

CONDITION MONITORING FOR LIFT SYSTEMS

Team Members

Meghna Sinha

Nipun Rautela

Anosh Damania

Sanjay

Pinni Venkata Abhiram

Guide

Geetha S.

School name: School of Computing Science and Engineering

Faculty name (ERP No): S.Geetha, (50587)

Academic Year: 2023-2024 (Winter Semester 23-24)

Title of the project:

Condition Monitoring for Lift Systems

Number of students involved in this project: 5

Table of Contents	Page No.
Chapter 1: Introduction	
1.1 Abstract	5
1.2 Problem Definition	5
1.3 Motivation	6
Chapter 2: Methodology	
2.1 Project Component Specification	7
2.2 Hardware Utilized	10
2.3 Parameters Analyzed	11
2.4 Monitoring with R-charts	13
2.5 Other Analysis Methods	15
Chapter 3: Architecture	
3.1 Predictive Condition Monitoring	16
3.2 Modules	29
Chapter 4: Implementation	
4.1 Load Cell Configuration	32
4.2 Ultrasonic Sensor Configuration	33
Chapter 5: Results	35
Chapter 6: Outcomes	37

Chapter 7: Single Unique Factor	39
Chapter 8: Site Visit	40
Chapter 9: User Benefits	41
Chapter 10: Sample Code	
10.1 Ultrasonic sensor code	42
10.2 Load cell code	44
Chapter 11: References	52

Chapter 1: Introduction

1.1 Abstract

The vertical transportation systems we commonly know as elevators play a crucial role in our modern urban landscape. They facilitate the movement of people and goods within buildings and high-rise structures, making them an indispensable part of our daily lives. It is imperative to ensure the dependable and safe functioning of elevators since any unexpected malfunction can lead to inconvenience, safety hazards, and financial repercussions.

To address these concerns, elevator condition monitoring systems have become pivotal technologies. They proactively manage the health of elevators, enhancing safety and optimizing maintenance operations. This report aims to provide a comprehensive understanding of elevator condition monitoring, shedding light on its significance, advantages, underlying technologies, and implementation strategies. Our project, by utilising various sensors delves into condition and prevention monitoring for lift systems, going into detail about the sensors to be embedded and the monitoring methods which can be applied on a particular system.

1.2 Problem Definition

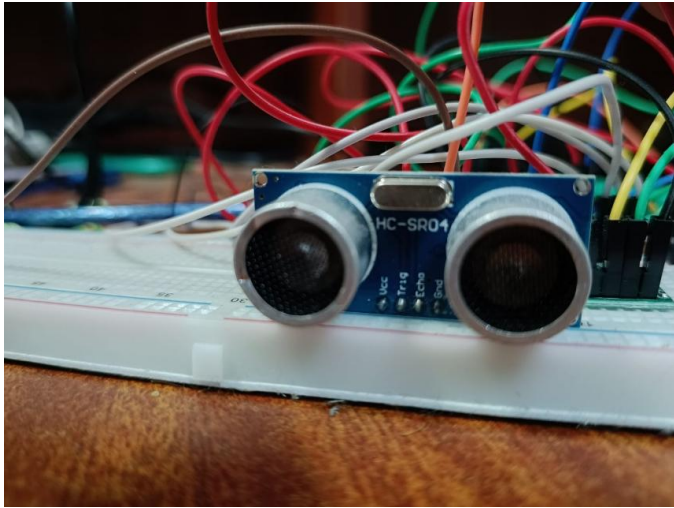

The current challenge in lift systems lies in the absence of proactive monitoring mechanisms, leading to unexpected failures, safety risks, and financial consequences. The lack of real-time insights into elevator health, exacerbated by factors such as wear and tear, environmental conditions, and component failures, underscores the critical need for an advanced condition monitoring system. Addressing this gap is imperative to ensure the reliable operation of elevators, enhance safety, and optimize maintenance operations, ultimately mitigating inconvenience and minimizing economic impact.

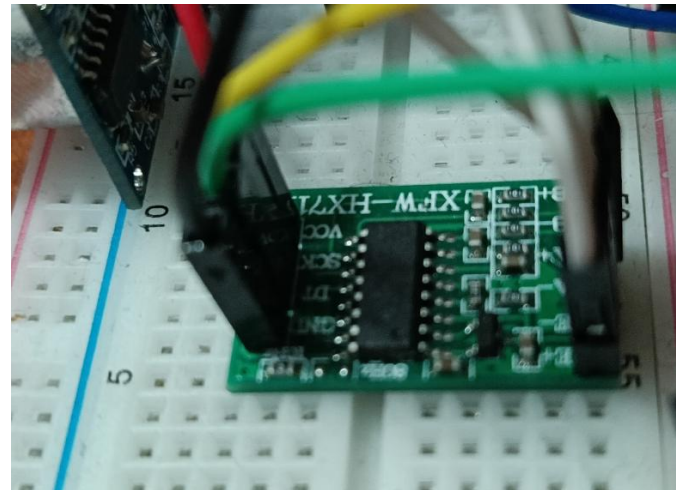
1.3 Motivation

- Enhance elevator safety through early issue detection, reducing the risk of unexpected failures and ensuring a secure vertical transportation experience.
- Reducing inconvenience caused by elevator downtime and minimizing disruptions for a seamless user experience.
- Cut maintenance costs, extend component lifespan, and optimize energy usage, resulting in significant long-term savings.
- Provide a reliable vertical transportation service, prioritizing user satisfaction by minimizing downtime and unexpected disruptions.

Chapter 2 Methodology

2.1 Project Component Specification

Component	Product Specification
 Ultrasonic sensors	Model: HC-SR04 Operating Voltage: 5V DC Detection Range: 2 cm to 400 cm Resolution: 3 mm Output Interface: Digital signal (Echo pulse) Trigger Signal: 10 μ s TTL pulse
 Load cells	Strain Gauge Load Cells Rated Capacity: 50kg Material: Stainless Steel Sensitivity: 2 mV/V Operating Temperature: -10°C to 80°C Excitation Voltage: 5V Protection Class: IP67



HX711 Amplifier

HX711 Amplifier
 Type: Precision 24-bit Analog-to-Digital Converter (ADC)
 Amplification Gain: Adjustable (128, 64)
 Operating Voltage: 2.6V to 5.5V
 Communication: Serial (SPI)
 Onboard Voltage Regulator: Yes



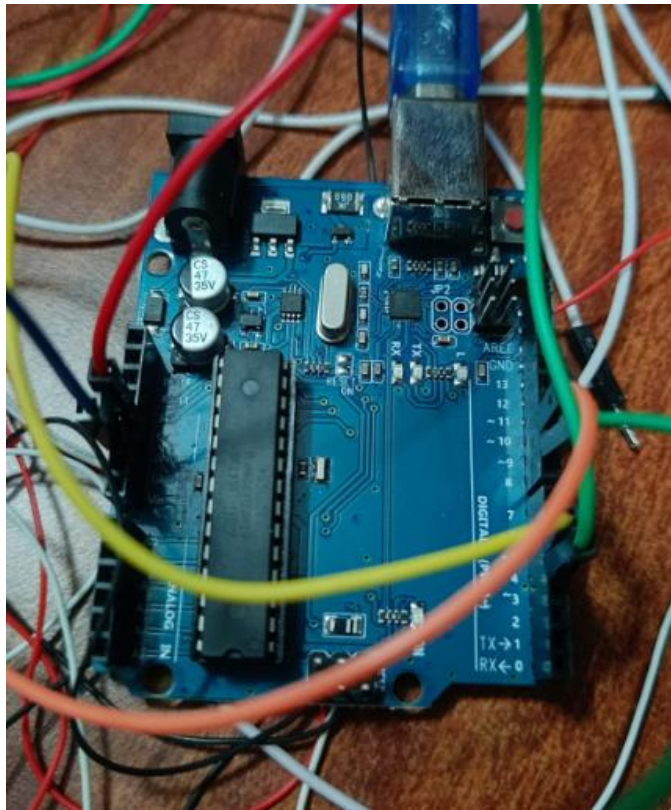
Camera

ESP32-CAM
 Microcontroller: ESP32
 Image Sensor: OV2640 (2 MP)
 Operating Voltage: 5V
 Wireless Connectivity: Wi-Fi (802.11 b/g/n)
 GPIO Pins: 9 (General Purpose Input/Output)
 Flash Memory: 4 MB
 Camera Interface: I2C, SPI
 Camera Resolution: 1600 x 1200 pixels (JPEG output)



Accelerometer

MEMS Accelerometer
 Axes: 3 (X, Y, Z)
 Acceleration Range: $\pm 4g$
 Sensitivity: 16384 LSB/g
 Operating Voltage: [Specify voltage, e.g., 3.3V or 5V]
 Communication: I2C or SPI
 Output Data Rate: Adjustable (e.g., 100 Hz, 200 Hz)
 Operating Temperature Range: -40°C to 85°C



Arduino

Arduino Uno
 Microcontroller: ATmega328P
 Operating Voltage: 5V
 Digital I/O Pins: 14 (including 6 PWM outputs)
 Analog Input Pins: 6
 Flash Memory: 32 KB (ATmega328P)
 Clock Speed: 16 MHz
 Communication: USB, UART, SPI, I2C

2.2 Hardware Utilization

Ultrasonic Sensors:

Ultrasonic sensors utilize high-frequency sound waves for precise object detection and distance measurement. By emitting ultrasonic pulses and measuring their return time after hitting an object, these sensors find applications in proximity sensing, obstacle avoidance, and fluid level measurements.

Half Bridge Load Cell (2x per Apparatus):

A strain gauge-based sensor, the half bridge load cell is commonly employed to measure force or weight. Comprising a Wheatstone bridge circuit, it exhibits sensitivity to mechanical strain, making it suitable for applications such as weighing scales, tension measurements, and force monitoring in industrial equipment.

Hx711 Amplifier:

The Hx711 Amplifier is an electronic device essential for amplifying low-level signals from sensors or transducers. This component plays a crucial role in signal conditioning, ensuring weak sensor output signals are boosted to a usable level for data acquisition and processing in applications like strain gauge measurements and load cell systems.

Camera:

Cameras, equipped with sensors, serve as optical devices capturing still images or recording videos. Widely used in photography, surveillance, and scientific research, cameras are indispensable tools for visually documenting information and recording events across various fields.

Accelerometer:

Accelerometers are motion sensors measuring acceleration, enabling the detection of changes in speed or direction. Commonly found in smartphones, vehicles, and industrial equipment, these devices contribute to features like screen orientation, crash detection, and vibration analysis.

Arduino:

Arduino, an open-source microcontroller platform, empowers users to create interactive electronic projects. With a user-friendly programming environment, it controls a variety of sensors, actuators, and devices, making it a popular choice for hobbyists, students, and professionals in the field of electronics and embedded systems.

2.3 Parameters Analyzed

Wear and Tear

Ultrasonic sensors emerge as pivotal tools in elevating the condition monitoring of elevator shafts, proficiently detecting wear, tear, and the onset of rust. Through strategic placement along the shaft, these sensors continuously measure surface thickness and integrity. As the elevator moves, the sensor emits high-frequency sound waves, which bounce back off the shaft's surface. Inconsistencies in the reflected signal, such as decreased material thickness or the presence of rust, serve as indicators of deterioration.

Regular analysis of ultrasonic data allows for tracking changes over time, establishing a baseline for the shaft's condition. This proactive monitoring approach facilitates early identification of potential issues, enabling timely interventions that prevent further damage, enhance elevator safety, and extend equipment lifespan. Ultrasonic sensors, owing to their non-invasive and high-precision capabilities, offer an efficient and cost-effective solution for structural health monitoring in elevators.

Fraying

Detecting fraying on elevator wires is paramount for maintaining safety and preventing failures. Our cutting-edge solution utilizes high-resolution cameras strategically placed within elevator shafts and cabins. Continuously capturing

images of the wires, these cameras employ advanced image processing and computer vision algorithms to identify irregularities and signs of wear.

The image processing algorithms can detect subtle changes in wire texture, color, or alignment—key indicators of fraying or damage. They recognize fine details in wire strands, highlighting any deviations from the expected pattern. Color and contrast analysis reveal discolorations or variations along the wire, indicating potential corrosion or degradation.

Regular monitoring through this camera-based system ensures prompt detection of wire issues, allowing for timely maintenance interventions. This proactive approach minimizes the risk of accidents, equipment malfunctions, and costly downtimes, enhancing the overall safety and reliability of elevator systems.

Leveraging camera technology for wire condition monitoring proves to be a cost-effective and efficient method that complements traditional preventive maintenance practices. This approach not only extends the lifespan of elevator systems but also provides peace of mind to passengers and facility managers alike.

Load

The load directly influences the risk of future mechanical failures, as elevators operate under specified load limits to ensure safety. However, over time, components experience wear and tear, altering the system's load capacity. To prevent catastrophic failures, it's crucial to regularly assess the maximum load the system can handle. Weekly or daily evaluations may suffice, with additional assessments triggered by defects, wear, tear, or post-maintenance fixes.

Continuous load measurement to ensure it stays within safety limits, established based on comprehensive parameter readings. While considering various parameters, emphasis is placed on critical factors such as the

condition of the main wire, given its significance over less impactful metrics like slight deviations in acceleration. Direct damage to physical components is closely monitored, recognizing that even minor impacts can lead to major breakdowns and safety risks. A statistical model will be developed to calculate the maximum load capacity, ensuring the elevator system remains reliable and safe.

Acceleration

Accelerometers, specialized sensors measuring acceleration, play a pivotal role in elevator system health monitoring. These sensors are strategically placed on critical components like the cabin, counterweight, and machinery. Continuous monitoring of acceleration data allows engineers to perform vibration analysis, a key diagnostic tool.

Through this analysis, we can identify unusual or excessive vibrations, which often signify underlying issues such as misalignment, imbalances, or wear in the mechanical components. The early detection facilitated by vibration analysis enables a pre-emptive maintenance approach, allowing for timely interventions to address potential problems before they escalate. This proactive strategy not only prevents breakdowns but also ensures the smooth and safe operation of the elevator system. Placing accelerometers strategically provides a comprehensive and effective means of monitoring the health and performance of the entire elevator system

2.4 Monitoring with R-charts

R-charts, also known as range charts, are a statistical tool used in quality control to monitor the variability of a process. In the context of our project, we use R-charts to track the range of variation in sensor readings over time.

Working:

Data Collection:

Gather data from vibration and ultrasonic sensors in the lift systems. Collect samples at regular intervals, capturing the variability in sensor readings.

Calculate Ranges:

For each sample, calculate the range of values between the maximum and minimum readings from both types of sensors.

Establish Control Limits:

Define upper and lower control limits based on historical data or acceptable thresholds for sensor variation. These limits help identify when the process is going out of control.

Plot R-chart:

Plot the calculated ranges over 'time' on the R-chart, with the x-axis representing time and the y-axis representing the range of values.

Monitoring:

Regularly monitor the R-chart for any points that fall outside the control limits. These points indicate potential issues with sensor readings or variations that may require attention.

Analysis:

Investigate and analyze the points beyond control limits. Sudden spikes or trends in the R-chart can indicate changes in the lift system's condition that may need further investigation.

Response:

Now we can take appropriate actions based on the analysis. This might involve calibrating or replacing sensors, performing maintenance, or addressing other factors affecting sensor readings.

Continuous Improvement:

Use insights gained from the R-chart to improve the condition monitoring process continuously. Adjust control limits or update procedures to enhance the effectiveness of the monitoring system.

2.3 Other Data Analysis Schemes

Time-Frequency Analysis:

Techniques such as Short-Time Fourier Transform (STFT) or Wavelet Transform can be applied to analyze how the frequency content of the sensor data changes over time. This can provide insights into transient events or frequency shifts indicating potential issues.

Statistical Process Control (SPC):

Besides R-charts, other SPC tools like X-bar charts (for monitoring the mean) and control charts for individual measurements can be used to detect abnormal variations in sensor readings.

Pattern Recognition:

Utilize pattern recognition algorithms to identify specific patterns or signatures associated with normal and abnormal operating conditions. This can enhance the system's ability to detect subtle changes indicative of potential problems.

Chapter 3 Architecture

3.1 Predictive Condition Monitoring

Data Collection:

We collected data in real time from Load Cells and Ultrasonic Sensor.

Sensor Integration:

Couple the data acquisition system with load cells and ultrasonic sensor. Ensure that appropriate linkages and settings are made for live communication of data from sensors to the software.

Sensor Calibration:

Ensure the calibration of load cells and ultrasonic sensors for reliable and precise readings. It is important to conduct the calibration step in order to obtain reliable data for predictive modeling.

Data Logging Setup: Have a rigorous data logging system for recording the sensory values after pre-specified time intervals. Make sure to date-stamp every record in order to preserve the chronological nature of the information. With this, the microcontroller such as “Arduino” can link with the sensors.

Streaming Protocol: Select a suitable continuous streaming protocol. As for example, one can transmit serial data streams through a USB cable if one uses an Arduino in combination with the mentioned sensors. The software used for capturing and processing such data should therefore be developed.

Data Storage: Create a data storage tool that can contain the data. Such forms could include but not limited to databases and flat files. The storage solution should be scalable to support large volumes of data.

Data Formatting: Cleaning collected data ready for machine learning processing. Make sure that your model's input is in the same format as the data, and ensure the right time-series lengths are used.

Real-time Preprocessing: Create on-the-go pre-processing techniques to purge the incoming data, replace missing elements plus any other preparatory steps and scaling if they are required. In this case, it provides the assurance that only good-quality predictive data gets included in the predictive model.

Continuous Data Flow: Establish a continuous data flow strategy that supplies the preprocessed data into the LSTM network. This entails creating circularity that constantly watches the sensor input, pre processes, and then sends to the LSTM model for forecasting.

Through proper attention directed towards these aspects of data collection and integration, we lay a good basis for predictive condition monitoring, which means the software must be able to process real time sensors into the machine learning pipeline for correct prediction.

2. Preprocessing:

This step basically ensures the cleaning of data and formatting the Sensor Data for Predictive Modeling.

Handling Missing Values:

Ensure that all values are captured in the obtained data. Use filling techniques that include interpolation and imputation depending on the type of missing data. Check that the integrity of the dataset is not compromised due to missing values.

Outlier Detection and Removal:

Spot and manage outliers in the dataset. The inclusion of outliers in this may impact negatively on the training process affecting how well a model performs.

Outliers can be identified and removed using techniques such as z-score analysis or IQR.

Normalization:

Normalize the input vector so that each feature is scaled to the same level. Normalization improves the speed of convergence during the training and the process prevents a certain feature from governing the learning process. They involve Min-Max scaling and Z-score normalization, among other forms of common normalization techniques.

3. Time Series Sequencing:

Creating sequences of data for time series forecasting with LSTMs. The sequential collection of data points from continuous sampling taken in equal times steps. Here, it entails arranging the data as an input-output pairs in which the input comprises of prior readings from sensor while output denotes the values to be predicted.

Lag Features:

Implement a number of lag features in order to replicate historical trends from the underlying data. Instead, add the values of earlier time steps as additional attributes on each sensor reading. This assists the LSTM model to capture the temporal features and the trends of the data.

Rolling Statistics:

Calculate the rolling statistics (rolling mean, rolling standard deviation), and so on, in order to portray the behaviour of the system at any given point in time. That means computing of statistical values involving an earlier set of data sample, and giving indications as to what is trending lately.

3. Data Splitting for Training and Testing:

Train-Test Split:

Split the pre-processed data set into a training and test set. In most cases, a good chunk of the data is utilized in training the model while a different set is often reserved for assessing its accuracy. Moreover, leave one more validation set for the process of tuning the hyperparameters while training.

4. Documentation and Logging:

Logging Preprocessing Steps: Record the pre-treatment performed on the data set. This documentation is critical to transparency, the ability to replicate and debugging. Record mean and standard error data particularly in case you use the figures for normalizing your outcome.

The software also does this by properly conditioning (preprocessing) the input data, which makes the trained LSTM model learn properly the complicated patterns and correlations within the sensor readings for more reliable timeline prediction.

5. Feature Engineering:

Extracting Relevant Features from Load Cells and Ultrasonic Sensor Data:

Load Cell Features:

- **Raw Load Cell Readings:** The primary variables of the data set should be these raw load cell measurements.
- **Rate of Change:** Acquire dynamic system behavior through computing of rates of change in load cell readings.
- **Cumulative Load:** Combine load cell data from several periods together and get cumulative load data.

Ultrasonic Sensor Features:

- **Distance Readings:** Consider the raw distance measurements from the ultrasonic sensor as crucial characteristics.
- **Velocity:** Estimate the speed of any object or surface using the relationship between the difference in distance and duration.
- **Presence Indicator:** Provide binary indicators using previously defined thresholds to indicate whether an object is within the sensing space or not.
- **Time Series Feature Engineering:**
 -
- **Lag Features:** Lag features of the load cell and ultrasonic data are added during time series feature engineering. The lag features are based on a past memory of how a system behaves and assist in determining the pattern of the temporal dependencies that surround it.
- **Rolling Statistics:** Keep providing the model with up to date information about the rolling statistics on the new lags that have recently been added.

Additional Features for Improved Predictions:

- **External Variables:** Add any other possible external variables that can influence the condition of the system if possible. Take a case where the readings of the temperature or humidity may be influenced by these environmental factors as they affect the readings of the load cell and the ultrasonic sensors.
- **Time-Related Features:** If applicable, introduce time indicators such as day of the week, hour of the day, and seasonality indicators into the system under consideration.

6. Train-Test Split:

Partitioning the Dataset for Model Training and Evaluation.

Dataset Division:

The preprocessing dataset should be split into separate segments for training, testing, and sometimes a validation stage.

Training Set:

The LSTM model uses this part of the data set for its training purposes. Therefore, the model must be able to identify patterns, trends, and interdependencies existent in the past sensor inputs.

Testing Set:

During the training, a distinct testing set is always retained in order to ensure that it does not interfere with the process of learning. It is used as a hidden set of test data to check the model's ability to predict outcomes on newly encountered, previously unseen data.

Validation Set (Optional):

While training their models, some data might be earmarked for validating processes. This set is used to refine hyperparameters and avoid overfitting.

Strategies for Splitting:

- **Temporal Split:** Ensure that the training set has prior temporal values and the test set includes latest temporal values if the data displays temporal order. The model does this in a manner that parallels actual situations, anticipates the model will make forecasts about future states using past observation.
- **Random Split:** A random split is ideal for unordered datasets. Nevertheless, a lot of care should be taken in order not to provide leaked data or create accidental biases.

7. Time Series Considerations:

Sequential Splitting:

It is important to preserve the temporal sequence of time series data during split of series. The future information must not leak into the respective training phases, so each training sequence should be followed by its corresponding testing sequence.

Sequence Length Determination:

Determine the optimal length of sequencing in the LSTM model. Here, we take into account the number of previous timesteps for computing the subsequent step. The experimentation on and analysing the nature of the time-series can assist in choosing the most excellent sequence length.

8. LSTM Model Architecture:

Designing the Long Short-Term Memory (LSTM) Neural Network:

Input Layer Configuration:

Design an input layer that accepts sequence of sensor data. This layer consists of the number of neurons that vary with respect to the selected sequence length and the number of features within every time step.

LSTM Layers:

Use several numbers of LSTM layers to make sure that you will be capturing the temporal dependencies properly. However, the specific number of LSTM layers and number of neurons per layer is dictated by the complexity of the data and the respective task.

Dropout for Regularization:

Dropout layers must be used after each LSTM layer in order to avoid over fitting of the model. It randomly drops a certain percentage of neurons during training in order to promote robustness and generalization.

Dense Output Layer:

Dense output layer with a single neuron on time series. For example, in regression tasks, the activation function for this layer can be linear.

Model Hyperparameters:

Number of Neurons: Try different numbers of neurons in each layer and balance model's complexity with computation time effectiveness.

Learning Rate:

Select a suitable learning rate for the optimizer (for example, Adam). Therefore, a smaller learning rate might be needed for a stable converge.

Number of Epochs:

Define the number of iterations for training, given the training statistics and likelihood of training overfitting. Such early stopping mechanisms might stop training where a model's performance saturates on the validation set.

Batch Size:

Consider training efficiency versus constraints by optimizing the batch size.

9. Compile the Model:**Loss Function:**

Ensure that you choose an appropriate loss function e.g. Mean-Squared Error(MSE) for the regression task. This is a measuring tool that determines if observed values match what was forecasted earlier.

Loss Function Selection:

Select a loss function that aligns with the predictive task. MSE is a common metric for time series forecasting and regression tasks. Some of these requirements might require adjustments.

Optimizer:

For instance, use an ADAM or RMS-prop optimiser in order to update the model weights during training so as to decrease the loss function.

Optimizer Selection:

Choose an optimiser to tune the model weights during training. Among others, the most common of such methods are ADAM, RMSprop, and SGD (stochastic gradient descent). It depends on the characteristics of the dataset as well as upon the training dynamics.

Learning Rate Setting:

Start with a specific default learning rate for the optimizer. Learning rates control how much each update changes weights. Due to its dependence on the model's convergence characteristic, it could be subjected to an adjustment.

Metrics Specification:

Include definitions for evaluation metrics (what to monitor) during training and during tests. Popular options in this context include MAE, MSE, and customized metric formulations depending on specific requirements.

10. Model Summary and Visualization:**Summary:**

Show a short description of the model structure and the size of parameters for each layer. This offers a glimpse of the model's sophistication.

Visualization:

Show the model architecture in diagram form and how information flows through the various layers of neural network. One can use visualization tools such as TensorBoard to see how training progresses.

11. Hyperparameter Adjustment:

Experimentation:

Observe training dynamics under varied optimizers, learning rates, and loss functions. Optimize these hyperparameters for efficient model performance.

Regularization Techniques:

However, over-fitting can be prevented by incorporating regularization methods like L2 regularization or drop-outs. Tune the strength according by the modelled behaviour.

Weight Initialization:

Initialise the model parameter values by experimenting with various weight initialization strategies. Initialization affects the stability of training.

The software does this by thoughtfully designing the LSTMs model architecture and tuning hyperparameters such that the neural network can appropriately learn and represent the temporal patterns existing within the sensor data.

Documentation:

Record Compilation Configuration:

Track the selected optimizer, learning rate, objective function, and metrics. Documentation facilitates transparency and reproducibility.

12. Model Training:

Data Input Preparation:

To begin with, organize batched versions of the training data. Make sure to keep the temporal order while handling a batch in the context of time series forecasting.

Training Loop:

Use an epoch loop that iterates over a certain number of epochs. The model receives batches of training data and undertakes weight updates during each cycle.

Validation Set Utilization (Optional):

If possible, monitor the model's performance and ensure the best fit using a validation set. The performance of the validation set provides insights for adjustments to hyperparameters and the decision to include or exclude early stopping.

Batch Size and Epochs:

Try out various combinations of batch size and epoch in order to achieve adequate efficiency in training and model convergence. A smaller number of epochs may be required per batches in order to converge.

Training Metrics:

Track training measures like the loss and evaluation metrics (e.g. MAE) in order to evaluate the rate of learning for the model. This can be important in determining how well the model has adjusted to the training data.

TensorBoard Integration:

While working with TensorFlow, embed TensorBoard to visualize in real-time the training metrics, the model architecture as well as the weight histograms. The tool allows for detailed evaluation of the training process.

Early Stopping:

Start with early stopping, i.e., stopping training when the model stops improving on the validation set. It avoids overfitting and thus does not overtrain.

Save Trained Model:

Save the saved model after training period. Therefore, predictions can be made or deployment in production using the model without retraining.

13. Model Evaluation:

Assessing Model Performance on Test Data.

Test Set Utilization:

For a proper assessment of its ability to generalize, use the separate test set, which is different from the model's training sample.

Prediction Generation:

Use the learned LSTM model on the test data for prediction. The performance of the model in predicting the future values is compared with the actual values in the test set.

Evaluation Metrics:

In addition, come up with methods to compute evaluation metrics to help quantify the quality and the performance of the model. Some of the common metrics used for regression tasks are MAE, MSE, and RMSE.

14. Real-Time Prediction:

Implementing Real-Time Prediction Mechanism:

Streaming Data Integration:

Develop an interface that can stream real time sensor data to predictive model. It is a process that entails continually collecting and analyzing up-to-date sensor records.

Preprocessing for Real-Time Data:

Utilise similar preprocessing techniques for cleaning, normalizing and formatting of the incoming real-time data. Prepare the data into a form compliant to the inputs demanded by the trained LSTM model.

Continuous Prediction Loop:

Create a cycle to provide the processed real time data to the LSTM model trained previously for making predictions. Such a loop must be able to run without any hitch so as to provide on-time forecasts.

Adapting to Data Frequency:

Make the frequency of predictions dependent from the frequency of incoming data. For example, if the values of the sensor readings come in every second, design the prediction loop in a way that will make the system to always be reactive in real time.

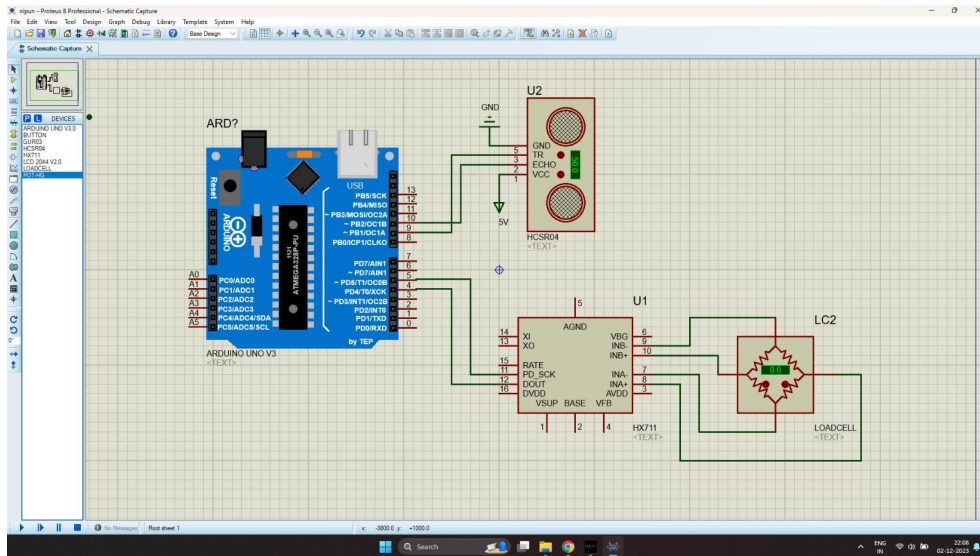
Alerting Mechanism:

Set-out thresholds for abnormalities or dangerous occurrences depending on the predicted values. Historical data analysis or knowledge on the domain may also help set these thresholds.

Alert Triggering:

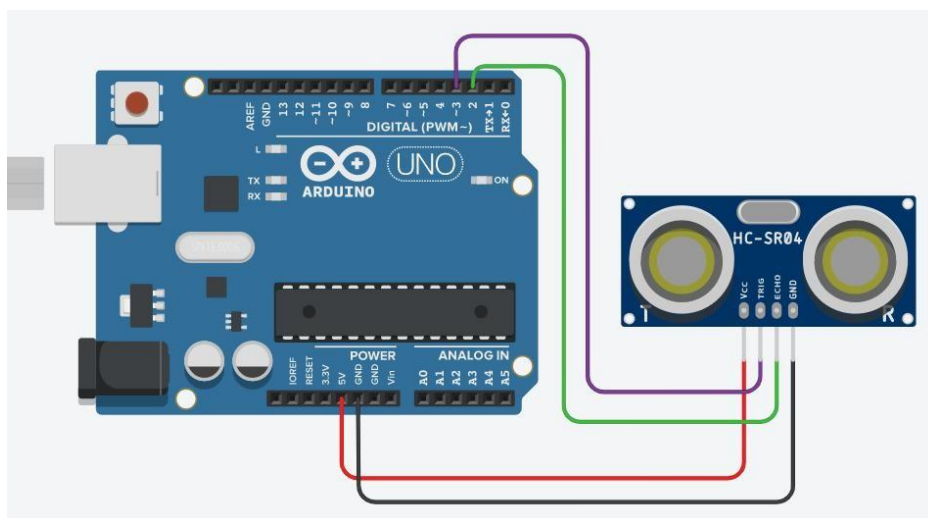
Create an alarm system that sends alerts or triggers an action on predicted value which goes more than the preset limits. For example, this might include sending emails, texts, or carrying out actions within the system.

3.2 Modules



HC-SR04 Ultrasonic Sensor:

It is a type of ultrasonic rangefinder called HC-SR04. This is made up of a transmitter, which emits ultrasonic pulses, and then a receiver that notices the reflecting signal. The sensor then computes the time for the signal to make a roundtrip in order to give very accurate measurements of its distance from nearby objects.

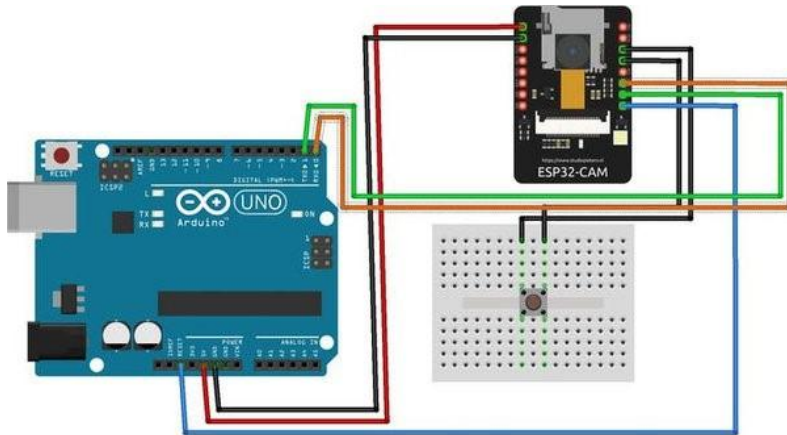


Arduino:

Arduino is an open-source electronics prototyping platform that consists of an integrated development environment(IDE) to write the code into a board . The interface is very easy to use, providing a good platform for creative people to develop interactive eLearning projects. Arduino is a simple and flexible tool that is commonly employed for prototypes as well as do-it-yourself tasks.

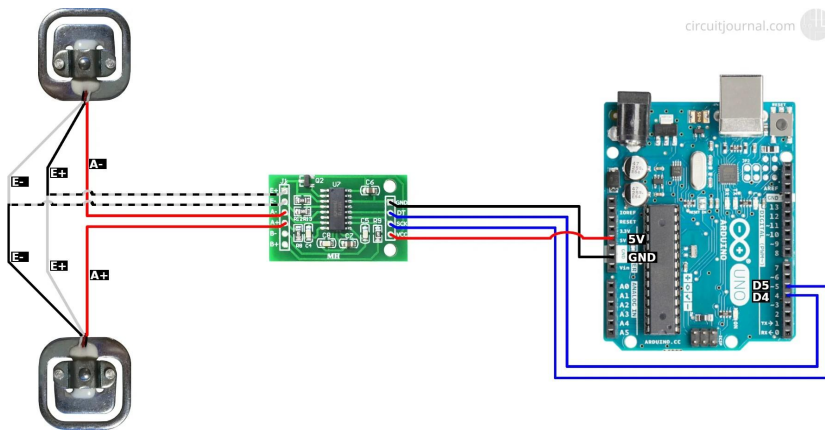
ESP32 Camera:

Esp32 camera is a development board derived from the esp32 microcontroller and the integrated camera module. This device has combined the strength from processing and communication in the ESP32 and the capability to take videos or pictures. This suits projects that are about IoT, and also for visual-oriented apps.



Half Bridge Load Cells:

Load cells that use a half bridge design act as transducers converting force or weight into an electrical signal. These usually involve strain gauges attached to a flexible structure. The strain gauges change their electrical resistance when a load is applied. With that, the change is made to quantify the correctly applied force or weight.

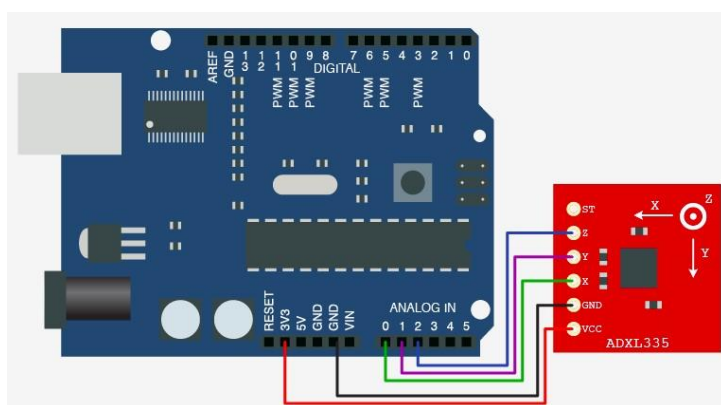


HX711 Amplifier:

Load cell precision amplifier known as HX711. It increases these tiny electrical signals given out by load cells to a form that matches with micro-controller capabilities. There are numerous ways in which HX711 can be used to interface load cells to microcontrollers such as Arduino for precise weighing in different fields.

Accelerometer:

The accelerometer is a sensor that measures acceleration consisting of static forces like gravity and dynamic forces like motion. It often gives out, as the motion accelerates values measured in different directions. There are commonly applied accelerometers in electronic devices for detecting screen orientation, tracking motion, or sensing impacts.



Chapter 4 Implementation

4.1 Load cell Configuration

The following is the procedure for interfacing with an HX711 load cell amplifier using the HX711_ADC library. The HX711 is a 24-bit analog-to-digital converter specifically designed for weight measurement applications.

Procedure:

Include Libraries:

Some necessary libraries are required for the functioning, especially the `HX711_ADC` library, which provides functions to interact with the HX711 load cell amplifier. It also includes the EEPROM library for handling non-volatile memory on some microcontrollers.

Define Pin Configuration:

The pins for data (DOUT) and clock (SCK) connections between the microcontroller and the HX711 are specified. These connections are crucial for communication with the load cell.

Initialize HX711_ADC Object:

An instance of the `HX711_ADC` class is created with the specified pin configuration. This object, named `LoadCell`, will be used for interacting with the HX711.

Setup Function:

- Serial communication is initialized for debugging purposes.
- The `LoadCell` object is initialized with some additional settings, including stabilization time and whether to perform taring (zeroing the scale) at startup.
- If any issues occur during initialization, an error message is printed.

Loop Function:

Data is continuously read from the load cell using ``LoadCell.update()``.

If new data is available, it's printed to the serial monitor.

The code checks for commands from the serial terminal:

- `'t'`: Initiates tare (zero) operation.
- `'r'`: Calls the ``calibrate()`` function for calibration.
- `'c'`: Calls the ``changeSavedCalFactor()`` function for manually changing the calibration value.

It also checks if the tare operation is complete and prints a message if so.

Calibration Function (``calibrate()``):

- Tares the load cell to account for any offset.
- Prompts the user to place a known mass on the load cell and send the `'t'` command to set the tare offset.
- Prompts the user to input the weight of the known mass from the serial monitor.
- Refreshes the dataset and calculates a new calibration value.
- Asks the user whether to save the new calibration value to EEPROM.

Change Calibration Value Function:

We allow the user to manually change the calibration value by entering a new value from the serial monitor. We also ask the user whether to save the new calibration value to EEPROM.

The code provides a flexible way to calibrate and interact with the HX711 load cell amplifier, making it suitable for various weight measurement applications.

4.2 Ultrasonic Sensor Configuration

The following procedure is for the configuration of ultrasonic sensors which can be used to detect the rust in lift systems.

Pin Configuration:

- ``trigPin``: Connected to the trigger pin of the ultrasonic sensor (for sending the signal).
- ``echoPin``: Connected to the echo pin of the ultrasonic sensor (for receiving the signal).

Setup Function:

- Configures ``trigPin`` as an output and ``echoPin`` as an input. These configurations define the roles of the pins in the Arduino.
- Initializes serial communication at a baud rate of 57600 to communicate with the Serial Monitor.

Loop Function:

- The main code block that runs repeatedly.
- Sets the ``trigPin`` to LOW and waits for 100 milliseconds. This is done to ensure a clean start.
- Sends a short pulse (10 microseconds) to the ``trigPin`` by setting it to HIGH and then LOW.
- Uses ``pulseIn`` to measure the duration (in microseconds) for which the ``echoPin`` is HIGH. This duration corresponds to the time taken for the ultrasonic pulse to travel to the object and back.
- Calculates the distance using the formula: $\text{distance} = \text{duration} * \text{speed of sound} / 2$.

The factor of 0.034 is used to convert the time to distance in centimeters (since the speed of sound is approximately 343 meters/second).

- Prints the calculated distance to the Serial Monitor.

This code reads distance measurements from an ultrasonic sensor and outputs the results to the Serial Monitor. The ultrasonic sensor sends out a sound pulse, and the Arduino measures the time it takes for the pulse to bounce back. The distance is then calculated based on the speed of sound.

Chapter 5 Results

The following snippet is of the successful working of the load cell.

```
Starting...
Startup is complete
Distance: 7.43
Load_cell output val: -0.00
Distance: 7.43
Load_cell output val: -0.00
Distance: 7.51
Load_cell output val: -0.00
Distance: 7.53
Load_cell output val: -0.00
Distance: 7.41
Load_cell output val: -0.00
Distance: 7.43
Load_cell output val: -0.00
Distance: 7.51
Load_cell output val: -0.00
Distance: 7.53
Load_cell output val: -0.00
Distance: 7.51
Load_cell output val: -0.01
Distance: 7.43
Load_cell output val: -0.01
Distance: 7.41
Load_cell output val: -0.01
Distance: 7.53
Load_cell output val: -0.01
Distance: 7.51
Load_cell output val: -0.01
Distance: 7.41
Load_cell output val: -0.01
Distance: 7.41
Load_cell output val: -0.01
Distance: 7.53
Load_cell output val: -0.01
Distance: 7.53
Load_cell output val: -0.01
Distance: 7.53
Load_cell output val: -0.01
Distance: 7.41
Load_cell output val: -0.01
```

Starting...": This is typically a message that appears when a process or application has begun its initialization sequence.

"Startup is complete": This indicates that the initialization process has finished successfully.

"Distance: 7.41" (and other similar lines): These lines give a numerical value labelled "Distance." This is the output from the ultrasonic sensor measuring distance.

"Load cell output val: -0.00" (and similar lines): A load cell is a type of transducer that converts force into an electrical signal. These lines show the output value from such a load cell.

The "-0.00" suggests that either there is no load on the cell or the value is so small that it rounds down to zero when displayed to two decimal places.

The regular pattern of the "Distance" and "Load cell output val" lines suggests this is a continuous monitoring log, a program designed to read from sensors in real-time.

Load cell:

The load cell measures the weight supported by the lift, ensuring that the load is within the specified capacity. In case of wear and tear the load capacity varies, using ESP 32 cam we detect the wear and tear which helps us to monitor the load

capacity in real-time. We also monitor variations in load, helping to detect anomalies such as overcrowding or abnormal loads that could lead to mechanical stress. The above output screenshot shows the variation in load indicating that the sensor is working

Ultrasonic sensor:

Studies say that the thickness of steel increases by 6 times during corrosion, using this knowledge we used the ultrasonic sensor which emits sound waves to detect any increase in metal thickness. In case of any increase in thickness, we can confirm the rust. In the output photo, we can see the distance being calculated which can be used to detect corrosion

ESP32 Cam:

The ESP32 CAM serves as a camera module, enabling visual inspection of the lift components. We used this to capture images or videos of critical areas,

allowing for detailed analysis of the physical condition of components. Visual data can be used to identify wear and tear, corrosion, or any physical anomalies in the lift system.

Chapter 6 Outcomes

Enhanced Safety:

- Proactive detection of potential safety issues.
- Real-time monitoring reduces the risk of elevator malfunctions.
- Swift response to abnormal vibrations contributes to emergency brake activation, averting accidents.

Predictive Maintenance Efficiency:

- Timely interventions minimize downtime and optimize elevator performance.
- Extended lifespan of elevator components.

Substantial Cost Savings

- Reduction in costly emergency repairs.
- Minimized downtime results in significant financial benefits.
- Targeted and cost-effective maintenance interventions.

Data-Driven Solutions:

- Comprehensive integration of advanced technologies.
- Holistic approach covers various aspects of elevator health.
- Data-driven decision-making enhances overall efficiency.

Versatility and Adaptability:

- Apparatus suitable for both new installations and retrofitting.
- Addresses diverse elevator conditions with adaptability.

Preservation of Building Assets:

- Early detection prevents deterioration of elevator components.
- Preserves building assets by avoiding potential malfunctions.

Legal Liability Mitigation:

- Averts potential legal liabilities associated with elevator malfunctions.
- Ensures compliance with safety standards, reducing legal risks.

Chapter 7 Single Unique Factor

Proactive Safety Features:

Ultrasonic sensors detect wear, tear, and rust in elevator shafts.

Cameras inspect wires for fraying, serving as an early warning system.

Acceleration monitoring responds to abnormal vibrations, activating emergency brakes and preventing accidents.

Predictive Maintenance Strategies:

Deep learning algorithms analyze sensor data to predict faults and wear patterns.

Timely interventions minimize downtime and optimize elevator performance.

Extends the lifespan of components, ensuring precise maintenance when needed.

Cost Savings Benefits:

Reduces the need for costly emergency repairs.

Minimizes downtime for significant financial benefits.

Early detection allows for targeted and cost-effective maintenance interventions.

Holistic and Integrated Nature:

Comprehensive solution covering structural integrity, wire conditions, and operational vibrations.

Versatile for both new installations and retrofitting.

Offers a unique and valuable addition to elevator condition monitoring.

Chapter 8 End User Benefits

Building Owners and Property Managers:

Benefit from increased property value through reliable elevator performance.
Reduce operational costs and enhance tenant satisfaction.

Facility Maintenance Teams:

Streamline maintenance activities through targeted interventions.
Improve efficiency by focusing efforts on critical components.

Elevator Service Providers:

Enhance customer satisfaction by offering proactive maintenance services.
Reduce emergency service calls through preventive measures.

Regulatory Authorities:

Ensure compliance with safety regulations.
Access comprehensive data for regulatory reporting and audits.

Occupants and Visitors:

Experience increased confidence in elevator safety.
Enjoy a smoother and more reliable elevator service.

Insurance Providers:

Mitigate risk through proactive maintenance, potentially lowering insurance premiums.
Access data for risk assessment and policy pricing.

Chapter 9 Site Visit

During the field visit for the Lift Condition Monitoring Project, we had the opportunity to gain valuable insights into the operational aspects and challenges faced by the existing lift systems within a multi-story building.

Site Assessment:

We conducted a thorough site assessment to understand the layout of the building, the number of lifts, and their respective capacities. We also examined the existing lift monitoring infrastructure including communication systems.

Interaction with Maintenance Personnel:

We engaged in discussions with the maintenance personnel responsible for overseeing lift operations and addressing any issues. By this we were able to gather information on common problems faced, response times to incidents, and the existing maintenance workflow.

Observation of Lift Operations:

On request, we were able to observe the lifts in operation during peak and off-peak hours to identify any unusual noises, delays, or irregularities. We also inquired about the current frequency of maintenance visits and the type of issues typically encountered.

Challenges and Opportunities:

We were able to identify specific challenges faced by the current lift monitoring system, such as delays in fault detection, reactive maintenance practices, etc.

Overall, we explored opportunities for improvement, considering the integration of advanced technologies, predictive analytics, and remote monitoring capabilities.

The field visit provided valuable knowledge, allowing for a more informed and tailored approach to developing an advanced lift monitoring system that

addresses the specific needs and challenges of the building's vertical transportation infrastructure.

Chapter 10 Sample Codes

10.1 Ultrasonic Sensor Code

```
const int trigPin = 9;
const int echoPin = 10;

long duration;
float distance;

void setup() {
  // put your setup code here, to run once:
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  Serial.begin(57600);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(trigPin, LOW);
  delay(100);

  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);
  distance = duration*0.034/2;

  Serial.print("Distance: ");
  Serial.println(distance);
}
```

10.2 Load cell code

Load Cell code:

```
// // #include "HX711.h"
// #include <HX711_ADC.h>

// const int HX711_dout = 4; //mcu > HX711 dout pin
// const int HX711_sck = 5; //mcu > HX711 sck pin

// HX711_ADC scale(HX711_dout, HX711_sck);
// float current_weight = 0;
// float scale_factor;

// void setup() {
// // put your setup code here, to run once:
// Serial.begin(57600);
// Serial.println("HX711 Demo");

// // scale.begin(HX711_dout, HX711_sck);
// scale.setCalFactor(-842.40f);
// // scale.tare();
// }

// void loop() {
// // put your main code here, to run repeatedly:
// current_weight = scale.getData();
// Serial.println(current_weight);
// delay(500);
// }

/*
```

```
-----  
HX711_ADC  
Arduino library for HX711 24-Bit Analog-to-Digital Converter for Weight  
Scales  
Olav Kallhovd sept2017  
-----
```

```
*/
```

```
/*
```

This example file shows how to calibrate the load cell and optionally store the calibration

value in EEPROM, and also how to change the value manually.

The result value can then later be included in your project sketch or fetched from EEPROM.

To implement calibration in your project sketch the simplified procedure is as follow:

```
LoadCell.tare();
```

```
//place known mass
```

```
LoadCell.refreshDataSet();
```

```
float newCalibrationValue = LoadCell.getNewCalibration(known_mass);
```

```
*/
```

```
#include <HX711_ADC.h>
```

```
#if defined(ESP8266) || defined(ESP32) || defined(AVR)
```

```
#include <EEPROM.h>
```

```
#endif
```

```
//pins:
```

```
const int HX711_dout = 4; //mcu > HX711 dout pin
```

```
const int HX711_sck = 5; //mcu > HX711 sck pin
```

```
//HX711 constructor:
```

```
HX711_ADC LoadCell(HX711_dout, HX711_sck);
```

```

const int calVal_eepromAdress = 0;
unsigned long t = 0;

void setup() {
  Serial.begin(57600); delay(500);
  Serial.println();
  Serial.println("Starting...");

  LoadCell.begin();
  //LoadCell.setReverseOutput(); //uncomment to turn a negative output
value to positive
  unsigned long stabilizingtime = 2000; // preciscion right after power-up can
be improved by adding a few seconds of stabilizing time
  boolean _tare = true; //set this to false if you don't want tare to be performed
in the next step
  LoadCell.start(stabilizingtime, _tare);
  if (LoadCell.getTareTimeoutFlag() || LoadCell.getSignalTimeoutFlag()) {
    Serial.println("Timeout, check MCU>HX711 wiring and pin designations");
    while (1);
  }
  else {
    LoadCell.setCalFactor(-37114.0); // user set calibration value (float), initial
value 1.0 may be used for this sketch
    Serial.println("Startup is complete");
  }
  while (!LoadCell.update());
  // calibrate(); //start calibration procedure
}

void loop() {
  static boolean newDataReady = 0;
  const int serialPrintInterval = 0; //increase value to slow down serial print
activity

```

```

// check for new data/start next conversion:
if (LoadCell.update()) newDataReady = true;

// get smoothed value from the dataset:
if (newDataReady) {
    if (millis() > t + serialPrintInterval) {
        float i = LoadCell.getData();
        Serial.print("Load_cell output val: ");
        Serial.println(i);
        newDataReady = 0;
        t = millis();
    }
}

// receive command from serial terminal
if (Serial.available() > 0) {
    char inByte = Serial.read();
    if (inByte == 't') LoadCell.tareNoDelay(); //tare
    else if (inByte == 'r') calibrate(); //calibrate
    else if (inByte == 'c') changeSavedCalFactor(); //edit calibration value
    manually
}

// check if last tare operation is complete
if (LoadCell.getTareStatus() == true) {
    Serial.println("Tare complete");
}

}

void calibrate() {
    Serial.println("***");
    Serial.println("Start calibration:");
}

```

```
Serial.println("Place the load cell on a level stable surface.");  
Serial.println("Remove any load applied to the load cell.");  
Serial.println("Send 't' from serial monitor to set the tare offset.");
```

```
boolean _resume = false;  
while (_resume == false) {  
  LoadCell.update();  
  if (Serial.available() > 0) {  
    if (Serial.available() > 0) {  
      char inByte = Serial.read();  
      if (inByte == 't') LoadCell.tareNoDelay();  
    }  
  }  
  if (LoadCell.getTareStatus() == true) {  
    Serial.println("Tare complete");  
    _resume = true;  
  }  
}
```

```
Serial.println("Now, place your known mass on the loadcell.");  
Serial.println("Then send the weight of this mass (i.e. 100.0) from serial  
monitor.");
```

```
float known_mass = 0;  
_resume = false;  
while (_resume == false) {  
  LoadCell.update();  
  if (Serial.available() > 0) {  
    known_mass = Serial.parseFloat();  
    if (known_mass != 0) {  
      Serial.print("Known mass is: ");  
      Serial.println(known_mass);  
      _resume = true;  
    }  
  }  
}
```



```
}  
}
```

```
LoadCell.refreshDataSet(); //refresh the dataset to be sure that the known  
mass is measured correct
```

```
float newCalibrationValue = LoadCell.getNewCalibration(known_mass);  
//get the new calibration value
```

```
Serial.print("New calibration value has been set to: ");  
Serial.print(newCalibrationValue);  
Serial.println(", use this as calibration value (calFactor) in your project  
sketch.");
```

```
Serial.print("Save this value to EEPROM adress ");  
Serial.print(calVal_eeepromAdress);  
Serial.println("? y/n");
```

```
_resume = false;  
while (_resume == false) {  
  if (Serial.available() > 0) {  
    char inByte = Serial.read();  
    if (inByte == 'y') {  
#if defined(ESP8266)|| defined(ESP32)  
      EEPROM.begin(512);  
#endif  
      EEPROM.put(calVal_eeepromAdress, newCalibrationValue);  
#if defined(ESP8266)|| defined(ESP32)  
      EEPROM.commit();  
#endif  
      EEPROM.get(calVal_eeepromAdress, newCalibrationValue);  
      Serial.print("Value ");  
      Serial.print(newCalibrationValue);  
      Serial.print(" saved to EEPROM address: ");  
      Serial.println(calVal_eeepromAdress);  
      _resume = true;
```

```

    }
    else if (inByte == 'n') {
        Serial.println("Value not saved to EEPROM");
        _resume = true;
    }
}
}

```

```

Serial.println("End calibration");
Serial.println("****");
Serial.println("To re-calibrate, send 'r' from serial monitor.");
Serial.println("For manual edit of the calibration value, send 'c' from serial
monitor.");
Serial.println("****");
}

```

```

void changeSavedCalFactor() {
    float oldCalibrationValue = LoadCell.getCalFactor();
    boolean _resume = false;
    Serial.println("****");
    Serial.print("Current value is: ");
    Serial.println(oldCalibrationValue);
    Serial.println("Now, send the new value from serial monitor, i.e. 696.0");
    float newCalibrationValue;
    while (_resume == false) {
        if (Serial.available() > 0) {
            newCalibrationValue = Serial.parseFloat();
            if (newCalibrationValue != 0) {
                Serial.print("New calibration value is: ");
                Serial.println(newCalibrationValue);
                LoadCell.setCalFactor(newCalibrationValue);
                _resume = true;
            }
        }
    }
}

```

```

    }
}
_resume = false;
Serial.print("Save this value to EEPROM address ");
Serial.print(calVal_eepromAddress);
Serial.println("? y/n");
while (_resume == false) {
    if (Serial.available() > 0) {
        char inByte = Serial.read();
        if (inByte == 'y') {
#ifdef ESP8266 || defined(ESP32)
            EEPROM.begin(512);
#endif
            EEPROM.put(calVal_eepromAddress, newCalibrationValue);
#ifdef ESP8266 || defined(ESP32)
            EEPROM.commit();
#endif
            EEPROM.get(calVal_eepromAddress, newCalibrationValue);
            Serial.print("Value ");
            Serial.print(newCalibrationValue);
            Serial.print(" saved to EEPROM address: ");
            Serial.println(calVal_eepromAddress);
            _resume = true;
        }
        else if (inByte == 'n') {
            Serial.println("Value not saved to EEPROM");
            _resume = true;
        }
    }
}
Serial.println("End change calibration value");
Serial.println("****");
}

```

Chapter 11 References

Q. Peng et al., "Analysis of Vibration Monitoring Data of Flexible Suspension Lifting Structure Based on Time-Varying Theory," *Sensors*, vol. 20, no. 22, p. 6586, Nov. 2020.

R. E. Melchers, "The effect of corrosion on the structural reliability of steel offshore structures," *Corrosion Science*, vol. 47, no. 10, pp. 2391-2410, 2005.

E. P. Carden and P. Fanning, "Vibration Based Condition Monitoring: A Review," *Structural Health Monitoring*, vol. 3, no. 4, 2004. Available:

Y. Avenas, L. Dupont, N. Baker, H. Zara and F. Barruel, "Condition Monitoring: A Decade of Proposed Techniques," in *IEEE Industrial Electronics Magazine*, vol. 9, no. 4, pp. 22-36, Dec. 2015, doi: 10.1109/MIE.2015.2481564.

I. Skog, I. Karagiannis, A. B. Bergsten, J. Härdén, L. Gustafsson and P. Händel, "A Smart Sensor Node for the Internet-of-Elevators—Non-Invasive Condition and Fault Monitoring," in *IEEE Sensors Journal*, vol. 17, no. 16, pp. 5198-5208, 15 Aug. 2017, doi: 10.1109/JSEN.2017.2719630.

O. Yaman and M. Karakose, "Auto correlation based elevator rope monitoring and fault detection approach with image processing," 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), Malatya, Turkey, 2017, pp. 1-5, doi:10.1109/IDAP.2017.8090176.

A. Q. Flores, J. B. Carvalho and A. J. M. Cardoso, "Mechanical fault detection in an elevator by remote monitoring," 2008 18th International Conference on Electrical Machines, Vilamoura, Portugal, 2008, pp. 1-5, doi: 10.1109/ICELMACH.2008.4800064.

Steelconstruction.info, "Corrosion of Structural Steel".