Angle Sensor Lifetime Prediction

(Case Study Intelligent Systems)

Guided by:

Dr-Ing. Proof. Jürgen Wittmann

Technische Hochschule Deggendorf (TC CHAM)

Written by:

Amin Nazari (12204809)

Gaurav Sharma (12200584)

Ibrahim Kemal Yasar (22111828)

Kirtan Soni (12201189)

Date: 25.01.2023

Table of Contents

1. Abstract

Angle Sensor Lifetime Prediction is a project aimed at developing a model to predict the lifetime of angle sensors. The project will begin with an introduction to angle sensors and the current methods used to predict their lifetime. The status of the project will be discussed, including the current state of research in the field and any relevant background information. The objectives and the scope of the project will include a thorough examination of the various factors that can affect the lifetime of angle sensors.

The literature survey will cover a range of techniques used to predict the lifetime of angle sensors. The concept of execution will involve the collection and pre-processing of data to be used in the model, as well as an exploration of different machine learning techniques including regression and Arrhenius models that could be used to predict the lifetime of angle sensors.

The data pre-processing step will involve cleaning and formatting the data, as well as any necessary feature engineering. The approaches and results section will discuss the different machine learning techniques used in the project, including regression and Arrhenius models, as well as recurrent neural networks (RNNs). The results of each approach will be compared and analysed to determine the most effective method for predicting the lifetime of angle sensors.

The project will also address potential risks and limitations, including the limitations of the data. A timeline for the project will be provided, along with any next steps for future research. The conclusion will summarize the findings of the project and discuss the potential applications of the developed model.

References and an appendix containing the complete code used in the project will be included at the end of the project. The project will provide valuable insights into the lifetime prediction of angle sensors, and the developed model could be used to improve the maintenance of angle sensors in various industries.

2. Introduction

The prediction of sensor lifetime is a crucial aspect in the maintenance and operation of industrial systems. It helps in reducing the costs associated with unexpected failures and downtime, and in improving the overall efficiency and reliability of the system.

In this report, we focus on the prediction of the lifetime of Tunnel Magneto-Resistive (TMR) angle sensors using different machine learning techniques. TMR sensors are widely used in various applications such as automotive, aerospace, and robotics due to their high sensitivity, accuracy, and low power consumption. However, their lifetime is affected by various factors such as temperature, humidity, vibration, and magnetic influence. To predict the lifetime of TMR sensors, we have used various machine learning algorithms such as Arrhenius equation with linear regression and Recurrent neural networks (RNN). We were provided with already collected data on the performance of TMR sensors under different reliability tests such as High-temperature operating life (HTOL). HTOL reliability test is a method of testing electronic components or devices at high temperatures over a prolonged period to simulate real-world conditions and identify any potential issues that may arise during use.

This report will also show the different approaches done towards solving different problems we faced during executing and using the data and what approaches were made to minimize the effects of the issues related to the huge amount of data that we were provided.

We have also drafted an in-detail explanation in this report of how we have predicted the life time of a TMR angle sensor using Arrhenius equation along with linear regression and using the RNN technique. By using different approaches, we aim to predict the lifetime of TMR sensors, thereby reducing maintenance costs and improving system uptime. In this report, we will also present our code structure with a proper explanation for a better insight of our study on the prediction of TMR sensor lifetime using ML and later in this report we have compared the results of different approaches. Additionally, we will also discuss the implications of our study and suggest areas for future research.

Overall, this report aims to provide a comprehensive analysis of the prediction of TMR sensor lifetime using artificial intelligence techniques such as machine learning and what else it will require to make the predictions more accurate and reliable.

3. Status

In this section of the report, we will explain about all the required components we had at the very beginning of this project in detail, at first, we will give a brief introduction about the sensors we worked with, also we will discuss about the process and the device which was used to collect all the data of the respective sensors. At the end we will briefly show how the sensors were calibrated and how all the readings/data was stored and provided to us.

Information about the sensors

We had the data of two angle sensors for this project:

1) Infineon TLE5501 E0001
2) TDK TAS2141-AAAB

Figure 1- Angle sensors

These both are TMR angle sensor.

TMR sensors are a type of magnetic sensor that are based on the phenomenon of tunnelling magnetoresistance. TMR is a phenomenon where the resistance of a material changes in response to a magnetic field. TMR sensors take advantage of this phenomenon to detect changes in magnetic fields.

A TMR sensor typically consists of a stack of thin films, including a thin insulating layer and two ferromagnetic layers as shown in figure 2. The resistance of the TMR sensor changes as the relative alignment of the magnetic moments of the two ferromagnetic layers changes in response to an external magnetic field. This change in resistance can be measured and used to determine the strength and direction of the magnetic field. These sensors are known for their high sensitivity and accuracy, and can be used in a wide range of applications, such as magnetic field sensing in hard disk drives, position, and speed sensing in electric motors, and magnetometry in navigation systems. They also have a wide range of temperature stability and have been known to have a high resistance to radiation.

As shown in the figure 2, it shown very clearly the output signals are generated with respect to the change in resistance as the external magnetic field or magnet is rotated at different angles.

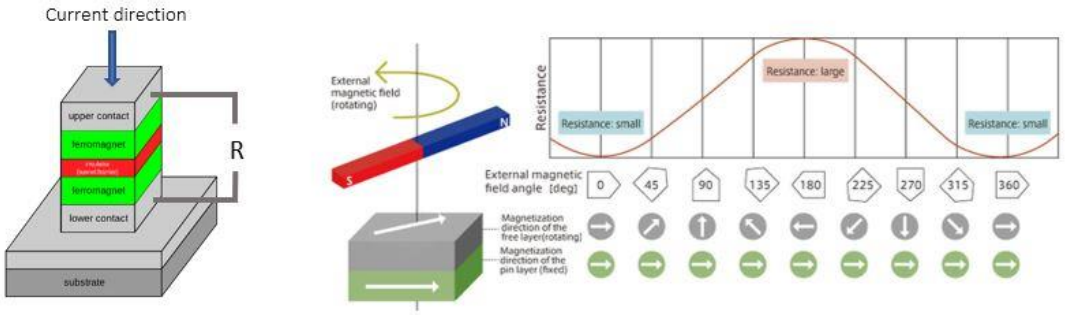


Figure 2- Angle sensor

Typically, these sensors give output as four different sinusoidal waveforms (Sin+, Sin-, Cos+, Cos-) as shown the below figure 3. The values of these four changes as per the angle and the direction at which the magnet is rotated. These signals are in voltage form and they are generated using two Wheatstone bridges (P-Bridge and N-Bridge) which are parallelly connected inside the sensor as shown in the below figure.

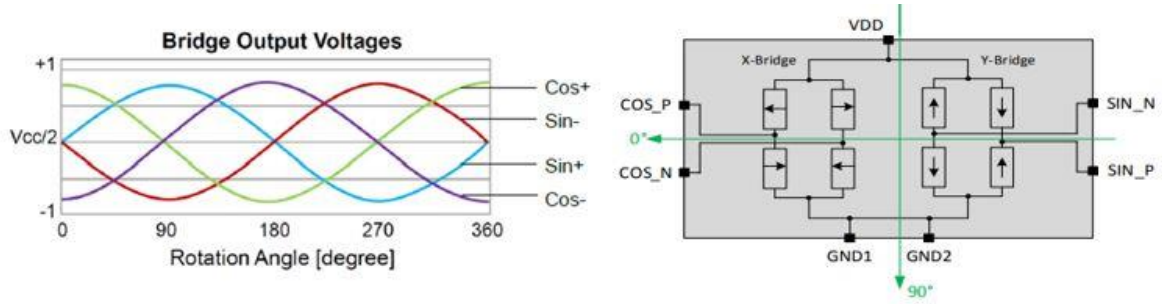


Figure 3- left: Four voltage signals from TMR sensor, right: Internal circuitry of TLE5501 E001 sensor as per datasheet [9] [10]

Information about the device

The data was collected using a device as shown in the below figure 4. This device's frame is constructed using machined aluminium, there is a shaft which is controlled by the stepper motor and at the end of the shaft there is a permanent magnet attached of approximately 61mT. This external field will be subjected to the TMR sensor to get the readings which it provides as an output. The TMR sensor is kept on the snensor holder which is kept exactly 6mm away form the permanent magnet provided as per the datasheet. The shaft is rotated by stepper motor with a feedback of an optical encoder (of 40,000 PPR) attached to the mtor for precise angle movement and then the readings are measured from the sensor and then by comparing these two readings, keeping encoder as a reference, the angle error is measured.

This whole device is controlled by Arduino Mega development board which consist ATmega 2560 microcontroller. The circuit diagram is also provided of the device in the below figure 4.

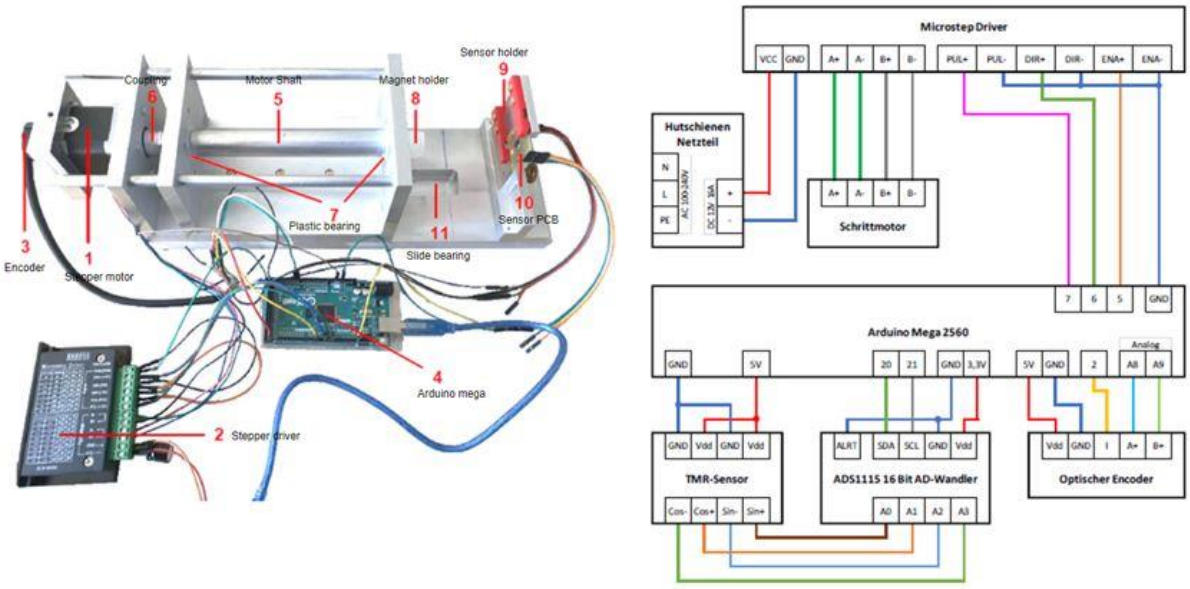


Figure 4- left: Image of the measurement device, right: Circuitry of the measurement device [11]

Information about the data measurement

The main readings we needed were angle error and hysteresis, the angle error can be calculated by comparing the readings with the optical encoder feedback. But the problem is no sensor gives output ideally, if we talk about the TMR sensor then the output voltage signals/waveform coming from it were shifted, compressed, and distorted. To process any further with these readings we need to calibrate the sensor and find out the values which are responsible for the distortion.

There are several parameters we need to keep in mind for correcting the distorted waveform like:

1. Offset
2. Amplitude
3. Non-orthogonality or Phase shifting

These three factors are mainly responsible for the bad waveform, to get a better picture of this refer the below figure 5. How to solve this issue, we will talk further in the report.

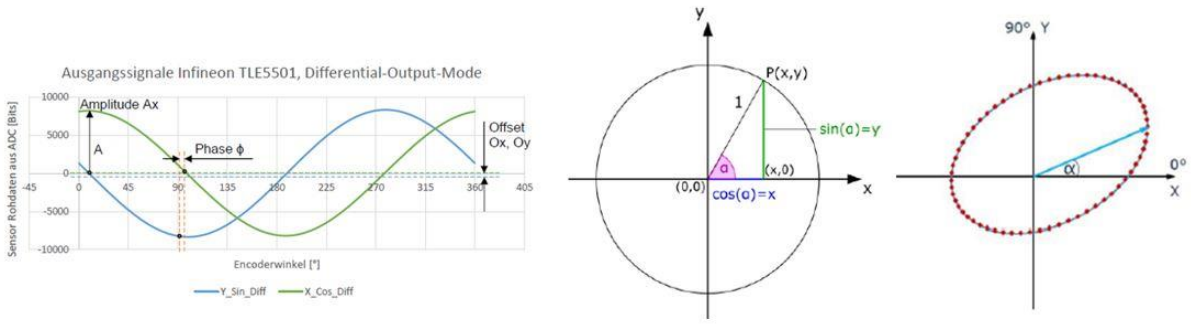


Figure 5- left: Output waveform from the TLE5501 E001 sensor, right: Output waveform in the phasor form

For getting the angle from the sinusoidal waveforms we need to convert them into phasor representation and ideally the process should be relatively simple, as shown in figure 8 above. We just need to calculate the angle **α** by using the formula **α = arc$tan(y/x)$,** but due the factors discussed above if we try to calculate the **α** we will get a very inaccurate value. So, for this we should calculate the value of these factors and use them to fit the waveform in its ideal form.

The process of measuring data is divided into two parts:

1. Calibration test
2. Functional test

In calibration test we calculate these values of a particular sensor and apply those values in the functional run to get more accurate angle readings from the sensor.

Calibration test

In the calibration run magnet was rotated with a series of angles and directions as per given in the datasheet, the steps are shown in the figure 6 for a better understanding. The readings from the TMR angle sensor were taken only at the end of 3rd step and 6th step. The resolution of stepper motor was set to 65 steps per one full rotation, so one step of stepper motor the shaft/magnet will rotate (360°/ 65) = 5.538° and the total readings will be:

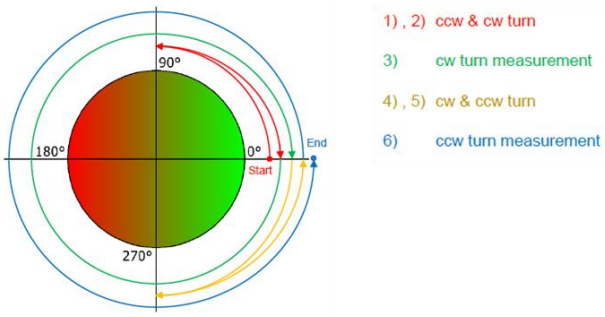


Figure 6- Calibration process

- 65 readings for 3rd step while the magnet rotates 0°- 360° clock wise.
- 65 readings for 6th step while the magnet rotates 360° - 0° anti clockwise.

Therefore, in total for the calibration run we will obtain 130 readings of angles with respect to the rotation of the shaft/magnet. The calibration test consists of three steps:

1. **Differential output mode:**

As the output as four waveforms we can convert them to two output waveforms (figure 7) with the help of equations given below.

$$X_{COS\_Diff} = COS_P - COS_N$$

$$Y_{SIN\_Diff} = SIN_P - SIN_N$$

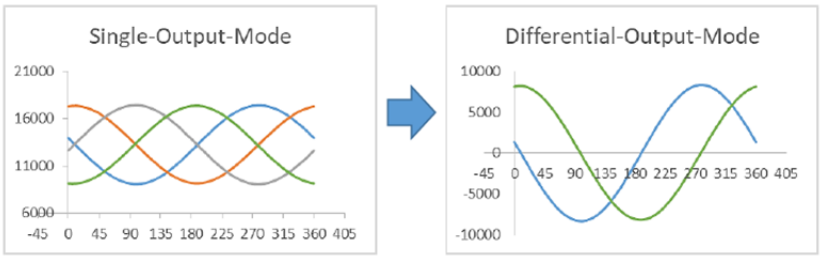


Figure 7- Differential output mode

2. **Calculate all the factors**

Here all the affecting parameters are calculated with the help of Fitting method.

Offset:
$$O_{xM} = \frac{O_{xcw}+O_{xccw}}{2}; \quad O_{yM} = \frac{O_{ycw}+O_{yccw}}{2}$$

Amplitude:
$$A_{xM} = \frac{A_{xcw}+A_{xccw}}{2}; \quad A_{yM} = \frac{A_{ycw}+A_{yccw}}{2}$$

Phase:
$$\varphi_{xM} = \frac{\varphi_{xcw}+\varphi_{xccw}}{2}; \quad \varphi_{yM} = \frac{\varphi_{ycw}+\varphi_{yccw}}{2}$$

$$\varphi_M = \varphi_{xM} - \varphi_{yM}$$

3. **Applying Fitting method**

As we discussed above, all the parameters which are calculated will be applied to the waveform which will in result fit the waveform in correct position.

1) **Offset correction**:

$$X1Cos = XCosDiff - OxM$$

$$Y1Sin = YSinDiff - OyM$$

2) **Amplitude normalisation**

$$X2Cos = \frac{\text{X1Cos}}{\text{AxM}}$$

$$Y2Sin = \frac{\text{Y1Sin}}{\text{AyM}}$$

3) **Non-orthogonality correction**

$$Y3Sin = \frac{\text{Y2Sin} - \text{X2Cos} \cdot \text{Sin}(-\varphi\text{M} \cdot \frac{\pi}{180})}{\text{Cos}(-\varphi\text{M} \cdot \frac{\pi}{180})}$$

4) **Angle calculation**

$$Angle\_Error = \alpha[deg] - \theta[deg]$$

5) **Angular error calculation**

$$\alpha = \arctan\left(\frac{\text{Y3Sin}}{\text{X2Cos}}\right) - \varphi xM$$

6) **Hysteresis Calculation**

$Hysterese\ (\approx 0°) = AngleError_{ccw}(n=64) - AngleError_{cw}(n=0)$
$Hysterese\ (\approx 5{,}625°) = AngleError_{ccw}(n=63) - AngleError_{cw}(n=1)$
$Hysterese\ (\approx 11{,}25°) = AngleError_{ccw}(n=62) - AngleError_{cw}(n=2)$
$Hysterese\ (\approx 360°) = AngleError_{ccw}(n=0) - AngleError_{cw}(n=64)$

Functional test

In the functional test the parameters which are derived above are used to take further readings with the HTOL reliability tests. The Functional test contains a total of 130 readings (CW and CCW) same as the content of calibration test.

The readings are saved in an excel sheet automatically using python code in the backend. So, after running either calibration or functional test all the 130 readings will be written in the mentioned directory as an excel sheet.

4. Objectives

We have two angle sensors from companies Infineon and TDK, which measures angle error. Sensor accuracy changes over a period due to change in parameters like Temperature, Electromagnetic field, self-aging, measurement error and others.

Our goal is to predict the life of angle sensors due to temperature aging, which is achieved by placing temperature sensor in heating oven (environment). This will stress the sensor and hence accelerate age of angle sensor.

Sensor reading is collected at various combination of heating duration and temperature condition. Our goal is to use this data to predict the service life with an HTOL (High Temperature Operating Life) reliability test of angle sensor using machine learning and analytical method. However only our goal is to predict the life cycle and hence reliability of angle sensor from high temperature variation.

We have angle error measurement data for heating duration and temperature as per below table:

		Heating duration				
		0 hr	500 hr	1000 hr	1500 hr	2000 hr
Temperature	125° C	x	x	x	x	x
	150° C	x	x	x	x	x

Table 1- Angle error measurement data

5. Scope

We have received big dataset of angle error measurement in various scenarios. But this dataset is enough to build model directly. Basically, raw data was not in usable condition for machine learning and analytical method because of inconsistency and noise. So we have to pre-process the existing dataset to suit in life time prediction model.

Angle error measurement data is stored in many excel files. All necessary data for model building and testing should be extracted from those excel files. Python script should be developed to extract data from those files for further processing.

Correct data may have noise and inconsistencies. So, we should study the type of noise, develop noise identification, noise classifications and finally noise handling techniques.

Also, existing data is not enough to train model directly. In other to deal with this problem we have to consider multiple method other than analytical method (regression). One good option is model based on machine learning (recurrent neural network) as it usually gives better result even with less dataset. Alone Regression method may not be sufficient due to absence of error reading for more temperature. So, it can be used along with following combination:-

1. HTOL (High Temperature Operating Life)
2. Arrhenius Equation
3. Thermal Acceleration Factor (AF)
4. Effect of AF on the life of a product

Finally, best method for predicting life of angle sensor should be proposed based of accuracy.

6. Literature Survey

AEC-Q100

Electronics play an important role in almost every industry today. As devices and machines become more intelligent, we rely more and more on integrated circuits and other electronic devices. Reliability of these elements is an important issue that needs to be addressed. Failure of one element can affect the overall functioning of smart devices or even lead to complete failure. The standards in this industry are very strict, so even more attention needs to be paid in the automotive industry. As can be seen from Table 3.1, the acceptance error rate is zero ppm, which is not easy to provide.

	Normal application	Automotive application
Ambient Temperature range	0°C to +85°C	-40°C to +150°C
Expected Operating Life	2-3 years	10 years+
Supply Lifetime	2-3 years	15-20 years
Acceptable Failure Rates	300 parts per million	Zero

Table 1- Reliability parameters for ICs in normal application and Automotive application

There are about 1000 electronic elements on a single electronic control units (ECU) board, so a single failure in a million parts will cause 1 in 1000 to fail. The automotive industry is highly competitive, and manufacturers don't want their customers to have a bad experience, so a 1 in 1,000 failure rate is unacceptable.

Additionally, automobiles are exposed to a variety of environmental conditions and stresses, including extreme temperatures, vibrations, and electrical noise. These conditions can adversely affect the performance and reliability of electronic components, and it is important to ensure that these components can withstand these conditions and continue to function properly. The AEC-Q100 standard was developed to address this need, specifying a set of requirements that automotive-grade electronic components must meet to be considered automotive-reliable.

AEC-Q100 is an automotive quality standard that specifies functional, electrical, and environmental performance requirements for integrated circuits (ICs). This standard was developed by the Automotive Electronics Council (AEC), a trade association representing the interests of the automotive electronics industry.

The AEC-Q100 standard aims to ensure that ICs used in automotive applications are highly reliable and can withstand the harsh operating environment of the vehicle. It covers a wide range of performance criteria such as temperature range, humidity, vibration and electrical properties such as supply voltage and current.

ICs that meet the AEC-Q100 standard are qualified for use in automotive applications and are usually marked "Q100". This standard is widely used in the automotive industry and is considered a benchmark for the quality and reliability of ICs used in automotive electronics. An AEC-Q100 qualified device means that the device has passed prescribed stress tests to ensure a certain level of quality/reliability.

In the AEC-Q100 standard, components are divided into different grades based on their intended operating temperature range. These grades are denoted by a number following the "Grade" designation, with higher numbers indicating a wider temperature range. Here is a list of the different grades defined in the AEC-Q100 standard:

Grade	Ambient Operating Temperature Range
0	-40°C to +150°C
1	-40°C to +125°C
2	-40°C to +105°C
3	-40°C to +85°C

Table 2- Grades in AEC-Q100 Standard

There are several different AEC-Q standards that have been developed by the Automotive Electronics Council (AEC). Here is a list of some of the main AEC-Q standards:

AEC-Q100: This is the main standard for automotive-grade electronic components. It defines the quality and reliability requirements for components such as microprocessors, memory devices, sensors, and power management devices.

AEC-Q200: This standard covers the environmental and performance requirements for passive components such as capacitors, resistors, and inductors.

AEC-Q300: This standard covers the environmental and performance requirements for connectors and interconnects.

AEC-Q400: This standard covers the environmental and performance requirements for passive and active components used in hybrid and electric vehicles.

AEC-Q500: This standard covers the environmental and performance requirements for components used in advanced driver assistance systems (ADAS).

AEC-Q600: This standard covers the environmental and performance requirements for components used in infotainment systems.

Each of these standards defines specific requirements and testing procedures for different types of electronic components and is intended to ensure the reliability of these components in the harsh operating environments found in vehicles.

One of the tests that is defined in AEC-Q100 is High Temperature Operating Lifetime(HTOL). This test is presented as an Accelerated Lifetime Simulation Test which guarantees that the IC will not fail after 1000 hours of exposure to high temperatures. Four grades are defined based on environment temperature (T_a). Part of AEC-Q100 standard which is related to this test is provided below:

Table 2: Qualification Test Methods (continued)

TEST GROUP B – ACCELERATED LIFETIME SIMULATION TESTS								
STRESS	ABV	#	NOTES	SAMPLE SIZE / LOT	NUMBER OF LOTS	ACCEPT CRITERIA	TEST METHOD	ADDITIONAL REQUIREMENTS
High Temperature Operating Life	**HTOL**	B1	H, P, B, D, G, K	77	3	0 Fails	JEDEC JESD22-A108	For devices containing NVM, endurance preconditioning must be performed before HTOL per Q100-005. **Grade 0:** +150ºC T_a for 1000 hours. **Grade 1:** +125ºC T_a for 1000 hours. **Grade 2:** +105ºC T_a for 1000 hours. **Grade 3:** +85ºC T_a for 1000 hours. HTOL NOTES: 1) HTOL stress times for the appropriate grade Ta are the min requirement; the Tj of the test (measured or calculated) should be available. 2) Tj may be used instead of Ta when performing HTOL provided that Tj of the device under HTOL conditions is equal to or higher than the Tj maximum operating (Tjopmax) of the particular device, but below the absolute maximum Tj. 3) If Tj is used to set the HTOL conditions, the minimum stress of 1000 hours at the Ta of the device is to be shown using activation energy of 0.7ev or other value technically justified. 4) V_{cc} (max) at which dc and ac parametrics are guaranteed. Thermal shut-down shall not occur during this test. **TEST before and after HTOL at room, cold, and hot temperature (in that order).**
Early Life Failure Rate	**ELFR**	B2	H, P, B, N, G	800	3	0 Fails	AEC Q100-008	Devices that pass this stress can be used to populate other stress tests. Generic data is applicable. **TEST before and after ELFR at room and hot temperature.**
NVM Endurance, Data Retention, and Operational Life	**EDR**	B3	H, P, B, D, G, K	77	3	0 Fails	AEC Q100-005	**TEST before and after EDR at room and hot temperature. Sample size and lot requirement applies to EACH of the NVM tests per Q100-005.**

Figure 8- HTOL test detail in AEC-Q100 standard

7. Concept of Execution

To study the variation of angle sensor error with time, the angle sensor error data was plotted against time on graphs for two different sensor types, one operating at 150°C and the other at 125°C. The data was then analysed by calculating the average of the two highest angle errors for both directional measurements. This provided a linear regression line that was used to understand the trend of angle error over time. The slope of the regression line was calculated to determine the rate of change of angle error with time for each sensor type. The results of this analysis provided a clear picture of how the angle sensor error changes over time for different sensor types and temperatures, giving insight into the lifetime prediction of angle sensors.

Calculating Activation Energy (Ea)

Then, we calculated activation energy from the slopes of the regression lines as follows:

Ea= m*k

m = slopes of regression lines

k = Boltzmann constant (8.617333262145 x 10^-5 eV/K)

High Temperature Operating Life (HTOL)

The concept of execution of the Angle Sensor Lifetime Prediction project will involve a comprehensive study of the High Temperature Operating Life (HTOL) of angle sensors. The HTOL testing was carried out in a controlled environment, where the temperature was gradually increased to simulate the effect of high temperature on the lifetime of the angle sensors.

The Arrhenius equation, which describes the relationship between the rate of a chemical reaction and the temperature of the reaction, was used to analyse the data collected from the HTOL testing. The Arrhenius equation was used to calculate the Thermal Acceleration Factor (AF) for the angle sensors. The Thermal Acceleration Factor represents the degree to which temperature affects the lifetime of a product. By measuring the AF, it was possible to determine the effect of temperature on the lifetime of the angle sensors.

The data collected from the HTOL testing was used to study the effect of the Thermal Acceleration Factor on the life of the angle sensors. The data was analysed to determine the relationship between the AF and the lifetime of the angle sensors. This provided insight into

the factors that affect the lifetime of angle sensors, and was used to develop a model for predicting the lifetime of angle sensors.

The developed model was trained and tested with the collected data, and the results were analyzed to evaluate the performance of the model. The final report presented the results of the HTOL testing, including the Thermal Acceleration Factor and the effect of AF on the life of the angle sensors. It also discussed the developed model and its performance.

In addition, the report also included the detailed methodology and experimental setup used in the HTOL testing and the procedures followed.

The results of this project have provided valuable insights into the lifetime prediction of angle sensors and the developed model could be used to improve the maintenance of angle sensors.

Bathtub Curve

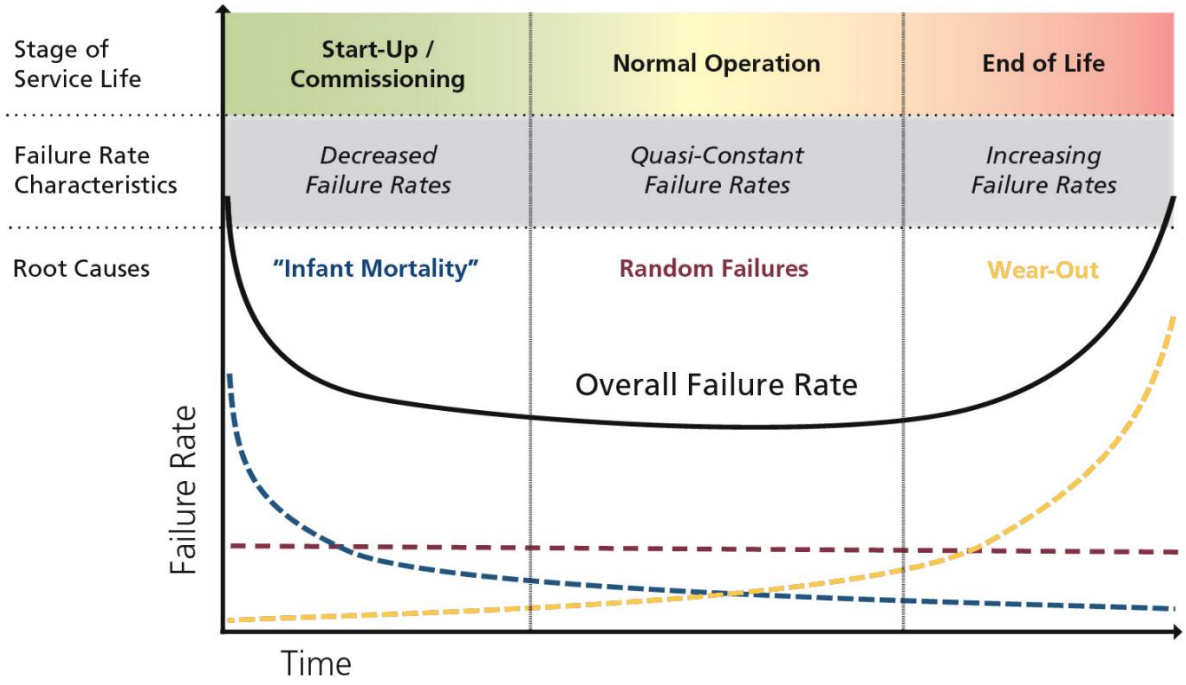


Figure 9- The bathtub curve [https://blog.infraspeak.com/bathtub-curve/]

The bathtub curve is a graphical representation of the failure rate of a product over its lifetime. It is divided into three distinct phases as shown in Figure 9: the early failure phase, the useful life phase, and the wear-out phase.

In the early failure phase, the failure rate is high due to defects and manufacturing problems. This phase is also known as the infant mortality phase. In the useful life phase, the failure rate is relatively low and constant, as the product is functioning well and is not yet subject to wear and tear. In the wear-out phase, the failure rate increases again as the product reaches the end of its useful life and begins to fail due to aging and wear.

The bathtub curve is commonly used in reliability engineering to predict the failure rate of a product over its lifetime. In the context of the Angle Sensor Lifetime Prediction project, the bathtub curve can be used to identify the early failure phase and the wear-out phase of angle sensors, and to predict when these phases are likely to occur. This information can be used to develop a model for predicting the lifetime of angle sensors and to plan for maintenance and replacement of the sensors. Additionally, by studying the bathtub curve, it could be possible to identify the most critical factors that affect the lifetime of angle sensors, such as temperature, vibration, and usage patterns.

Arrhenius Equation

$$\textbf{Process Rate (PR)} = Ae^{-(Ea/kT)}$$

A = A constant

Ea = Thermal activation energy in electron-volts (eV)

k = Boltzman's constant, 8.62×10^{-5} eV/K

T = Absolute temperature in degrees Kelvin (Deg C + 273.15)

The Arrhenius equation describes how the rate of a chemical reaction changes with temperature. The equation shows that as temperature increases, the rate of the reaction also increases. This is because at higher temperatures, more molecules have the energy required to overcome the activation energy barrier and participate in the reaction.

Acceleration Factor

$$AF(T1 \text{ to } T2) = PR2 / PR1 = Ae^{-(Ea/kT2)} / Ae^{-(Ea/kT1)}$$

$$AF(T1 \text{ to } T2) = e^{(Ea/k)(1/T1 - 1/T2)}$$

A = A constant (has canceled out of the formula)

Ea = Thermal activation energy in electron volts (eV)

k = Boltzman's constant, 8.62×10^{-5} eV/K

T = Absolute temperature in degrees Kelvin (degrees C + 273.15)

T1 = product junction (application, operating temperature)

T2 = life-test temperature (125°C, 150°C)

To calculate the Acceleration Factor that is the ratio of the process rate at two temperatures using the Arrhenius equation for the Angle Sensor Lifetime Prediction, the following steps can be followed:

- Determine the product junction or application temperature (T1) for the angle sensor. This could be the operating temperature at which the sensor is typically used.
- Determine the life-test temperature (T2) for the angle sensor. This is typically a higher temperature than the application temperature, such as 125°C or 150°C.
- Measure the activation energy (Ea) of the angle sensor by performing High Temperature Operating Life (HTOL) testing, and using the Arrhenius equation to calculate the activation energy.
- Use the Arrhenius equation to calculate the Acceleration Factor (AF).
- Compare the calculated Acceleration Factor (AF) with a reference value, such as 1. The value of AF greater than 1 means that the product will fail faster at T2 than at T1, and the value of AF less than 1 means that the product will fail slower at T2 than at T1.
- Compare the calculated Acceleration Factor (AF) with a reference value, such as 1. The value of AF greater than 1 means that the product will fail slower at T1 than at T2, and the value of AF less than 1 means that the product will fail faster at T1 than at T2.
- Use the calculated Acceleration Factor (AF) to predict the lifetime of the angle sensor at different temperatures.

Acceleration Factor Calculation Example

Calculate the thermal acceleration factor (AF) between the stress test temperature at 150C and the product junction temperature of **125C:**

T1 (application) = 125C -> 398K

T2 (life-test stress) =150C -> 423K

Ea=0.7eV

$$\textbf{AF(125C to 150C)} = e^{(0.7eV/8.62x10\text{^-}5)(1/398 \; - \; 1/423)} = \textbf{3.34}$$

*t, use (lifetime) = t, stress time * AF*

This implies that one hour of operation at 150°C life-test temperature is equivalent to 3.34 hours of use at the application's 125°C product junction (application, operating temperature).

8. Data Pre-processing

The First step in data processing is to extracting interesting information from the Excel sheets. The Pandas library is used for this. Among the excel sheets, the FF (Funktionsfahrt) sheets are chosen. Sensor ID and angle errors are extracted. Temperature, sensor type, hour, and experiment type (oU/mU) are specified for each reading based on file name and sheet names.

For each sheet 130 rows are extracted. The Step number column is added to define the step motor position for each row which varies from 0 to 129.

	θ[deg]	Angle_Error	Sensor type	mit Umbau	sensor_ID	hour	T	dir	step
0	0.000	0.223380	TDK	False	15	0	150	ccw	0
1	5.958	0.087145	TDK	False	15	0	150	ccw	1
2	11.682	0.072615	TDK	False	15	0	150	ccw	2
3	17.262	0.094996	TDK	False	15	0	150	ccw	3
4	22.797	0.134054	TDK	False	15	0	150	ccw	4
...	...	...	...	...	...	...	...	...	...
70585	23.022	0.281262	TDK	True	8	1500	125	cw	125
70586	17.559	0.239377	TDK	True	8	1500	125	cw	126
70587	11.826	0.353817	TDK	True	8	1500	125	cw	127
70588	6.120	0.240162	TDK	True	8	1500	125	cw	128
70589	0.441	0.193240	TDK	True	8	1500	125	cw	129

70590 rows × 9 columns

Figure 10- sample rows of global data frame

Furthermore, a direction column is added to distinguish clockwise and counterclockwise readings. For each sheet, the first 65 readings are labeled as CCW and next 65 rows are labeled as CW. Totally 70590 rows of data are extracted.

Plotting

Data visualization contributes greatly to perceiving it. For this reason, and also for future use, an individual data frame based on the sensor ID is created for each sensor. Plotting results are provided below, where each chart represents the angle error of the sensor at each heating time.

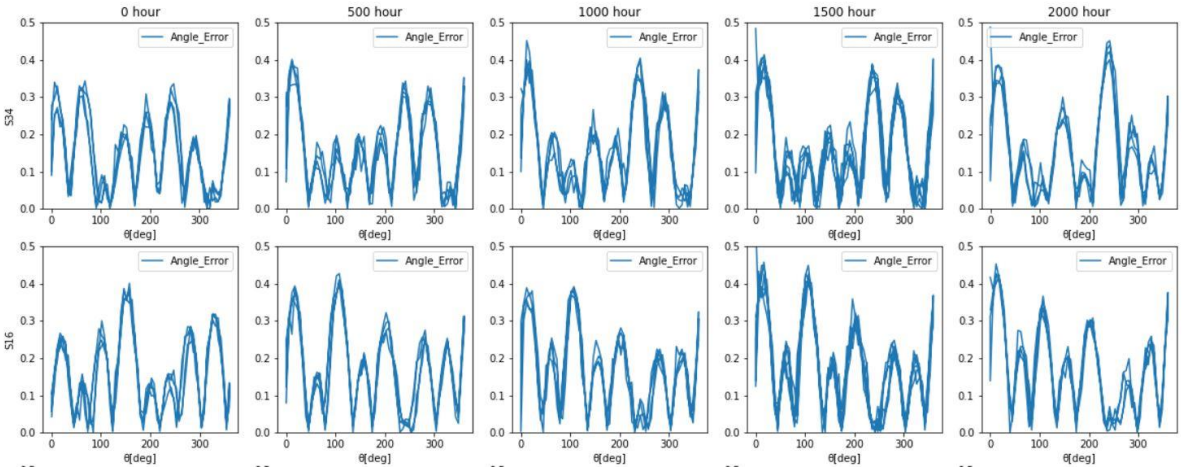


Figure 11- Sample charts for sensor 34 & 16

Outliers

After demonstration the data some outlier points were found as it is shown below.

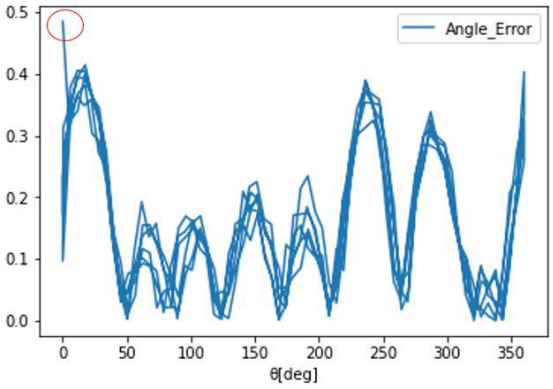


Figure 12- Outlier data

After checking all the datasets, it is detected that the outlier points are only in the starting step which could be due to experimental errors. As a result, the measurement starting point is not a reliable value and should be deleted or replaced with a new value. To maintain the consistency of the dataset, it was decided not to remove these starting points and all the starting points were replaced by the second step measurement error value.

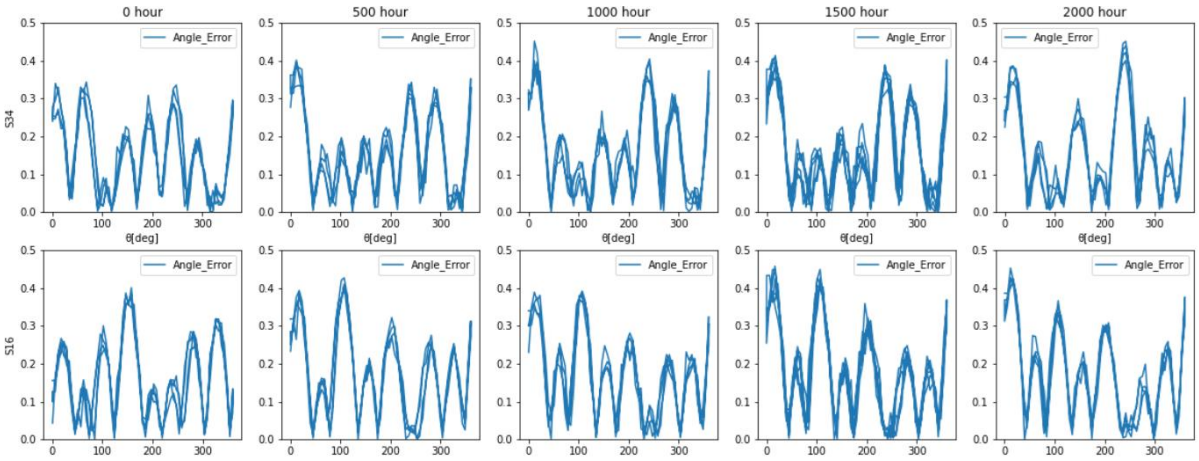


Figure 13- charts for Sensor 16 & 34 after removing outliers

9. Approaches and Results

In this section of the report, we will discuss about different machine learning approaches we took in order to predict the sensor lifetime using all the data at different temperature we had after the HTOL reliability test.

Linear Regression and Arrhenius equation

Till now we have discussed about the literature related to this report and how we can derive an acceleration factor, further in this section we will discuss how we implemented that derivation in our code. We will also discuss how we delt with all the noise in the data that we had, that was a very crucial thing because at first when we tried doing it without cleaning the data and outliers the results didn't make any sense at all. After complete data pre-processing we were able to get good results.

Before discussing directly about the Arrhenius equation and Linear regression, it is necessary to know the pre required steps in order to totally understand the approach. The data we had were in form of excel sheets as mentioned above and data in that format is of no use when we use it directly, we need to sort and assign classes to the data, for example a data frame. It

was quite troublesome to extract only the required data from an entire excel file which consists of 30 different sheets containing different data as shown in the below figure14.

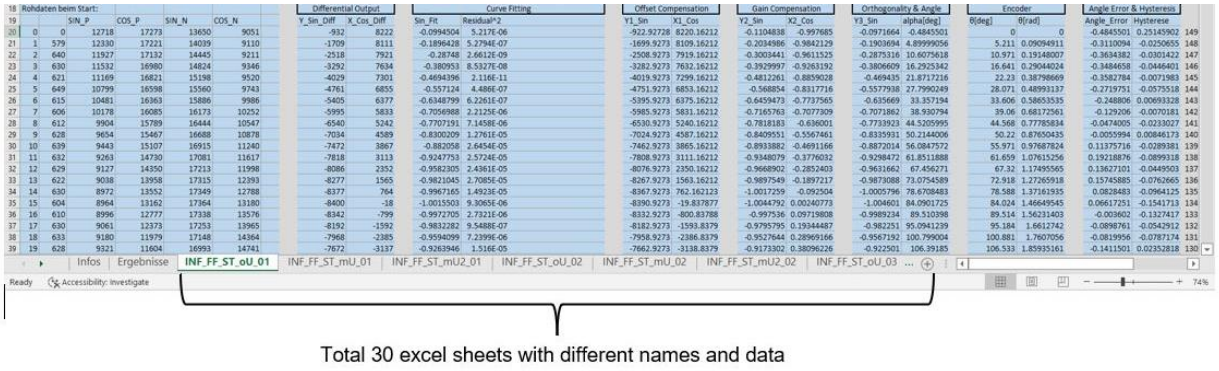


Figure 14- Sample datasheet

For extracting the required data, we made a python code, which will take out the value from a row and column from the excel sheet inside a particular excel file and will keep on appending to the assigned variable, we did that for each class.

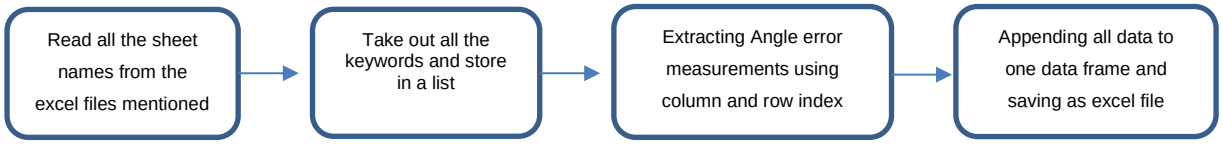
In order to understand about this method, it is necessary to understand how these files were named. If we take an example of a particular file:

INF	Infineon
TDK	TDK
KF	Kalibrierfahrt
FF	Funktionsfahrt
VT	Vergleichsteil
ST	Stressteil
oM	mit Magnet
mM	ohne Magnet
oU	ohne Umbau
mU	mit Umbau

Excel file name: 500h_125C_Stressteils_INF_0Kalibriert
Sheet1: INF_FF_ST_oU_0
Sheet2: INF_FF_ST_mU_01.

The Excel file name means it contains readings form a sensor stressed for 500 hours at 125 degrees Celsius. The rest annotations are shown in the above table.

In this method we used the names of files to sort all the data and finally save into an excel file which can be used directly with the code. The below shown flow chart will show you the clear view of how our code structure for data sorting has been designed.



For reading the names we initiated two lists with names of all the excel files we want to consider from the mentioned folder.

```
os. chdir('Data/')
file_TDK = ["0h_F_125C_Stressteile_TDK.xlsm", "0h_F_150C_Stressteile_TDK.xlsm",
file_INF= ["0h_F_125C_Stressteile_INF.xlsm", "0h_F_150C_Stressteile_INF.xlsm", '
```

Then we defined all the necessary variables along with the final data frames in which the data will be appended.

```
mylist1 = ['oU', 'mU']
rotation = ['CW', 'CCW']
temp_list = []
type_list = []
hour_list = []
temp_list_TDK = []
type_list_TDK = []
hour_list_TDK = []
mylist_INF = ['01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15']
mylist_TDK = ['01', '02', '03', '04', '05', '06', '07', '08', '09']
count = 0
fdata = pd.DataFrame(columns = ['ID', 'Angle_Error', 'rotation', 'umbau', 'type', 'temp', 'hour'])
fdata_TDK = pd.DataFrame(columns = ['ID', 'Angle_Error', 'rotation', 'umbau', 'type', 'temp', 'hour'])
```

Then we set up an ‘else if’ conditions which will be ran on each excel sheets and each excel files of both the sensors using a for loop.

```
for a in file_TDK:
    if "TDK" in a:
        type_list_TDK.append('TDK')
    if "0h_F_" in a:
        hour_list_TDK.append('0')
    elif "500h_G_" in a:
        hour_list_TDK.append('500')
    elif "1000h" in a:
        hour_list_TDK.append('1000')
    elif "1500h" in a:
        hour_list_TDK.append('1500')
    elif "2000h" in a:
        hour_list_TDK.append('2000')
    if "125C" in a:
        temp_list_TDK.append('125')
    if "150C" in a:
        temp_list_TDK.append('150')
```

```
for a in file_INF:
    if "INF" in a:
        type_list.append('INF')
    if "0h_F_" in a:
        hour_list.append('0')
    elif "500h_G_" in a:
        hour_list.append('500')
    elif "1000h" in a:
        hour_list.append('1000')
    elif "1500h" in a:
        hour_list.append('1500')
    elif "2000h" in a:
        hour_list.append('2000')
    if "125C" in a:
        temp_list.append('125')
    if "150C" in a:
        temp_list.append('150')
```

We set up a nested for loop of each variable like direction, temperature, hour, disturbance type and in each iteration, it will go inside the excel file inside the sheet and take out the required data and append it into a specific variable. We do this for both sensors separately.

```
for a in file_INF:
    for z in rotation:
        if "0h_F_" in a:
            col_name = 'w'
            if z == 'CW':
                Skip_Rows = 15
                N_Rows = 65
            elif z == 'CCW':
                Skip_Rows = 80
                No_Rows = 65
        else:
            col_name='ac'
            if z == 'CW':
                Skip_Rows = 19
                N_Rows = 65
            elif z == 'CCW':
                Skip_Rows = 84
                No_Rows = 65
        for y in mylist1:
            for x in mylist_INF:
                data1 = pd.read_excel(a, sheet_name = "{}_FF_ST_{}_{}".format(type_list[count],y,x)
                data1 = abs(data1.values)
                data1 = pd.DataFrame(data1, columns=["Angle_Error"], index = None)
                fdata = pd.concat([fdata,data1], ignore_index=True, sort = False)
```

```
umbau = pd.DataFrame(columns = ['umbau'], index = None)
iD = pd.DataFrame(columns = ['ID'], index = None)
rot = pd.DataFrame(columns = ['rotation'], index = None)
Typ = pd.DataFrame(columns = ['type'], index = None)
Temp = pd.DataFrame(columns = ['temp'], index = None)
Hr = pd.DataFrame(columns = ['hour'], index = None)
for y in list(reversed(mylist1)):
    umbau1 = pd.DataFrame(['{}'.format(y)]*975, columns = ['umbau'], index = None)
    umbau = pd.concat([umbau1, umbau], ignore_index=True, sort = False)
umbau = pd.concat([umbau]*2*len(file_INF),ignore_index=True)
for z in list(reversed(rotation)):
    rotation1 = pd.DataFrame(['{}'.format(z)]*1950, columns = ['rotation'], index = None)
    rot = pd.concat([rotation1, rot], ignore_index=True, sort = False)
rot = pd.concat([rot]*len(file_INF), ignore_index=True)
for tmp in list(reversed(temp_list)):
    Temp1 = pd.DataFrame(['{}'.format(tmp)]*3900, columns = ['temp'], index = None)
    Temp = pd.concat([Temp1, Temp], ignore_index=True, sort = False)
for hr in list(reversed(hour_list)):
    Hr1 = pd.DataFrame(['{}'.format(hr)]*3900, columns = ['hour'], index = None)
    Hr = pd.concat([Hr1, Hr], ignore_index=True, sort = False)
for typ in list(reversed(type_list)):
    Typ1 = pd.DataFrame(['{}'.format(typ)]*3900, columns = ['type'], index = None)
    Typ = pd.concat([Typ1, Typ], ignore_index=True, sort = False)
for x in list(reversed(mylist_INF)):
    iD0 = pd.DataFrame(['{}'.format(x)]*65, columns = ['ID'], index = None)
    iD = pd.concat([iD0, iD], ignore_index=True, sort = False)
iD = pd.concat([iD]*4*len(file_INF),ignore_index=True)
count = count+1
```

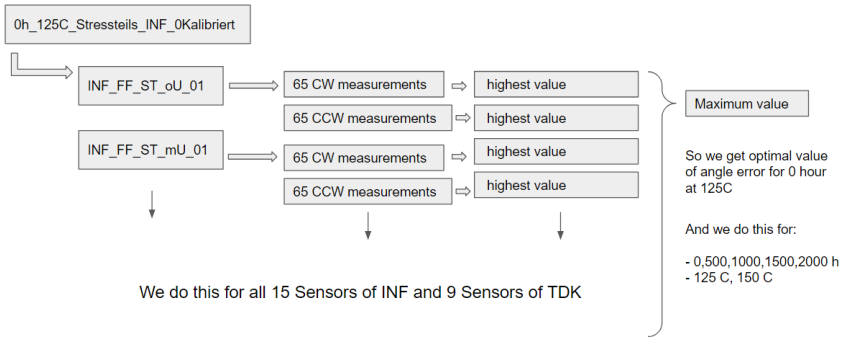
Finally, we get all the data sorted and stored into a two data frames, one for Infineon and another for TDK sensors, then we concept both into one final data frame and save it as an excel file.

```
DATA = pd.concat([fdata_INF,fdata_TDK], ignore_index=True, sort = False)
print(DATA)
```

```
       ID  Angle_Error rotation umbau type temp  hour
0      01     0.253484       CW    oU  INF  125     0
1      01     0.265910       CW    oU  INF  125     0
2      01     0.241020       CW    oU  INF  125     0
3      01     0.214987       CW    oU  INF  125     0
4      01     0.132727       CW    oU  INF  125     0
...    ..          ...      ...   ...  ...  ...   ...
61435  09     0.243152      CCW    mU  TDK  150  2000
61436  09     0.274816      CCW    mU  TDK  150  2000
61437  09     0.291352      CCW    mU  TDK  150  2000
61438  09     0.217006      CCW    mU  TDK  150  2000
61439  09     0.191529      CCW    mU  TDK  150  2000

[61440 rows x 7 columns]
```

```
DATA.to_excel("angle_error_clean.xlsx")
```

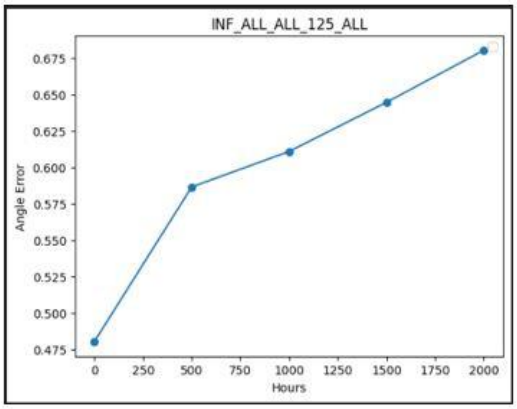
Until this point, we have everything we need to derive the prediction using Arrhenius equation and Linear regression. Now at first, we import and read the previously saved excel sheet containing all the data and save into a data frame, then we iterate through all rows and separate the data according to the attributes in format **variable name = TYPE_ID_ROTATION_TEMP_HOUR**. We must consider the maximum angle generated in one format. To understand this properly investigate the flow diagram below.

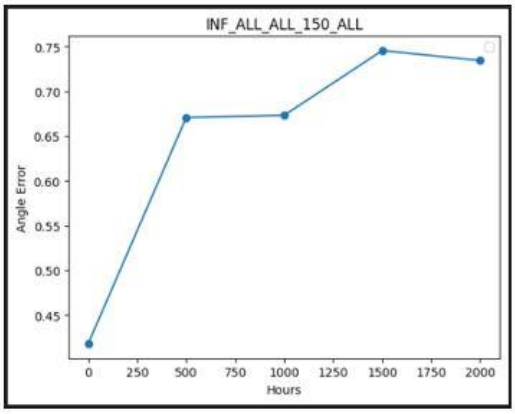


The above is an example of calculating the angle error for **INF_ALL_ALL_125_0** we will do same for all the INF readings at 125 degrees Celsius and then we will append 0h,500h,1000h and 1500h maximum values together to get a final list for **INF_ALL_ALL_125_ALL**.

```
INF_ALL_ALL_125_0 = []
INF_ALL_ALL_125_500 = []
INF_ALL_ALL_125_1000 = []
INF_ALL_ALL_125_1500 = []
INF_ALL_ALL_125_2000 = []

def Average(lst):
    return sum(lst) / len(lst)

for i in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]:
    for r in rot:
        for h in Hour:
            AE = globals()["INF" +"_" + str(i) + "_" + r + "_" + "125" + "_" + str(h)]
            if h == 0:
                INF_ALL_ALL_125_0.append(AE)
                AE=[]
            elif h == 500:
                INF_ALL_ALL_125_500.append(AE)
                AE=[]
            elif h == 1000:
                INF_ALL_ALL_125_1000.append(AE)
                AE=[]
            elif h == 1500:
                INF_ALL_ALL_125_1500.append(AE)
                AE=[]
            elif h == 2000:
                INF_ALL_ALL_125_2000.append(AE)
                AE=[]

# print(INF_ALL_ALL_125_0)
# print(INF_ALL_ALL_125_500)
# print(INF_ALL_ALL_125_1000)
# print(INF_ALL_ALL_125_1500)
# print(INF_ALL_ALL_125_2000)

INF_ALL_ALL_125_0 = max(INF_ALL_ALL_125_0)
INF_ALL_ALL_125_500 = max(INF_ALL_ALL_125_500)
INF_ALL_ALL_125_1000 = max(INF_ALL_ALL_125_1000)
INF_ALL_ALL_125_1500 = max(INF_ALL_ALL_125_1500)
INF_ALL_ALL_125_2000 = max(INF_ALL_ALL_125_2000)

INF_ALL_ALL_125_ALL = [INF_ALL_ALL_125_0, INF_ALL_ALL_125_500, INF_ALL_ALL_125_1000, INF_ALL_ALL_125_1500, INF_ALL_ALL_125_2000]

print(INF_ALL_ALL_125_ALL)
```

We will do same for the 150 C values of INF and for TDK sensors. Then we plot the graph for every lists.





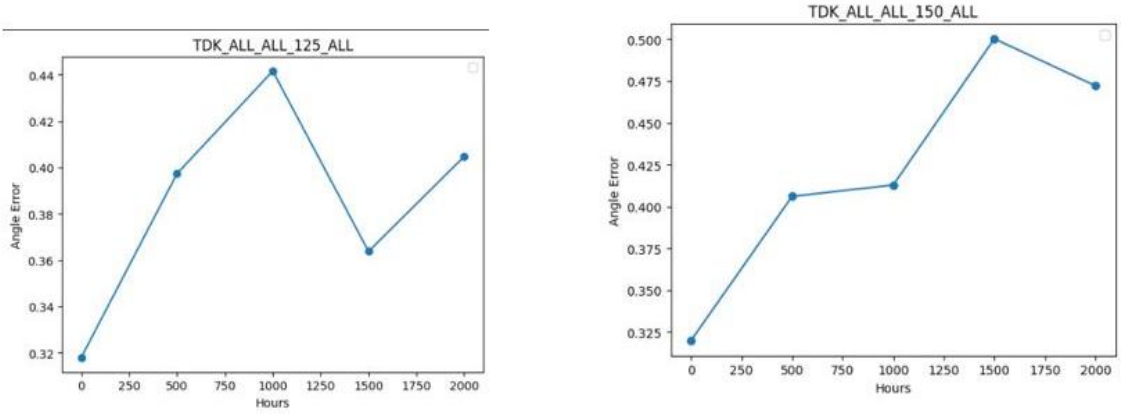


Figure 15- angle error vs hour graph

Now we have required graphs and then we fit the values in the linear regression prediction model and figure out the linear regression for both values of temperature of two sensors. I will now explain the process with the Infineon sensor.

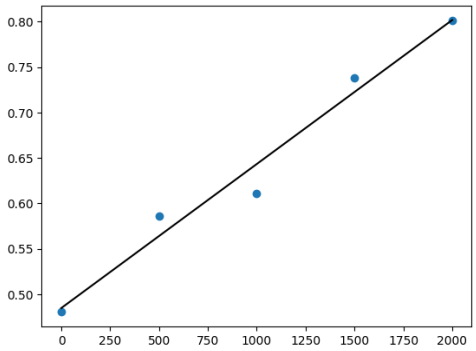


Figure 16- linear regression line @125°

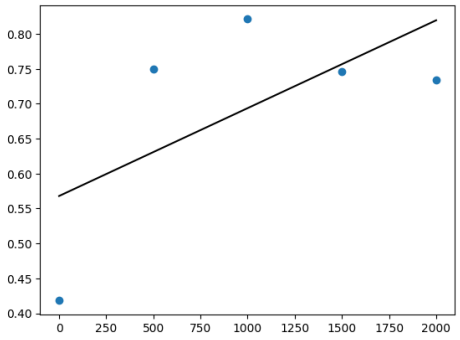


Figure 17- linear regression line @150°

We take these two lines and plot in a single graph for further process. We define drift criterion for the graph, for our model we have set it to 0.56 and then find the intersection point for both lines as shown in the figure 18. We found that the intersection point for 125C line was 560h and for the 150C it was 375h. We also defined sensor failure threshold @ 0.85 degrees. So, we iterate through each prediction of linear regression and find out the failure hours for both temperature range. We found that the sensor fails on 3722 hours at 125C and on 2424 hours at 150C. Now with the intersection points we find,

Two coordinates:

X-axis = 1/ (temperature in K)

Y-axis = ln (intersection point)

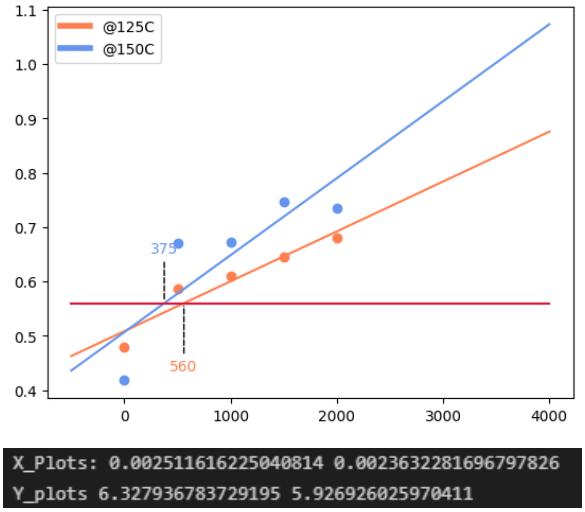


Figure 18-Drift criteria

Figure 19- Final slope

Then we will plot these points as shown in figure 19 and find out the slope of that line with which we will be able find the activation energy Ea with the formula:

Ea = Kb * Slope = 0.0000862*2702.45 = 0.233

Now with the Ea we can find Acceleration factor.

AF = e^((Ea/Kb)*(1/T1-1/T2))

AF = 2.21338

Now using this Acceleration factor, we can predict the lifetime of the sensor at any other given temperature value. The code implementation of this calculation is shown below in figure.

```
Kb = 0.0000862
T1 = input("Please enter temperature in celcius to predict: ") #application temperature (Temp to predict)
T1 = int(T1)
T2 = 150 #life-test temperature (Already know temperature profile)
Hours_survived = 2424

Ea = slope * Kb
print("Ea:", Ea)
AF = e**((Ea/Kb)*((1/(T1+15+273.15))-(1/(T2+15+273.15))))
Hours_to_survive = AF * Hours_survived
print("Activation Function:",AF)
print("@{}C this sensor can survive for maximum {} hours".format(T1,Hours_to_survive))
```

```
Ea: 0.23295087488480518
Activation Function: 2.2133776881161396
@100C this sensor can survive for maximum 5365.2275159935225 hours
```

Results of Arrhenius equation and Linear regression

With this model we tried to predict the approximate hour of survival for the Infineon TLE5501 E001 sensor for different temperature values and it showed relatively good results. For now,

we only data for two temperature range so we tried to predict the survival hours of the sensor at 125C using the acceleration factor found with the 150C temperature range and it showed somewhat close value.

With the linear regression we predicted that the sensor survived for 3722 hours at 125C and with the acceleration factor we predicted the sensor will survive for approximate 3520 hours. So, we can say that this model is ~94% accurate.

The result is shown in the below figure.

```
Kb = 0.0000862
T1 = input("Please enter temperature in celcius to predict: ") #application temperature (Temp to predict)
T1 = int(T1)
T2 = 150 #life-test temperature (Already know temperature profile)
Hours_survived = 2424

Ea = slope * Kb
print("Ea:", Ea)
AF = e**((Ea/Kb)*((1/(T1+15+273.15))-(1/(T2+15+273.15))))
Hours_to_survive = AF * Hours_survived
print("Activation Function:",AF)
print("@{}C this sensor can survive for maximum {} hours".format(T1,Hours_to_survive))
```

✓ 31.1s

```
Ea: 0.23295087488480518
Activation Function: 1.4524058393650405
@125C this sensor can survive for maximum 3520.6317546208584 hours
```

Recurrent Neural Network

A recurrent neural network (RNN) is a type of neural network that is designed to process sequential data, such as time series, sequences of text, and speech [8]. In contrast to feedforward neural networks, which process input data only once and produce a single output, RNNs process input data multiple times, maintaining an internal state that is updated at each time step. This allows RNNs to capture patterns and dependencies in sequential data that are not present in individual input instances.

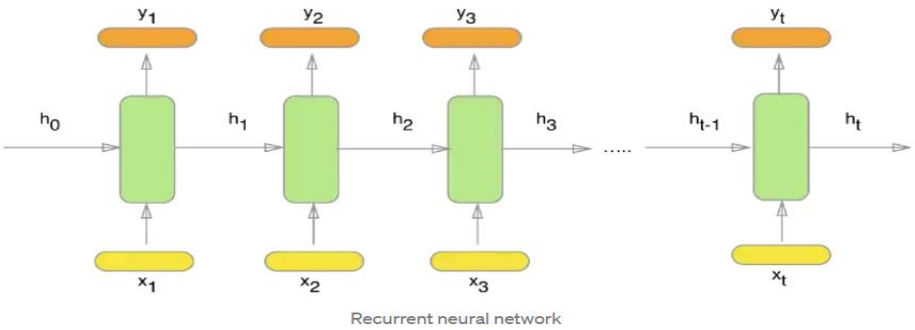


Figure 20- Recurrent Neural Network structure [7]

RNNs can be used to analyse sequential data in a variety of ways, such as:
Language modelling: RNNs can be trained to predict the next word in a sentence, given the previous words.
Speech recognition: RNNs can be trained to transcribe speech to text.
Time series forecasting: RNNs can be trained to predict future values in a time series, such as stock prices or weather forecasts.
Anomaly detection: RNNs can be trained to detect anomalies in sequential data, such as in network traffic or sensor data.
Sentiment analysis: RNNs can be trained to analyse the sentiment of text, such as in social media posts or product reviews.
It is worth mentioning that there are other variants of RNN such as LSTM (Long Short Term Memory) and GRU (Gated Recurrent Units) that are designed to handle the problem of vanishing gradients that traditional RNNs suffers from [3].

Implementation

This part will discuss steps toward implementing RNN models and its results which is provided in 3 sections.

Data providing: Examining the presented data, we can conclude that there are 8 local maxima ranging from 0 to 360 degrees for each graph. Using this idea, the graph is divided into 8 subregions and the maximum angular error for each subregion is calculated. These eight local maxima are then fed as inputs to the RNN model.
Moreover, for each sensor there are two experiments (mit Umbau/ ohne Umbau) and each experiment is consisting of clockwise and counter clockwise values. There are a total of four graphs per sensor per heating time. These four graphs are extracted and considered new sensor data. This way the dataset is quadrupled. These four graphs are extracted and considered as new sensor data. In this way our dataset would be quadrupled.
To summarize, for each sensor we make 4 samples to feed to our RNN model. Model is aiming to predicting maximum error in 2000 hour. For this reason, 8 local maximum values in each heating hour are extracted. Eventually inputs will be arrays of size 4×8 where 4 represents four different time frames and 8 represents local maximums in each time frame. For providing input and output out of each data frame, max_8df () function is developed.
As we got good results in predicting 2000-hour maximum error using data from 0-500-1000-1500 hour, we decided to do the prediction also by using 0-500-1000 hour, and 0-500 and even just 0 hour. The results for each prediction will be discussed in result section.

Model development: The angular error data provided in this project comes from two different manufacturers, each with two sets of sensors heated at different temperatures. Therefore, different models were developed for each class, labelled INF125, INF150, TDK125 and TDK150, indicating the manufacturer and heating temperature.

As already mentioned, in addition to the simple RNN model, GRU and LSTM models are available which include some gates to tackle the vanishing gradient problem. All three kinds of RNN models are developed in this project to compare and evaluate them. Below a sample code for creating a simple RNN model for INF125 class is provided.

```
[161] model_RNN_INF125=Sequential()
      model_RNN_INF125.add(SimpleRNN(50, activation='relu' ,input_shape=(4,8)))
      model_RNN_INF125.add(Dense(1))
      model_RNN_INF125.compile(optimizer='adam', loss='mse')
      model_RNN_INF125.summary()
```

```
Model: "sequential_40"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_12 (SimpleRNN)   (None, 50)                2950

 dense_40 (Dense)            (None, 1)                 51

=================================================================
Total params: 3,001
Trainable params: 3,001
Non-trainable params: 0
_________________________________________________________________
```

Figure 21-Sample code for creating simple RNN model

As GRU and LSTM models also includes gates, they have more parameters comparing to simple RNN with same number of layers. In below table, different models with 50 layers are compared regarding their number of parameters to train.

Model	Number of parameters
RNN_50	2950
GRU_50	9000
LSTM_50	11800

Table 3- Different RNN models and corresponding number of parameters

Result: Each classes samples are spilite to train and test datasets where the models are trained using train dataset and then Mean Square Error (MSE) is computed for test dataset. For checking the reliability and for better comparison each model is trained 3 times with

different train and test datasets and the average value is computed. Below results for INF125 are provided.

using 0-1500 hours

Model Loss	1	2	3	Avg
LSTM50	0.000881	0.0014	0.001	0.001094
GRU50	0.0015	0.0011	0.0021	0.001567
RNN50	0.0011	0.0015	0.000938	0.001179

using 0-500 hours

Model Loss	1	2	3	Avg
LSTM50	0.0014	0.0013	0.0017	0.001467
GRU50	0.0013	0.0015	0.0013	0.001367
RNN50	0.0027	0.002	0.0024	0.002367

using 0-1000 hours

Model Loss	1	2	3	Avg
LSTM50	0.0011	0.0011	0.0011	0.0011
GRU50	0.000876	0.0011	0.0011	0.001025
RNN50	0.0014	0.0011	0.0011	0.0012

using Just 0 hours

Model Loss	1	2	3	Avg
LSTM50	0.0041	0.0033	0.003	0.003467
GRU50	0.003	0.0025	0.0031	0.002867
RNN50	0.003	0.0022	0.0024	0.002533

Figure 22- MSE values for INF125

Four tables are shown in above figure where each one corresponds to predicting the 2000-hour error using different groups of time. MSE error for 0–1500-hour group is magnificently small for all three models. Eliminating 1500 hours data does not affect the prediction performance that much. MSE error increases from 0-1500 group to 0-500 group meaningfully. Although the increase is much higher when the prediction is done just by using 0 hour data. All three different RNN models are performing mostly the same.

Other results are also provided as below.

using 0-1500 hours

Model Loss	1	2	3	Avg
LSTM50	0.000928	0.000667	0.001	0.000865
GRU50	0.0625	0.0013	0.000885	0.021562
RNN50	0.000698	0.000736	0.000622	0.000685

using 0-500 hours

Model Loss	1	2	3	Avg
LSTM50	0.0017	0.0011	0.001	0.001267
GRU50	0.000935	0.000807	0.000721	0.000821
RNN50	0.000522	0.0018	0.001	0.001107

using 0-1000 hours

Model Loss	1	2	3	Avg
LSTM50	0.000543	0.00052	0.00045	0.000504
GRU50	0.000479	0.000859	0.0008	0.000713
RNN50	0.000886	0.0014	0.0018	0.001362

using Just 0 hours

Model Loss	1	2	3	Avg
LSTM50	0.0029	0.0029	0.0029	0.0029
GRU50	0.0025	0.0014	0.0017	0.001867
RNN50	0.001	0.0019	0.0021	0.001667

Figure 23-MSE values for TDK125

using 0-1500 hours

Model Loss	1	2	3	Avg
LSTM50	0.0014	0.000792	0.0013	0.001164
GRU50	0.0011	0.000707	0.0014	0.001069
RNN50	0.0011	0.000792	0.001	0.000964

Table 4- Results for INF150 using 0-1500 hours data

using 0-1500 hours

Model Loss	1	2	3	Avg
LSTM50	0.0022	0.0602	0.0021	0.0215
GRU50	0.0022	0.0015	0.0015	0.001733
RNN50	0.0016	0.000849	0.002	0.001483

Table 5- results for TDK150 using 0-1500 hours data

10. Risks & Limitations

Imbalanced dataset: In classification methods, labels in the dataset should be distributed equally to prevent bias in final model. As it is provided in below figure, most of the graphs in our dataset in each class behaves same over time. As a result, classification methods couldn't be used in this case study.

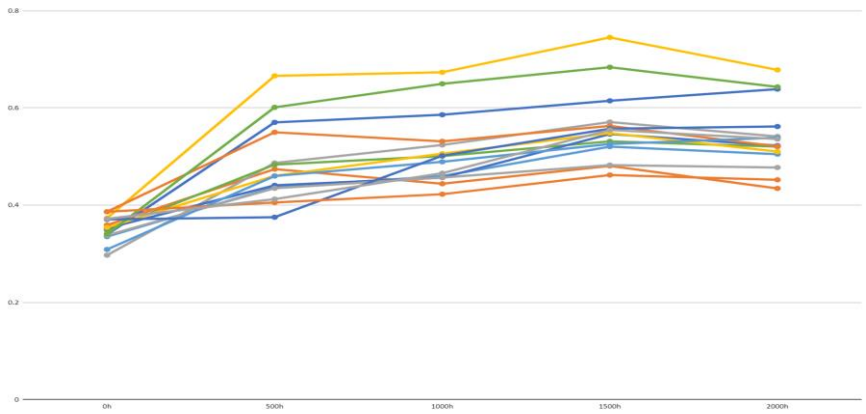


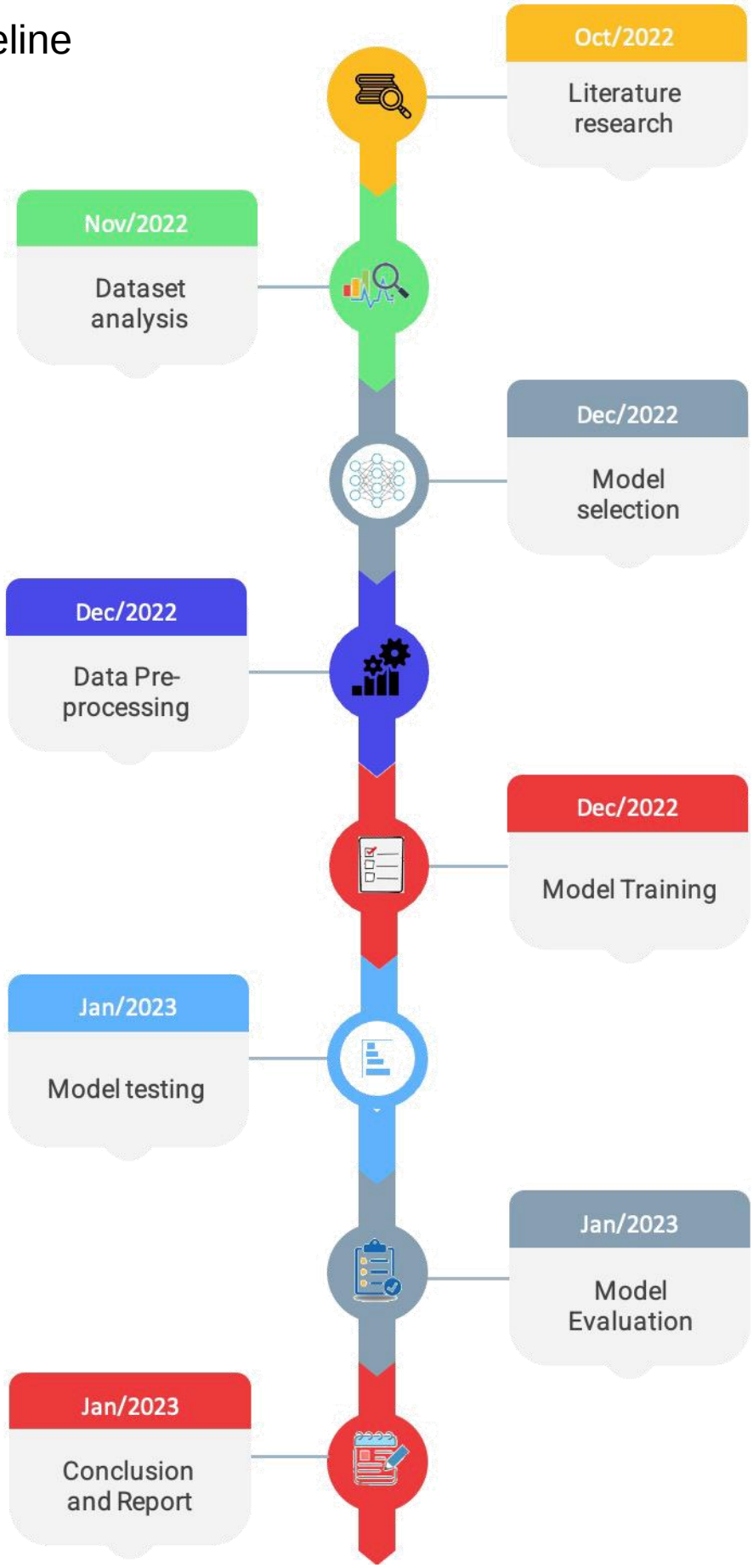
Figure 24- maximum error angle over time

Hysteresis: As these sensors work based on magnetic field variations, their output is affected by hysteresis. For coping with this issue, the direction of each reading is considered in dataset, and it will consider in model developing.

Experimental error: As the creating process is semi-automatic, some errors due to human mistake could be in dataset. The outlier's problem is explained and discussed under data pre-processing topic.

Lack of data: The provided data are from 4 different time points which are not enough for making a accurate prediction. For further work it is suggested to measuring angle error in more time points like 24h, 48h and so on.

11. Timeline


Oct/2022
Literature research
Nov/2022
Dataset analysis
Dec/2022
Model selection
Dec/2022
Data Pre-processing
Dec/2022
Model Training
Jan/2023
Model testing
Jan/2023
Model Evaluation
Jan/2023
Conclusion and Report

12. Next Steps

The next steps for the Angle Sensor Lifetime Prediction project can involve the collection of more detailed data on the angle error of sensors. This data can be collected at more frequent time intervals, including 0, 24, 48, 72, 168, 240, 500, 1000, 1500, and 2000 hours.

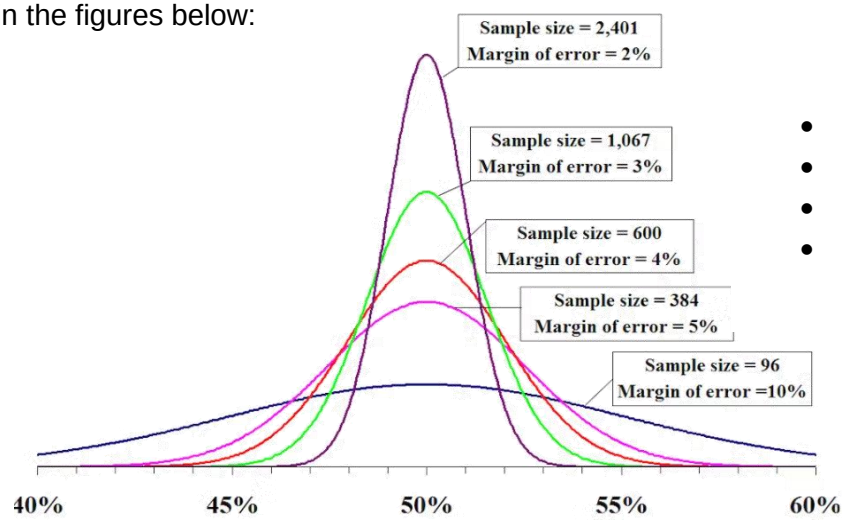
Additionally, measurement stability and sensor stability can be calculated after certain measurements to better understand the factors affecting the lifetime of the sensors.

To improve the precision of the data, the lifetime drift of electrical parameters and the angle error of the sensors can be studied. This can provide a more comprehensive view of the factors affecting sensor lifetime, allowing for more accurate and precise predictions.

Using this data, the machine learning model can be fine-tuned and optimized to calculate sensor lifetime predictions with a high degree of accuracy and precision. This can involve the use of advanced techniques such as recurrent neural networks (RNNs) and the incorporation of additional data such as temperature, vibration, and usage patterns. The fine-tuned model can be tested with the new data and compared with the previous results to evaluate the performance.

The project can also explore the possibility of applying the model to other types of sensors to validate the generality of the model and its potential for broader applications in various industries.

The statistical relationship between the number of samples and the margin of error is shown in the figures below:



- 48 sensor data
- 5 time intervals
- 5.41% margin of error
- 10.82 point confidence interval

Figure 25- Confidence interval

https://commons.wikimedia.org/wiki/Category:Confidence_interval

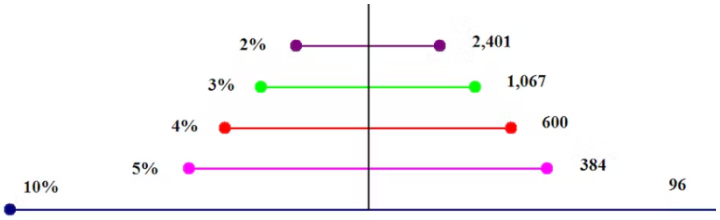


Figure 26-Confidence interval statistic
[https://commons.wikimedia.org/wiki/Category:Confidence_interval]

- 48 senspr data
- 10 time intervals
- ~6 point confidence interval
- < 3% margin of error

13. Conclusion

- Angle sensor lifetime predicted analysed using Regression & Arrhenius Equation (analytical method) and Recurrent Neural network (LSTM).
- Data was pre-processed to meet the requirement of analytical method and Recurrent Neural network.
- Python script was developed for extract and process error angle measurement data from excel files.
- Exploratory data analysis (EDA) was done to study the data pattern.
- Following methods were studies for dealing with Outliers or noise.
    1. Sigma limit
    2. Average of top n values
    3. First values
- “First value” method was turned out be most effective for dealing with outliers.

14. References

[1] Wittmann Jürgen and Bergholz, W. (2019) Quality Management in Technology 2019.

[2] Info.accelrf.com [Why Test for Reliability in ICs?]. Retrieved December 21 2022, from https://info.accelrf.com/blog/why-test-for-reliability-in-ics.

[3] Anysilicom.com. [A tribute to Failure Analysis Engineers]. Retrieved December 21, 2022, from https://anysilicon.com/dr-morris-chang-hero-tribute-failure-analysis-engineers.

[4] Infraspeak. [Bathtub Curve: what is it and how to adjust maintenance]. Retrieved November 22, 2020, from https://blog.infraspeak.com/bathtub-curve/.

[5] Wikimedia.org. [Confidence Interval]. Retrieved December 21, 2022, from https://commons.wikimedia.org/wiki/Category:Confidence_interval.

[6] Automotive Electronics Council. (2001) STRESS TEST QUALIFICATION FOR INTEGRATED CIRCUITS.AEC - Q100 - REV-E

[7] https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

[8] https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

[9] https://www.infineon.com/cms/en/podurct/sensor/magnetic-sensors/magnetic-position-sensors/angle-sensors/tle5501-e0001/

[10] https://product.tdk.com/en/search/sensor/angle/tmr-angle/info?part_no=TAS2141-AAAB

[11] Master thesis: Establishment of a measurement laboratory to investigate the 0h performance and early failure behavior of analog magnetoresistive angle sensors.

15. Appendix

```
# %%
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# %%
DATA = pd.read_excel("angle_error_clean.xlsx")
print(DATA)

# %%
Type = ['INF', 'TDK']
rot = ['CW','CCW']
Temp = [125,150]
Hour = [0,500,1000,1500,2000]
top_max_val = 3

#variable name = TYPE_ID_ROTATION_TEMP_HOUR

for t in Type:
    if t == 'INF':
        iD = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
    else:
        iD = [1,2,3,4,5,6,7,8,9]
    for i in iD:
        for r in rot:
            for x in Temp:
                for y in Hour:
                    globals()[t +"_" + str(i) + "_" + r + "_" + str(x) + "_" + str(y)] =
DATA.loc[(DATA['ID'] == i) & (DATA['rotation'] == r) & (DATA['type'] == t) & (DATA['temp'] ==
x) & (DATA['hour'] == y)]["Angle_Error"].nlargest(n=1).mean()

# %%
INF_ALL_ALL_125_0 = []
INF_ALL_ALL_125_500 = []
INF_ALL_ALL_125_1000 = []
INF_ALL_ALL_125_1500 = []
INF_ALL_ALL_125_2000 = []

def Average(lst):
    return sum(lst) / len(lst)

for i in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]:
    for r in rot:
        for h in Hour:
            AE = globals()["INF" +"_" + str(i) + "_" + r + "_" + "125" + "_" + str(h)]
            if h == 0:
                INF_ALL_ALL_125_0.append(AE)
                AE=[]
            elif h == 500:
                INF_ALL_ALL_125_500.append(AE)
                AE=[]
```

```
            elif h == 1000:
                INF_ALL_ALL_125_1000.append(AE)
                AE=[]
            elif h == 1500:
                INF_ALL_ALL_125_1500.append(AE)
                AE=[]
            elif h == 2000:
                INF_ALL_ALL_125_2000.append(AE)
                AE=[]

# print(INF_ALL_ALL_125_0)
# print(INF_ALL_ALL_125_500)
# print(INF_ALL_ALL_125_1000)
# print(INF_ALL_ALL_125_1500)
# print(INF_ALL_ALL_125_2000)

INF_ALL_ALL_125_0 = max(INF_ALL_ALL_125_0)
INF_ALL_ALL_125_500 = max(INF_ALL_ALL_125_500)
INF_ALL_ALL_125_1000 = max(INF_ALL_ALL_125_1000)
INF_ALL_ALL_125_1500 = max(INF_ALL_ALL_125_1500)
INF_ALL_ALL_125_2000 = max(INF_ALL_ALL_125_2000)

INF_ALL_ALL_125_ALL = [INF_ALL_ALL_125_0, INF_ALL_ALL_125_500,
INF_ALL_ALL_125_1000, INF_ALL_ALL_125_1500, INF_ALL_ALL_125_2000]

print(INF_ALL_ALL_125_ALL)

# %%
plt.plot(Hour,INF_ALL_ALL_125_ALL, marker = 'o')
plt.title('INF_ALL_ALL_125_ALL')
plt.xlabel('Hours')
plt.ylabel('Angle Error')
plt.legend()
plt.show()

# %%
INF_ALL_ALL_150_0 = []
INF_ALL_ALL_150_500 = []
INF_ALL_ALL_150_1000 = []
INF_ALL_ALL_150_1500 = []
INF_ALL_ALL_150_2000 = []

def Average(lst):
    return sum(lst) / len(lst)

for i in [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]:
    for r in rot:
        for h in Hour:
            AE = globals()["INF" +"_" + str(i) + "_" + r + "_" + "150" + "_" + str(h)]
            if h == 0:
                INF_ALL_ALL_150_0.append(AE)
                AE=[]
            elif h == 500:
                INF_ALL_ALL_150_500.append(AE)
```

```
            AE=[]
        elif h == 1000:
            INF_ALL_ALL_150_1000.append(AE)
            AE=[]
        elif h == 1500:
            INF_ALL_ALL_150_1500.append(AE)
            AE=[]
        elif h == 2000:
            INF_ALL_ALL_150_2000.append(AE)
            AE=[]

# print(INF_ALL_ALL_150_0)
# print(INF_ALL_ALL_150_500)
# print(INF_ALL_ALL_150_1000)
# print(INF_ALL_ALL_150_1500)
# print(INF_ALL_ALL_150_2000)

INF_ALL_ALL_150_0 = max(INF_ALL_ALL_150_0)
INF_ALL_ALL_150_500 = max(INF_ALL_ALL_150_500)
INF_ALL_ALL_150_1000 = max(INF_ALL_ALL_150_1000)
INF_ALL_ALL_150_1500 = max(INF_ALL_ALL_150_1500)
INF_ALL_ALL_150_2000 = max(INF_ALL_ALL_150_2000)

INF_ALL_ALL_150_ALL = [INF_ALL_ALL_150_0, INF_ALL_ALL_150_500,
INF_ALL_ALL_150_1000, INF_ALL_ALL_150_1500, INF_ALL_ALL_150_2000]

print(INF_ALL_ALL_150_ALL)

# %%
plt.plot(Hour,INF_ALL_ALL_150_ALL, marker = 'o')
plt.title('INF_ALL_ALL_150_ALL')
plt.xlabel('Hours')
plt.ylabel('Angle Error')
plt.legend()
plt.show()

# %%
TDK_ALL_ALL_125_0 = []
TDK_ALL_ALL_125_500 = []
TDK_ALL_ALL_125_1000 = []
TDK_ALL_ALL_125_1500 = []
TDK_ALL_ALL_125_2000 = []

def Average(lst):
    return sum(lst) / len(lst)

for i in [1,2,3,4,5,6,7,8,9]:
    for r in rot:
        for h in Hour:
            AE = globals()["TDK" + "_" + str(i) + "_" + r + "_" + "125" + "_" + str(h)]
            if h == 0:
                TDK_ALL_ALL_125_0.append(AE)
                AE=[]
            elif h == 500:
```

```
            TDK_ALL_ALL_125_500.append(AE)
            AE=[]
        elif h == 1000:
            TDK_ALL_ALL_125_1000.append(AE)
            AE=[]
        elif h == 1500:
            TDK_ALL_ALL_125_1500.append(AE)
            AE=[]
        elif h == 2000:
            TDK_ALL_ALL_125_2000.append(AE)
            AE=[]

# print(INF_ALL_ALL_125_0)
# print(INF_ALL_ALL_125_500)
# print(INF_ALL_ALL_125_1000)
# print(INF_ALL_ALL_125_1500)
# print(INF_ALL_ALL_125_2000)

TDK_ALL_ALL_125_0 = max(TDK_ALL_ALL_125_0)
TDK_ALL_ALL_125_500 = max(TDK_ALL_ALL_125_500)
TDK_ALL_ALL_125_1000 = max(TDK_ALL_ALL_125_1000)
TDK_ALL_ALL_125_1500 = max(TDK_ALL_ALL_125_1500)
TDK_ALL_ALL_125_2000 = max(TDK_ALL_ALL_125_2000)

TDK_ALL_ALL_125_ALL = [TDK_ALL_ALL_125_0, TDK_ALL_ALL_125_500,
TDK_ALL_ALL_125_1000, TDK_ALL_ALL_125_1500, TDK_ALL_ALL_125_2000]

print(TDK_ALL_ALL_125_ALL)

# %%
plt.plot(Hour,TDK_ALL_ALL_125_ALL, marker = 'o')
plt.title('TDK_ALL_ALL_125_ALL')
plt.xlabel('Hours')
plt.ylabel('Angle Error')
plt.legend()
plt.show()

# %%
TDK_ALL_ALL_150_0 = []
TDK_ALL_ALL_150_500 = []
TDK_ALL_ALL_150_1000 = []
TDK_ALL_ALL_150_1500 = []
TDK_ALL_ALL_150_2000 = []

def Average(lst):
    return sum(lst) / len(lst)

for i in [1,2,3,4,5,6,7,8,9]:
    for r in rot:
        for h in Hour:
            AE = globals()["TDK" + "_" + str(i) + "_" + r + "_" + "150" + "_" + str(h)]
            if h == 0:
                TDK_ALL_ALL_150_0.append(AE)
                AE=[]
```

```
        elif h == 500:
            TDK_ALL_ALL_150_500.append(AE)
            AE=[]
        elif h == 1000:
            TDK_ALL_ALL_150_1000.append(AE)
            AE=[]
        elif h == 1500:
            TDK_ALL_ALL_150_1500.append(AE)
            AE=[]
        elif h == 2000:
            TDK_ALL_ALL_150_2000.append(AE)
            AE=[]

# print(INF_ALL_ALL_125_0)
# print(INF_ALL_ALL_125_500)
# print(INF_ALL_ALL_125_1000)
# print(INF_ALL_ALL_125_1500)
# print(INF_ALL_ALL_125_2000)

TDK_ALL_ALL_150_0 = max(TDK_ALL_ALL_150_0)
TDK_ALL_ALL_150_500 = max(TDK_ALL_ALL_150_500)
TDK_ALL_ALL_150_1000 = max(TDK_ALL_ALL_150_1000)
TDK_ALL_ALL_150_1500 = max(TDK_ALL_ALL_150_1500)
TDK_ALL_ALL_150_2000 = max(TDK_ALL_ALL_150_2000)

TDK_ALL_ALL_150_ALL = [TDK_ALL_ALL_150_0, TDK_ALL_ALL_150_500,
TDK_ALL_ALL_150_1000, TDK_ALL_ALL_150_1500, TDK_ALL_ALL_150_2000]

print(TDK_ALL_ALL_150_ALL)

# %%
plt.plot(Hour,TDK_ALL_ALL_150_ALL, marker = 'o')
plt.title('TDK_ALL_ALL_150_ALL')
plt.xlabel('Hours')
plt.ylabel('Angle Error')
plt.legend()
plt.show()

# %%
print(INF_ALL_ALL_125_ALL)

# %%
from sklearn.linear_model import LinearRegression
from matplotlib.lines import Line2D
from math import log,e

Hour_n = [-500,0,500,1000,1500,2000,2500,3000,3500,4000]

model = LinearRegression()
model.fit(np.array([Hour]).reshape((-1, 1)),np.array([INF_ALL_ALL_125_ALL]).reshape((-1,
1)))

model1 = LinearRegression()
```

```
model1.fit(np.array([Hour]).reshape((-1, 1)),np.array([INF_ALL_ALL_150_ALL]).reshape((-1,
1)))

X_predict = [[-500],[0],[500],[1000],[1500],[2000],[2500],[3000],[3500],[4000]]
y_predict = model.predict(X_predict)
y_predict1 = model1.predict(X_predict)

drift_criterion = 0.56
Thershold = 0.85

#To find the intersection with limit line
x_125=0
x_150=0
y_150=0
y_125=0
while y_125 < drift_criterion:
    x_125+=1
    y_125 = model.predict([[x_125]])
print("125C:",y_125[0][0],x_125)
while y_150 < drift_criterion:
    x_150+=1
    y_150 = model1.predict([[x_150]])
print("150C:",y_150[0][0],x_150)

#To find the failure hour point of the sensor
x_125_failure=2000
x_150_failure=2000
y_150_failure=0
y_125_failure=0
while y_125_failure < Thershold:
    x_125_failure+=1
    y_125_failure = model.predict([[x_125_failure]])
print("For 125C Sensor Failed @: {}h".format(x_125_failure))
while y_150_failure < Thershold:
    x_150_failure+=1
    y_150_failure = model1.predict([[x_150_failure]])
print("For 150C Sensor Failed @: {}h".format(x_150_failure))

plt.scatter(np.array([Hour]).reshape((-1, 1)),np.array([INF_ALL_ALL_125_ALL]).reshape((-1,
1)), color='coral')
plt.plot(np.array([Hour_n]).reshape((-1, 1)), y_predict, color='coral')
plt.scatter(np.array([Hour]).reshape((-1, 1)),np.array([INF_ALL_ALL_150_ALL]).reshape((-1,
1)), color='cornflowerblue')
plt.plot(np.array([Hour_n]).reshape((-1, 1)), y_predict1,color='cornflowerblue')
plt.plot([-500,4000],[drift_criterion,drift_criterion],color='crimson')

arrowprops={'arrowstyle': '-', 'ls':'--'}
plt.annotate(str(x_125), xy=(x_125,y_125[0][0]), xytext=(x_125, 0.1),
             textcoords=plt.gca().get_xaxis_transform(),
             arrowprops=arrowprops,
             va='top', ha='center',color='coral')
plt.annotate(str(x_150), xy=(x_150,y_150[0][0]), xytext=(x_150, 0.4),
             textcoords=plt.gca().get_xaxis_transform(),
             arrowprops=arrowprops,
```

```
        va='top', ha='center',color='cornflowerblue')

custom_lines = [Line2D([0], [0], color='coral', lw=4),
                Line2D([0], [0], color='cornflowerblue', lw=4)]
plt.legend(custom_lines, ['@125C', '@150C'])


# %%
x_125_plotx = 1/(125+273.15)
x_150_plotx = 1/(150+273.15)
print("X_Plots:",x_125_plotx,x_150_plotx)
print("Y_plots",log(x_125),log(x_150))
plt.plot([x_125_plotx,x_150_plotx],[log(x_125),log(x_150)], color='coral', marker = 'o')
slope = (log(x_125) - log(x_150))/(x_125_plotx - x_150_plotx)
print("Slope:", slope)

# %%
Kb = 0.0000862
T1 = input("Please enter temperature in celcius to predict: ") #application temperature (Temp
to predict)
T1 = int(T1)
T2 = 150 #life-test temperature (Already know temperature profile)
Hours_survived = 2424

Ea = slope * Kb
print("Ea:", Ea)
AF = e**((Ea/Kb)*((1/(T1+15+273.15))-(1/(T2+15+273.15))))
Hours_to_survive = AF * Hours_survived
print("Activation Function:",AF)
print("@{}C this sensor can survive for maximum {} hours".format(T1,Hours_to_survive))
```