Sensor Anomaly Detection in Wireless Sensor Networks, Reproducing and Expanding Upon Previous Work

Harry Rybacki

University of North Carolina at Greensboro
hrybacki@gmail.com

Abstract

As the number of Wireless Sensor Networks (WSNs) continues to grow with uses monitoring, tracking, and controlling a wide variety of industry and private applications so does the need to accurately and efficiently detect anomalies in the sensor data they accumulate. A wide variety of statistical-based, clustering-based, classification-based, and nearest-neighbor methods have been proposed[2]. Dr. Shan Suthaharan's work using post-data-transformation, ellipsoid boundary estimation has proved to be incredibly accurate while also increasing efficiency when compared to similar ellipsoid boundary estimation based anomaly detection algorithms (ADAs)[1]. There is however, room to decrease the computational complexity of the transformations involved in Dr. Shan's ADA. The goals of this project are to firstly transform Dr. Suthaharan's previous work into an easily reproducible environment using Python and IPython notebooks and secondly to create a solid performance benchmark which an be used in future work that will decrease the complexity of Dr. Suthaharan's ADA while maintaining approximately the same level of accuracy.

I. Introduction

In order to accurately and efficiently detect anomalies from sensor data within Wireless Sensor Networks (WSNs), we have decided to reproduce and enhance directly related, previous research performed by Dr. Shan Suthaharan.

DESCRIBE SECTIONS HERE

I. Document Conventions

Italicized words represent common terms used throughout the requirements document which may be looked up in Appendix A: Glossary.

II. Project Vision and Scope

This primary goal of this research project is to translate previous research conducted by Dr. Shan Suthaharan into an easily reproducible environment, using *Python* and *IPython Notebooks*, thereby permitting any researcher who desires to easily reproduce his experiments. The secondary goal is to create a solid performance benchmark, using the aforementioned technologies, that will act as the stepping stones for further research into making the transformation, sensor data boundary modeling, and inverse transformation modules of the algorithm more efficient. Time permitting, the tertiary goal is to begin optimizing aforementioned algorithm modules and creating comparative benchmarks to assist follow up research.

III. Stakeholders

The stakeholders for this research project include Dr. Shan Suthaharan, other researchers within this and complementary fields, as well as individuals, groups, and organizations whom either currently or in the future will setup, maintain, or use *WSNs*.

IV. Assumptions and Constraints

Although the original implementation was written in *MATLAB*, the reproduction will be written in *Python* and composed in *IPython*

Notebooks to ensure the easiest form of experimentation and reproduction possible.

II. Requirements

I. Functional Requirements

1. Read *WSN* Data:

    - Researchers require the ability to read *WSN* data from a variety of sources. To accommodate this, a standard schema should be outlined for easy of integration and processing of data after any data migrations deemed necessary.

2. Process WSN Data:

    (a) Perform Transformation on *WSN* Data:

    - After reading the data, measurements need to be determined and transformed into a Gaussian distribution field.

    (b) Perform Boundary Modeling on *WSN* Data:

    - After being transformed, boundaries must be determined and true measurements segregated from others.

    (c) Perform Inverse Transformation on *WSN* Data:

    - Finally, *anomalies* should be determined and traced back to their originating sensors.

3. Visually Model *WSN* Data:

    - Researches require the ability to take the resulting data from the processing phase and model it visually allowing them to identify true measurements from anomalies.

4. Benchmark Performance of *ADA*:

    - In order to increase detection *efficiency* while maintaining detection *effectiveness*, researchers require an easy-to-use benchmarking tool. These benchmarks provide a definitive method of quantifying improvements as well as track progress of experimentation.

5. Allow Easy Modification and Re-execution of *ADA*:

    - Researches require a simple environment for re-executing the read, process, visually model, and benchmark phases. This environment will encourage experimentation as well as encourage an iterative development and enhancement workflow.

II. Non-functional Requirements

1. Programming Languages:

    - Python
    - Bash or related scripting language as needed

2. Development Environment:

    - OSX 10.9
    - VIM
    - *IPython Notebooks*
    - Firefox/Chrome

3. Performance Requirements:

    - As outlined in [2], the ADA created by Dr. Suthaharan has a very high *detection effectiveness*. However, despite making advances in *detection efficiency*[1] when compared to other *ADAs*, the transformation is still very computationally complex; this is the area this research project aims to quantify in preparation for follow up research.

III. Security Requirements

None.

IV. Software Quality Attributes

Two metrics are used to determine the quality of an *Anomaly Detection Algorithm (ADA)*: *Detection effectiveness* and *Detection efficiency*. These metrics will be used to determine the quality of Dr. Suthaharan's *ADA* as it is reproduced throughout this research project. As outlined in [2], the *ADA* created by Dr. Suthaharan has a very high *detection effectiveness*. However, despite making advances in *detection efficiency*[1] when compared to other *ADAs*, the transformation is still very computationally complex; this is the area this research project aims to quantify in preparation for follow up research.

III. Specifications

Table 1: *Project Specifications*

Functional Requirement	Sub-Requirement	Priority
Process WSN		**Essential**
Data	Perform transformation on WSN data	**Essential**
	Perform boundary modeling on WSN data	**Essential**
	Perform inverse transformation on WSN data	**Essential**
Visually Model		**Essential**
Resulting Data	Create/Display plot of initial measurements	**Essential**
	"" "" post-processing measurements	**Essential**
	"" "" highlighting true measurements	**Essential**
	"" "" datapoint relation with source sensors	Desirable
Benchmark		**Essential**
Performance	Separate modules into blocks that can be re-executed	**Essential**
of ADA	Perform benchmarking of individual blocks	**Essential**
	Output benchmark results of individual blocks	**Essential**
	Output composite benchmark results of all blocks	Desirable
Modification and		Desirable
Re-execution of ADA	Allow Easy Modification and Re-execution of ADA	**Essential**
	Perform re-execution on individual blocks	**Essential**
	Perform re-execution on all blocks	Desirable

IV. Feasibility Study

TBA

V. Process Models - Drafts

Reference Appendix B, Figures 1 - 6.

VI. Data Models - Drafts

Reference Appendix B, Figure 7.

VII. Data Dictionary

- **sensor_data_storage**: Database or file storing *WSN* sensor data.
- **true_measurements**: Unformatted dictionary of original measurement data taken from sensor_data_storage.
- **true_measurement_tuples**: Formatted dictionary of tuples mapping sensor data to humidity and temperature data.
- **randomized_measurement_tuples**: Randomized dictionary of tuples mapping

sensor data to humidity and temperature data.

- **difference_tuples**: Intermediate dictionary of tuples created from randomized_measurement_tuples and used to calculate ellipsoid_data.
- **ellipsoid_data**: Data set used for calculating and plotting scatter plot.
- **lookup_table**: Table used mapping difference_tuples data points to their true_measurement counterpart.
- **transformed_measurement_to_orig_sensor**: Map of transformed data points to sensor that original detected them.
- **code_blocks**: Blocks of code within *IPython Notebook*.
- **block_benchmarks**: Set of performance benchmarks for each code block within the *IPython Notebook*
- **benchmark_report**: Summary of performance benchmarks from all code blocks.
- **orig_data_scatterplot**: Scatterplot of original data.
- **transformed_data_scatterplot**: Scatterplot ellipsoid and anomalies from transformed data.

VIII. Data Models - Revised

Reference Appendix B, Figure 8.

IX. Acknowledgement

1. Add Dr. Shan Suthaharan's Info
2. Add Sponsor's info

References

[Figueredo and Wolf, 2009] Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy - a cross-cultural study. *Human Nature*, 20:317–330.

[Suthaharan] Suthaharan, S. (2010). Sensor Data Boundary Estimation for Anomaly Detection in Wireless Sensor Networks.

[Rassam, Zainal, and Maarof, 2013] Rassam, M., Zainal, A., & Maarof, M. (2013). Advancements of Data Anomaly Detection Research in Wireless Sensor Networks: A Survey and Open Issues. Sensors.

[Hodge, Austin, 2004] Hodge, V., & Austin, J. (2004). A survey of outlier detection methodoligies. *Artificial Intelligence*, 22:85-126.

[Chandola, Banerjee, Kumar, 2009] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A Survey. *ACM Computing Surveys*.

[Zhang, Meratnia, Havings, 2010] Zhang, Y., Meratnia, N., & Havinga, P. (2010). Outlier Detection Techniques for Wireless Sensor Networks: A Survey. *IEEE Communications Surveys & Tutorials*, 159-170.

X. Appendix A: Glossary of Terms

- **Anomaly**: An anomaly is an observation that seems to be inconsistent with the rest of a dataset[3] or the process of finding data patterns that deviate from expected behavior[4].
- **Anomaly Detection Algorithm (ADA)**: An algorithm used to detect anomalies.
- **Detection effectiveness of an ADA**: The detection accuracy, detection rate, and number of false alarms of an ADA[5].
- **Detection efficiency of an ADA**: The energy consumption and memory utilization of an ADA[2].

- **MATLAB**: MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.
- **Python**: Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C.
- **IPython**: IPython is a command shell for interactive computing in multiple programming languages, originally developed for the Python programming language, that offers enhanced introspection, rich media, additional shell syntax, tab completion, and rich history.
- **IPython Notebook**: The IPython Notebook is a web-based interactive computational environment where you can combine code execution, text, mathematics, plots and rich media into a single document.
- **Wireless Sensor Network (WSN)**: A wireless sensor network (WSN) of spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, pressure, etc. and to cooperatively pass their data through the network to a main location.

XI. Appendix B: Figures

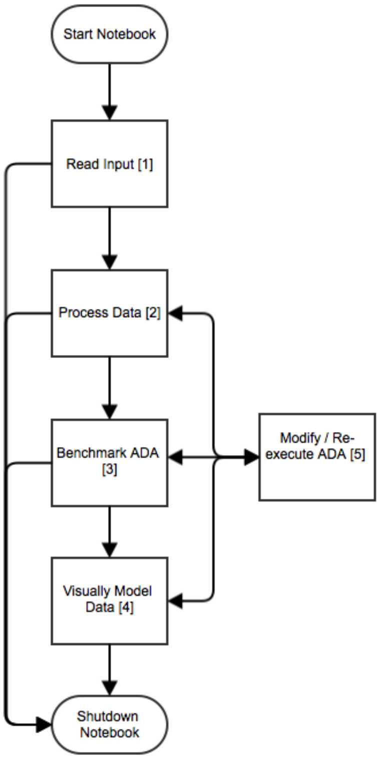

Figure 1: *Process Model Overview - Draft*

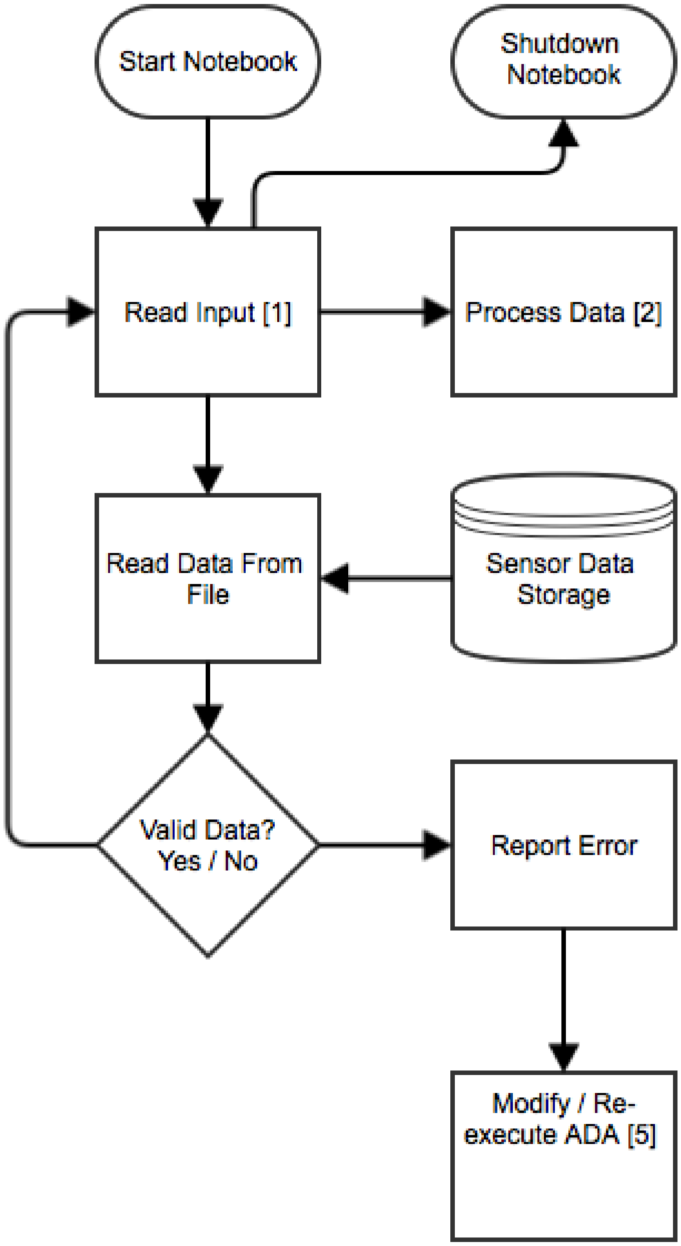

Figure 2: *Process Model - Read Input - Draft*

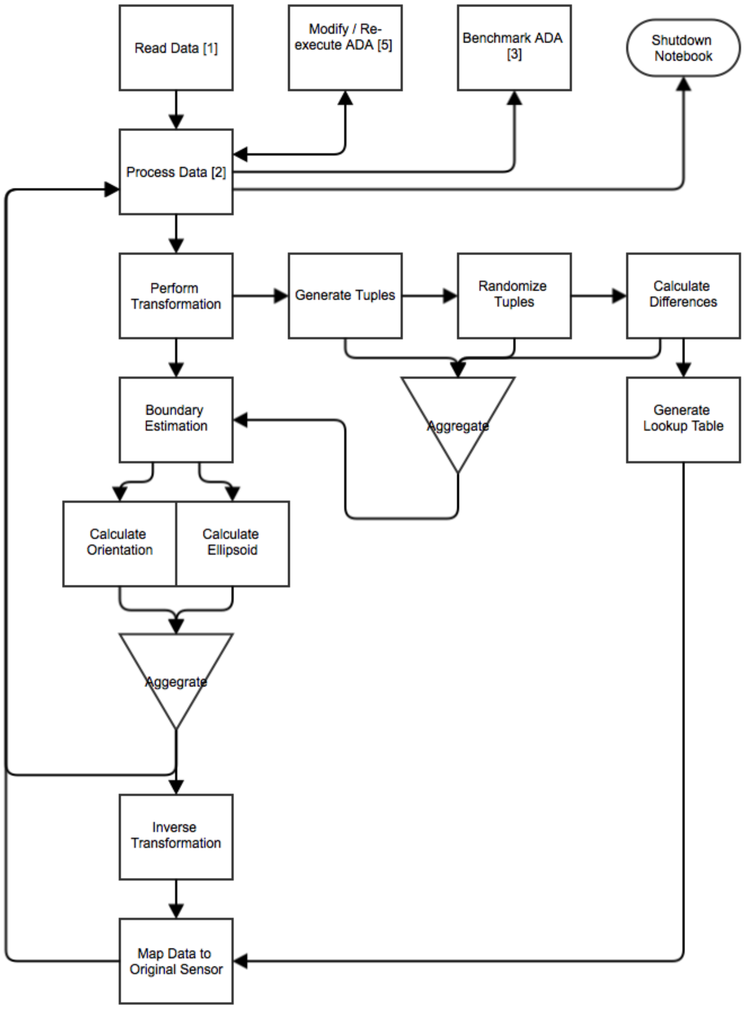

Figure 3: *Process Model - Process Data - Draft*

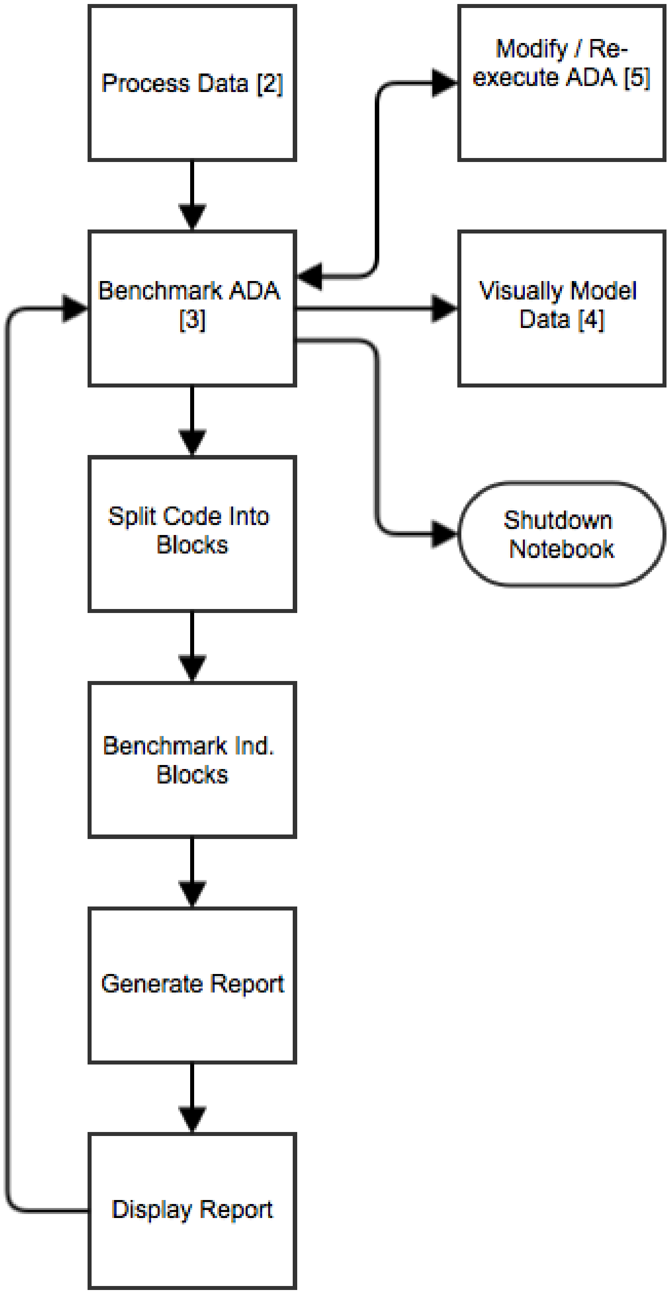

Figure 4: *Process Model - Benchmark ADA - Draft*

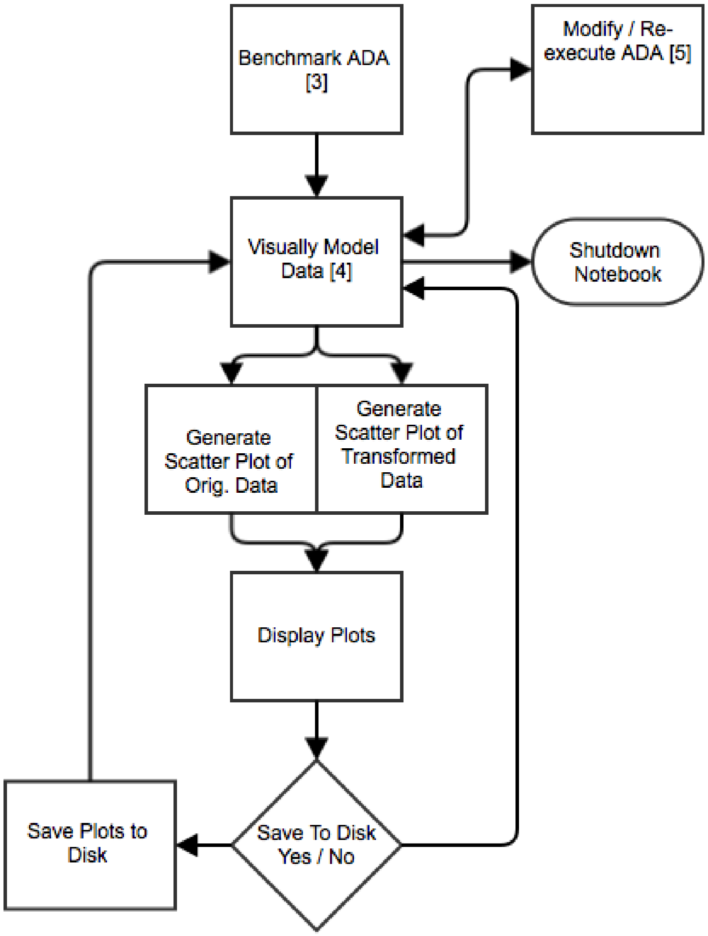

Figure 5: *Process Model - Visually Model Data - Draft*

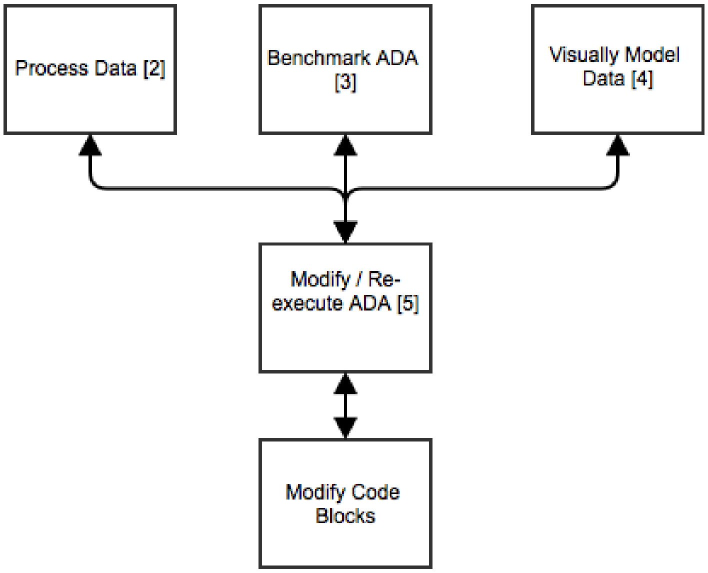

Figure 6: *Process Model - Modify or Re-execute ADA - Draft*

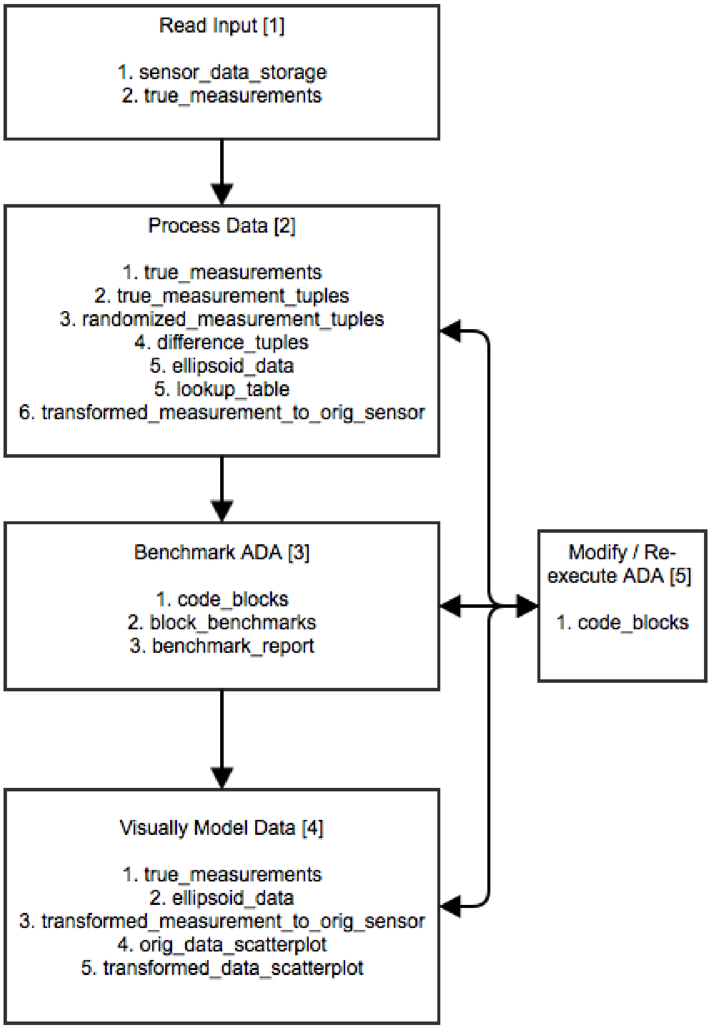

Figure 7: *Data Model Overview - Draft*

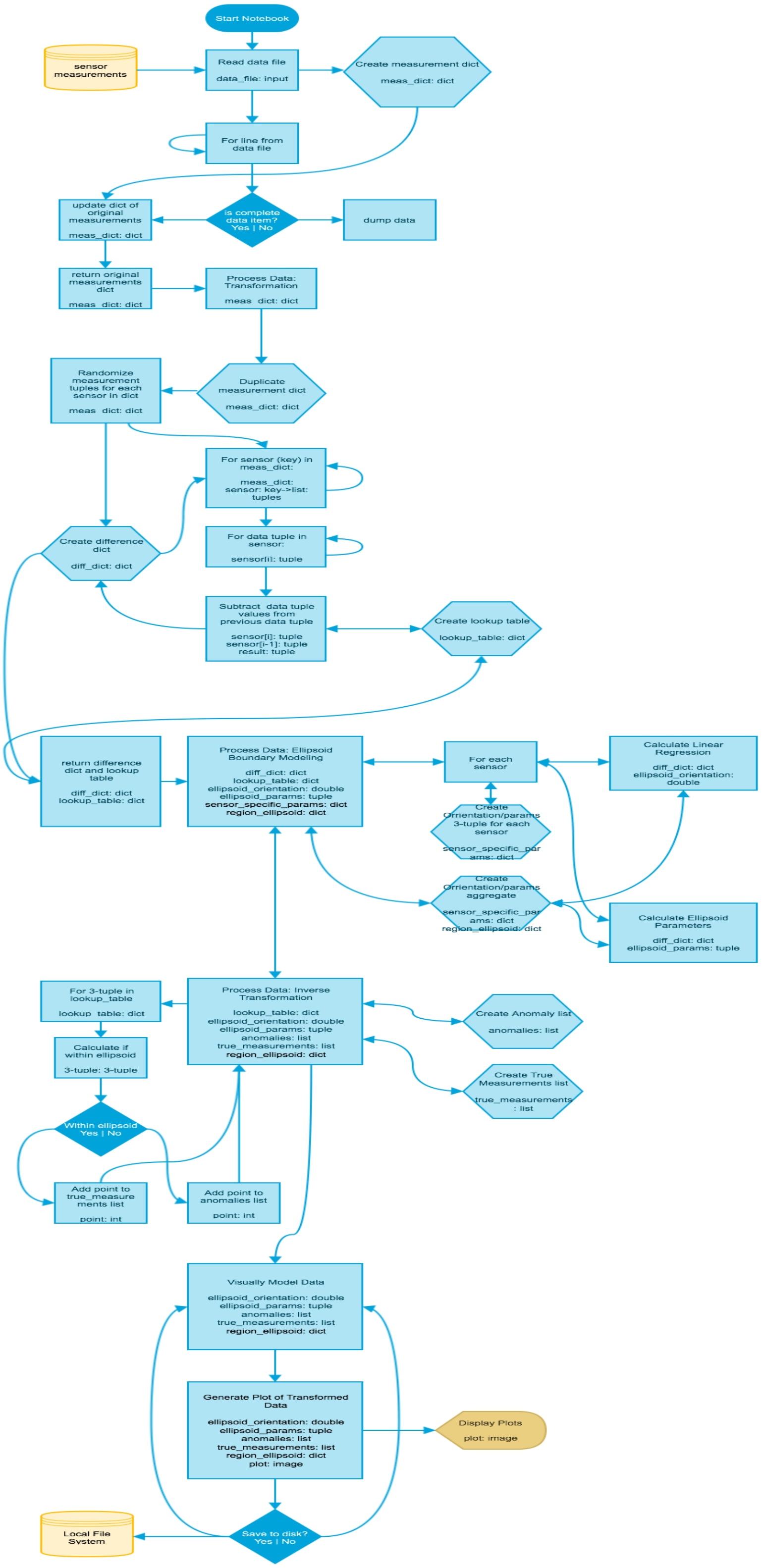
Start Notebook
sensor measurements
Read data file
data_file: input
Create measurement dict
meas_dict: dict
For line from data file
update dict of original measurements
meas_dict: dict
is complete data item? Yes | No
dump data
return original measurements dict
meas_dict: dict
Process Data: Transformation
meas_dict: dict
Randomize measurement tuples for each sensor in dict
meas_dict: dict
Duplicate measurement dict
meas_dict: dict
For sensor (key) in meas_dict:
meas_dict: sensor: key->list: tuples
Create difference dict
diff_dict: dict
For data tuple in sensor:
sensor[i]: tuple
Subtract data tuple values from previous data tuple
sensor[i]: tuple
sensor[i-1]: tuple
result: tuple
Create lookup table
lookup_table: dict
return difference dict and lookup table
diff_dict: dict
lookup_table: dict
Process Data: Ellipsoid Boundary Modeling
diff_dict: dict
lookup_table: dict
ellipsoid_orientation: double
ellipsoid_params: tuple
sensor_specific_params: dict
region_ellipsoid: dict
For each sensor
Calculate Linear Regression
diff_dict: dict
ellipsoid_orientation: double
Create Orrientation/params 3-tuple for each sensor
sensor_specific_params: dict
Create Orrientation/params aggregate
sensor_specific_params: dict
region_ellipsoid: dict
Calculate Ellipsoid Parameters
diff_dict: dict
ellipsoid_params: tuple
For 3-tuple in lookup_table
lookup_table: dict
Process Data: Inverse Transformation
lookup_table: dict
ellipsoid_orientation: double
ellipsoid_params: tuple
anomalies: list
true_measurements: list
region_ellipsoid: dict
Create Anomaly list
anomalies: list
Calculate if within ellipsoid
3-tuple: 3-tuple
Create True Measurements list
true_measurements: list
Within ellipsoid Yes | No
Add point to true_measurements list
point: int
Add point to anomalies list
point: int
Visually Model Data
ellipsoid_orientation: double
ellipsoid_params: tuple
anomalies: list
true_measurements: list
region_ellipsoid: dict
Generate Plot of Transformed Data
ellipsoid_orientation: double
ellipsoid_params: tuple
anomalies: list
true_measurements: list
region_ellipsoid: dict
plot: image
Display Plots
plot: image
Local File System
Save to disk? Yes | No

Figure 8: *Revised dataflow digram.*