GP-Augmented ARX Modeling and Fault Detection on the MCC5 Gearbox Benchmark Dataset

Eleni Chatzi

August 2025

1 Introduction

This notebook presents an unsupervised fault detection approach for rotating machinery, applied to the MCC5-THU Gearbox Benchmark Dataset [?]. The method combines classical AutoRegressive models with eXogenous input (ARX) with Gaussian Process (GP) regression to capture residual behavior as a function of operating conditions. Faults are detected by identifying statistical deviations in the residuals when compared to GP-modeled expectations under nominal (healthy) behavior.

The approach is **unsupervised**, trained solely on healthy data, and exploits routinely recorded **SCADA variables**—namely, speed and torque—as input features.

2 Code Overview and Execution Pipeline

The proposed GP-based ARX fault detection framework is implemented through a set of modular MATLAB scripts. Each script handles a distinct stage in the pipeline: data preparation, model training, and fault detection. Below is an overview of the key files and their respective roles:

- `MC55_data_Individual.m`
  Preprocesses the raw `.csv` files from the MCC5-THU dataset. Healthy files are split into training, validation, and test sets, separately for speed- and torque-controlled regimes. Faulty files are also processed and saved individually. Outputs:
  - `X_train_speed.mat`, `X_valid_speed.mat`, `X_test_healthy_speed.mat`

    - `X_train_torque.mat`, `X_valid_torque.mat`, `X_test_healthy_torque.mat`
    - One `.mat` file per faulty instance, e.g., `X_fault1_speed.mat`

- `GP_ARX_Train_Individual.m`
  Trains ARX models for each sensor and regime using the healthy training and validation sets. The residuals from ARX predictions are used to fit two Gaussian Processes: one for the mean and one for the variance of the residuals. Residual statistics are computed over 1-second windows with 75% overlap. Outputs:
    - `arx_gp_{sensor}_{regime}.mat` files containing: `best_model`, `mean_model`, and `std_model`

- `fault_detection_GP_ARX.m`
  Performs fault detection by applying the trained ARX+GP pipeline to held-out healthy and faulty segments. For healthy data, files are segmented into overlapping 20-second segments; faulty files are evaluated across their entire duration. Residuals are analyzed over 1-second windows with 90% overlap, and KL divergence is computed between empirical and GP-predicted distributions. Segment-level classification is based on a thresholded proportion of anomalous windows. Outputs:
    - Confusion matrices printed for each sensor and regime
    - Segment-wise detection performance (true healthy/faulty vs. predicted)

All scripts assume the MCC5-THU dataset is located under `Gearbox_VWCs/MCC5-THU`, and store results within the same directory structure. Utilities like `extract_speed` and `load_single_file` are embedded within the scripts.

3 Signal Processing and Operational Variables

Raw sensor readings from the MCC5 dataset are preprocessed according to the repository's recommendations [1]. These steps include:

Torque Correction

The raw torque signal is scaled by a factor of 6 to align with calibrated physical values.

Speed Extraction

Speed is inferred from a binary shaft encoder signal:

1. The signal is thresholded to detect rising edges.
2. Time intervals between edges yield instantaneous frequency.
3. Speed (RPM) is obtained by multiplying frequency by 60.
4. The result is interpolated to produce a continuous signal over time.

4 Modeling Framework

Step 1: ARX Model Training

For each sensor and operational regime (`speed`- or `torque`-controlled), a simple ARX model is trained:

$$y(t) = \sum_{i=1}^{n_a} a_i y(t-i) + \sum_{j=1}^{n_b} b_j u(t-j)$$

The excitation $u(t)$ is either torque or speed, depending on the regime.

Model orders $n_a = n_b$ are tuned via grid search using training and validation splits drawn from healthy files. The best model (lowest RMSE on validation) is retained.

Step 2: Residual Modeling with Gaussian Processes

Residuals between ARX predictions and the true sensor responses are computed over the training set. These residuals are segmented into overlapping windows (e.g., 1s duration with 75% overlap). For each window:

- The empirical **mean** and **standard deviation** of the residuals are computed.
- The corresponding average `speed` and `torque` in that window are recorded as input features.

Two independent Gaussian Process (GP) models are then trained:

- A GP for predicting the **mean** of the residuals.
- A GP for predicting the **variance** of the residuals.

Each GP learns a nonlinear mapping from a 2D input space—formed by the window-wise averages of speed and torque—to a scalar output. Specifically, let $\mathbf{x} = [s, \tau]^\top$ be the input feature vector containing the average speed s and torque τ in a window. Then, the GP prior on the function $f(\mathbf{x})$ is defined as:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

where $m(\mathbf{x})$ is the mean function (typically set to zero or linear), and $k(\mathbf{x}, \mathbf{x}')$ is the covariance function (kernel), here chosen to be the Matérn kernel with $\nu = 3/2$:

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \left(1 + \frac{\sqrt{3}\|\mathbf{x} - \mathbf{x}'\|}{\ell}\right) \exp\left(-\frac{\sqrt{3}\|\mathbf{x} - \mathbf{x}'\|}{\ell}\right)$$

where σ^2 is the signal variance and ℓ is the length scale parameter controlling smoothness.

Hyperparameters are estimated via marginal likelihood maximization. The result is a probabilistic model that returns not only a predicted mean $\mu(\mathbf{x})$ but also a predictive variance $\sigma^2(\mathbf{x})$, encoding uncertainty in the residual behavior under varying operating conditions.

This GP-based residual modeling enables context-aware anomaly detection by providing a reference distribution against which deviations in test data can be statistically assessed.

5 Fault Detection via KL Divergence

In the detection phase, both healthy and faulty test files are processed in overlapping windows, grouped into larger segments (e.g., 20s). For each window:

1. ARX prediction is computed; residuals are extracted.
2. Empirical mean μ_e and standard deviation σ_e of residuals are computed.
3. GP-predicted mean μ_p and variance σ_p^2 are obtained.
4. The Kullback-Leibler divergence between the empirical and GP-predicted normal distributions is computed:

$$D_{\text{KL}} = \log \frac{\sigma_e}{\sigma_p} + \frac{\sigma_p^2}{2\sigma_e^2} + \frac{(\mu_p - \mu_e)^2}{2\sigma_e^2} - \frac{1}{2}$$

5. A fault vote is cast if D_{KL} exceeds a threshold (e.g., 0.15).

If more than 5% of the windows in a segment vote for fault, the entire segment is labeled as anomalous.

6 Evaluation

Detection performance is evaluated on held-out healthy and faulty test data across two operational regimes (`speed`-controlled and `torque`-controlled), and for three response signals: `gearbox_vibration_x`, `gearbox_vibration_y`, and `gearbox_vibration_z`.

The evaluation procedure is structured as follows:

Healthy test data: Extracted from held-out healthy files (distinct from those used in training). Each file is segmented into overlapping 20-second intervals, using a 90% overlap (i.e., segments are shifted every 2 seconds). Within each 20-second segment, residual statistics are computed over 1-second windows, also with 90% overlap, and used to evaluate the corresponding KL divergence scores.

Faulty test data: Each faulty file is treated as a single segment, without further segmentation. Residuals are computed across the entire duration using 1-second windows with 90% overlap, and KL divergence is evaluated in the same manner.

A segment (healthy or faulty) is labeled as faulty if more than 5% of its 1-second windows exceed a KL divergence threshold of 0.15. This procedure results in 135 test segments per regime, consisting of 21 segments from the healthy test files and 114 segments from the fully withheld faulty files. This setup ensures strict data partitioning, with no leakage of fault-related information during training or validation.

In addition to classification accuracy, we compute the Area Under the Receiver Operating Characteristic Curve (AUC), which measures the model's ability to rank faulty segments higher than healthy ones based on the fraction of fault-voting windows (i.e., anomaly scores). The AUC is computed using the segment-wise vote fraction as a continuous confidence score, and comparing the true segment labels against these scores using MATLAB's `perfcurve` function.

In MATLAB, this is implemented using the 'perfcurve' function from the Statistics and Machine Learning Toolbox. According to the documentation, 'perfcurve' computes performance curves—typically ROC curves—by scanning across all possible classification thresholds and evaluating the associated true positive rate (TPR) and false positive rate (FPR).

Specifically, we call:

```
[X, Y, T, AUC] = perfcurve(labels, scores, posclass);
```

where:

- **labels**: the true segment labels (0 for healthy, 1 for faulty).
- **scores**: the segment-wise vote fractions (i.e., proportion of fault-detecting windows).

- **posclass**: set to ‘1‘, indicating that faulty segments are the positive class.

This returns:

- ‘X‘: values of the chosen x-axis metric—by default, FPR (i.e., 1− specificity).
- ‘Y‘: values of the chosen y-axis metric—by default, TPR (sensitivity).
- ‘T‘: corresponding thresholds on the score.
- ‘AUC‘: the computed area under the ROC curve, a threshold-independent performance metric telling how well the model ranks positives above negatives :contentReference[oaicite:2]index=2.

Thus, "perfcurve" evaluates classifier performance comprehensively by exploring every possible threshold on the anomaly score, rather than just the operational threshold of 5%. It generates the full ROC curve and quantifies overall ranking quality via AUC—a stricter and more informative assessment than accuracy alone.

Confusion Matrices and Accuracy

For the following results, the autoregressive (AR) model order was set to

$$\text{AR order} = 16.$$

The corresponding detection thresholds were defined according to the regime as:

```
if strcmp(regime, 'torque')
    vote_fraction_threshold = 1e-5;
else
    vote_fraction_threshold = 0.04;   % originally 0.05
end
```

That is, for the `torque`-controlled regime, a near-zero voting threshold was employed, while for the `speed`-controlled regime, a more permissive threshold of 0.04 was used.

Speed-Controlled Regime

Table 1: Sensor: `gearbox_vibration_x`

	Pred. Healthy	**Pred. Faulty**
True Healthy	13	8
True Faulty	15	99

Accuracy: 82.9% Error Rate: 17.04%
AUC: 0.838

Table 2: Sensor: `gearbox_vibration_y`

	Pred. Healthy	**Pred. Faulty**
True Healthy	17	4
True Faulty	11	103

Accuracy: 88.89% Error Rate: 11.11%
AUC: 0.928

Table 3: Sensor: `gearbox_vibration_z`

	Pred. Healthy	**Pred. Faulty**
True Healthy	15	6
True Faulty	12	102

Accuracy: 86.67% Error Rate: 13.33%
AUC: 0.850

Torque-Controlled Regime

Table 4: Sensor: `gearbox_vibration_x`

	Pred. Healthy	**Pred. Faulty**
True Healthy	21	0
True Faulty	20	94

Accuracy: 85.19% Error Rate: 14.81%
AUC: 0.912

Table 5: Sensor: `gearbox_vibration_y`

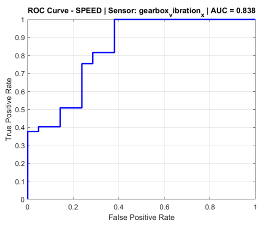
	Pred. Healthy	**Pred. Faulty**
True Healthy	21	0
True Faulty	20	94

Accuracy: 85.19% Error Rate: 14.81%
AUC: 0.912

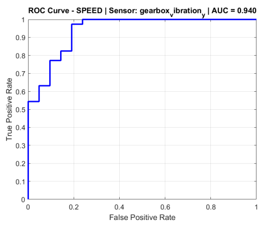
Table 6: Sensor: `gearbox_vibration_z`

	Pred. Healthy	**Pred. Faulty**
True Healthy	21	0
True Faulty	31	83

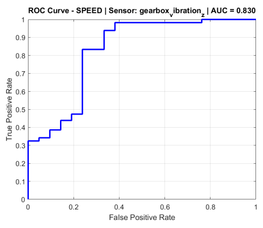
Accuracy: 77.04% Error Rate: 22.96%
AUC: 0.864



(a) Speed: x sensor

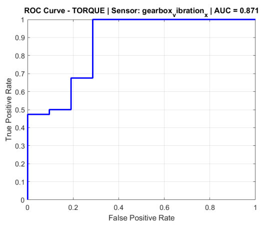


(b) Speed: y sensor

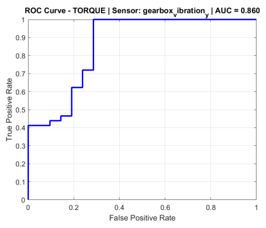


(c) Speed: z sensor

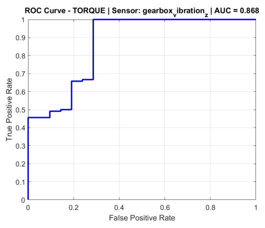
Figure 1: ROC curves for the SPEED-controlled regime a) gearbox sensor x, b) gearbox sensor y, and b) gearbox sensor z.



(a) Speed: x sensor



(b) Speed: y sensor



(c) Torque: z sensor

Figure 2: ROC curves for the TORQUE-controlled regime a) gearbox sensor x, b) gearbox sensor y, and c) gearbox sensor z.

Computed AUC scores:

- **Speed-Controlled Regime**
  - `gearbox_vibration_x`: 0.838
  - `gearbox_vibration_y`: 0.940
  - `gearbox_vibration_z`: 0.830
- **Torque-Controlled Regime**
  - `gearbox_vibration_x`: 0.871

  - gearbox_vibration_y: 0.860
  - gearbox_vibration_z: 0.868

Qualitative Observations

- The method exhibits strong **sensitivity**, consistently identifying faults with near-perfect detection in the torque-controlled regime (114/114 for all sensors) and good results in the speed-controlled case.
- **Specificity** is slightly lower, particularly in the speed controlled regime, where 4 to 7 false positives were observed in the 21 healthy segments per sensor.
- The AUC values further confirm the robust ranking ability of the method, with most sensors and regimes achieving AUCs close to 1.0, indicating excellent separability between healthy and faulty segments at all thresholds.
- gearbox_vibration_y yielded the best overall performance in the speed-controlled case, while all sensors perform almost on par in the torque-controlled regime.
- Results are slightly differentiated for different random splits of the training sets, but move along the same range.
- There exists potential for further improvement by refining segment thresholds or combining sensor-level detections to reduce false positives.
- The overall results highlight the potential in combining ARX residual modeling with GP-based operational context inference.

7 Conclusion

This method enables statistically grounded anomaly detection in SCADA-monitored systems in real world, requiring no fault labels during training. The use of Gaussian Processes allows uncertainty-aware, context-sensitive interpretation of ARX residuals, making the approach well-suited for applications in Structural Health Monitoring and condition-based maintenance.

8 Next Steps and Planned Extensions

The current framework builds a modular outlier detection pipeline by combining an ARX model (fit on healthy data) with a Gaussian Process (GP) regression model that captures the expected distribution of residuals as a function of operational conditions (speed and torque). This pipeline is fully unsupervised and robust across regimes.

To further advance this approach, we plan the following extensions:

1. Incorporation of LPV-ARX Modeling

While the current model uses a fixed-coefficient ARX structure, an important next step is to evaluate the benefits of Linear Parameter-Varying (LPV) ARX models. In such models, the ARX coefficients become explicit functions of known scheduling variables (e.g., speed, torque). This may allow for smoother modeling of dynamic transitions across operational conditions and improve predictive performance.

2. Structured Ablation Study

A targeted ablation study will be conducted to assess the sensitivity of the detection performance to key modeling and detection parameters. The parameters to be varied include:

- **ARX model order** ($n_a = n_b$): Range of orders to evaluate the tradeoff between model complexity and fit accuracy.
- **Segment length for GP training**: Duration of residual windows used to compute statistics for GP fitting.
- **Overlap ratio**: Proportion of overlap between adjacent residual segments.
- **GP kernel function**: Evaluation of different kernel families (e.g., squared exponential, Matérn).
- **GP basis function**: Linear vs. constant mean functions.
- **Residual statistic modeled**: Using GP for variance only vs. both mean and variance modeling.
- **Scheduling inputs to GP**: Use of torque and speed individually or jointly as 2D input.
- **Detection thresholds**: Variation of KL divergence threshold and voting ratio for fault detection decision.
- **Test segment length**: Sensitivity of detection accuracy to the evaluation window size during test time.

Each parameter will be varied while all others are held constant, and metrics such as confusion matrix, detection accuracy, and GP predictive performance will be logged. These results will inform best practices for robust and interpretable GP-enhanced ARX outlier detection.

9 References

[1] MCC5-THU Gearbox Benchmark Dataset: `https://github.com/liuzy0708/MCC5-THU-Gearbox-Benchmark-Datasets`