

# SPCUP 2020: IEEE Signal Processing Cup 2020

Daniele Scapin, Gabriele Ferraresso, Marco Perin, Piero Simonetto, Riccardo Lorigiola

**Abstract**—In this letter, we proposed our method to solve the problem of unsupervised abnormality detection. It involves four main steps: peaks and abnormalities during data acquisition search, anomaly check and system values update.

First and second steps take place at the same time during program flow but they search autonomously the anomaly. Peaks search is implemented with the use of polyfit and Kalman evaluation, for prediction of the measures and calculation of the percentual change. Instead abnormalities during data acquisition uses a modified version of the isolation forest to verify if during the measures are occurred discrepancies.

The third part leverages of the concept that if every sensor in the autonomous system reports an anomaly, it means that the behavior is not anomaly. The anomaly check uses a tree structure to get feedback from each sensor.

The system values update is used to improve response time and to update the variables needed in other sections.

**Index Terms**—Abnormalities detection, peaks search, Kalman filter, isolation forest, tree structure

## I. INTRODUCTION

### TO DO

(Descrizione dettagliata flow del programma, associata al flow diagram)

## II. PEAKS SEARCH: FINDPEAKSWRAPPER

*FindPeaksWrapper*( $\dots$ ) is used to find noticeable variation in the sensors data flow. The program can be divided in four main elements: polyfit, Kalman filter, peak presence, variables update.

### A. Polyfit

For each sensor, for each new data acquisition, *poly\_fit* function predicts the next values, based on data already captured (it uses *polyfit* and *polyval* Matlab functions). This evaluation is done only if the number of elements to analyse is greater than degree of polyfit evaluation plus three (in the program degree plus three). This restriction is caused by the necessity of the presence of one element to verify and of two more elements for *polyfit*, so the parametric estimation has at least one degree of freedom.

### B. Kalman Filter

The Kalman filter is used in two cases: individually on the sensor if it has no correlation with others or for a multi-sensor Kalman filter evaluation. In both cases the filter is activated by the evaluation of slope of the regression line (the

coefficient is generated inside *polyfit* function). The Kalman filter needs the state transition matrix to be built (obtained from the differential equations of the model). In this case the model is a black box model thus it can't be described exactly. For these reason, the Kalman filter is used only if the slope is near zero (constant behaviour).

The corresponding transition matrix evaluates a constant behaviour if the sensor has no correlation with other sensors.

For multi-Kalman filter evaluation instead, the sensors used are speed, velocity and acceleration (linear or angular), collected inside *id\_kalman* variable. In these case, the state transition matrix describes a uniformly acceleration motion. This specific evaluation uses also *kalman\_ok* variable.

*kalman\_ok* is generated previously by the *multi\_kalman\_ok* function. Due to the possible time desynchronization, the function estimates time compatibility of specific sensors to permit greater accuracy of Kalman's filter.

The Kalman filter is used anyway because if the autonomous system makes continuous and without variations paths/movements, it will generate a more accurate evaluation than *polyfit*. This method needs also process covariance matrix ( $Pn$ ), measure covariance matrix ( $Q$ ) and sensor precision matrix ( $Rn$ ). All the matrix are composed by the all matrices are a composition of the respective individual matrices of the sensors, using the internal *diagonalizer* function. The process covariance matrix  $Pn$  is updated in each cycle of Kalman evaluation and and restore to initial condition (element 2 diagonal matrix) if Kalman filter evaluation is no longer possible. The covariance matrix of the measures ( $Q$ ) is set by diagonalize *var2* (the motivation of these setting is explained in next sections). The sensor precision matrix  $Rn$  is constant (element 0.1 diagonal matrix) to allow evaluation even in critical cases (behavior not completely constant).

### C. Peaks Presence

The peak presence is verified by calculating if the percentage of change between values expected and measured exceeds gap [%].

The value of gap, given by input of *FindPeaksWrapper* function, is modified inside *peak\_presence* function. This change is due to the fact that if the values approach zero, small variations generate a high percentage change.

### D. Internal variables update

Inside *FindPeaksWrapper*() there are few variables that are used both as input and as output (updated in every cycle). The most important and noteworthy is *var2*.

*var2* is average squared change between measure and prediction and it is used in the covariance matrix of the measure. It is used as covariance matrix of the measure because assuming the

This competition is sponsored by the IEEE Signal Processing Society and MathWorks

All author is with the University of Padue, Padova, ITA

noises as gaussian noises (processes is subjected to ambient, thermal and internal noise)

$$y_{measure} = y_{real} + e_{measure} \quad e_{measure} \sim N(0, \Sigma_{measure})$$

$$y_{estimate} = y_{real} + e_{estimate} \quad e_{estimate} \sim N(0, \Sigma_{estimate})$$

The measures of different axis are considered unrelated (covariance matrix is diagonal). Unidimensional estimation of variance

$$e = y_{measure} - y_{estimate} \quad N(0, \theta)$$

$$e_i = y_{measure_i} - y_{estimate_i}$$

$$l_e(\theta) = -\log \left( \prod_{i=1}^n p_{e_i}(\theta) \right)$$

$$= \frac{n}{2} \log(2\pi) + \frac{n}{2} \log(\theta) + \sum_{i=1}^n \frac{e_i^2}{2\theta}$$

$$\frac{\partial l_e(\theta)}{\partial \theta} = \frac{n}{2\theta} - \sum_{i=1}^n \frac{e_i^2}{2\theta^2}$$

$$\hat{\theta} = \frac{\sum_{i=1}^n e_i^2}{n}$$

### E. Input Output

In *FindPeaksWrapper* most of elements are cell element due to the different vector dimension and variables used. Detailed specification in *FindPeaksWrapper(...)*

Input

- t: time vector of sensors
- y: sensors measurement vectors
- type: type of sensor
- num: maximum number of elements used in polyfit evaluation
- gap: minimum percentage variation to report an anomaly
- gap\_k: maximum coefficient of linear regression to determine constant behaviour
- kalman\_ok: data matrix that activates multi-Kalman evaluation

Output

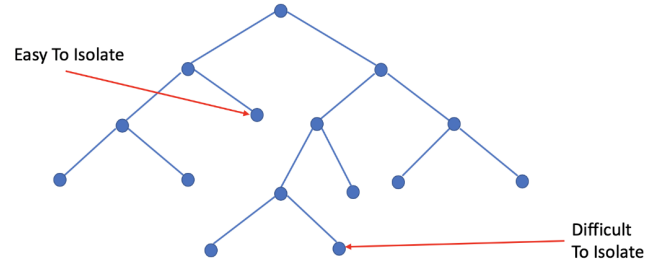
- already\_analysed: no new elements to analyse
- anomaly\_out: anomalies in the last captured data set
- index\_out: index of the first element in anomaly\_out referred to all measured data
- variation: differences between predicted and measured values
- y\_calc: output vector of the predicted values
- data\_type: general description of evaluated sensors
- kalman\_ok: update of input

## III. ISOLATION FOREST

*IsolationForest* is a Statistical method to analyze small dataset and find anomalies. An anomaly is when an observation deviates so much from others. In this case, anomalies are observation that not correspond to the right functioning of the Drone. We used this algorithm to analyze data when the observation received are approximately constant, we took this decision because we want to exclude the anomalies that could be find on the right functioning of the Drone. For example if we are studing height parameter we don't add to isolation forest the observation of the drone when it's taking off, because there is a very high variation of the observation in a small time and this can be identified as a false positive anomaly. However, when the drone is flying, a high variation of the height in a small time is very likely an anomaly. This function can be divided in four main elements: *IsolationTree*, *iForest* and *AnomaliesFinder*

### A. Isolation Tree

This function randomly creates a binary tree from a given dataset of observation. It randomly take a subsample of the data. Our data are the observation and every observation can have one or more dimension. The function randomly select a dimension and a value in that dimension and divides observation in two groups (observation with a greater or equal value and observation with lower value). This process is recursively repeated until all observation are isolated. Isolated observation are on the leaves of the tree.



### B. iForest

This function creates a lot of different isolation trees that's why it's called forest. Every tree is different because it's randomly generated by *IsolationTree* function, however the observations inside every tree typically are the same. *iForest* generate a forest that can be analyzed from *AnomaliesFinder* function.

### C. Anomalies Finder

This is the most important function of *IsolationForest* because it's used to find anomalies. It takes the forest and calculates the average height of every observation and anomalies will have an average height lower than normal observations.

Now the anomaly score  $s$  of an instance  $x$  on a database of  $n$  instances is defined as:

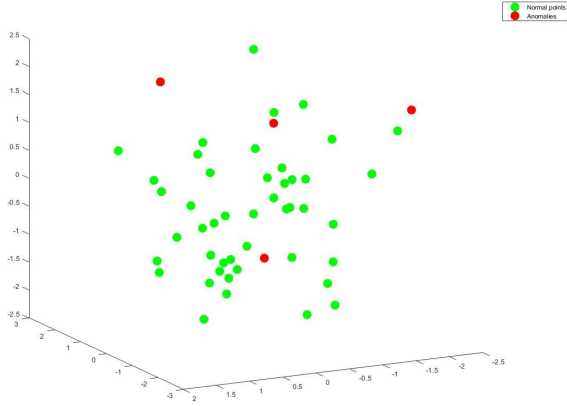
$$s(x, n) = 2 \frac{E(h(x))}{c(n)}$$

$$\text{where } c(n) = 2H(n-1) - \frac{n-1}{n}$$

$$\text{and } H(i) = \ln(i) + 0.5772156649$$

$s(x, n)$  give us an information about the observation:

- If  $s(x, n)$  is lower than 0.5,  $x$  is a normal observation
- If  $s(x, n)$  is closer to 1,  $x$  probably is an anomaly observation
- If  $s(x, n)$  is around 0.5 for all observation it's safe to assume that the dataset doesn't contain anomalies.



The main disadvantage of Isolation Forest is Swamping. Swamping is when normal instance are very closed to anomalies making them hard to be isolated. This problem can be minimized by subsampling the dataset.

#### D. Input

- NumTree: Number of trees inside the forest
- maxPoint: Maximum points inside every tree of the forest (for subsampling)
- sk: Anomaly threshold
- type: Type of the new element
- newEl: New point of the forest
- numLastPoints: Number of last points to use in the forest

#### E. Output

- Last: True if newEl is abnormal, false otherwise
- Abnormal: True if there are abnormalities, False otherwise
- posOfAnomaly: Index of abnormal points
- h: Average height of each point in the forest
- s: Anomaly score of each point in the forest

## IV. ANOMALY CHECK

This part contains classes and functions to structure data in a prioritized way, to avoid unnecessary controls over sensor that are less likely to give data representing a true anomaly

## V. SYSTEM VALUES UPDATE

The intantion is to build a real time program. To do this, the Bag manager has to adjust the reading time interval based on the time needed for the previous cycle. For this reason, keeping the time of a balanced cycle is extremely crucial, so that the function *OptimizeParameter()* keeps track of it and set the parameter of the various components based on the previous values and the variation of time that it's involved in all the program. The main idea is that if the program is fast enough, the algorithms can be even more precise otherwise they can be relaxed a bit.

## REFERENCES

- [1] Isolation Forest.
- [2] Hariri Forest.
- [3] Swamping and masking in anomaly detection in isolation forest.
- [4] Anomaly score of isolation forest.
- [5] Anomaly detection with isolation forest visualization.