

1415 N CHERRY AVE
CHICAGO, IL 60642
(312) 281-6900
DMDII.ORG
DMDII@UILABS.ORG



DMDII
a UI LABS Collaboration

DIGITIZING AMERICAN MANUFACTURING

DMDII FINAL PROJECT REPORT

Bottom-Up Plug-and-Play Hardware/Software Toolkit for Monitoring, Diagnostics and Self-Correction	
Principle Investigator / Email Address	Phil Callihan/philc@ncms.org
Project Team Lead	National Center for Manufacturing Sciences (NCMS)
Project Designation	DMDII-15-14-09
UI LABS Contract Number	0220160029
Project Participants	Perisense ACE Clearwater Enterprises Georgia Tech Research Corporation
DMDII Funding Value	\$726,278.00
Project Team Cost Share	\$1,038,987.00
Award Date	January 3, 2017
Completion Date	December 31, 2018

SPONSORSHIP DISCLAIMER STATEMENT: This project was completed under the Cooperative Agreement W31P4Q-14-2-0001, between U.S. Army - Army Contracting Command - Redstone and UI LABS on behalf of the Digital Manufacturing and Design Innovation Institute. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of the Army.

DISTRIBUTION STATEMENT A. Approved for public release; distribution unlimited.

TABLE OF CONTENTS

Page(s)	Section
2	I. Executive Summary
4	II. Project Overview
23	III. KPI's & Metrics
25	IV. Technology Outcomes
39	V. Accessing the Technology
40	VI. Industry Impact & Potential
43	VII. Tech Transition Plan & Commercialization
47	VIII. Workforce Development
47	IX. Conclusions/Recommendations
50	X. Lessons Learned
51	XI. Definitions
52	XII. Appendices

I. EXECUTIVE SUMMARY

The National Center for Manufacturing Sciences (NCMS) assembled a multi-disciplinary team to develop a sensor toolkit for legacy shop floor machines that captures data which is then transmitted to a secure cloud and converted into reports that allows companies to make quality, production, and operations decisions. The team included software and hardware development provided by the Georgia Institute of Technology, further development and commercialization by Perisense, and manufacturing test environment provided by ACE Clearwater.

The sensor toolkit successfully captures data in real-time from manufacturing equipment, processes, parts, and worker interactions, and converts it into actionable intelligence for immediate and long-term improvements. The sensor toolkit is compatible with machines with legacy controllers and empowers operators and supervisors at small and medium-sized manufacturers (SMMs) as well as large companies to increase quality and efficiency. It is also affordable for companies who do not have costly, complex legacy enterprise resource planning (ERP) tools, to improve their real-time understanding of shop floor problems. The completed toolkit uses robust and commercially available sensors for data acquisition combined with secure cloud servers and intuitive analytics tools to deliver actionable information for real-time machine process diagnostics and monitoring.

This solution provides end users with new visibility into operations, empowering and encouraging all stakeholders to improve quality, safety, and productivity. Data is displayed via a web interface that can be viewed on desktop computers, laptops, or portable mobile devices allowing monitoring from the factory floor or remote offices. The data captured also can be linked to conventional ERP systems and serve as a platform for advanced analytics and business intelligence.

Industry Problem

U.S. manufacturers face competitive pressures both domestically and globally. Developing nations are aggressively seeking to attract manufacturing operations to their low-cost locations while countries like China and India are moving up the value chain, seeking larger slices of the high-tech manufacturing market. At home, domestic manufacturers are challenged with a rapidly changing technology landscape and the imperative to stay current or have their businesses fail. The competition is fiercest for U.S. based SMMs that must compete on price and under the constraint of the higher cost of operating in a developed nation. U.S. based SMMs also must comply with strict environmental and safety requirements from which foreign competitors are exempt.

Many SMMs have older legacy machines, that are productive but lack monitoring and data analytic capability. Sometimes these machines serve niche market sectors where investment is difficult to justify. In other cases, SMMs are skeptical that the return on investment (ROI) justifies the purchase of new equipment.

New equipment with the capability to monitor and report performance data also requires capital expenditures and can cause production delays during installation. New equipment with built-in monitoring also requires information technology (IT) networks and cyber security requirements which add additional barriers to adoption. Equipment connected to the internet requires safeguards to prevent hacking, intrusion, and regular maintenance which adds to the cost of implementation.

SMMs are faced with a difficult situation—fail to upgrade production equipment and risk becoming less competitive as rivals upgrade or invest precious capital and lose production time to install expensive new equipment with monitoring capability along with the expensive required infrastructure.

SMMs wrestle with the challenge and promise of Industry 4.0 or I4- the current trend of automation and data exchange between humans and equipment on the factory floor while dealing with the reality of fierce competition. These smart factory principles look to revolutionize manufacturing but SMMs will only realize these benefits if they can find a way to affordably retrofit their current equipment.

The project team developed a solution that provides SMMs with the ability to easily and affordably monitor and optimize their current factory floor equipment in order stay competitive with both domestic and foreign competitors. The data gathered by the sensor tool kit also allows SMMs to implement condition-based maintenance (CBM) on older equipment which can further help reduce costs.

Large manufacturers can also benefit from this sensor toolkit. The affordability and ease of installation makes this solution ideal for deployments in test cells or for deployment in remote manufacturing sites. It can also be used to validate data from equipment instrumented by other means.

How the Problem was Addressed

The development team chose components and developed software using the following criteria:

- Affordability—whenever possible the sensor toolkit would use commercial off-the-shelf (COTS) components and open source software tools.
- Capability—the sensor toolkit needed to capture data that SMMs would find valuable to understanding and optimizing operations.
- Robustness—the toolkit sensors had to be able to operate in an industrial setting with a minimum of maintenance.
- Ease of use—the sensor toolkit had to display data in a simple, easy to understand format so that company executives and machine operators could both understand the data quickly without extensive training. It also had to be easy to install with a minimum of downtime.
- Minimal IT Footprint—The toolkit must not require extensive re-configuration of the existing IT network.

Summary of Project Outcomes and Recommendation

The sensor toolkit solution developed by the team was demonstrated in a welding application at ACE Clearwater, an SMM supplier to the Department of Defense (DOD). Such processes are difficult to monitor and costly to automate, especially in short-run or widely variegated production.

Results were available via mobile or desktop devices with sensors used to monitor environmental conditions that might impact weld quality. This monitoring was delivered through synchronous data during active welding operations, and was readily accessible during each operation.

This project team successfully created an impactful sensor toolkit that will enable manufacturers to easily and affordably monitor legacy shop floor equipment. The product was deployed in multiple

environments for over 18 months and proved rugged enough to operate on the factory floor while reporting data to a secure cloud.

The following recommendations would help promote further adoption development of the sensor toolkit.

- Perform SMM outreach about the advantages and utility of retrofitting legacy equipment.
- Education to address security concerns regarding cloud solutions.
- Conduct further pilot projects to encourage wider deployment among SMM and document additional use cases.
- Development of additional of sensors.
- Advertising of the discounted sensor toolkit for DMDII/MxD members.

DMDII SMM members can purchase a base product package from Perisense including four sensors including one year of storage and monitoring for \$5,500 (regular price \$6,000) plus travel costs for installation. Large OEM DMDII/MxD members can purchase a product package from Perisense including ten sensors including one year of storage and monitoring for \$14,000 (regular price \$15,000) plus travel costs for installation.

II. PROJECT REVIEW

Replacing legacy machine tools could cost hundreds of thousands of dollars not to mention the downtime and integrations costs required for new installations. Current factory sensing, controls, and management software applications are designed for large enterprises and the accompanying costs of implementation and maintenance are prohibitive for most SMMs.

But machine monitoring by selecting the right mix of sensors for each a use case can provide the following crucial benefits to SMMs:

- Reduce downtime from the analysis of real usage data;
- Provide data to support operator observations and;
- Support adoption of condition-based maintenance.¹

Even basic monitoring can pay huge dividends for SMMs. Simply monitoring power usage of a spindle motor can help a company manage the quality of its equipment and improve maintenance schedules.

“...Directly monitoring the power consumed by the spindle motor allows you to understand exactly what is happening with the tool. A new, sharp tool requires less power to cut than a worn, dull tool...If the power goes up suddenly, that may mean a crash or a broken tool.”²

¹ <https://www.shopfloorautomations.com/machine-monitoring-benefits/>

² <https://todaysmachiningworld.com/magazine/how-it-works-tool-monitoring/>

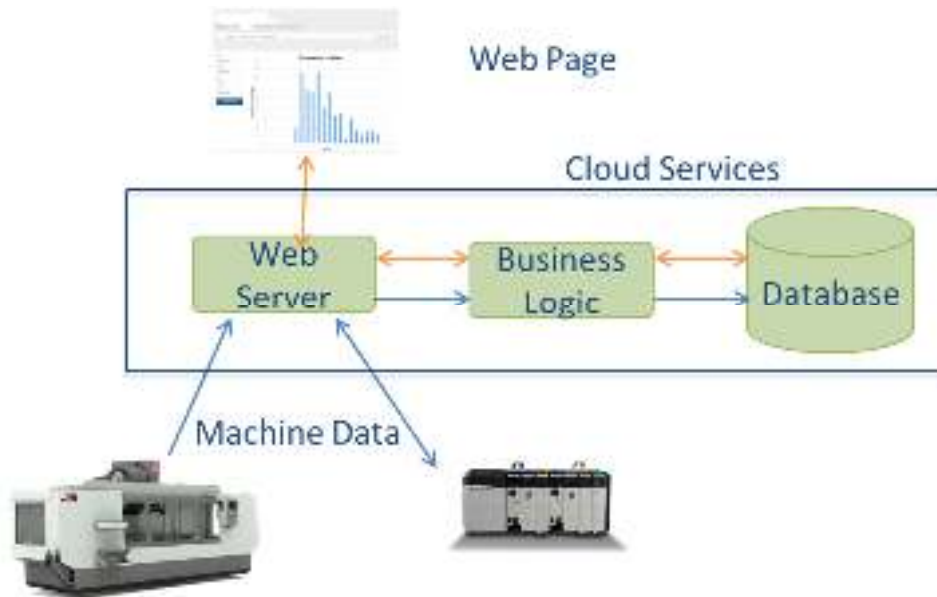


Figure 1-- Cloud Architecture for Sensor Toolkit

The team successfully created a capable sensor toolkit solution while addressing cost and implementation barriers that have prevented widespread adoption by SMMs. The sensor toolkit captures data in real-time from manufacturing equipment, processes, parts, and then converts it into actionable intelligence for immediate and long-term improvements. This solution empowers operators and supervisors at SMMs to increase quality and efficiency with ease and decisiveness.

The sensor toolkit also helps SMMs too small to use costly, complex ERP tools, improve their real-time understanding of shop floor problems. The sensor toolkit deployments use capable and commercially available sensors for data acquisition combined with cloud servers and intuitive analytics tools that deliver actionable information for process diagnostics, and monitoring.

Project Scope

The overarching goal was to develop and demonstrate a means of providing SMMs with an affordable, capable tool for real-time machine monitoring, diagnostics while providing factory floor visibility and reporting at a far lower cost than acquiring new equipment and deploying existing ERP and manufacturing management software tools.

Georgia Tech and Perisense identified the following top-level goals at the beginning of the project:

- Identification and acquisition of a suite of sensors suitable for supporting welding process data acquisition in welding cells.
- Integration of the sensor suite with wireless communications including Wi-Fi, and cellular connections.
- Development of a cloud-based platform for acquiring sensor data from the welding system(s), welding and other shop floor equipment for management personnel.
- Reporting system that uses data acquired from the sensor and human communication data to report quality and productivity metrics to shop floor and management.

Objectives

Specific objectives that support these goals included:

- Development of a set of retrofittable wireless sensors suitable for use in SMMs and other manufacturing facilities.
- Development of a platform for acquiring sensor data and recording interactions of shop floor personnel with systems, at specified intervals and transferring it to a database.
- Development of a data base architecture and data analytics package that allows actionable intelligence to be produced from the sensor and shop floor communications data.
- Development of a reporting system that clearly communicates this actionable information on an on-demand basis via computers or mobiles devices.
- Demonstration of the prototype system at ACE Clearwater and making it available to DMDII/MxD members at a discount to facilitate further deployment.

The team projected the following work schedule for the sensor toolkit development:

Work Schedule									
The Work Schedule is summarized in the following table:									
Table 5. Work Schedule									
Program Months	1-3	4-6	7-9	10-12	13-15	16-18	19-21	22-24	
Task 1 Deliverable - Baseline Metric Assessment									
1.1 - Development of specifications for sensors									
1.2 - Development of data collection plan									
1.3 - Achieve consensus on sensors and data collection plan									
Task 2 Deliverable - System Requirement Document									
Task 2 - Define capabilities for collecting machine data, application internal functions									
2.1 - Define capabilities for collecting machine data									
2.2 - Test capabilities with project group									
2.3 - Define application internal functions									
2.4 - Validate internal functions									
2.5 - Refine solutions									
Task 3 Deliverable - Sensor Specification									
Task 3 - Select, Acquire and Test Sensors									
3.1 - Select sensors									
3.2 - Acquire sensors									
3.3 - Test sensors for noise sensitivity									
3.4 - Harden sensors if required									
3.5 - Report on suite of tested sensors that meet specs									
Task 4 Deliverable - Manufacturing Application Document									
Task 4 - Design, Develop and Test Manufacturing Application									
4.1 - Design manufacturing application									
4.2 - Develop manufacturing application									
4.3 - Test manufacturing application									
4.4 - Preliminary software/hardware documentation									
Task 5 Deliverable - Installation Report									
Task 5 - Install Sensors & Software in ACE shop; verify functionality									
5.1 - Install sensors at ACE									
5.2 - Install software									
5.3 - Verify functionality									
Task 6 Deliverable - Beta Test Analysis Report									
Task 6 - System Operation Trial Period									
6.1 - Operate & acquire data from manufacturing software									
6.2 - Validate data. Collect feedback									
6.3 - Analyze data to develop metrics									
6.4 - Develop report of performance metrics									
6.5 - Develop report on management & worker feedback on app.									
Task 7 Deliverable - Product Demonstrations									
Task 7 - Conduct Internal review; Implement Fixes Where Required									
7.1 - Conduct internal review									
7.2 - Define software & hardware fixes required									
7.3 - Define improvements & future work required									
Task 8 Deliverable - Draft Installation Guide									
Task 8 - Accessibility of Manufacturing Software Application									
8.1 - Install software application in GridCell network									
8.2 - Develop installation guide for beta users, DDMI									

Figure 2-- Project Work Schedule

Responsibility for tasks and deliverables were distributed among the development team as follows.

Task 1: Baseline Metric Assessment

Task 1.1	Development of specifications for sensors	Perisense, Georgia Tech, ACE Clearwater
Task 1.2	Development of data collection plan	Perisense
Task 1.3	Achieve consensus on sensor selection and data collection plan	Perisense, Georgia Tech, ACE Clearwater

Task 2: Deliverable – System Requirement Document

Define capabilities for collecting machine data, application internal functions

Task 2.1	Define capabilities for collecting machine data	Perisense
----------	---	-----------

Task 2.2	Test capabilities with project group	Perisense, Georgia Tech, ACE Clearwater
Task 2.3	Define application internal functions	Perisense
Task 2.4	Validate internal functions	Perisense, Georgia Tech, ACE Clearwater
Task 2.5	Refine solutions	Perisense, Georgia Tech, ACE Clearwater

Task 3: Deliverable – Sensor Specification

Select, acquire, and test sensors

Task 3.1	Select sensors	Perisense, Georgia Tech, ACE Clearwater
Task 3.2	Acquire sensors	Perisense
Task 3.3	Test sensors for noise sensitivity	ACE Clearwater
Task 3.4	Harden sensors where required	Perisense, Georgia Tech
Task 3.5	Report on suite of tested sensors	Perisense, Georgia Tech

Task 4: Deliverable – Manufacturing Application Document

Design, develop, and test, manufacturing application

Task 4.1	Design manufacturing application	Perisense
Task 4.2	Develop manufacturing application	MakerSweet, ACE Clearwater
Task 4.3	Test manufacturing application	MakerSweet, ACE Clearwater
Task 4.4	Preliminary software/hardware application	Perisense

Task 5: Deliverable – Installation Report

Install sensors and software at ACE Clearwater; verify functionality

Task 5.1	Install sensors at ACE Clearwater	ACE Clearwater
Task 5.2	Install software	Perisense
Task 5.3	Verify functionality	Perisense, ACE Clearwater

Task 6: Deliverable – Beta Test Analysis Report

System Operation Trial Period

Task 6.1	Operate and acquire data from manufacturing software	ACE Clearwater
Task 6.2	Validate data, collect feedback	MakerSweet, ACE Clearwater
Task 6.3	Analyze data to develop metrics	Perisense
Task 6.4	Develop report of performance metrics	Perisense
Task 6.5	Develop report on operation of the sensors and application	MakerSweet, ACE Clearwater

Task 7: Deliverable – Product Demonstration
Conduct Internal Review, Refine Product

Task 7.1	Conduct internal review	Perisense, Georgia Tech, ACE Clearwater
Task 7.2	Define software and hardware fixes	Perisense, ACE Clearwater
Task 7.3	Define improvements and future work	Perisense

Task 8: Deliverable – Draft Installation Guide
Accessibility of Manufacturing Software Application

Task 8.1	Install software application at DMDII	NCMS, Perisense
Task 8.2	Develop installation guide and final report	NCMS, Perisense

Approach

To minimize development times and costs the project team utilized inexpensive COTS components and open source software. The sensor toolkit had to be affordable and expandable compared to commercially available alternatives.

“...Most modern manufacturing equipment includes process monitoring capabilities including communication technologies such as MTConnect and UPC-UA. However, modern manufacturing equipment is very expensive to purchase. Many companies do not want to make the capital investment and purchase new machines when continual maintenance of their current machines is relatively cheap in comparison. Yet, older equipment often cannot reap the efficiency benefits provided by modern machine monitoring technologies. A need exists to provide a low-cost solution to retrofit these machines with modern machine monitoring capabilities...Numerous external commercial sensors have been developed in the last five years to wirelessly monitor manufacturing operations and determine machine health. The Bosch XDK Cross Domain Development Kit [3] and the Fluke Vibration Sensor [4] are examples of commercial sensor packs that contain accelerometers to measure the vibration characteristics of manufacturing equipment. This data can be used at a base level to determine machine run time, overall usage analytics, and implement basic crash detection techniques. However, the cost of these sensors both exceed \$200 per unit, also resulting in an expensive capital purchase to adequately monitor a handful of machines in a small machine shop...A need exists to provide a low-cost machine monitoring solution that can be used both as a complete product and expandable platform. The desired product can be quickly implemented with base functionality and requires little or no development by the user. Additionally, its capabilities can be expanded if the user has a need for a custom implementation. This low-cost solution must strike a balance between an industry-ready machine monitoring product and a modularly expandable implementation platform.”³

The development team evaluated components and integrated software using the following criteria:

- Ease of use—sensor toolkit had to display data in a simple, easy to understand format so that company executives and machine operators could both understand the data quickly without extensive training. It also had to be easy to install with a minimum of downtime.

³ Saleeby, K. (2019). Development of a low-cost wireless accelerometer platform for machine monitoring applications. Georgia Institute of Technology, Atlanta, Georgia, United States of America. Used with permission.

- Affordability—whenever possible the toolkit would use COTS components and open source software tools.
- Robustness—sensor toolkit had to be capable of operating in an industrial setting with a minimum of maintenance.
- Minimal IT footprint—sensor toolkit must not require extensive configuration the existing IT network.

The project team included the following organizations:

- Perisense—a talented team of IT and manufacturing professionals with a proven track record of creating innovative manufacturing solutions. The company provides valuable information about the performance and productivity of legacy manufacturing assets delivered via low-cost robust sensors that extract previously unobtainable machine data and convert it to actionable intelligence that improves manufacturers' quality and results. Perisense developed the hardware platforms (microprocessor, sensors, case), software to process the captured data, and application to display the results in a web dashboard via computer or mobile device.
- Georgia Institute of Technology (Georgia Tech)—is a top-ranked public college and one of the leading research universities in the United States. Georgia Tech provides a technologically focused education to more than 25,000 undergraduate and graduate students in fields ranging from engineering, computing, and sciences, to business, design, and liberal arts. The Georgia Tech project team includes an internationally renowned expert on manufacturing technology and processes and top-flight graduate students that operates a cluster of high-performance open architecture CNC systems supported by graduate and post-graduate students and faculty with significant experience in machine cell communication, control and architecture design. Georgia Tech helped prototype initial hardware designs, evaluated early sensors, validated software architecture, and provide remote testing of the solution.
- National Center of Manufacturing Sciences (NCMS)—an organization with a long track record of successful commercialization of manufacturing research and development (R&D) projects. NCMS was formed in 1986 to strengthen North American manufacturers and respond to global competition. The balance between long-standing experience and fresh innovation requires a unique intersection of highly capable companies, access to efficient, effective contracting vehicles and relationships built on credibility and trust. NCMS conducted extensive surveys of SMMs to encourage the use of internet of things (IOT)⁴ and digital manufacturing tools and provided project management, coordinated progress reporting with the sponsor, as well as providing access to SMMs to validate potential use cases.
- ACE Clearwater (ACE)—a manufacturer of complex metal forms, components and welded assemblies for the aerospace and power generation industries. ACE is a preferred supplier to major aerospace primes for work on exotic materials and complex assemblies and component. ACE provided a production test environment and feedback.

The team identified the following subcomponents as being necessary for a commercially successful sensor toolkit:

⁴ Internet of things (IoT) is the extension of Internet connectivity into physical devices and everyday objects. Embedded with electronics, Internet connectivity, and other forms of hardware (such as sensors), these devices can communicate and interact with others over the Internet, and they can be remotely monitored and controlled. https://en.wikipedia.org/wiki/Internet_of_things

- Infrastructure—cloud computing resources required to store, process, capture, and visualize data.
- Hardware—CPU board necessary to run the software and manage the sensors collecting machine data.
- Sensors—endpoints collecting various data from the machines on the factory floor.
- Connectivity—physical and networking medium for transmitting the data to cloud services.
- Code—programming code to manage the on-site solution, transmission of the captured data, and configuration of cloud services to receive and visualize data.

Three primary use cases were initially considered for the sensor toolkit:

- Small Machine Shop
- Paper Factory
- Educational Tool Shop

“...Three common uses cases were selected for analysis to determine the ways in which Industry 4.0 technologies could benefit their operations. Our selected use cases were inspired by the needs of three different environments associated with manufacturing processes.

The first use case is a small-scale machine shop, or “Job shop”, where quantities of discrete parts made range from 1 to 10,000. This shop’s capabilities include CNC machining, metal forging equipment, and other equipment for discrete processes. The second use case is paper mill for continuous manufacturing of pulp and paper products. This shop’s capabilities include paper machines with high-viscosity pumps, revolving pulp presses and dryers, a chemical distribution infrastructure. The third use case is an educational machine shop where students learn the fundamentals of prototyping and manufacturing processes. This shop’s capabilities included small hand-held power tools, small raw material processing equipment such as band saws and grinders, manual mills and lathes, and CNC 3-axis mills.”⁵

The sensor toolkit evolved from prototypes created by Georgia Tech and Perisense to find the least expensive, yet capable, and robust solutions to meet the monitoring needs of manufacturers. Prototype sensor kits were tested at several Michigan companies including an SMM and a large OEM factory as well as in the Georgia Tech simulated factory environment.

⁵ Saleeby, K. (2019). Development of a low-cost wireless accelerometer platform for machine monitoring applications. Georgia Institute of Technology, Atlanta, Georgia, United States of America. Used with permission.



Figure 3-- Georgia Tech Demonstration “Fog” Computing Sensor

Georgia Tech also experimented with sensors utilizing the “fog” computing⁶ paradigm where data is collected and partially processed at the collection point before being sent to a cloud location for additional processing and storage.



Figure 4-- Prototype Sensor Kits Used BeagleBone and Arduino Processors

The software design focused code running on a lightweight open source Linux kernel running on the prototype boards (BeagleBone, Arduino, Onion, and ultimately Raspberry Pi) to process the data from the sensors and transmit back to the cloud for storage and processing. Georgia Tech created the initial prototypes before the team settled on the Raspberry Pi as the base CPU unit for the sensor toolkits. These initial prototypes helped validate that machine monitoring and operational visualization was possible using inexpensive retrofit sensor toolkit solutions.

The team tested several different CPUs during the prototype phase. Georgia Tech and Perisense built several sensor test kits based on commercially available hardware. Each prototype test kit performed well in test deployments, successfully capturing and reporting data.

⁶ “Fog computing” is a term created by Cisco that refers to extending cloud computing to the edge of an enterprise's network. Also known as edge computing or fogging, fog computing facilitates the operation of compute, storage, and networking services between end devices and cloud computing data centers. While edge computing is typically referred to the location where services are instantiated, fog computing implies distribution of the communication, computation, and storage resources and services on or close to devices and systems in the control of end-users. Fog computing is a medium weight and intermediate level of computing power. Rather than a substitute, fog computing often serves as a complement to cloud computing. https://en.wikipedia.org/wiki/Fog_computing

CPU Type	Features	Cost
BeagleBone	Single board Linux, Programmable Real-time unit, onboard capacity for data analysis.	\$75
Arduino	Small size, 32-bit ARM, expandable, developer support.	\$25
Onion Omega	Single board Linux, onboard Wi-Fi, expandable, supports many programming language.	\$5
Raspberry Pi	Expandable, large developer community, extensive python, small size and power requirements, product roadmap.	\$40

The team eventually selected the Raspberry Pi as the CPU for development of the sensor toolkits. This choice was made by evaluating CPU performance, commercial availability, cost, and developer support necessary to demonstrate commercial viability of the sensor toolkit.

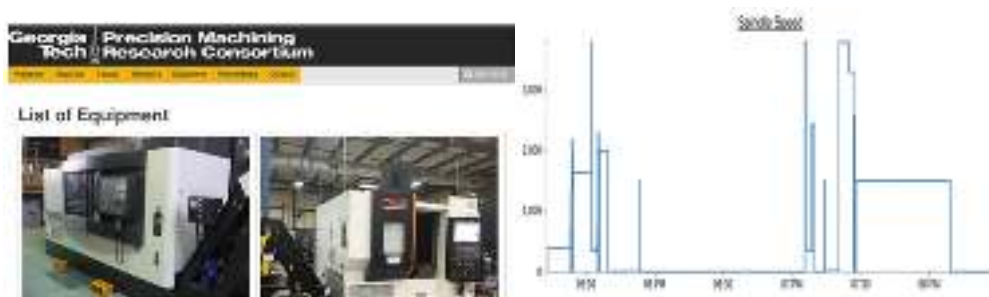


Figure 5-- Georgia Tech Test Equipment and Report

Georgia Tech also was instrumental validating that geographically dispersed sensor kits could simultaneously capture and display data from multiple machines and different locations while transmitting data in the MTConnect format directly to the database. Georgia Tech prototypes used the MQTT⁷ protocol to deliver data via an MQTT broker (client/server) in the MQTT format.

⁷ MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimise network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium. <http://mqtt.org/faq>

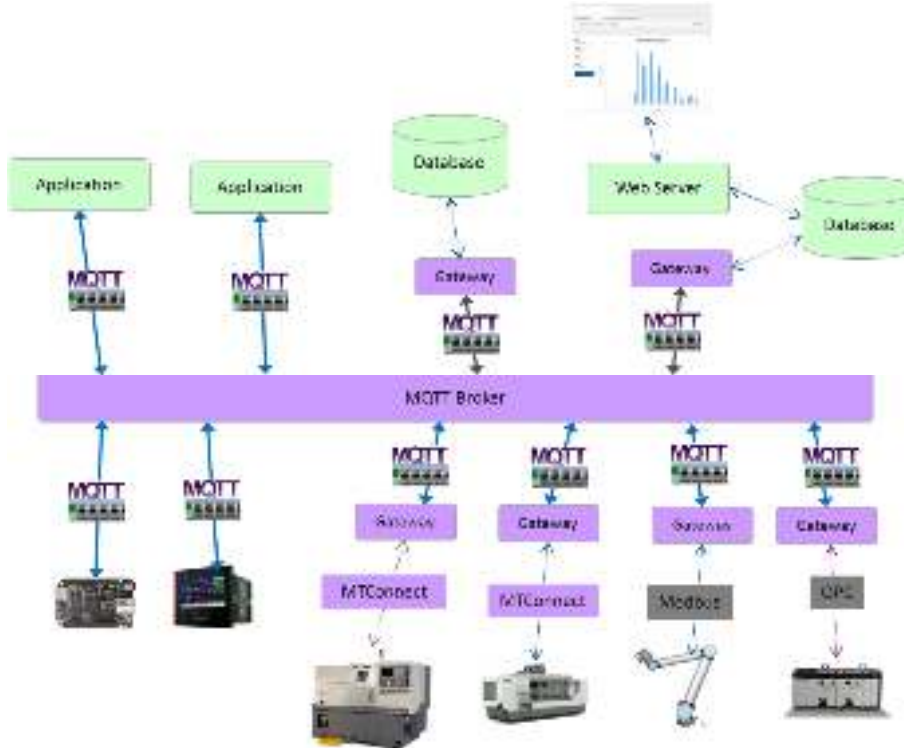


Figure 6-- Georgia Tech MQTT Testing Diagram

The MQTT model identified by Georgia Tech in the early prototype phase was used throughout the project to collect and organize data.

The development team also successfully captured and displayed data via an iPhone app during the early prototype phase.

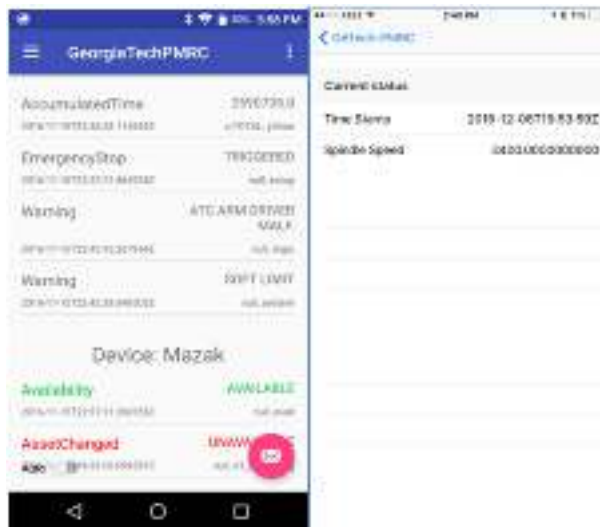


Figure 7-- Prototype iPhone Reporting App

After development of the pre-prototype sensor kits, the development began work a sensor kit that would evolve into a commercial offering.

Welding Application Monitoring

ACE was interested in using the sensor toolkits to monitor its welding applications. The company had several welding stations, some in the open shop and others in closed booths and wanted to track particulates in the air and identify the potential impact on operations.

Obtaining data to assist in defining the acceptable quality and productivity space for specific applications had proven a challenge. ACE deals with a wide range of parts in relatively low volumes, adding to the relative complexity of defining weld quality via process data. They suspected that particulates in the air impacted weld quality and wanted to track the differences between work being done on the shop floor versus closed booths.

ACE joined the project in June 2017. The first air particulate sensor toolkits were implemented around September 2017. The chart below shows the level of air particulates in an enclosed weld booth, open weld booth and in an office environment. The data shows the average in an enclosed weld booth is about 28-30 μA (microgram) of particulates per m^3 of air; open weld booth is about 50-55; office environment (for comparison only) is about 6-8.

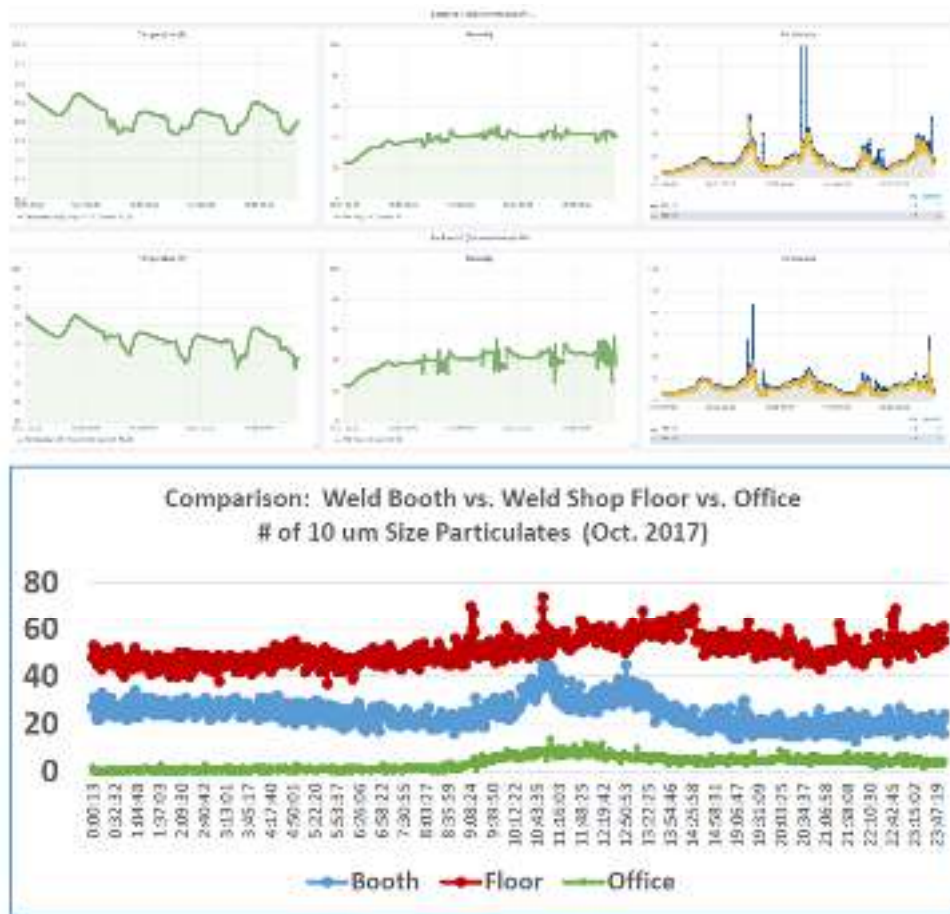


Figure 8-- Sample Reports from ACE Clearwater showing Temperature, Humidity, and Particulates

Over the course of the project, the air particulate monitors were improved to include temperature and humidity, as well as to improve Wi-Fi connectivity. ACE at the time had only weld booths enclosed. Based on the data, ACE enclosed all weld booths by February 2019 to reduce overall air particulates for our welding process.

The approach provided ACE with new visibility into the impact of particulates on welding processes. Approximately 1 MB of data was collected stored on a cloud server hosted on Amazon Web Services (AWS) over the course of 18 months. The data was collected in time-ordered sets that is available to be imported into a conventional ERP system later. The AWS cloud serves as a platform for advanced analytics and business intelligence related for the welding application on an ongoing basis.

The sensor toolkit also provides data and third-party test bed for developing custom applications on the cloud platform, much like the Apple AppStore, Salesforce AppExchange, Oracle Cloud Marketplace, or Siemens Mindsphere. The team opted to rely on an HTML to render dashboards, but the database could also support app development on the iPhone or Android platform.

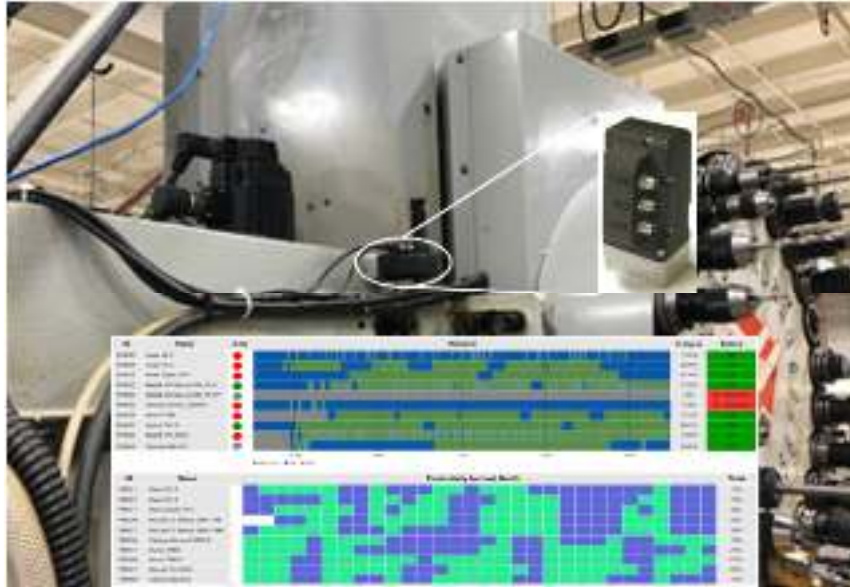


Figure 9-- Example Sensor Toolkit Installation and Report

The sensor toolkit allows for small deployments that can scale up quickly and easily while being affordable for manufacturing companies of all sizes. A SMM can purchase a base product resulting from this project with four sensors including one year of storage and monitoring for \$6,000 (discounts are available for DMDII/MxD members).

Installation by a field engineer takes less than 30 minutes in most cases and depending on the application the machine installation may not require any downtime. This project has resulted in meaningful equipment monitoring solution within the reach of most SMMs.

Industry Outreach

During the development of the sensor toolkits NCMS and Perisense interviewed over two dozen manufacturers were including OEMs and SMMs to identify and validate the following sensor toolkit requirements:

- Capability to measure operator productivity and machine duty cycle.
- Compatibility with older, non-instrumented legacy machine tools.
- Proactive identification of emerging issues with legacy equipment.
- Affordability and resilient solution with minimal downtime for installation.
- Easy to use with minimal training.

The team also had several workshops with Michigan based SMMs who had previously expressed interest in a low-cost solution to specifically monitor power usage.⁸ Participants were engaged in discrete manufacturing and served a number of industrial sectors including aerospace, automotive and medical. Firm sizes ranged from under 10 employees to between 50 and 100. Energy usage was identified as a

⁸ Industrial Scale Demonstration of Smart Manufacturing Achieving Transformational Energy Productivity Gains (DOE/EERE/ Office of Advanced Manufacturing Program (AMO))

major challenge and agreement was broad and unequivocal enough to suggest that energy usage is a universal concern for manufacturers.

These focus groups were composed of people in leadership positions at SMMs who closely monitored business expenses and were greatly impacted by the variable energy charges primarily driven by seasonal changes. They were also aware of overall efficiencies of newer equipment but identified capital costs of upgrading as major deterrent. Energy metrics and dashboard to identify potential savings and verification of savings over the life of equipment were identified as possible drivers for adoption.

The manufacturers surveyed were large OEMs and SMMs including members of the automotive, defense, and aerospace sectors help identify the following sensors as having the broadest initial appeal: accelerometer, particulate, thermometer, and current meter.

Sensor Toolkit	
Sensor	Potential Application
Accelerometer	Lathes, Drills
Particulates	Welding, Additive Manufacturing
Thermometer	Motors, Environmental Conditions
Current Meter	Machine Usage, Energy Conservation

Expandability, Installation, and Reporting

The sensor toolkit can support multiple sensors daisy chained together to gather a variety of data points from a single machine. For example--a sensor kit on a lathe could be installed with an accelerometer to monitor RPMs, a thermometer to track the health of the motor, and current meter to track overall usage of the equipment. An SMM could then determine the correlation of scrap and part defects to the data collected and identify how tool wear impacts energy usage and schedule maintenance accordingly.

Basic installation is done by a Perisense field engineer or company technical staff under their supervision. The sensor is secured via a fastener or magnetic connection depending on the type of equipment and environment.



Figure 10-- Sample Milling Application Showing Temperature, Current, and Vibration Data

The data is displayed with a dashboard that is rendered using HTML and displayed on standard web browsers (Google Chrome, Firefox, or Microsoft Explorer/Edge). The team evaluated creating a dedicated phone app but ultimately decided on the HTML dashboard approach that could detect and customize reports for user on whatever platform they were using (desktop, laptop, or mobile device.)

The reporting dashboard can show data from each sensor or aggregate data from multiple machines, or locations as necessary. Perisense works with each customer to create reports to best meet their monitoring goals. The reporting can also be configured sent alerts on key events and track historical data.

The sensor toolkit was also designed to support multiple custom sensors as identified by end users depending on plant environment and applications. Any COTS industrial grade sensor could be integrated into the toolkit.

Response to Industry Needs

The outreach activities confirmed that SMMs do have strong interest in applications that can improve the productivity and quality of their operations and monitor machine tool energy usage but the solution must be affordable, capable, and compatible with their current equipment.

The sensor toolkit addressed each SMM need as identified from surveys and focus groups.

- Capability--Sensor toolkit is useful for a variety of applications such as stamping, injection molding, welding, and machining to monitor runtime, part count, process repeatability, and maintenance.
- Low acquisition cost--Developed solution has reporting that looks and feels like a simple web page with intuitive consumer-friendly interfaces rather than Oracle or SAP. The application also provides analysis points in the form of actionable intelligence, with affordable sensors being the primary cost of acquisition. The system also incorporates standard report types and infographics that can be easily accessed by users. These easy to grasp reports are also a key selling point during product demonstrations.

- Low operating cost--Large-scale ERP systems require a small army of database administrators to maintain them as well as in-house servers and/or external data centers. The team built a system that uses cloud storage eliminating the need for SMMs to manage a data center or add servers to their existing network. This offers a significant cost savings over existing systems. An expandable cloud-based system enables SMMs to quickly and easily scale up to hundreds or thousands of sensors.
- Flexibility--System reporting allows each individual user and/or department to customize their view of the system depending on their needs. This allows feeds from individuals, machines, and workstations to be aggregated and accessed with flexibility. Individuals can add a machine or work cell to their feed, or remove them as issues emerge, are resolved, and as permitted by system administrators.
- Robustness--Solution is easy to use and has durable sensors and a network infrastructure that operates in harsh factory environments that may include electronic interference and high decibel noise. Components and software have been selected with sophisticated electrical and acoustic noise filtering. The Georgia Tech testbed environment has a numerous machines tools running to support research activities. The selected sensors were developed to operate in an industrial environment. Georgia Tech verified the operation of each sensor prior to deployment at ACE.

The team developed the basic architecture and framework and validated the concept of a peer-to-peer network that allows machines, processes, and people to access data in real-time. The solution supports SMM legacy hardware at moderate cost, while providing real benefits; when combined with the use of data analytics methods to analyze data and aggregate providing business intelligence via intuitive dashboards. The team relied on COTS tools that support current IOT industry standards and protocol.

The team implemented an initial application to monitor gas tungsten arc welding at ACE. It is important to note that while welding is the initial test process, other sensors can be swapped out or daisy chained together to monitor virtually any other legacy machine tool. Stamping presses, welding, machining, and injection molding machines are examples of manufacturing operations that can be instrumented on this platform.



Figure 11-- Products from ACE Clearwater Test Deployment

The sensor toolkit utilizes a scalable, extensible architecture in which sensors are treated as another “peer” on the platform and sensor data can be streamed to other users or an analytics engine. This

peering allows multiple sensors to be daisy chained allowing each sensor toolkit to have multiple sensors onboard all transmitting data for analysis to the cloud.

Once in the cloud this data can be aggregated and analyzed for trends using the data collection and analysis tools from Perisense.



Figure 12-- Example Weekly, Monthly, and Annual Reports

Benefits

This approach realized the following benefits:

- Sensor toolkit is affordable for SMMs who are cost conscious when introducing new products and services. Custom components purchased at the prototype stage, purchased in small quantities would have driven up costs.
- Common inexpensive components which are already in use many other products.
- Standard interfaces allowed for the connection of many possible sensors allowing the solution to be customized for different environments.
- Cloud computing services allowed for quick configuration and development while removing the burden of needing to purchase on-premises computer servers for SMMs to deploy the final product.

The sensor toolkit deployment at ACE Clearwater was used to monitor the impact of particulates but the data could also be used to identify other issues.

For example, a company might discover that:

- Wire feeders are inconsistent, causing variances in bead size or porosity defects—companies have expressed interest in adding a specialized sensor to do this (Perisense is in talks with an OEM to add an additional sensor to track this).
- Some operators in manual welding processes have poor travel speed consistency, which affects weld quality and conformance to heat affect zone size.
- Welding current or voltage may vary more than is tolerable with impacts on porosity, bead geometry, spatter, etc.
- Absent welds in some locations are caused by failure of arc initiation on automated lines.
- It is possible to compare welding stations and note which have significantly higher variations in operating parameters than others, making it possible to identify the root cause of variability and reduce these, thus improving quality through changes to the welding procedures, joint designs, etc.
- Ambient conditions on the plant floor such as temperature and humidity can be monitored over long periods of time and cross-referenced to weld quality, providing users with difficult to track long time frame quality impacts.
- Wide variations in the cycle time of certain machines and operators that can be used to assess the productivity of individual operators or identify inefficiencies in the way a specific welding operation is set up. Some possible examples include:
 - Operators spending less minutes on shift welding than others.
 - Stations with excessive set up times, possibly indicating poor tooling design or a need to tighten tolerances on incoming part geometry.
 - Evaluating waste and attempt to address resource management.
 - OEMs analyzing weld operation data from multiple suppliers in order to develop vendor scorecards and improve incoming materials quality.
 - Stations with excessive set up times, possibly indicating poor tooling design or a need to tighten tolerances on incoming part geometry.
 - Evaluating waste and attempt to address resource management.
 - OEMs analyzing weld operation data from multiple suppliers in order to develop vendor scorecards and improve incoming materials quality.

Most importantly, this approach when coupled with the easy-to-grasp dashboard makes data available at all worker levels and time frames desired by the company to improve quality and worker satisfaction, while providing management with the key insights and actionable intelligence needed to enhance business results.

The completed sensor toolkit is now a commercially available product that includes the real-time application, sensor installation, and basic analytics. Future developments will include the addition of new processes, analytics improvements, a wider selection of data gathering options, more granular capabilities for information sharing, and integration with third-party systems. The sensor toolkit will also include a cellular data transmission option to allow deployments where no guest networks exist or direct network connections are impractical.

III. KPI'S & METRICS

The sensor toolkit can monitor various key improvement indicators (KPIs) related to legacy machine tool equipment.

Improve Welding and Joining Processes--In the context of a welding operation, data can be gathered on a range of operating parameters with known impact on quality and productivity including travel speed, wire feed rate, gas flow rate, welding current and voltage. Some welding systems available in the market today have built-in sensors to provide this data, and retrofitting is also available. However, none of the providers of OEM systems and sensors, or retro-fit welding sensors, provide a broadly-connected, real-time system that can be used on a multi-level basis to allow operators to analyze system performance in real-time for monitoring and troubleshooting; neither do they aggregate data as reports to management on a scheduled basis for operational analysis.

Analyze Worker Reporting--Workers on the factory floor often detect problems visually, audibly, or otherwise, but their descriptions aren't considered "data" and are often undervalued or ignored. Text analysis of a worker's log might reveal comments about vibration, stalls, or events that sensors may not even detect. Textual analysis has become increasingly sophisticated in the last few years and that trend will continue. Computers will become more proficient at scanning a piece of text (or voice converted to text) for themes and repetition; essentially, text and voice can thus be classified and analyzed in the same way as structured data.

Historical Data--Sensor toolkit aggregates historical process data that can then be analyzed to identify patterns and relationships among discrete process steps and inputs. At this point, data will be used to then optimize factors that have been proven to create desired effects or outcomes.

Analytics to Improve National Competitiveness--Data collection and analysis can also be the ultimate in pre-competitive collaboration between enterprises. Sensor toolkits can aggregate data from multiple manufacturers to create a diverse, anonymous, standardized data set. Each manufacturer can use this information to evaluate their operation against peak performance data for given metrics. Such a system could be a powerful engine for U.S. national industrial competitiveness.

Welding Application- Ace Clearwater

The sensor toolkits were deployed at ACE for 18 months and helped them identify when conditions of high particulates were in the air during welding operations. When these conditions were identified by the sensor toolkit, supervisors would be alerted to intervene on the shop floor to rectify the situation.

ACE focused on welds on flight critical repair components where high particulate counts could adversely impact the quality of the work.



Figure 13-- Welding report showing temperature, humidity, and particulates

During testing Perisense identified the following KPIs that sensor toolkits could track for welding customers.

Metric	Present State (Baseline)	Future State (Project Goal)
Duty Cycle (financial)	Yes	Welder time use, productivity, and energy consumption data.
Part to Part Process Consistency (technical)	Yes	Part to part consistency within a batch, ability to match with visual and post-NDI results.
Impact of Humidity and Temperature on Productivity and Rejection Rate (technical)	Yes	Demonstrate ability to monitor temperature and humidity; relate to worker duty cycle, part production rate, and number of repairs/rejects.

Potential Business Outcomes

The following potential business outcomes have been identified:

- Users targeting production bottlenecks by quantifying equipment usage.
- Discovery of equipment performing below needed specifications.
- Reduce scrap and re-work by identifying optimal environmental conditions.
- Scheduling of preventive maintenance based on CBM rather than TBM principles.

Business Cases

- Large OEM, Tier 1 or Tier 2:

Large OEMs can deploy sensor toolkits to their existing plant floor machinery installation to verify data gathered from other sources or add functionality to legacy equipment. The sensor toolkit could also be useful to monitor smaller remote facilities that have disparate equipment perhaps acquired through a merger or acquisition.

- SMM Use Case:

Smaller Tier 2 or 3--Manufacturing engineers at a small shops can buy a low cost, retrofit sensor package and data analytics to provide visibility into operations, obtain quality alerts on the floor, measure OEE, and identify key areas in the shop that require improvements. New equipment using MT Connect⁹ could provide manufacturers with similar data but is cost prohibitive for many manufacturers.

The sensor toolkit offers an affordable, impactful alternative to purchasing a new CNC machine, "...In many job shops, producing profit in a short-term time frame can be exceedingly difficult when common CNC machines of quality for small parts range from \$30,000 - \$500,000 in price."¹⁰

IV. TECHNOLOGY OUTCOMES

System Overview

The sensor toolkit addresses key shortcoming suffered by SMMs--affordable visibility into the quality, safety, and efficiency of their operations including machines, parts, processes, and people. It provides data to reduce work cycle costs and improve quality for SMMs that have unconnected legacy machines and that is easily retrofitted with plug-and-play sensors to provide easy to understand dashboards via desktop and laptop computers or mobile devices.

The sensor toolkit solution utilizes cloud computing technology to provide SMMs with the analysis tools to integrate information in intuitive ways that empower both manufacturing management and workers to address quality assurance and decision-making. This solution also frees SMMs from the need to have onsite computer servers and complicated IT infrastructure while being capable of easily scaling up to meet demand.

The project resulted in the following sensor toolkit solution.

⁹ MTConnect is a [standard for formatting and transmitting data](#) produced by manufacturing equipment so that it can be processed by software applications. Common uses for this technology are real-time machine monitoring, and historical reporting using charts, graphs, and other visual representations of a shop's performance. <https://www.shopfloorautomations.com/what-does-mtconnect-cost/>

¹⁰ Saleeby, K. (2019). Development of a low-cost wireless accelerometer platform for machine monitoring applications. Georgia Institute of Technology, Atlanta, Georgia, United States of America. Used with permission.

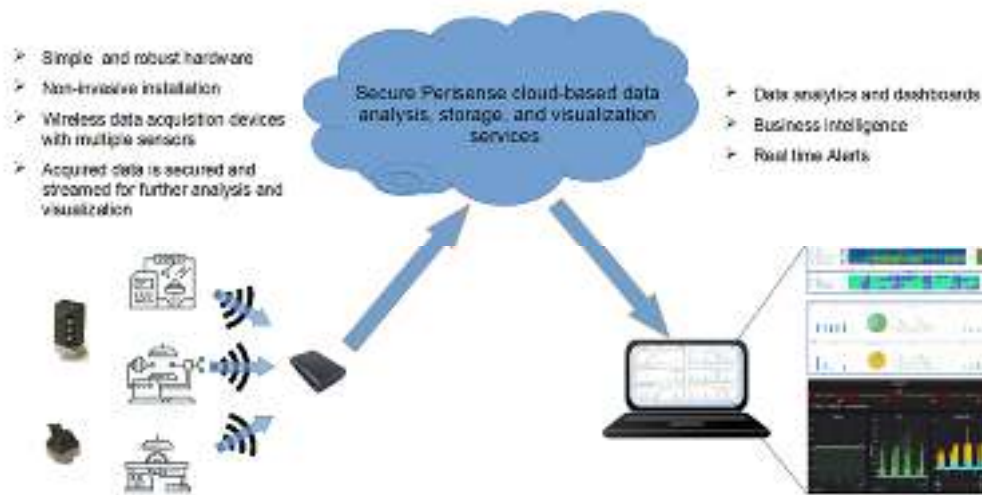


Figure 14-- Sensor Toolkit Overview

System Requirements

The sensor toolkit has following requirements for deployment:

- Legacy factory equipment such as a machine tool or welding apparatus.
- Local network connected to the internet or access to data cellular network.
- Computers or mobile devices to view the data reports.
- Access to the equipment for approximately 30 minutes for installation (not all installations require machine downtime.)

The following sensors are currently available and can be linked together to run simultaneously. Custom sensors can also be integrated on a case-by-case basis.

Sensor ToolKit	
Sensor	Potential Application
Accelerometer	Lathes, Drills
Particulates	Welding, Additive Manufacturing
Thermometer	Motors, Environmental Conditions
Current Meter	Machine Usage, Energy Conservation

DMDII/MxD members can purchase a discounted base product package from Perisense including four sensors including one year of storage.

System Architecture

The system architecture is comprised of the following subcomponents:

- Infrastructure—Cloud computing resources necessary to store, process, and visualize data.
- Base Hardware CPU—Processor to run the software and connect the sensors collecting data.
- Sensors—Endpoints collecting data from the machines on the factory floor.
- Connectivity—Physical and wireless channels for transmitting the data to cloud services.
Perisense designed a wireless hub to support multiple sensors and ease deployment utilizing cellular data networks.

Features & Attributes

Once the Raspberry Pi architecture was selected as the base hardware CPU the sensor toolkit went through three revisions.

Infrastructure	Base Hardware CPU	Sensor	Connectivity
Amazon Web Services (AWS) – Services used for device fleet management, data storage, processing, analysis and visualization of the data	Raspberry Pi3B Raspberry PI0W	Accelerometer (ADXL345), Particulate (SDS011), Thermometer (TMP006, DS18B20), Current meter (SCT013). Provisioned for connection of various other sensors	Wireless connectivity provided by special Perisense developed router (serving all the devices within the area of 10,000 square feet)

This configuration represented the final baseline configuration of the product (version 2) tested by project team. Version 2 of the solution is a major upgrade from version 1 with additional reporting capability, analysis and visualization provided by AWS. The wireless (Wi-Fi) capability was further enhanced adding a using a custom wireless router developed by Perisense engineers.

Infrastructure--AWS is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. It provides complete control of computing resources on Amazon’s proven computing environment. AWS reduces the time required to obtain and boot new server instances to minutes, allowing quick capacity changed, both up and down, as computing requirements change. AWS enables users to pay only for capacity that is actually needed.

Using AWS provided the product team with the flexibility to offer an affordable solution for smaller SMMs while providing the scalability to server larger customers as well.

Base Hardware CPU--Base computing hardware continued to use the same versions of Raspberry Pi from the intermediate version of the product with the following specifications:

Raspberry Pi 3 Model B--Earliest model of the third-generation Raspberry Pi replacing the Raspberry Pi 2 Model B in February 2016:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM

- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- Upgraded switched Micro USB power source up to 2.5A

Raspberry Pi Zero W--Launched at the end of February 2017, the Pi Zero W has the functionality of the original Pi Zero, but with added connectivity capabilities:

- 1GHz, single-core CPU
- 512MB RAM
- Mini HDMI and USB On-The-Go ports
- Micro USB power
- HAT-compatible 40-pin header
- Composite video and reset headers
- CSI camera connector
- 802.11 b/g/n wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

Sensors—Endpoints for data collection with the following capabilities:

Accelerometer (ADXL345) is a small, thin, ultra low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ± 16 g. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I2C digital interface. The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 mg/LSB) enables measurement of inclination changes less than 1.0° . Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion by comparing the acceleration on any axis with user-set thresholds. Tap sensing detects single and double taps in any direction. Freefall sensing detects if the device is falling. These functions can be mapped individually to either of two interrupt output pins. An integrated memory management system with a 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor activity and lower overall system power consumption. Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation. The ADXL345 is supplied in a small, thin, 3 mm \times 5 mm \times 1 mm, 14-lead, plastic package.

Accelerometer Features:

- Ultra low power: as low as 23 μ A in measurement mode and μ A in standby mode at $V_S = 2.5$ V
- Power consumption scales automatically with bandwidth
- User-selectable resolution

- Fixed 10-bit resolution
- Full resolution, where resolution increases with g range, up to 13-bit resolution at ± 16 g (maintaining 4 mg/LSB scale factor in all g ranges)
- Embedded memory management system with FIFO technology minimizes host processor load
- Single tap/double tap detection
- Activity/inactivity monitoring
- Free-fall detection
- Supply voltage range: 2.0 V to 3.6 V
- I/O voltage range: 1.7 V to VS
- SPI (3- and 4-wire) and I2C digital interfaces
- Flexible interrupt modes mappable to either interrupt pin
- Measurement ranges selectable via serial command
- Bandwidth selectable via serial command
- Wide temperature range (-40°C to $+85^{\circ}\text{C}$)
- 10,000 g shock survival
- Pb free/RoHS compliant
- Small and thin: 3 mm \times 5 mm \times 1 mm LGA package

Particulate Sensor (SDS011) uses the principle of laser scattering to measure the particle concentration between 0.3 to 10 μm . It provides digital output and has a built-in fan to enhance reliability.

Characteristics:

1. Accurate and reliable: laser detection, stable, good consistency.
2. Quick response: response time is less than 10 seconds when the scene changes.
3. Easy integration: UART output (or IO output can be customized), fan built-in.
4. High resolution: resolution of 0.3 $\mu\text{g}/\text{m}^3$; Nova Fitness Co., Ltd. SDS011 sensor 2.
5. Certification: products have passed CE/FCC/RoHS certification.

Working principle: Using laser scattering principle. Light scattering can be induced when particles go through the detecting area. The scattered light is transformed into electrical signals and these signals will be amplified and processed. The number and diameter of particles can be obtained by analysis because the signal waveform has certain relations with the particle's diameter.

Infrared Thermopile Sensor (TMP006) is temperature sensor that measures the temperature of an object without the need to make contact the object. This sensor uses a thermopile to absorb the infrared energy emitted from the object being measured and uses the corresponding change in thermopile voltage to determine the object temperature.

Infrared sensor voltage range is specified from -40°C to $+125^{\circ}\text{C}$ to enable use in a wide range of applications. Low power consumption along with low operating voltage makes the device suitable for battery-powered applications. The low package height of the chip-scale format enables standard high-volume assembly methods, and can be useful where limited spacing to the object being measured is available.

Sensor Features:

- Complete Solution in 1,6-mm × 1,6-mm Wafer Chip-Scale Package (WCSP) Device (DSBGA)
- Digital Output:
 - Sensor Voltage: 7 $\mu\text{V}/^\circ\text{C}$
 - Local Temperature: -40°C to $+125^\circ\text{C}$
- SMBus™-Compatible Interface: at 3.3 V
- Pin-Programmable Interface Addressing
- Low Supply Current: 240 μA
- Low Minimum Supply Voltage: 2.2 V

Thermometer Sensor (DS18B20) is a digital thermometer that provides 9-bit to 12-bit Celsius temperature measurements and has an alarm function with nonvolatile user-programmable upper and lower trigger points. The DS18B20 communicates over a 1-Wire bus that by definition requires only one data line (and ground) for communication with a central microprocessor. It has an operating temperature range of -55°C to $+125^\circ\text{C}$ and is accurate to $\pm 0.5^\circ$ over the range of -10°C to $+85^\circ\text{C}$. In addition, the DS18B20 can derive power directly from the data line (“parasite power”), eliminating the need for an external power supply.

Each DS18B20 has a unique 64-bit serial code, which allows multiple DS18B20s to function on the same 1-Wire bus. Thus, it is simple to use one microprocessor to control many DS18B20s distributed over a large area. Applications that can benefit from this feature include HVAC environmental controls, temperature monitoring systems inside buildings, equipment, or machinery, and process monitoring and control systems.

Features:

- Unique 1-Wire® Interface Requires Only One Port Pin for Communication
- Each Device has a Unique 64-Bit Serial Code Stored in an On-Board ROM
- Multidrop Capability Simplifies Distributed Temperature-Sensing Applications
- Requires No External Components
- Can Be Powered from Data Line; Power Supply Range is 3.0V to 5.5V
- Measures Temperatures from -55°C to $+125^\circ\text{C}$ (-67°F to $+257^\circ\text{F}$)
- $\pm 0.5^\circ\text{C}$ Accuracy from -10°C to $+85^\circ\text{C}$
- Thermometer Resolution is User Selectable from 9 to 12 Bits
- Converts Temperature to 12-Bit Digital Word in 750ms (Max)
- User-Definable Nonvolatile (NV) Alarm Settings
- Alarm Search Command Identifies and Addresses Devices Whose Temperature is Outside Programmed Limits (Temperature Alarm Condition)
- Available in 8-Pin SO (150 mils), 8-Pin μSOP , and 3-Pin TO-92 Packages
- Software Compatible with the DS1822
- Applications Include Thermostatic Controls, Industrial Systems, Consumer Products, Thermometers, or Any Thermally Sensitive System

Current Meter (SCT013) is a split core current transformer to measure electrical current.

These sensors were chosen for its affordability and recommended use for industrial instrumentation applications.

Characteristics:

- Opening size: 13mm*13mm,
 - Non-linearity $\pm 3\%$ (10%-120% of rated input current)
 - 1m leading wire, standard $\Phi 3.5$ three core plug output.
 - Current output type and voltage output type (voltage output type built-in sampling resistor)
- Purpose: Used for current measurement, monitor and protection for AC motor, lighting equipment, air compressor etc.
- Core material: Ferrite
- Mechanical strength: Number of switching is not less than 1000 times (test at 25°C)
- Safety index: Dielectric strength (between shell and output) 1000V AC/1min
 - Fire resistance property: In accordance with UL94-Vo
 - Work temperature: -25°C~+70°C

The product also has been provisioned to allow the connection of many other sensors via a daisy chain¹¹ connection.

Connectivity--Network connectivity is provided by wireless capability (Wi-Fi) integrated into the base Raspberry Pi hardware. An important addition to the baseline capability.

¹¹ In electrical and electronic engineering, a daisy chain is a wiring scheme in which multiple devices are wired together in sequence. Daisy chains may be used for power, analog signals, digital data, or a combination thereof. [https://en.wikipedia.org/wiki/Daisy_chain_\(electrical_engineering\)](https://en.wikipedia.org/wiki/Daisy_chain_(electrical_engineering))

Business Requirements

Infrastructure	Base Hardware CPU	Sensors	Connectivity
Amazon Web Services (AWS) – limited to SQL database used for storage and elastic cloud computer services	Raspberry Pi3B Raspberry PI0W	Accelerometer (ADXL345), Particulate (SDS011), Thermometer (TMP006, DS18B20), Current meter (SCT013). Provisioned for connection of various other sensors	Wireless connectivity provided by special Perisense developed router (serving all the devices within the area of 10000 square feet)
Meets Requirements	Meets Requirements	Meets Requirements	Meets Requirements

The final sensor kit hardware (version 2) is a complete solution and which meets every business requirement. AWS supports all required applications and also allows virtual environments to be spun up to support installations of any size. It also has virtually 100% uptime due to its distributed architecture. This version still supports Wi-Fi but also utilizes a custom wireless cellular hub that makes installation at any location possible with no IT intervention. The Raspberry PI and sensors provide the processing power necessary to support the open source python applications.

Version Comparison

	Infrastructure	Base Hardware CPU	Sensors	Connectivity
Prototype "Version 0"	Plotly – only data visualization with very limited data storage ability	Raspberry Pi2	Accelerometer (ADXL345), Particulate (PPD42NS)	Wired, requires separate internet drop per device
Intermediate "Version 1"	Amazon Web Services – limited to SQL database– used for storage and elastic cloud computer server used for visualization	Raspberry Pi3B Raspberry Pi0W	Accelerometer (ADXL345), Particulate (PMS1003), Thermometer (HTU21D5)	Wireless, connected to a wired internet bridge
Final "Version 2"	Amazon Web Services – different services used for device fleet management, data storage, processing, analysis and visualization of the data	Raspberry Pi3B Raspberry Pi0W	Accelerometer (ADXL345), Particulate (SDS011), Thermometer (TMP006, DS18B20), Current meter (SCT013), Provisioned for connection of various other sensors	Wireless connectivity provided by special Pensece developed router (serving all the devices within the area of 10000 square feet)

Figure 15-- Product Configurations as Tested

The team also created multiple pre-prototypes before deciding on the Raspberry Pi processor. Georgia Tech developed early sensor kits based on the BeagleBone as well as the software code to test the configurations.

BeagleBone is a barebone development board with a Sitara ARM Cortex-A8 processor running at 720 MHz, 256 MB of RAM, two 46-pin expansion connectors, on-chip Ethernet, a microSD slot, and a USB host port and multipurpose device port which includes low-level serial control and JTAG hardware debug connections.

BeagleBone Black is a newer version of the BeagleBone that increases RAM to 512 MB, the processor clock to 1 GHz, and it adds HDMI and 2 GB of eMMC flash memory. The BeagleBone Black also ships with Linux kernel 3.8, upgraded from the original BeagleBone's Linux kernel 3.2, allowing the BeagleBone Black to take advantage of Direct Rendering Manager (DRM).

The final decision to go with Raspberry Pi for the project was driven by cost and capability. Raspberry Pi was less expensive and had all the needed features onboard without the need to add additional board components.

Future development plan for mass production will use a custom board.

Software Development Documentation/Design Document

The software design goal was to provide simplest and most effective connection between the hardware sensors and end user. The project team chose the Python programming language, an open source with a massive programming development community to meet this challenge. The sensor toolkit utilizes individual sensor drivers provided by sensor suppliers. The data acquisition software acquires sensor data which is then transmitted to AWS for analysis and visualization.

AWS routes the data to a relational database for visualization utilizing open source third party software Grafana (<https://grafana.com/>). Grafana provides an interface allowing to build customer facing dashboard by using various built-in graphical tools. A typical customer facing Grafana dashboard will include several panels such as time-based plots, on/off indicator buttons, gauge indicators, etc. Perisense used standard SQL database queries to populate the dashboard.



Figure 16-- Equipment Temperature, Current, and Accelerometer Data

Users & Use Cases

Project Participant-- ACE Clearwater

ACE manufactures complex metal forms, components and welded assemblies for the aerospace and power generation industries specializing in exotic materials, complex assemblies, and components.

ACE also has additional facilities with the following capabilities:

- One of the largest drop hammer operations in the country, model shop, foundry, spinning, planishing, check & straighten, hand-trimming, shear, and deburr. It also benefits from a complete closed-loop, organic clean line operation and quality team for final inspections, in-house, and adds an additional 250,000 square feet of manufacturing.
- Hydroforming, with a 400-ton hydraulic press, machine shop and 5-axis laser cutting equipment. It has also been, for four years, the site for ACE Manufacturing Day celebrations for hundreds of local school children to learn what about aerospace manufacturing.

ACE used the sensor toolkit to monitor temperature, humidity, atmospheric particulates, and weld current during welding of flight critical replacement parts for the Department of Defense (DOD).

ACE joined the project in June 2017. The first air particulate sensor toolkits were implemented around September 2017. The chart below shows the level of air particulates in an enclosed weld booth, open weld booth and in an office environment. The data shows the average in an enclosed weld booth is about 28-30 ug (microgram) of particulates per m³ of air; open weld booth is about 50-55; office environment (for comparison only) is about 6-8.

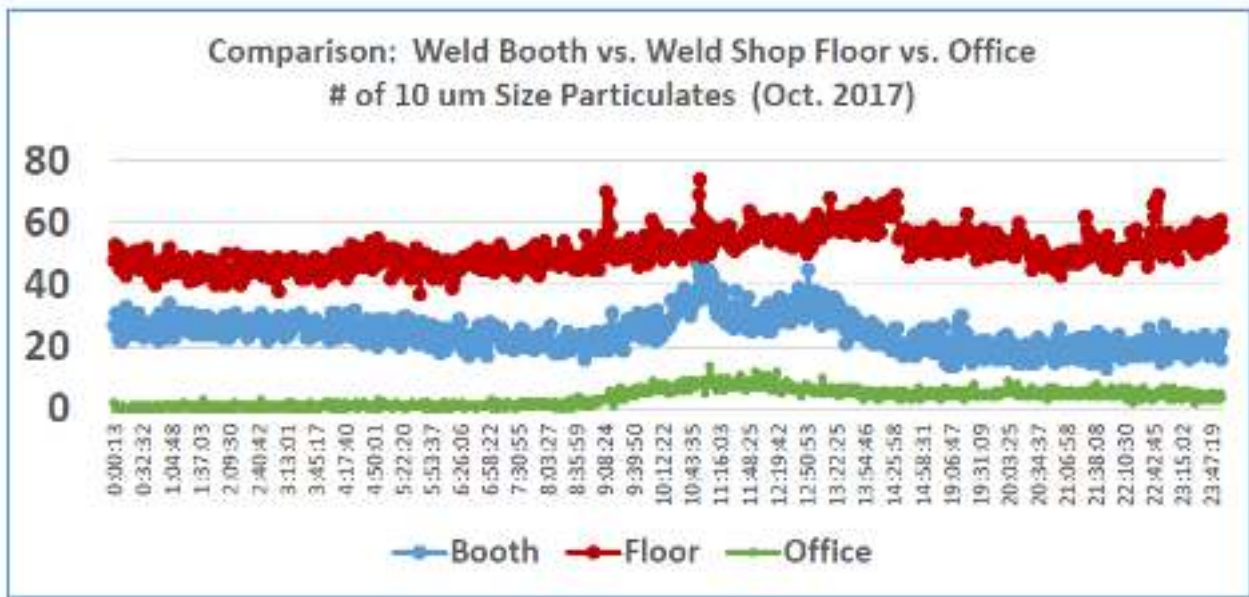


Figure 17--- ACE Particulate Reports

Over the course of the project, the air particulate monitors were improved to include temperature and humidity, as well as to improve Wi-Fi connectivity. ACE at the time had only two weld booths enclosed. Based on the data, ACE enclosed all of its weld booths by February 2019 to reduce overall air particulates for welding processes. This has reduced our air particulate levels to around 9-11 (see current charts below, averages are circled red).



Figure 18-- Welding report showing temperature, humidity, and particulates

Use Cases

Large OEM, Tier 1 or Tier 2:

Large OEMs can purchase a product package from Perisense including ten sensors including one year of storage and monitoring for \$15,000 (DMDII/MxD member price \$14,000) plus travel costs for installation by Perisense engineers. The customer can choose from among the following sensors: accelerometer, particulates, thermometer and current meter. Sensors are covered under warranty for one year and maintenance contracts for subsequent years are available. After the first-year storage and monitoring is \$1 a day per sensor. Custom sensors can be integrated for an additional cost.

Potential OEM deployment include installation on similar equipment (a cell of drill presses or lathes) to compare performance and power usage as well as monitoring pieces of equipment both before and after maintenance.

OEMs can also purchase the solution for installation downstream in their supply chain to monitor partner equipment at remote locations.

SMM Use case:

Smaller Tier 2 or 3--Example: As the manufacturing engineering and IT manager at a small maker of precision equipment used in OEM plants, I can buy a low cost, retrofit sensor package and data analytics to provide visibility into my operations, obtain quality alerts on the floor, measure OEE, and identify key areas in the shop that require improvements.

An SMM can purchase a base product package from Perisense including four sensors including one year of storage and monitoring for \$6,000 (DMDII/MxD member price \$5,500) plus travel costs for installation by Perisense engineers. The customer can choose from among the following sensors: accelerometer, particulates, thermometer and current meter. Sensors are covered under warranty for one year and maintenance contracts for subsequent years are available. After the first-year storage and monitoring is \$1 a day per sensor. Custom sensors can be integrated for an additional cost.

The base product package would allow an SMM to monitor the performance of a milling machine (accelerometer), the utilization rate of a CNC machine (current meter), and identify potential equipment failures (thermometer). A spike in motor temperature can indicate a bad bearing that needs to be repaired.

Digital Manufacturing and Design Innovation Institute (DMDII)/MXD

The Digital Manufacturing and Design Innovation Institute (DMDII) is where innovative manufacturers go to forge their futures. In partnership with UI LABS and the DOD, DMDII equips U.S. factories with the digital tools and expertise they need to begin building every part better than the last. As a result, our more than 300 partners increase their productivity and win more business.



Figure 19-- Solution Deployed at DMDII/MXD

DMDII has invested approximately \$90 million in more than 60 applied R&D projects in areas including design; product development; systems engineering; future factories; agile, resilient supply chains; and cybersecurity.

The Institute operates from a nearly 100,000-square-foot innovation center near downtown Chicago. Its factory floor features some of the most advanced manufacturing equipment in the world, which partners can use for experimentation and training on everything from augmented reality to advanced simulation techniques.

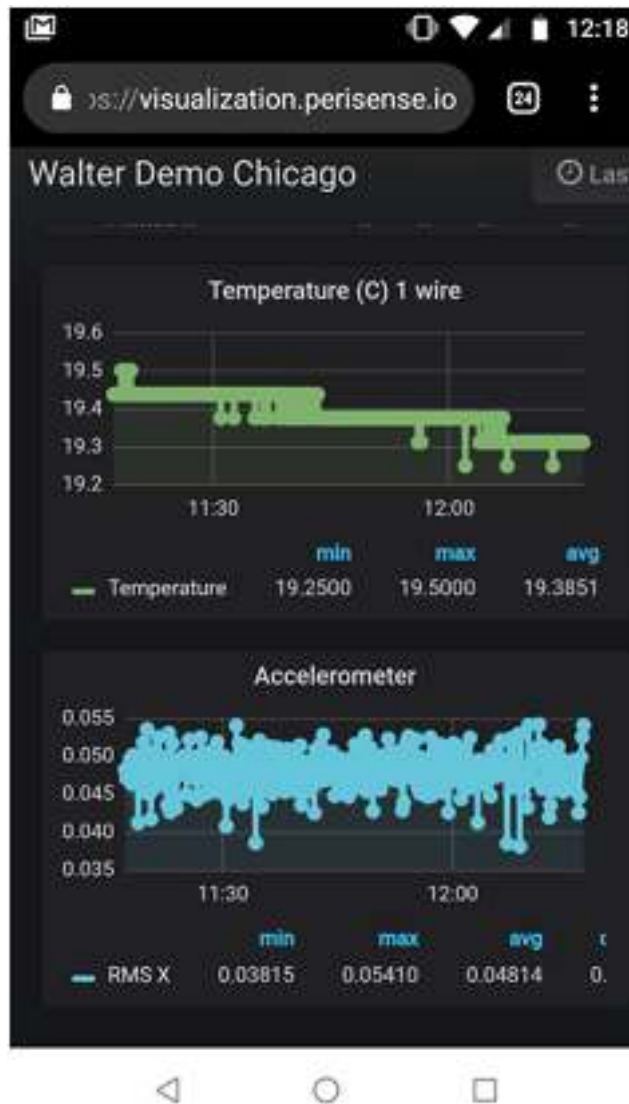


Figure 20-- Mobile Data reporting from DMDII Demo

DMDII’s vision is to increase U.S. manufacturing productivity by making “every part better than the last.”

In February of 2019, DMDII was renamed MxD, which stands for manufacturing times digital. MxD’s mission is to drive the digital future of manufacturing, pioneering new technologies that make America’s industrial base and warfighters more resilient and agile

V. ACCESSING THE TECHNOLOGY

Background Intellectual Property

The team is claiming no background intellectual property on this project.

Technical and Systems Requirements

The following requirements are necessary to deploy the baseline solution:

- Factory equipment such as a machine tool or welding apparatus.
- Local network connected to the internet or active Wi-Fi or cellular network.
- Computers or mobile devices to view the data reports.

DMDII/MxD members can purchase a base product package from Perisense including four sensors including one year of storage and monitoring for \$5,500 plus travel costs for installation by Perisense engineers. The customer can choose from among the following sensors: accelerometer, particulates, thermometer and current meter. Sensors are covered under warranty for one year and maintenance contracts for subsequent years are available. After the initial year storage and monitoring costs \$1 a day per sensor. Custom sensors can be integrated for an additional cost.

VI. INDUSTRY IMPACT & POTENTIAL

Potential Impact

According to statistics from the U.S. Census¹² bureau there are over quarter of a million manufacturing firms in the United States, and 94% employ less than a hundred workers.

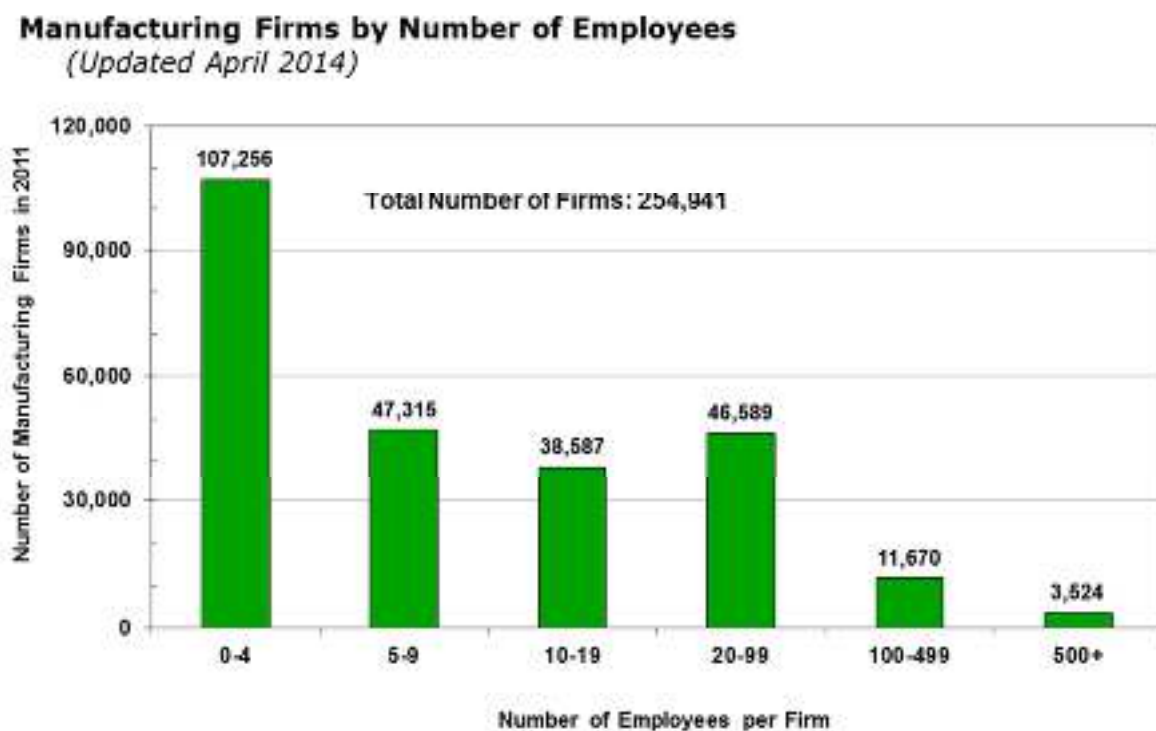


Figure 21— U.S. Census Bureau and MAPI Manufacturing Employment Data

Many of these companies are potential users of the product resulting from this project. While the initial sensor toolkit includes just four sensors it can support hundreds of additional commercially available sensors.

¹² <http://www.themanufacturinginstitute.org/Research/Facts-About-Manufacturing/Economy-and-Jobs/Company-Size/Company-Size.aspx>

Other Industry Potential

The product resulting from this project could be further enhanced by analyzing the results with machine learning capability.¹³

- Initial exploration of machine learning via retrofit data--data quantity expanded over time
- Very limited effort but demonstrated that data quality is sufficient to allow ML methods to be applied successfully.
 - Used to demonstrate part discrimination.

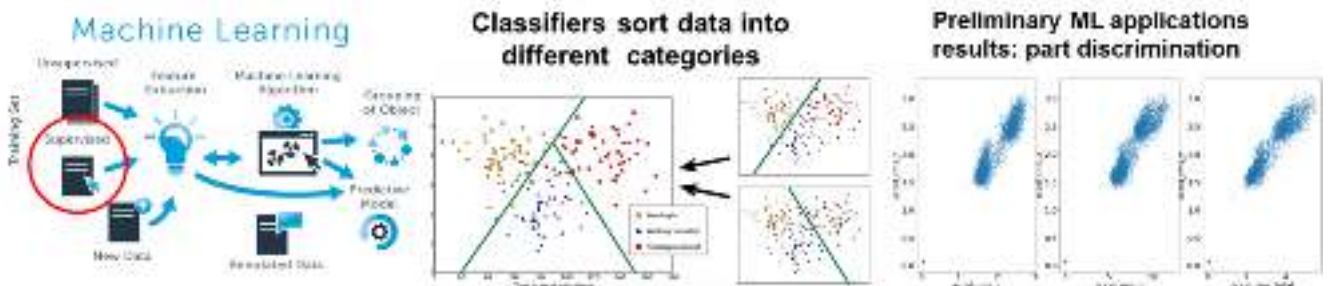


Figure 22— Potential Machine Learning Applications for Sensor Toolkit Data

Expanding Sensor “Library” and Applications

- The flexibility of the node based, multiple port design allows multiple measurements to be made, and customized to various processes at low cost.
- Identifying and prioritizing measurement types and applications benefit suppliers and users
 - Via DMDII/MxD and MEP investigating the possibility of working with a company on a custom application.
- Good case study for value creation due to low cost of implementing solutions for low volume processes.

Another application identified by user groups is the ability to monitor power usage of equipment. SMMs are particularly sensitive to the variable costs of energy usage and can use this toolkit to monitor and adjust behavior.

¹³ Machine Learning (ML) is the scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task. Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task. Machine learning algorithms are used in the applications of email filtering, detection of network intruders, and computer vision, where it is infeasible to develop an algorithm of specific instructions for performing the task. Machine learning is closely related to computational statistics, which focuses on making predictions using computers.
https://en.wikipedia.org/wiki/Machine_learning

Necessary Steps for Cross Industry Migration

Next steps for possible uses in other environments and industries were identified:

- As a development program, the sensor toolkits are not designed for permanent installation in manufacturing settings that require the highest levels of safety standards due to excessive heat or chemicals.
- Customer discovery conversations have indicated that packaging, particularly meeting UL and Class 1, Division 1 explosion safety is an important consideration.

These topics receive relatively little attention in development activities, but should be considered early in the design stage, and may be relevant to other efforts underway at DMDII/MxD.

VII. TECH TRANSITION PLAN & COMMERCIALIZATION

Future Plans

Next steps towards mass production:

- Tooling for faster production of robust enclosures—current enclosure are 3D printed as needed. This process is slow and would not support mass production.
- Development kits—there has been interest in a development kit which would expand the number of sensors types for test deployments.
- Custom system boards—Raspberry Pi boards while being very capable are not easily sourced for mass purchase.
- Packaging—for shipping and to easily identify what type of sensor is being deployed and to help track inventory. Current packaging is very basic (a cardboard box with stickers). As production is scaled up newer package with SKU to identify different configurations would be beneficial.

Some users have expressed interest in having their data stored in other hosting environments:

- Microsoft Azure
- Mindsphere
- Rackspace Cloud
- IBM Cloud
- Google Cloud

The solution itself while currently being hosted on AWS can be ported to virtually any hosting environment. The team is evaluating which clouds offer the best marketing opportunities to drive further adoption.

The team is also considering options for customers who don't want their data stored in any cloud and want local server options within their enterprise.

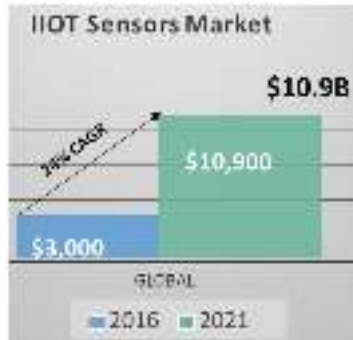
Technology Transition Plan

Perisense, Inc. www.perisense.io, is a Delaware C Corporation based in Ann Arbor, MI developing hybrid software/hardware products for the Industrial Internet of Things (IIOT). Huge changes in manufacturing, enterprise software, machine monitoring and data analysis, known as “Manufacturing 4.0” are drastically increasing the amount of data manufacturers needed to remain competitive. Yet, per the Society of Manufacturing Engineers (SME), half or more of older “legacy” machines can't be included because they don't have the sensors and connectivity needed. This huge data gap limits manufacturers' opportunities to optimize plant and enterprise operations. Perisense's unique value proposition is a scalable, robust, end-to-end solution for acquiring and analyzing legacy machine data at an affordable cost.



Figure 23— Sensor Toolkit Features

Market Analysis-- The IOT market has grown rapidly due to accelerated adoption; estimated overall market size is \$100B/year in 2016, with growth to \$930B/year projected by 2025. Perisense’s segments, IOT sensors and analytics, are projected to reach almost \$40B by 2021. Perisense targets two groups of customers making up about half of the IOT sensors and analytics market: manufacturing and logistics. We focus on the 25-50% of this \$20B market that needs retrofit data acquisition and analysis capabilities, representing an addressable market of \$5-10B by 2021. Closely linked, logistics and manufacturing use similar data and can be cross sold for larger accounts and higher sales.



Source: GrandView Research— “IOT Analytics Market by Component, End Use and Segment”

Source: Markets and Markets Reports "IOT Analytics Market by Application”

Figure 24— Sensor Kit Market Data

Within these focused segments, Perisense has a compelling value proposition as our extensible system provides a low risk, low cost entry opportunity with flexibility to add unlimited amounts of connectivity and analytics as we demonstrate the value of our services. A Fortune 100 OEM recently requested a quote from Perisense for a smaller plant because “SAP won’t do anything in one of our plants for less than a million dollars and in this case such a high cost can’t be justified, but we still need the data.” Microsoft has approached Perisense about joining its Azure platform because “we are always looking for good partners and I believe you could fill an important gap which is the retrofit of legacy systems for the

IOT.” Our first customer bought from Perisense due to dissatisfaction with MTConnect based offerings already installed in his plant.

Competitive Analysis-- Most legacy machine data is sampled from a Programmable Logic Controller (PLC); it is low resolution data acquired just once or twice per cycle with low time stamp accuracy. Perisense has developed copyrighted software and proprietary edge computing to obtain data independent of the PLC related directly to product quality, at process relevant frequency and intervals. Data rates can be customized to the application. There are few robust, capable, moderate cost retrofit

Company	Merges retrofit sensor & PLC data	Retrofit sensors	Samples PLC	Plug & Play"
Perisense	✓	✓	Not needed	✓
Forcam		✓	✓	
Machine Metrics			✓	
Beet			✓	
Trumble			✓	

sensor products in this space. Forcam offers a three machine “starter kit” at a cost of \$24,950 with only 60 days of data service; this is about 4X the cost of a Perisense sensor with one year of services. Perisense will benefit from its partnership with Siemens to expand our marketing and sales reach at little or no cost. As our competition evolves, we will pursue more partners e.g., SAP, Microsoft etc. to remain highly competitive in this sector.

Perisense Company Overview

Value Proposition--Perisense is the best value provider of high quality data acquisition, storage and analysis for legacy machines.

Core Competency--Perisense has world class talent in manufacturing, sensor design and application, and data analysis.

Product Overview--The Perisense sensor toolkit solution for legacy machine data acquisition, cloud storage and analysis/reporting stands out for its low cost and high value. Our plug-and-play product minimizes installation costs and disruptions, and includes: 1) rugged, semi-customizable sensor and wireless data acquisition for legacy machinery and other assets (e.g., forklifts) that is a gateway to high margin software services; 2) secure data upload to a cloud database for storage and analysis; 3) data and software services including delivery of alerts, analytics and displays based on user KPIs and requirements. Perisense acquires data at higher resolution than competitors that sample Programmable Logic Controllers (PLCs) and can integrate data from multiple plants and locations. We can also ingest and integrate data acquired from other sources (historical data, flat files, streams from PLCs, RFID data). Our exceptional software team has produced proprietary edge computing and copyrighted software for efficient data transfer between sensors and the datacenter, while supporting immediate alerts as required by users. Perisense is also preparing two provisional patent applications: a novel sensor for real-time detection of weld defects in high value parts, and an innovative approach to easily entering supervisor or operator generated codes to the database without a need for a screen or keyboard-based interface.

Perisense is testing its third generation product at five companies in California, Ontario and Michigan, has obtained over 3,000 days of data and received product input from a wide range of other companies including Rolls Royce, GM, Siemens, Delta, Whirlpool and Caterpillar. Our product is available for sale to early users at a reduced price so we can incorporate customer feedback as we build out our platform. A higher volume Generation 4 product will roll out in early 2019, to include developing a custom board and sourcing of the board and enclosures to manufacturers in Michigan.

The team has identified the following barriers to adoption of this solution:

- General lack of awareness among SMMs that retrofitting legacy manufacturing equipment is possible. Many believe that the only way to realize gains from monitoring and data analysis require purchasing brand new equipment.
- Lack of awareness of the benefits from equipment monitoring.
- Belief that retrofitting equipment prohibitively expensive either in capital costs or from potential downtime during installation.
- Security concerns regarding cloud solutions.
- Concerns from SMMs whether they have the IT expertise to manage such a solution.

Additional Information

VIII. WORKFORCE DEVELOPMENT

N/A

IX. CONCLUSIONS/RECOMMENDATIONS

This project successfully created a product that will enable manufacturers to easily and affordably monitor legacy shop floor equipment. The product was deployed in multiple environments and proven rugged enough to operate on the factory floor.

The following recommendations would help promote adoption of this solution:

- SMM outreach about the advantages and options of retrofitting legacy equipment.
- Education to address the security concerns regarding cloud solutions.
- A pilot plan to underwrite a wider deployment among SMM to document additional use cases.

DMDII/MxD SMM members can purchase a base product package from Perisense including four sensors including one year of storage and monitoring for \$5,500 plus travel costs for installation by Perisense engineers. The customer can choose from among the following sensors: accelerometer, particulates, thermometer and current meter. Sensors are covered under warranty for one year and maintenance contracts for subsequent years are available. After the first-year storage and monitoring is \$1 a day per sensor.

Potential uses include monitoring the performance of a milling machines, utilization rate of a CNC machines, or identifying upcoming equipment failures. Perisense can also integrate additional sensors and create custom reports as identified by customer need.

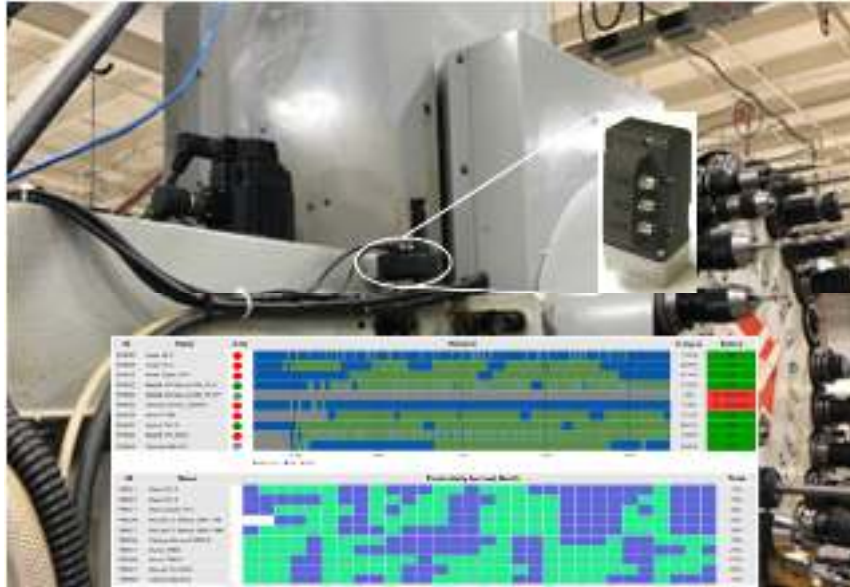


Figure 25-- Example Sensor Toolkit Installation and Report available for DMDII/MxD Members

DMDII/MxD larger OEM members can purchase a product package from Perisense including ten sensors including one year of storage and monitoring for \$15,000 (DMDII/MxD member price \$14,000) plus travel costs for installation by Perisense engineers. The customer can choose from among the following sensors: accelerometer, particulates, thermometer and current meter. Sensors are covered under warranty for one year and maintenance contracts for subsequent years are available. After the first year storage and monitoring is \$1 a day per sensor. Custom sensors can be integrated for an additional cost.

Potential OEM deployment include installation on similar equipment (a cell of drill presses or lathes) to compare performance and power usage as well as monitoring pieces of equipment both before and after maintenance.

OEMs can also purchase the solution for installation downstream in their supply chain to monitor partner equipment at remote locations.




Product Data Sheet: Sensors and Hub	
	<p>Product Description Perisense delivers machine operating data to a cloud data base via a rugged retrofit Multi-Sensor Node that acquires vibration, temperature, and current data. Our software turns the data into business metrics such as OEE, heat maps, and machine group performance over time, as well as real time alerts when user selected out of limits conditions occur.</p> 
	<p>Perisense Node Standard Product Details The Perisense Multi-Sensor Node attaches magnetically or mechanically to the monitored machine. The base package includes three sensors; multiple additional sensors can be added to the system upon request to meet customer requirements. Several packaging options are available to address application and environmental conditions (e.g., harsh environment, space limitation, etc.) The base system includes:</p> <ul style="list-style-type: none"> • WiFi connectivity (supports >70' distance to the Hub with reliable and stable data streaming) • 3 Axis accelerometer: range ±2g, 14g, 18g, 116g, resolution 13 bits, frequency range typical 500Hz up to 3.2kHz • Temperature sensor: range -55C to 125C, accuracy ±0.5C • Current sensor: ranges 15A, 30A, 50A, 100A, 1000A, frequency range 50Hz-1kHz
<p>Features</p> <ul style="list-style-type: none"> - Plug and Play functionality - Fast installation - Non-invasive solution - Reliable - Robust - Secure <p>Standard sensors</p> <ul style="list-style-type: none"> - Accelerometer - Temperature - Current <p>Additional Sensors</p> <ul style="list-style-type: none"> - Distance/stand off - Liquid level - Visible light, UV and IR - Ultrasonic & Acoustic - Hygrometer - Particulates - Hall effect - Frequency counter 	<p>Mini Remote Node</p>  <p>Base Node</p> <p>Perisense Hub Product Details The Perisense Multi-Sensor Hub aggregates data from 5-10 remote nodes within 100'. The Hub performs edge computing and transfers data to a cloud based server for further storage and analysis. To ensure the security of the customer's internal manufacturing data, acquired data is uploaded via a separate wired ION (Internet-Only Network) connection employing state of the art security protocols. Related features:</p> <ul style="list-style-type: none"> • Data ingestion rates >10kB/sec/Node • Data latency (from acquisition to arrival to AWS RDS) ~300msec

Figure 26-- Product Data Sheet

A wider deployment to gather data from a large deployment of companies could provide analytics to improve and benchmark national competitiveness. Sensor toolkits could aggregate data from multiple manufacturers to create a diverse, anonymous, standardized data set. Such a system could be a powerful engine for U.S. national industrial competitiveness.

Final “Version 2” (Available for DMDII/MxD members)

Infrastructure	Base Hardware CPU	Sensor	Connectivity
<p>Amazon Web Services (AWS) – Services used for device fleet management, data storage, processing, analysis and visualization of the data</p>	<p>Raspberry Pi3B Raspberry PI0W</p>	<p>Accelerometer (ADXL345), Particulate (SDS011), Thermometer (TMP006, DS18B20), Current meter (SCT013). Provisioned for connection of various other sensors</p>	<p>Wireless connectivity provided by special Perisense developed router (serving all the devices within the area of 10000 square feet)</p>

X. LESSONS LEARNED

Problems Encountered

Network Security

The initial deployment had problems configuring the solution to successfully connect and communicate data via the ethernet network at the test site. As a DOD supplier the network was properly secured to prevent intrusions and unauthorized equipment on the factory floor. These barriers were successfully navigated but caused installation delays preventing data from being successfully transmitted.

This was an important lesson for the project team. Many manufacturing organizations have stringent requirements and procedures for the installation of computer equipment that require internet access. Smaller companies may have limited or poorly documented networks or in some cases no network access at all.

These problems at the first installation site resulted in Perisense developing a custom router with optional cellular wireless connectivity.



Figure 27-- Perisense Custom Router

Factors Complicating Welding

Welding is aggressive from the point of view of electrical and acoustic noise, smoke and dust, and because we are testing on a manual rather than automated system there is additional complexity in measuring things like travel speed in the absence of an automated gantry. With its heavy manual component and significant electrical noise, welding has significant challenges when it comes to monitoring solutions. While there are welding systems available in the market today with built-in

sensors providing real-time weld process data, most SMMs have limited automation and instrumentation, offering a major opportunity for improvement.

Some major vendors also provide PC-based programs to analyze data and conduct statistical analysis. For example, Miller Electric has a remote database to which users can send data and receive performance reports on their operation. Another example is Lincoln Electric, which provides a welding system set up and procedure application that can be used to select parameters in the field for welding operations, but it cannot be used to collect or analyze data.

Surprises

The relatively low overhead of the software and sensors made cost and availability the major driver in selecting a CPU. Throughout the project, the number of possible hardware combinations exceeded the pace of software development. Open source software tools allowed sensors to swapped in and out to find the best solution for each case.

With the proliferation of industrial grade sensors available, the major challenge became creating a robust enclosure that could be attached to equipment and survive on the factory floor.

Plan/Scope of Work/Proposal Claim Deviations

There were no major deviations from the proposed project plan.

Risks Realized

N/A

XI. DEFINITIONS

AWS--Amazon web services

Azure--Microsoft cloud hosting environment

Cloud--Shared computer resources accessed via the internet

CBM--Condition-based maintenance

COTS--Commercial off-the-shelf

CPU--Central Processing Unit

DOD--Department of Defense

ERP--Enterprise resource planning

HTML--Hypertext markup language

IT--Information technology

IOT--Internet of things

KPI--key performance indicator

Machine Tool--A machine for shaping or machining metal or other rigid materials, usually by cutting, boring, grinding, shearing, or other forms of deformation. Machine tools employ some sort of tool that does the cutting or shaping. All machine tools have some means of constraining the workpiece and

provide a guided movement of the parts of the machine. https://en.wikipedia.org/wiki/Machine_tool

OEE--Overall Equipment Effectiveness

OEM--Original equipment manufacturer

PLM--Product lifecycle management (PLM)

Python--an interpreted, high-level, general-purpose programming language

R&D--Research and development

SMM--Small and medium manufacturers

TBM--Time-based maintenance

Welding--A fabrication or sculptural process that joins materials, usually metals or thermoplastics, by using high heat to melt the parts together and allowing them to cool causing fusion. Welding is distinct from lower temperature metal-joining techniques such as brazing and soldering, which do not melt the base metal. <https://en.wikipedia.org/wiki/Welding>

Wi-Fi--technology for radio wireless local area networking of devices based on the IEEE 802.11 standards.

XII. APPENDICES

Section should minimally include the following:

- a. List Document Deliverables

- b. Demos
 1. Setup Instructions
 2. Bill of Materials
 3. Exceptions
 4. Additional Relevant Materials

- c. Validation & Testing
 1. Plans
 2. Results
 3. Installation Reports
 4. User Guide/Installation Manual
 5. Additional Relevant Materials

PHP Driver Code for Sensors

PYTHON driver code for the PPD42NS

```
from __future__ import print_function
import math
import pigpio
```

```
class sensor:
    """
```

```
    A class to read a Shinyei PPD42NS Dust Sensor, e.g. as used
```

in the Grove dust sensor.

This code calculates the percentage of low pulse time and calibrated concentration in particles per 1/100th of a cubic foot at user chosen intervals.

You need to use a voltage divider to cut the sensor output voltage to a Pi safe 3.3V (alternatively use an in-line 20k resistor to limit the current at your own risk).

```
"""
def __init__(self, pi, gpio):
    """
    Instantiate with the Pi and gpio to which the sensor
    is connected.
    """

    self.pi = pi
    self.gpio = gpio

    self._start_tick = None
    self._last_tick = None
    self._low_ticks = 0
    self._high_ticks = 0

    pi.set_mode(gpio, pigpio.INPUT)

    self._cb = pi.callback(gpio, pigpio.EITHER_EDGE, self._cbf)

def read(self):
    """
    Calculates the percentage low pulse time and calibrated
    concentration in particles per 1/100th of a cubic foot
    since the last read.

    For proper calibration readings should be made over
    30 second intervals.

    Returns a tuple of gpio, percentage, and concentration.
    """
    interval = self._low_ticks + self._high_ticks

    if interval > 0:
        ratio = float(self._low_ticks)/float(interval)*100.0
        conc = 1.1*pow(ratio,3)-3.8*pow(ratio,2)+520*ratio+0.62;
    else:
        ratio = 0
        conc = 0.0
```

```

self._start_tick = None
self._last_tick = None
self._low_ticks = 0
self._high_ticks = 0

return (self.gpio, ratio, conc)

def _cbf(self, gpio, level, tick):

    if self._start_tick is not None:

        ticks = pigpio.tickDiff(self._last_tick, tick)

        self._last_tick = tick

        if level == 0: # Falling edge.
            self._high_ticks = self._high_ticks + ticks

        elif level == 1: # Rising edge.
            self._low_ticks = self._low_ticks + ticks

        else: # timeout level, not used
            pass

    else:
        self._start_tick = tick
        self._last_tick = tick

def pcs_to_ugm3(self, concentration_pcf):
    """
    Convert concentration of PM2.5 particles per 0.01 cubic feet to µg/ metre cubed
    this method outlined by Drexel University students (2009) and is an approximation
    does not contain correction factors for humidity and rain
    """

    if concentration_pcf < 0:
        raise ValueError('Concentration cannot be a negative number')

    # Assume all particles are spherical, with a density of 1.65E12 µg/m3
    densitypm25 = 1.65 * math.pow(10, 12)

    # Assume the radius of a particle in the PM2.5 channel is .44 µm
    rpm25 = 0.44 * math.pow(10, -6)

    # Volume of a sphere = 4/3 * pi * radius^3
    volpm25 = (4/3) * math.pi * (rpm25**3)

```

```

# mass = density * volume
masspm25 = densitypm25 * volpm25

# parts/m3 = parts/foot3 * 3531.5
# µg/m3 = parts/m3 * mass in µg
concentration_ugm3 = concentration_pcf * 3531.5 * masspm25

return concentration_ugm3

def ugm3_to_aqi(self, ugm3):
    """
    Convert concentration of PM2.5 particles in µg/ metre cubed to the USA
    Environment Agency Air Quality Index - AQI
    https://en.wikipedia.org/wiki/Air_quality_index
    Computing_the_AQI
    https://github.com/intel-iot-
    devkit/upm/pull/409/commits/ad31559281bb5522511b26309a1ee73cd1fe208a?diff=split
    """

    cbreakpointspm25 = [ [0.0, 12, 0, 50],\
        [12.1, 35.4, 51, 100],\
        [35.5, 55.4, 101, 150],\
        [55.5, 150.4, 151, 200],\
        [150.5, 250.4, 201, 300],\
        [250.5, 350.4, 301, 400],\
        [350.5, 500.4, 401, 500], ]

    C=ugm3

    if C > 500.4:
        aqi=500

    else:
        for breakpoint in cbreakpointspm25:
            if breakpoint[0] <= C <= breakpoint[1]:
                Clow = breakpoint[0]
                Chigh = breakpoint[1]
                llow = breakpoint[2]
                lhigh = breakpoint[3]
                aqi=(((lhigh-llow)/(Chigh-Clow))*(C-Clow))+llow

    return aqi

if __name__ == "__main__":

```



```

import time

while True:
    pi = pigpio.pi() # Connect to Pi.

    s = sensor(pi, 7) # set the GPIO pin number

    # Use 30s for a properly calibrated reading.
    time.sleep(30)

    # get the gpio, ratio and concentration in particles / 0.01 ft3
    g, r, c = s.read()

    if (c==1114000.62):
        print("Error\n")
        continue

    print("Air Quality Measurements for PM2.5:")
    print(" " + str(int(c)) + " particles/0.01ft^3")

    # convert to SI units
    concentration_ugm3=s.pcs_to_ugm3(c)
    print(" " + str(int(concentration_ugm3)) + " ugm^3")

    # convert SI units to US AQI
    # input should be 24 hour average of ugm3, not instantaneous reading
    aqi=s.ugm3_to_aqi(concentration_ugm3)

    print(" Current AQI (not 24 hour avg): " + str(int(aqi)))
    print("")

    pi.stop() # Disconnect from Pi.

    time.sleep(5)

```

PYTHON driver code for the ADXL345 (3 axes accelerometer from Analog Device)

```

import smbus
import adxl345.base

class ADXL345(adxl345.base.ADXL345_Base):

    STD_ADDRESS = 0x1D
    ALT_ADDRESS = 0x53

    def __init__(self, alternate=False, port=1):
        """ Initialize the driver

```

:param alternate: use the standard or alternate I2C address as selected by pin SDO/ALT_ADDRESS

:param port: number of I2C bus to use
"""

```
self.bus = smbus.SMBus(port)
if alternate:
    self.i2caddress = ADXL345.ALT_ADDRESS
else:
    self.i2caddress = ADXL345.STD_ADDRESS
```

```
def get_register(self, address):
    bytes = self.bus.read_i2c_block_data(self.i2caddress, address, 1)
    return bytes[0]
```

```
def get_registers(self, address, count):
    bytes = self.bus.read_i2c_block_data(self.i2caddress, address, count)
    return bytes;
```

```
def set_register(self, address, value):
    self.bus.write_byte_data(self.i2caddress, address, value)
```

```
from __future__ import division
import math
```

```
class ADXL345_Base:
```

```
# Registers
REG_DEVICE_ID = 0x00
REG_THRESH_TAP = 0x1D
REG_OFSX = 0x1E
REG_OFSY = 0x1F
REG_OFSZ = 0x20
REG_DUR = 0x21
REG_LATENT = 0x22
REG_WINDOW = 0x23
REG_THRESH_ACT = 0x24
REG_THRESH_INACT = 0x25
REG_TIME_INACT = 0x26
REG_ACT_INACT_CTL = 0x27
REG_THRESH_FF = 0x28
REG_TIME_FF = 0x29
REG_TAP_AXES = 0x2A
REG_ACT_TAP_STATUS = 0x2B
REG_BW_RATE = 0x2C
REG_POWER_CTL = 0x2D
REG_INT_ENABLE = 0x2E
REG_INT_MAP = 0x2F
REG_INT_SOURCE = 0x30
```

```
REG_DATA_FORMAT = 0x31
REG_DATA_X0 = 0x32
REG_DATA_X1 = 0x33
REG_DATA_Y0 = 0x34
REG_DATA_Y1 = 0x35
REG_DATA_Z0 = 0x36
REG_DATA_Z1 = 0x37
REG_FIFO_CTL = 0x38
REG_FIFO_STATUS = 0x39
```

```
# Full Resolution scale factor (0x100 LSB/g  $\approx$  3.9/1000 mg/LSB)
SCALE_FACTOR = 1/0x100
```

```
def __init__(self):
    self._full_resolution = True
    self._range = 0
```

```
def get_register(self, address):
    raise NotImplementedError("This method should be implemented by subclasses")
```

```
def get_registers(self, address, count):
    raise NotImplementedError("This method should be implemented by subclasses")
```

```
def set_register(self, address, value):
    raise NotImplementedError("This method should be implemented by subclasses")
```

```
def get_device_id(self):
    return self.get_register(ADXL345_Base.REG_DEVICE_ID)
```

```
def set_data_rate(self, hz, low_power=False):
    if hz >= 3200:
        rate = 3200
        rate_code = 0b1111
    elif hz >= 1600 and hz < 3200:
        rate = 1600
        rate_code = 0b1110
    elif hz >= 800 and hz < 1600:
        rate = 800
        rate_code = 0b1101
    elif hz >= 400 and hz < 800:
        rate = 400
        rate_code = 0b1100
    elif hz >= 200 and hz < 400:
        rate = 200
        rate_code = 0b1011
    elif hz >= 100 and hz < 200:
        rate = 100
        rate_code = 0b1010
```

```

elif hz >= 50 and hz < 100:
    rate = 50
    rate_code = 0b1001
elif hz >= 25 and hz < 50:
    rate = 25
    rate_code = 0b1000
elif hz >= 25/2 and hz < 25:
    rate = 25/2
    rate_code = 0b0111
elif hz >= 25/4 and hz < 25/2:
    rate = 25/4
    rate_code = 0b0110
elif hz >= 25/8 and hz < 25/4:
    rate = 25/8
    rate_code = 0b0101
elif hz >= 25/16 and hz < 25/8:
    rate = 25/16
    rate_code = 0b0100
elif hz >= 25/32 and hz < 25/16:
    rate = 25/32
    rate_code = 0b0011
elif hz >= 25/64 and hz < 25/32:
    rate = 25/64
    rate_code = 0b0010
elif hz >= 25/128 and hz < 25/64:
    rate = 25/128
    rate_code = 0b0001
elif hz < 25/128:
    rate = 25/256
    rate_code = 0

if low_power:
    rate_code = rate_code | 0x10

self.set_register(ADXL345_Base.REG_BW_RATE, rate_code)
return rate;

def _equal(self, value, reference, error_margin=0.1):
    return value >= (reference - error_margin) and value <= (reference + error_margin)

def _convert(self, lsb, msb):
    """ Convert the gravity data returned by the ADXL to meaningful values """
    value = lsb | (msb << 8)
    if value & 0x8000:
        value = -value ^ 0xFFFF
    if not self._full_resolution:
        value = value << self._range
    value *= ADXL345_Base.SCALE_FACTOR

```

```

return value

def _set_power_ctl(self, measure, wake_up=0, sleep=0, auto_sleep=0, link=0):
    power_ctl = wake_up & 0x03

    if sleep:
        power_ctl |= 0x04
    if measure:
        power_ctl |= 0x08
    if auto_sleep:
        power_ctl |= 0x10
    if link:
        power_ctl |= 0x20

    self.set_register(ADXL345_Base.REG_POWER_CTL, power_ctl)

def _send_data_format(self, self_test=0, spi=0, int_invert=0, justify=0):
    data_format = self._range & 0x03

    if justify:
        data_format |= 0x04
    if self._full_resolution:
        data_format |= 0x08
    if int_invert:
        data_format |= 0x20
    if spi:
        data_format |= 0x40
    if self_test:
        data_format |= 0x80

    self.set_register(ADXL345_Base.REG_DATA_FORMAT, data_format)

def _set_fifo_mode(self, mode=0, trigger=0, samples=0x1F):
    fifo_ctl = samples & 0x1F
    fifo_ctl = fifo_ctl | ((mode & 0x03) << 6)

    if trigger:
        fifo_ctl |= 0x20

    self.set_register(ADXL345_Base.REG_FIFO_CTL, fifo_ctl)

def power_on(self):
    self._set_power_ctl(True)

def power_off(self):
    self._set_power_ctl(False)

def set_range(self, range, full_resolution=True):

```

```

        """ Set the G range and the resolution. Valid range values are 2, 4, 8, 16. Full resolution set
        either 10-bit or 13-bit resolution """
        if range == 2:
            range_code = 0x0
        elif range == 4:
            range_code = 0x1
        elif range == 8:
            range_code = 0x2
        elif range == 16:
            range_code = 0x3
        else:
            raise ValueError("invalid range [" + str(range) + "] expected one of [2, 4, 8, 16]")

        self._range = range_code
        self._full_resolution = full_resolution
        self._send_data_format()

    def get_axes(self):
        """ return values for the 3 axes of the ADXL, expressed in g (multiple of earth gravity) """
        bytes = self.get_registers(ADXL345_Base.REG_DATA_X0, 6)
        x = self._convert(bytes[0], bytes[1])
        y = self._convert(bytes[2], bytes[3])
        z = self._convert(bytes[4], bytes[5])
        return {'x': x,
                'y': y,
                'z': z}

    def get_fifo_count(self):
        count = self.get_register(ADXL345_Base.REG_FIFO_STATUS)
        return count & 0x7F

    def get_fifo(self):
        """ return an array of the whole FIFO """
        fifo_count = self.get_fifo_count()
        fifo = []
        for num in range(0, fifo_count):
            fifo.append(self.get_axes())
        return fifo

    def enable_fifo(self, stream=True):
        if stream:
            self._set_fifo_mode(mode=0x02)
        else:
            self._set_fifo_mode(mode=0x01)

    def disable_fifo(self):
        self._set_fifo_mode(mode=0x00)

```

```

def set_offset(self, x, y, z):
    """ set hardware offset for the 3 axes of the ADXL, units are g """
    def convert_offset(value):
        value = value / ADXL345_Base.SCALE_FACTOR / 4
        bytes = int(value)& 0xFF
        return bytes

    self.set_register(ADXL345_Base.REG_OFSX, convert_offset(x))
    self.set_register(ADXL345_Base.REG_OFSY, convert_offset(y))
    self.set_register(ADXL345_Base.REG_OFSZ, convert_offset(z))

def calibrate(self):
    """ Auto calibrate the device offset. Put the device so as one axe is parallel to the gravity field
    (usually, put the device on a flat surface) """
    self.set_offset(0, 0, 0)
    samples = self.get_axes()

    x = samples['x']
    y = samples['y']
    z = samples['z']

    abs_x = math.fabs(x)
    abs_y = math.fabs(y)
    abs_z = math.fabs(z)

    # Find which axe is in the field of gravity and set its expected value to 1g absolute value
    if self._equal(abs_x, 1) and self._equal(abs_y, 0) and self._equal(abs_z, 0):
        cal_x = 1 if x > 0 else -1
        cal_y = 0
        cal_z = 0
    elif self._equal(abs_x, 0) and self._equal(abs_y, 1) and self._equal(abs_z, 0):
        cal_x = 0
        cal_y = 1 if y > 0 else -1
        cal_z = 0
    elif self._equal(abs_x, 0) and self._equal(abs_y, 0) and self._equal(abs_z, 1):
        cal_x = 0
        cal_y = 0
        cal_z = 1 if z > 0 else -1
    else:
        raise ValueError("Could not determine ADXL position. One axe should be set in field of
        gravity")

    offset_x = cal_x - x
    offset_y = cal_y - y
    offset_z = cal_z - z

    self.set_offset(offset_x, offset_y, offset_z)

```

```

return {'x': offset_x,
        'y': offset_y,
        'z': offset_z}

```

Python Code for Particulate PMS1003 Sensor

```

import serial
import time
import datetime
import csv
import sqlite3
import random

def hexShow(argv):
    result = ""
    hLen = len(argv)
    for i in xrange(hLen):
        hvol = ord(argv[i])
        hhex = '%02x'%hvol
        result += hhex+' '

t = serial.Serial('/dev/ttyAMA0',9600, timeout=1.5)
time_now=datetime.datetime.now()
old_time=0
conn = sqlite3.connect("Ace_Clearwater_Welding_Booth_1.db")
c = conn.cursor()
#c.execute("""CREATE TABLE Sensor_Data (time text, pm25(ug/m^3) float, pm10(ug/m^3) float,
temperature float, humidity float);""")
c.execute("""CREATE TABLE IF NOT EXISTS Sensor_Data (time text, PM25 float, PM10 float,
temperature float, humidity float);""")

while True:
    t.flushInput()
    time.sleep(10)
    retstr = t.read(32)
    hexShow(retstr)
    if len(retstr)==32:
        if(retstr[0]==b"\x42" and retstr[1]==b"\x4d"):
            pm25=ord(retstr[12])+ord(retstr[13])
            pm10=ord(retstr[14])+ord(retstr[15])
            print "Concentration for pm2.5=%.1f ug/m^3 and pm10=%.1f
ug/m^3"%(pm25,pm10)
            time_now=datetime.datetime.now()
            data_list=[time_now.strftime('%H:%M:%S'), str(pm25),str(pm10)]
            with open("PMS1003" + time_now.strftime('-%d-%m-%y') + ".csv", 'a') as fp:
                if old_time==time_now.strftime('-%d-%m-%y'):

```



```

        writer = csv.DictWriter(fp, fieldnames = ["Time", "pm2.5(ug/m^3)",
"pm10(ug/m^3)"], delimiter = ';')
        writer.writerow({'Time': time_now.strftime('%H:%M:%S'),
'pm2.5(ug/m^3)': pm25, 'pm10(ug/m^3)': pm25})
        #fp.flush()
    else:
        writer = csv.DictWriter(fp, fieldnames = ["Time", "pm2.5(ug/m^3)",
"pm10(ug/m^3)"], delimiter = ';')
        writer.writeheader()
        writer.writerow({'Time': time_now.strftime('%H:%M:%S'),
'pm2.5(ug/m^3)': pm25, 'pm10(ug/m^3)': pm25})
        old_time=time_now.strftime('%d-%m-%y')
        #fp.flush()

    #writer.write
    #fp.write("{}\n".format(data_list))
else:
    print "Problem "
    stop

# write to Database
temp_temp = round(random.uniform(65,95),1)
temp_hum = round(random.uniform(0,100), 1)
c.execute("INSERT INTO Sensor_Data VALUES (?, ?, ?, ?,
?);",(time_now.strftime('%H:%M:%S'), pm25, pm10, temp_temp, temp_hum))
conn.commit()

```

PYTHON driver code for the HTU21D Temperature & Humidity Sensor

```
from smbus import SMBus
```

```

I2C_ADDR = 0x40
CMD_TRIG_TEMP_HM = 0xE3
CMD_TRIG_HUMID_HM = 0xE5
CMD_TRIG_TEMP_NHM = 0xF3
CMD_TRIG_HUMID_NHM = 0xF5
CMD_WRITE_USER_REG = 0xE6
CMD_READ_USER_REG = 0xE7
CMD_RESET = 0xFE

```

```

class HTU21D:
    def __init__(self, busno):
        self.bus = SMBus(busno)

    def read_temperature(self):
        self.reset()
        msb, lsb, crc = self.bus.read_i2c_block_data(I2C_ADDR, CMD_TRIG_TEMP_HM, 3)

```

```

    return -46.85 + 175.72 * (msb * 256 + lsb) / 65536

def read_humidity(self):
    self.reset()
    msb, lsb, crc = self.bus.read_i2c_block_data(I2C_ADDR, CMD_TRIG_HUMID_HM, 3)
    return -6 + 125 * (msb * 256 + lsb) / 65536.0

def reset(self):
    self.bus.write_byte(I2C_ADDR, CMD_RESET)

if __name__ == '__main__':
    htu = HTU21D(1)
    print htu.read_temperature()
    print htu.read_humidity()

```

PYTHON driver code for the SDS011 Dust Sensor

```

from enum import IntEnum
import logging
import time
import struct
import serial
import math

class SDS011(object):
    """Class representing the SD011 dust sensor and its methods.
       The device_path on Win is one of your COM ports,
       on Linux it is one of "/dev/ttyUSB..." or "/dev/ttyAMA..."
    """

    """
    The serial communication uses encoded bytes:
    each serial telegram starts with 0xAA and ends with 0xAB.
    The telegram sent to the sensor:
    the second byte is 0xB4, the 16th and 17th byte is 0xFF.
    In the response from SDS011 the second byte is 0xC5.
    If it is a response sent automatically by the sensor in "Initiative" Report Mode,
    the second byte is 0xC0.
    The third byte is always the command byte.
    In response to a request command or a sensor initiated response,
    the second byte is 0xC0.
    A telegram to the sensor in Report Mode:
    -----
    Setting to Initiative:
    Message aa:b4:02:01:00:00:00:00:00:00:00:00:00:00:00:00:ff:ff:01:ab
    Response aa:c5:02:01:00:00:cc:0b:da:ab
    """

```

Setting to Passive:
Message aa:b4:02:01:01:00:00:00:00:00:00:00:00:ff:ff:02:ab
Response aa:c5:02:01:01:00:cc:0b:db:ab

'''

```
logging.getLogger(__name__).addHandler(logging.NullHandler())
```

```
__SerialStart = 0xAA  
__SerialEnd = 0xAB  
__SendByte = 0xB4  
__ResponseByte = 0xC5  
__ReceiveByte = 0xC0  
__ResponseLength = 10  
__CommandLength = 19  
__CommandTerminator = 0xFF
```

```
class Command(IntEnum):  
    """Enumeration of SDS011 commands"""  
    ReportMode = 2,  
    Request = 4,  
    DeviceId = 5,  
    WorkState = 6,  
    Firmware = 7,  
    DutyCycle = 8
```

```
class CommandMode(IntEnum):  
    """Command to get the current configuration or set it"""  
    Getting = 0,  
    Setting = 1
```

```
class ReportModes(IntEnum):  
    """Report modes of the sensor:  
    In passive mode one has to send a request command,  
    in order to get the measurement values as a response."""  
    Initiative = 0,  
    Passiv = 1
```

```
class WorkStates(IntEnum):  
    """the Work states:  
    In sleeping mode it does not send any data, the fan is turned off.  
    To get data one has to wake it up"""  
    Sleeping = 0,  
    Measuring = 1
```

```
class UnitsOfMeasure(IntEnum):  
    """The unit of the measured values.  
    Two modes are implemented:  
    The default mode is MassConcentrationEuropean returning
```

```

values in microgram/cubic meter (mg/m3).
The other mode is ParticleConcentrationImperial returning values in
particles / 0.01 cubic foot (pcs/0.01cft).
The concentration is calculated by assuming
different mean sphere diameters of pm10 or pm2.5 particles.
"""

# µg / m3, the mode of the sensors firmware
MassConcentrationEuropean = 0,
# pcs/0.01 cft (particles / 0.01 cubic foot )
ParticelConcentrationImperial = 1
# Constructor

def __init__(self, device_path, **args):
    """
    The device_path on Win is one of your COM ports.
    On Linux one of "/dev/ttyUSB..." or "/dev/ttyAMA..."
    """
    logging.info("Start of SDS011 constructor. The device_path: %s", device_path)
    self.__timeout = 2
    if 'timeout' in args.keys():      # serial line read timeout
        self.__timeout = int(args['timeout'])
    self.__unit_of_measure = self.UnitsOfMeasure.MassConcentrationEuropean
    if 'unit_of_measure' in args.keys():  # in mass or values in concentration
        if isinstance(args['unit_of_measure'], self.UnitsOfMeasure):
            self.__unit_of_measure = args['unit_of_measure']
        else:
            raise ValueError("unit_of_measure give is not of type SDS011.UnitOfMeasure.")
    self.__device_path = device_path
    self.device = None
    try:
        self.device = serial.Serial(device_path,
                                    baudrate=9600, stopbits=serial.STOPBITS_ONE,
                                    parity=serial.PARITY_NONE,
                                    bytesize=serial.EIGHTBITS,
                                    timeout=self.__timeout)
        if self.device.isOpen() is False:
            if not self.device.open():
                raise IOError(
                    "Unable to open USB to SDS011 for device %s" % device_path)
    except:
        raise IOError("SDS011: unable to set serial device %s" %
                       device_path)

    # ToDo: initiate whith the values, the sensor has to be queried for
    # that
    self.__firmware = None
    self.__reportmode = None
    self.__workstate = None

```

```

self.__duty_cycle = None
self.__device_id = None
self.__read_timeout = 0
self.__duty_cycle_start = time.time()
self.__read_timeout_drift_percent = 2
# within response the __device_id will be set
first_response = self.__response()
if len(first_response) == 0:
    # Device might be sleeping. So wake it up
    logging.warning("SDS011: While constructing the instance "
                    "the sensor is not responding. \n"
                    "Maybe in sleeping, in passive mode, or in a "
                    "duty cycle? Will wake it up.")
    self.__send(self.Command.WorkState,
                self.__construct_data(self.CommandMode.Setting,
                                      self.WorkStates.Measuring))
    self.__send(self.Command.DutyCycle, self.__construct_data(
        self.CommandMode.Setting, 0))
# at this point, device is awake, shure. So store this state
self.__workstate = self.WorkStates.Measuring
self.__get_current_config()
logging.info("SDS011 Sensor has firmware: %s", self.__firmware)
logging.info("SDS011 Sensor reportmode: %s", self.__reportmode)
logging.info("SDS011 Sensor workstate: %s", self.__workstate)
logging.info("SDS011 Sensor dutycycle: %s, None if Zero",
             self.__duty_cycle)
logging.info("SDS011 Sensor device ID: %s", self.device_id)
logging.log(16, "The SDS011 constructor is successfully executed.")

# conversion parameters come from:
# http://ir.uiowa.edu/cgi/viewcontent.cgi?article=5915&context=etd
def mass2particles(self, pm, value):
    """Convert pm size from  $\mu\text{g}/\text{m}^3$  back to concentration  $\text{pcs}/0.01\text{sq}'''
    if self.__unit_of_measure == self.UnitsOfMeasure.MassConcentrationEuropean:
        return value
    elif self.__unit_of_measure == self.UnitsOfMeasure.ParticelConcentrationImperial:

        pi = 3.14159
        density = 1.65 * pow(10, 12)

        if pm == 'pm10':
            radius = 2.60
        elif pm == 'pm2.5':
            radius = 0.44
        else:
            raise RuntimeError('SDS011 Wrong Mass2Particle parameter value for pm.\n \
                               "%s" given, "pm10" or "pm2.5" expected.' % pm)
        radius *= pow(10, -6)$ 
```

```

        volume = (4.0 / 3.0) * pi * pow(radius, 3)
        mass = density * volume
        K = 3531.5
        concentration = value / (K * mass)
        return int(concentration + 0.5)

# Destructor
def __del__(self):
    # it's better to clean up
    if self.device is not None:
        self.device.close()

# ReportMode
@property
def device_path(self):
    """The device path of the sensor"""
    return self.__device_path

# ReportMode
@property
def reportmode(self):
    """The report mode, the sensor has at the moment"""
    return self.__reportmode

@reportmode.setter
def reportmode(self, value):
    """Setter for report mode. Use self.ReportMode IntEnum"""
    if (isinstance(value, self.ReportModes) or
        value is None):
        self.__send(self.Command.ReportMode, self.__construct_data(
            self.CommandMode.Setting, value))
        self.__reportmode = value
        logging.info("SDS011 set reportmode: %s", value)
    else:
        raise TypeError("Report mode must be of type SDS011.ReportModes")

# workstate
@property
def workstate(self):
    """The workstate of the sensor as a value of type self.WorkStates"""
    return self.__workstate

@workstate.setter
def workstate(self, value):
    if (isinstance(value, self.WorkStates) or
        value is None):
        self.__send(self.Command.WorkState, self.__construct_data(
            self.CommandMode.Setting, value))

```

```

        self.__workstate = value
        logging.info("workstate setted: %s", value)
    else:
        raise TypeError("Report Mode must be of type SDS011.WorkStates")
# dutycycle

@property
def dutycycle(self):
    """The duty cycle the sensor has as a value of type int"""
    return self.__dutycycle

@dutycycle.setter
def dutycycle(self, value):

    if (isinstance(value, int) or
        value is None):
        if value < 0 or value > 30:
            raise ValueError(
                "SDS011 duty cycle has to be between 0 and 30 inclusive!")
        self.__send(self.Command.DutyCycle, self.__construct_data(
            self.CommandMode.Setting, value))
        self.__dutycycle = value
        # Calculate new timeout value
        self.__read_timeout = self.__calculate_read_timeout(value)
        self.__dutycycle_start = time.time()
        logging.info("SDS011 set duty cycle timeout: %s", self.__read_timeout)
        logging.info("SDS011 set Duty cycle: %s", value)
        self.__get_current_config()
    else:
        raise TypeError("SDS011 duty cycle should be of type int")

@property
def device_id(self):
    """The device id as a string"""
    return "{0:02x}{1:02x}".format(self.__device_id[0], self.__device_id[1]).upper()

@property
def firmware(self):
    """The firmware of the device"""
    return self.__firmware

@property
def unit_of_measure(self):
    """The unit of measure the sensor returns the values"""
    return self.__unit_of_measure

@property
def timeout(self):
    return self.__timeout

```

```

def __construct_data(self, cmdmode, cmdvalue):
    """Construct a data byte array from cmdmode and cmdvalue.
    cmdvalue has to be self.CommandMode type and cmdvalue int.
    Returns byte array of length 2"""
    if not isinstance(cmdmode, self.CommandMode):
        raise TypeError(
            "SDS011 cmdmode must be of type {0}", type(self.CommandMode))
    if not isinstance(cmdvalue, int):
        raise TypeError("SDS011 cmdvalue must be of type {0}", type(int))
    retval = bytearray()
    retval.append(cmdmode)
    retval.append(cmdvalue)
    logging.log(16, "SDS011 data %s for commandmode %s constructed.",
                cmdvalue, cmdmode)
    return retval

def __get_current_config(self):
    """Get the sensor status at construction time of this instance:
    the current status of the sensor."""
    # Getting the Dutycycle
    response = self.__send(self.Command.DutyCycle,
                           self.__construct_data(self.CommandMode.Getting, 0))
    if response is not None and len(response) > 0:

        dutycycle = response[1]
        self.__dutycycle = dutycycle
        self.__read_timeout = self.__calculate_read_timeout(dutycycle)
        self.__dutycycle_start = time.time()
    else:
        raise RuntimeError("SDS011 duty cycle is not detectable")
    response = None

    # Getting reportmode
    response = self.__send(self.Command.ReportMode,
                           self.__construct_data(self.CommandMode.Getting, 0))
    if response is not None and len(response) > 0:
        reportmode = self.ReportModes(response[1])
        self.__reportmode = reportmode
    else:
        raise RuntimeError("SDS011 report mode is not detectable")
    response = None

    # Getting firmware
    response = self.__send(self.Command.Firmware,
                           self.__construct_data(self.CommandMode.Getting, 0))
    if response is not None and len(response) > 0:
        self.__firmware = "{0:02d}{1:02d}{2:02d}".format(

```



```

        response[0], response[1], response[2])
    else:
        raise RuntimeError("SDS011 firmware is not detectable")
    response = None

def __calculate_read_timeout(self, timeoutvalue):
    newtimeout = 60 * timeoutvalue + \
        self.__read_timeout_drift_percent / 100 * 60 * timeoutvalue
    logging.log(18, "SDS011 timeout calculated for %s: %s",
        timeoutvalue, newtimeout)
    return newtimeout

def get_values(self):
    """Get the sensor response and return measured value of PM10 and PM25"""
    logging.log(16, "SDS011 get get_values entered")
    if self.__workstate == self.WorkStates.Sleeping:
        raise RuntimeError("The SDS011 sensor is sleeping and will not " +
            "send any values. Will wake it up first.")
    if self.__reportmode == self.ReportModes.Passiv:
        raise RuntimeError("The SDS011 sensor is in passive report mode "
            "and will not automaticly send values. "
            "You need to call Request() to get values.")

    self.__dutycycle_start = time.time()
    while self.dutycycle == 0 or \
        time.time() < self.__dutycycle_start + self.__read_timeout:
        response_data = self.__response()
        if len(response_data) > 0:
            logging.info(
                "SDS011 received response from sensor %d bytes.", len(response_data))
            return self.__extract_values_from_response(response_data)
        raise IOError(
            "SDS011 No data within read timeout of %d has been received." % self.__read_timeout)

def request(self):
    """Request measurement data as a tuple from sensor when its in ReporMode.Passiv"""
    response = self.__send(self.Command.Request, bytearray())
    retval = self.__extract_values_from_response(response)
    return retval

def __extract_values_from_response(self, response_data):
    """Extracts the value of PM25 and PM10 from sensor response"""
    data = response_data[2:6]
    value_of_2point5micro = None
    value_of_10micro = None
    if len(data) == 4:
        value_of_2point5micro = self.mass2particles(
            'pm2.5', float(data[0] + data[1] * 256) / 10.0)

```

```

    value_of_10micro = self.mass2particles(
        'pm10', float(data[2] + data[3] * 256) / 10.0)
    logging.log(14, "SDS011 get_values successful executed.")
    if self.dutycycle != 0:
        self.__dutycycle_start = time.time()
        return (value_of_10micro, value_of_2point5micro)
    elif self.dutycycle == 0:
        raise ValueError("SDS011 data is missing")

def __send(self, command, data):
    """The method for sending commands to the sensor and returning the response"""
    logging.log(16, "SDS011 send() entered with command %s and data %s.",
        command.name, data)
    # Proof the input
    if not isinstance(command, self.Command):
        raise TypeError("The command must be of type SDS011.Command")
    if not isinstance(data, bytearray):
        raise TypeError("SDS011 data must be of type byte array")
    logging.log(16, "SDS011 input parameters checked")
    # Initialise the commandarray
    bytes_to_send = bytearray()
    bytes_to_send.append(self.__SerialStart)
    bytes_to_send.append(self.__SendByte)
    bytes_to_send.append(command.value)
    # Add data and set zero to the remainder
    for i in range(0, 12):
        if i < len(data):
            bytes_to_send.append(data[i])
        else:
            bytes_to_send.append(0)
    # last two bytes before the checksum is the CommandTerminator
    bytes_to_send.append(self.__CommandTerminator)
    bytes_to_send.append(self.__CommandTerminator)
    # calculate the checksum
    checksum = self.__checksum_make(bytes_to_send)
    # append the checksum
    bytes_to_send.append(checksum % 256)
    # and append the terminator for serial sent
    bytes_to_send.append(self.__SerialEnd)

    logging.log(16, "SDS011 sending: %s", "".join("%02x:" % b for b in bytes_to_send))
    # send the command
    written_bytes = self.device.write(bytes_to_send)
    self.device.flush()
    if written_bytes != len(bytes_to_send):
        raise IOError("SDS011 Not all bytes written")
    #self.__debugprt(3,"Sended and flushed: %s" % bytes_to_send)
    if len(bytes_to_send) != 19:

```

```

        logging.info("SDS011 sent: %d bytes, expected 19.", len(bytes_to_send))
    # check the receive value
    received = self.__response(command)
    if len(received) != 10:
        logging.info("SDS011 received: %d bytes, expected 10.", len(received))
    if len(received) == 0:
        raise IOError("SDS011 sensor is not responding.")
    # when no command or command is request command,
    # second byte has to be ReceiveByte
    if ((command is None or command == self.Command.Request) and
        received[1] != self.__ReceiveByte):
        raise ValueError(
            "SDS011 expected to receive value {0:#X} on a value request.\
            Received:\'{1}\'.format(self.__ReceiveByte, received[1]))
    # check, if response is response of the command, except Command.Request
    if command is not self.Command.Request:
        if received[2] != command.value:
            raise ValueError(
                "SDS011 response does not belong to the command sent afore.")
        else:
            returnvalue = received[3: -2]
    else:
        returnvalue = received
    # return just the received data. Further evaluation of data outside
    # this function
    logging.log(18, "Leaving send() normal and returning %s", "".join("%02x:" % b for b in
received[3: -2]))
    return returnvalue

```

```

def __response(self, command=None):
    """Get and check the response from the sensor.
    Response can be the response of a command sent or
    just the measurement data, while sensor is in report mode Initiative"""
    # receive the response while listening serial input
    bytes_received = bytearray(1)
    one_byte = bytes(0)
    while True:
        one_byte = self.device.read(1)
        """If no bytes are read the sensor might be in sleep mode.
        It makes no sense to raise an exception here. The raise condition
        should be checked in a context outside of this fuction."""
        if len(one_byte) > 0:
            bytes_received[0] = ord(one_byte)
            # if this is true, serial data is coming in
            if bytes_received[0] == self.__SerialStart:
                single_byte = self.device.read(1)
                if (((command is not None and command != self.Command.Request)
                    and ord(single_byte) == self.__ResponseByte) or

```

```

        ((command is None or command is self.Command.Request)
         and ord(single_byte) == self.__ReceiveByte):
            bytes_received.append(ord(single_byte))
            break
    else:
        if self.__duty_cycle == 0:
            logging.error("SDS011 A sensor response has not arrived within timeout limit. "
                          "If the sensor is in sleeping mode wake it up first!"
                          "Returning an empty byte array as response!")
        else:
            logging.info("SDS011 no response. Expected while in duty cycle.")
    return bytearray()

thebytes = struct.unpack('BBBBBBBB', self.device.read(8))
bytes_received.extend(thebytes)
if command is not None and command is not self.Command.Request:
    if bytes_received[1] is not self.__ResponseByte:
        raise IOError("SDS011 no ResponseByte found in the response.")
    if bytes_received[2] != command.value:
        raise IOError(
            "Third byte of serial data \"{0}\" received is not the expected response \
            to the previous command: \"{1}\"".format(bytes_received[2], command.name))
if command is None or command is self.Command.Request:
    if bytes_received[1] is not self.__ReceiveByte:
        raise IOError("SDS011 Received byte not found on the Value Request.")
# check checksum
if self.__checksum_make(bytes_received[0:-2]) != bytes_received[-2]:
    raise IOError("SDS011 Checksum of received data is invalid.")
# set device_id if device Id is None, proof it, if it's not None
if self.__device_id is None:
    self.__device_id = bytes_received[-4:-2]
elif self.__device_id is not None and not self.__device_id.__eq__(bytes_received[-4:-2]):
    raise ValueError("SDS011 Data received (%s) does not belong "
                     "to this device with id %s.",
                     bytes_received, bytes_received[-4:-2], self.__device_id)
logging.log(18, "SDS011 The response() was successful")
return bytes_received

def reset(self):
    """
    Sets Report mode to Initiative. Workstate to Measuring and Duty cycle to 0
    """
    self.workstate = self.WorkStates.Measuring
    self.reportmode = self.ReportModes.Initiative
    self.duty_cycle = 0
    logging.info("Sensor resetted")

def __checksum_make(self, data):

```

```

"""
Generates the checksum for data to be sent or received from the sensor.
The data has to be of type byte array and must start with 0xAA,
followed by 0xB4 or 0xC5 or 0xC0 as second byte.
The sequence must end before the position of the checksum.
"""

logging.log(14, "SDS011 building the checksum for data %s.", data)
# Build checksum for data to send or receive
if len(data) not in (self.__CommandLength - 2, self.__ResponseLength - 2):
    raise ValueError("SDS011 Length data has to be {0} or {1}.".format(
        self.__CommandLength - 2, self.__ResponseLength))
if data[0] != self.__SerialStart:
    raise ValueError("SDS011 data is missing the Startbit")
if data[1] not in (self.__SendByte, self.__ResponseByte, self.__ReceiveByte):
    raise ValueError(
        "SDS011 data is missing SendBit-, ReceiveBit- or ReceiveValue-Byte")
if data[1] != self.__ReceiveByte and data[2] not in list(map(int, self.Command)):
    raise ValueError(
        "SDS011 The data command byte value \"{0}\" is not valid.".format(data[2]))
#checksum = command.value + bytes_to_send[15] + bytes_to_send[16]
checksum = 0
for i in range(2, len(data)):
    checksum = checksum + data[i]
checksum = checksum % 256
logging.log(14, "SDS011 Checksum calculated is {}.".format(checksum))
return checksum

```

PYTHON driver code for the TMP006 Temperature Sensor

```

import time
import struct
from micropython import const
from adafruit_bus_device.i2c_device import I2CDevice

__version__ = "0.0.0-auto.0"
__repo__ = "https://github.com/adafruit/Adafruit_CircuitPython_TMP006.git"

# Default device I2C address.
_TMP006_I2CADDR = const(0x40)

# Register addresses.
_TMP006_VOBJ = const(0x00)
_TMP006_TAMB = const(0x01)
_TMP006_CONFIG = const(0x02)
_TMP006_MANUID = const(0xFE)
_TMP006_DEVID = const(0xFF)

# Config register values.

```

```

_TMP006_CFG_RESET = const(0x8000)
_TMP006_CFG_MODEON = const(0x7000)
CFG_1SAMPLE = const(0x0000)
CFG_2SAMPLE = const(0x0200)
CFG_4SAMPLE = const(0x0400)
CFG_8SAMPLE = const(0x0600)
CFG_16SAMPLE = const(0x0800)
_TMP006_CFG_DRDYEN = const(0x0100)
_TMP006_CFG_DRDY = const(0x0080)

class TMP006:
    """Class to represent an Adafruit TMP006 non-contact temperature measurement
    board.
    """

    # Class-level buffer for reading and writing data with the sensor.
    # This reduces memory allocations but means the code is not re-entrant or
    # thread safe!
    _BUFFER = bytearray(4)

    def __init__(self, i2c, address=_TMP006_I2CADDR, samplerate=CFG_16SAMPLE):
        self._device = I2CDevice(i2c, address)
        self._write_u16(_TMP006_CONFIG, _TMP006_CFG_RESET)
        time.sleep(.5)
        if samplerate not in (CFG_1SAMPLE, CFG_2SAMPLE, CFG_4SAMPLE, CFG_8SAMPLE,
                              CFG_16SAMPLE):
            raise ValueError('Unexpected samplerate value! Must be one of: '\
                              'CFG_1SAMPLE, CFG_2SAMPLE, CFG_4SAMPLE, CFG_8SAMPLE, or CFG_16SAMPLE')
        # Set configuration register to turn on chip, enable data ready output,
        # and start sampling at the specified rate.
        config = _TMP006_CFG_MODEON | _TMP006_CFG_DRDYEN | samplerate
        self._write_u16(_TMP006_CONFIG, config)
        # Check device ID match expected value.
        dev_id = self.read_register(_TMP006_DEVID)
        if dev_id != 0x67:
            raise RuntimeError('Init failed - Did not find TMP006')

    @property
    def active(self):
        """True if sensor is active."""
        return self._read_u16(_TMP006_CONFIG) & _TMP006_CFG_MODEON != 0

    @active.setter
    def active(self, val):
        control = self._read_u16(_TMP006_CONFIG)
        if val:
            control |= _TMP006_CFG_MODEON
        else:

```

```

        control &= ~(_TMP006_CFG_MODEON)
self._write_u16(_TMP006_CONFIG, control)

@property
def temperature(self):
    # pylint: disable=bad-whitespace, invalid-name, too-many-locals
    """Read object temperature from TMP006 sensor."""
    if not self.active:
        raise RuntimeError('Can not read from sensor when inactive.')
    while not self._data_ready():
        pass
    vobj = self._read_sensor_voltage()
    tamb = self._read_die_temperature() + 273.15 # to kelvin
    # see TMP006 User Guide, section 5.1
    S0 = 6.4e-14 # nominal value
    a1 = 1.75e-3
    a2 = -1.678e-5
    TREF = 298.15
    b0 = -2.94e-5
    b1 = -5.7e-7
    b2 = 4.63e-9
    c2 = 13.4

    S = S0 * (1 + a1*(tamb - TREF) + a2*(tamb - TREF)**2)
    VOS = b0 + b1*(tamb - TREF) + b2*(tamb - TREF)**2
    fVOBJ = (vobj - VOS) + c2*(vobj - VOS)**2

    TOBJ = (tamb**4 + (fVOBJ/S))**0.25

    return TOBJ - 273.15 # back to celsius

def _data_ready(self):
    return (self.read_register(_TMP006_CONFIG) & _TMP006_CFG_DRDY) != 0

def _read_sensor_voltage(self):
    vobj = self.read_register(_TMP006_VOBJ)
    vobj = struct.unpack(">h", vobj.to_bytes(2, 'big'))[0]
    return vobj * 156.25e-9 # volts

def _read_die_temperature(self):
    tamb = self.read_register(_TMP006_TAMB)
    tamb = struct.unpack(">h", tamb.to_bytes(2, 'big'))[0]
    return (tamb >> 2) / 32. # celsius

def read_register(self, register):
    """Read sensor Register."""
    return self._read_u16(register)

```

```

def _read_u16(self, address):
    with self._device as i2c:
        self._BUFFER[0] = address & 0xFF
        i2c.write(self._BUFFER, end=1, stop=False)
        i2c.readinto(self._BUFFER, end=2)
        return self._BUFFER[0]<<8 | self._BUFFER[1]

def _write_u16(self, address, val):
    with self._device as i2c:
        self._BUFFER[0] = address & 0xFF
        self._BUFFER[1] = (val >> 8) & 0xFF
        self._BUFFER[2] = val & 0xFF
        i2c.write(self._BUFFER, end=3)

```

PYTHON driver code for the DS18B20 Temperature Sensor

```

from os import path, listdir, system
from glob import glob

```

```

class DS18B20(object):
    """This class represents a temperature sensor of type DS18B20"""
    DEGREES_C = 0x01
    DEGREES_F = 0x02
    KELVIN = 0x03
    BASE_DIRECTORY = "/sys/bus/w1/devices"
    SLAVE_PREFIX = "28-"
    SLAVE_FILE = "w1_slave"
    UNIT_FACTORS = {DEGREES_C: lambda x: x * 0.001, DEGREES_F: lambda x: x * 0.001 * 1.8 +
32.0, KELVIN: lambda x: x * 0.001 + 273.15}

class DS18B20Error(Exception):
    """Exception Baseclass for DS18B20 sensor errors"""
    pass

class NoSensorFoundError(DS18B20Error):
    """Exception when no sensor is found"""
    def __init__(self, sensor_id):
        self._sensor_id = sensor_id

    def __str__(self):
        if self._sensor_id:
            return "No DS18B20 temperature sensor with id '%s' found" % self._sensor_id
        return "No DS18B20 temperature sensor found"

class SensorNotReadyError(DS18B20Error):
    """Exception when the sensor is not ready yet"""
    def __str__(self):

```



```

        return "Sensor is not yet ready to read temperature"

class UnsupportedUnitError(DS18B20Error):
    """Exception when unsupported unit is given"""
    def __str__(self):
        return "Only Degress C, F and Kelvin are currently supported"

    @classmethod
    def get_available_sensors(cls):
        """Returns all available sensors"""
        sensors = []
        for sensor in listdir(cls.BASE_DIRECTORY):
            if sensor.startswith(cls.SLAVE_PREFIX):
                sensors.append(sensor[3:])
        return sensors

    @classmethod
    def get_all_sensors(cls):
        """Returns an instance for every available DS18B20 sensor"""
        return [DS18B20(sensor_id) for sensor_id in cls.get_available_sensors()]

    def __init__(self, sensor_id=None, load_kernel_modules=True):
        """If no sensor id is given the first found sensor will be taken"""
        self._type = "DS18B20"
        self._id = sensor_id
        if load_kernel_modules:
            self._load_kernel_modules()
        self._sensor = self._get_sensor()

    def get_id(self):
        """Returns the id of the sensor"""
        return self._id

    def get_type(self):
        """Returns the type of this temperature sensor"""
        return self._type

    def exists(self):
        """Returns True if the sensor exists and is available to read temperature"""
        path = self._get_sensor()
        return path is not None

    def _get_sensor(self):
        """Returns the sensors slave path"""
        sensors = self.get_available_sensors()
        if self._id and self._id not in sensors:
            raise DS18B20.NoSensorFoundError(sensor_id)

```

```

if not self._id and sensors:
    self._id = sensors[0]

    return path.join(DS18B20.BASE_DIRECTORY, DS18B20.SLAVE_PREFIX + self._id,
DS18B20.SLAVE_FILE)

def _get_sensor_value(self):
    """Returns the raw sensor value"""
    with open(self._sensor, "r") as f:
        data = f.readlines()

    if data[0].strip()[-3:] != "YES":
        raise DS18B20.SensorNotReadyError()
    return float(data[1].split("=")[1])

def _get_unit_factor(self, unit):
    """Returns the unit factor depending on the unit constant"""
    try:
        return DS18B20.UNIT_FACTORS[unit]
    except KeyError:
        raise DS18B20.UnsupportedUnitError()

def get_temperature(self, unit=DEGREES_C):
    """Returns the temperature in the specified unit"""
    factor = self._get_unit_factor(unit)
    sensor_value = self._get_sensor_value()
    return factor(sensor_value)

def get_temperatures(self, units):
    """Returns the temperatures in the specified units"""
    sensor_value = self._get_sensor_value()
    temperatures = []
    for unit in units:
        factor = self._get_unit_factor(unit)
        temperatures.append(factor(sensor_value))
    return temperatures

def _load_kernel_modules(self):
    """Load kernel modules needed by the temperature sensor"""
    system("modprobe w1-gpio")
    system("modprobe w1-therm")

```

d. **Sample Reports from Test Site**

Welding Application- Ace Clearwater

Customer makes most use of the particulates data. These are very difficult welds on flight critical repair components. The manager tracks these measurements and will go out to the shop to see what's going on when the levels go above target levels,

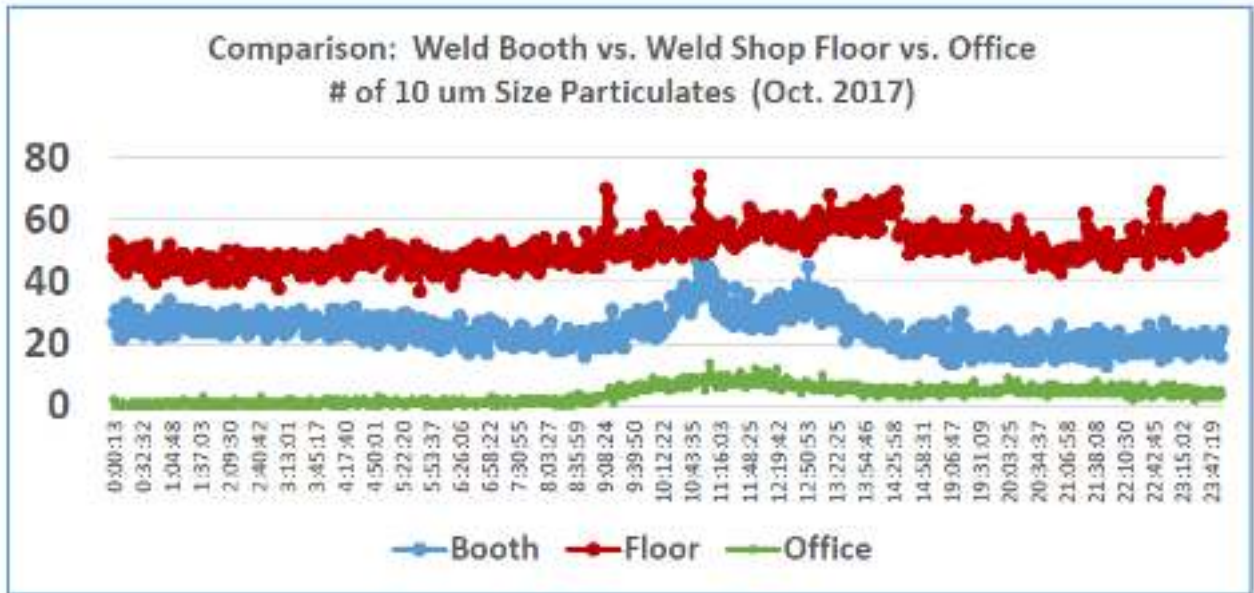


Figure 28--- ACE Particulate Reports

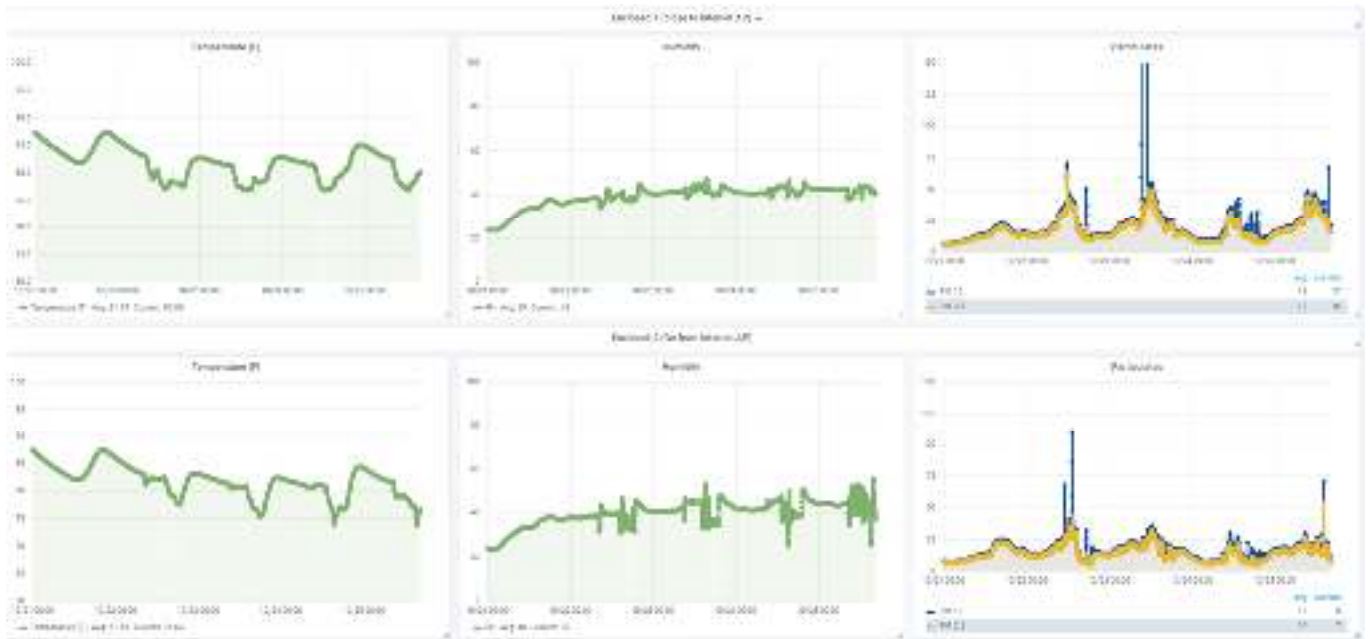


Figure 29--- ACE Temperature, Humidity, and Particulate Reports



Figure 30--- Temperature, Current, and Accelerometer Milling Machine Report

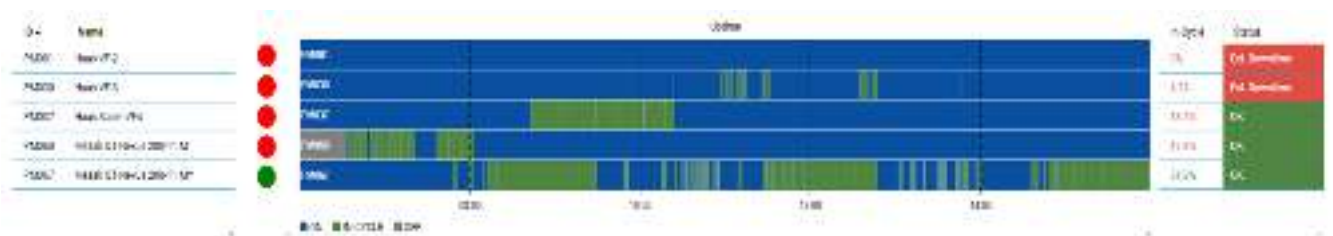


Figure 31--- Sample Machine Availability Report



Figure 32--- Sample Machine Usage Report

Verion Progression

Alpha Hardware (version 0)

Infrastructure	Base Hardware CPU	Sensors	Connectivity
Plot.ly – provided data visualization with limited data storage ability.	Raspberry PI2	Accelerometer (ADXL345), Particulate (PPD42MS)	Wired, requires separate internet drop per device

This configuration represented a functional prototype with basic reporting capability.

Infrastructure

The data storage and visualization capability were provided by Plotly (<https://plot.ly/>), a cloud service provider that integrates with the python program language used by the development team.

Base Hardware CPU

The base computing hardware selected was a Raspberry Pi 2 Model B (<https://www.raspberrypi.org/>) with the following specifications:

- A 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM
- 100 Base Ethernet connectivity
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

Sensors

The preliminary sensors and capabilities:

Accelerometer (ADXL345) is a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ± 16 g. Digital output data is formatted as 16-bit two's complement and is accessible through either a SPI (3- or 4-wire) or I2C digital interface. The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 mg/LSB) enables measurement of inclination changes less than 1.0° . Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion by comparing the acceleration on any axis with user-set thresholds. Tap sensing detects single and double taps in any

direction. Freefall sensing detects if the device is falling. These functions can be mapped individually to either of two interrupt output pins. An integrated memory management system with a 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor activity and lower overall system power consumption. Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation. The ADXL345 is supplied in a small, thin, 3 mm × 5 mm × 1 mm, 14-lead, plastic package.

Accelerometer Features

- Ultralow power: as low as 23 μ A in measurement mode and μ A in standby mode at VS = 2.5 V (typical)
- Power consumption scales automatically with bandwidth
- User-selectable resolution
- Fixed 10-bit resolution
- Full resolution, where resolution increases with g range,
- up to 13-bit resolution at ± 16 g (maintaining 4 mg/LSB
- scale factor in all g ranges)
- Embedded memory management system with FIFO
- technology minimizes host processor load
- Single tap/double tap detection
- Activity/inactivity monitoring
- Free-fall detection
- Supply voltage range: 2.0 V to 3.6 V
- I/O voltage range: 1.7 V to VS
- SPI (3- and 4-wire) and I2C digital interfaces
- Flexible interrupt modes mappable to either interrupt pin
- Measurement ranges selectable via serial command
- Bandwidth selectable via serial command
- Wide temperature range (-40°C to $+85^{\circ}\text{C}$)
- 10,000 g shock survival
- Pb free/RoHS compliant
- Small and thin: 3 mm × 5 mm × 1 mm LGA package

Particulate Sensor (PPD42NS) creates Digital (Lo Pulse) output to Particulate Matters(PM). Lo Pulse Occupancy time (LPO time) is in proportion to PM concentration. The output from “P1” is for PM whose size is around 1 micro meter or larger.

Particulate Sensor Features:

Detectable particle size	approx. 1 μ m (minimum.)
Detectable range of concentration	0~28,000 pcs/liter (0~8,000pcs/0.01 CF=283ml)
Supply Voltage	DC5V +/- 10% (CN1:Pin1=GND, Pin3=+5V) Ripple Voltage within 30mV
Operating Temperature Range	0~45 $^{\circ}$ C
Operating Humidity Range	95%rh or less (without dew condensation) Power consumption 90mA

Storage temperature	-30~60°C
Time for stabilization	1 minute after power turned on
Dimensions	59(W) × 45(H) × 22(D) [mm] Weight 24g(approx.)
Output Method	Negative Logic, Digital output, Hi : over 4.0V(Rev.2) Lo : under 0.7V (As Input impedance : 200kΩ) OP-Amp output, Pull-up resistor : 10kΩ

These sensors were chosen for its affordability and recommended use for industrial instrumentation applications.

Connectivity

Network connectivity is provided by wired ethernet.

Business Requirements

Infrastructure	Base Hardware CPU	Sensors	Connectivity
Plot.ly – provided data visualization with limited data storage ability.	Raspberry PI2	Accelerometer (ADXL345), Particulate (PPD42MS)	Wired, requires separate internet drop per device
Limited	Meets Requirements	Meets Requirements	Limited

The version 0 hardware was an initial proof-of-concept and each main component was not expected to meet every business requirement. Plot.ly was able to display reported data but lacked the capability for custom applications. Likewise, the wired connectivity was only supported for testing at this prototype stage. The Raspberry Pi and sensor were considered to meet requirements as the open source python applications did not require extensive computing power.

Beta Hardware (Version 1)

Infrastructure	Base Hardware CPU	Sensors	Connectivity
Amazon Web Services (AWS) – limited to SQL database used for storage and elastic cloud computer services	Raspberry Pi3B Raspberry PI0W	Accelerometer (ADXL345), Particulate (PMS1003), Thermometer (HTU21DS)	Wireless, connected to a wired internet bridge

Infrastructure

This configuration represented the intermediate configuration and was a major upgrade from the prototype with additional reporting capability provided by moving to AWS and adding wireless Wi-Fi capability.

The data storage and visualization capability was upgraded to Amazon Web Services (AWS) a cloud service provider that integrates with the python program language used by the development team.

AWS is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. It provides complete control of computing resources on Amazon’s proven computing environment. AWS reduces the time required to obtain and boot new server instances to minutes, allowing quick capacity changed, both up and down, as computing requirements change. AWS changes the economics of computing by allowing you to pay only for capacity that you actually use and provided the team’s developers with the capability to build failure resilient applications and isolate them from common failure scenarios.

Using AWS provided the product team with the flexibility to offer an affordable solution for smaller SMMs while providing the scalability to server larger customers as well.

Base Hardware CPU

The base computing hardware was upgraded with two new versions of Raspberry Pi 2 with the following specifications:

Raspberry Pi 3 Model B-- the earliest model of the third-generation Raspberry Pi, It replaced the Raspberry Pi 2 Model B in February 2016.

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- 40-pin extended GPIO
- 4 USB 2 ports
- 4 Pole stereo output and composite video port
- Full size HDMI
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- Micro SD port for loading your operating system and storing data
- Upgraded switched Micro USB power source up to 2.5A

Raspberry Pi Zero W-- launched at the end of February 2017, the Pi Zero W has the functionality of the original Pi Zero, but with added connectivity capabilities.

- 1GHz, single-core CPU
- 512MB RAM
- Mini HDMI and USB On-The-Go ports
- Micro USB power
- HAT-compatible 40-pin header
- Composite video and reset headers
- CSI camera connector
- 802.11 b/g/n wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

Sensors

Version 1 sensors and capabilities:

Accelerometer (ADXL345) is a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ± 16 g. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I2C digital interface. The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 mg/LSB) enables measurement of inclination changes less than 1.0° . Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion by comparing the acceleration on

any axis with user-set thresholds. Tap sensing detects single and double taps in any direction. Freefall sensing detects if the device is falling. These functions can be mapped individually to either of two interrupt output pins. An integrated memory management system with a 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor activity and lower overall system power consumption. Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation. The ADXL345 is supplied in a small, thin, 3 mm × 5 mm × 1 mm, 14-lead, plastic package.

Accelerometer Features:

- Ultralow power: as low as 23 μA in measurement mode and μA in standby mode at $V_S = 2.5\text{ V}$
- Power consumption scales automatically with bandwidth
- User-selectable resolution
- Fixed 10-bit resolution
- Full resolution, where resolution increases with g range, up to 13-bit resolution at $\pm 16\text{ g}$ (maintaining 4 mg/LSB scale factor in all g ranges)
- Embedded memory management system with FIFO technology minimizes host processor load
- Single tap/double tap detection
- Activity/inactivity monitoring
- Free-fall detection
- Supply voltage range: 2.0 V to 3.6 V
- I/O voltage range: 1.7 V to V_S
- SPI (3- and 4-wire) and I2C digital interfaces
- Flexible interrupt modes mappable to either interrupt pin
- Measurement ranges selectable via serial command
- Bandwidth selectable via serial command
- Wide temperature range (-40°C to $+85^\circ\text{C}$)
- 10,000 g shock survival
- Pb free/RoHS compliant
- Small and thin: 3 mm × 5 mm × 1 mm LGA package

Particulate Sensor (PMS1003) is a digital and universal particle concentration sensor, which can be used to obtain the number of suspended particles in the air and output the concentration of particles in the form of digital interface. This sensor can be inserted into variable instruments related to the concentration of suspended particles in the air or other environmental improvement equipment to provide correct concentration data in time.

The PMS1003 works using the laser scattering principle -- scattering by using a laser to radiate suspending particles in the air, then collect scattering light in a certain degree, and finally obtain the curve of scattering light change with time. In the end, equivalent particle diameter and the number of particles with different diameter per unit volume can be calculated by microprocessor based on MIE theory.

Mainly output as the quality and number of each particles with different size per unit volume, the unit volume of particle number is 0.1L and the unit of mass concentration is $\mu\text{ g}/\text{m}^3$. There are two options for digital output: passive and active. Default mode is active after power up. In this mode sensor would

send serial data to the host automatically. The active mode is divided into two sub-modes: stable mode and fast mode. If the concentration change is small the sensor would run at stable mode with the real interval of 2.3s, if the change is big the sensor would be changed to fast mode automatically with the interval of 200~800ms, the higher of the concentration, the shorter of the interval.

Thermometer Sensor (HTU21D) is a digital humidity sensor with temperature output that is embedded in a reflow solderable Dual Flat No leads (DFN) package with a small 3 x 3 x 0.9 mm footprint. This sensor provides calibrated, linearized signals in digital, I²C format.

HTU21D(F) digital humidity sensors are dedicated humidity and temperature plug and play (PnP) transducers for OEM applications where reliable and accurate measurements are needed. Direct interface with a micro-controller is made possible with the module for humidity and temperature digital outputs. These low power sensors are designed for high volume and cost sensitive applications with tight space constraints.

Every sensor is individually calibrated and tested. Lot identification is printed on the sensor and an electronic identification code is stored on the chip – which can be read out by command. Low battery can be detected and a checksum improves communication reliability. The resolution of these digital humidity sensors can be changed by command (8/12bit up to 12/14bit for RH/T).

Optional PTFE filter/membrane (F) protects HTU21D digital humidity sensors against dust and water immersion, as well as against contamination by particles. PTFE filter/membranes preserve a high response time. The white PTFE filter/membrane is directly stuck on the sensor housing.

Thermometer Sensor Features:

- Full interchangeability with no calibration required in standard conditions
- Instantaneous desaturation after long periods in saturation phase
- Compatible with automatized assembly processes, including Pb free and reflow processes
- Individual marking for compliance to stringent traceability requirements

These sensors were chosen for its affordability and recommended use for industrial instrumentation applications.

Connectivity

Network connectivity is provided by wireless capability (Wi-Fi) integrated into the base Raspberry Pi hardware.

Business Requirements

Infrastructure	Base Hardware CPU	Sensors	Connectivity
Amazon Web Services (AWS) – limited to SQL database used for storage and elastic cloud computer services	Raspberry PI2	Accelerometer (ADXL345), Particulate (PPD42MS)	Wireless, connected to a wired internet bridge
Meets Requirements	Meets Requirements	Meets Requirements	Limited

The Beta version 1 hardware was a functional prototype and each main component was expected to meet every business requirement. Amazon Web Services (AWS) replaced Plot.ly and not only does it support practically any application, it also allows virtual environments to be spun up to support installations of any size. It also has virtually 100% uptime due to its distributed architecture. This version supported Wi-Fi but deployment at ACE Clearwater showed that even using guest hardware could be an implementation obstacle. The Raspberry PI and sensor were considered to meet requirements as the open source python applications did not require extensive computing power.