Iterative Retina for High Track Multiplicity in a Barrel-Shaped Tracker and High Magnetic Field

W. Deng, Zixuan Song, Guangming Huang, Gilles De Lentdecker, Frederic Robert, and Yifan Yang

Abstract—Real-time particle track reconstruction in high-energy physics experiments at colliders running at high luminosity is very challenging for trigger systems. To perform pattern recognition and track fitting in online trigger system, the artificial Retina algorithm has been introduced in the field. Retina can be implemented in the state-of-the-art field-programmable gate array (FPGA) devices. Our developments use Retina in an iterative way to identify tracks in a barrel-shaped tracker embedded in a high magnetic field and with high track multiplicity. As a benchmark, we simulate LHC t-tbar events from 14-TeV proton–proton collisions, with a pile-up of 200 interactions. The produced particles are propagated using GEANT4 in a 4-T magnetic field from the interaction point through a six-layer barrel tracker made of silicon modules. With this sample, the performance of the hardware design [FPGA resource usage and latency] is evaluated. Both track reconstruction efficiency and purity of the Retina track finding are over 90%. To improve further the resolution on the track parameters, we are investigating the addition of a Kalman filter process on the FPGA, after the Retina step. First results obtained with an emulator of the Kalman filter are also discussed in this article.

Index Terms—Field-programmable gate array (FPGA), Kalman filter, retina algorithm, track reconstruction, track trigger.

I. INTRODUCTION

FAST track finding and fitting in online trigger system for the high-energy physics experiments at colliders running at high luminosity are very important. The purpose of fast online tracking is to identify the tracks and to reconstruct them with the highest precision on their track parameters. The main objective is to improve the trigger selection efficiency. Fast online track finding and fitting not only save the bandwidth of data transmission between the front-end readout system and the offline computing center but also can provide preprocess-

II. RETINA AND ITERATIVE RETINA FOR TRACK FITTING

The Retina algorithm is inspired from the processing of visual images by the brain where each neuron is sensitive to a small region of the retina. The strength of each neuron is proportional to how close the actual image projected on the retina region is to the particular shape that particular neuron is tuned to [3] and [4].

When it is used as pattern recognition algorithm for track fitting, the Retina algorithm should find the expected particle trajectories by scanning a discrete multidimension parameter space where each bin or cell represents a stored template of a particle trajectory which intersects the detector layers in $(r_{n, \text{temp}}, \phi_{n, \text{temp}})$ points (where $n$ runs on the detector layers).

Contrarily to the well-known Hough Transform algorithm [5] where each detector hit would correspond to a straight line in the parameter space and a set of hits coming...
charged particle which is related to its Pt (in GeV/c)

$$R_0 = \frac{Pt}{qBc}$$

that is : $R_0 = \frac{Pt[GeV]}{0.15B}$.

(2)

Assuming large curvature radius, we can use the small-angle approximation, that is

$$\sin(\varphi_n - \varphi_0) \approx \varphi_n - \varphi_0$$

(3)

which implies the relationship

$$\varphi_n = \varphi_0 + \frac{0.6r_n}{Pt}.$$  

(4)

The Retina inputs are the hit coordinates $(r_n, \varphi_n)$ on the $n$th layer in the transverse plane. Instead of finding the cell in the $(1/Pt, \varphi_0)$ plane where a set of lines corresponding to the hits from the same particle would intersect, as it is done in a Hough Transform, Retina computes for each hit coming from the detector the distance to all $(r_n^{\text{temp}}, \varphi_n^{\text{temp}})$ hits from any mapped tracks intersecting the same detector layer. Those distances are accumulated over the $N$ layers using a weighted sum

$$\sum_{n=1}^{N} e^{-(d_n/\sigma)^2}$$

(5)

where $d_n$ is the distance between the detector hit and the hit on the same layer of the mapped track, and $\sigma$ is a parameter of the weighting function. $\sigma$ depends on the granularity of the Retina parameter space; it is defined as: $e^{-[2\sigma/(M\sigma)]^2} = 0.9$, where $M$ is the number of bins along $\varphi_0$ in the parameter space.

This sum is computed for all cells of the $(1/Pt, \varphi_0)$ parameter space in a fully parallel way.

Since the Retina algorithm uses a discrete parameter space $(1/Pt, \varphi_0)$, the finer the granularity, the higher the precision on the parameters of the associated track. However, a high granularity also implies a larger number of bins to process which will increase the FPGA resource usage and/or the latency. A compromise will therefore be needed.

To avoid scan the entire parameter space with the highest granularity, we propose to iteratively run Retina with a coarse binning within regions of interest, to scan at the end only a limited region of the parameter space with the highest granularity. We called this method Iterative Retina.

Fig. 2 shows such an example where we run twice Retina on $4 \times 4$ bins, reaching at the end the precision equivalent to the granularity of $16 \times 16$ bins within the same $(1/Pt, \varphi_0)$ parameter space. The first Retina iteration serves to find “regions of interest” in the parameter space where the second iteration should find the track in. In this example, the first Retina iteration identifies the so-called Super cell 7 (in blue in Fig. 2) as the cell of the $4 \times 4$ parameter space with the highest sum. The second iteration will only process this region of interest but with a higher granularity.

The gain in processing with respect to the usual Retina is the following: for a parameter space granularity of $n \times n$ cells with traditional Retina, one would have to scan $n^2$ cells, while with the iterative Retina made of $i$ iterations one has to scan $i \times m^2$ cells, where $n = m^i$. This means decreasing the
interested: the vertex position along the parameters to describe a track in the tracker region we are of the charged particles. Our strategy is to use Iterative Retina number of cells to scan from Fig. 3. One quadrant in the r–z plane of the future CMS Phase 2 outer tracker layout [8], showing placement of the 2S (red) and PS (navy) modules. The barrel part of this tracker (|\eta| < 0.6) is used as benchmark for our studies.

number of cells to scan from \( n^2 = m^{2i} \) to \( i \times m^2 \). At each step, the regions of interest are defined as the cells with the Retina output value which is over the fixed threshold.

So using Iterative Retina, the computation and resource consumption of the algorithms can be significantly reduced with respect to the usual Retina.

III. TRACKER SEGMENTATION AND ITERATIVE RETINA CONFIGURATION FOR FPGA IMPLEMENTATION

Our research and development work focuses on the development of fast tracking algorithms to be run on FPGA and we use the CMS Phase 2 outer tracker detector and fully simulated t–bar events within the LHC CMS experiment as a benchmark for our developments. Note that our results do not represent the most up-to-date CMS track trigger performance [6], [7].

Fig. 3 shows one quadrant in the r–z plane of the CMS Phase 2 outer tracker made of silicon detector layers. Since our algorithm is designed for a barrel-shaped tracker, let us focus on the pseudo-rapidity range \(|\eta| < 0.6\), where the tracker is made of six concentric layers. The radius of those six layers are from \( r_1 = 20 \) cm (innermost) to \( r_6 = 115 \) cm (outermost). The inner three layers are made of PS modules, and the outer three layers are made of 2S modules [8]. The direction of the magnetic field is along the z-axis and the strength is 3.8 T.

Assuming that the tracks originate at \( r = 0 \), there are four parameters to describe a track in the tracker region we are interested: the vertex position along the z-axis \( z_0 \), the initial angle of track in the xy-plane \( \phi_0 \), the pseudo-rapidity \( \eta \), and Pt of the charged particles. Our strategy is to use Iterative Retina to reconstruct the first two parameters, Pt and \( \phi_0 \), and then apply a Kalman filter [9] to reconstruct all four parameters with higher resolution. In the following, the unit of \( \phi_0 \) is centi-radian (crad), the unit of Pt is GeV/c, and the unit of \( z_0 \) is centimeter (cm).

To implement the Iterative Retina with a resource-limited FPGA device, we segment the tracker barrel into 60 sectors: 10 \( \phi \times 6 \eta \) sectors, as shown in Fig. 4. The colored areas represent the overlap region between neighboring sectors where detector hits may have to be assigned to both sectors. For each sector, we use one processor to implement the Iterative Retina. Consequently, one Iterative Retina processor can only one-tenth of the range \([-\pi, \pi]\) of the \( \phi_0 \) parameter and the range \( \text{Pt} \geq 2 \) GeV/c, which means \( 1/\text{Pt} \leq 0.5 \) (GeV/c)\(^{-1}\).

With this tracker segmentation, we have studied three different granularities of the \((1/\text{Pt}, \phi_0)\) parameter space; for each granularity configuration, Retina is run twice.

- **First configuration called hereafter Loop_48**: 8 bins in \( 1/\text{Pt} \times 6 \) bins in \( \phi_0 \) to reach after the second iteration the equivalent granularity of 64 bins in \( 1/\text{Pt} \times 36 \) bins in \( \phi_0 \), which means a cell size of 0.0156 (GeV/c)\(^{-1}\) × 1.745 (crad).
- **Second configuration called hereafter Loop_80**: 10 bins in \( 1/\text{Pt} \times 8 \) bins in \( \phi_0 \) to reach after the second iteration the equivalent granularity of 100 bins in \( 1/\text{Pt} \times 64 \) bins in \( \phi_0 \), which means a cell size of 0.010 (GeV/c)\(^{-1}\) × 0.98 (crad).
- **Third configuration called hereafter Loop_200**: 20 bins in \( 1/\text{Pt} \times 10 \) bins in \( \phi_0 \) to reach after the second iteration the equivalent granularity of 400 bins in \( 1/\text{Pt} \times 100 \) bins in \( \phi_0 \), which means a cell size of 0.0025 (GeV/c)\(^{-1}\) × 0.63 (crad).

As mentioned previously, our simulated t–bar event samples are used to evaluate the performance of our algorithm. Each Iterative Retina processor receives as input only the hits which belong to one of the sector described above. Fig. 5 shows the number of tracks per sector produced from the t–bar events. With the proposed tracker segmentation, ~99% of the t–bar events have less than three tracks to be reconstructed per sector or per Retina processor.

IV. FIRMWARE DESIGN OF THE ITERATIVE RETINA

After getting promising results from simulation, we implemented the Iterative Retina algorithm into FPGA devices to
Fig. 5. Distribution of the number of tracks in one tracker sector for t-tbar events, with a pile-up of 200.

Fig. 6. Top firmware design block diagram of Iterative Retina.

test its online performance. Fig. 6 shows the top-level firmware design of our Iterative Retina algorithm; it includes five units.

- The first unit is a block ram which stores and refreshes the position information of hits coming from the corresponding tracker sector.

- The second unit is the array of Retina cells to scan. The granularity of the parameter space ($M \times K$, $M$ bins for $\phi$ and $K$ bins for $1/Pt$) is configurable. The weighted sum (see Section II) for all cells is processed in parallel and each cell outputs the weighted sum at the same time.

- The third unit is the Sorting unit which sorts the output value of all the cells and selects the ones which are above the chosen threshold or select the one with the highest value.

- The fourth unit is a Control unit consisting of a state machine. The main task of the control unit is to handle the workflow of the Retina iterations. At the beginning, the control unit sends the fixed configuration information to all Retina calculation cells in the Retina calculation array unit and starts the first iteration of Retina algorithm. The configuration information includes the number of iterations and the binning of the parameter space. After the first iteration, the control unit receives the results output from the Retina array unit; the results are the location information of cells picked by the first iteration of Retina and it resends the refreshed configuration and processes the second iteration of Retina. Note that the number of iterations can be chosen by the user, but in our application two iterations seem sufficient. Except the first iteration, the subsequent iterations may use several arrays of Retina cells depending on the number of cells found over threshold by the previous iteration. We have limited the number of Retina arrays running in parallel to 15. When the requested number of iterations of Retina is achieved, the control unit loads the input data of the next event and restarts the cycle.

- The last unit is the IPbus [10] communication unit. It sends the results’ data from the Retina processor to a computer for offline analysis. The bandwidth of IPbus is up to 1 Gbit/s; it is suitable for our current firmware implementation. If needed for future developments, we could upgrade to the PCIe protocol.

Fig. 7 shows a block diagram of the firmware implementation of the computation of one Retina cell, for example, the cell ($m$, $k$). The Retina computation of (5) is made of three steps: the first one computes the distances $d_n$ between the input hit positions ($r_n$, $\phi_n$) and the expected position ($r_{temp}^n$, $\phi_{temp}^n$) of the template track corresponding to the cell ($m$, $k$), on each detector layer. This function is designed by the high-level synthesis (HLS) tools of the Xilinx Vivado toolkit. Note that as we expect 99% of the t-tbar events with a pile-up of 200 interactions to have at most three tracks per tracker sector, we have limited the input here to 18 detector hits. The second step (green blocks in Fig. 7 and named EXP below) computes the square of the distances and the exponential operation. Since that step uses only one input and one output, we use the FPGA lookup table (LUT) resources to implement it. The mathematical functions and both block ram and distributed ram are suitable for it. In addition, we have checked that adding 10% of random hits to the t-tbar event track hits do not degrade the Retina performance.

The firmware design has been tested with a Xilinx KC705 evaluation board [11], where a Kintex7 series FPGA chip is used. The resource usage and latency of the Iterative
TABLE I
RESOURCE USAGE OF ITERATIVE RETINA IMPLEMENTED IN KC705 FPGA DEVICE

| Firmware Design block     | DSP  | LUT  | BRAM | FF   |
|----------------------------|------|------|------|------|
| KC705                     | 840  | 203800 | 445  | 407600 |
| One calculation cell      | 1    | 441  | 0    | 261  |
| Retina calculation Array(200 cells) | 200 | 87,735 | 0 | 52,208 |
| Control unit              | 0    | 2704 | 0    | 270  |
| Sorting unit              | 0    | 6415 | 0    | 1358 |
| LOOP_200 case             | 200  | 112326 | 0.5  | 58289 (23.81%, 55.2%, 0.11%, 14.3%) |

TABLE II
LATENCY OF ITERATIVE RETINA IMPLEMENTED IN KC705 FPGA DEVICE

| Clock frequency(Hz) | Steps of Iteration Retina | clk cycles cost | Latency (ns) |
|---------------------|---------------------------|-----------------|--------------|
| Distance calculation| 24                        | 120             | 20           |
| Square & Exp function| 2                         | 10              | 2            |
| Accumulate function | 4                         | 20              | 2            |
| First iteration of Retina | 30                      | 150             | 2            |
| Super cell selecting | 4                         | 20              | 2            |
| Second iteration of Retina +Sorting output | 30*N+4             | 5              | (150*N+4)*5   |
| Control & Command    | 8                         | 40              | 40           |
| 200M                 |                           | 496(max)        | 2480         |

Retina are shown in Tables I and II, respectively. To process one event, it takes 496 clock cycles at the clock frequency of 200 MHz, and it costs nearly half of the KC705 FPGA resources to compute 200 Retina parameter space cells, the highest granularity we tried. To process one parameter space cell, it costs 1 DSP and 441 LUTs in the FPGA device. From Table II, we can see that the current online performance would result in a latency of 2.48 $\mu$s, which is typically good enough to meet most of the LHC experiments’ hardware trigger requirement. Those numbers are preliminary and should be considered with care before trying any extrapolation to a full tracker, like the CMS one. Due to the limited resources of the KC705 FPGA, the Retina array calculations of the subsequent iterations could not be run in parallel but are serialized, which is reflected in Table II. That means with a bigger FPGA, like the existing Xilinx Ultrascale FPGA, the latency would typically reduce by 420 ns at a clock frequency of 200 MHz.

V. KALMAN FILTER FOR TRACK PARAMETER RECONSTRUCTION

Considering the reconstruction resolution requirement for LHC tracker trigger systems, typically a few percentage on the Pt, it is hard to meet this requirement with only the Iterative Retina although it has a good performance in efficiency and purity. Because in Iterative Retina we scan the 1/Pt instead of the Pt parameter, in case of high Pt (> 10 GeV/c) the resolution of Pt will decrease a lot. It is worse than 10% at Pt $\geq$ 10 GeV/c with the highest Retina granularity. This is caused by the principle of pattern recognition, when the Pt of the particle is high enough, the curvature radius, $R_0$, of the track approaches infinity and the shape of the trajectory becomes infinitely close to a straight line. So for pattern recognition algorithms, it is hard to distinguish the curve from those infinitely close to straight line tracks.

Consequently, after Iterative Retina, we add a Kalman filter to reconstruct the parameters of the tracks found by Iterative Retina with higher precision. Kalman filters are widely used as parameter reconstruction algorithm for many applications [12]. Fig. 8 shows the data flow of our Kalman filter implementation subsequent to Iterative Retina. The input of the Kalman filter are the detector hits associated with the track found by Iterative Retina and the track parameters (1/Pt, $\phi_0$) returned by Iterative Retina [which means the coordinates of the cell with the highest sum in (5)]. Those track parameters will serve as initial state vector for the Kalman filter.

In the following, we briefly remind the concept of the Kalman filter. A detailed description can be found in [9] and [12]. The Kalman filter is an iterative process, which propagates its state vector (the track parameters in this case), from one detection layer to the next detection layer. On this next detection layer, the state vector is updated according to the measurements (the detector hits) on this same layer and their uncertainties. As the state vector is updated along the propagation of the track through successive detection layers, the Kalman filter outputs the most precise track parameters at the end of the process. Note that while Iterative Retina was built to return two track parameters (1/Pt and $\phi_0$), the Kalman filter will be built to reconstruct the track in 3-D and to return four track parameters (1/Pt, $\phi_0$, cot$\theta$, and $z_0$), where $\theta$ is the polar angle and $z_0$ is the longitudinal impact parameter.

For the $i$th track returned from Iterative Retina, the Kalman filter starts with the initial state vector $x_i(0)$, and the associated covariance matrix $P$. After $t$ layers, the Kalman filter equations are the following.

- The state at the layer $t + 1$ is obtained by extrapolating the state from layer $t$ to layer $t + 1$

$$\hat{x}_{i+1|t} = F_t \hat{x}_{i|t}$$  \[6\]

where $F_t$ is the propagation matrix which propagates the states according to the particle trajectory. $\hat{x}$ indicates the estimate of $x$. The propagation being subject to error due to multiple scattering and energy losses, we define $Q_t$ the propagation of noise from layer $t$ to layer $t + 1$. 

![Fig. 8. Diagram to show the input and output of Kalman filter for track fitting application in tracker detector.](image-url)
The covariance matrix therefore evolves like
\[ P_{t+1|t} = Q_t + F_t P_{t|t} F_t^T \]  
where \( P_{t+1|t} \) is the extrapolated covariance.

- The extrapolated parameters are then projected onto the measurement space using the projector matrix \( H_t \)

\[ \hat{m}_t = H_t \hat{x}_{t|t-1} \]  
where \( \hat{m}_t \) are the resulting measurements which are compared with the measured data (the hits), the vector \( m \) with the associated covariance \( R_t \). From those equations, one can calculate the Kalman gain matrix

\[ W_t = P_{t+1|t} H_t^T (H_t P_{t+1|t} H_t^T + R_t)^{-1} \]

which describes the gain in precision that the measurement brings to the extrapolation.

The projector matrix, the covariance matrix, and the measurement error matrices are defined as

**Projector matrix**
\[
H_t = \begin{bmatrix}
-r & 1 & 0 & 0
\end{bmatrix}
\]

**Covariance matrix**
\[
P_{t|t} = \begin{bmatrix}
\sigma_a^2 & \sigma_{ab} & 0 & 0 \\
\sigma_{ab} & \sigma_b^2 & 0 & 0 \\
0 & 0 & \sigma_c^2 & \sigma_{cd} \\
0 & 0 & \sigma_{cd} & \sigma_d^2
\end{bmatrix}
\]

**Measurement error matrices**
\[
R_t = \begin{bmatrix}
\frac{\sigma_{\phi}^2}{2} & 0 \\
0 & \frac{\sigma_z^2}{2}
\end{bmatrix}, \quad \sigma_{\phi}^2 = \left( \frac{2}{\sqrt{2}} \right)^2, \quad \sigma_z^2 = \left( \frac{1.65625 l}{\sqrt{12}} \right)^2.
\]

The initial covariance matrix is determined from the uncertainties in the Retina step: \( \sigma_{ab} \) and \( \sigma_{\phi} \) are equal to 0, \( \sigma_a \) and \( \sigma_b \) are set to \( 1/(12)^{1/2} \) of the corresponding Retina cell parameter width, the value of \( \sigma_\phi \) is \( 1/(12)^{1/2} \) of the corresponding \( \eta \) sector width we focus on, and the value of \( \sigma_d \) is equal to the rms of the LHC beam spot width (5 cm). The matrix \( R_t \) indicates the uncertainties on the hit measurements in the CMS tracker detector where \( p \) is the strip pitch and \( l \) is the strip length (5 cm in the 2S modules and 1.5 mm in the PS modules).

**VI. Results Analysis and Conclusion**

To evaluate the improvements in the track parameter resolution brought by the Kalman filter, so far, we have implemented it in software. The Kalman filter processes the data sent through IPbus by Iterative Retina running on the FPGA. The porting of the Kalman filter to firmware (ignoring for now the KC705 FPGA resource limit) should not be a problem as this task has already been performed by others [7], [13].

Table III shows the efficiency and the purity of the track finding by Iterative Retina for the three granularity configurations, as well as the number of ghost, mismatched, and duplicated tracks. A track is defined as matched to the simulated track if five hits are associated with the simulated track. Mismatched tracks are simulated tracks which are not found by Retina. The efficiency is defined as 1—(mismatched tracks/total tracks). Ghost tracks are tracks found by Retina which do not have hits associated with a simulated track. Finally, a duplicated track is a simulated track for which Retina found several tracks, typically a cluster of adjacent cells in the parameter space.

As shown in Table III, the efficiency and purity in the three granularity configurations are similar and over 90%. So with the Iterative Retina, most of the tracks in the t-tbar events with a pile-up of 200 interactions can be identified even in case of reduced granularity. However, increasing the granularity significantly reduces the number of ghosts and improves the purity for a moderate loss of efficiency, \( \sim 2\% \).

Figs. 9 and 10 represent the distribution of reconstructed \( P_t \) for the three Iterative Retina granularity configurations; from top to bottom: Loop-48, Loop-80, and Loop-200, respectively. The left plots are the results after Iterative Retina and the right plots are the results with the Kalman filter.
Fig. 10. Distribution of the reconstructed $\phi_0$ as a function of simulated $\phi_0$ for the three Iterative Retina granularity configurations; from top to bottom: Loop-48, Loop-80, and Loop-200, respectively. The left plots are the results after Iterative Retina and the right plots are the results with the Kalman filter. From Figs. 9 and 10, we can also see the improvements in the resolution provided by Iterative Retina with higher granularity in the parameter space.

Fig. 11. Resolution on Pt (bottom) and $\phi_0$ (top) as a function of the particle Pt after Iterative Retina (triangles) and after Iterative Retina + Kalman filter (stars). Note that in Fig. 11, the 10-GeV/c bin includes all tracks with Pt $\geq$ 10 GeV/c. As expected, the resolution improves with Retina granularity, typically by a factor of 2 from a granularity of 8 bins in 1/Pt $\times$ 6 bins in $\phi_0$ to a granularity of 20 bins in 1/Pt $\times$ 10 bins in $\phi_0$, both with two Retina iterations. The Kalman filter significantly improves it further. The resolution on Pt improves to 1% over the considered range. It is interesting to note that even for the lowest Iterative Retina granularity, the Kalman filter makes such an improvement, which means that one can lower the Retina parameter space granularity and save some FPGA resources. This implies that one has some flexibility regarding the implementation of the Kalman filter on the FPGA and that the FPGA resource usage can further be optimized.

As a conclusion, although many improvements are still needed to bring Retina as a potential track trigger algorithm for an experiment like CMS, we have demonstrated that it could be implemented on an FPGA and used in combination with a Kalman filter to find and fit curved high Pt tracks in a barrel-shaped tracker surrounded by a high magnetic field and with a high track multiplicity environment. The latency, the tracking reconstruction efficiency, and the resolution on the track parameters are close to the ones expected from a track trigger like the CMS one.

REFERENCES

[1] W. Deng, “Study of track reconstruction using Retina algorithm for charged particles in magnetic field,” in Proc. Top. Workshop on Electron. Particle Phys. (TWEENP), 2018, p. 21.
[2] S. Agostinelli, “GEANT4: A simulation toolkit,” Nucl. Instrum. Methods Phys. Res. Section A, Accel., Spectrometers, Detectors Associated Equip., vol. 506, no. 3, pp. 250–303, 2003.
[3] R. Cenci et al., “First results of an ‘artificial Retina’ processor prototype,” EPJ Web Conf., vol. 217, p. 00005, Nov. 2016, doi: 10.1051/epjconf/201612700005.
[4] A. Piaci, “Reconstruction of tracks in real time in the high luminosity environment at LHC,” M.S. thesis, Facolta Di Scienze Matematiche, Fisiche E Naturali, Università degli Studi di Pisa, Pisa, Italy, 2014.
[5] R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures,” Commun. Assoc. Comput. Mach., vol. 15, no. 1, pp. 11–15, Jan. 1972.
[6] E. Bartz et al., “FPGA-based tracking for the CMS level-1 trigger using the tracklet algorithm,” J. Instrum., vol. 15, no. 6, Jun. 2020, Art. no. P06024.
[7] T. James, “A hardware track-trigger for CMS at the high luminosity LHC,” Ph.D. dissertation, Dept. Phys., Imperial College London, London, U.K., Feb. 2018.
[8] TDR of CMS Experiment, Department of CMS Collaboration Team, “The phase-2 upgrade of the CMS tracker technical design report,” Tech. Rep. CERN-LHCC-2017-009; CMS-TDR-17-001, CMS Collaboration, Jul. 2017.
[9] R. Frühwirth, “Application of Kalman filtering to track fitting in the DELPHI detector,” Tech. Rep. CERN-DELPHI-87-23-PROG-70, 1987.
[10] C. G. Larrea et al., “IPbus: A flexible Ethernet-based control system for xTCA hardware,” J. Instrum., vol. 10, no. 2, Feb. 2015, Art. no. C02019.
[11] Xilinx UG810. KC705 Evaluation Board for the Kintex-7 FPGA User Guide. Accessed: Feb. 4, 2019. [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/kc705/ug810_KC705_Eval_Bd.pdf
[12] A. C. Harvey, “Applications of the Kalman filter in econometrics,” in Advances in Econometrics, T. Bewley, Ed. New York, NY, USA: Cambridge Univ. Press, 1994, p. 285.
[13] C. Wang, E. D. Burnham-Fay, and J. D. Ellis, “Real-time FPGA-based Kalman filter for constant and non-constant velocity periodic error correction,” Precis. Eng., vol. 48, pp. 133–143, Apr. 2017.