The Rotating-Target Algorithm for Jointly Detecting Asynchronous Tracks

Elnaz Banan Sadeghian and John R. Barry, *Senior Member, IEEE*

***Abstract*— Two-dimensional magnetic recording promises to increase areal density through the joint detection of multiple tracks of interest. This paper concerns the problem of joint detection of multiple tracks that are written asynchronously, meaning that neither the bit boundaries (phase) nor the bit rate (frequency) are aligned between neighboring tracks. We propose the rotating-target algorithm for jointly detecting multiple asynchronous tracks from one or more readback waveforms. The proposed approach is based on the joint Viterbi algorithm and a time-varying target that results when the asynchrony of the tracks is absorbed into the underlying target. Timing estimation for the tracks being detected is embedded inside the joint Viterbi detector using per-survivor processing. Performance results show that the proposed algorithm closely matches the performance of a fictitious system in which neighboring tracks are synchronous, and further that it significantly outperforms a previously reported detector that separately detects the two tracks.**

***Index Terms*— Synchronization, multitrack detection, intertrack interference (ITI), per-survivor processing (PSP), two-dimensional magnetic recording (TDMR), multiple-input multiple-output (MIMO).**

I. Introduction

FIRST-GENERATION implementations of two-dimensional magnetic recording (TDMR) use multiple readers to recover the bits from only a single track of interest [1]. The synchronization problem in this single-track setting is straightforward, since off-the-shelf one-dimensional strategies based on a phase-locked loop (PLL) [2] or interpolative timing recovery [3], [4] can be applied after the multiple-input single-output equalizer front end. In this case the equalizer outputs are synchronized to the track of interest, regardless of the timing offsets of the interfering tracks [5]. The result is an instance of modular design, in which the functions of synchronization and detection are implemented separately.

To realize the full potential of TDMR, however, future implementations will jointly detect *multiple* tracks of interest using a *joint* or *muiltitrack* detector [6], which drastically changes the synchronization problem. The problem is illustrated in Fig. 1 for the special case of two tracks of interest having different bit periods, and two overlapping readers. The core issue is the impossibility of being synchronous to both tracks simultaneously: being synchronous to one necessarily implies being asynchronous to the other. The implication is that synchronization and detection can no longer be performed separately, but instead must be performed jointly.

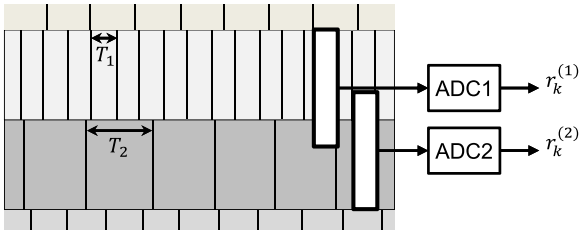

Fig. 1. An example of two tracks of interest whose timing differ in frequency and phase, and two readers with significant overlap.

In this paper we present the *rotating-target (ROTAR)* algorithm as a solution to the problem of jointly detecting multiple asynchronous tracks from multiple readback waveforms. The ROTAR algorithm modifies a joint Viterbi detector to have a time-varying target that accounts for the asynchrony of the tracks being detected. The proposed algorithm uses per-survivor processing (PSP) to estimate the timing offsets of the tracks of interest, a generalization of the per-survivor timing recovery scheme developed for one-dimensional magnetic recording [7].

II. Channel Model and Assumptions

We consider the problem of jointly detecting K tracks from N readback waveforms. We assume a perfectly equalized partial response channel with independent timing offsets for each of the K tracks, so that the readback waveform from the i-th of N read heads is:

$$r_i(t) = \sum_{j=1}^{K} \sum_{n} a_n^{(j)} h_{i,j}(t - nT - \tau_n^{(j)}) + n_i(t) \qquad (1)$$

where $a_n^{(j)} \in \{\pm 1\}$ is the n-th bit of track $j \in \{1, \ldots K\}$, $h_{i,j}(t)$ is the bit response at reader i from track j, assumed to be bandlimited to half the bit rate, $\tau_n^{(j)} \geq 0$ is the timing offset for the n-th bit of track j, defined relative to the analog-to-digital converter (ADC) sampling period T, and $n_i(t)$ is the additive noise for the i-th read head. We assume independent white and Gaussian noise with power-spectral density $N_0/2$ for each of the read heads. The assumption that the $\{\tau_n^{(j)}\}$ be nonnegative is equivalent to an assumption that the ADC sampling rate is large enough to avoid signal aliasing.

Manuscript received May 2, 2016; revised August 16, 2016; accepted August 17, 2016. Date of publication August 29, 2016; date of current version October 11, 2016. This work was supported in part by IDEMA ASTC and in part by the National Science Foundation under Grant CNS-1513884.

The authors are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: ebsadeghian@gatech.edu; john.barry@ece.gatech.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2016.2603723

0733-8716 © 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

The i-th readback waveform is filtered by a low-pass antialiasing filter and then sampled at the ADC rate $1/T$, yielding

$$r_k^{(i)} = \sum_{j=1}^{K} \sum_n a_n^{(j)} h_{i,j}(kT - nT - \tau_n^{(j)}) + n_k^{(i)}, \qquad (2)$$

where $n_k^{(i)}$ is the k-th sample of the filtered noise $n_i(t)$, with zero mean and variance $N_0/(2T)$. Collecting the N samples from each of the N read heads at time k into the vector $\mathbf{r}_k = [r_k^{(1)}, ..., r_k^{(N)}]^T$, and using (2), we arrive at a multiple-input multiple-output (MIMO) model:

$$\mathbf{r}_k = \sum_{j=1}^{K} \sum_n a_n^{(j)} \mathbf{h}_j(kT - nT - \tau_n^{(j)}) + \mathbf{n}_k, \qquad (3)$$

where $\mathbf{h}_j(t) = [h_{1,j}(t), h_{2,j}(t), ..., h_{N,j}(t)]^T$ is the vector-valued bit response (across all N readers) for track j, and $\mathbf{n}_k = [n_k^{(1)}, n_k^{(2)}, ..., n_k^{(N)}]^T$.

III. Detection Algorithms

A. The Case of a Single Isolated Track

Before attacking the general problem of detecting multiple asynchronous tracks from multiple readback waveforms, we first examine the simpler case of detecting a single isolated track ($K = 1$) from a single readback waveform ($N = 1$) of the form $r(t) = \sum_n a_n h(t - nT - \tau_n) + n(t)$, where a_n is the n-th bit of the track, $h(t)$ is the bit response whose bandwidth is equal to half the bit rate, and τ_n is the timing offset of the n-th bit. To be concrete, we will assume a constant frequency offset here, so that $\tau_n = n\Delta T$, where ΔT is the frequency offset parameter. Sampling at the ADC rate $1/T$ yields:

$$r_k = r(kT) = \sum_n a_n h(kT - nT - \tau_n) + n_k. \qquad (4)$$

In the following we describe two strategies for implementing the maximum-likelihood (ML) sequence detector: 1) the conventional modular strategy, and 2) an alternative strategy. The latter strategy will eventually be generalized and adopted for the multiple-track scenario.

1) The Conventional Modular Strategy: The usual modular approach is to separately synchronize and detect. This is illustrated in the lower branch of Fig. 2. First, the ADC samples $\{r_k\}$ are passed to an interpolative timing recovery (ITR) block, which aims to recover the readback samples that would have arisen were the readback waveform sampled at the correct sampling times, resulting in:

$$\begin{aligned} \hat{r}_k &= r(kT + \tau_k) \\ &= \sum_n a_n h(kT - nT - \tau_n + \tau_k) + \hat{n}_k \\ &\approx \sum_n a_n h(kT - nT - \tau_k + \tau_k) + \hat{n}_k \qquad (5) \\ &= \sum_{\ell=0}^{\mu} h_\ell a_{k-\ell} + \hat{n}_k, \qquad (6) \end{aligned}$$

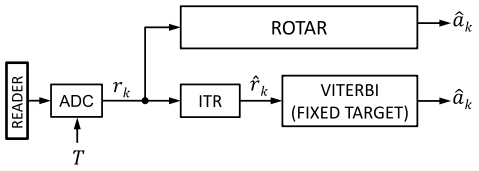

Fig. 2. Conventional modular strategy (lower branch) versus alternative strategy (upper branch) in synchronization and detection of a single isolated track.

where the approximation (with τ_n replaced by τ_k) is valid when the timing offset varies slowly enough that it is approximately constant over the duration for which the target $h(t)$ is significant, where $\{h_k = h(k(T + \Delta T))\}$ is the bit response sampled at the bit rate, and where the interpolated noise $\{\hat{n}_k\}$ has the same statistics as the original $\{n_k\}$. For convenience we assume a causal target $\mathbf{h} = [h_0, h_1, \ldots, h_\mu]^T$ with memory μ, so that $h_k = 0$ for both $k < 0$ and $k > \mu$. After synchronization, the interpolated samples of (6) may then be passed to a 2^μ-state Viterbi detector, designed for the target $\mathbf{h}$.

2) An Alternative Strategy: Rather than resampling the ADC outputs via interpolation, however, an alternative approach would be to feed them directly to a detector that internally accounts for the asynchrony, as illustrated in the upper branch of Fig. 2. (This is the approach taken by the ROTAR algorithm introduced in Sect. III-B.2.) This is possible because we can approximate the noiseless ADC output $s_k = r_k - n_k$ from (4) as the convolution of the bit sequence $\{a_k\}$ with a *time-varying* impulse response, as derived below:

$$\begin{aligned} s_k &= \sum_n a_n h(kT - nT - \tau_n) \\ &\approx \sum_n a_n h(kT - nT - \tau_k) \qquad (7) \\ &\approx \sum_{\ell=-M/2}^{\mu+M/2} h(\ell T - \tau_k) a_{k-\ell}, \qquad (8) \end{aligned}$$

where the approximation in the second line is the same as in (5). As a sanity check, the time-varying convolution in (8) reduces to the time-invariant convolution in (6) for the special case when $\tau_k = 0$ for all k, i.e., for the special case when the ADC is synchronized to the bit rate. In that case, the limits of the last sum in (8) range from $\ell = 0$ to μ. In contrast, in the general case when the ADC is not synchronized, the limits of the sum would in principle extend from $\ell = -\infty$ to ∞ for a bandlimited bit response. In practice, however, there will only be a small number of terms that contribute significantly to the sum. To account for this, we introduce a new variable M, which we assume to be even, representing the extra memory used to represent the time-varying impulse response, beyond the memory μ of the original target. The second approximation in (8) is because M is finite, and is accurate for even moderate choices of M.

An example of a time-varying target is shown in Fig. 3, assuming a frequency offset of $\tau_k = k\Delta T$ with $\Delta T/T = 2 \times 10^{-4}$, and a sector of length $L = 10^4$ bits. The extra memory in this illustration is $M = 8$. At the beginning of the sector (Fig. 3a), the resampled target is a zero-padded

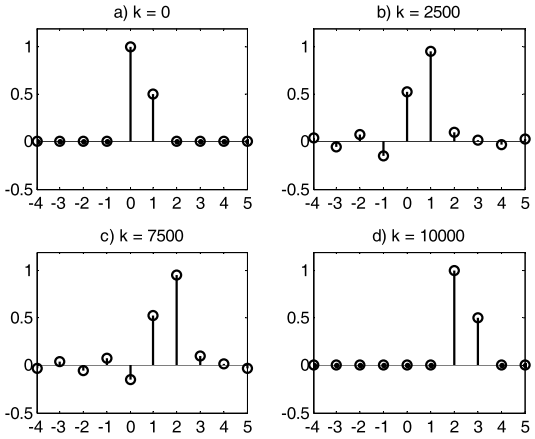

Fig. 3. An illustration of a moving target for the case of frequency offset with $\Delta T/T = 2 \times 10^{-4}$, and a sector length $L = 10^4$, assuming $M = 8$: (a) The target $\mathbf{h}[0]$ at time $k = 0$; (b) the target $\mathbf{h}[2500]$ at one quarter of the sector; (c) the target $\mathbf{h}[7500]$ at three quarters of the sector; and (d) the target $\mathbf{h}[10000]$ at the end of the sector.

version of the synchronous target $\mathbf{h} = [h_0, h_1] = [1, 0.5]$, with only two nonzero taps. As we move forward through the sector, the target drifts to the right and the number of nonzero taps increases. At one quarter of the way through the sector (Fig. 3b), the target is shifted by $\tau_k/T = 0.5$ bit periods to the right, and clearly has more than two significant taps. Likewise at three quarters of the way through the sector (Fig. 3c), where the target has shifted by $\tau_k/T = 1.5$ bit periods, there are more than two significant taps. By the end of the sector (Fig. 3d), the target has shifted by two full bit periods, and again has only two nonzero taps.

With the aid of (8), the unsynchronized ADC output may be viewed as the output of a time-varying finite-state machine with independent noise, so that the ML detector can be implemented by a $2^{\mu+M}$-state Viterbi algorithm based on the time-varying target

$$\mathbf{h}[k] = [h(-MT/2 - \tau_k), \ldots, h((\mu + M/2)T - \tau_k)]^T$$

with memory $\mu + M$. The time-varying target prevents us from precomputing the expected outputs for each state transition in the trellis; instead they must be computed anew at each stage according to the convolution in (8).

The example of Fig. 3 seems to suggest that the amount of extra memory M required to accommodate the moving target will depend on not only the severity of the frequency offset but also the length of the sector. The extra memory is a significant drawback because it increases the number of states and therefore the complexity of the detector. Fortunately there is an efficient strategy for significantly reducing the memory requirements, regardless of the frequency offset parameter and the sector length, as described in Sect. III-B.2.

B. Joint Detection of Multiple Asynchronous Tracks

Let us turn our focus back to the joint detection of K tracks from N readback waveforms. Unlike the case of the isolated track of the previous section, we can no longer separately synchronize and detect in a modular way. Nevertheless, the alternative strategy based on a time-varying target perfectly suits our purpose. Applying the time-varying convolution approximation from (8) separately to each track's contribution to reader i, the vector of readback samples from (3) can be written as:

$$\mathbf{r}_k \approx \sum_{j=1}^{K} \sum_{\ell=-M_j/2}^{\mu+M_j/2} \mathbf{h}_j(\ell T - \tau_k^{(j)}) a_{k-\ell}^{(j)} + \mathbf{n}_k, \tag{9}$$

where M_j is the extra memory parameter assigned to track j. This is a noisy output of a finite-state machine, so that the K tracks can be jointly detected using the Viterbi algorithm. The expected outputs for a transition (p, q) at time k are:

$$\mathbf{o}_k(p, q) = \sum_{j=1}^{K} \sum_{\ell=-M_j/2}^{\mu+M_j/2} a_{k-\ell}^{(j)}(p, q) \mathbf{h}_j(\ell T - \tau_k^{(j)}),$$

where $\{a_k^{(j)}(p, q)\}$ are the bits of track j on the survivor path which arrives at the transition from state p at time k to state q at time $k + 1$. The branch metric for this transition (p, q) at time k is $\gamma_k(p, q) = ||\mathbf{r}_k - \mathbf{o}_k(p, q)||^2$.

1) High Complexity: The complexity of the joint Viterbi algorithm is dominated by the number of states $\prod_{j=1}^{K} 2^{(\mu+M_j)}$. To illustrate how quickly the complexity can grow, consider the example of Fig. 1, with $K = 2$ tracks of interest and $N = 2$ readers. Suppose that the channel response in the absence of frequency offset ($\Delta T_1 = \Delta T_2 = 0$) is:

$$\mathbf{H}(D) = \begin{bmatrix} 1 + 0.5D & 0.4 + 0.16D \\ 0.4 + 0.16D & 1 + 0.5D \end{bmatrix}, \tag{10}$$

where $H^{(i,j)}(D)$ denotes the response at reader i from track j. The responses from both tracks have memory $\mu = 1$. Hence, if there were no timing offsets, the standard joint Viterbi algorithm would require 4 states. Suppose instead that $\Delta T_1/T = 2 \times 10^{-5}$ and $\Delta T_2/T = 2 \times 10^{-4}$, and that the length of the sector is $L = 10^4$ bits. Therefore, by the end of the sector, the responses of tracks 1 and 2 will shift by 0.2 bit periods and 2 bit periods, respectively. In order to capture these movements and also to include the significant taps of the moving responses, a reasonable choice for the extra memories would be $M_1 = 2$ and $M_2 = 4$, which would result in a total of $2^3 \times 2^5 = 256$ states. Consequently, compared with the case where both tracks are synchronously written, implementing the Viterbi detector for this example increases the number of states from 4 to 256. Fortunately, as described below, a more efficient implementation of the time-varying target can achieve the same performance with a significant reduction in the amount of extra memory required.

2) The ROTAR Algorithm: We have seen how frequency offset causes the time-varying impulse response to both shift and elongate as time progresses. Nevertheless, we can avoid the need for large values of the extra memory parameters $\{M_j\}$, even for extreme cases of large frequency offsets and large sector lengths, if we modify the detector to dynamically track only the significant coefficients of the time-varying impulse response. Towards that objective, let us decompose the k-th timing offset for track j into its integer and fractional parts:

$$\tau_k^{(j)} = d_k^{(j)} T + \theta_k^{(j)}, \tag{11}$$

where

$$d_k^{(j)} = \left\lfloor \tau_k^{(j)}/T \right\rfloor \in \{0, 1, 2, \dots\}$$

is the integer part, and

$$\theta_k^{(j)} = \tau_k^{(j)} - d_k^{(j)} T \in [0, T)$$

is the fractional part. With this definition, we can approximate (3) as:

$$\begin{aligned}
\mathbf{r}_k &= \sum_{j=1}^{K} \sum_{n} a_n^{(j)} \mathbf{h}_j (kT - nT - \tau_n^{(j)}) + \mathbf{n}_k \\
&\approx \sum_{j=1}^{K} \sum_{n} a_n^{(j)} \mathbf{h}_j (kT - nT - \tau_k^{(j)}) + \mathbf{n}_k \\
&= \sum_{j=1}^{K} \sum_{n} a_n^{(j)} \mathbf{h}_j (kT - nT - d_k^{(j)} T - \theta_k^{(j)}) + \mathbf{n}_k \\
&\approx \sum_{j=1}^{K} \sum_{\ell=-M_j/2}^{\mu+M_j/2} a_{k-\ell-d_k^{(j)}}^{(j)} \mathbf{h}_j (\ell T - \theta_k^{(j)}) + \mathbf{n}_k. \qquad (12)
\end{aligned}$$

The two approximations above are exactly the same as the two used in (7) and (8) for the isolated track case: The first approximation is valid when the timing offsets are approximately constant over the duration of the bit response, and the second approximation is valid for sufficiently large parameters $\{M_j\}$.

The model in (12) captures the essence of the proposed ROTAR algorithm. Here, the timing offset $\tau_k^{(j)}$ is distributed among the two sides of the convolution: The integer part acts as a *time-varying delay* on the bits, while it is only the fractional part that shifts the responses. The term *rotating target* is derived from the behavior of the target in (12) over the duration of a sector: Each time the target approaches a shift of one full bit, the delay d_k increments by one, and the response "rotates" or reverts back to its original unshifted form.

The model in (12) is also a noisy version of the output of a time-varying finite-state machine, and thus the ML detector can be implemented using, for example, the Viterbi algorithm with expected outputs for a transition (p, q) at time k computed as:

$$\mathbf{o}_k(p, q) = \sum_{j=1}^{K} \sum_{\ell=-M_j/2}^{\mu+M_j/2} a_{k-\ell-d_k^{(j)}}^{(j)}(p, q) \mathbf{h}_j (\ell T - \theta_k^{(j)}). \qquad (13)$$

Because of the time-varying fractional delay, the expected outputs will vary with time and cannot be precomputed. Furthermore, because of the time-varying integer delay of the bits, the structure of the trellis will change each time the integer delay increments.

The key advantage of the rotating property is that it enables us to use small values of the memory parameters $\{M_j\}$, and thus small overall complexity, without any performance loss. In particular, since the fractional delay parameters $\{\theta_k^{(j)}\}$ are limited to the range $[0, T)$, the vast majority of the signal energy of the delayed bit responses can be captured by choosing either $M_j = 2$ or $M_j = 4$ for the asynchronous tracks, with $M_j = 2$ being a reasonable choice for most applications; the complexity disadvantages of moving to $M_j = 4$ will likely outweigh the marginal performance advantages. Thus, the *rotating target* strategy significantly reduces the extra memory required, independent of both the severity of the frequency offset and the sector length.

The pseudocode of the proposed ROTAR algorithm is shown in Algorithm 1 and 2. Algorithm 1 calls Algorithm 2 to rearrange the states whenever an increment in the integer offset of any of the input tracks is detected. Algorithm 1 adopts the genie-aided assumption that the timing offsets of all the input tracks are known. (The case of unknown timing offsets that must be estimated is handled later, by Algorithm 3.) The inputs to Algorithm 1 are the ADC outputs, the responses from all K tracks and the original memory μ (assumed to be the same for all tracks) of the responses, the maximum of extra memories of all tracks $M_{max} = \max_j M_j$ that will extend the trellis, and the genie-aided timing offsets. The output is the set of detected bits of all K tracks.

Algorithm 1 begins by setting the initial state to state 0, in line 1 where the partial path metric of state 0 and all other states at time 0, respectively, are set to 0 and ∞. Also, the survivor path for every state p at time 0 ($\mathbf{S}_0(p)$) is initialized with an empty vector in line 2. The algorithm then proceeds to the main loop (line 3 through line 16) which steps through each stage of the trellis. The integer and the fractional offsets are computed for every track j in line 4 and line 5, respectively. Line 6 through line 15 step through all state transitions at stage k. The expected outputs and the transition metrics are computed, respectively in line 7 and line 8, for the two transitions from those states p which lead to the state q. Line 9 checks if the integer offset of any of the tracks is incremented. In case of an increment, the trellis should be rearranged and therefore line 10 calls Algorithm 2 to rearrange the partial path metrics $\{\Phi_k(p)\}$ and the survivor paths $\{\mathbf{S}_k(p)\}$ for all states p at time k. Thereafter, the algorithm continues exactly as in the standard Viterbi. The predecessor of state q at time $k+1$ is computed in line 12. The partial path metric of state q at time $k+1$ is updated in line 13. Also in line 14, the survivor path of state q is updated by concatenating (denoted as operator $|$) the survivor path of the predecessor of state q with the predecessor of state q. Finally, the estimated bits of all K tracks are extracted from the survivor path that minimizes the path metric at the end of the trellis.

The inputs to Algorithm 2 are the partial path metrics, the survivor paths, and the predecessors of all states p at time k, and also the integer bit delays of all tracks j. The outputs are the rearranged partial path metrics, and the rearranged survivor paths of all states p at time k. The algorithm begins by declaring an empty vector $collectnewstates$ in line 1. From line 2 through line 15, the algorithm finds a new state for every old state and stores it in the vector $collectnewstates$, as follows: First, in line 3, the function **de2bin** converts the decimal $oldstate$ to its corresponding binary (over alphabet $\{-1, +1\}$) vector denoted as $OLDSTATE$, so that $OLDSTATE = [OLDSTATE^{(1)}, \cdots, OLDSTATE^{(K)}]$, where $OLDSTATE^{(j)}$ denotes the part of the binary vector $OLDSTATE$ which corresponds to track j. Likewise in line 4, the predecessor of the $oldstate$ is

Algorithm 1 ROTAR With Known Timing

Inputs: ADC outputs $\{\mathbf{r}_k\}$, responses $\{\mathbf{h}_j(t)\}$, μ, M_{max}, $\tau_k^{(j)}\ \forall k, \forall j$
Output: $\{\hat{\mathbf{a}}_j\}$
1 **Init**: $\Phi_0(0) = 0$, $\Phi_0(p) = \infty\ \forall p \neq 0$
2 **Init**: $\mathbf{S}_0(p) = [\,]\ \forall p$
3 **for** $k = 0$ **to** $L + \mu + M_{max} - 1$ **do**
4 $\quad d_k^{(j)} = \left\lfloor \tau_k^{(j)}/T \right\rfloor\ \forall j$
5 $\quad \theta_k^{(j)} = \tau_k^{(j)} - d_k^{(j)} T\ \forall j$
6 $\quad$ **for** $q = 0$ **to** $Q - 1$ **do**
7 $\quad\quad$ Compute $\mathbf{o}_k(p, q)$ using (13) $\forall p \to q$
8 $\quad\quad \gamma_k(p, q) = ||\mathbf{r}_k - \mathbf{o}_k(p, q)||^2\ \forall p \to q$
9 $\quad\quad$ **if** $d_k^{(j)} \neq d_{k-1}^{(j)}\ \forall j$ **then**
10 $\quad\quad\quad (\{\Phi_k(p)\}, \{\mathbf{S}_k(p)\})$ $= \mathbf{Rearrange\ States}\,(\{\Phi_k(p)\}, \{\pi_k(p)\}, \{\mathbf{S}_k(p)\}, \{d_k^{(j)}\})$
11 $\quad\quad$ **end**
12 $\quad\quad \pi_{k+1}(q) = \underset{p}{\text{argmin}}\{\Phi_k(p) + \gamma_k(p, q)\}$
13 $\quad\quad \Phi_{k+1}(q) = \Phi_k(\pi_{k+1}(q)) + \gamma_k(\pi_{k+1}(q), q)$
14 $\quad\quad \mathbf{S}_{k+1}(q) = [\mathbf{S}_k(\pi_{k+1}(q))|\pi_{k+1}(q)]$
15 $\quad$ **end**
16 **end**
17 Extract $\{\hat{\mathbf{a}}_j\}$ from the survivor path that minimizes $\Phi_{L+\mu+M_{max}}$

converted to a binary vector denoted as $\Pi = [\Pi^{(1)}, \cdots, \Pi^{(K)}]$ where $\Pi^{(j)}$ denotes the part of the predecessor corresponding to track j. An empty vector $newstate$ is declared in line 5. From line 6 through line 13, the algorithm steps through each track: In line 7, the track with an increment in its integer offset is detected. In case of an increment, the part of the predecessor corresponding to track j should replace the part of the new state corresponding to track j. Hence $NEWSTATE^{(j)}$ is set to $\Pi^{(j)}(p)$, in line 8. Otherwise, the part of the new state corresponding to track j should be the same as the part of the old state corresponding to track j. Hence, in line 10, $NEWSTATE^{(j)}$ is set to $OLDSTATE^{(j)}$. In line 12, the binary vector $newstate$ collects every part of the new state corresponding to every track, one by one. In line 14, this vector is converted to decimal to represent the new state for the $oldstate$. Here, $collectnewstates$ collects all new states for all old states from 0 to $Q - 1$. The old partial path metrics and the old survivor paths are stored in Φ_{im} and $\mathbf{S}_{im}$, respectively in line 16 and line 17. Line 18 to line 22 determines the partial path metric and the survivor path for every $newstate$. We should note that, the mapping from every old state to its corresponding new state is not one-to-one. For example, we might find that both states 2 and 6 change to new state 3. In this case, the new state 3 takes over the old state which has the smaller partial path metric. Thus, in line 19, the indices of the duplicate mappings for every $newstate$ are found and in line 20, the smaller partial path metric is selected as the partial path metric of the $newstate$. Similarly, in line 21, the survivor path of the state with the smaller partial path metric is selected as the survivor path of the $newstate$. Finally in line 23, those new states that do not appear in the vector $collectnewstates$ are killed by setting their partial path metrics to ∞.

Algorithm 2 Rearrange States

Inputs: $\{\Phi_k(p)\}, \{\mathbf{S}_k(p)\}, \{\pi_k(p)\}, \{d_k^{(j)}\}$
Output: $\{\Phi_k(p)\}, \{\mathbf{S}_k(p)\}$
1 $collectnewstates = [\,]$
2 **for** $oldstate = 0$ **to** $Q - 1$ **do**
3 $\quad OLDSTATE = \mathbf{de2bin}\,(oldstate)$
4 $\quad \Pi = \mathbf{de2bin}\,(\pi_k(oldstate))$
5 $\quad newstate = [\,]$
6 $\quad$ **for** $j = 1$ **to** K **do**
7 $\quad\quad$ **if** $d_k^{(j)} \neq d_{k-1}^{(j)}$ **then**
8 $\quad\quad\quad NEWSTATE^{(j)} = \Pi^{(j)}$
9 $\quad\quad$ **else**
10 $\quad\quad\quad NEWSTATE^{(j)} = OLDSTATE^{(j)}$
11 $\quad\quad$ **end**
12 $\quad\quad newstate = [newstate|NEWSTATE^{(j)}]$
13 $\quad$ **end**
14 $\quad collectnewstates = [collectnewstates|$ $\mathbf{bin2dec}\,(newstate)]$
15 **end**
16 $\Phi_{im}(p) = \Phi_k(p)\ \forall p$
17 $\mathbf{S}_{im}(p) = \mathbf{S}_k(p)\ \forall p$
18 **for** $newstate = 0$ **to** $Q - 1$ **do**
19 $\quad \mathbf{ind} = \mathbf{Find}\,(collectnewstates\,(newstate) = collectnewstates)$
20 $\quad \Phi_k(newstate) = \min\,(\Phi_{im}(\mathbf{ind}))$
21 $\quad \mathbf{S}_k(newstate) = \mathbf{S}_{im}\,(\underset{ind \in \mathbf{ind}}{\text{argmin}}(\Phi_{im}(\mathbf{ind})))$
22 **end**
23 $\Phi_k(p) = \infty\ \forall p \notin collectnewstates$

To help appreciate the complexity reduction of the ROTAR algorithm let us revisit the example in Sect. III-B.1: There we saw that a high-complexity implementation of a joint Viterbi detector with a non-rotating target required the memory parameter values of $M_1 = 2$ and $M_2 = 4$, which results in 256 states. Using ROTAR, however, we can get similar performance using $M_1 = 2$ and $M_2 = 2$, which results in only $2^3 \times 2^3 = 64$ states. Moreover, since the responses from track 2 shift by two bit periods by the end of the sector, we only need to rearrange the structure of the trellis twice through the entire length of the trellis. The computational complexity of this rearrangement in Algorithm 2 is no greater than the computational complexity required to process one stage of the trellis, or equivalently one bit for each track. The extra complexity of Algorithm 2 is thus negligible in relation to the dramatic reduction in overall complexity afforded by the ROTAR algorithm.

3) Locking All ADC's to One Track: The motivation for the ROTAR algorithm stems from the simple observation that it is impossible to simultaneously synchronize the ADC's to multiple asynchronous tracks. Nevertheless, this does not mean

Algorithm 3 ROTAR With PSP

Inputs: ADC outputs $\{\mathbf{r}_k\}$, responses $\{\mathbf{h}_j(t)\}$, μ, M_{max}

Output: $\hat{\mathbf{a}}_j \ \forall j$

1 Implement line 1–line 2 from Algorithm 1

*2 **Init**: $\hat{\tau}_0^{(j)}(p) = 0 \ \forall j, \forall p$

*3 **Init**: $sum^{(j)}(p) = 0 \ \forall j, \forall p$

4 **for** $k = 0$ **to** $L + \mu + M_{max} - 1$ **do**

*5 $\quad \hat{d}_k^{(j)} = \max_p \lfloor \hat{\tau}_k^{(j)}(p)/T \rfloor \ \forall j$

*6 $\quad \hat{\theta}_k^{(j)}(p) = \hat{\tau}_k^{(j)}(p) - \hat{d}_k^{(j)} T \ \forall j, \forall p$

7 $\quad$ **for** $q = 0$ **to** $Q - 1$ **do**

8 $\quad\quad$ Compute $\hat{\mathbf{o}}_k(p, q)$ using (13) $\forall p \to q$

9 $\quad\quad$ $\gamma_k(p, q) = ||\mathbf{r}_k - \hat{\mathbf{o}}_k(p, q)||^2 \ \forall p \to q$

10 $\quad\quad$ **if** $\hat{d}_k^{(j)} \neq \hat{d}_{k-1}^{(j)} \ \forall j$ **then**

11 $\quad\quad\quad$ $(\{\Phi_k(p)\}, \{\mathbf{S}_k(p)\})$ $= \textbf{Rearrange States}(\{\Phi_k(p)\}, \{\pi_k(p)\}, \{\mathbf{S}_k(p)\}, \{\hat{d}_k^{(j)}\})$

12 $\quad\quad$ **end**

13 $\quad\quad$ Implement line 12–line 14 from Algorithm 1

*14 $\quad\quad$ $\hat{\epsilon}_k^{(j)}(q) = \sum_{i=1}^{N} r_k^{(i)} \hat{o}_{k-1}^{(i,j)}(q) - r_{k-1}^{(i)} \hat{o}_k^{(i,j)}(q) \forall j$

*15 $\quad\quad$ $\textbf{sum}(q) = \textbf{sum}(\pi_{k+1}(q)) + \hat{\boldsymbol{\epsilon}}_{k-1}(\pi_{k+1}(q))$

*16 $\quad\quad$ $\hat{\boldsymbol{\tau}}_{k+1}(q) = \hat{\boldsymbol{\tau}}_k(\pi_{k+1}(q)) + \alpha \hat{\boldsymbol{\epsilon}}_k(q) + \beta \textbf{sum}(q)$

17 $\quad$ **end**

18 **end**

19 Extract $\{\hat{\mathbf{a}}_j\}$ from the survivor path that minimizes $\Phi_{L+\mu+M_{max}}$

that we must resort to using free-running ADC's with sampling rates $1/T$ that are not matched to the bit rate of any of the tracks of interest. Instead, while it is impossible to synchronize the ADC's to multiple tracks simultaneously, we can always synchronize them to *one* of the tracks, and there is a significant complexity advantage in doing so. Synchronizing all of the ADC's to one track enables us to set the corresponding M_j parameter to zero, significantly reducing the number of trellis states. Synchronizing to one of the tracks can be implemented using a single timing-error detector for the track along with a single PLL that feeds either all of the ADC's (for a real-time implementation) or a bank of interpolative filters, one for each ADC (for a digital implementation).

4) ROTAR Algorithm With PSP: For the case when the timing offsets are not known, a timing estimation strategy should be used with ROTAR. We propose to use per-survivor processing inside ROTAR to estimate the timings [8]. The algorithm runs a separate PLL for each survivor path, so that every node in the trellis has its own estimate of the timing offsets.

The pseudocode of the proposed ROTAR algorithm with PSP is presented in Algorithm 3. The changes in Algorithm 3 compared to Algorithm 1 are marked with an asterisk. Line 1 implements line 1 and line 2 from Algorithm 1. In line 2, the estimated timing offsets for all states are initialized to zero. Also, in line 3 a variable $sum^{(j)}(p)$ is defined and initialized to zero for every state p and track j. This variable will be used later, in line 15 and line 16, in the PLL update equation. Line 4–line 17 step through each stage of the trellis. In line 5, the maximum estimated integer offset among all states p is selected as the integer offset for all tracks j. This is implemented to help PLL convergence. Line 6 computes the fractional offset for every track j and every state p. Line 7 through line 13 are similar to Algorithm 1, considering that the timings are estimated and are different for each state p. Line 13 implements line 12 – line 14 from Algorithm 1. In line 14, the estimate of the timing offset of every track j is calculated by taking a sum over the estimates which every reader i provides for track j. Here, we have used the Mueller-Muller estimate [9] to compute the error estimate from every reader i for every track j. The expected outputs $\{\hat{o}_k^{(i,j)}(q)\}$ are the expected outputs from track j to reader i on the survivor path ending at state q. These outputs are included in the already computed expected outputs of line 8 and are defined according to

$$\hat{o}_k^{(i,j)}(q) = \sum_{\ell=-M_j/2}^{\mu+M_j/2} a_{k-\ell-\hat{d}_k^{(j)}}^{(j)}(q) h_{i,j}(\ell T - \hat{\theta}_k^{(j)}(\pi_{k+1}(q))), \quad (14)$$

where $\{a_k^{(j)}(q)\}$ are the bits of track j on the survivor path ending at state q at time $k+1$. In line 15, the vector variable $\textbf{sum}(q) = [sum^{(1)}(q), ..., sum^{(K)}(q)]^T$ sums over the past values of the estimated error vector $\hat{\boldsymbol{\epsilon}}_k = [\hat{\epsilon}_k^{(1)}, ..., \hat{\epsilon}_k^{(K)}]^T$ on the survivor path which ends at state q at time $k+1$. Finally, in line 16, the estimated timing offsets of all K tracks are updated through a second-order PLL.

IV. Numerical Results

We present performance results of the ROTAR algorithm for the case of $K = 2$ asynchronous tracks with $N = 2$ readers, as illustrated in Fig. 1, where the channel model is (10). The unknown frequency offset parameters for track 1 and track 2, respectively, are $\Delta T_1/T = 2 \times 10^{-5}$ and $\Delta T_2/T = 2 \times 10^{-4}$. The sector length is $L = 40$ kbits, which results in a maximum slip of 0.8 and 8 bit periods, respectively, for track 1 and track 2 at the end of the sector. The second-order PLL parameters are $\alpha = 0.001$ and $\beta = \alpha^2/4$.

The bit-error rate performance of the proposed ROTAR algorithm with PSP is shown and compared with two other detectors in Fig. 4. The figure plots the average of the bit-error probability for the two tracks being detected, as a function of SNR. (Not shown are the individual error rates for each track, which are a close match to the average because of the symmetry in this example.)

The curve labeled "ROTAR 16" shows the performance of a 16-state ROTAR algorithm whose memory parameters are $M_1 = 0$ and $M_2 = 2$. To enable $M_1 = 0$, both ADC's are locked to track 1 using standard techniques. In particular, prior to detection a standard ITR block consisting of a SISO equalizer, a Viterbi symbol detector, a Mueller-Muller timing-error detector, a second-order PLL, and an interpolation filter

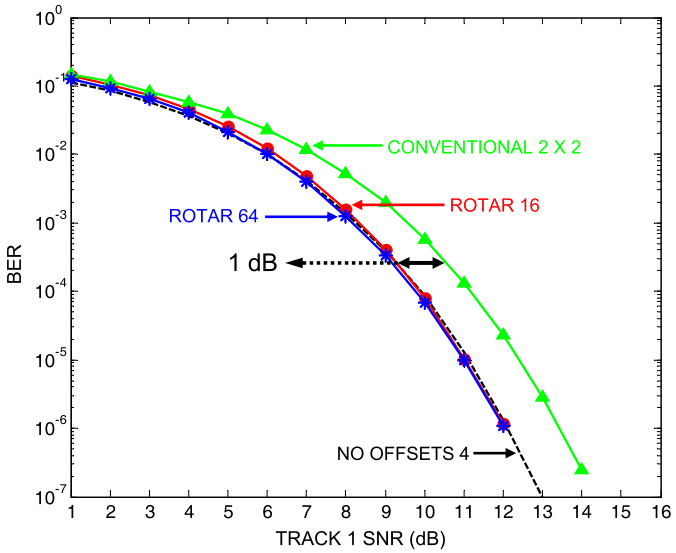

Fig. 4. BER performance of the ROTAR algorithm with PSP.

plus a secondary interpolation filter were used to lock both readback waveforms to track 1.

The curve labeled "ROTAR 64" shows the performance of a 64-state ROTAR algorithm with memory parameters $M_1 = 2$ and $M_2 = 2$, fed directly by the ADC's, without any intervening synchronizer that locks to one of the tracks.

The figure shows that the 16-state and 64-state ROTAR detectors have nearly identical performance. Considering the significant reduction in complexity, the advantage of locking to one track is clear.

Also shown in Fig. 4 is the performance of a conventional receiver that separately detects the two tracks of interest using a pair of independent two-input single-output (MISO) equalizers followed by a pair of independent one-dimensional, 2-state Viterbi detectors with PSP. The performance using this conventional approach is represented by the curve to the right with the triangle markers. We observe that ROTAR outperforms the conventional approach by 1 dB. This performance gain is due to the fact that, in the presence of ITI, joint detection is superior to one-dimensional detection.

Lastly, we also show in Fig. 4 the performance of a fictitious system for which the two tracks were written synchronously with each other and also with the ADC sampling rate. The performance of the 4-state joint Viterbi detector for this synchronous case is represented by the dashed lines. Both of the ROTAR detectors are seen to closely match the performance of the synchronous system, despite the presence of frequency offsets.

V. Conclusions

We have proposed the rotating target (ROTAR) algorithm for jointly detecting multiple asynchronous tracks from multiple readback waveforms. ROTAR applies the Viterbi algorithm to a time-varying rotating target that accounts for the asynchrony of the different tracks being detected. To keep complexity low, the timing offsets are decomposed into their integer and fractional parts, and only the fractional parts are used to rotate the target. A further reduction in complexity is realized by locking the ADC's to one track. For the case of unknown timing offsets, ROTAR can use per-survivor processing to embed synchronization inside the joint Viterbi detector.

References

[1] R. Wood, R. Galbraith, and J. Coker, "2-D magnetic recording: Progress and evolution," *IEEE Trans. Magn.*, vol. 51, no. 4, Apr. 2015, Art. no. 3100607.

[2] P. M. Aziz and S. Surendran, "Symbol rate timing recovery for higher order partial response channels," *IEEE J. Sel. Areas Commun.*, vol. 19, no. 4, pp. 635–648, Apr. 2001.

[3] J. R. Barry, A. Kavčić, S. W. McLaughlin, A. Nayak, and W. Zeng, "Iterative timing recovery," *IEEE Signal Process. Mag.*, vol. 21, no. 1, pp. 89–102, Jan. 2004.

[4] B. Vasić and E. M. Kurtas, Eds., *Coding and Signal Processing for Magnetic Recording Systems*. New York, NY, USA: CRC Press, 2004.

[5] B. Vasić *et al.*, "A study of TDMR signal processing opportunities based on quasi-micromagnetic simulations," *IEEE Trans. Magn.*, vol. 51, no. 4, Apr. 2015, Art. no. 9401307.

[6] S. Dahandeh, M. F. Erden, and R. Wood, "Areal-density gains and technology roadmap for two-dimensional magnetic recording," in *Proc. TMRC*, Aug. 2015, paper F1.

[7] P. Kovintavewat, J. R. Barry, M. Faith, M. F. Erden, and E. Kurtas, "A new timing recovery architecture for fast convergence," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2. May 2003, pp. II-13–II-16.

[8] P. Kovintavewat, J. R. Barry, M. F. Erden, and E. Kurtas, "Per-survivor timing recovery for uncoded partial response channels," in *Proc. IEEE Int. Conf. Commun.*, Paris, France, Jun. 2004, pp. 2715–2718.

[9] K. Mueller and M. Müller, "Timing recovery in digital synchronous data receivers," *IEEE Trans. Commun.*, vol. COM-24, no. 5, pp. 516–531, May 1976.

Elnaz Banan Sadeghian received the B.S. degree in electrical engineering from Shahid Beheshti University, Tehran, Iran, in 2005, and the M.S. degree in biomedical engineering from the Amirkabir University of Technology, Tehran, in 2008. She is currently pursuing the Ph.D. degree in electrical engineering with the Georgia Institute of Technology, Atlanta, GA, USA. Her current research interests are in the area of signal processing and communication theory, including synchronization, equalization, and coding as applied to magnetic recording channels.

John R. Barry (SM'04) received the B.S. (*summa cum laude*) degree from the University at Buffalo, The State University of New York, Buffalo, NY, USA, in 1986, and the M.S. and Ph.D. degrees from the University of California at Berkeley, Berkeley, CA, USA, in 1987 and 1992, respectively, all in electrical engineering. His Ph.D. research explored the feasibility of broadband wireless communications using diffuse infrared radiation.

He is a Professor with the School of Electrical and Computer Engineering at the Georgia Institute of Technology. He has held engineering positions in communications and radar systems with Bell Communications Research, Murray Hill, NJ, USA, the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA, Hughes Aircraft Company, Glendale, CA, USA, and General Dynamics Company, Falls Church, VA, USA, since 1985. He has co-authored a book entitled *Digital Communication*–Third Edition (Kluwer, 2004), co-edited a book entitled *Advanced Optical Wireless Communication Systems* (Cambridge University Press, 2012), and authored a book entitled *Wireless Infrared Communications* (Kluwer, 1994).

Dr. Barry currently serves as a Guest Editor of the special issue of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS. He was a recipient of the David J. Griep Memorial Prize in 1992, the Eliahu Jury Award from UC Berkeley, the Research Initiation Award from National Science Foundation, and the IBM Faculty Development Award in 1993.