Machine 1 EMS

**Problem Statement:**

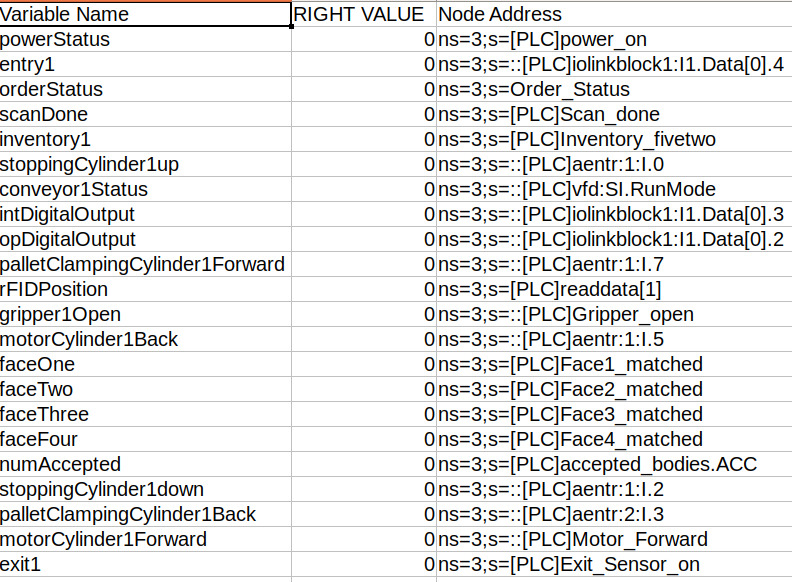
**To create a program that would be able to monitor the status of the Workstation-1 machine and be able to detect the fault: the part associated with and the type of fault.**

**Type of Data:**

All the tags of Machine-1 are divided into two types:

* Functional: The tags which constantly change throughout the operation and are an indicator of process as it progresses through the workstation
* State: The tags which are unchanged throughout the operation and are an indicator of ‘health’ of the machine. If some state tags are wrong, then is means that the machine is not in a right state to operate
  + State Tags common for all components
  + State Tags specific for each component

Operational Times (Timestamps) - Timestamps of select IoT Data, to keep track of the progress of the job through time.



Working Approach:

Idea:

An event is the part of the process of the machine. They occur one after the other. In case of change of event, out of all the tags, the functional tags have their values changed. The other tags can be said to represent the health of the machine. The functional tags together monitor the process within the machine while on job.

The idea is to compare the node values at each event change with the correct values. The correct values are stored in a matrix where each row has the values of functional nodes at each change. At each event change, the values of all the functional nodes is checked with corresponding row. The row numbers represent the the event number, which are incremented chronologically.

To check the difference between

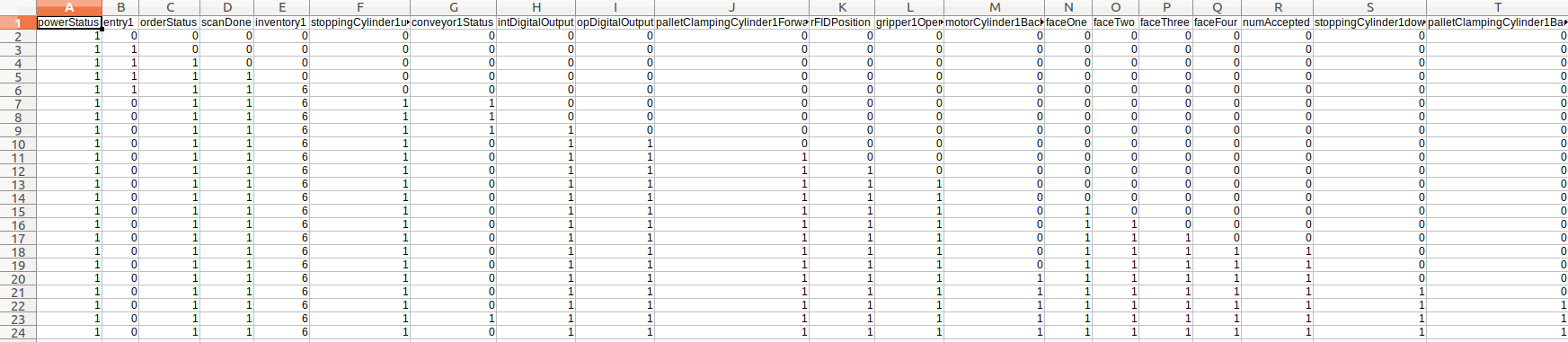
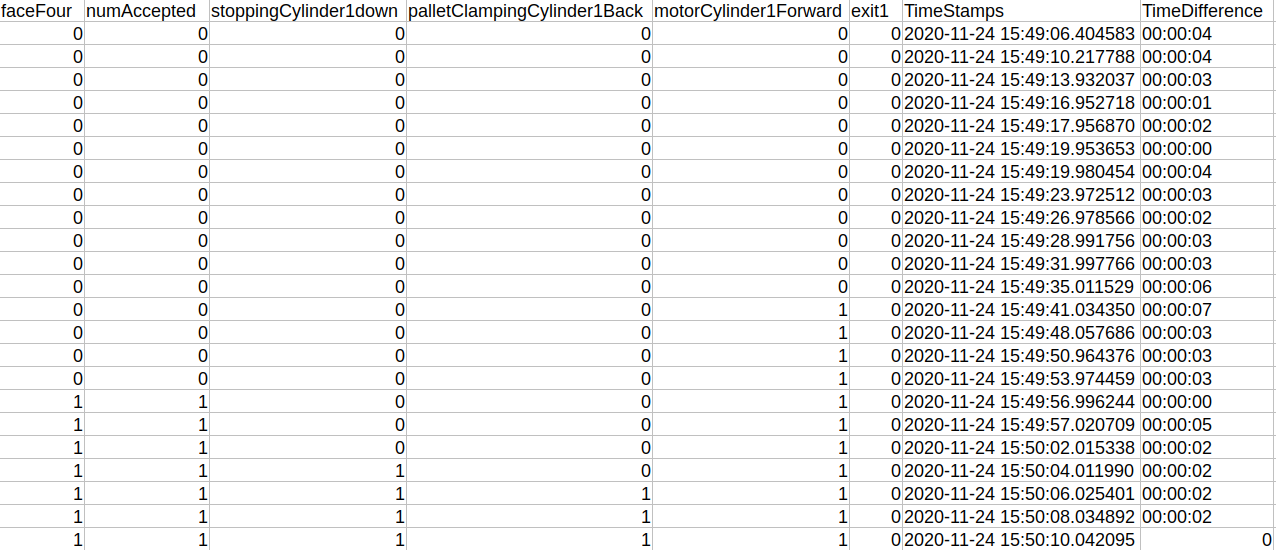
**The idea is to compare the tag values at each timestamp(event change) with the corresponding ones from the correct\_tags file(having all the expected values).**

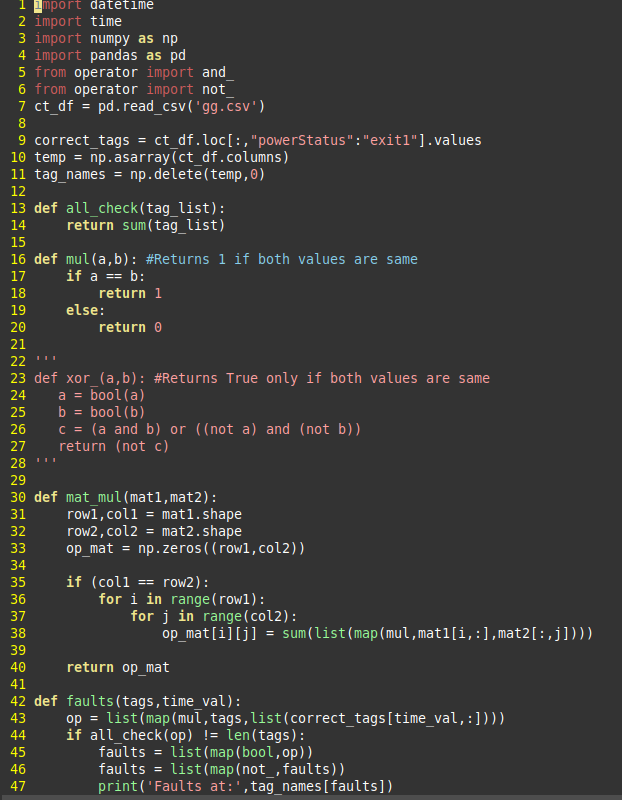
Functional tags(event generating tags) are monitored through subscription class. There are multiple threads running

1. One of which will start the subscription event,
2. Second thread will call the error detection function on every subscribed event and
3. The third thread has a timer which monitors the time between each event and if the expected time is exceeded, it will isolate the error area.

Steps:

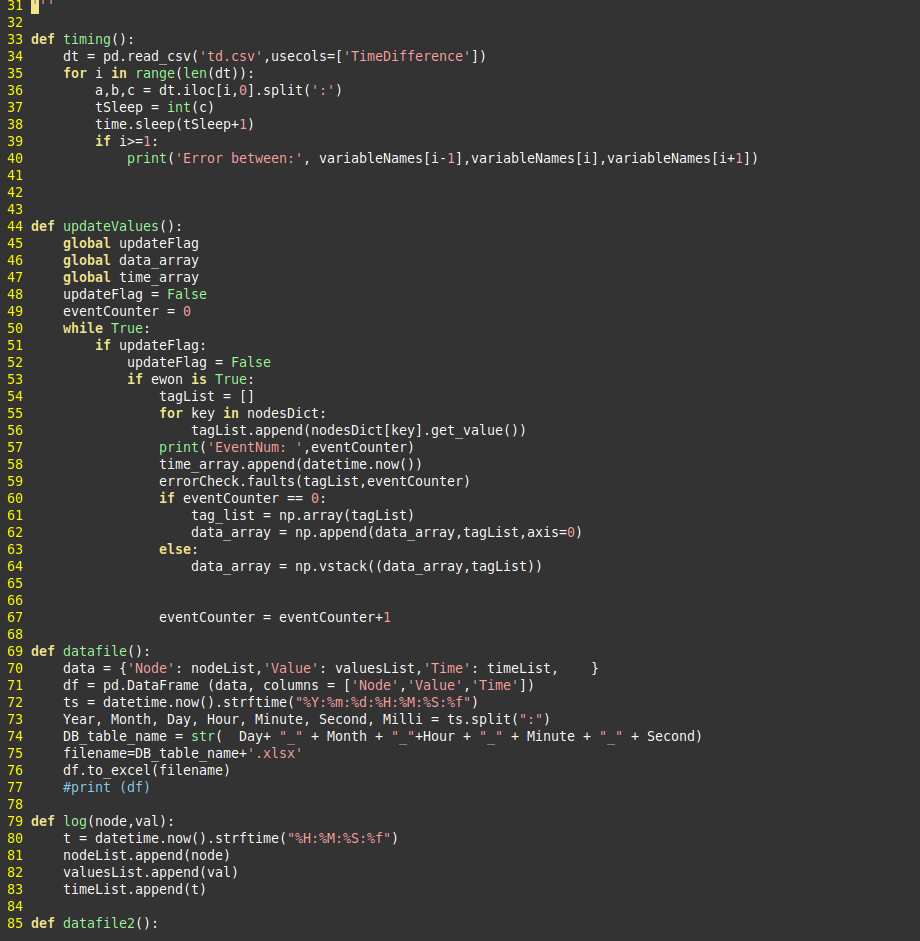
Version 1(Completed):

* There is a correct\_tags excel file containing expected values in rows appended according to various events. Each column represents a particular tag value and the second last column is the time stamp values. The last column contains the time difference between each event or the time taken by each event.
* The Client on every event calls the error checking function and passes the list containing current value of all tags. The error checking function compares the current and expected values. One of the parameters that also goes into the error checking function is the eventCounter. That is a simple counter which increments once each event occurs.
* 
* 
* The error checking function reads from a correct\_tags excel file. Then according to the eventCounter value, it reads the corresponding row and checks the incoming tags and tags from correct\_tags.
  + We have avoided adding if else by comparing the two lists through a xor gate like implementation. Xor gate returns 1 if both values are the same and 0 if both values are different.
  + The benefit of this approach is that, since we are comparing values of all the functional tags at every event, so if there is any issue in some previously visited station, that error will also be reflected in real time.



Link to version 1 demo: <https://drive.google.com/file/d/1vBbOk34BTxcZq0KAQjPR9IAvvRKG__zC/view?usp=sharing>

Version 2(Completed):

* Another thread(timing) is to monitor the time difference between the events.
  + This thread reads from column TimeDifference of correct\_tags.csv, which lists the time taken by each event to complete
* In execution:
  + The timing thread, sleeps for n+1 seconds as determined by the time taken by the nth event from the list. The delay is added to take into account some very minute delays that occur when running the actual machine.
  + It runs in a loop in timing thread which iterates over the all the time differences and sleep accordingly
  + A function is running a loop. If after event A, event B was supposed to trigger an event in 2 seconds and no such event is triggered in 2 seconds, then this thread which was not active(time.sleep) for 2 seconds will raise an error, isolating the area corresponding to A, B and C
* 

Links to version 2 videos

<https://drive.google.com/file/d/1zdjPbPBdx3jJ2bUBlo6qjsjY8WaUvsWp/view?usp=sharing>

<https://drive.google.com/file/d/1YNuf2N2_WwrRN2ceb6osJnhLRndOd4Jo/view?usp=sharing>

Version 3(Future Development):

Versions 1 and 2 are merged to form version 3. The idea is to generate errors due to delay only when an event has not occurred between the stipulated time period of that event. If the event occurs on time, the functional tags will be checked as it was being checked in version 1.

The area responsible for delay, i.e. the three tags corresponding to event n-1, event n and event n+1. There can be an error due to faulty nature of components/tags relating to event n-1 which may not allow to move on. There may be some error in the components of event n+1 due to which the event is not occurring at all.

The reason for error will be checked in the version 4.

Version 5(Review Pending):

* Once the potential areas are isolated, the core reason for the fault, is detected through checking all the tags responsible for those parts. The reason is then isolated through an algorithm to check for it’s status. For eg if we find an entry sensor was not working in the correct time period, we can check the corresponding tags of the entry sensor device. From here there are two possibilities. If we find out there is some error in the entry sensor, the corresponding tags will be giving out the faults. So we can conclude that the error was due to the entry sensor’s faulty operation. If we find that the entry sensor is working correctly, we can conclude that the error maybe due to fault in the job cycle of the machine(the palette may have been removed from the start)
* The isolated faults are cross checked with the connection diagram to give the relevant connections to check manually

Future Work and explorations

* *There will be a log function to collect the tag data and generate the correct\_tags file. The log function is made to run for the first operation or when we know for a fact that the machine’s operation is correct.*
  + *It can also be run multiple times to get all data about time. This maybe necessary because in one run, we may not be able to record the small discrepancies in timestamp which occur during operation. For an event it takes 5.02 seconds instead of the recorded 5 seconds is not an error, but it shows up as an error, if we have not taken into consideration these discrepancies. In short it gives us the range of the correct cycle.*
  + *Another advantage of doing this is that, we do not need to write the correct\_tags file, the program will record it automatically*
* Idea:
* Store the timestamps of when the event tags are triggered, and use a clustering algorithm to set the time difference between each events
* Steps:
* The machine is run multiple times, and the timestamp for each event is stored
* *Then making each event as a feature, and the timestamps as values, we train a clustering algorithm, k-means to divide to find out a decision boundary. The decision boundary will help it to determine whether time difference is acceptable or not.*

**Other UnSuccessful Approaches:**

**Approach 1:**

Idea:

We detect the condition of the machine by checking on the state of its tags. Since tags provide a comprehensive view of the machine, exploring the relations between the tags and resulting faults will help isolate the fault in the machine.

Steps:

* All kinds of faults are noted
* The resulting relationship with relevant tags is formed with simple boolean gates giving rise to a boolean expression. For e.g. -
  + If the sensor is not giving values since the correct voltage is not provided to the sensor, we find that all other relevant to the expression are correct and only the sensor\_low\_voltage tag is on, we can conclude that the fault is the low voltage supply to the sensor
  + If some error that so occurs which does not have its state measured, like the wired connection between SMPS and PLC. We can draw up a conclusion by checking the tags of SMPS and PLC. If SMPS is 1 and PLC is 0, we can conclude that this error is due to a faulty connection of wiring between SMPS and PLC and we can direct the said person to the area.
  + When a power problem occurs in the sensor module, the sensor power tag gets 1 value. The resulting expression: entry\_power\_issue = SMPS and SensorPowerSupply. This expression will only give 1 (indicating error) when SMPS is working properly and there is some power issue is there in the sensor side. If there is a fault at SMPS this expression will not give a 1 value irrespective of the value of the SensorPowerSupply tag. The corresponding SMPS power issue tag will be active.

Advantages:

* Fault isolation is excellent in this case
* The sequence of operation is also preserved even though the checking of tags happens all at a time
* It can detect fault as and when it occurs, even if the machine is running

Disadvantages:

* Heavy dependence on domain knowledge since all relations have to be written by hand
* Not scalable: Model made for a small no. of components is not scalable to model with a large no. of components
* Does not take into account time: This model does not have a concept of time into it and can do no find out the fault in case there occurs a delay when doing the job given all other tags are in their right values

**Approach 2:**

Idea:

We track the progress of the job through various sensor data and it’s corresponding time stamps.

Steps:

* We note the timestamps of the changes in data of the sensors which keep track of the progress of the job through the machine, like entry sensor, operational sensor or gripper status
* We create a secondary list (diff-list) containing the time difference of consecutive elements of the list of timestamps
* In case the diff-list has a value greater than a default list of the actual time difference it should take to perform the said task, we can determine the particular area between which fault occurred.
* We can then check only the limited tags required to identify faults in that region of the machine.

Advantages:

* We can reduce the number of tags we are checking and thus reduce the computational load
* Errors due to slowing down of process can be found
* Additional faults like id operator remove the palette midway could be detected through this

Disadvantages:

* We still have to manually write all the relationships
* May give true negatives in case some components could not be isolated well

**Approach 3:**

Idea:

We train a classifier to read the values of tags and classify the type of fault

Steps:

* Making a dataset on all possible permutations and combination of tags
* Labeling the permutations and combinations with relevant fault tag
* Training the model on data
* Testing the classification with new data

Advantages:

* This model can be scalable to include new components
* No need to identify relationships manually

Disadvantages:

* To create a proper training dataset, since we have 3800 tags, it corresponds to 3800 features. 3800 features make 2^3800 values of input data points only. It would be too much on a computationally heavy
* While creating a dataset, we have to manually analyze the relations and input the corresponding classification.
* Since we are already providing all the possible error combinations, the predictive capability of the ML is not used at all. It becomes a rule-based system

**Approach 4:**

Idea:

Use the ability of the architecture of the Recurrent Neural Network(RNN) to simulate the connection between inputs as a sequence and build the fault classification program even when the machine is running.

Steps:

* Feed data to the networks similar to how data is fed when predictive text generation()
* The RNN will get the inputs step by step from the running machine
* By default, there should be an error, but if there is no error, another input is taken up and the sequence is continued
* The number of tags making up the sequence taken here is less and we can identify the area/type of error.
* We then use the corresponding relationships with either a flowchart or boolean expressions to isolate the exact fault

Advantages:

* It has the ability to understand sequences. It would be useful to represent the operations in sequence as the job moves forward in the machine
* It has the ability to integrate timestamps as a feature for detection
* It can take varied size inputs. This will help in monitoring the machine live as if the fault occurs midway between operations, we can input the tags available to us rather than waiting for tags from the entire operation to detect the fault.

Problems:

* Idea was to add a new tag to the sequence of tags being checked by having the parameter check the time difference between them and then add it to the sequence. If the time difference between the two events is too much, then we do not add it to the sequence and we predict it as a fault with the remaining tags. If the time difference is within limit, we add the tag to the sequence of things that have to be checked.
* This idea requires one to change the entire structure of RNN. This will undermine the entire idea of RNN to be used for sequences and thus RNN will not serve its purpose. To create changes in the weights of the code while training requires changing the backend of tensorflow. There were unsuccessful trails while implementing and changing such.

**Approach-5**

Idea:

Store the timestamps of when the event tags are triggered, and use a clustering algorithm to set the time difference between each events

Steps:

* The machine is run multiple times, and the timestamp for each event is stored
* Then making each event as a feature, and the timestamps as values, we train a clustering algorithm, k-means to divide to find out a decision boundary. The decision boundary will help it to determine whether time difference is acceptable or not.

Disadvantages:

* To check whether the a time tag is delayed, the whole function has to be run once and after getting all the timestamps the algorithm can say whether there has been anomalies
* If some events are repeated, this approach fails, as the latest events timings are recorded and is generated as an anamoly