الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

UNIVERSITE BADJI MOKHTAR - ANNABA
BADJI MOKHTAR – ANNABA UNIVERSITY

جامعة باجي مختار – عنابـــة

Faculty : TECHNOLOGY

Department : ELECTRONIC

Domain : SCIENCES AND TECHNICS

Sector : AUTOMATIC

Specialty : AUTOMATIC AND INFORMATIC INDUSTRIAL

THESIS

Presented in order to obtain the diploma of Master

Entitled :

Modeling and fault diagnosis based multivariate statistical method. Theoretical development and implementation using Raspberry-Pi card

Presented by: *TOUATI Kaddour*

Supervisor : *HARKAT Mohamed Faouzi* *Grade Prof* *Université UBMA*

Committee Members:

CHAKER Karima	M.C.B	UBMA	President
HARKAT Mohamed Faouzi	Prof	UBMA	Supervisor
AIT IZEM Tarek	M.C.B	UBMA	Reviewer

Année Universitaire : 2021/2022

kaddour.touati1@gmail.com

To the memory of my father Rebani TOUATI
(1953-2006).
To my wonderful mother.
To my family
To my beloved wife

Acknowledgements

More than 16 years ago, I graduated from Badji Mokhtar Annaba University with the rank of State Engineer, and after being busy with improving my professional career, the idea of returning to university studies was almost impossible. I did it, although it was a difficult challenge, but Allah granted me success. First,I would like to give my sincerest thanks to my supervisor ***Prof. Mohamed-Faouzi Harkat*** *for his support and guidance in order to finalize this dissertation.*

My appreciation also goes out to my family and friends for their encouragement and support all through my studies.

I would also like to thank ''Master 2 AII'' students for the good treatment and the respect they showed me.

Abstract

The work presented in this thesis is one of the required means and contribute to the development of the industrial technologies sector, which is diagnosis and fault detection system in industrial processes. It intends to promptly detect and identify abnormalities and enhance the reliability and safety of the processes where we chose the KPCA which is one of the statistical methods based on data processing. Kernel Principal Component Analysis (KPCA) is a widely used for diagnosis and fault detection method. It has gained much interest due to its ability in monitoring of nonlinear systems.

- In the first chapter, a reminder of the principle of different diagnosis methods is provided, including of-course the statistical methods that we adopted in our project.

- Chapter 2 presents theoretical background of principal component analysis (PCA), Kernel PCA (KPCA), and Reduced KPCA (RKPCA) approach. The RKPCA is developed as a solution to eliminate the weak point of KPCA method which is the high computing time and large storage space when a large-sized training dataset is used.

- The third chapter presents the Raspberry Pi Card (model 4B), and brief definition of Python Language, the combination of the two tools (Hardware and software) is used for the implementation of theoretical study.

- The final chapter is devoted to an application on a real process, which is the chemical process TENNESSEE EASTMAN. By presenting the implementation under the Raspberry Pi card and Python, make two different simulation using conventional KPCA and the new scheme RKPCA, affront the results and demonstrating the suitability and feasibility of the proposed method to be implemented on a real process.

Table of Contents

List of Tables

List of Figures

List of Abbreviations and Terms

X	training data matrix
x	measurement vector
m	number of variables
N	number of samples
Σ	covariance matrix of X
Λ	diagonal eigenvalue matrix
σ	width of a Gaussian function
ϕ	mapping function
K	kernel matrix
PCA	Principal Component Analysis
KPCA	Kernel Principal Component Analysis
RBF	Radial Basis Function
ℓ	number of retained principal components
RKPCA	reduced kernel principal component analysis
CPV	Cumulative Percentage of Variance
T^2	Hotelling's statistic
Q	squared prediction error
φ	combined statistic
$\tau_\alpha^{T^2}$	control limit associated with T^2 index
τ_α^{Q}	control limit associated with Q index
τ_α^{φ}	control limit associated with φ index
FAR	False Alarm Rate
MDR	Missed Detection Rate

Introduction

Due to the intense concurrence in the field of industrial processes, it has been resorted to the automation as a solution. But one of the important challenges that faces this solution is the Fault Detection in order to ensure productivity, improve the quality, increase production process utilization and reduce maintenance costs. Various Fault Detection methodologies have been developed. For example: model-based and Data-driven methods. Model-based methods such as the observer-based method, parity space and parameter estimation techniques used mainly mathematical models to generate residuals that are used to detect the faults ([1] [2] [3]). Data-driven approaches are seen as the most cost-effective technique for dealing with plant-wide industrial complex processes. Multivariate statistical process monitoring (MSPM) is considered as one of the data-driven approaches. It has been developed and widely used for process monitoring in chemical systems. Principal component analysis (PCA) possibly is the most popular among these (MSPM) methods. PCA seeks of axes that maximize the variance of data, making it easy to evaluate the error by projecting observations onto residual axes. It has been applied successfully for fault detection in linear systems ([4];[5];[6]; [7]). However, PCA performs poorly when the process exhibits strong nonlinear correlations between its variables, to address this issue, many studies have been developed by means of kernel machines. Particularly, kernel principal component analysis (KPCA) is proposed as a generalization of PCA to nonlinear cases [8]. The main idea of KPCA is to map the input data into a high-dimensional feature space via nonlinear mapping functions, which make data structure more linear, and then perform PCA in that feature space. By using KPCA for Fault Detection, the fault detection is presented by error indices such as the squared prediction error (SPE) or Q, which monitors the residual subspace, while the Hotelling statistics T 2 is for monitoring the principal subspace, and a combination of both indices is added that monitors the whole measurement space. Despite its accuracy and successful, KPCA suffers from a high computational cost and requires more memory storage space. As an alternative and effective method for dealing with the problem of required storage space and computation time when using Kernel principal component (KPCA) for faults detection in processes with nonlinear nature, a novel reduced KPCA (RKPCA) scheme is developed [8], which is represented in deep details in chapter 2, while the first chapter represents the different approaches of diagnosis and fault detection. Chapter 3 includes brief definition of the proposed implementation tools, which are the Raspberry Pi cards (model 4B) and Python codes. The last chapter is dedicated for performance test and evaluation of RKPCA for fault detection by implementation on the Tennessee Eastman process using Raspberry Pi card and Python codes.

Chapter 1

Process monitoring

1.1 Introduction

As a result of the continuous development of industrial facilities, it is necessary to take into account the proper functioning of these systems to ensure safety, discover the defect and treat it in a timely manner. These necessities conduct to create different mechanisms for diagnosing and discovering the defect. Below we briefly present some methods of diagnosis from different points of view.

1.2 Fault detection and diagnosis

For better overview of our topic, we introduce some definitions used in the diagnosis domain [9] :

- Normal operation of a system: A system is in situation of normal operation when their characterizing variables (output, input, system parameters) remain in the vicinity of their values nominal, otherwise the system is failed.
- Fault: is a deviation of an observed variable, or a process parameter, outside an acceptable range [10] .
- Failure: is the cause of an anomaly, such as a failure of a cooling or a regulator.
- Fault detection: is a decision about whether the system is not in a normal operating state.
- Location of the fault: After detection of a defect, the task is to determine the location or the elements at the origin of the defect. Following the figure 1.1, we can summarize three types of faults [11]: Sensor faults, Actuator faults and Process faults

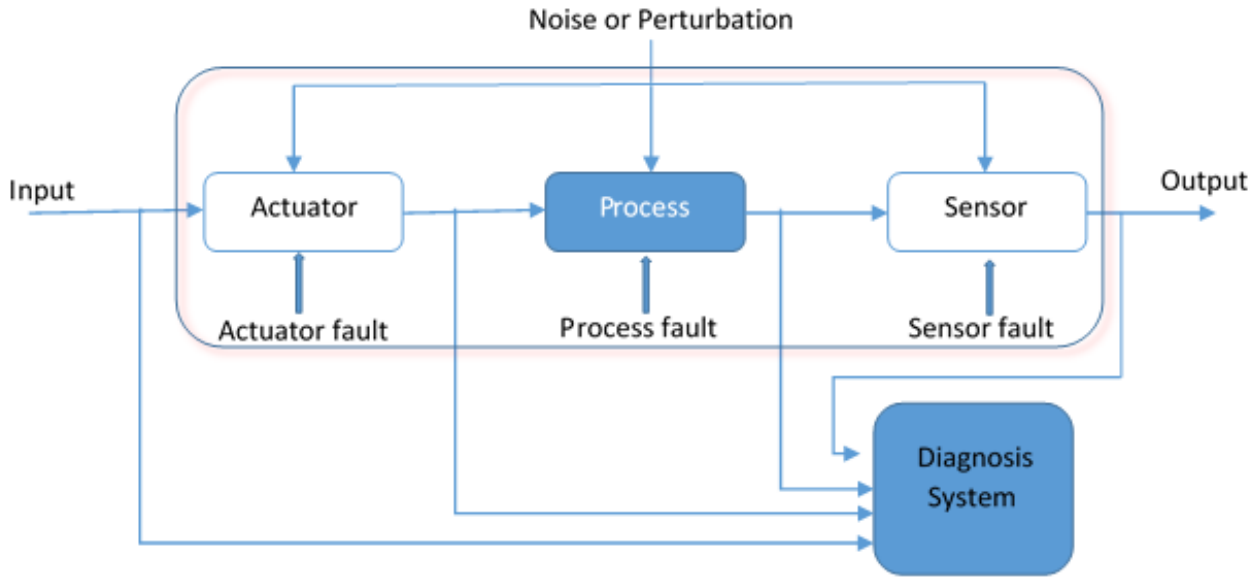


Figure 1.1: General diagram of a diagnosis system

1.2.1 Sensor faults:

A sensor is a device that transforms the state of a physical quantity measured in a usable quantity. The sensors are distinguished from the measure by the fact that it is only a simple interface between a process information. They are used to communicate the information about the state and internal behavior of the process. Thus, a sensor fault characterizes wrong information of the physical size to be measured.

- For closed-loop systems, measurements from these sensors are used to generate the control signal. Therefore, the development of the control signal is ineffective, if the information taken into account by the command algorithm are wrong and/or inconsistent. Therefore, the presence of a sensor defect gives an inaccurate and ineffective control signal.

1.2.2 Actuator faults:

The actuator is an element of the operative part capable of producing a physical phenomenon (displacement, heat release, light emission...), actuators transform a type energy to another. Consequently, the actuator defects act at the level of the operational. It adds up to the systemâs control signals, and generate problems related to organs which affect the state of the system.

1.2.3 Process faults:

Process faults are faults that affect the process of the system itself. These are the faults that cannot be classified nor among the faults of actuators neither nor among the sensor faults.

1.3 Diagnosis principle and Definitions

1.3.1 Principle of diagnosis:

Generally, faults in an industrial system can occur on each of these three parts:

- The actuators
- The process

- The sensors

The fault diagnosis therefore consists in the determination of the type, amplitude, location and moment of occurrence, it includes three successive steps:

- Detection of the fault
- The isolation of the fault
- Identification of the fault

1.3.2 Definition

Monitoring a system is a continuous real-time task to determine system status. It is done through the recording of information that indicate the occurrence of any anomalies in the behavior of the system. As for supervision, it consists in making appropriate decisions, when the monitoring stage of the system, in order to maintain the nominal operation of the system despite the appearance of defects. All these tasks aim to ensure the optimal performance of the system, in terms of availability, reliability and maintainability. This is equivalent to preventing failure. When diagnosing a system, it is appropriate to differentiate between faults and disturbance. A disturbance is an unknown and uncontrolled input that acts on a system. However the fault is internal to the system.

1.4 Classification of Fault Diagnosis Methods

The classification of diagnosis methods are related to the available knowledge of the process or its representation and are classified different ways, there are methods that require accurate system models (plants), quantitative models or qualitative models. However, there are methods that do not require any form of model information and rely only on historic system data. While there have been some excellent reviews in the field of fault diagnosis, it is of interest that classification of fault diagnosis methods very often is not consistent. This is mainly due to the fact that researchers are often focused on a particular branch, such as analytical models, of the broad discipline of fault diagnosis. As shown in Figure 1.2, fault diagnosis methods are broadly classified into three main categories ([12] [13]; [14]; [15]; [16]): model-based, hardware-based and history-based.

1.4.1 Model-Based Fault Diagnosis

Model-based fault diagnosis methods usually deploy a model developed based on some fundamental understanding of the physics of the plant or process. In general, model-based fault diagnosis methods are broadly classified as qualitative or quantitative.

- Qualitative methods Qualitative model-based fault diagnosis methods utilise a model where the inputâoutput relationship of the plant is expressed in terms of qualitative functions cantered around different units in the process. Qualitative model-based fault diagnosis is broadly classified into abstraction hierarchy, fault trees, diagraphs and fuzzy systems.
- Quantitative Methods Quantitative model-based fault diagnosis methods utilise a model where the inputâoutput relationship of the plant is expressed in terms of mathematical functions. As shown in Figure 1.2, quantitative model-based fault diagnosis is broadly classified into analytical redundancy, rarity space, Kalman filter (KF), parameter estimation and diagnostic observers.

1.4.2 Hardware-Based Fault Diagnosis

Hardware-based fault diagnosis methods do not deploy a mathematical model of the physics of the plant or process. In general, hardware-based fault diagnosis methods are broadly classified into hardware redundancy, voting techniques, special hardware, limit checking and frequency analysis.

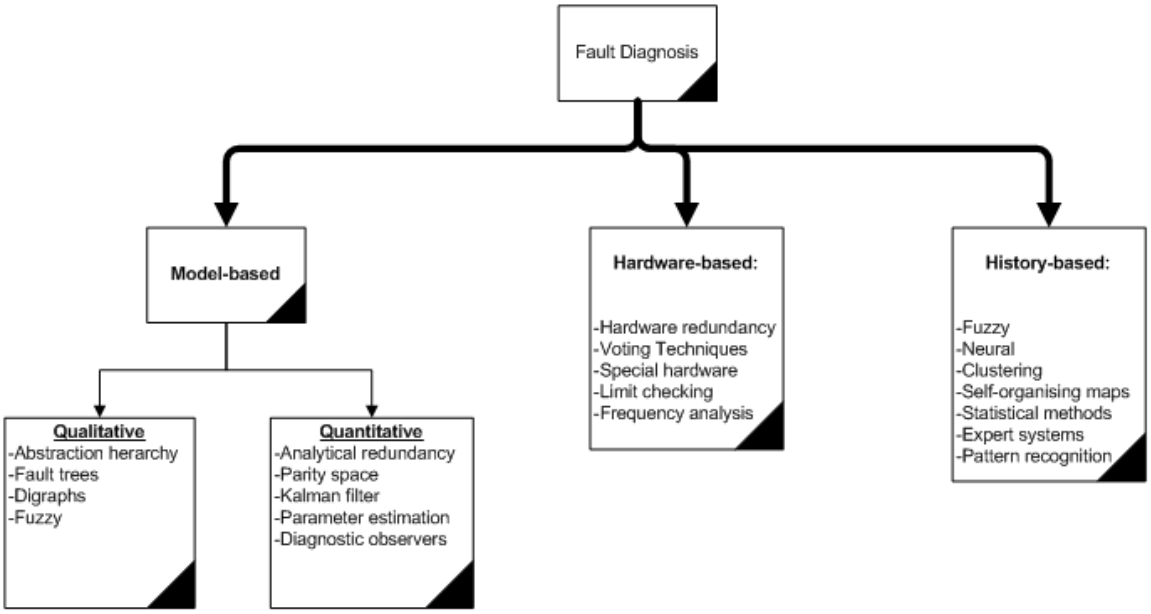


Figure 1.2: Classification of fault diagnosis methods

1.4.3 History-Based Fault Diagnosis

In fault diagnosis literature, one can find a huge overlap between model-based fault diagnosis and history-based fault diagnosis. As previously mentioned, model-based fault diagnosis methods usually deploy a model developed based on some fundamental understanding of the physics of the plant or process. History-based fault diagnosis methods do not deploy a mathematical model of the physics of the plant or process, but a model derived from known and measured input and output process data. The fundamental idea of history-based fault diagnosis is to generate a model of the process, which mathematically relates measured inputs to measured outputs, and then use this model against the real process to generate residual. In general, history-based fault diagnosis methods are broadly classified into FL, neural networks, clustering, self-organising maps (SOM), statistical methods, experts systems and pattern recognition.

1.5 The different steps of diagnosis

- Data acquisition: The diagnosis methods requires the availability of information related to the system that will be monitored, the following functions must be carried out: - signal conditioning and pre-processing. - validation of the measurement signal.
- developing fault indicators: Based on the measurements taken and the observations from the operators in charge of the installation, it is a question of building indicators making it possible to highlight any faults that may appear within the system.
- Detection stage: This is the operation that makes it possible to decide whether the system is in normal operation or not.
- Localisation: localisation step follows the detection step, it assigns the fault to a particular subsystem (sensor, actuator, control device,process...).
- Decision-making: the incorrect functioning of the system having been observed, it is a question of deciding on the procedure to be followed in order to maintain the desired performance of the

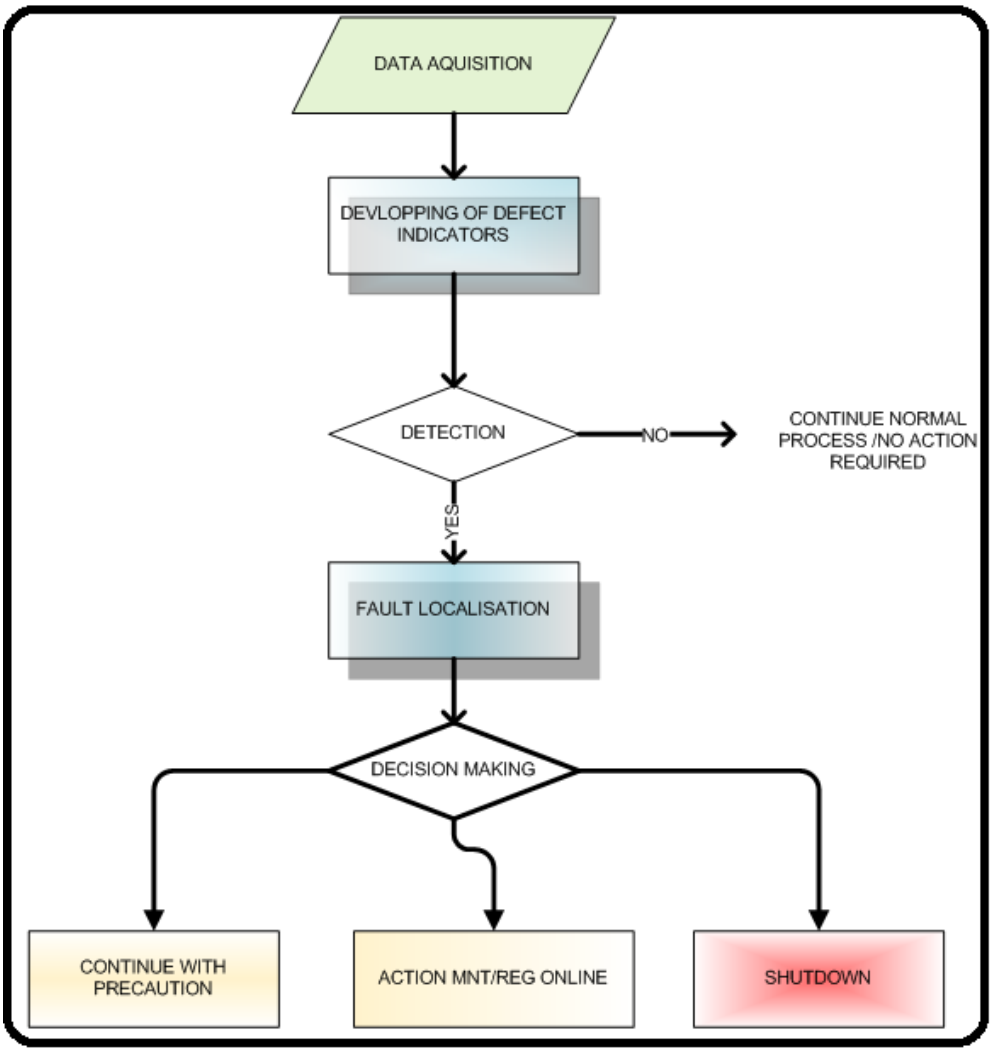


Figure 1.3: The different steps of a diagnosis system

monitored system .

1.6 Conclusion

The purpose of diagnosis is to detect rapidly the various defects existing on a process (lack of instrumentation, parameters, structural) to avoid degradation of its performance and increase the safety of operators and the environment, the choice of suitability fault diagnosis methods is primarily a question of the quality of the available mathematical model of the system, knowledge of the system and system structure. In addition to this, the reachable quality of fault isolation decisively depends on the number of available measurements. Among these methods, we have chosen the method that is compatible with our project, which is statistical methods, and one of its techniques (PCA, KPCA) will be detailed.

Chapter 2

Statistical process monitoring and KPCA background

2.1 Introduction

As previously explained about the fault diagnosis methods, we have chosen the way that fit with our project, which is a statistical method. Principal component analysis (PCA) possibly is the most popular among these statistical methods, PCA seeks of axes that maximize the variance of data, making it easy to evaluate the error by projecting observations onto residual axes. It has been applied successfully for fault detection in linear systems, Kernel principal component analysis (KPCA) is proposed as a generalization of PCA to non linear cases [17, 18, 19, 20, 6]. unfortunately, the use of Kernel PCA for fault detection represents an issue of storage space requirement and computation time, a novel reduced KPCA (RKPCA) scheme is developed as an effective alternative in order to resolve the issue [8]. in this chapter we will demonstrate a details about the proposed RKPCA method and its performance on the Tennessee Eastman processes.

2.2 Conventional PCA method

PCA is a widely used multivariate statistical method, which can transform the original variables into a set of new orthogonal variables, so that most information is contained in the first few components with the largest variance [21, 22, 23].

Let $\boldsymbol{x}(k) \in \mathbb{R}^m$ denotes a sample measurement vector of m sensors at time k. Assuming that there are N samples, a data matrix $X = [\boldsymbol{x}(1)\, \boldsymbol{x}(2) \ldots \boldsymbol{x}(N)]^T \in \mathbb{R}^{N \times m}$ is standardized to zero mean and unit variance, then PCA can be performed through the eigenvalue decomposition of the covariance matrix of X, Σ.

$$\Sigma = P \Lambda P^T \tag{2.1}$$

where $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_m)$ is the diagonal eigenvalue matrix ($\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_m$ are the eigenvalues) and $P = (p_1, p_2, ..., p_m)$ is the eigenvector matrix (p_i, $i = 1, 2, ..., m$, represent the normalized and mutually orthogonal eigenvectors associated with the eigenvalues). Then, the matrix X is transformed into independent variables T through:

$$T = XP \tag{2.2}$$

$T = [T_1, ..., T_m]$ contains the principal components, which are orthogonal to each other. In summary, the PCA model is determined based on an eigen-decomposition of the covariance matrix Σ and the selection of the number ℓ of components to be retained. Eigenvalues, eigenvectors and principal

components matrices can be partitioned as:

$$\Lambda = \begin{bmatrix} \Lambda_\ell & 0 \\ 0 & \Lambda_{m-\ell} \end{bmatrix} \tag{2.3}$$

$$P = \begin{bmatrix} P_\ell & P_{m-\ell} \end{bmatrix}, \qquad T = \begin{bmatrix} T_\ell & T_{m-\ell} \end{bmatrix} \tag{2.4}$$

where ℓ represents the number of retained principal components to be kept in the PCA model.
By taking into account the first ℓ highest eigenvalues and their corresponding eigenvectors, the matrix X is decomposed as:

$$X = T_\ell P_\ell^T + E \tag{2.5}$$

where $T_\ell = XP_\ell$ and E is the residual matrix.
A sample vector $\mathbf{x}(k) \in \mathcal{R}^m$ can be projected onto the principal and residual subspaces, respectively,

$$\begin{aligned} \hat{\boldsymbol{x}}(k) &= P_\ell \boldsymbol{t}_\ell(k) \\ &= C_\ell \mathbf{x}(k) \end{aligned} \tag{2.6}$$

where $\hat{\boldsymbol{x}}(k)$ is the estimation vector of $\boldsymbol{x}(k)$, $C_\ell = P_\ell P_\ell^T$ and,

$$\boldsymbol{t}_\ell(k) = P_\ell^T \boldsymbol{x}(k) \in \mathbb{R}^\ell \tag{2.7}$$

is the vector of the first ℓ scores of latent variables.
The vector of $m - \ell$ last scores of latent variables, which represents the projection of measurement data in the residual subspace, is given by:

$$\boldsymbol{t}_{m-\ell}(k) = P_{m-\ell}^T \boldsymbol{x}(k) \in \mathbb{R}^{m-\ell} \tag{2.8}$$

2.2.1 KPCA method

The main idea of KPCA is to map data into a feature space via a nonlinear mapping and then a linear PCA is performed in feature space. Given a set of normalized training data $X = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & . & . & . & \boldsymbol{x}_N \end{bmatrix}^T \in \mathbb{R}^{N \times m}$, where m is the number of process variables and N is the number of measurements. A non linear mapping in the feature space $\mathcal{H}$, $\phi : \boldsymbol{x}_i \in \mathcal{R}^m \rightarrow \phi_i = \phi(\boldsymbol{x}_i) \in \mathbb{R}^h$ maps the training dataset into a high dimensional feature space, where $h >> m$ is the dimension in feature space. An important property of the feature space is that the dot product of two vectors $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j), i, j = 1, ..., N$, can be determined as:

$$\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) = \boldsymbol{k}(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.9}$$

where $\boldsymbol{k}$ is the kernel function. One of the most used kernel functions is the radial basis function (RBF) which is given by:

$$\boldsymbol{k}(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp\left[\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right] \tag{2.10}$$

where σ is the width of a Gaussian function which controls the flexibility of the kernel. As recommended in [24], a typical choice for σ is the average minimum distance (d) between two points in the training dataset, i.e., $\sigma^2 = c\frac{1}{N}\sum_{i=1}^{N} \min_{j \neq i} d^2(\boldsymbol{x}_i, \boldsymbol{x}_j)$ where c is a user defined parameter.
Assuming that the vectors in the feature space are scaled to zero mean and unit variance, the mapped data is arranged as $\mathcal{X} = [\phi(\boldsymbol{x}_1)\ \phi(\boldsymbol{x}_2) \ldots \phi(\boldsymbol{x}_N)]^T$. The covariance matrix $\mathcal{C}$ of the dataset in the feature space is defined as follows:

$$\begin{aligned} (N-1)\mathcal{C} &= \mathcal{X}^T \mathcal{X} \\ &= \textstyle\sum_{i=1}^{N} \phi_i \phi_i^T \end{aligned} \tag{2.11}$$

Solving the following eigenvector equation is equivalent of KPCA in the feature space

$$\begin{aligned} \mathcal{X}^T\mathcal{X}\boldsymbol{v} &= \sum_{i=1}^{N} \phi_i\phi_i^T\boldsymbol{v} \\ &= \lambda\boldsymbol{v} \end{aligned} \tag{2.12}$$

The mapping function ϕ_i is not explicitly defined, one can evaluate the Gram matrix $\mathcal{X}^T\mathcal{X}$ using the kernel function $\boldsymbol{k}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$.
With the use of the kernel trick, one can define the matrix K with $\boldsymbol{k}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ elements,

$$\begin{aligned} K = \mathcal{X}\mathcal{X}^T &= \begin{bmatrix} \phi_1^T\phi_1 & ... & \phi_1^T\phi_N \\ . & & . \\ . & ... & . \\ . & & . \\ \phi_N^T\phi_1 & ... & \phi_N^T\phi_N \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{k}(\boldsymbol{x}_1, \boldsymbol{x}_1) & ... & \boldsymbol{k}(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ . & & . \\ . & ... & . \\ . & & . \\ \boldsymbol{k}(\boldsymbol{x}_N, \boldsymbol{x}_1) & ... & \boldsymbol{k}(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} \end{aligned} \tag{2.13}$$

KPCA seeks to resolve the eigenvector equation in the feature space. Let $\boldsymbol{\alpha}$ be an eigenvector of the matrix K and λ its corresponding eigenvalue,

$$\boldsymbol{v} = \lambda^{-1}\mathcal{X}^T\boldsymbol{\alpha} \tag{2.14}$$

The matrix of the ℓ retained principal loading of the KPCA in the feature space is denoted by $P = [\boldsymbol{v}_1, ..., \boldsymbol{v}_\ell] \in \mathbb{R}^{N\times\ell}$ and the $N-\ell$ last principal loading is denoted by $\tilde{P} = [\boldsymbol{v}_{\ell+1}, ..., \boldsymbol{v}_N] \in \mathbb{R}^{N\times(N-\ell)}$.

$$P = \left[\tfrac{1}{\lambda_1}\mathcal{X}^T\boldsymbol{\alpha}_1, ..., \tfrac{1}{\lambda_\ell}\mathcal{X}^T\boldsymbol{\alpha}_\ell\right] \tag{2.15}$$

For a given measurement $\boldsymbol{x}$ and its mapped vector $\phi = \phi(\boldsymbol{x})$, the scores are calculated as,

$$\boldsymbol{t} = P^T\phi \in \mathbb{R}^\ell \tag{2.16}$$

$$\tilde{\boldsymbol{t}} = \tilde{P}^T\phi \in \mathbb{R}^{N-\ell} \tag{2.17}$$

2.3 Proposed methodology

The dimension of the KPCA model depends on the number of samples of the training dataset. Reducing the KPCA model leads to reduce this number of samples. Therefore, a new scheme based on PCA dimension reduction technique is proposed. It aims to extract only uncorrelated observations from the training dataset to be used in developing an appropriate KPCA model [25, 8]. As presented in section 2.2, PCA technique can be used to reduce the size of a given dataset . Let us consider training data matrix $X = [\boldsymbol{x}_1\, \boldsymbol{x}_2 \,...\, \boldsymbol{x}_N]^T \in \mathbb{R}^{N\times m}$ is standardized to zero mean and unit variance where $\boldsymbol{x}_k \in \mathbb{R}^m$ is a sampled vector at time k of m variables and N represents the number of observations. To reduce the number of samples N of the original data matrix X, PCA is performed through the eigenvalue decomposition of the covariance matrix of $Y \in \mathbb{R}^{m\times N}$, such that $Y = X^T$,

$$\Sigma_Y = P_Y\Lambda_Y{P_Y}^T \in \mathbb{R}^{N\times N} \tag{2.18}$$

where $\Lambda_Y = diag(\lambda_{Y,1}, \lambda_{Y,2}, ..., \lambda_{Y,N})$ is a diagonal eigenvalue matrix ($\lambda_{Y,1} \geq \lambda_{Y,2} \geq ... \geq \lambda_{Y,N}$ are the eigenvalues) and $P_Y = (p_{Y,1}, p_{Y,2}, ..., p_{Y,N})$ is the eigenvector matrix ($p_{Y,i}$, $i = 1, 2, ..., N$, represent the normalized and mutually orthogonal eigenvectors associated with their corresponding eigenvalues). Then, the matrix Y is transformed into independent variables T_Y through:

$$T_Y = YP_Y \tag{2.19}$$

where $T_Y = [T_{Y,1}, ..., T_{Y,N}]$ contains the principal components, which are orthogonal to each other.

By taking into account the ℓ_s first highest eigenvalues and their corresponding eigenvectors P_{Y,ℓ_s}, the principal components of data matrix Y, which represent the independent samples of the original data matrix X are given by,

$$T_{Y,\ell_s} = YP_{Y,\ell_s} \in \mathbb{R}^{m \times \ell_s} \tag{2.20}$$

where m is the number of variables and ℓ_s is the number of retained principal components of Y or the number of retained samples of the original data matrix X. Once the retained number of samples is determined and the matrix T_Y is computed. The reduced training data matrix X_r is defined as,

$$X_r = T_{Y,\ell_s}^T = \left[\begin{array}{cc} x'_1 & x'_2 \cdots x'_{\ell_s} \end{array} \right]^T \in \mathbb{R}^{\ell_s \times m} \tag{2.21}$$

This reduced training data is used to compute the KPCA model as presented in section 2.2.1, called reduced KPCA (RKPCA). The fault detection charts based on RKPCA method are duly presented in section 2.4.

2.4 Fault detection based RKPCA model

Using KPCA for fault detection imposes a high computational cost when the training dataset is large since the collected measurements are used for both modeling and fault detection. Thus, it is important to use the proposed RKPCA model to reduce the computational complexity and storage costs of the KPCA model. This model is determined based on an eigen-decomposition and the selection of the number of components to be retained. This late is denoted by ℓ. The literature gives many methods for selecting ℓ [26, 22]. In this study, the Cumulative Percentage of Variance (CPV) is particularly appropriate since it can be easily redefined. CPV measures the amount of variation captured by the first ℓ latent variables as:

$$CPV(\ell) = \frac{\sum_{j=1}^{\ell} \lambda_j}{\sum_{j=1}^{m} \lambda_j} 100\% \tag{2.22}$$

and ℓ is selected such that the CPV is greater than a given threshold. After that, RKPCA based fault detection is performed using the Hotelling's T^2, Squared Predictive Error (SPE) or Q and combined φ statistics [19, 27, 28]. The Hotelling's T^2 index is calculated as $T^2 = \boldsymbol{t}^T \Lambda^{-1} \boldsymbol{t}$, where $\Lambda = diag(\lambda_1, ..., \lambda_\ell)$ is the covariance of the scores $\boldsymbol{t}$ in the feature space. From Equation (2.16), the T^2 is calculated using kernel functions as [27, 20, 29],

$$T^2 = \boldsymbol{k}(\boldsymbol{x})^T P \Lambda^{-1} P^T \boldsymbol{k}(\boldsymbol{x}) \tag{2.23}$$

where $k(x') = \left[\begin{array}{cc} k(x'_1, x) & k(x'_2, x) \cdots k(x'_{\ell_s}, x) \end{array} \right]^T$.
The control limit associated with this monitoring index is given by [27, 29],

$$\tau_\alpha^{T^2} = \frac{\ell(N-1)(N+1)}{N(N-\ell)} F_\alpha(\ell, N-\ell) \tag{2.24}$$

where $F_\alpha(\ell, N-\ell)$ an F-distribution with ℓ and $N-\ell$ degrees of freedom and with a level of significance α.

The SPE index is also known as the Q statistic and it is defined in the feature space as [27, 29]:

$$\begin{cases} Q = \tilde{\boldsymbol{t}}^T \tilde{\boldsymbol{t}} \\ \quad = \phi(\boldsymbol{x})^T \tilde{P}\tilde{P}^T \phi(\boldsymbol{x}) \\ \quad = \phi(\boldsymbol{x})^T (I - PP^T)\phi(\boldsymbol{x}) \\ \quad = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}) - \phi(\boldsymbol{x})^T PP^T \phi(\boldsymbol{x}) \\ \quad = \boldsymbol{k}(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{k}^T(\boldsymbol{x}) C \boldsymbol{k}(\boldsymbol{x}) \end{cases} \tag{2.25}$$

where I is the identity matrix, $\tilde{P}\tilde{P}^T = I - PP^T$ and $C = PP^T$.
The control limit of the Q index is determined from the χ^2 approximation and defined as:

$$\tau_\alpha^Q = g\chi^2_{h,\alpha} \tag{2.26}$$

where $g = \frac{b}{2a}$ and $h = \dfrac{2a^2}{b}$, with a and b are the mean and variance of the Q index, respectively.

A combined index φ is proposed in [30, 29] and it aims to monitor the principal and residual space in the feature space simultaneously. The φ statistic is a combination of the Q and T^2 indices weighted by their thresholds and it is defined as follows.

$$\varphi = \frac{Q(x)}{\tau_\alpha^Q} + \frac{T^2}{\tau_\alpha^{T^2}} \tag{2.27}$$

The combined index has a control limit which is given by

$$\tau_\alpha^\varphi = g^\varphi \chi^2_{h^\varphi,\alpha} \tag{2.28}$$

where $g^\varphi = \frac{b^\varphi}{2a^\varphi}$ and $h = \dfrac{2(a^\varphi)^2}{b^\varphi}$, with a^φ and b^φ are the mean and variance of the φ statistic, respectively. The proposed monitoring strategy based on the RKPCA is performed in two steps. The off-line step where the reduced KPCA model is computed and the on-line step where the testing dataset is used to generate fault detection indices for process monitoring. Algorithm 1 summarizes the different steps in the proposed method.

2.5 Conclusion

In this chapter, a reduced kernel principal component analysis (RKPCA) model is developed for process monitoring. The main idea consists to reduce the number of samples used in the KPCA model. The reduced KPCA model will be implemented on the hardware Raspberry card using the software Python code.

Algorithm 1 Reduced KPCA model for process monitoring

Input: Let $X \in \mathbb{R}^{N \times m}$ the training data matrix

Training Data

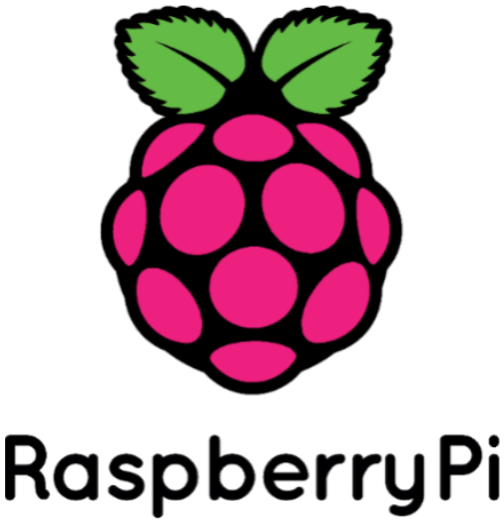
1. Perform linear PCA on the matrix $Y = X^T$, and compute principal components as: $T_Y = YP_Y$;
   - Compute the covariance matrix;
   - Determine eigenvalue decomposition determine of the covariance matrix;
   - The number of retained samples ℓ_s is determined using CPV criterion; The number ℓ_s is selected if the CPV is higher than 99 %.
   -Compute the reduced data matrix $X_r = T^T_{Y,\ell_s} \in \mathbf{R}^{\ell_s \times m}$;
   - Compute the mean and variance of the reduced training dataset; scale reduced data;
2. Apply KPCA on the reduced data matrix X_r;
   - Map the reduced data matrix to the feature space;
   - Construct the reduced kernel matrix and scale it,
   - Solve the eigenvalue decomposition problem;
   - Determine number of retained kernel principal components using the CPV criterion;
   - Compute the monitoring indices T^2, Q and φ and their corresponding control limits at a given confidence level;

Testing Data

1. For each new measurement sample at time k, $\mathbf{x}(k) \in \mathcal{R}^m$, scale it with mean and variance of the training data;
2. Compute the kernel vector $\boldsymbol{k}_k = \boldsymbol{k}(\boldsymbol{x}_k, \boldsymbol{x}_i), i = 1, ..., \ell_s, \in \mathbb{R}^{\ell_s \times 1}$ and scale it;
3. Compute the fault detection indices at time k, T^2_k, Q_k and φ_k, if any index exceeds its control limit, a fault is declared.

Chapter 3

Raspberry-Pi card



3.1 Introduction

From building a single board computer for personal purposes and entertainment to selling over 40 million boards in the world, Raspberry Pi has come a long way.

Raspberry Pi devices are developed by a UK-based organization that aims to bring digital computing power to people in all parts of the world, which enable to use low cost and high single board PCs and software

3.2 RASPBERRY PI CARD PRESENTATION:

Raspberry Pi may be a series of small single-board computers (SBCs) developed within the UK by the Raspberry Pi Foundation in association with Broadcom [31]. The Raspberry Pi project originally leaned towards the promotion of teaching basic technology in schools and in developing countries. The initial model became more popular than anticipated [32] , selling outside its target marketplace for uses like robotics. It's widely utilized in many areas, like for weather monitoring [33] thanks to its low cost, modularity, and open design. It's typically utilized by computer and electronic hobbyists, because of its adoption of HDMI and USB devices. After the discharge of the second board type, the Raspberry Pi Foundation founded a brand-new entity, named Raspberry Pi Trading, and installed Eben Upton as CEO, with the responsibility of developing technology [34].The inspiration was rededicated as an academic charity for promoting the teaching of basic technology in schools and developing countries. Most Pis are made during a Sony factory in Pencoed , Wales [34]; while others are made in China and Japan [35].

3.3 Raspberry Pi 4B characteristics and SPECS:

The Raspberry Pi 4 Model B is the latest board launched by the Raspberry Pi (foundation in June 2019), this model has the latest high-performance quad-Core 64-bit Broadcom 2711, Cortex A72 processor clocked at 1.5GHz speed, its processor uses 20 percent less power and offers 90percent greater performance than the previous model. The Raspberry Pi 4 model B comes in three different variants: 2 GB, 4 GB, and 8 GB LPDDR4 SDRAM; the other new features of the board are dual-display support up to 4k resolutions via a pair of micro-HDMI ports, hardware video decodes at up to 4Kp60, dual-channel 2.4/5.0GHz wireless LAN, true Gigabit Ethernet, two USB 3.0 ports, Bluetooth 5.0, and PoE capability (via a separate PoE HAT board). As mentioned in the figure (3.1 Raspberry Pi 4 Board Layout), the Raspberry pi 4 (Model B) board consists of:

- The Broadcom BCM2711 chip consists of Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz.
- 2GB, 4GB, and 8GB of LPDDR4 SDRAM (depending on the version of the board)
- Dual-channel 2.4/5.0 GHz, IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- Two USB 3.0 ports and two USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header
- Two micro-HDMI ports (support up to 4kp60 resolution)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port

- 265 (4k@60 decode), H264 (1080@60 decode and 1080@30 encode)
- OpenGL ES 3.0 graphics
- Micro-SD card slot for loading operating system and data storage
- 5V/3A DC via USB-C connector

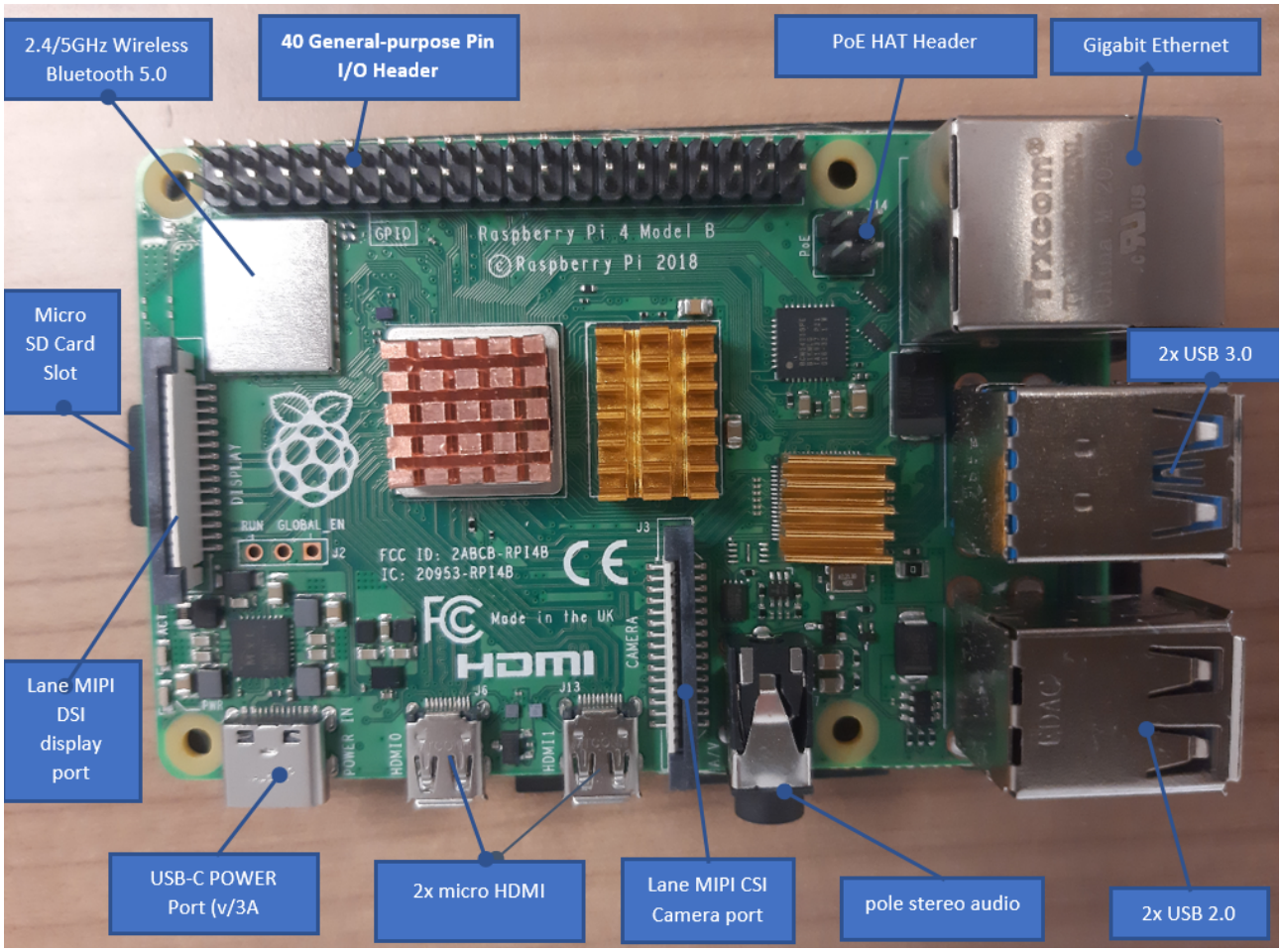


Figure 3.1: Raspberry Pi 4 Board Layout

3.4 How to Set Up a Raspberry Pi

3.4.1 Preparation

To start the setup of the raspberry card, additional to the board we will need :

- A power source
- A microSD card (at least 8GB)
- A keyboard (wired or wireless)
- A mouse or other pointing device (could be built into the keyboard)
- A monitor or TV to connect to (via HDMI)
- HDMI cables : If you just want to experiment with the Pi or use it to control physical objects like lights, motors and sensors, don't need to give it its own screen and keyboard, we can control the device from the desktop of your PC or Mac, using VNC or SSH remote access software.

3.4.2 Downloading and Installing Raspberry Pi OS

Once we prepare all the components, we proceed the following steps to set up the Raspberry Pi using a Windows, Mac or Linux-based PC (steps below are related to Windows, but it should be the same on all three).

- 1. Insert a microSD card / reader into the computer.
- 2. Download and install the official Raspberry Pi Imager. Available for Windows, macOS or Linux, this app will both download and install the latest Raspberry Pi OS. There are other ways to do this, namely by downloading a Raspberry Pi OS image file and then using a third-party app to âburn it,â but the Imager makes it easier.
- 3. Click Choose OS and select Raspberry Pi OS (32-bit) from the OS menu (there are other choices, but for most uses, 32-bit is the best). FIGURE 3.2

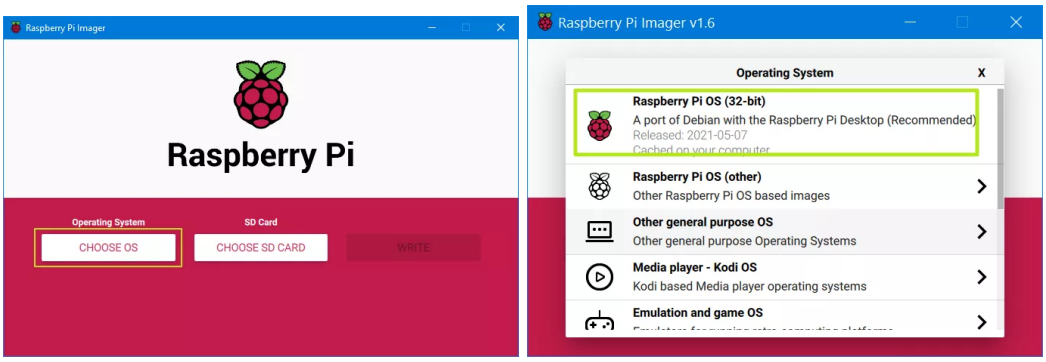


Figure 3.2: Select Raspberry Pi OS

- 4. Click Choose SD card and pick the one to use FIGURE 3.3.
- 5. Click Write. The app will now take a few minutes to download the OS and write to your card.

3.4.3 Configuring Raspberry Pi OS

On first boot, you will be given a "Welcome to the Raspberry Pi" dialog box, which takes you through the process of choosing important settings.

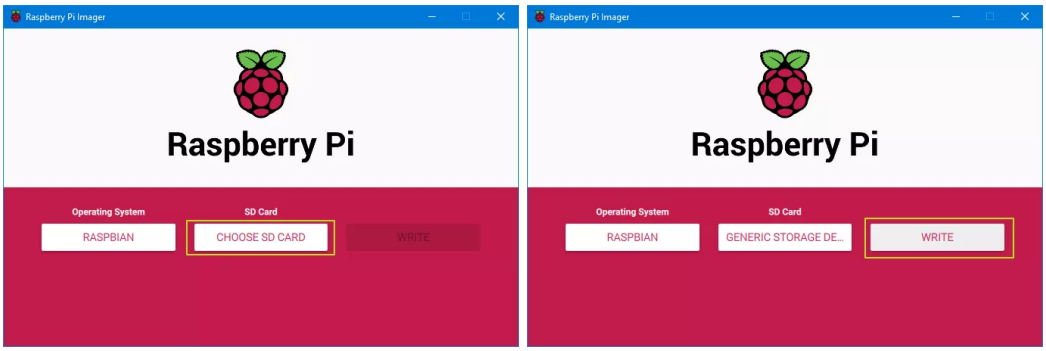


Figure 3.3: Select the affected SD card, Download and Write the OS

- 1. Click Next on the dialog box and then select your country, language and keyboard type (Figure: 3.4).

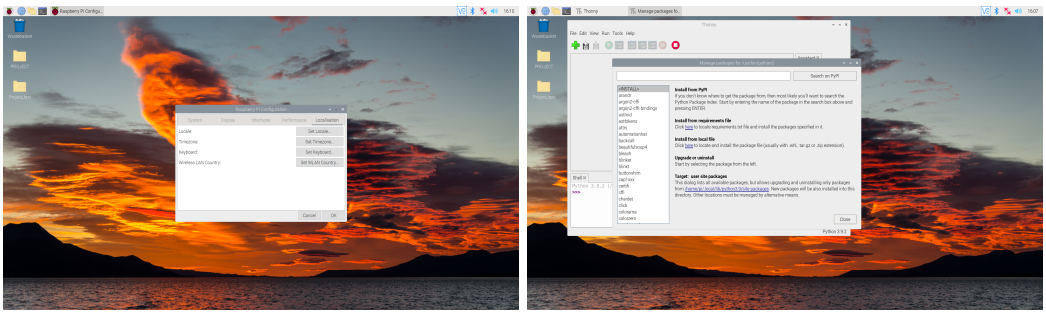


Figure 3.4: Configuring Raspberry Pi OS

- 2. Change your default password on the next screen or leave it blank for it remain as ”raspberry.”
- 3. Select the appropriate Wi-Fi network
- 4.Other option: by clicking on the Pi icon in the upper left corner of the screen and navigating to check the installed programs and change the preference.
- 5. Setting-up the Python program: the latest version of Python 3.9.2 will be installed by default with OS. Using package installation tools from Thonny Python IDE navigator bar (Figure 3.4), we can install :
  - The Jupyter notebook library (required for execution of the Python script)
  - All necessary Python codes and libraries.

3.5 Introduction to Python:

Definition:

Python is the most used programing language within the field of Machine Learning, Big Data and Data Science. Created in 1991, the Python artificial language emerged at the time as some way to automate the more boring parts of writing scripts or quickly prototype applications. In recent years, however, this programing language has become one amongst the foremost widely employed in the sector of software development, infrastructure management and data analysis. This can be a thrust behind the large Data explosion.

Main features of Python:

- Open-source: its use is free and also the source files are available and modifiable;
- Simple and extremely readable;
- Equipped with a really extensive basic library;
- Great deal of libraries available: for calculation scientific, statistics, databases, visualization...;
- High portability: independent of the operating system(Linux , windows, macOS);
- Object-oriented ;
- Dynamic typing: typing (association with a variable of its type and allocation of memory area accordingly) is finished automatically when the program is run, which allows great flexibility and speed of programming, but which is bought by overconsumption memory and performance loss;
- Presents support for the combination of other languages.

Source code operation:

There are two main techniques for translating the source code into machine language:

- Compilation: a third-party application, called a compiler, transforms the lines of code into an executable get in the language machine . After any modification to the program, you have to recompile before seeing the result.
- Interpretation: an interpreter takes care of translating line by line the machine language program. This sort of language offers more great convenience for development, but the executions are often slower. Within the case of Python, we are able to assume for begin that it's an interpreted language, which calls upon compiled mods. For expensive algorithmic operations, the Python language can interface with libraries written in low level language like C language.

Python strengths:

- Very high-level language;
- Its readability;
- Executable programming language

3.6 Conclusion

According to all characteristics represented in this chapter, it is concluded that the Raspberry Card represent a good option for the implementation of our project as well as it has a good performance in term of processing, small size, cost and flexibility of its operating system. Python is the second tools

that we presented in this chapter which it is a high-level language, easy to use, contains a large number of libraries and its simulation system is characterised by simplicity and smoothness, it does not require a large storage capacity. all Pythons properties constitute a complement with the Raspberry Pi card to be ideal tools for the implementation of our project.

Chapter 4

The implementation of Reduced KPCA technique to TE process

In this chapter, the proposed RKPCA method is applied to the Tennessee Eastman process simulation data and is compared with the classical KPCA. The performance of the RKPCA fault detection method is evaluated based on two metrics: (i) the false alarm rate (FAR) which is defined as the percentage of wrong fault declared in fault free region,and(ii) the missed detection rate (MDR) that defined as the percentage of faulty observations not detected

4.1 TE process description

Tennessee Eastman (TE) is an interesting and challenging problem in industrial process control. The TE process was first proposed by Downs dan Vogel in 1993 [[36]]. The problems include multivariable process control, nonlinear control, diagnostic and monitoring, education, and others. The TE process represents a process simulator which mimics the real process. The process consists of six principals units, such as an exothermic reactor, a product condenser, a vapour liquid separator, a recycle compressor and a product stripper, the process flow sheet is given in Figure 4.1. It consists of 41 measured variables (Table 4.1)and 11 manipulated variables Table 4.2). The 41 measured variables contain 22 continuous process variables and 19 composition variables.

The TE process has 22 continuous process measurements, 12 manipulated variables, and 19 composition measurements sampled less frequently. [36, 37, 38]. A total of 52 variables are used for fault detection in this study. A set of 21 faults are introduced to the process. All these 21 faults is given in Table 4.3.

4.2 Modeling Fault detection of TE process

Amount of 22 sets of data has been generated from the TE process simulator, one set for the normal operation condition which named Training Data and 21 sets for Testing Data, each sheet contains the data of 1024 samples. The fault in each testing data set is introduced from sample 224. The experiment is conducted in Jupyter Note book (Python 3.9) environment on Raspberry PI 4 model B card .

Continuous process variables	Description	Composition variables	Description
$Xmeas_1$	A feed(stream 1)	$Xmeas_{23}$	Composition A (stream 6)
$Xmeas_2$	D feed(stream 1)	$Xmeas_{24}$	Composition B (stream 6)
$Xmeas_3$	E feed(stream 1)	$Xmeas_{25}$	Composition C (stream 6)
$Xmeas_4$	Total feed(stream 4)	$Xmeas_{26}$	Composition D (stream 6)
$Xmeas_5$	Recycle flow(stream 8)	$Xmeas_{27}$	Composition E(stream 6)
$Xmeas_6$	Reactor feed rate(stream 6)	$Xmeas_{28}$	Composition F (stream 6)
$Xmeas_7$	Reactor pressure	$Xmeas_{29}$	Composition A(stream 9)
$Xmeas_8$	Reactor level	$Xmeas_{30}$	Composition B(stream 9)
$Xmeas_9$	Reactor temperature	$Xmeas_{31}$	Composition C(stream 9)
$Xmeas_{10}$	Purge rate (stream 9)	$Xmeas_{32}$	Composition D (stream 9)
$Xmeas_{11}$	Product separator temperature	$Xmeas_{33}$	Composition E(stream 9)
$Xmeas_{12}$	Product separator level	$Xmeas_{34}$	Composition F (stream 9)
$Xmeas_{13}$	Product separator pressure	$Xmeas_{35}$	Composition G (stream 9)
$Xmeas_{14}$	Product separator underflow (stream 10)	$Xmeas_{36}$	Composition H (stream 9)
$Xmeas_{15}$	Stripper level	$Xmeas_{37}$	Composition D(stream 11)
$Xmeas_{16}$	Stripper pressure	$Xmeas_{38}$	Composition E (stream 11)
$Xmeas_{17}$	Stripper underflow(stream 11)	$Xmeas_{39}$	Composition F(stream 11)
$Xmeas_{18}$	Stripper temperature	$Xmeas_{40}$	Composition G(stream 11)
$Xmeas_{19}$	Stripper stream flow	$Xmeas_{41}$	Composition H(stream 11)
$Xmeas_{20}$	Compressor work		
$Xmeas_{21}$	Reactor cooling water outlet temp		
$Xmeas_{22}$	Separator cooling water outlet temp		

Table 4.1: Mesured process variables in the TE process.

Variables	Description
$XMV1_{42}$	D feed flow (stream 2)
$XMV2_{43}$	A Feed flow (stream 3)
$XMV3_{44}$	E Feed flow (stream 1)
$XMV4_{45}$	Total feed flow (stream 4)
$XMV5_{46}$	Compressor Recycle Valve
$XMV6_{47}$	Purge Valve (stream 9)
$XMV7_{48}$	Separator Pot Liquid flow (stream 10)
$XMV8_{49}$	Stripper Liquid Product flow (stream 11)
$XMV9_{50}$	Stripper Stream Valve
$XMV10_{51}$	Reactor Cooling Water flow
$XMV11_{52}$	Condenser Cooling Water flow

Table 4.2: Manipulated variables in the TE process.

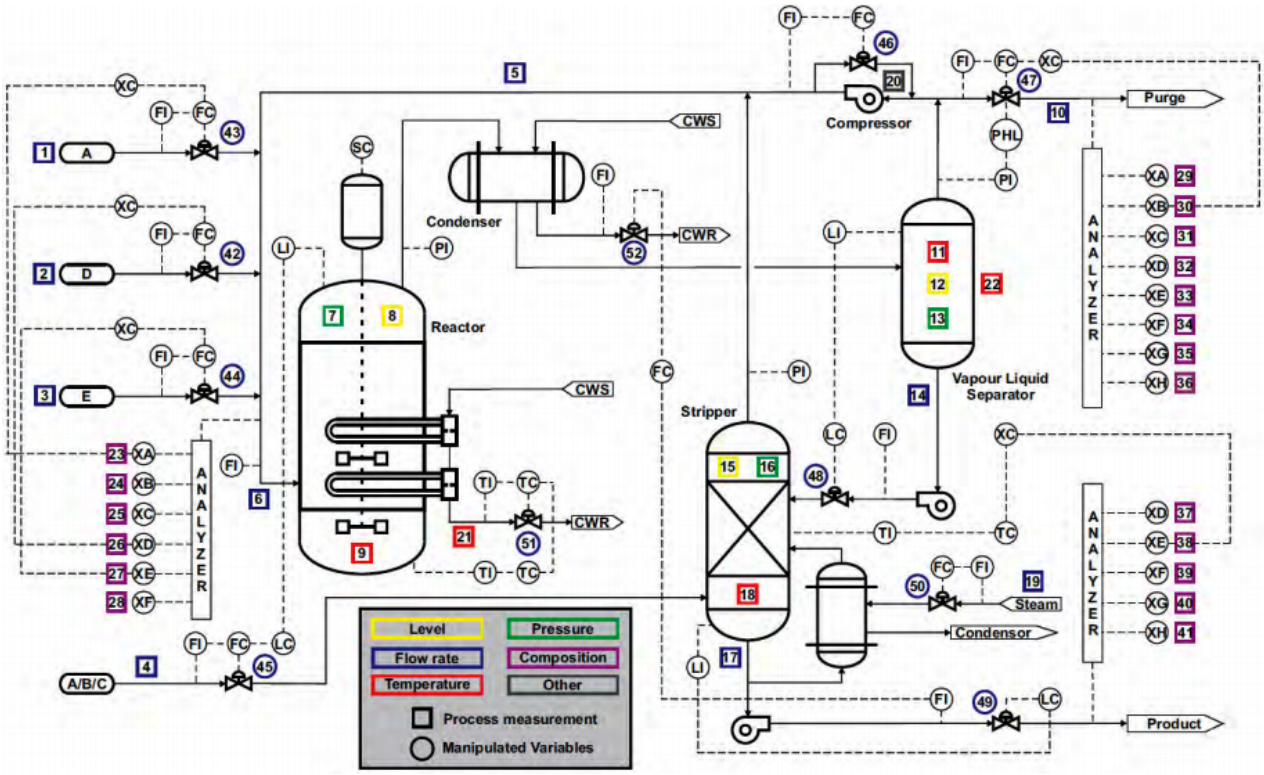


Figure 4.1: Tennessee Eastman process.

Table 4.3: Summary of process faults of TE Process.

Faults	Process variable	Type
IDV_1	A/C feed ratio, B composition constant (stream 4)	Step
IDV_2	B composition, A/C feed ratio constant (stream 4)	Step
IDV_3	D feed temperature (stream 2)	Step
IDV_4	Reactor cooling water inlet temperature	Step
IDV_5	Condenser cooling water inlet temperature	Step
IDV_6	A feed loss (stream 1)	Step
IDV_7	C header pressure loss-reduced availability (stream 4)	Step
IDV_8	A, B and C feed compositions (stream 4)	Random variation
IDV_9	D feed temperature (stream 2)	Random variation
IDV_{10}	C feed temperature (stream 4)	Random variation
IDV_{11}	Reactor cooling water inlet temperature	Random variation
IDV_{12}	Condenser cooling water inlet temperature	Random variation
IDV_{13}	Reaction kinetics	Slow shift
IDV_{14}	Reactor cooling water valve	Sticking
IDV_{15}	Condenser cooling water valve	Sticking
IDV_{16}	Unknown	Unknown
IDV_{17}	Unknown	Unknown
IDV_{18}	Unknown	Unknown
IDV_{19}	Unknown	Unknown
IDV_{20}	Unknown	Unknown
IDV_{21}	Valve position constant (stream 4)	Constant position

4.2.1 OFF-Line Simulation

The performance evaluation of the KPCA and RKPCA methods is Based on false alarms rate (FAR) and the missed detection rate (MDR). where FAR is the percentage rate of wrong alarming in healthy set, and MDR represents the percentage rate of the non-detected faulty samples in a faulty region.

The following Steps represent the OFF-Line simulation:

1. Import all necessary libraries (Figure4.2);
2. Introduce the Training Data in order to take the mean and standard deviation of the data for centering (Figure4.2);
3. Calculate Covariance Matrix and get the Eigenvalues and Eigenvector;
4. Sort the Eigenvalues/Eigenvector and extract the number of the PC retained by using CPV function;
5. Apply KPCA on the reduced data matrix Xr and map it to the feature space ;
6. Construct the reduced kernel matrix and scale it, and extract the number of retained kernel principal components;
7. Compute the monitoring indices T^20, $Q0$ and $\varphi 0$ and their corresponding thresholds at a confidence level of 95% ;
8. Introduce the Testing Data by choosing the fault to be implemented from TE Process Data-Base;
9. Normalize the Testing Data and compute the monitoring indices T^2, Q and φ ;
10. Compute FAR and MDR by comparing the resulted monitoring indices with the thresholds calculated for Training Data;
11. Plotting the indices of the Testing Data (Figure4.3).

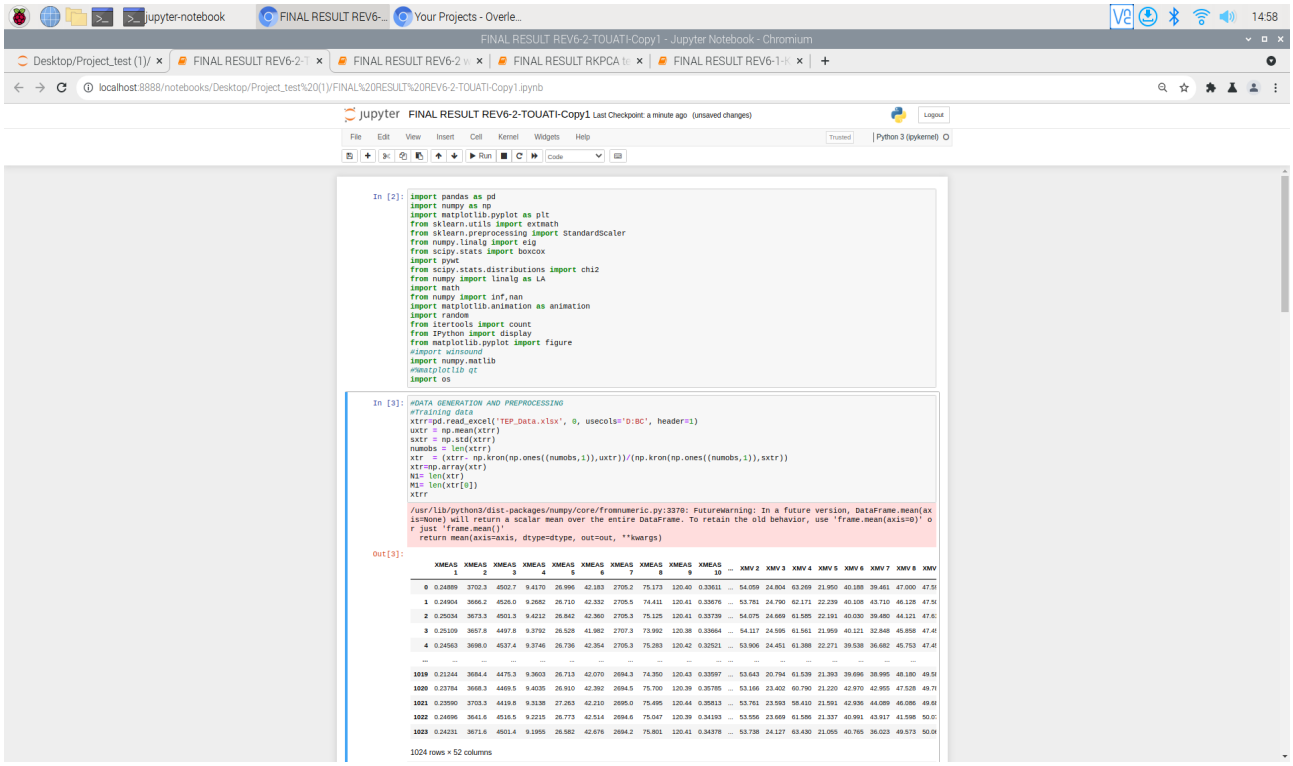


Figure 4.2: OFF-LINE PROGRAM SIMULATION

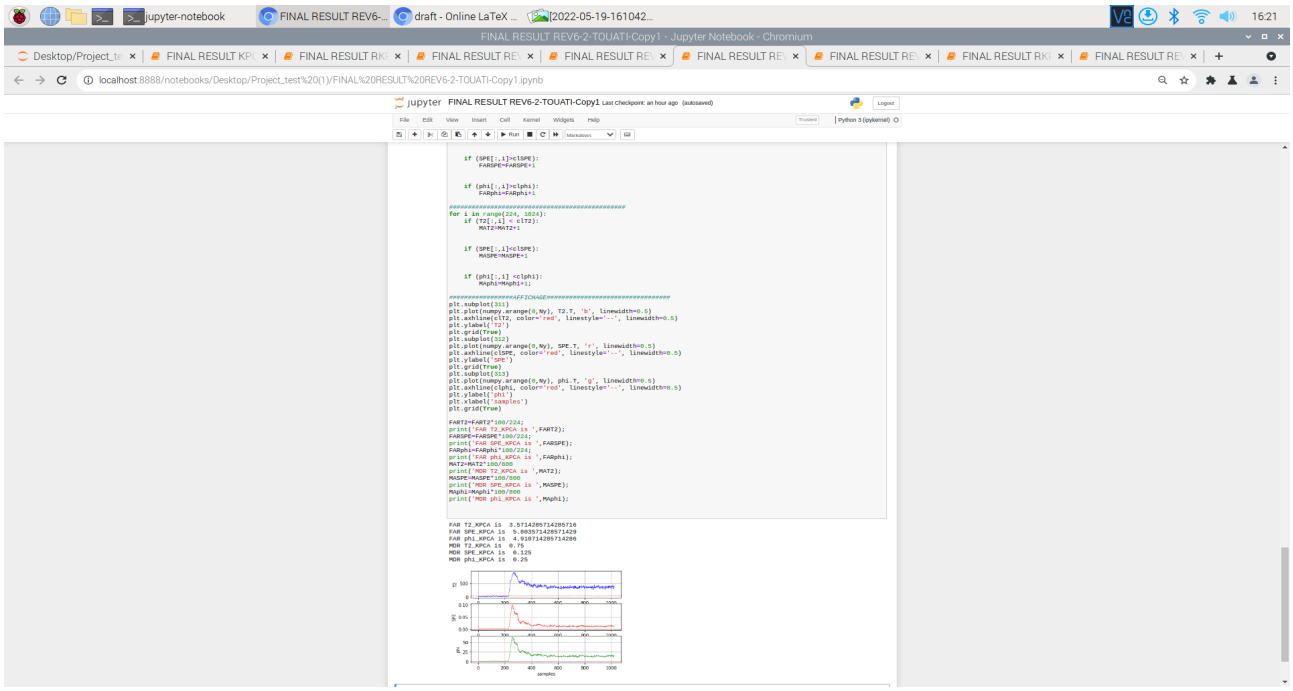


Figure 4.3: OFF-LINE PROGRAM SIMULATION

Figures 4.4 to 4.7 show respectively T^2, Q and φ charts using KPCA and Reduced-KPCA for fault detection purposes in the case of fault IDV_1 and IDV_7 of TE process. The thresholds for the used statistics, shown in red dotted line, are developed at a 95% confidence level. A fault is detected if one of the three indices exceeds its corresponding threshold. It can be seen that the fault is clearly detected, in main time we maintain the same performance after using the Reduced-KPCA method comparing with KPCA method.

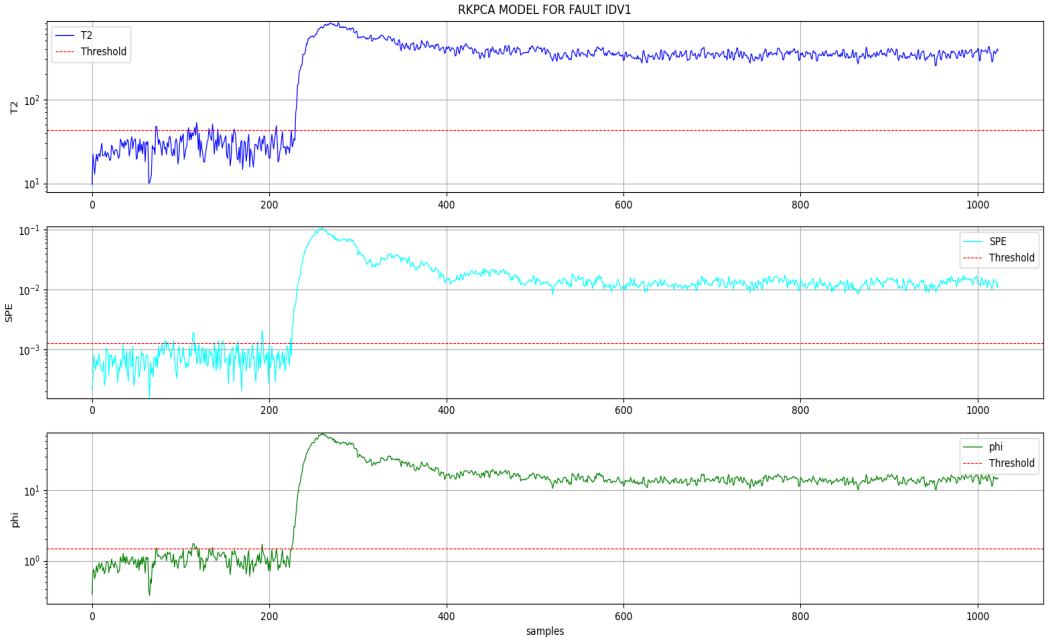


Figure 4.5: Time evolution of the T^2, Q and φ statistic based RKPCA model for fault IDV_1

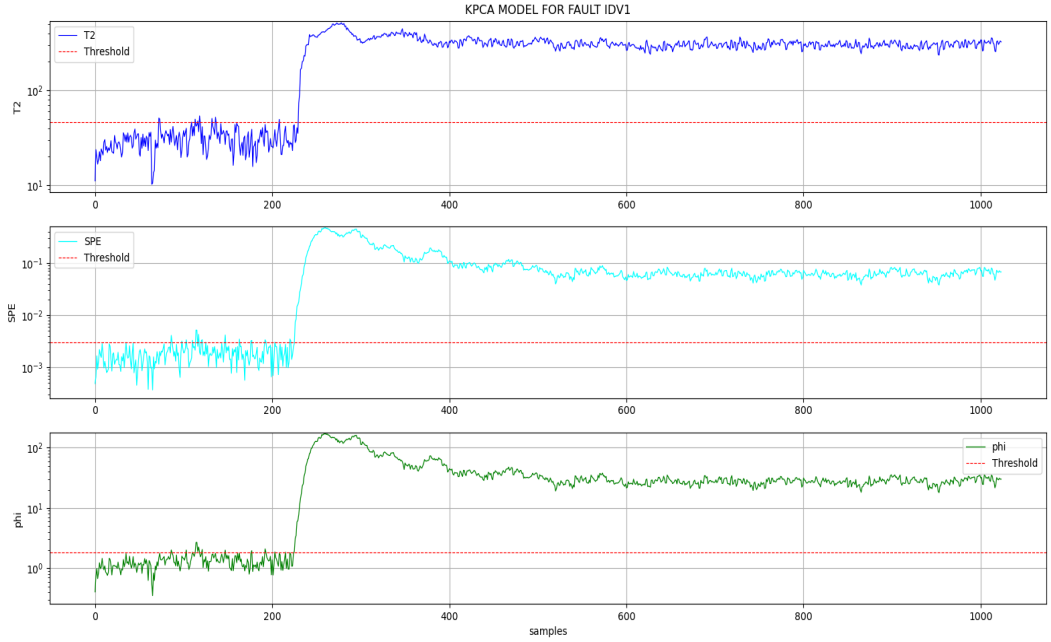


Figure 4.4: Time evolution of the T^2, Q and φ statistic based KPCA model for fault IDV_1

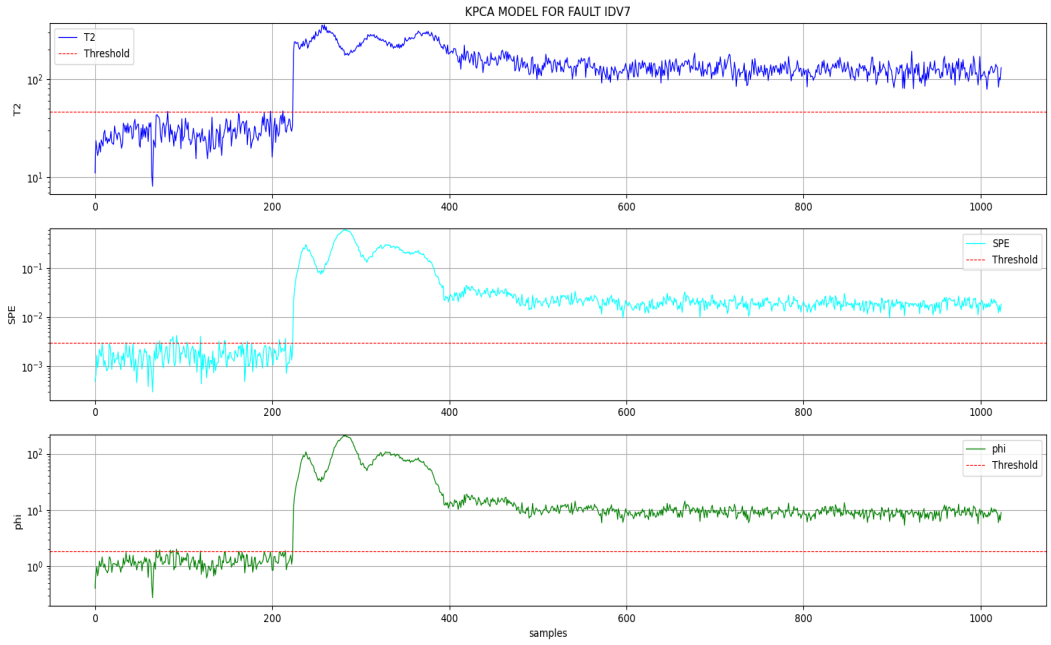


Figure 4.6: Time evolution of the T^2, Q and φ statistic based KPCA model for fault IDV_7

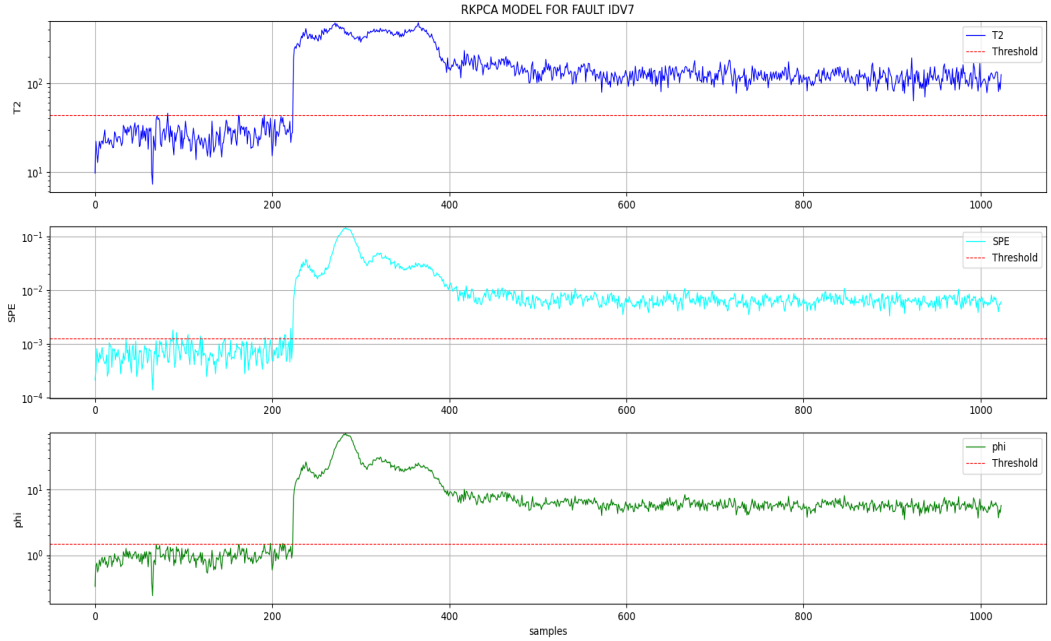


Figure 4.7: Time evolution of the T^2, Q and φ statistic based RKPCA model for fault IDV_7

Table 4.4 summarizes the FAR/MDR percentage values contributed by kernel and the reduced kernel PCA methods in the associated T^2, Q and φ statistics. The bold values highlight the performance of the RKPCA method. However, by making rapprochement with the conventional KPCA method, the RKPCA technique presents a similar performance which makes the overall proposed algorithm reliable for operation, on the other hand, the MDR is still acceptable and ensuring the successful detection of most faulty samples of different types and amplitudes of process faults.

	KPCA			Reduced KPCA		
Faults	T^2	Q	φ	T^2	Q	φ
1	03.57/00.75	09.37/00.25	06.69/00.12	04.46/00.75	08.03/00.12	04.91/00.25
2	03.57/01.25	06.25/01.00	04.91/01.12	04.01/01.25	05.80/01.5	03.57/01.37
3	03.57/92.75	09.82/81.87	08.48/82.12	01.78/93.12	08.92/83.75	05.80/88.12
4	02.23/38.62	06.25/00.00	03.57/00.00	03.57/37.00	03.57/00.00	02.23/00.00
5	02.23/67.75	06.25/47.12	03.57/46.75	03.57/69.00	03.57/54.12	02.23/58.50
6	00.89/86.75	08.92/00.00	04.01/00.00	01.33/01.00	04.46/00.00	00.89/00.00
7	01.33/00.00	07.14/00.00	03.57/00.00	01.33/00.00	04.91/00.00	00.89/00.00
8	02.67/02.50	08.03/01.75	04.01/01.62	02.67/02.62	12.05/01.50	03.57/02.00
9	06.69/91.62	11.60/84.37	13.39/85.62	04.91/92.12	10.71/88.62	10.71/90.12
10	01.78/56.25	06.25/27.00	03.57/27.25	02.67/60.62	03.57/31.75	02.23/34.62
11	03.12/38.87	08.92/18.00	08.03/14.50	02.23/39.62	05.35/22.62	04.46/18.12
12	05.35/01.00	12.05/00.62	09.37/00.62	04.46/01.25	10.26/01.37	07.14/00.87
13	02.67/04.87	02.67/04.00	03.12/04.12	02.67/05.00	04.46/04.50	02.67/04.37
14	03.57/00.00	08.92/00.00	08.03/00.00	03.12/00.00	10.71/00.12	05.35/00.00
15	01.33/90.75	05.35/83.00	04.46/78.75	01.33/91.50	04.01/84.00	01.78/83.75
16	10.26/72.37	11.60/33.50	14.73/33.12	10.26/74.62	08.03/38.75	12.05/43.37
17	02.67/20.25	08.48/02.50	06.69/03.25	02.67/16.62	07.14/03.25	03.57/04.12
18	02.67/81.12	10.71/08.37	10.26/08.00	01.78/09.62	07.14/08.62	04.46/09.12
19	01.78/76.50	07.58/51.12	06.69/46.25	02.67/76.37	05.80/62.00	02.67/61.50
20	02.67/56.25	03.12/28.75	03.57/26.87	03.12/57.50	04.01/34.00	02.23/33.75
21	06.69/51.50	17.85/38.62	16.96/38.87	05.80/56.37	11.16/36.75	07.14/43.00
Average	03.39/44.36	08.43/24.37	06.22/23.75	03.35/37.42	06.84/26.54	04.31/27.47

Table 4.4: T^2, Q and φ contributions in FAR and MDR through conventional and reduced KPCA for different faults of the TE process

4.2.2 ON-Line Simulation

1. Normalize test data using the mean and standard deviation obtained from the reduced data-set in step 7 from the OFF-Line simulation of Training Data;

2. Build The reduced test kernel matrix;

3. Compute T^2, Q and φ ;

4. Compare T^2, Q and φ with their corresponding thresholds;

5. Plot the result and make a decision (Fault or no Fault);

4.3 Conclusions

A general overview about diagnosis and fault detection is presented in the first section as a prelude to the one of based-statistical diagnosis approach for nonlinear multivariate processes on which our subject is based and it is the kernel principal component analysis method. Reduced KPCA technique is a reduced model based on using PCA to the training dataset in such a way to avoid redundant observations and after that apply KPCA to the reduced dataset, by using the combination of two tools for machine learning: the hardware Raspberry card and the software Python code we implement and simulate our proposed technique RKPCA for the Tennessee Eastman process, the simulation of RKPCA model comparing with the simulation of conventional KPCA model by analysing the FAR and MDR of the both model on different simulated faults of TE process demonstrates a reliability, detection sensitivity and stable monitoring performance of RKPCA scheme. The results we get by using RKPCA scheme, Python code and Raspberry Pi card show good monitoring process and make our project (Fault diagnosis by RKPCA using Python on Raspberry Pi card) more practical for real-process applications.

Bibliography

[1] I.E. Frank. A nonlinear PLS model. *Chemolab*, 8:109?119, 1990.

[2] R. Isermann. Model-based fault-detection and diagnosis : status and applications. *Annual Reviews in Control*, 29:71–85, 2005.

[3] V. Venkatasubramanian, R. Rengaswamy, S. Kavuri, and K. Yin. A review of process fault detection and diagnosis Part II: Qualitative models and search strategies. *Computers and Chemical Engineering*, 27:313–326, 2003.

[4] H Henry Yue and S Joe Qin. Reconstruction-based fault identification using a combined index. *Industrial & engineering chemistry research*, 40(20):4403–4414, 2001.

[5] M Ziyan Sheriff, Majdi Mansouri, M Nazmul Karim, Hazem Nounou, and Mohamed Nounou. Fault detection using multiscale pca-based moving window glrt. *Journal of Process Control, 2017, 54, 47.*

[6] Majdi Mansouri, Mohamed Nounou, Hazem Nounou, and Nazmul Karim. Kernel pca-based glrt for nonlinear fault detection of chemical processes. *Journal of Loss Prevention in the Process Industries*, 40:334–347, 2016.

[7] Xun Wang, Uwe Kruger, and George W Irwin. Process fault diagnosis using recursive multivariate statistical process control. In *Proceeding of 16th IFAC World Congress. Prague, Czech Republic: the IFAC*, volume 16, 2005.

[8] M-F Harkat, A Kouadri, R Fezai, M Mansouri, H Nounou, and M Nounou. Machine learning-based reduced kernel pca model for nonlinear chemical process monitoring. *Journal of Control, Automation and Electrical Systems*, 31(5):1196–1209, 2020.

[9] Rosario Toscano. *Commande et diagnostic des systèmes dynamiques: modélisation, analyse, commande par PID et par retour d'état, diagnostic.* ellipses, 2005.

[10] David Mautner Himmelblau. *Fault detection and diagnosis in chemical and petrochemical processes*, volume 8. Elsevier Science Limited, 1978.

[11] V. Venkatasubramanian, R. Rengaswamy, S. Kavuri, and K. Yin. A review of process fault detection and diagnosis part I : Quantitative model-based methods. *Computers and Chemical Engineering*, 27:293–311, 2003.

[12] MJ Grimble. *Fault Diagnosis in Dynamic Systems: Theory and Applications.* Prentice Hall, 1989.

[13] Paul M Frank. Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy: A survey and some new results. *automatica*, 26(3):459–474, 1990.

[14] Rolf Isermann and Peter Balle. Trends in the application of model-based fault detection and diagnosis of technical processes. *Control engineering practice*, 5(5):709–719, 1997.

[15] Srinivas Katipamula and Michael R Brambley. Methods for fault detection, diagnostics, and prognostics for building systemsâa review, part i. *Hvac&R Research*, 11(1):3–25, 2005.

[16] Jie Chen and Ron J Patton. *Robust model-based fault diagnosis for dynamic systems*, volume 3. Springer Science & Business Media, 2012.

[17] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. pages 583–588, 1997.

[18] Ruixiang Sun, Fugee Tsung, and Liangsheng Qu. Evolving kernel principal component analysis for fault diagnosis. *Computers & Industrial Engineering*, 53(2):361–371, 2007.

[19] Jong-Min Lee, ChangKyoo Yoo, Sang Wook Choi, Peter A Vanrolleghem, and In-Beum Lee. Nonlinear process monitoring using kernel principal component analysis. *Chemical engineering science*, 59(1):223–234, 2004.

[20] Peiling Cui, Junhong Li, and Guizeng Wang. Improved kernel principal component analysis for fault detection. *Expert Systems with Applications*, 34(2):1210–1219, 2008.

[21] J Edward Jackson and Govind S Mudholkar. Control procedures for residuals associated with principal component analysis. *Technometrics*, 21(3):341–349, 1979.

[22] Ian T Jolliffe. A note on the use of principal components in regression. *Applied Statistics*, pages 300–303, 1982.

[23] S Joe Qin. Statistical process monitoring: basics and beyond. *Journal of chemometrics*, 17(8-9):480–502, 2003.

[24] Yogesh Rathi, Samuel Dambreville, and Allen Tannenbaum. Statistical shape analysis using kernel pca. In *Image processing: algorithms and systems, neural networks, and machine learning*, volume 6064, page 60641B. International Society for Optics and Photonics, 2006.

[25] Okba Taouali, Ines Jaffel, Hajer Lahdhiri, Mohamed Faouzi Harkat, and Hassani Messaoud. New fault detection method based on reduced kernel principal component analysis (rkpca). *The International Journal of Advanced Manufacturing Technology*, 85(5):1547–1552, 2016.

[26] Sergio Valle, Weihua Li, and S Joe Qin. Selection of the number of principal components: the variance of the reconstruction error criterion with a comparison to other methods. *Industrial & Engineering Chemistry Research*, 38(11):4389–4401, 1999.

[27] Sang Wook Choi, Changkyu Lee, Jong-Min Lee, Jin Hyun Park, and In-Beum Lee. Fault detection and identification of nonlinear processes based on kernel pca. *Chemometrics and intelligent laboratory systems*, 75(1):55–67, 2005.

[28] F Bencheikh, MF Harkat, A Kouadri, and A Bensmail. New reduced kernel pca for fault detection and diagnosis in cement rotary kiln. *Chemometrics and Intelligent Laboratory Systems*, 204:104091, 2020.

[29] Carlos F Alcala and S Joe Qin. Reconstruction-based contribution for process monitoring with kernel principal component analysis. *Industrial & Engineering Chemistry Research*, 49(17):7849–7857, 2010.

[30] H Henry Yue and S Joe Qin. Reconstruction-based fault identification using a combined index. *Industrial & engineering chemistry research*, 40(20):4403–4414, 2001.

[31] Raspberry Pi. Raspberry pi foundationâabout usâ, 2020.

[32] Tuncay KARA and Ahmet YÖNETKEN. Robot arm design and control with raspbery pi.

[33] Liz Upton. The raspberry pi in scientific research. *The Raspberry Pi in scientific research*, 2013.

[34] Hoz Torre, José Ramón, et al. Diseño e implementación de un sistema domótico de vigilancia controlado por dispositivos embebidos. 2021.

[35] L Tung. Raspberry pi: 14 million sold, 10 million made in the uk. *ZDnet Information available at: https://www. zdnet. com/article/14-million-raspberry-pis-sold-10-million-made-in-the-uk*, 2017.

[36] James J Downs and Ernest F Vogel. A plant-wide industrial process control problem. *Computers & chemical engineering*, 17(3):245–255, 1993.

[37] Leo H Chiang, Evan L Russell, and Richard D Braatz. *Fault detection and diagnosis in industrial systems*. Springer Science & Business Media, 2000.

[38] Evan L Russell, Leo H Chiang, and Richard D Braatz. *Data-driven methods for fault detection and diagnosis in chemical processes*. Springer Science & Business Media, 2012.

Annexes

SIMULATION PROGRAM

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import extmath
from sklearn.preprocessing import StandardScaler
from numpy.linalg import eig
from scipy.stats import boxcox
import pywt
from scipy.stats.distributions import chi2
from numpy import linalg as LA
import math
from numpy import inf,nan
import matplotlib.animation as animation
import random
from itertools import count
from IPython import display
from matplotlib.pyplot import figure
import winsound
import numpy.matlib
%matplotlib qt
```

[ ]:
```
#DATA GENERATION AND PREPROCESSING
#Training data
xtrr=pd.read_excel('TEP_Data.xlsx', 0, usecols='D:BC', header=1)
uxtr = np.mean(xtrr)
sxtr = np.std(xtrr)
numobs = len(xtrr)
xtr  = (xtrr- np.kron(np.ones((numobs,1)),uxtr))/(np.kron(np.
 ↪ones((numobs,1)),sxtr))
xtr=np.array(xtr)
N1= len(xtr)
M1= len(xtr[0])
print(N1, M1)
```

[ ]:
```
#Covariance Matrix
X1=xtr
```

```
N, m= X1.shape
Cov=np.dot(X1, X1.T)
```

[ ]:
```
#Calculate eigen values and vectors
Cov=np.array(Cov)
d0, vp0=eig(Cov)
#Sort Eigen vector and Eigen values
idx= d0.argsort()[::-1]
d0= d0[idx]
vp0= vp0[:,idx]
```

[ ]:
```
#Extract the number of PC retained (Function CPV)
alpha = [0]*1024

for iin range(1, len(d0)):
    alpha[i]=np.sum(d0[0:i])/np.sum(d0)
    if((alpha[i]>=0.99).all()):
        L=i break
L
```

[ ]:
```
#Reduced Kenrnel Matrix
newdata = np.dot(X1.T,vp0[:,0:L])
X1=newdata.T
xxtrain=X1
Nx = len(xxtrain)
Mx = len(xxtrain[0])
mean_X=xxtrain.mean(0)
std_X=np.std(xxtrain)
column=np.ones((Nx,Mx))
xxtrain=xxtrain-mean_X*column
xxtrain=xxtrain.real
xxtrain =  xxtrain/std_X*column

#distance Matrix(X)
N=len(xxtrain)
XX=xxtrain*xxtrain
XX=XX.sum(axis=1)
D=np.kron(np.ones((N,1)),XX)+np.kron(np.ones((N,1)),XX.T)-2*(np.
 ↪dot(xxtrain,xxtrain.T))
D[D<0] = 0
DIST=np.sqrt(D)
DIST[DIST<=1] = 0
DIST[DIST==0]=inf
```

[ ]:
```
#%%%%%%%%%%%%%%% Computing parameter c %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DIaST=DIST.min(axis=0)
```

```
c1=DIaST.mean(axis=0)
c=20*c1
c=c*c/2
#%%%%%%%%%%%%%%% Kernel Matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N=len(xxtrain)
K=np.zeros((Nx,Nx), dtype=float)
for i in range(0, Nx):
    for jin range(0, Nx): kint=LA.norm(xxtrain[i,:]-
        xxtrain[j,:])**2/c K[i,j]=math.exp(-kint)
K[K == -inf] = 0 K[K
    == nan] = 0 K[K
    == inf] = 0
```

[ ]:
```
#%%%%%%%%%%%%%%%% Compute Gram matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Summationkx=K.sum(axis=0)
Summationkx1=Summationkx.transpose()
Summationkx2=Summationkx1.sum()
n1=np.ones((Nx,Nx))
N2=n1/N
N1=np.array(N2)
Kp=K-N2@K-K@N2+N2@K@N2
```

[ ]:
```
# Extract the retained PC (npc)
vectors1, values1 =eig(Kp/41)
idx= vectors1.argsort()[::-1]
vectors1= vectors1[idx]
values2= values1[:,idx]
ab=sorted(vectors1, reverse= True)
lamda=vectors1
npc=1
while ((lamda[:npc].sum())/(lamda.sum()) ) < 0.90 : npc=npc+1
```

[ ]:
```
s=np.diag(ab)
P2=values2
P=P2
lamda=s
for i in range(0, npc):
    P[:,i]=P2[:,i]/ (LA.norm(P[:,i])* math.sqrt(N*s[i,i])) # same as matlab

########### t matrix ############################################
t1=np.zeros((Nx,npc), dtype=float)
tt1=np.zeros((Nx,npc), dtype=float)
t=np.zeros((1,npc), dtype=float)

for i in range(0, Nx):
    for j in range (0, npc):
```

```
            t[:,j]=np.dot(P[:,j].T,Kp[:,i])
            t1[i,:]=t
```

[ ]:
```
#####################Calculate error indicators T² , SPE and Phi for the␣
 ↪training data #########
T21=np.zeros((Nx,npc), dtype=float)
T20=np.zeros((1,Nx), dtype=float)
SPE0=np.zeros((1,Nx), dtype=float)
for i in range(0, Nx):
            yy=np.linalg.inv(lamda[0:npc,0:npc])
            T21[i,:]=np.dot(t1[i,:], yy)
            T20[:,i]=np.dot(T21[i,:], t1[i,:].T)
            SPE0[:,i]=1- np.dot(t1[i,:], np.transpose(t1[i,:]))-(2/
 ↪N)*(Summationkx1[i])+(1/(N*N))*Summationkx2
clT2=np.percentile(T20,95)
S0=(np.std(SPE0))**2
miu=np.mean(SPE0)
P1=np.round(2*miu**2/S0)
clSPE=np.percentile(SPE0,95)
phi0=T20/clT2 + SPE0/clSPE
pp=((npc/clT2)+(miu/clSPE))/((npc/clT2**2)+(S0/clSPE**2))
clphi=np.percentile(phi0,95)
```

[ ]:
```
###################Introduce the testing data###################################
xtstr1=pd.read_excel('TEP_Data.xlsx', 2, usecols='D:BC', header=1)
x_data_plot=[]
T2_plot=[]
SPE_plot=[]
phi_plot=[]
plt.subplot(311)
plt.subplot(312)
plt.subplot(313)
plt.gcf()
#################introduce data row by row as online␣
 ↪simulation#######################
for r in range (0, 1024):
    xtstr=xtstr1.iloc[[r]]
    numobstst =len(xtstr)
    xxtest = (xtstr- np.kron(np.ones((numobstst,1)),uxtr))/(np.kron(np.
 ↪ones((numobstst,1)),sxtr))
    xxtest=np.array(xxtest)
    Ny=len(xxtest)

    Ky=np.zeros((Ny,N), dtype=float)

    for i in range(0, Ny):
```

```
        for jin range(0, N): kintyy=(LA.norm(xxtest[i,:]-xxtrain[j,:]))**2/c
            Ky[i,j]=math.exp(-kintyy)

    K[K == -inf] = 0
    K[K == nan] = 0
    K[K == inf] = 0 SummationKy=Ky.T.sum(axis=0)
    Summationky=SummationKy.transpose()

    etest=np.ones((Ny,N))
    etest=(1/N)*etest
    kp00=Ky-(np.dot(etest, K))-(np.dot(Ky, N1))+(np.dot(np.dot(etest,K),N1))
    kp1=kp00.T
    tnew=np.empty((1,npc))
    tnew1=np.empty((Ny,npc))
    for i in range(0, Ny):
        for j in range (0, npc): tnew[:,j]=np.dot(P[:,j].T,kp1[:,i])
        tnew1[i,:]=tnew

    T2=np.empty((1,Ny)) T22=np.empty((Ny,npc)) SPE=np.empty((1,Ny))
#############Error indicators for test data T² et  SPE ################## for i
    in range(0, Ny):
            T2[:,i]=np.dot(tnew1[i,:], np.dot(np.linalg.inv(lamda[0:npc,0:npc]),␣
 ↪tnew1[i,:].T))
            SPE[:,i]=1-np.dot(tnew1[i,:],tnew1[i,:].T)-2/N*Summationky[i]+(1/
 ↪(N*N))*Summationkx2
    phi=T2/clT2+SPE/clSPE
#############Plotting#################
    x_data_plot.append(r)
    T2_plot.append(T2[:,i].T)
    SPE_plot.append(SPE[:,i].T)
    phi_plot.append(phi[:,i].T)
    plt.subplot(311)
    plt.plot(x_data_plot, T2_plot, linewidth=0.5) plt.axhline(clT2, color='red',
    linestyle='--', linewidth=0.2) plt.ylabel('T2')
    plt.grid(True)
    plt.subplot(312)
    plt.plot(x_data_plot, SPE_plot, 'b', linewidth=0.5) plt.axhline(clSPE,
    color='red', linestyle='--', linewidth=0.2) plt.ylabel('SPE')
    plt.grid(True)
```

```
    plt.subplot(313)
    plt.plot(x_data_plot, phi_plot, 'g', linewidth=0.5)
    plt.axhline(clphi, color='red', linestyle='--', linewidth=0.2)
    plt.ylabel('phi')
    plt.xlabel('samples')
    plt.grid(True)
    plt.show()
```

Authored by: Kaddour TOUATI
kaddour.touati1@gmail.com