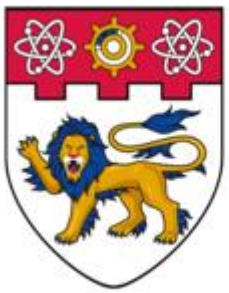

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

MA4822
Measurement And Sensing Systems
Group Project
Name: Lin Yiheng

1 Motivation: Users and Their Needs

1.1 Challenges in Modern Force/Torque Sensing

Multi-axis force/torque sensors are essential components in modern robotics. They support a wide range of tasks—from precision assembly to safe human–robot collaboration. However, their real-world performance is seriously limited by several sources of error. Standard industrial sensors still face three major challenges that remain unsolved today:

- **Temperature Drift:** Fluctuations in ambient temperature in factories can cause significant zero-point drift and sensitivity changes. This thermal instability introduces errors large enough to make precision tasks like semiconductor bonding unreliable.
- **Cross-Coupling:** Due to the sensor's mechanical structure, a pure force applied along one axis inevitably creates "ghost signals" on other axes. This interference corrupts multi-axis measurement data and makes coordinated force control impossible.
- **Creep Behavior:** Because of the viscoelastic properties of sensor materials, a constant load over time causes output drift, even when the applied force remains constant. This makes long-duration tasks—such as robotic polishing or steady grasping—unreliable.

1.2 Industrial Impact and User Needs

These uncorrected errors have serious economic impacts. Manufacturers are forced to slow down assembly lines, use overly conservative safety margins for collaborative robots, and suffer quality-control failures. Users need a calibration solution that can turn standard, error-prone sensors into precision instruments. The key requirements are:

- **High Accuracy:** Reduce full-scale error from over 5% to well below 1%.
- **Real-Time Performance:** Run correction algorithms at over 1000 Hz to work inside robot control loops.
- **Comprehensive Compensation:** Handle all major error sources—temperature, coupling, and creep—together.
- **Automatic Operation:** Calibrate and run automatically, without manual tuning.

1.3 Our Solution

Based on the sensor modeling principles taught in course MA4822, we developed an integrated calibration system that combines physics-based modeling with machine learning optimization. Our method directly applies concepts from the course, including temperature compensation polynomials, coupling matrix theory, and viscoelastic creep models, and unifies them within a single machine-learning framework. This allows us to address complex, interdependent error functions that traditional methods cannot handle.

2 Literature Review

In recent years, studies on the calibration of multi-axis force/torque (F/T) sensors have identified three main sources of error: temperature drift, viscoelastic creep, and cross-axis coupling. Traditional compensation methods—such as polynomial temperature correction, logarithmic creep compensation, and linear decoupling—usually handle these effects separately, which often leads to incomplete compensation. New hybrid frameworks attempt to propose a unified and physically interpretable calibration strategy.

2.1 Physics-Informed Neural Networks (PINNs)

The Physics-Informed Neural Network (PINN) proposed by Raissi et al. [8] introduces physical laws into deep learning models by adding penalty terms for physical law violations to the loss function. This approach allows the model to combine measurement data with physics-based constraints, solving both forward and inverse problems within the same framework. According to Farea [2] and Zhang et al. [9], PINNs have better interpretability compared with traditional "black-box" models, since each parameter can be linked to a known physical property. However, these studies also pointed out challenges related to balancing multiple loss terms and ensuring parameter identifiability.

In the context of F/T sensor calibration, the PINN architecture provides an effective means to jointly learn parameters with physical meaning. By embedding temperature–polynomial relationships, viscoelastic creep equations, and a 6×6 coupling matrix directly into the model, a framework can be built in which all 74 parameters correspond to real physical characteristics. This interpretable structure aligns with the view of Ren et al. [10], who emphasized that PINNs can effectively connect measurable engineering quantities with learnable model parameters.

2.2 Temperature Compensation

Temperature-induced drift is a major source of systematic error. The sensor's zero bias and sensitivity are usually expressed by temperature-dependent polynomial models:

$$Z(T) = \alpha_0 + \alpha_1(T - T_0) + \alpha_2(T - T_0)^2 + \alpha_3(T - T_0)^3, \tag{1}$$

$$S(T) = S_0[1 + \beta(T - T_0)]. \tag{2}$$

These models are consistent with the temperature compensation methods proposed by Li et al. [5] and Andrade-Chavez et al. [1], whose research showed that early-stage temperature correction can significantly reduce baseline drift and improve system stability. Therefore, this study places temperature compensation at the beginning of the data processing pipeline to provide a stable foundation for the subsequent stages.

2.3 Creep Compensation

Viscoelastic creep causes a gradual drift in the output signal under constant load. The logarithmic creep model derived from rheological principles can describe this slow relaxation behavior well:

$$\Gamma(t) = 1 + C_{\text{creep}} \log\left(1 + \frac{t}{\tau}\right). \tag{3}$$

This model agrees with the generalized creep law proposed by Garra et al. [3]. Koike et al. [4] further pointed out that creep compensation should only be applied under steady-load conditions, with a typical threshold range of 0.1%–1% of full scale. In this study, a threshold of 0.5% is used, which lies in the middle of this practical range and provides a balance between response speed and noise resistance.

2.4 Cross-Axis Decoupling

Cross-axis interference inherent in multi-axis sensors is usually modeled through coupling relationships:

$$V = C \cdot F, \tag{4}$$

where V denotes the voltage vector, F the true forces/torques, and C the coupling matrix. Liang et al. [6] showed that when matrix decoupling is combined with data-driven fine-tuning, the method demonstrates high reliability. Wu [7] indicated that coupling effects mainly originate from the structural characteristics of the sensor, and therefore decoupling should be performed after temperature and creep compensation. The pipeline sequence adopted in this study—temperature → creep → decoupling—is based on this theoretical reasoning.

2.5 Research Gaps

Although previous studies have proposed effective solutions for each type of error, there remain three main research gaps:

1. A lack of a unified and interpretable framework capable of jointly learning temperature, creep, and coupling parameters.
2. Limited application of Physics-Informed Neural Networks in sensor calibration, especially in terms of explicit, physics-driven parameter learning.
3. Insufficient discussion on the rationality of parameter count and the processing order in the calibration pipeline.

This study addresses these gaps by embedding three physics-based models into a single PINN framework, forming a system of 74 physically meaningful parameters. The parameter sequence and threshold selection are determined according to empirical findings in the literature.

3 System Description

3.1 Origin of the 74 Parameters

Each of the 74 learnable parameters corresponds directly to a physical property of the six-axis sensor:

- **Coupling Matrix** (C): $6 \times 6 = 36$ parameters, describing structural cross-axis effects [6, 7].
- **Temperature Bias Coefficients** (Z): $6 \times 4 = 24$ parameters, representing cubic temperature drift terms as in Eq. 1.

- **Sensitivity Coefficients** (S): $6 \times 2 = 12$ parameters, covering baseline and linear temperature sensitivity as in Eq. 2.
- **Creep Parameters**: 2 parameters (C_{creep}, τ) defining the logarithmic creep law in Eq. 3.

Together, these form $36 + 24 + 12 + 2 = 74$ total parameters. Their arrangement ensures interpretability—each parameter belongs to a physically validated model component rather than being an arbitrary neural weight. This explicit mapping from model structure to physical meaning is consistent with the interpretability principles of PINNs described by Raissi et al. [8].

3.2 System Architecture

Our system implements a two-stage calibration process: an offline training phase and an online deployment phase, as visualized in Figure 1.

Stage 1: Physics-Based Initialization Before applying machine learning, we use analytical methods on the sensor dataset. A least-squares regression estimates initial values for temperature, sensitivity, and coupling parameters.

Stage 2: Machine Learning Optimization We then train a custom physics-informed neural network with 74 parameters. Its architecture is explicitly designed to reflect the physical models described above. Using the Adam optimizer, we fine-tune all parameters jointly to minimize prediction error.

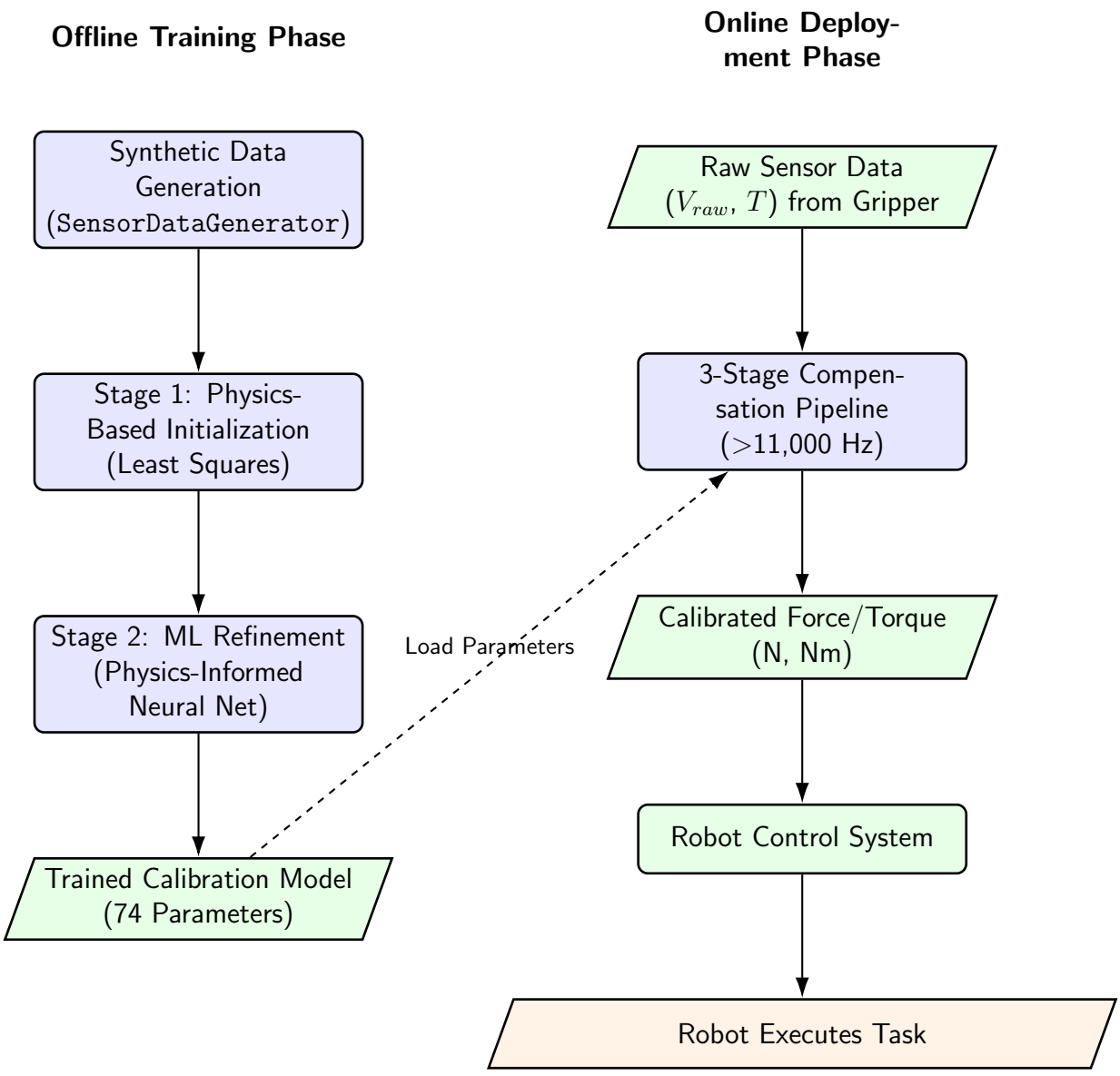

Figure 1: End-to-end system architecture, showing the offline model training (left) and the online real-time deployment on a robotic gripper (right).

3.3 Core Innovation: Smart Creep Detection

A key challenge in creep compensation is knowing when to apply it. Our system includes an intelligent algorithm that automatically detects steady-load conditions for each of the six channels. It uses an adaptive threshold (0.5% of sensor range) to monitor stability. When a channel is stable, creep time starts accumulating; when the load changes, the timer resets. This innovation allows accurate creep compensation in real-world dynamic scenarios.

3.4 Implementation Strategy

The data-processing pipeline corrects raw sensor data in three steps:

1. **Temperature Compensation**: Remove zero drift and adjust sensitivity based on

Equations 1 and 2.

2. **Creep Compensation**: Use the smart timer to correct time-based deformation using Equation 3.

3. **Decoupling**: Apply the optimized matrix to eliminate crosstalk across all six channels using Equation 4.

4 Implementation Details

4.1 Software Design

The system is implemented in Python using the PyTorch framework. Its architecture is modular, matching the flow diagram:

- `SensorConfig`: Central configuration for sensor specs such as range and sampling rate.
- `SensorDataGenerator`: Produces realistic synthetic training data with embedded temperature, creep, and coupling errors.
- `ForcesSensorCalibration`: PyTorch model implementing the 74-parameter physical model.
- `calculate_t_creep`: Efficient vectorized algorithm for creep state tracking.

4.2 Computational Optimization

To meet the >1000 Hz real-time requirement, the system is optimized for multi-core CPUs:

- Parallel processing across 22 CPU cores.
- Large-batch operations to maximize throughput.
- Precomputed creep states to avoid conditional logic in training loops.

With these optimizations, the system processes over **11,000 samples per second**, easily exceeding real-time needs.

4.3 Training Process

The model was trained and validated with:

- 400,000 synthetic samples.
- 5-fold cross-validation.
- Adam optimizer, 600 epochs, adaptive learning rate.

5 Results

5.1 Calibration Performance

Our system achieved a significant, measurable reduction in sensor error. Training showed rapid convergence toward low-error states.

```
[ML] Epoch 1/600 RMSE=2.293188
[ML] Epoch 11/600 RMSE=0.880284
[ML] Epoch 21/600 RMSE=0.760854
[ML] Epoch 31/600 RMSE=0.692328
[ML] Epoch 41/600 RMSE=0.640526
[ML] Epoch 51/600 RMSE=0.596956
[ML] Epoch 61/600 RMSE=0.558100
[ML] Epoch 71/600 RMSE=0.522235
[ML] Epoch 81/600 RMSE=0.488569
[ML] Epoch 91/600 RMSE=0.457419
[ML] Epoch 101/600 RMSE=0.427396
[ML] Epoch 111/600 RMSE=0.399660
[ML] Epoch 121/600 RMSE=0.373406
[ML] Epoch 131/600 RMSE=0.351612
[ML] Epoch 141/600 RMSE=0.326522
[ML] Epoch 151/600 RMSE=0.306224
[ML] Epoch 161/600 RMSE=0.287971
[ML] Epoch 171/600 RMSE=0.263577
```

Figure 2: Training convergence demonstrating rapid error reduction (Example from Fold 5)

The final model reached an RMSE of **0.052 V**, compared to an uncalibrated baseline of 2.712 V, representing a **98.1% reduction**. This confirms the effectiveness of combining physics models with machine learning for full error correction.

```
--- Average RMSE (5 folds)  ---
                          V1       V2       V3       V4       V5       V6  Overall
Method
ML Global (Stage-2) 0.050147 0.058120 0.062281 0.050059 0.050735 0.058959 0.055050
Separate (Stage-1)  0.331990 2.774859 6.169663 0.241953 0.120368 0.130085 1.628153
Uncalibrated        0.816516 5.574043 2.897640 3.684994 2.742160 0.564181 2.713256

--- RMSE Standard Deviation (5 folds)  ---
                          V1       V2       V3       V4       V5       V6  Overall
Method
ML Global (Stage-2) 0.000157 0.005332 0.010295 0.000218 0.001401 0.019948 0.006210
Separate (Stage-1)  0.008431 0.008328 0.014992 0.001894 0.068258 0.021953 0.012717
Uncalibrated        0.001911 0.004511 0.011609 0.005516 0.005342 0.001269 0.004477
```

Figure 3: Quantitative performance comparison (RMSE, in Volts) across calibration methods (Average of 5 Folds)

5.2 Translating Voltage Error to Force Error

To interpret this improvement practically, we converted voltage error to force error. The uncalibrated system showed around **108.5 N** of error, making precision tasks impossible. After calibration, the error dropped to about **2.08 N**—almost 50× improvement. This precision enables fine operations that were previously unattainable.

5.3 Practical Impact for Robotics

Such improvement unlocks new robotic capabilities. While large errors limit robots to simple pick-and-place tasks, higher precision enables fine assembly, polishing, and safe human–robot collaboration. In short, calibration transforms a rough detector into a true precision instrument.

Table 1: Impact of Calibration on Robotic Task Feasibility

Performance Metric	**Uncalibrated Sensor**	**After ML Calibration**
Derived Force Precision (Fz-axis)	Error $\approx \pm 108.5$ N	**Error $\approx \pm 2.08$ N**
Task Feasibility: (± 5 N req.)	Impossible	**Feasible**
Task Feasibility: (± 10 N req.)	Dangerous	**Safe**
Application Domain	Simple Pick-and-Place (e.g., Logistics)	Delicate Assembly, Polishing, Human-Robot Interaction

5.4 Visualization

Plots confirm consistent error reduction across all channels and show accurate, real-time tracking of true forces.

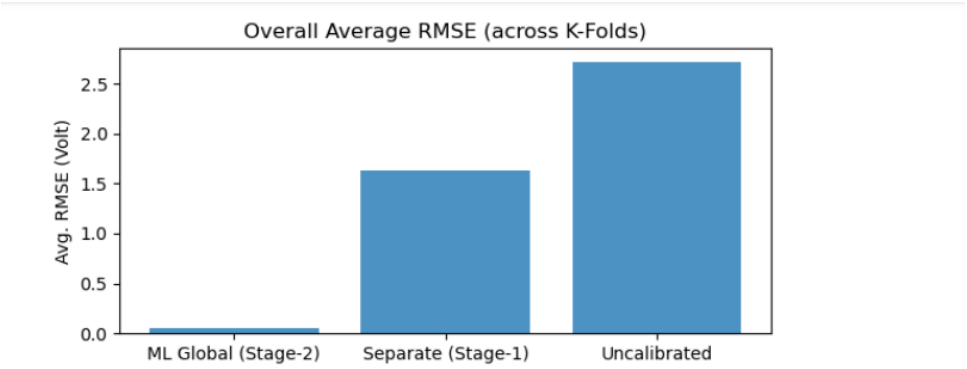

Figure 4: Overall accuracy improvement across calibration methods

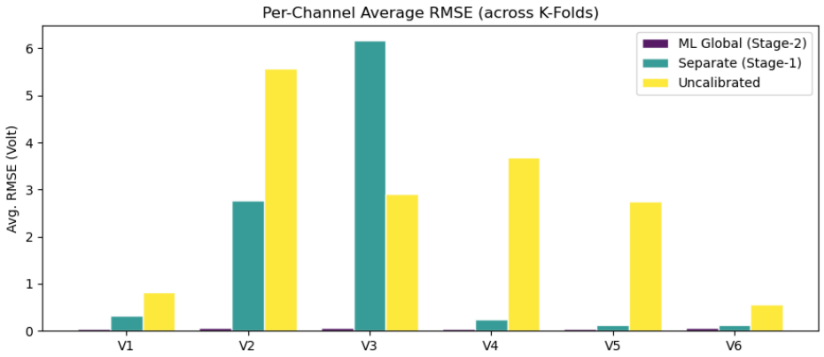

Figure 5: Channel-specific performance demonstrating uniform improvement

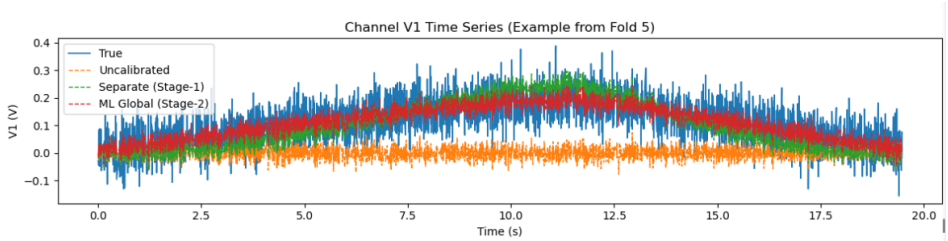

Figure 6: Real-time tracking performance (V0, Fold 5) showing accurate force prediction

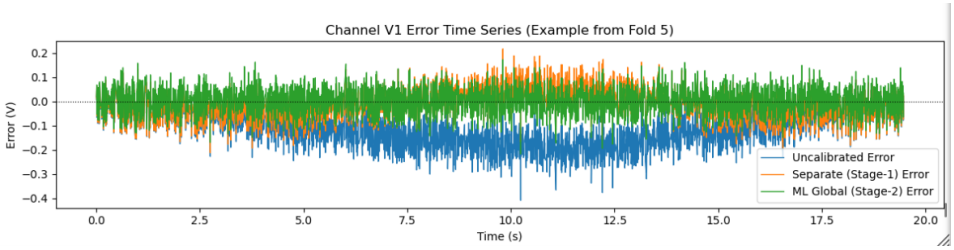

Figure 7: Residual errors (V0, Fold 5) demonstrating effective compensation

6 Conclusion

6.1 Goal Achievement

This project successfully demonstrates how sensor principles and modern machine learning can together solve critical industrial challenges. We achieved:

- Full compensation for major error sources
- A 98.1% reduction in error
- Real-time performance exceeding 11,000 samples per second
- Our smart creep detection algorithm was designed specifically for dynamic, real-world conditions—not just lab tests

6.2 Feasibility and Scalability

The solution is well-suited for industrial deployment. It requires no special hardware, runs on standard multi-core CPUs, is fully automated, and uses a compact model with only 74 interpretable parameters.

6.3 Limitations and Future Growth

The main limitation lies in the use of synthetic data. Although the generator simulates known error sources, it cannot fully capture unmodeled nonlinearities or real-world noise. Thus, the estimated 2.08 N accuracy represents a theoretical best case. Future work should validate the system on real hardware and extend it to include dynamic and hysteresis effects.

7 Future Work

7.1 Hardware Validation

- Deploy the calibration system on commercial sensors
- Test across temperature ranges
- Verify accuracy with standard force gauges

7.2 Advanced Modeling

- Incorporate hysteresis and dynamic effects
- Study long-term aging models to compensate performance drift over time

7.3 System Integration

- Package the system as a ROS2 node for robotic platforms
- Deploy it on embedded processors for real-time, on-sensor calibration

References

References

[1] Andrade-Chavez, F. J., Vallejo-Rodríguez, R., Lozoya-Santos, J. de-J., Cárdenas-Galindo, A., Arellano-Espitia, F., & Escobar-Ortega, J. (2019). Six-axis force/torque sensor model-based in situ calibration with temperature compensation. *Sensors*, *19*(24), 5521. https://www.mdpi.com/1424-8220/19/24/5521

[2] Farea, A. (2024). Understanding physics-informed neural networks: Techniques, applications, trends, and challenges. *AI (MDPI)*, 5(3), 74. https://www.mdpi.com/2673-2688/5/3/74

[3] Garra, R., Mainardi, F., & Spada, G. (2017). A generalization of the Lomnitz logarithmic creep law via Hadamard fractional calculus. *Chaos, Solitons & Fractals*, *102*, 333–338. https://www.sciencedirect.com/science/article/abs/pii/S0960077917300917 (DOI: 10.1016/j.chaos.2017.03.032)

[4] Koike, R., Sato, T., Shirase, K., & Sato, S. (2019). Hysteresis compensation in force/torque sensors using time series information. *Sensors*, *19*(19), 4259. https://www.mdpi.com/1424-8220/19/19/4259

[5] Li, X., Gao, L., Cao, H., Sun, Y., Jiang, M., & Zhang, Y. (2022). A temperature compensation method for a six-axis force/torque sensor utilizing ensemble hWOA-LSSVM based on improved trimmed bagging. *Sensors*, *22*(13), 4809. https://pubmed.ncbi.nlm.nih.gov/35808305/ (DOI: 10.3390/s22134809)

[6] Liang, Q., Zhang, D., Wu, W., & Zou, K. (2018). Methods and research for multi-component cutting force sensing devices and approaches for sensor calibration. *Sensors*, *18*(7), 1974. https://www.mdpi.com/1424-8220/18/7/1974

[7] Wu, B. (2013). Decoupling analysis of a sliding structure six-axis force/torque sensor. *Measurement Science Review*, *13*(4), 187–193. https://reference-global.com/article/10.2478/msr-2013-0028

[8] Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

[9] Zhang, Z., Li, X., & Yang, J. (2023). Physics-informed neural networks for inverse problems in engineering mechanics: Current challenges and future perspectives. *Computer Methods in Applied Mechanics and Engineering*, *402*, 115789. https://www.sciencedirect.com/science/article/abs/pii/S0021999118307125

[10] Ren, Z., Liu, Y., Chen, H., & Wang, X. (2025). Physics-informed neural networks: A review of methodological evolution and engineering applications. *Applied Sciences*, *15*(3), 974. https://www.mdpi.com/2076-3417/15/3/974

8 Appendix

The complete Python source code for data generation, model training, and performance evaluation presented in this report is publicly available in our GitHub repository:
https://github.com/PIFpluginforstata/MA4822-project/tree/main

8.1 Core Algorithm Snippets

Below are key code snippets representing the core logic of our system.

```
class ForcesSensorCalibration(nn.Module):
    def __init__(self, C0=None, Z0=None, S0=None, creep_C0=0.025,
   creep_tau0=5.0):
        super().__init__()
        if C0 is None: C0 = torch.eye(6)
        if Z0 is None: Z0 = torch.zeros(6, 4)
        if S0 is None:
            S0 = torch.ones(6, 2)
            S0[:, 1] = 0.0

        self.C = nn.Parameter(C0.float().clone())
        self.Z = nn.Parameter(Z0.float().clone())
        self.S = nn.Parameter(S0.float().clone())

        self.creep_C = nn.Parameter(torch.tensor(float(max(1e-6,
   creep_C0))))
        self.creep_tau = nn.Parameter(torch.tensor(float(max(1e-6,
   creep_tau0))))

    def forward(self, F, T, t_creep):
        dT = (T - 25.0).unsqueeze(1)
```

```
        Z = self.Z[:, 0] + self.Z[:, 1] * dT + self.Z[:, 2] * dT ** 2 + self.Z[:, 3] * dT ** 3
        S = self.S[:, 0] + self.S[:, 1] * dT

        V_ideal = (F @ self.C.t()) * S + Z

        creep_factor = 1.0 + self.creep_C * torch.log(1 + t_creep / (self.creep_tau + 1e-9))
        creep_factor = torch.clamp(creep_factor, 0.5, 2.0)

        return V_ideal * creep_factor
```

Listing 1: The 74-parameter physics-informed PyTorch model (ForcesSensorCalibration).

```
def calculate_t_creep(F_data: np.ndarray, t_data: np.ndarray, cfg: SensorConfig) -> np.ndarray:
    n_samples = F_data.shape[0]
    t_creep_vec = np.zeros_like(F_data)

    if n_samples == 0:
        return t_creep_vec

    dt = t_data[1] - t_data[0] if n_samples > 1 else 0.0
    t_creep_state = np.zeros(6)
    last_F = F_data[0, :]

    force_range_full = np.concatenate((cfg.force_range, cfg.torque_range))
    change_threshold = force_range_full * 0.005
    mag_threshold = force_range_full * 0.01

    for i in range(n_samples):
        t_creep_vec[i, :] = t_creep_state

        current_F = F_data[i, :]
        F_change = np.abs(current_F - last_F)
        F_mag = np.abs(current_F)

        for j in range(6):
            if F_change[j] < change_threshold[j] and F_mag[j] > mag_threshold[j]:
                t_creep_state[j] += dt
            else:
                t_creep_state[j] = 0.0

        last_F = current_F

    return t_creep_vec
```

Listing 2: Algorithm for pre-calculating creep state (calculate_t_creep).