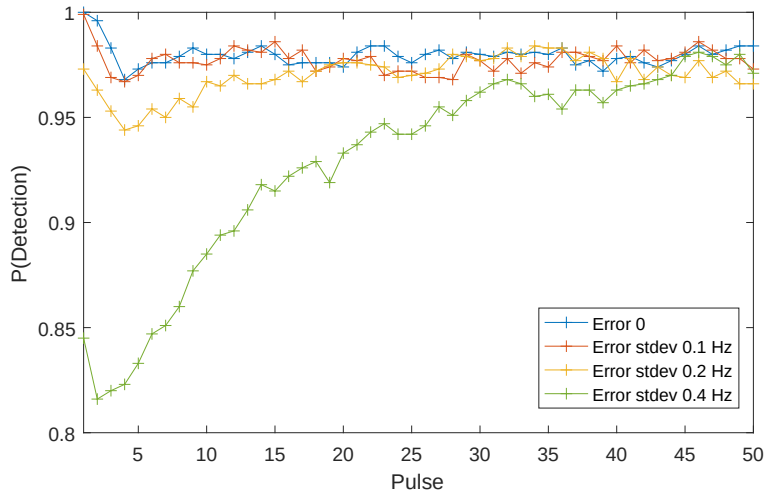
If we combine lines from these two figures, as in B.5, we can make an additional observation: a perfect initial distribution of the number of targets gives a higher detection probability than a uniform initial distribution over the number of targets in the first pulses even if the latter has no error in the targets' location while the former has a large error in the target's location.

Figure B.6 shows a comparison of perfect initial distributions over the number of targets (denoted with a '+') and uniform initial distributions over the number of targets (denoted with a 'o'). Based on this figure we can observe that for a given distance between targets, the perfect and uniform initial distributions over the number of targets eventually result in the same detection probability after a number of pulses.

From these results we conclude that when trying to improve the initialization of the multi-target PF considered in this appendix, it is more important to provide the PF with a correct estimate of the number of targets than to provide a precise location.



(a) Uniform initial distribution



(b) Perfect initial distribution

Figure B.4: Detection probabilities for different initial distributions over the number of targets (distance between targets 0.07 Hz)

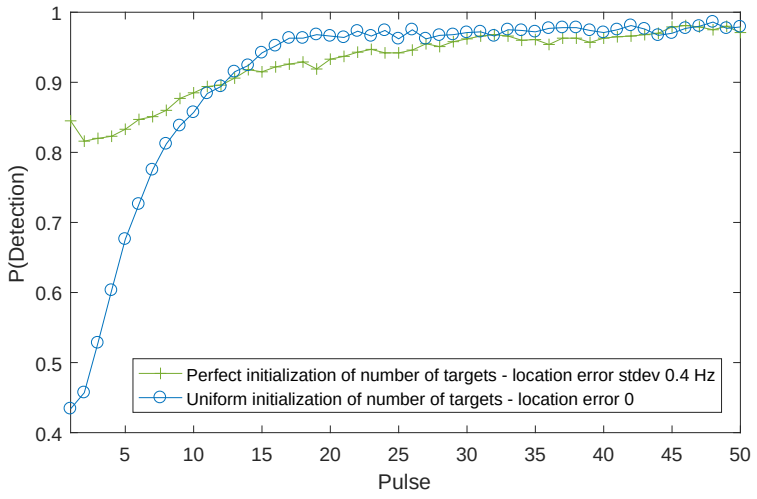


Figure B.5: Comparison of a perfect initial distribution over the number of targets with $\sigma_\epsilon = 0.4$ to a uniform distribution over the number of targets with $\sigma_\epsilon = 0$

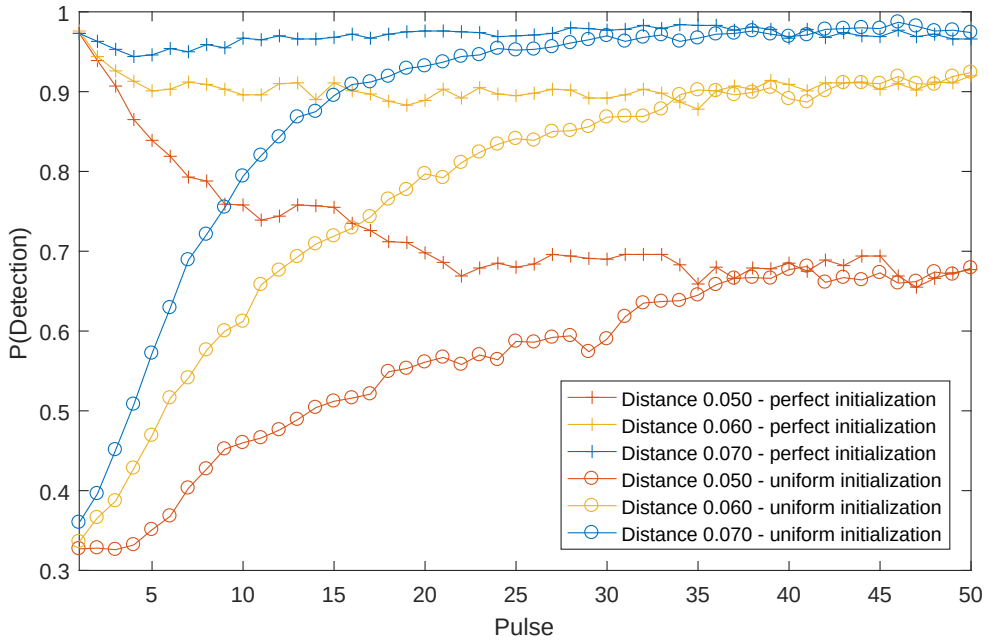


Figure B.6: Comparison of different intial distributions over the number of targets for different distances between targets, $\sigma_\epsilon = 0.2$

C. Analysis of single-target Sequential Bayesian framework implementations

In the first phase of this project we investigated several implementations of the sequential Bayesian framework. The one that is most important for this project, the particle filter, was already discussed in quite some detail. This appendix describes the implementations that are not essential in this report but might still help to gain insight in the framework and thereby the particle filter.

Deterministic dynamic model

During the internship that preceded this graduation project we have shown that (sequential) Maximum Likelihood Estimation (MLE) is a special case of the sequential Bayesian framework [15]. This case is obtained when a deterministic dynamic model (i.e. the process noise $\omega_k = 0$ in equation (4.1)) is assumed, the initial distribution $p(s_0)$ is uniform and the point estimate is extracted by selecting the grid point with the maximum likelihood. Equivalently, the sequential MLE can be obtained from the SIR particle filter by taking a non-adaptive grid (i.e. no resampling and a deterministic dynamic model) and extracting a point estimate by selecting the maximum likelihood particle. The differences between the two become even smaller when the sequential MLE uses a weighted average to obtain its point estimate. Doing so alleviates some of the issues caused by the 'grid-basedness'[1]. In figure C.1 the sequential MLE with a weighted average as point estimate and sequential MLE with the most likely gridpoint as point estimate a SIR PF. The number of grid points was equal to the number of particles: 500.

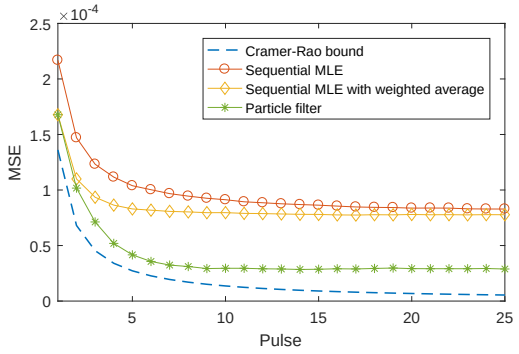


Figure C.1: MSE of sequential MLE with and without weighted average and a PF - 500 particles/grid points

In figure C.1 we see that the adaptive grid of the PF allows it to converge to a lower MSE than the sequential MLE with or without a weighted average.

[1] That is, in sequential MLE where the point estimate is the most likely grid point, the estimation error is bounded from below by the distance between the true state and the closest grid point. When a weighted average is used it is possible to get closer.

Mismatched deterministic dynamic model

If a deterministic dynamic model is assumed, but this assumption turns out to be wrong, the dynamic model is mismatched. The figure below shows the MSE of sequential MLE and non-sequential MLE for an example of such a case. More specifically: the frequency linearly drifts away from the deterministic dynamics:

$$\begin{aligned} \text{Dynamic model:} \quad & s_k = f(s_{k-1}) \\ \text{Truth:} \quad & s_k = f(s_{k-1}) + \text{drift} \end{aligned} \tag{C.1}$$

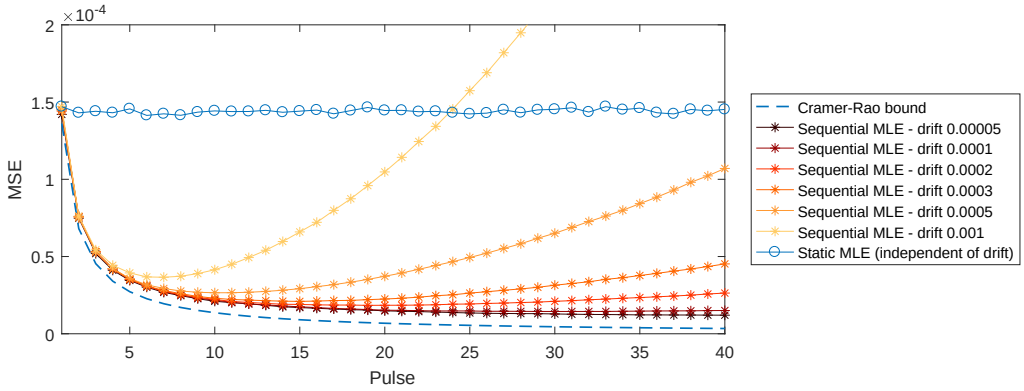


Figure C.2: MSE of sequential MLE and non-sequential MLE for various amounts of drift

If one encounters a situation like this, a possible fix could be to choose a fixed number of pulses as memory. E.g. in figure C.2, if we expect the drift with respect to the deterministic dynamic model to be less than 0.001 we could limit the history that is used to compute the sequential MLE to 5 pulses.

D. Theoretical growth orders of memory and run-time

In section 2.3 it was explained that the empirical run-time, which is the measure of algorithm efficiency that was used during this project, has the problem that it is influenced by many factors outside the algorithm itself. This problem can be avoided by using a more theoretical approach to analyzing the efficiency of an algorithm: the memory- and run-time growth orders of an algorithm. These growth orders tell us how the algorithms' memory usage and run-time will scale as the inputs grow in size. This is independent of the outside factors that have a great influence on practical measures: it only depends on the algorithm, which is the main advantage of this method. Growth orders of algorithms are commonly denoted using the Big-$\mathcal{O}$ notation. This notation will not be explained here; readers new to it could take a look at e.g. Chapter 2 of [36], which offers a very readable introduction to it. The most common use of the Big-$\mathcal{O}$ notation is with inputs that depend on one parameter. For example, multiplying two vectors with n entries has a run-time of $\mathcal{O}(n)$, multiplying an $n \times n$ matrix with a vector of length n has $\mathcal{O}(n^2)$, etc. The run-time can also depend on multiple parameters. For example, multiplying an $n \times m$ matrix by an $m \times p$ matrix has run-time $\mathcal{O}(nmp)$. For analyzing the algorithms used during this project the growth orders will be expressed in terms of the input sizes n and m.

An important drawback of growth orders is that they are not necessarily useful when the input sizes are still relatively small. For example, an algorithm that runs in $T(n) = 0.001n^3 = \mathcal{O}(n^3)$ will run faster than an algorithm that runs in $T(n) = 1000n^2 = \mathcal{O}(n^2)$ for any $n < 10^6$. However, when for example $n = 10^{12}$ the constants in the $T(n)$'s in this example will be negligible and the second algorithm is much faster than the first. Therefore describing an algorithms efficiency using growth orders does not need to give an accurate indication of the efficiency in practice, but will be increasingly accurate as the input sizes grow. When comparing the growth orders of algorithms we will look at what happens between the moment the algorithm receives its input (y) and the moment it provides its output ($\hat{s}$), since that is the part where the algorithms might differ. In the rest of this subsection we analyze the theoretical growth orders of the memory and run-time of a PF, Dynamic CS and the HPFCS.

PF

The memory usage of a PF scales linearly with the number of particles: the state of each of the particles and their weights have to be saved explicitly. Moreover, this is the case for each of the PFs that is used in parallel. These are all constant multiplicative factors of the growth order of memory and will therefore not impact the growth order. The memory usage of a PF does not depend directly on the size of the bandwidth, but it is likely that more particles are needed as the bandwidth increases in size. The only factor that depends on the size of the input is the storage of the measurement, which has memory growth of order $\mathcal{O}(m)$. However, the memory used by the PF is likely to be dominated by storing the particle cloud in practice, since each of the particles has to be saved explicitly.

Like for the memory, the order is affected by the size of the measurements m, as in $y \in \mathcal{C}^m$: when computing the likelihood of a particle (which is done through equation (4.10)), each of the m entries of the measurement is compared to the signal corresponding to the state as hypothesized by this particle. This is done for each particle, so in practice the number of particles plays the most important role for the run-time. The overall run-time growth order of the PF is $\mathcal{O}(N_p m)$. Furthermore, if the covariance matrix has to be taken into account because of correlated measurements, the computation of the likelihood involves the matrix-vector multiplication of the residual and the covariance matrix. This would increase the run-time to $\mathcal{O}(N_p m^2)$.

Dynamic CS

Since the post-processing of the YALL1 output and determining the Mod-BPDN weights is simple in terms of computational costs, the memory and run-time of Dynamic CS is dominated by the YALL1 iterations. For the order of memory growth of YALL1 it should be noted that Ψ and Φ (and therefore also A), do not need to be stored as full matrices in all situations. For example, when Ψ is the Fourier basis, storing the full matrix in memory can be avoided by specifying the parameters of the transformation that multiplying by Ψ effectively performs. In the case where the rows of Φ are a subset of the rows of an identity matrix, only the m indices of these rows have to be saved. This is important for the memory usage of the algorithms that use CS, since the memory usage of these algorithms increases from $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$ when all these matrices are saved in full. Because of the iterative character of YALL1, the memory usage does not increase over the number of pulses or iterations.

The run-time is largely determined by the iterations of the ADM. Like with the order of memory growth, an important aspect of the order of run-time growth is the matrix A. In each of the N_{iter} iterations of the YALL1 algorithm that is used for this project, a number of matrix-vector multiplications is performed, which has a run-time of order $\mathcal{O}(nm)$. And again it might be possible to avoid this matrix-vector multiplication by defining an operation to replace it. For the example where Φ is a subset of the rows of an identity matrix, the multiplication could be replaced by simply selecting the entries of the vector that correspond to the nonzero elements of Φ. With the matrix-vector multiplication, the growth order of the run-time of YALL1 is of order $\mathcal{O}(N_{\text{iter}} nm)$.

HPFCS

Since the HPFCS is a combination of CS and PF, it inherits their run-time and memory growth orders. As discussed in chapter 4, an important advantage of the HPFCS is that it does not use multiple PFs in parallel, as would be the case for a 'PF-only' solution. Another important factor in the run-time of the HPFCS is how often CS is executed. This depends on the sensitivity of the trigger criterion. Again, there is a trade-off: the more often CS is executed, the less likely it is to miss an opportunity where CS could have identified the second target but the more expensive the algorithm as a whole becomes. To determine the order of run-time growth however, this factor does not play a role. The situation where CS is never executed can be disregarded since then the HPFCS will never identify the second target. Therefore the order of run-time growth does not depend on how often CS is executed. Instead it depends only on the run-times of the PF and CS, where the order of run-time growth is the maximum of the two: $\mathcal{O}(\max(n, m)) = \mathcal{O}(n)$.

The two different trigger criteria both only make use of the residual, which makes their memory growth order $\mathcal{O}(m)$. Both perform a number of multiplications with these residuals, making both their run-time growth orders $\mathcal{O}(m)$. However, the Ljung-Box statistic calculates the sample autocorrelation for each of the h lags, each of which has run-time of order $\mathcal{O}(m)$. The likelihood criterion requires only one such computation, but does involve the multiplication by the covariance matrix,

which increases its run-time order to $\mathcal{O}(m^2)$. Note that the autocorrelation criterion requires the signal to be uncorrelated or whitened. This should be taken into account when investigating the computational resources used by this criterion: if the autocorrelation criterion is used the signal has to be prewhitened at every timestep (before the Ljung-Box statistic is calculated). While in a situation where the likelihood criterion is used, prewhitening is only necessary when CS is actually performed.