Encoded Check Driven Concurrent Error Detection in Particle Filters for Nonlinear State Estimation

Chandramouli N Amarnath, Md Imran Momtaz and Abhijit Chatterjee
School of Electrical and Computer Engineering. Georgia Institute of Technology, Atlanta GA 30332.
Email: chandamarnath@gatech.edu, momtaz@gatech.edu, abhijit.chatterjee@ece.gatech.edu

***Abstract*—In this paper we propose a framework for concurrent detection of soft computation errors in particle filters which are finding increasing use in robotics applications. The particle filter works by sampling the multi-variate probability distribution of the states of a system (samples called particles, each particle representing a vector of states) and projecting these into the future using appropriate nonlinear mappings. We propose the addition of a 'check' state to the system as a linear combination of the system states for error detection. The check state produces an error signal corresponding to each particle, whose statistics are tracked across a sliding time window. *Shifts in the error statistics across all particles* are used to detect soft computation errors as well as anomalous sensor measurements. Simulation studies indicate that errors in particle filter computations can be detected with high coverage and low latency.**

***Index Terms*—Autonomous Systems, Particle Filters, Error detection, State-space check, Resilience**

I. Introduction and Prior Work

Complex autonomous systems such as robots have to manage high degrees of uncertainties in their operating environment as well as internal failures in sensors, actuators and computational subsystems. In this context, accurate state estimation is important for appropriate feedback control and is often facilitated by the use of particle filters [1]. The particle filter represents samples of the probability distribution of the system states, termed the *posterior* distribution, using representative *particles*. Each particle is a vector of values of all the state variables of the system. A sufficiently large set of such particles is mapped to the next time step in the prediction phase of the particle filter's operation using the system's state transition function. The filter then uses the system's sensor measurements of a subset of states in the next time step to update the particle set and its attributes to represent the distribution of states corresponding to that time step. The particle filter's states are estimated with a duty cycle determined by the rate at which sensor measurements are evaluated. Software running on a digital processor is used to perform state estimation using the particle filter as well as control actuation from the estimated states using a nonlinear control algorithm. The particle filter is able to cope with nonlinearity and non-Gaussian measurement statistics more robustly as compared to the Extended or Unscented Kalman Filter [2] at the cost of higher computational complexity. This higher computational complexity, mitigated by dedicated electronics in contemporary systems [3], [4], makes the underlying electronics vulnerable to computation soft errors [5]. Due to the safety-critical nature of the state estimation task performed by particle filters, such errors must be detected rapidly with high coverage.

Prior work in error detection involving particle filters has investigated anomalies in sensed input data [6] and not soft computation errors. Earlier work on algorithm-based fault-tolerance [7] used linear checksums computed concurrently with execution of signal processing algorithms to enable soft error detection and correction. Real number codes have been used for fault detection in matrix computations [8]. There has also been work on use of encoded state space checks for linear Kalman filters [9]. The methods explored in [7]–[9] are amenable to only linear transformations of input data. The lack of explicit matrix computations in particle filters as well as nonlinearity makes real number codes or encodings as in [9] unsuitable for error detection. Recently, machine learning (ML) algorithms [10] have been used to implement checks for nonlinear systems. Also, reduced-precision redundancy and statistical error compensation such as detailed in [11] have been used for error detection. However, none of the above techniques can be applied directly to particle filters because the latter operates on *samples of the probability distribution of the system states* each of which evolves across time steps, requiring a *distributed checking scheme across all particles* as opposed to unified algorithmic checks employed in the state of the art.

In this paper, a low overhead, low cost solution to the problem of soft error detection in particle filter computations is proposed. In addition to the system state variables, a check state which is a linear sum of all the system state variables is computed. The incremental changes in the check state values are monitored across time and the aggregate statistics of such changes is computed in real time. Detected shifts in the probability distribution of the incremental changes in the check state value are used to flag the presence of single or multi-bit soft errors in the particle filter computations. High coverage, low latency detection of such error is achieved. We simulate soft errors as bit errors (single bit flips) and word errors (multiple bit flips) in fixed-point particle filter computations during state estimation. Soft errors can induce errors in state estimation, which affects controller performance due to the importance of state estimation in the control process.

Section II presents a brief description of the particle filtering process as applied to state estimation. We introduce the proposed error detection approach in Section III. Experimental results are discussed in Section IV. Finally, we conclude in

Section V.

II. The Particle Filter: Background

The particle filter algorithm proceeds in the following sequence of steps [12]:

- Initialize a starting distribution $\pi(x_0|z_0) = \pi(x_0)$ where $\pi(.)$ is the estimated distribution of the states x_0 (x_k) at time $t = 0$ (t_k), and starting weights for each particle as $w_0^i = \frac{1}{N}$, where N is the number of particles in the estimate of the distribution. z_k is the measurement made by the system at time step k. Here $\pi(x_k|z_k)$ denotes the estimated distribution of the states conditional on the measurements at time step k, z_k. Initially $k = 0$.
- Prediction step: Project the density estimates ahead using the state function $f(x, u)$ to get $\pi(x_{k+1}|z_k)$, the predicted distribution.
- Update particle weights from the new measurement and the previous weights.
- Estimate from the measurement and updated weights using Bayes' Theorem to get the corrected distribution $\pi(x_{k+1}|z_{k+1})$.
- Increment k. Repeat the second step with the current distribution.

The prediction step uses the state model and previous distribution estimates to get the predicted distribution at time k as $\pi(x_k|z_{k-1}) = \int \pi(x_k|x_{k-1})\pi(x_{k-1}|z_{k-1})dx_{k-1}$,where $\pi(x_k|x_{k-1})$ is obtained from the state transition function and $\pi(x_{k-1}|z_{k-1})$ from the previous state update.

The samples are reweighted based on the measurement distribution $\pi(z_k|x_k)$ and the current sensor measurement z_k. For the ith particle at time step k the weight is now $w_k^i = w_{k-1}^i \pi(z_k|x_k)$ and all weights normalized such that $\sum_{i=1}^N w_k^i = 1$. This is followed by estimation of the updated distribution using Bayes Theorem, giving $\pi(x_k|z_k) \approx \sum_{i=1}^N w_k^i \delta(x - x_k^i)$, where $\delta(x - x_k^i)$ is the Dirac delta function centered at x_k^i and w_k^i the weight of x_k^i. The state estimates at the prediction and the update step are obtained from the expectations of the distributions that the predict and update steps result in, to give the predicted state estimate $x_{pred,k}$ and corrected state estimate $x_{corr,k}$ at time step k.:

$$x_{pred,k} = \mu(\pi(x_k|z_{k-1})) \tag{1}$$

$$x_{corr,k} = \mu(\pi(x_k|z_k)) \tag{2}$$

where μ is the expectation of the distribution. The particle filter approximates the posterior distribution at each time step via a weighted Monte Carlo sampling of the probability distribution of the system states. With increasing numbers of particles (fixed for a given implementation), the Monte Carlo sampling asymptotically characterizes the posterior distribution. The Sequential Importance Sampling (SIS) algorithm laid out in [12] samples from an importance distribution that is a 'stand-in' for the unknown posterior. Following that, the particles are re-weighted based on the Monte Carlo approximation of the posterior distribution and normalized.

The particle filtering algorithm often ends up with the problem of degeneracy, whereby different particles are assigned largely different weight values. To avoid the degeneracy problem we perform resampling to rebalance the particle weights. Resampling generates N new samples $\{\tilde{x_k^i}\}_{i=1}^N$ from the approximation of the posterior distribution at that time step, removing low weight particles and generating new samples in regions of high weight [13].

III. Proposed Error Detection Approach

We propose in this work a soft error detection framework for (nonlinear) particle filters that is outlined in Figure 1.

For the purpose of error detection, an additional check state C_{t-k} is associated with each state estimate X_{t-k} (in Figure 1, the subscripts for C and E below are not shown for brevity), with C_{t-k} being a linear mapping of the particle filter (system) states and $t-k$ being the time step k steps before the current time. The *prediction step* of the particle filter uses the nonlinear state function $f(X, u)$ to produce a predicted state estimate $\hat{X}_{t-k+1}$ from X_{t-k}. Sensor measurements at time $t-k$ are then used to *update* the state estimate to X^*_{t-k+1} from $\hat{X}_{t-k+1}$. This yields two more check state values from the predicted and corrected state estimates above, namely $\hat{C}_{t-k+1}$ and C^*_{t-k+1}, respectively. We take the difference of these and the original check state C_{t-k} corresponding to X_{t-k} to obtain the predicted and corrected check errors, $\hat{E}_{t-k+1}$ and E^*_{t-k+1}. These are generated at each time step from the state estimation process. A sliding-window mean of $\hat{E}_{t-k+1}$ and E^*_{t-k+1} is then taken over $k-1$ time steps and a statistical test is used to determine to see if the distribution of the corresponding error values has changed to a significant degree. If so, we flag an error at that time step.

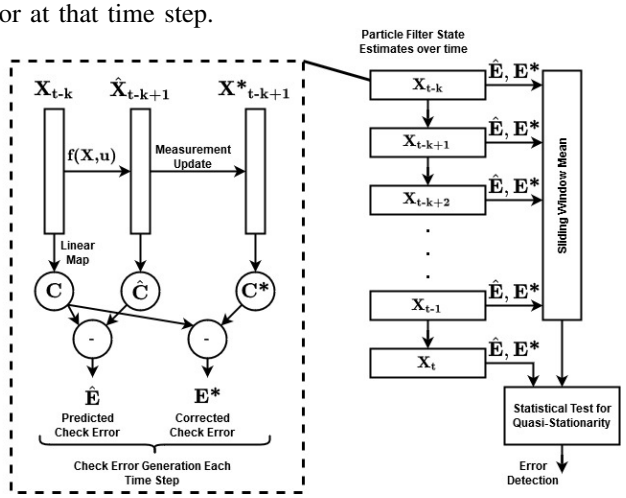

Fig. 1: Overall Nonlinear Error Checking Process

We apply this to the particle filter with a state equation of the form $x(k+1) = f(x(k), u(k))$ where $x(k)$ denotes the system states at time step k, $u(k)$ denotes control input at time step k, and $f(.)$ is the state transition function. The check state can be formulated as:

$$C(t) = \sum_{i=1}^{n} \alpha_i x_i(t) + \sum_{j=1}^{m} \lambda_j u_j(t) \tag{3}$$

where $C(t)$ is the check state, α_i is the weight applied to the ith state, λ_j the weight applied to the jth control input, $x_i(t)$ the ith state at time t, and $u_j(t)$ the jth control input at time t. There are n states and m control inputs.

A. The State Check

Under the assumption that the state transition noise distribution of the system changes slowly enough to be approximately equal, the difference of the predicted and corrected values of the state check as per Equations (1) and (2) should be zero or close to zero. The state check is thus $E_s(t+1) = C(t+1) - C(t)$. Substituting for C(t), we get

$$E_s(t+1) = C(t+1) - \sum_{i=1}^{n} \alpha_i x_i(t) - \sum_{j=1}^{m} \lambda_j u_j(t) \quad (4)$$

where C(t) is the particle filter's estimate of the check state at time t, similarly the control and state vectors u and x are drawn from the particle filter's estimates. We thus have two state check quantities $\hat{E}_s$ and E_s^* depending on whether the predicted or corrected estimates are used. Their difference is called the state check error for the particle filter,

$$\epsilon(t) = \hat{E}_s(t) - E_s^*(t) \quad (5)$$

To avoid contamination of the quantity by unbalanced Monte Carlo sampling, degeneracy, and transient noise, we take these quantities as averages over a sliding window to get

$$\epsilon_{avg}(t) = \frac{1}{W} \sum_{i=0}^{(W-1)} (\hat{E}_s(t-i) - E_s^*(t-i)) \quad (6)$$

as the final state check error value, with W being the window length as measured in time steps. The state check error spikes when the state transition noise changes its distribution characteristics within a short time. Such changes in the transition noise can be from sudden changes in inputs which are interpreted as transition noise over the single time step by the particle filter, and the state check is used to detect such errors.

B. The Mean Check

The mean check is intended to measure the shift in the particle filter's estimate of the check state distribution between time steps, using the difference between the predicted and the corrected distributions. We define the mean check $M(t)$ as:

$$M(t) = \mu(P(C^*(t))) - \mu(P(\hat{C}(t))) \quad (7)$$

where $\mu(.)$ is the empirical mean of the estimated distribution, $P(C^*(t))$ is the probability distribution estimated by the particle filter for the check state after correction, and $P(\hat{C}(t))$ is the distribution predicted by the particle filter before correction, using the previous time step's distribution estimate. We take this over a sliding window to minimize the effects of unbalanced Monte Carlo sampling, degeneracy, and transient noise:

$$M_{avg}(t) = \frac{1}{W} \sum_{i=0}^{(W-1)} (\mu(P(C^*(t-i))) - \mu(P(\hat{C}(t-i)))) \quad (8)$$

with the window length in time steps being W. Ideally, shifts in the mean of the check state distribution are statistically insignificant so long as the particle filter itself operates at time steps small enough to accurately estimate the system's states and the particle filter has stabilized after initialization. Should the mean of the check state distribution estimate shift due to errors, detection is handled by the mean check.

C. Thresholding the Mean and State Checks

We flag errors based on a statistical threshold determined by a hypothesis test. We use Student's t test to choose the confidence intervals [14]. This assumes that the given quantity being tested can be approximated as a normal distribution with prescribed mean and variance. We thus make this assumption implicitly for $M_{avg}(t)$ and $\epsilon_{avg}(t)$. and calculate a running mean and standard deviation of $\epsilon_{avg}(t)$ and $M_{avg}(t)$ to set their respective confidence intervals. The limits of the confidence intervals are thus:

$$L = \mu \pm k\sigma \quad (9)$$

where μ is the running mean of the check under evaluation ($M_{avg}(t)$ or $\epsilon_{avg}(t)$), and σ is its standard deviation, both of which are measured over a sliding window. In this work we use $k = 1.96$ as the multiplier to signify that errors in the extreme five percent of the distribution are statistically significant. Once the state check error or the mean check breach the bounds of Equation 9 the system raises a flag signifying error detection.

Algorithm: Particle Filter Error Detection

1 **Initialize** N samples $x_{i,0}$ from known distribution $\pi(x_0)$ with initial weights $\frac{1}{N}$
2 $k \leftarrow 0$; detected $\leftarrow$ False;
3 **while** *System is running* **do**
4 **Prediction:** Generate predicted distribution as $x_{i,k+1} \sim \pi(x_{k+1}|x_k)$ from state transition function
5 **Predicted Checks:** Calculate Predicted State Check $\hat{E}_s$, mean of predicted distribution $\mu(P(\hat{C}(t)))$
6 **Update:** Update weights as per (10) and the measurements taken in at $k+1$.
7 **Estimation:** Estimate the posterior $\pi(x_k|z_k) \approx \sum_{i=1}^{N} w_k^i \delta(x - x_k^i)$.
8 **Corrected Checks:** Calculate Corrected State Check E_s^*, mean of corrected distribution $\mu(P(C^*(t)))$
9 **Check State Calculation:** $\epsilon(t) \leftarrow \hat{E}_s(t) - E_s^*(t)$; $M(t) \leftarrow \mu(P(C^*(t))) - \mu(P(\hat{C}(t)))$;
10 **Windowed Average:** $\epsilon_{avg}(t) \leftarrow \frac{1}{W}\sum_{i=0}^{(W-1)}(\hat{E}_s(t-i) - E_s^*(t-i))$; $M_{avg}(t) \leftarrow \frac{1}{W}\sum_{i=0}^{(W-1)}(\mu(P(C^*(t-i))) - \mu(P(\hat{C}(t-i))))$;
11 **Bound Calculation:** $L_\epsilon \leftarrow \mu_\epsilon \pm k\sigma_\epsilon$;$L_M \leftarrow \mu_M \pm k\sigma_M$;
12 **if** $|\epsilon_{avg}(t) - \mu_\epsilon| > k\sigma_\epsilon$ *or* $|M_{avg}(t) - \mu_M| > k\sigma_M$ **then**
13 $detected \leftarrow$ True
14 **Resampling:** If degeneracy is seen in the distribution, trigger resampling.
15 $k \leftarrow k+1$

D. Error Detection Implementation Flow:

The overall error detection flow is presented in Algorithm: Particle Filter Error Detection. In Lines 1-2 we initialize the system components, with N particles drawn with uniform weights from the initial proposal distribution $\pi(x_0)$, and the system time steps beginning from this point at $k = 0$. We

initialize the variable $detected$ in Line 2 as initially false, and it is made True to flag an error.

At each time step the system projects all particles ahead using the system state transition function as in Line 4, and using the predicted distributions we calculate the predicted state check $\hat{E}_s$ and the empirical mean of the predicted distribution $\mu(P(\hat{C}(t)))$ in Line 5. The update and estimation steps of the particle filter of Lines 6 and 7 then act to re-weight the distribution and estimate states based on the measurements made by the system in that time step. We use this information to calculate the state check for the corrected distribution E_s^* and the mean of that distribution $\mu(P(C^*(t)))$ in Line 8.

We then calculate the state check error and the mean check for the current time step in Line 9 using Equation 5 and 7 respectively. This is then used in Line 10 to calculate $M_{avg}(t)$ and $\epsilon_{avg}(t)$ across the sliding window as per Equations 8 and 6 respectively. The bounds are then calculated in Line 11 as per Equation 9 to get L_ϵ and L_M as bounds for the state check error and mean check respectively. We use the value of $k = 1.96$ in our work to set the bound in Line 12. We then check for the presence of an error, seeing if $\epsilon(t)$ or $M(t)$ breach their bounds in Line 12 of the algorithm and raising the $detected$ flag to indicate that in the following line if they do. Once this is done, the particle filter checks for degeneracy and resamples as needed before continuing with state estimation.

IV. Experimental Results: Case Study on Vehicle Tracking:

We have tested the proposed error detection approach on a particle filter for estimating the states of an autonomous vehicle using MATLAB simulations. We use the kinematic bicycle model [15] for the autonomous vehicle, given by: $\dot{x} = v\cos(\phi + \beta)$, $\dot{y} = v\sin(\phi + \beta)$, $\dot{\phi} = \frac{v}{l_r}\sin(\beta)$, and $\dot{v} = a$. Here x and y are position coordinates, v the velocity scalar, a the acceleration input scalar, ϕ is vehicle orientation angle, β the angle of velocity of center of mass. l_r is distance from vehicle center of gravity to vehicle rear. Canonically $\beta = tan^{-1}(\frac{l_r}{l_f+l_r}tan(\delta_f))$ where $l_f(l_r)$ is the distance to the front (rear) axle and δ_f is the front steering angle. We assume that l_f is zero to give $\beta = \delta_f$ for simplicity. We thus have β as being the effective steering angle in this work, and we refer to it as the steering angle.

Errors were injected in the form of bit errors (single bit flips) and word errors (multiple bit flips) for single time steps (transient faults) or multiple time steps (burst errors) in fixed-point computations of the particle filter on a digital processor. A word length of 16 bits was assumed with 10 bits representing the integer part of the number. For errors in particle weights, due to the small values of the weights, the integer part of the number was represented in 7 bits. Errors were injected into the processes of distribution estimation, calculation of the predicted state, and the final state estimation steps of the particle filter. Further, errors could be due to bit or word errors in individual particle values or in the weight calculations.

The test scenario itself consists of ten seconds of simulated time with the vehicle moving at 0.5m/s and turning at the rate of 0.01rad/s. The particle filter was used to estimate the vehicle states with the measurements being the x and y coordinates of the vehicle sampled at 100Hz. 1,000 particles were used for the importance sampling process. The check weights were taken as $\alpha_i = \lambda_j = 1$ for all i, j. The control actuation of the vehicle was appended to the vehicle state vector to simplify the state space vehicle model, restricting the particle filter to modeling of the vehicle-relevant state variables rather than states of the associated controller as well. Errors were injected at random times with burst errors present for random durations. The average detection latency and coverage for each error type over 1,000 simulation runs was monitored. A window length of 35 time steps was used for simulation. Due to the sensor time steps being 10ms, the minimum possible detection latency is 10ms. The results are shown in Table I. From Figures 2-5 we see that the particle filter's state estimate stabilizes at 1-2cm of tracking error after 2s, and the check error values stabilize in under 1s.

A. Errors in State Prediction and Estimation

Errors in state prediction are injected as bit flips in the computation of $x_{pred,k}$, and errors in state estimation as bit flips in the computation of $x_{corr,k}$ as in Section II. In Table I we see that detection of single bit flip transient errors in state estimation and prediction over the range of the full word show a detection coverage of 81-82%. This is due to situations as in Figure 2 where the bit flip occurs in the LSB and does not materially affect tracking performance. As we see in Figure 2 the tracking performance deviates by 1.5 cm due to the LSB bit flip injection in state estimation, causing deviation in the state check error, but not enough for detection. A lower value of k can flag such cases at the cost of increased false alarms. Looking at errors in the more significant bits such as the integer bits and upper five bits we see total coverage and immediate detection with latencies of close to the 10ms minimum. For burst errors we note coverage of more than 90% due to their greater impact on vehicle performance. An example is seen in Figure 3 where a transient bit flip in state estimation causes a 4cm deviation in tracking and detection occurs immediately after fault injection using the state check, with the state check error breaching its lower confidence bound.

B. Errors in Distribution Sampling

We also injected bit flips in the distribution sampling step of the particle filter, targeting a single state of a high-weight particle as well as the particle weights. We injected bit flips in the upper bits of one state in the highest-weighted particle in the distribution $\pi(x_{k-1}|z_{k-1})$ at the start of state estimation. We note that errors in lower bits cause little impact on performance, as seen in Figure 4 where a transient bit flip in position 8 of one state of the chosen particle has no impact on tracking error or on the mean check. By contrast, a bit flip in the upper bits of the integer portion of the word as in Figure 5 causes a large deviation in tracking error that is immediately detected by the mean check. Coverage for this fault is seen to be 92.8% in Table I for transient single bit errors in the upper bits, while more significant faults such as burst errors

Transient Error					
Fault Location	Bitflip Location	Bit Error		Word Error	
		Coverage (%)	Average Detection Latency (ms)	Coverage (%)	Average Detection Latency (ms)
State Estimation	Full word	81.1	10.66	99.9	10.01
	Integer bits	100	10.01	100	10
	Upper five bits	100	10	100	10
State Prediction	Full word	82.8	10.7	99.8	10.04
	Integer bits	100	10	100	10
	Upper five bits	100	10	100	10
Particle Weights	Full word	69.8	12.88	96.6	10.22
Single Particle	Upper five bits	92.8	10.95	99.8	10.1
Burst Error					
Fault Location	Bitflip Location	Bit Error		Word Error	
		Coverage (%)	Average Detection Latency (ms)	Coverage (%)	Average Detection Latency (ms)
State Estimation	Full word	97	17.54	100	10.05
	Integer bits	100	10.05	100	10.19
	Upper five bits	100	10	100	10
State Prediction	Full word	96.3	20.71	100	10.11
	Integer bits	100	10.01	100	10
	Upper five bits	100	10	100	10
Particle Weights	Full word	78.5	24.4	98.3	10.87
Single Particle	Upper five bits	100	13.44	99.6	10.04

TABLE I: Soft Error Detection Results

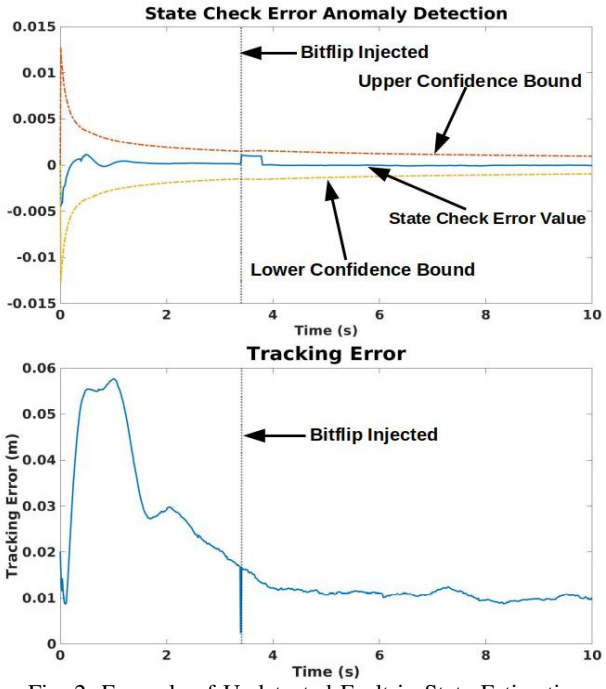

Fig. 2: Example of Undetected Fault in State Estimation

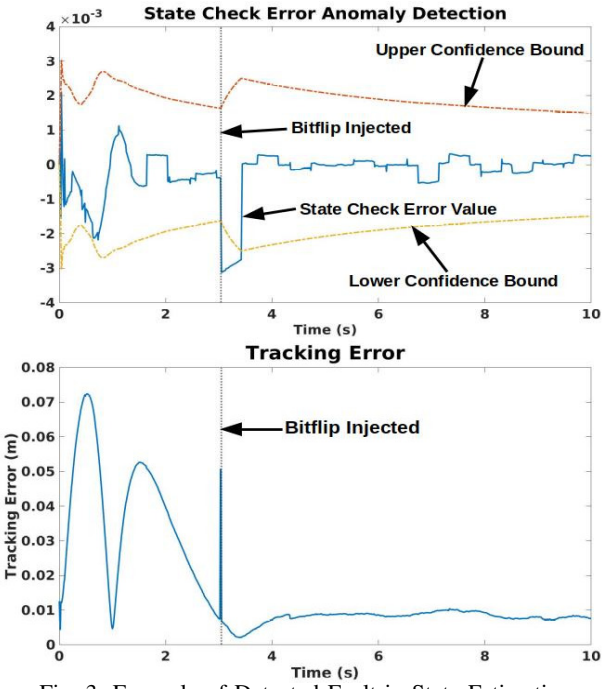

Fig. 3: Example of Detected Fault in State Estimation

and word errors are detected with near-total coverage. The detection latency ranges from 10ms to 20ms.

Bitflips in particle weights are injected over the full word for a single random particle weight. We inject these after state estimation prior to state prediction and measurement in the following time step. We see near-total coverage for burst and word errors with minimal latency in Table I, while the distortion in the distribution due to less significant bit flips does not trigger detection in the case of single bit flip transients as we see from the 69.8% coverage in Table I. This relation of erro detection to severity of errors is borne out by the greater coverage and lower latency seen for detection of transient word errors. This is similar to Figure 2 where the distortion in distribution estimation produced by the weight change may be insufficient to significantly affect vehicle operation. A greater detection latency and coverage for burst errors in these two cases is indicative of detection being caused by their impact on vehicle operation over time.

Median detection time in all cases was observed to be 10ms, or one sensor time step. The majority of the remainder of detections were accomplished in the next time step, with remaining errors causing errors in state estimation from gradual distribution drift and hence detected later. LSB burst errors caused more such drift, with correspondingly later detections. Correction of errors may be possible by restarting the particle filtering process with coefficients saved from the last error-free time step and we intend to explore this in future work.

V. Conclusions

In this paper we have presented a novel approach to error detection in nonlinear systems, and experimentally validated

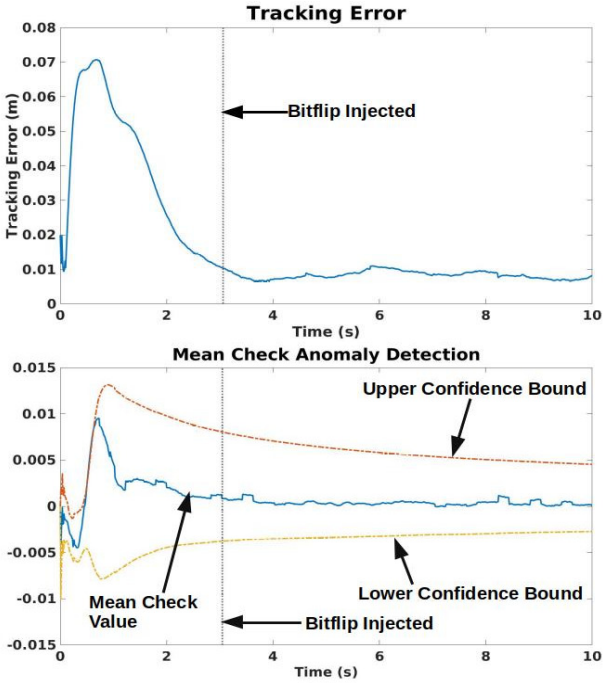

Fig. 4: Example of Undetected Single-Particle Fault

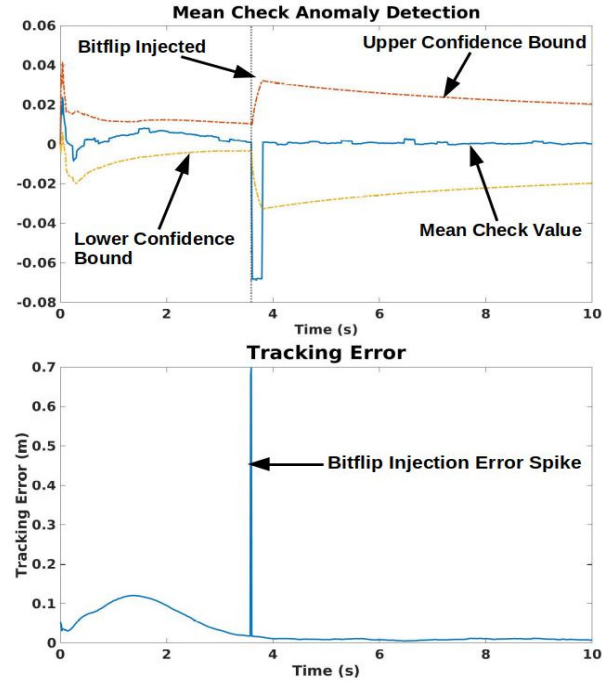

Fig. 5: Example of Single-Particle Detected Fault

our approach on the particle filter. Our experimental results show that a high degree of coverage with a very low detection latency can be achieved with this approach.

ACKNOWLEDGMENT

This research was supported by the Semiconductor Research Corporation under Auto Task 2892.001 and in part by the U.S. National Science Foundation under Grant S&AS:1723997.

REFERENCES

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[2] S. Konatowski, P. Kaniewski, and J. Matuszewski, "Comparison of estimation accuracy of ekf, ukf and pf filters," *Annual of Navigation*, vol. 23, Dec. 2016. DOI: 10.1515/aon-2016-0005.

[3] l. yi hao lo, M.-H. Chuang, M.-H. Cheng, and Y.-H. Huang, "Multiple-pe particle filter processor ic for mobile positioning systems," Nov. 2017, pp. 251–255. DOI: 10.1109/ICAM.2017.8242179.

[4] P. Engineer, R. Velmurugan, and S. Patkar, "Scalable implementation of particle filter-based visual object tracking on network-on-chip (noc)," *Journal of Real-Time Image Processing*, Mar. 2019. DOI: 10.1007/s11554-018-0841-5.

[5] H. Jiang, H. Zhang, R. C. Harrington, J. A. Maharrey, J. S. Kauppila, L. W. Massengill, and B. L. Bhuva, "Impact of supply voltage and particle let on the soft error rate of logic circuits," in *2018 IEEE International Reliability Physics Symposium (IRPS)*, Mar. 2018, pp. 4C.4-1-4C.4–4. DOI: 10.1109/IRPS.2018.8353586.

[6] E. Lampiri, "Sensor anomaly detection and recovery in a nonlinear autonomous ground vehicle model," in *2017 11th Asian Control Conference (ASCC)*, IEEE, 2017, pp. 430–435.

[7] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE transactions on computers*, vol. 100, no. 6, pp. 518–528, 1984.

[8] V. S. S. Nair and J. A. Abraham, "Real-number codes for fault-tolerant matrix operations on processor arrays," *IEEE Trans. Comput.*, vol. 39, no. 4, pp. 426–435, Apr. 1990, ISSN: 0018-9340. DOI: 10.1109/12.54836.

[9] S. Pandey, S. Banerjee, and A. Chatterjee, "Concurrent error detection and tolerance in kalman filters using encoded state and statistical covariance checks," in *Proceedings, IOLTS*, 2016, pp. 161–166.

[10] S. Banerjee, A. Chatterjee, and J. A. Abraham, "Efficient cross-layer concurrent error detection in non-linear control systems using mapped predictive check states," in *2016 IEEE International Test Conference, Fort Worth, TX, USA*, 2016, pp. 1–10.

[11] B. Shim, N. R. Shanbhag, and S. Lee, "Energy-efficient soft error-tolerant digital signal processing," in *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, vol. 2, pp. 1493–1497.

[12] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *Trans. Sig. Proc.*, vol. 50, no. 2, Feb. 2002. DOI: 10.1109/78.978374.

[13] J. S. Liu and R. Chen, "Sequential monte carlo methods for dynamic systems," *Journal of the American Statistical Association*, vol. 93, no. 443, pp. 1032–1044, 1998. DOI: 10.1080/01621459.1998.10473765.

[14] E. L. Lehmann and J. P. Romano, *Testing statistical hypotheses*, Third, ser. Springer Texts in Statistics. New York: Springer, 2005, ISBN: 0-387-98864-5.

[15] R. Rajamani, *Vehicle Dynamics and Control*. Jan. 2006. DOI: 10.1007/0-387-28823-6.