3-23-2017

Increasing Cyber Resiliency of Industrial Control Systems

Andrew J. Chaves

INCREASING CYBER RESILIENCY
OF INDUSTRIAL CONTROL SYSTEMS

THESIS

Andrew J. Chaves, 2nd LT, USAF

AFIT-ENG-MS-17-M-013

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Army, the United States Department of Defense or the United States Government.

AFIT-ENG-MS-17-M-013

INCREASING CYBER RESILIENCY
OF INDUSTRIAL CONTROL SYSTEMS

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Cyber Operations

Andrew J. Chaves, B.S.S.E.
2nd LT, USAF

March 2017

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENG-MS-17-M-013

INCREASING CYBER RESILIENCY OF INDUSTRIAL CONTROL SYSTEMS

THESIS

Andrew J. Chaves, B.S.S.E.
2nd LT, USAF

Committee Membership:

Lieutenant Colonel Mason J. Rice, Ph.D.
Chair

Lieutenant Colonel John M. Pecarina, Ph.D.
Member

Mr. Stephen J. Dunlap
Member

AFIT-ENG-MS-17-M-013

Abstract

Industrial control systems (ICS) are designed to be resilient, capable of recovering from process faults and failures with limited impact to operations. Current ICS resiliency strategies use redundant PLCs. However, these redundant PLCs, being of similar make and model, can be exploited by the same cyber attack, defeating the ICS's resiliency strategy.

This research proposes a resiliency strategy for ICS that employs an active defense technique to remove the cyber common cause failure. The resiliency of the active defense strategy is compared to traditional ICS resiliency by implementing both strategies in a semi-simulated wastewater treatment plant aeration basin that experiences a cyber attack. The active defense technique was shown to maintain effective treatment of the wastewater through the cyber attack where the traditional implementation allowed a process disruption that prevented the effective treatment of the wastewater.

AFIT-ENG-MS-17-M-013

To my AFIT classmates

Acknowledgements

I would like to thank my entire thesis committee and research team for their guidance and help through out this process.

I would like to specifically thank Stephen Dunlap for sharing his knowledge and time. This research would not have been possible without him.

I would also like to thank LTC Rice for serving as my advisor and guiding me through the thesis process.

Andrew J. Chaves

Table of Contents

Page

List of Figures

Figure Page

List of Tables

INCREASING CYBER RESILIENCY

OF INDUSTRIAL CONTROL SYSTEMS

I. Introduction

1.1 Motivation

January 28, 1986, America watched as Space Shuttle Challenger and its crew of seven disintegrated 73 seconds into flight. Hot gases leaked from joints in the solid rocket booster that exposed the shuttle to extreme aerodynamic forces it was not designed to handle. The joints of the solid rocket booster sections are sealed using rubber O-rings that compress under the high forces during flight. Two O-rings are used in case the primary O-ring does not establish a perfect seal [13]. Regardless of this redundancy, cold temperatures did not allow the primary or redundant O-ring to compress and seal the joint, leading to a catastrophic failure and a grim day for the United States.

Common cause failures such as the cold temperatures for the Challenger's O-rings remove any redundancy designed into the system. Redundancy is only effective if a single point of failure between the components does not exist.

Similar to the astronautical community, process control also designs redundancy into their systems. Similarly, process control uses identical controllers for both the primary and redundant controller. The focus of redundancy in process control is on controller hardware failures that occur over time due to normal usage, not necessarily a physical or cyber event causing the failure [10]. This lack of diversity in primary and redundant controllers leaves the industrial control system (ICS) susceptible to a cyber attack that exploits the vulnerability found in both controllers (a common cause failure). This research proposes a combined active defense and controller resiliency strategy for process control to reduce the likelihood of a cyber attack being a common cause failure.

1.2 Research Goal and Approach

Since current ICS resiliency strategies do not use diverse primary and redundant programmable logic controllers (PLCs), the overall goal of this research is to design and test a resiliency strategy for ICS that incorporates diverse PLCs, this strategy is referred to as active defense. To be of value, the active defense strategy needs to:

1. Perform in a similar manner to traditional ICS resiliency strategies in terms of its ability to accurately control the process during a failure.
2. Outperform traditional ICS resiliency strategies during a cyber attack (i.e., cyber is not a common cause failure for the strategy).

The goal of this research is to demonstrate how the active defense strategy improves the cyber resiliency of an ICS without sacrificing control performance. In pursuit of this goal, this research took the following steps.

1. Developed criteria for measuring the resiliency of an ICS.
2. Developed a test bed to measure the resiliency of varying ICS resiliency strategies.
3. Developed hardware and software that enables two PLCs of different vendors to function as a redundant pair.

1.3 Contributions

The proposed research makes the following contributions to the process control community:

1. A model aeration basin was created that incorporates actual ICS components to measure ICS resiliency. The hardware-in-the-loop model can be used for future experimentation or ICS training.

2. Hardware and software were successfully designed to allow PLCs of different vendors to function as a redundant pair. While this solution requires more work to be accurately implemented in process control, it shows multi-vendor resiliency schemes are feasible.

3. During a cyber attack scenario, an active defense resiliency strategy incorporating primary and redundant PLCs of different vendors was shown to out-perform traditional ICS resiliency strategies. While active defense strategies are typically found in informational technology (IT) environments, the benefit of a combined active defense and controller resiliency strategy for ICS was demonstrated in this research.

4. Experimental factors were varied to determine the impact each factor has on the performance of the resiliency strategies. Understanding what factors impact resiliency performance allows for the design of improved ICS resiliency strategies. The active defense and traditional resiliency strategies were applied in multiple configurations by modifying the following factors.

   (a) The processes's sensitivity.

   (b) PLC vendor.

   (c) The redundancy scheme (e.g., hot or cold).

In general, this research proposes an active defense strategy to improve the cyber resiliency of an ICS by reducing the likelihood of cyber being a common cause failure for the system.

II. Background

2.1 Active Defense

The SANS Institute [7] generalizes cyber security strategy by describing actions that can be carried out to improve cyber security. This sliding scale of cyber security includes architecture, passive defense, active defense, intelligence and offense. If implemented properly, lower level actions (e.g., architecture and passive defense) can bolster security more than high level actions (i.e., offensive operations).

Architecture refers to the design, implementation and maintenance of the system. Essentially, security should be designed into the system, not bolted on after the fact [7].

Passive defense includes any added system that protects against threats, but requires no human interaction [7]. Firewalls, intrusion detection systems and anti-virus are common passive defense techniques. While effective, architecture and passive defense techniques are not enough to keep out a sophisticated and well-resourced attacker [7].

Active defense begins once the attacker is inside the network. Active defense is accomplished through the identification of the attack, developing an understanding of the attacker and having the flexibility to respond to the attack [7]. The response to the attack only includes operations within the network, not a counterattack upon the adversary.

2.1.1 Moving Target Defense.

Moving target defense (MTD) focuses on changing the attack surface presented to the attacker at any one time [8]. By varying the attack surface, the resources and time required to compromise the system are increased [19].

A static configuration allows the attacker to become familiar with a system, allowing for identification of system weak points and exploit development [19]. Moving target defense attempts to maintain attacker unfamiliarity and prevent the attack from moving beyond reconnaissance [9]. This constant state of unfamiliarity can be created by two different movement types [19]: (i) system configuration movement; and (ii) transforming the individual configurations. System configuration movement is randomly switching between each of the individual configurations. Individual configurations are defined by a multitude of factors (e.g., IP addresses, port numbers and operating systems). Combining both movement types creates nearly endless combinations limiting the attacker's ability to detect a pattern within the system.

Implementations of moving target defense vary greatly, but each focuses on platform diversity and rotation through these platforms. Morphisec Cyber Security [9] identifies three levels at which moving target defense takes place. At the network level, network topology such as IP addresses, port numbers and traffic can be changed to confuse the attacker. At the host level, OS level resources and configurations can be modified. Lastly, the application layer allows for modifying memory, compliers, source code, versions and routing execution through different hosts.

Argonne National Laboratory's multiple OS Rotational Environment [1] uses multiple servers, each running different operating systems. This level of diversity increases attacker uncertainty and the resources required for a successful attack. Using the same idea, Argonne National Laboratory also switches between web servers to limit the amount of time the attacker can interact with the server. This limited interaction time makes reconnaissance difficult, hopefully leading to less zero day developments.

2.1.2 Moving Target Defense in ICS.

As with most cyber security techniques, MTD has been developed for traditional IT environments. Davidson et al. [3] notes the unique challenges of employing MTD on ICS (e.g., reliance on data availability, deterministic control loops and reliability requirements). However, the authors did note data space randomization, address space randomization and configuration randomization seemed feasible.

This paper focuses on creating a combined MTD and controller resiliency strategy for ICS without modification to program and application memory or execution. The strategy can be employed as an active defense technique when an attack against the ICS has been detected or as a backup when a traditional hardware failure occurs.

This strategy only provides movement between configurations. For this reason, the alternate configuration must be kept isolated and hidden from the attacker. Since the strategy is both a redundancy and active defense technique, the resiliency of the ICS is improved compared to current ICS redundancy schemes that are susceptible to a cyber common cause failure.

2.2 Resiliency

Resilient control systems are designed so that the occurrence of undesirable events is minimized [18]. Zhu et al. [18] outlines five criteria to measure the resiliency of a control system.

1. **Protection Time:** The time that the system can withstand an incident without performance degradation.

2. **Degrading Time:** The time that the system reaches its maximal performance disruption due to an incident.

3. **Identification Time:** The time the system takes to identify an incident.

4. **Recover Time:** The time that the system needs to recover (e.g., normal operation) after an incident.

5. **Performance Degradation** Maximal system performance disruption due to an incident.

These metrics can be improved in different manners, (e.g, protection time can only be improved by incident avoidance). Other metrics (e.g., degrading time) can be improved by incident mitigation and minimization.

2.3 Resiliency through Redundancy

Eliminating undesirable incidents from occurring within an ICS is difficult as sensors, actuators, controllers and networks will all experience failures at some point. Since the incident cannot be avoided, ICS must minimize the impact of the incident. This mitigation (or minimization) is traditionally done through redundancy of components. For example, purchasing and installing multiple sensors and actuators such as utilizing two of the same flow meters (sensor redundancy) or pumps (actuator redundancy) when only one is necessary for operation. Having a duplicate PLC ready to take over process control if the primary PLC fails is an example of control redundancy.

A failure in the control portion of the system can have more severe and further reaching effects than a failed sensor or actuator. A sensor or actuator generally only influences a small piece of the system, where a controller touches multiple sensors and actuators throughout the system. Any interruption in control can cause cascading and compounding effects within the process. Since the controller influences multiple portions of the process, control redundancy is generally more complex and expensive than sensor or actuator redundancy. Due to complexity and cost there are different levels of control redundancy based on the criticality of the process being controlled [14]. However, general characteristics apply to all resiliency strategies regardless of complexity.

2.3.1 Characteristics of Redundancy Solutions.

Downer [4] discusses redundancy and reliability in complex technical systems, proposing four qualities that define a redundant system.

1. **Complexity:** Adding redundant components can increase the reliability of the system, but too many redundant components can actually become a source of unreliability. This increase in unreliability stems from the complexity of managing multiple redundant components with added software or hardware. As redundancy tends to increase complexity, simplistic design tends to increase reliability. To be properly redundant, a system must implement redundancy in moderation to avoid unreliability driven by complexity.

2. **Independence:** Redundancy is only achieved if the redundant component successfully takes over for a failed primary component (i.e., common cause failures should be avoided).

3. **Propagation:** Propagation of failure, also known as cascading effects occurs when one failure spurs the failure of another component, subsystem or entire system.

4. **Human:** The link between all components in any system is the human. The human builds, tests and validates the system. Poor installation, maintenance or implementation can lead to an immediate failure in the primary and redundant components.

Finding the balance between each of these characteristics is the challenge engineers face when designing redundant solutions–even more so as the cyber environment adds additional challenges. Redundancy approaches in process control must adapt to this new cyber threat and create a resilient system.

2.3.2 Redundancy in Process Control Today.

Redundancy within ICS is generally implemented during the installation of the system. Traditional redundancy in ICS is accomplished by the integrator installing identical controllers for both the primary and redundant systems. For example, if an integrator installs Allen-Bradley as the primary hardware, the redundant hardware will likely be Allen-Bradley.

Multiple vendors offer a redundancy-enabled, high availability PLC. This high availability PLC is a top of the line model with a CPU capable of handling the extra cycles redundancy requires. With a redundancy-enabled PLC, the customer then purchases the appropriate number of redundant CPU modules and redundant input/output modules. The redundancy modules facilitate switchover and the cross loading of data from the primary CPU to the redundant CPU [11].

Allen-Bradley offers a redundancy module (1756 RM2), that allows a user to link another rack containing a redundant CPU and communications modules as seen in Figure 1 [11].

2.3.3 Functions of Redundancy Modules.

Redundancy modules allow the ICS to switchover from the faulted or failed primary controller to the redundant controller. A switchover occurs when the fault or failure is detected and a control signal is sent to activate the redundant PLC. The redundancy module ensures the primary PLC has been cross loading process data to

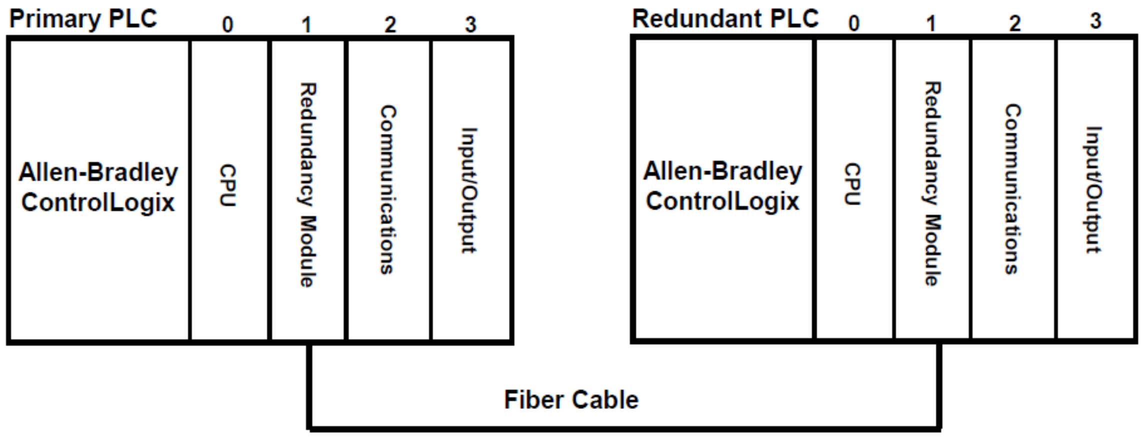


Figure 1. Allen-Bradley redundancy solution.

the redundant PLC so it can take control without causing a disruption in the process.

Allen-Bradley accomplishes switchover and cross loading using two redundancy modules. One redundancy module is located within the primary rack containing the controlling I/O, CPU and communications. This redundancy module is connected via fiber optic cable to another redundancy module within a symmetric rack [11].

2.3.3.1 Switchover.

Switchover includes recognizing an error state and transitioning from the primary PLC to the secondary PLC. The speed in which a redundancy module can execute these tasks determines the length of time in an uncontrolled state.

The total time to switchover (detection and transition) varies and depends upon program size, program design, network speed and CPU speed. The Allen-Bradley 1756-RM2 claims that with a network update time of 10 ms, switchover should take between 80 ms and 220 ms. With this speed, Allen-Bradley claims the redundant PLC will be in control of the process in time to avoid causing a bump in high priority outputs [11]. General Electric's high availability PLC (RX3I Hot Standby) can facilitate switchover within a single logic scan or 3.133 ms, claiming a bumpless transition as well [5].

The speed at which switchover must occur, just like the level of redundancy, is driven by the type of process. If the process does not deviate beyond an acceptable limit from its set point during periods where it is uncontrolled (e.g., PLCs are offline), switchover time is not as critical [5].

2.3.3.2 Cross Loading.

Cross loading provides the redundant controller with the same process data the primary controller is receiving. Cross loading is necessary to avoid a control bump when the system switches from the primary controller to the secondary controller. Cross loading generally occurs after each PLC logic scan. The primary controller's CPU and redundancy module facilitate cross loading, thus cross loading can impact PLC scan times.

Just as switchover time is driven by process tolerance, so is the cross loading of data. For extremely sensitive processes where each millisecond counts, there are ways to optimize data cross loading. Creating a ladder logic structure of large programs to avoid using jumps and deleting unused tags can reduce scan time. Aliasing tags instead of using moves and creating more tags to store inputs and outputs can also reduce scan time. Most processes can be controlled properly with an acceptable amount of disturbance with program optimization and using a highly capable CPU with low switchover time [11].

2.3.4 Common Redundancy Schemes.

Redundancy implementations vary, allowing for different levels of redundancy at different prices and complexity. Each implementation has different levels of sophistication for accomplishing switchover and cross loading. Table 1 summarizes the different redundancy schemes [14].

Regardless of the strategy, redundancy in process control currently uses identical PLCs for both the primary and redundant controller. From a cyber redundancy perspective, these identical controllers mean a successful cyber attack against the primary controller will likely lead to a successful cyber attack on the redundant controller.

2.4 Cyber Redundancy in ICS

ICS rely on robust design to defend against cyber attacks, if they attempt to defend at all. That is, ICS are designed to prevent an attack from ever occurring, not to limit the attack's effects and return to normal operation. To establish resiliency, ICS must be designed to be fault and cyber tolerant [6].

Instead of hoping to completely prevent intrusion, the modern process control architecture should have an active defense and resiliency strategy. This active defense strategy is becoming more necessary as passive defense techniques such as perimeter defense and intrusion detection are thwarted by attackers' growing knowledge and skill [15].

Babineau et al. [2] proposes four methods to create cyber tolerance on a naval ship control system which are applicable to most control systems. While Babineau et al. does not mention active defense, the methods proposed are consistent with active defense strategies.

1. **Diversification:** Increasing the degree of difficulty for an attacker by employing redundant but diverse components within the system that cannot be targeted with the same attack.

2. **Configuration Hopping:** Preventing the attacker from understanding which configuration she is attempting to attack.

3. **Data Continuity Checking:** Cross checking process data from multiple sources

to ensure integrity and identify modification.

4. **Tactical Forensics:** Determining if a system failure is due to cyber or physical malfunction.

Also discussed by Babineau et al. are criteria for rating redundant solutions that implement these strategies. The scoring criteria are organized into two categories, each category with sub-criteria.

1. **Security Score Factors**
   - **Deterrence:** The more difficult the target is to attack, the less likely it is to be attacked.
   - **Real-Time Defense:** Defending against an underway attack.
   - **Restoration:** Recovering after an attack has been executed.
2. **Cost Score Factors**
   - **Collateral System Impacts:** Impact of the solution on the performance of the rest of the system.
   - **Implementation Cost:** The cost to install, use and maintain the solution.
   - **Life Cycle Cost:** The cost associated with keeping the solution operational over the system's life.

This paper focuses on showing how a diverse redundancy implementation for ICS can also serve as an active defense technique to increase resiliency. Diversity will be implemented using primary and redundant programmable logic controllers of different vendors to improve cyber tolerance according to the security score factors. Note that cost score factors are left for future work.

Table 1. Redundancy schemes.

Cold	**Switchover:** A notification regarding the fault is pushed to an operator overseeing the process. The operator addresses the issue by troubleshooting or requesting service on the unit. No standby PLC immediately available. **Cross Loading:** None.
Warm	**Switchover:** PLC operates in "shadow mode" where it is constantly looking for a heartbeat signal from the primary PLC. If the heartbeat is lost, the redundant PLC assumes control. **Cross Loading:** None.
Hot	**Switchover:** Same as warm redundancy. **Cross Loading:** The redundant PLC is receiving the same process data as primary controller allowing it to assume control without causing a bump within the process. A redundancy module provides the CPU cycles necessary to transfer process data to the redundant controller after each PLC logic scan.
Voting	**Switchover:** Multiple PLCs are receiving real-time process data as in hot redundancy, but all PLCs are outputting to a decision agent. This agent compares each output, selecting the output that is in the majority as the control signal. **Cross Loading:** Same as hot redundancy.

III. Sewage Treatment Test-Bed

A wastewater treatment plant was chosen as the physical process on which to implement the diverse redundancy strategy. This type of facility was chosen due to having unrestricted access to a fully functioning wastewater site. The owner of the facility provided in depth instruction on each stage of the process and documentation of the plant (e.g., ladder logic, schematics and operating ranges). With the knowledge provided by the wastewater plant, a well-informed model of the aeration basin was created that includes actual ICS equipment.

3.1 Sewage Treatment Background

Sewage treatment cleans wastewater from business and residential areas, outputting the clean water into a nearby river. The process is linear with water entering large screens to remove any large debris (e.g., plastic bags, sticks and other trash). From the screens, water is then pumped via the lift station to a higher elevation to allow it to flow through the treatment process via gravity. After the initial screens and lift station, sand and grit are settled out and removed in preliminary treatment. After preliminary treatment, bacteria are used to remove the organic matter within wastewater [16]. The biological process used to clean wastewater requires specific environmental conditions to allow bacteria to optimally decompose the organic matter within the water. This process occurs in the aeration basin and will serve as the model for the experiment.

The aeration basin is a large concrete basin divided into multiple zones. Underneath the aeration basin are diffusers which are connected to blowers. The diffusers bubble oxygen provided by the blowers at different rates into each zone of the basin. There are two main zones within the aeration basin, each zone receiving different amounts of oxygen. The aerobic zone receives the most oxygen while the combined anaerobic and anoxic zone receives the least oxygen [16]. The varying levels of oxygen ensure the bacteria are in an optimal environment to decompose the organic matter in the water.

Within each zone of the aeration basin, sensors are used to determine if the wastewater is in optimal conditions for decomposition. For example, Dissolved Oxygen (DO) (the amount of free O2 molecules within the water) is used to determine and control bacteria health within the aerobic zone. Oxygen Reduction Potential (ORP) (the concentration of electron acceptors and donators within the water) is used to determine and control the bacteria health within the combined anaerobic and anoxic zone. While over simplified, the aerobic and combined anaerobic and anoxic zones comprise the aeration basin which is where biological treatment takes place. Figure 2 shows where the aeration basin fits within the overall wastewater treatment process while Figure 3 shows the design of the aeration basin itself. Following aeration, wastewater enters the final clarifier where the bacteria and decomposed organic matter settle out of the water. To kill any remaining bacteria that did not settle out in the clarifier, the water is treated with ultra-violet light before exiting the plant.

The ICS components within the aeration basin include:

1. Blowers to provide air to the aeration basin.

2. Variable frequency drive (VFD) to control the speed of the blowers.

3. Motor driven valves to control the amount of air going into each zone of the aeration basin.

4. Dissolved oxygen meter to measure the DO in the aerobic zone of the aeration basin.

5. ORP meter to measure the ORP in the anaerobic zone of the aeration basin.

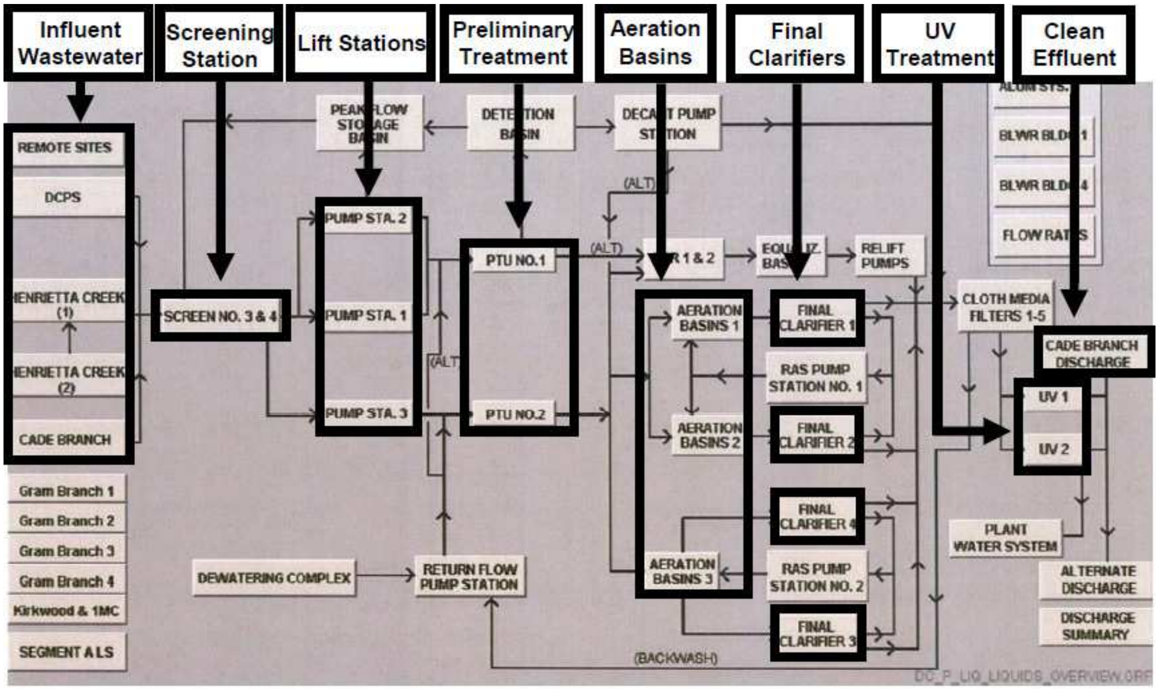


Figure 2. Wastewater treatment process overview.

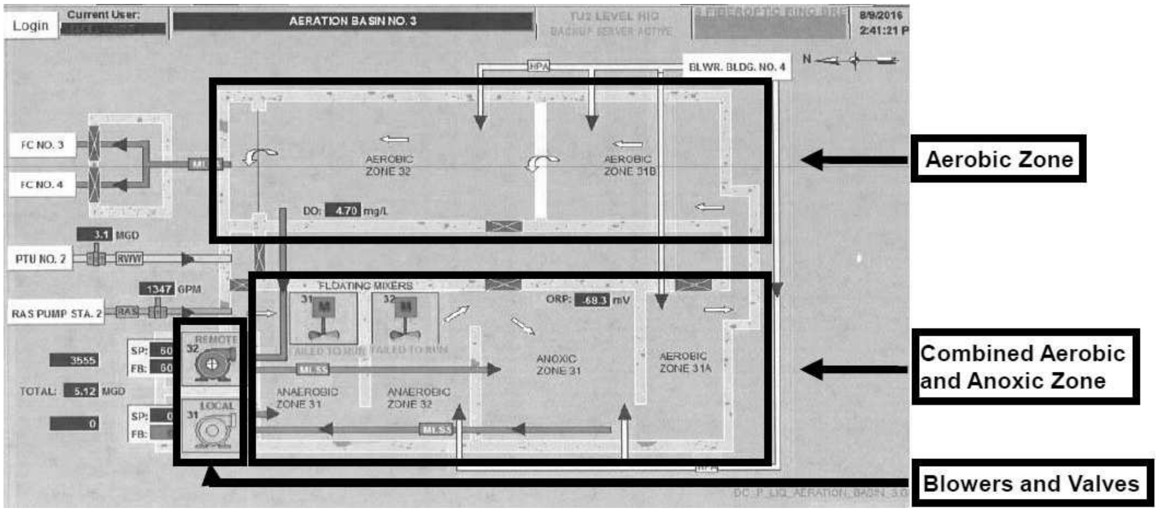


Figure 3. Aeration basin design.

3.2 Aeration Basin Model

Figure 4 shows an overview of the aeration basin model installed in a 42 cm x 55 cm x 30 cm Pelican case. Figure 5 shows a detailed description of the lower half of the case, identifying the two PLCs, solid state relays, network switch and Y-Box.

Figure 6 shows a detailed description the upper portion of the case, identifying the two digital displays, VFD, blower, alarm lights and electromechanical relay.

The aeration basin model implements a primary PLC and a redundant PLC from different manufacturers. The primary PLC is an Allen-Bradley ControlLogix containing the following modules: 1756 CPU, analog output, analog input, digital input, digital output and ethernet module. The second PLC is a General Electric series 90-30 containing the following modules: 90-30 CPU with ethernet, digital input, digital output and mixed analog input/output. The variable frequency drive used within the aeration basin model is an Allen-Bradley Powerflex 40.

The model uses a Y-Box to emulate the sensors and actuators within the process. The Y-Box is capable of receiving current and voltage and outputting current and voltage (see [17] for a detailed description of the Y-Box). Using the Y-Box, a Python program sends simulated process data to the PLC. The PLC responds by controlling an actuator such as VFD or valve. The Python program receives these actuator outputs from the PLC and updates the inputs (DO and ORP). The new DO and ORP are then output to the PLC to repeat the cycle.

The aeration basin continuously loops through the following steps to simulate the aeration basin.

1. The PLC outputs valve settings for the aerobic and combined anaerobic and anoxic zone from 0-100 percent. Valve position determines how much air is allowed into each zone. The PLC also instructs the VFD to increase or decrease frequency to maintain a constant air pressure from the blowers. The more open the valve, the more oxygen needed from the blowers to maintain pressure, thus the higher frequency needed from the VFD.

2. The Y-Box forwards this valve signal via a serial connection to a python program.

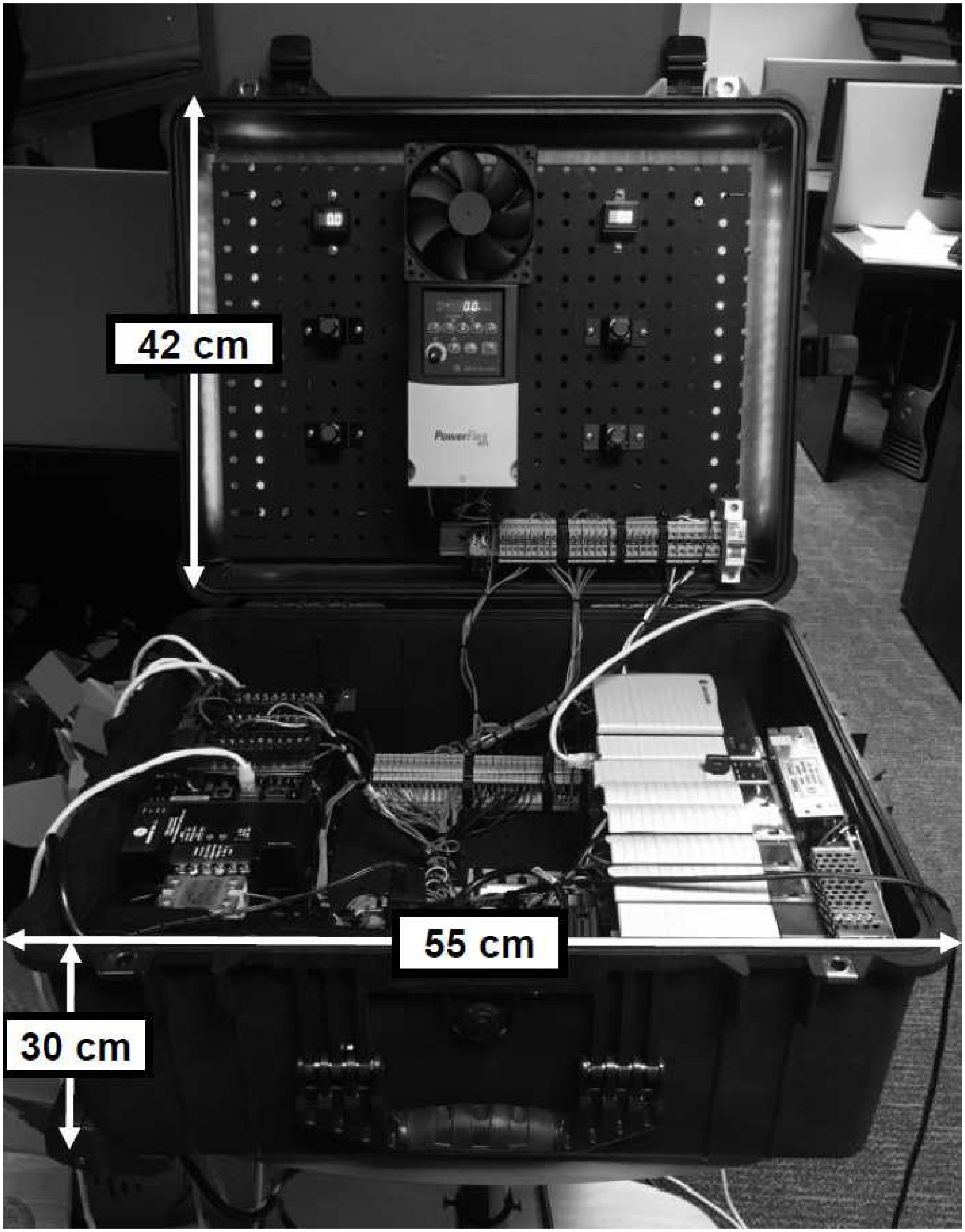


Figure 4. Aeration basin model.

3. The Python program increases or decreases the DO and ORP levels based on the position of the valve, sending the updated DO and ORP value to the Y-Box

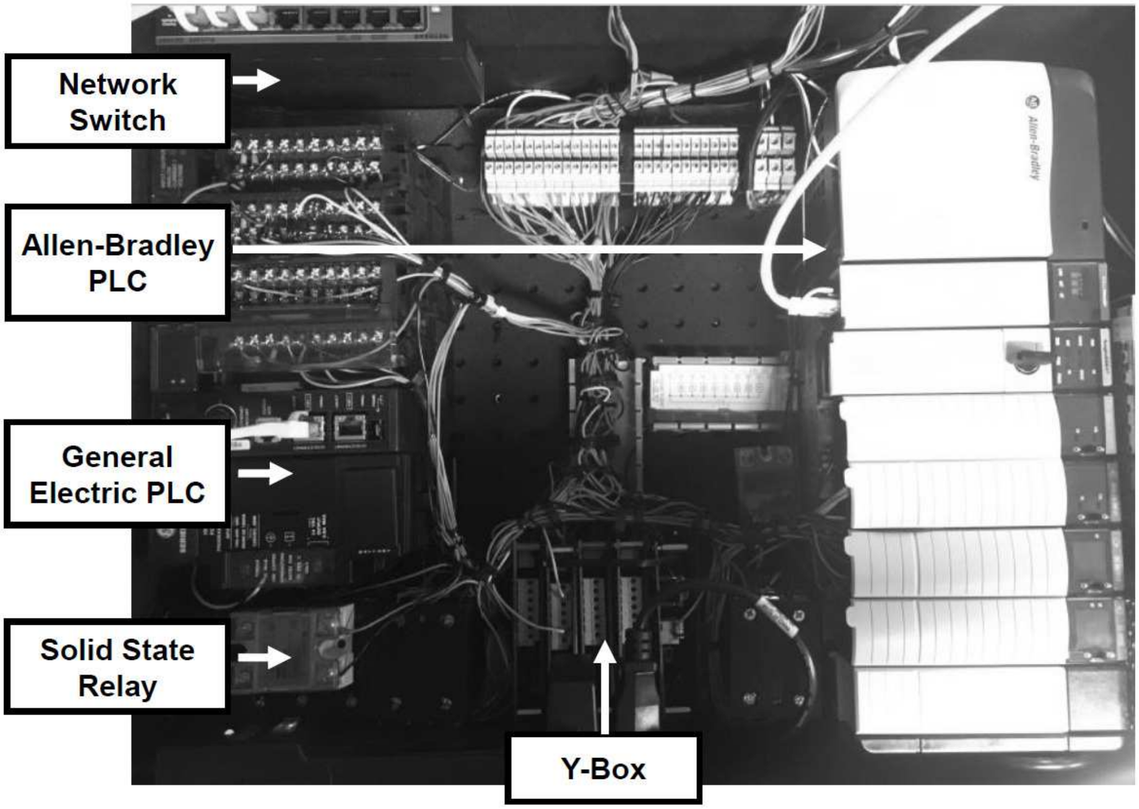


Figure 5. Bottom portion of aeration basin model.

via a serial connection. The more open the valve, the larger the increase in DO and ORP.

4. The Y-Box receives the updated DO and ORP value and forwards it as a 0-20 mA signal to the PLC.

5. The PLC receives the updated DO and ORP value and determines the next valve position to output to reach (or maintain) the DO and ORP set points.

3.3 Diverse Redundancy Implementation

To implement the diverse redundancy strategy, another PLC of a different vendor is used as the redundant PLC. The redundant PLC can operate in a hot or cold implementation.

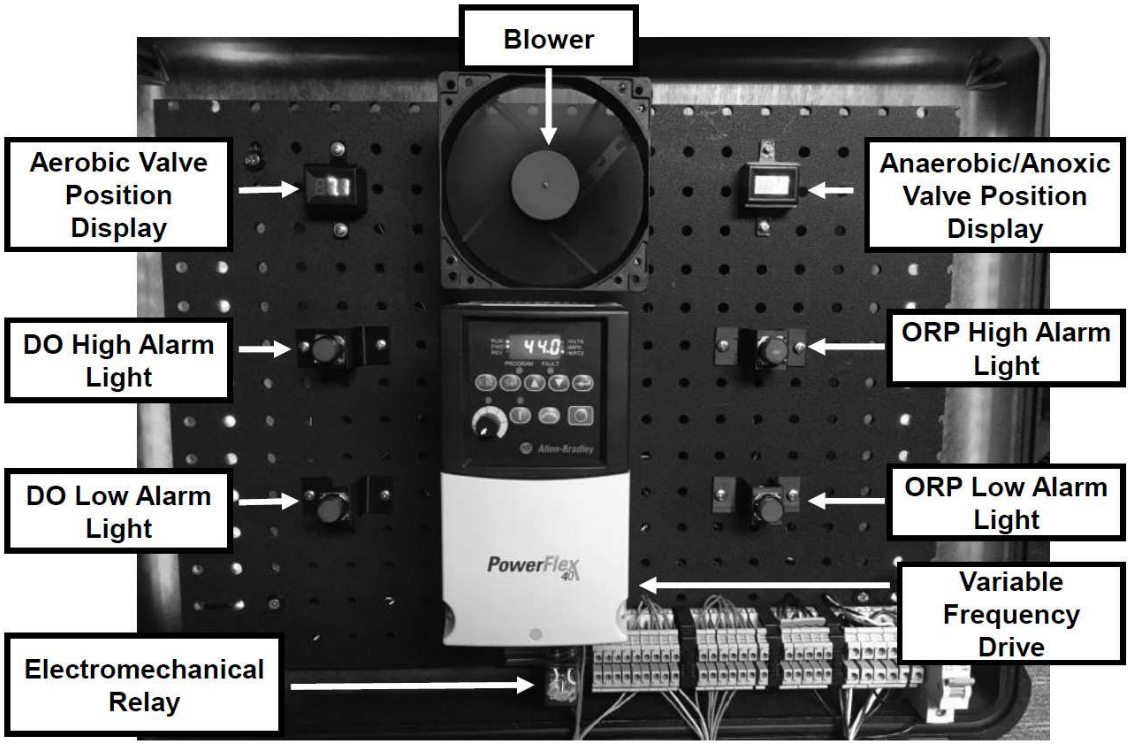


Figure 6. Top portion of aeration basin model.

To achieve hot standby, both PLCs must receive inputs from the process at all times. Since the PLCs are of different vendors, modules that facilitate the transfer of data and switchover are not available for either manufacturer. Thus, a new design was necessary.

3.3.1 Switchover.

To accomplish switchover, one PLC's outputs must drive the control of the process while the other PLC's outputs are silenced or ignored. Using relays for each PLC output, a control scheme was implemented to choose which output circuit would control the process.

A relay receives a control signal (on or off) and either completes a circuit (on) or opens the circuit (off). For example, a light bulb wired into a relay will only light up if the relay is receiving an "on" control signal. Connecting each PLC's output to a relay sets one of the PLC's output to "on" while turning the other PLC's output off. Daisy chaining all relay control signals for an individual PLC's outputs together allows for the silencing (or activation) of that PLC's outputs.

Connecting a single relay to both PLCs' relays allows for a selection between the primary PLC's output or the redundant PLC's outputs. The relay operates as a switch, completing the circuit for one PLC's set of outputs or the others. The Y-Box is in control of activating or deactivating the relay. The PLCs receive this Y-Box control signal as well, thus the PLCs know when they are serving as the primary or redundant controller. Figure 7 shows the switchover design.

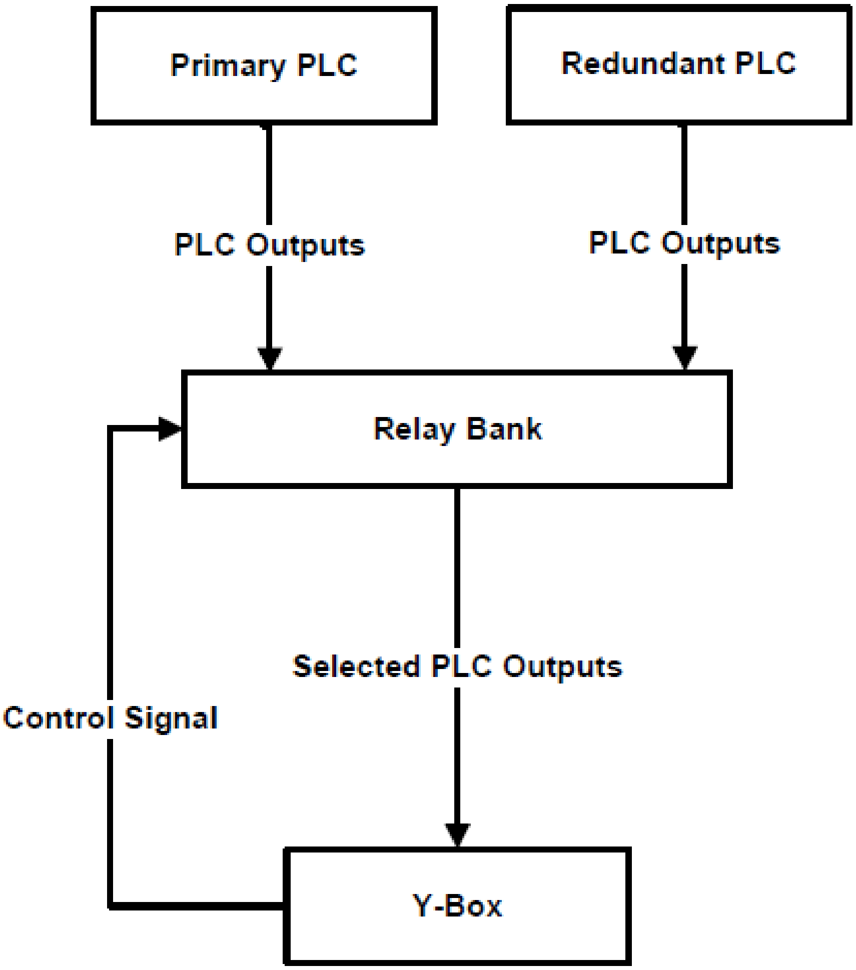


Figure 7. Switchover implementation.

3.3.2 Cross Loading.

To allow both PLCs to receive the inputs, the original design was to continue the 0-20 mA input from one PLC's input module to the other PLC's input module, wiring the PLC inputs in series. This solution does not function properly as PLC hardware is designed with the assumption that the PLC is the last device in the loop [5, 11]. Placing one PLC before another (wiring one input into another) caused inaccurate input readings. Several options were considered to address this issue, but a current splitter was chosen as it allows for symmetrical inputs to go to both PLCs without requiring extra sensors. Figure 8 shows a block diagram for the cross loading implementation.

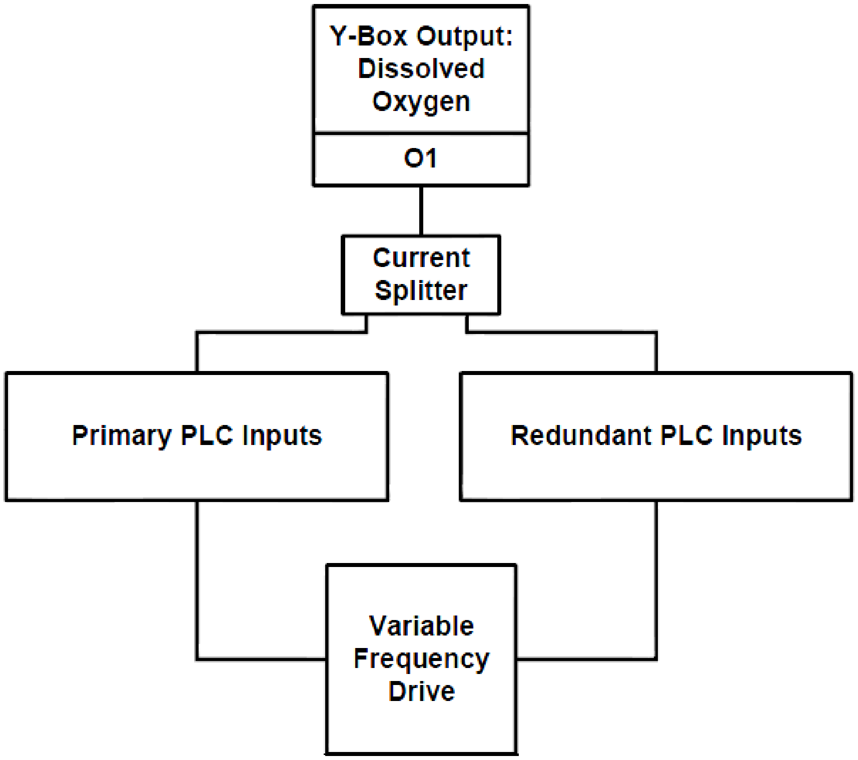


Figure 8. Cross loading implementation.

IV. Implementation Challenges

A Proportional Integral Derivative (PID) controls an input (process variable) by outputting a value (control variable) to cause the process variable to move towards the set point. If the process variable does not move towards the set point, the PID increases its control variable. As the process variable moves towards the set point, the PID reduces its control variable as to not overshoot the set point. To obtain and maintain the process's set point, the PID continuously measures the process variable and sends a control variable to move process values towards their set point.

A PID expects that its output (control variable) will cause a change in its input (process variable). If the right amount of change is not observed, the PID increases or decreases its control variable appropriately. If a PID is receiving a process variable value, but its control variable output is being silenced (as is the case for a redundant controller's PID), the PID's contro variable output may not align with the process variable value it is receiving. Since the primary and redundant PIDs are not completely identical nor run synchronously, the redundant PLC's PID control variable output will not align with the process variable it is receiving (the input caused by the primary PLC's PID control output). Over time, the redundant PLC's PID either increases its control variable output to the maximum (or decreases it to the minimum) to get the process variable to respond appropriately. If the redundant PID were to take over the process in this state, it would send the max (or min) control variable value to the actuators (causing a bump in the process), not the control variable value that corresponds with the set point.

Unfortunately, the model contains slight variances leading to PID instability when standing by in redundant mode. During pilot studies the PIDs used in the ladder logic caused inaccurate cross loading. To overcome this instability, the redundant PID was disabled by moving the current process variable into the set point of the

PID. The PID functions as if the process is in steady state and maintains its last control output. When switchover occurs, the PID will not send an extreme value that knocks the process from steady state.

V. Experimental Design

This section explains the experiment and analyses that are performed on the proposed active defense technique.

5.1 Hardware

Pilot studies were conducted to ensure the hardware was performing switchover and cross loading for the redundant PLC properly. Current and voltage measurements were taken at different points within the aeration basin model to accomplish this validation. The time for a switchover is measured from when the switchover signal is sent to when the redundant PLC's output impacts the process. The python code used to test the time for switchover can be found in appendix A, section 1.3.

5.2 Aeration Basin Simulation Experiment

The experiment consists of running the aeration basin model through a cyber attack scenario while varying multiple experiment factors. Process data is collected for each combination of factors allowing the calculation of resiliency metrics.

Each trial will follow these steps:

1. The primary PLC in the aeration basin model will control the DO to the set point of 2.0 mg/L. The PLC will maintain this set point establishing a steady state. Each trial runs for two minutes to establish this initial steady state.
2. The primary PLC will experience a cyber attack causing a deviation from the set point. This will vary depending on the trial's factors (e.g., redundancy strategy and active defense).

3. The aeration basin model will employ the redundancy and active defense strategy it was assigned for the specific trial to overcome the cyber attack and re-establish the 2.0 mg/L DO set point.

4. The trial is concluded once the model re-establishes a DO steady state at the 2.0 mg/L set point.

The aeration basin simulation python code can be found in Appendix A, Section 1.1. The ladder logic executed on the Allen-Bradley PLC can be found in Appendix B while the General Electric's ladder logic can be found in Appendix C.

5.3.1 Cyber Attack Scenario.

Two different cyber attacks will be executed based on which controller is serving as the primary controller. Only one type of cyber attack will be executed during each trial.

1. **Allen-Bradley Attack:** A Common Industrial Protocol (CIP) packet is sent to place the Allen-Bradley controller in program mode. The Allen-Bradley PLC will remain in program mode for ten seconds, halting execution of ladder logic. The Allen-Bradley PLC is then set back to run mode.

2. **General Electric Attack:** A TCP replay attack is executed to place the General Electric controller in program mode. The General Electric PLC remains in program mode for ten seconds, halting execution of ladder logic. The General Electric PLC is then set back to run mode.

Following the attack, the redundant controller (if available) will take over control for the remainder of the simulation. Note that the Allen-Bradley attack is ineffective on the General Electric PLC and vice versa.

5.3.2 Experiment Factors.

Trials of the experiment will be defined by three factors which are summarized in Table 2.

Table 2. Experiment factors.

	Level 1	**Level 2**	**Level 3**
Redundancy Strategy	HAD	CAD	HTR
Redundant PLC	General Electric	Allen-Bradley	
Process Tolerance	High	Medium	Low

- **Redundancy Strategy:** Redundancy strategy is varied to determine if the active defense technique increases resiliency for hot and cold redundancy configurations. Hot Active Defense (HAD) is a hot redundancy configuration that implements the active defense technique. Cold Active Defense (CAD) is a cold redundancy configuration that also implements the active defense technique. Hot Traditional Redundancy (HTR) is a hot redundancy configuration that does not implement the active defense technique (primary and secondary controller are identical).

- **Redundant PLC:** The redundant PLC is varied to determine PLC brand impacts the resiliency strategies' performance.

- **Process Tolerance:** Different process tolerances are tested to understand the impact tolerance has on the switchover speed necessary to avoid causing a bump in the process. High tolerance (level 1) corresponds to DO values changing at a slow rate, as low tolerance (level 3) corresponds to DO values changing at a fast rate. Since the simulation must run quickly to achieve multiple trials, the high tolerance level represents approximately 1500 times the rate at which DO would fall in an actual aeration basin.

All factor level combinations will be run through the aeration basin model thirty times for a total of 540 trials. The python code that automates the experiment, running all factor combinations through the simulation can be found is Appendix A, Section 1.2.

5.3.3 Resiliency Metrics.

The resiliency metrics outlined in Zhu et al. [18] map well to the security score factors outlined in Babineau et al. [2].

Deterrence is increased as the system's protection time increases. Real-time defense and restoration increase as identification time, recover time, degrading time and performance degradation decrease. Due to these relationships, an active defense and redundancy implementation that increases the system's resiliency metrics also improves the system's ability to deter, defend and recover.

Resiliency is measured for each active defense and redundancy implementation. Note that all times being measured are relative to the faster rate of the aeration basin.

- **Protection Time:** This criterion will not be measured in the experiment. The experiment will assume the primary controller is exploited by a cyber attack. However, if an attacker knows the ICS employs two different controllers, they may be deterred from attempting the attack due to the increased complexity, ultimately increasing the system's protection time.

- **Degrading Time:** Relative degrading time will be measured within the experiment by tracking how long it takes to achieve performance degradation. The time it takes to reach the minimum DO level within the aeration basin model will determine degrading time.

- **Identification Time:** The proposed solution does not incorporate an attack detection mechanism. Since attack detection is left for future work, identification time will not be measured within the experiment.

- **Recover Time:** Recover time will be measured by timing how long the process takes to return to steady state after the cyber attack occurs.

- **Performance Degradation:** Performance degradation will be measured by observing the change in DO levels. Specifically, the deviation of DO values from the set point during the experiment.

Figure 9 illustrates these measurements for a single trial.

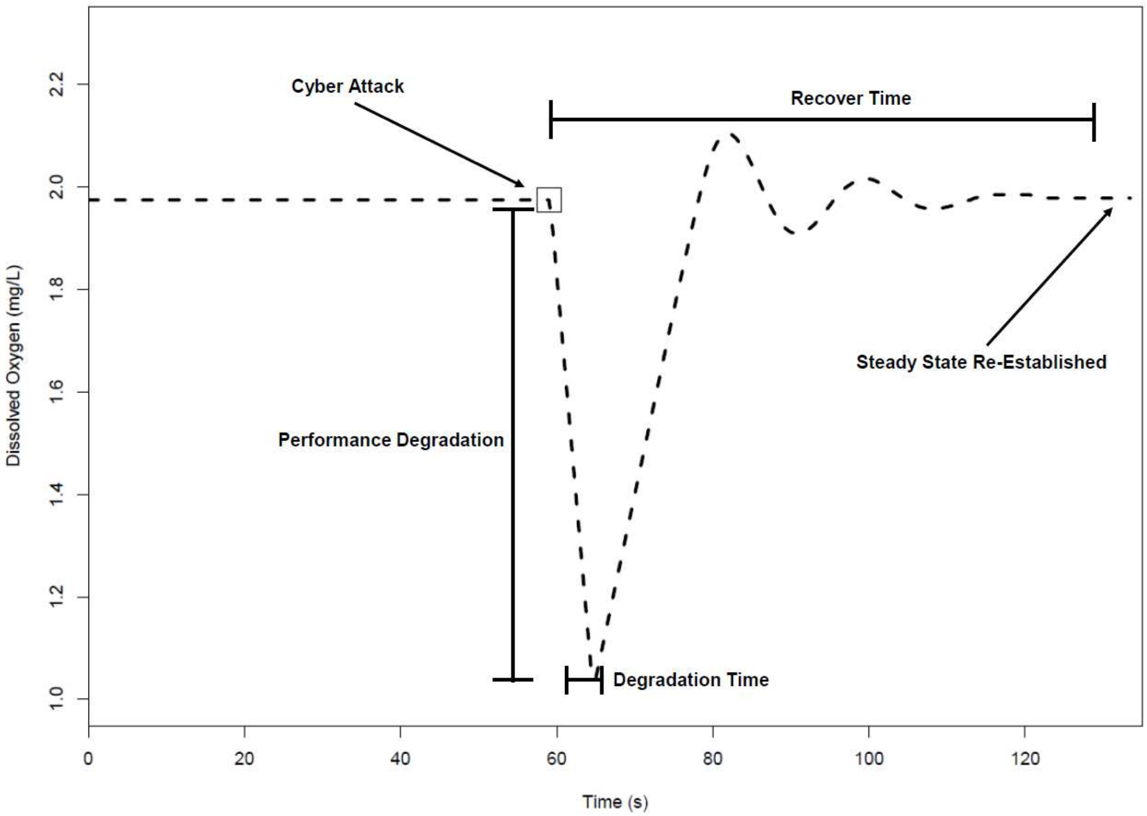


Figure 9. Measurement of resiliency metrics (DO levels in aeration basin).

VI. Results

This section discusses the results of the experiment.

6.1 Redundancy Characteristics

The general characteristics of a redundant system are revisited to show that the active defense technique's design follows the suggestions provided in Section 4.1.

1. **Complexity:** The solution uses relays to accomplish switchover as well as to silence the redundant PLC outputs. These devices are simple, requiring only a digital control signal. No modifications were made to PLC hardware or modules to accomplish switchover or cross loading.

2. **Independence:** The proposed redundancy implementation uses Allen-Bradley and General Electric PLCs making it unlikely an identical cyber attack will be successful against both PLCs.

3. **Propagation:** Switching over to the redundant PLC after a cyber attack occurs silences the outputs of the corrupted primary PLC. This silencing of outputs does not allow the corrupted PLC to affect the process.

4. **Human:** Since the solution is mainly implemented with hardware (relays and wires), there is less risk of software bugs allowing the redundancy system to be exploited and bypassed.

The proposed redundancy implementation adds independence and limits the propagation of a cyber induced failure while avoiding complexity. Including this redundancy implementation in the initial design of the of the ICS can help avoid common cause failures and potentially increase the resiliency of the system.

6.2 Hardware

While the primary PLC was controlling the process, the current coming out of the relay controlling the redundant PLC's outputs was zero mA, showing the relays properly silenced the redundant PLC's outputs. Current was again measured from the redundant PLC's outputs, but following a switchover. The current measured ranged from 12-15 mA, which corresponds to a valve position of 60-80 within the aeration basin model. These values were checked against the engineering workstation and are correct, showing the redundant PLC's outputs are active following a switchover.

DO measurements from both the General Electric and Allen-Bradley PLC were taken to determine error between the controllers. On average, the GE PLC read the DO value approximately 0.05 mg/L or 2.5 percent higher than the Allen-Bradley. A difference of .05 mg/L in DO levels does not negatively effect the treatment of wastewater.

A mean switchover speed of 13.06 ms with a standard deviation of 0.0047 was recorded. This speed is approximately 10 ms slower than GE's RX3I hot redundancy module and 77 ms faster than Allen-Bradley's 1756 hot redundancy module, making this feasible for most applications.

6.3 Measuring Increased Resiliency

This section includes a discussion and analysis of each implementation's (i.e., HAD, CAD and HTR) resiliency metrics.

6.3.1 HAD.

When the cyber attack was executed this implementation immediately switched from the primary PLC to the hot redundant PLC of a different manufacturer. Since the switchover took place within 13 ms and the redundant controller had knowledge of the process, the process experienced virtually no disturbance besides the measurement error in DO (.05 mg/L). Any bump the process may have experienced was less than error within experiment and could not be determined.

Figure 10 shows the DO levels of the process before and after the cyber attack for all three implementations. Note that Figure 10 does not identify steady state or switchover for HAD due to its nearly instantaneous switchover and lack of process disturbance. In this trial, all three implementations are being controlled initially by the Allen-Bradley PLC and establish a steady state DO level just below the 2.0 mg/L set point. The implementations do not maintain 2.0 mg/L set point due to PLC calibration.

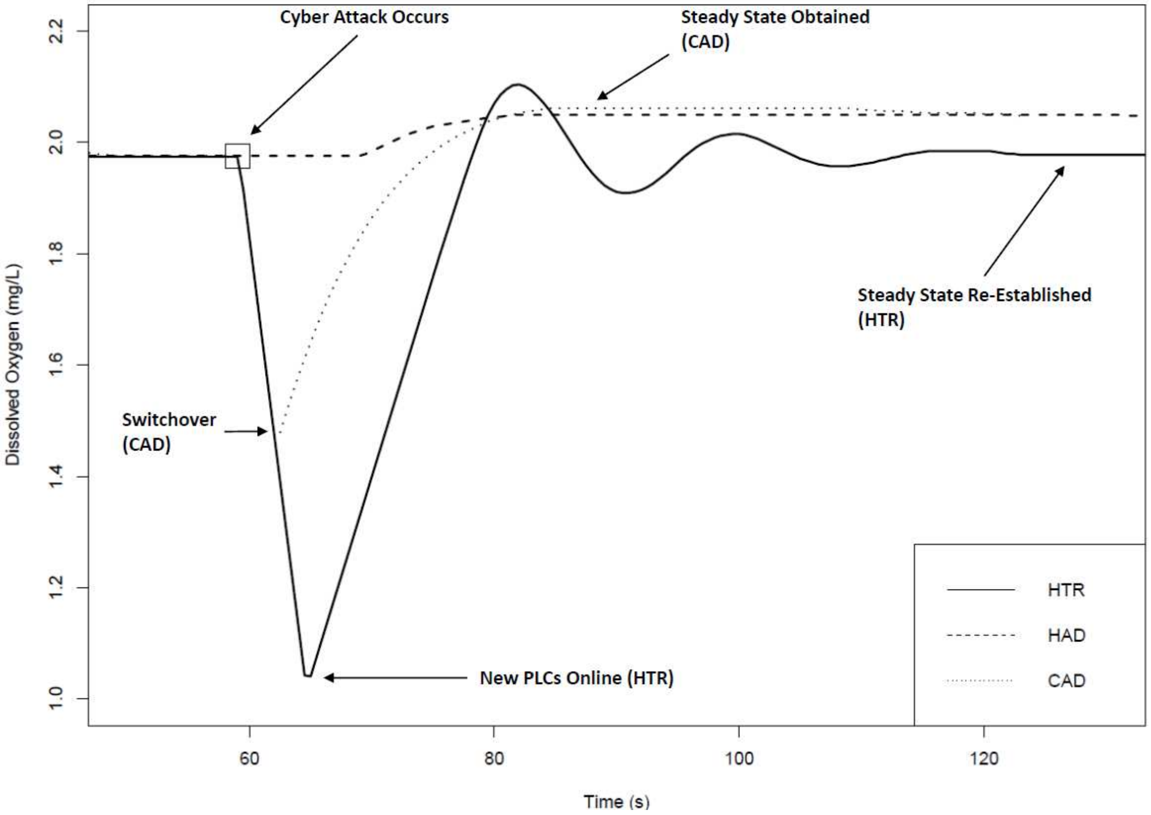


Figure 10. DO levels over time for all three implementations (GE redundant PLC and medium process tolerance).

The HAD implementation shows virtually no process disruption following the cyber attack meaning the bacteria in the aeration basin are still removing the organic matter effectively.

CAD shows a moderate disruption, but it maintains a high enough DO level and recovers to steady state in a relatively short amount of time to still remove organic matter from the water, just not optimally.

HTR shows a very large disruption as both the primary and redundant controller were exploited by the cyber attack which lead to increased downtime. While organic matter is still being removed from the water, the rate at which the removal is occurring is significantly reduced, potentially leading to increased treatment times. If the treatment time becomes too large (due to the low DO content), the aeration basin can bottleneck the entire sewage treatment process, leading to back flow of sewage into residential areas.

All three implementations eventually recover from the attack and establish a steady state near the 2.0 mg/L set point. HAD and CAD are now using the redundant GE PLC which measures the DO value approximately .05 mg/L higher than the Allen-Bradley due to PLC calibration. For this reason, HAD and CAD establish their steady state slightly above the 2.0 mg/L set point. HTR's redundant PLC was identical to its primary PLC (Allen-Bradley), thus it established a steady state slightly below the 2.0 mg/L set point (the same steady state value it established at the beginning of the trial). Note that for HTR, the primary PLC is re-enabled to function as the identical redundant PLC.

6.3.2 CAD.

The cyber attack scenario rendered the primary PLC unavailable causing the redundant controller of a different manufacturer to boot up and take control. Since the redundant controller was in a cold state, the process was uncontrolled for 5-8

Table 3. Resiliency metrics for CAD by process tolerance.

		Mean	SD	Min	Max
High Tolerance	DT	7.731	1.910	5.303	9.879
	RT	94.600	21.260	51.480	101.500
	PD	0.331	0.099	0.171	0.439
Medium Tolerance	DT	7.694	1.969	5.327	9.919
	RT	61.910	20.061	34.740	95.310
	PD	0.628	0.216	0.282	0.848
Low Tolerance	DT	7.680	1.968	5.305	9.938
	RT	78.410	14.386	49.750	105.450
	PD	1.277	0.385	0.527	1.677

seconds while the controller booted. Table 3 summarizes the resiliency metrics (e.g., degradation time (DT), recover time (RT) and performance degradation (PD)) for the CAD implementation based on process tolerance where DT and RT are given in seconds and PD is given in milligrams per liter (mg/L).

Due to the CAD implementation leaving the process uncontrolled while its redundant PLC booted, the process tolerance level had a large impact on its resiliency metrics. DO levels dropped approximately four times more for the low tolerance process compared to the high tolerance process level. This larger drop in DO for the low tolerance process level led to a slower recover time compared to the high tolerance process as well.

For all process tolerance levels, the efficiency of organic matter decomposition is reduced, with medium and low tolerances processes reaching DO values where treatment begins to be ineffective.

While the CAD did experience high levels of performance degradation, it can still treat wastewater effectively if it recovers from the loss in DO quickly. However, CAD's ratio of degradation time to recover time (the time it took to lose the DO compared to the time it took to regain the same amount of DO) is fairly low. It took approximately 7-10 times longer to recover the DO than it did to lose it.

This long recover time is likely due to the cold redundancy configuration of the PLCs. The redundant PLC's PID takes time to learn how to return the process to steady state. Due to the CAD's high performance degradation and recover time for processes with low and medium tolerance, the CAD implementation is likely only feasible for highly tolerant processes.

6.3.3 HTR.

The cyber attack rendered the primary PLC inoperable bringing the identical redundant controller online. However, the redundant controller was susceptible to the same attack, thus the HTR implementation was unable to control the process for the full ten seconds. Table 4 summarizes the resiliency metrics (e.g., degradation time (DT), recover time (RT) and performance degradation (PD)) for the HTR implementation based on process tolerance where DT and RT are given in seconds (s) and PD is given in milligrams per liter (mg/L).

For high tolerance, the HTR implementation likely continued to treat the water as its DO dropped approximately 25 percent, staying above 1.5 mg/L, but its treatment was likely not at an optimal rate. For both medium and low process tolerance, HTR was likely incapable of treating water at a high enough rate to avoid causing a bottleneck in the treatment process. Similar to CAD, HTR is only feasible for systems with high tolerance to a lack of control. However, HTR's resiliency metrics are heavily dependent on the attack and the ICS personnel's ability to troubleshoot,

Table 4. Resiliency metrics for HTR by process tolerance.

		Mean	SD	Min	Max
High Tolerance	DT	10.840	0.311	10.420	11.590
	RT	81.070	17.969	62.618	109.580
	PD	0.485	0.017	0.477	0.523
Medium Tolerance	DT	11.020	.433	10.420	11.750
	RT	81.620	5.966	70.980	89.550
	PD	0.968	0.043	.933	1.021
Low Tolerance	DT	10.710	0.116	10.410	10.970
	RT	92.650	23.7519	68.090	123.96
	PD	1.849	0.001	1.848	1.851

fix, or replace the exploited PLCs. Ten seconds was chosen as the relative duration of the cyber attack, but an actual attack may cause more or less downtime and thus, significantly more or less process disruption. The HTR implementation is at the mercy of the attack and relies heavily on ICS personnel.

6.3.4 Comparing CAD and HTR.

CAD and HTR implementations are compared to show the potential increase in resiliency provided by the active defense technique. Figures 11, 12 and 13 show how resiliency metrics for CAD and HTR differed when using an Allen-Bradley primary controller, General Electric redundant controller and medium tolerance process level.

Since the data was not normally distributed, a Mann-Whitney-Wilcoxon Test was used to determine differences among the two implementation's resiliency metrics. Comparisons were made between the two implementations based on process tolerance

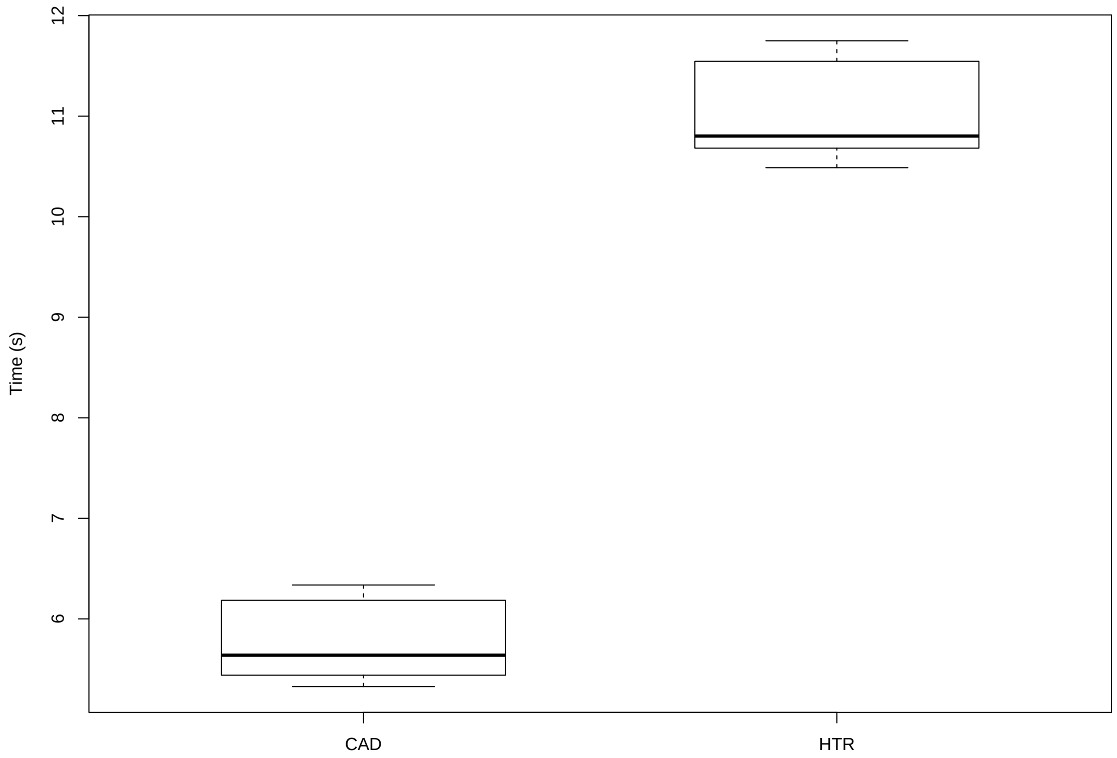


Figure 11. Degradation time for CAD and HTR implementations (AB primary PLC, GE redundant PLC with medium process tolerance).

level and the PLC serving as the redundant controller. All combinations of process tolerance and redundant controllers for the two implementations showed a significant difference ($p < 0.05$).

CAD had lower degradation time and performance degradation compared to HTR for all process tolerance levels. For recover time, CAD recovered faster than HTR for both low and medium tolerance, but recovered slower at high tolerance by approximately 13 seconds. In the high tolerance process, CAD and HTR experience small drops in DO, with averages of 0.331 and 0.485 mg/L respectively. With the drop in DO values between the implementations being small and fairly similar, the accuracy with which the PLC's PID can bring the process back to steady state has a large

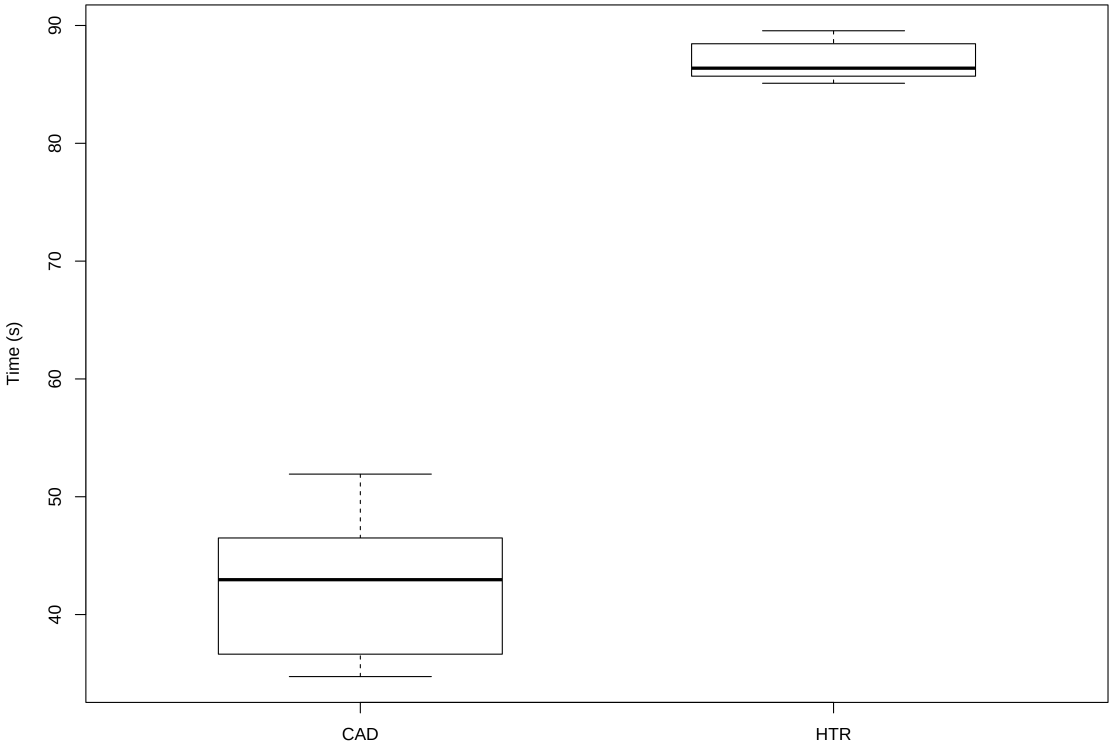


Figure 12. Recover time for CAD and HTR implementations (AB primary PLC, GE redundant PLC with medium process tolerance).

impact on the recover time. This is not the case for low and medium tolerance levels as HTR loses much more DO than CAD. Due to the large difference in the DO loss for HTR and CAD, HTR recovers slower simply because it has more DO to regain. With similar drops in DO for the high tolerance process, HTR is capable of recovering faster due to its redundant PLC's PID more accurately controlling the process than CAD's redundant PLC's PID. HTR's redundant PID is initially more accurate than CAD's redundant PID because it is identical to the primary PLC's PID, avoiding the problems discussed in Section 8.0. If CAD's primary PLC and redundant PLC PIDs were tuned identically, this difference in accuracy between HTR and CAD's redundant PIDs would likely not exist, allowing the CAD to recover faster for the

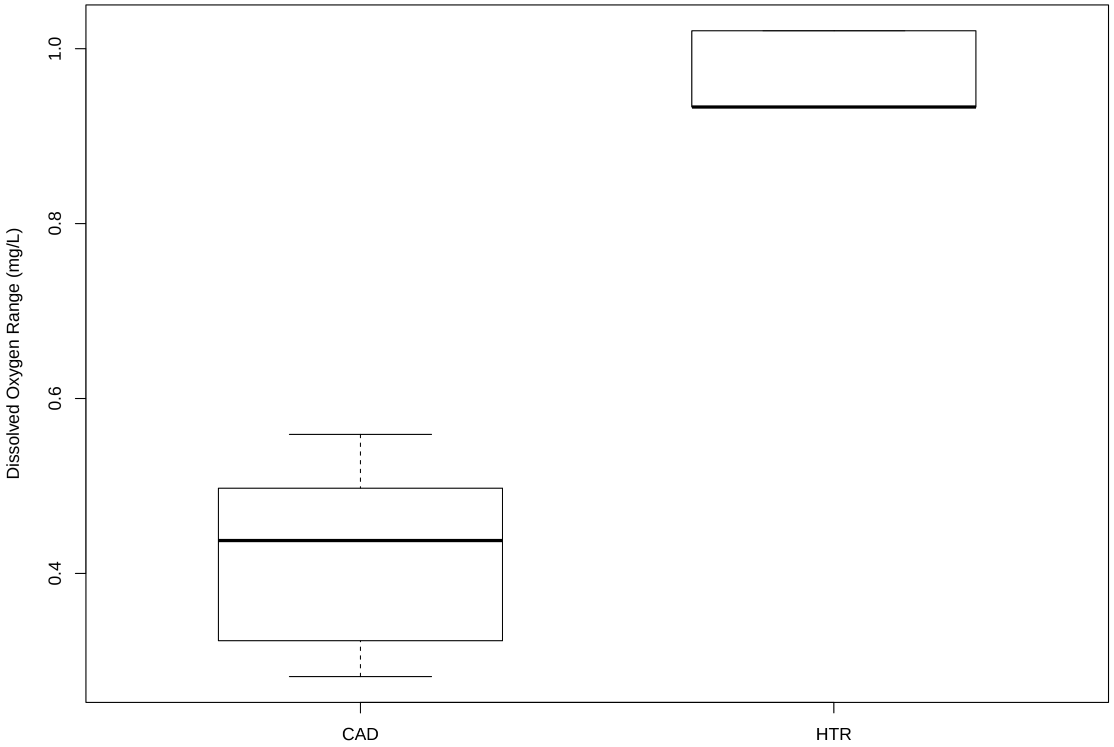


Figure 13. Performance degradation for CAD and HTR implementations (AB primary PLC, GE redundant PLC with medium process tolerance).

high tolerance process level.

While CAD did recover slower for a high tolerance process, it demonstrated improved resiliency in all other metrics and process tolerance levels. With proper PID tuning, it would likely recover faster than HTR at the high tolerance level as its drop in DO level is not as large as HTR. The lower process disruption observed for the CAD implementation compared to the HTR implementation demonstrates the increase in resiliency the active defense technique can provide in a cyber attack scenario. Increasing overall resiliency metrics increases the ICS's ability to deter, defend and restore.

6.3.5 Observations.

Factors such as which PLC served as the redundant or primary controller and the tolerance level of the process impacted resiliency metrics in all three redundancy implementations.

6.3.5.1 Primary vs. Redundant Controller.

- **HAD:** Since the disruption in the process could not be detected for HAD, it is difficult to determine if the redundant controller had an impact on its resiliency metrics.

- **CAD:** All CAD's resiliency metrics were impacted by which PLC served as primary or redundant PLC. These impacts are likely due to the different boot times of the PLCs. The process was uncontrolled for different amount of times depending on which PLC was the redundant controller. This leads to different minimum DO values, time to achieve that minimum DO value and time to recover from that minimum DO value. The Allen-Bradley tended to boot faster than the General Electric, thus the CAD implementation achieved better resiliency metrics while the Allen-Bradley served as the redundant PLC. This impact is shown in Figure 14 where recover time is plotted based on which PLC was the redundant PLC.

- **HTR:** The controller that served as the primary or redundant PLC only affected the recovery time. This is likely due to differences in the PIDs that control the aeration basin's valve to bring the process back to steady state. With the Allen-Bradley serving as the redundant controller, the HTR implementation recovered to steady state faster. If the General Electric's PID was more finely tuned, it would likely perform similarly to the Allen-Bradley PID.

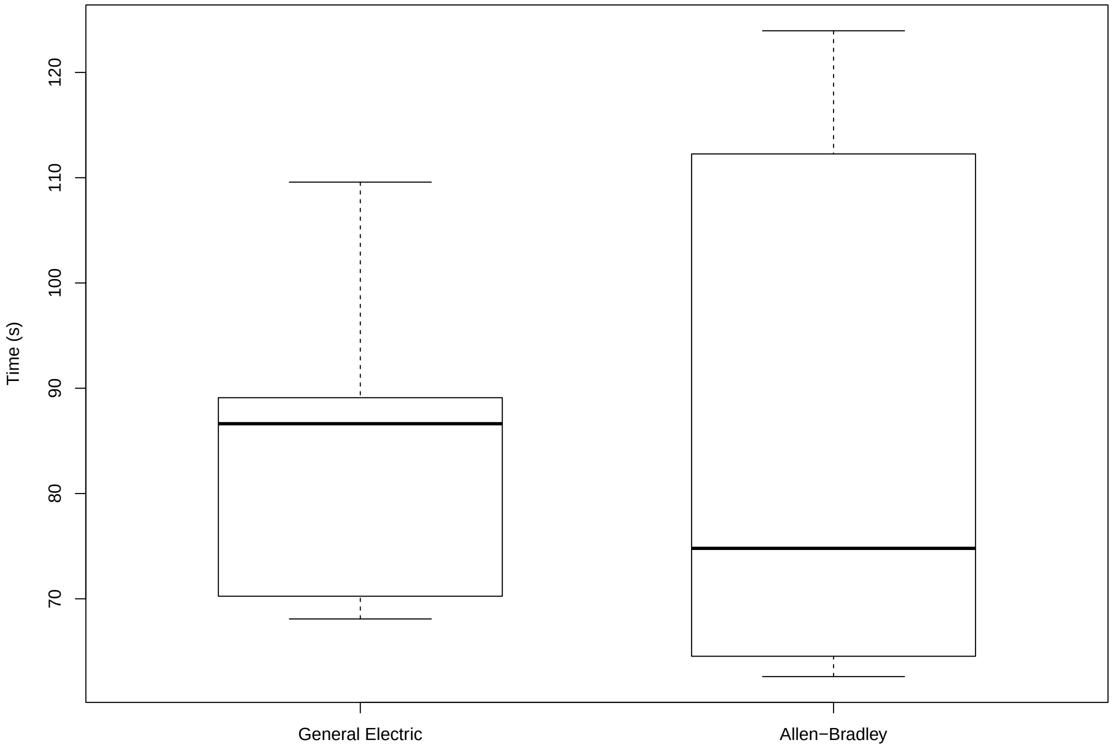


Figure 14. CAD recover time by redundant controller (medium process tolerance).

6.3.5.2 Process Tolerance.

As partially discussed, the process's tolerance to a lack of control also impacted the implementation's resiliency metrics. It was expected that a process of high tolerance would score better in performance degradation and recover time while degradation time would remain constant regardless of process tolerance.

- **HAD:** Since the disruption in the process could not be detected for HAD, it is difficult to determine if the process tolerance had an effect on its resiliency metrics. However, since HAD experienced virtually no detectable process bump at any tolerance level, it is possible switchover and cross loading occurred with enough speed and accuracy for tolerance level to have no impact.

- **CAD:** The performance degradation and degradation time were as expected, but recover time was better for a process with medium tolerance when GE was the redundant controller and low tolerance when Allen-Bradley was the redundant controller. This is most likely due to the PID's configuration that controls the valves in the aeration basin. When GE was the redundant controller, its PID was constantly overshooting the process set point at high tolerance while at low tolerance, the process changed too rapidly making it difficult for the PID to accurately control the process. The medium tolerance level provided enough DO change for the PID to avoid overshooting the set point and a slow enough rate of change that the PID's outputs are timely allowing for accurate control of the process. When Allen-Bradley was the redundant controller, its PID happened to be tuned more accurately for the low tolerance process level, thus it quickly returned to steady state at this tolerance level.

  Figures 15, 16 and 17 show the impact process tolerance had on the resiliency metrics for the CAD implementation with a AB primary PLC and an General-Electric redundant PLC.

- **HTR:** Degradation time and performance degradation behaved as expected, but recover time was nearly identical for high tolerance (81.070 s) and medium tolerance levels (81.620 s). This is likely due to the same reasons the CAD implementation recovered faster at a medium tolerance level when GE was the redundant controller and lower tolerance level when the Allen-Bradley was the redundant controller. HTR's PIDs collectively performed more accurately at the medium tolerance level, allowing HTR to recover in about the same time as the high tolerance process level regardless of the larger drop in DO.

While process tolerance did largely impact resiliency metrics, tailoring the PLC's PID to perform optimally at a given tolerance level can mitigate the observed impacts.

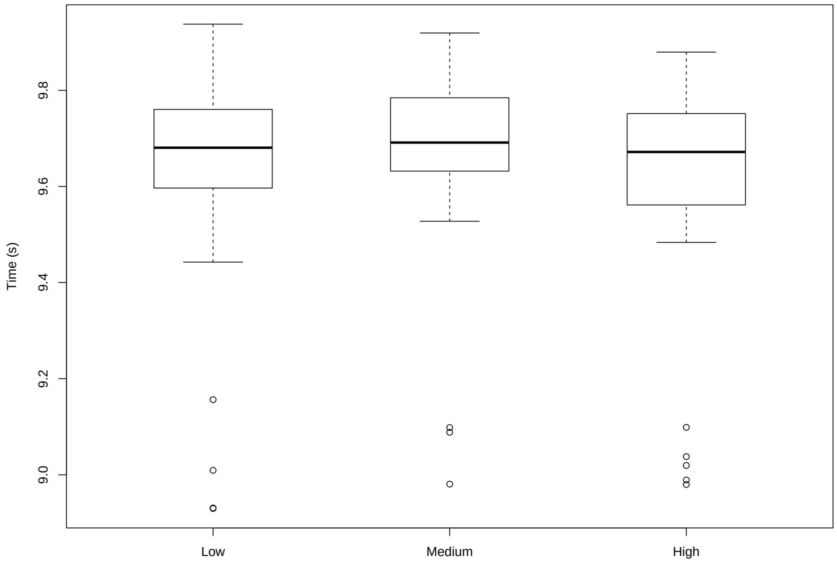


Figure 15. Degradation time by process tolerance (Allen-Bradley primary PLC, GE redundant PLC).

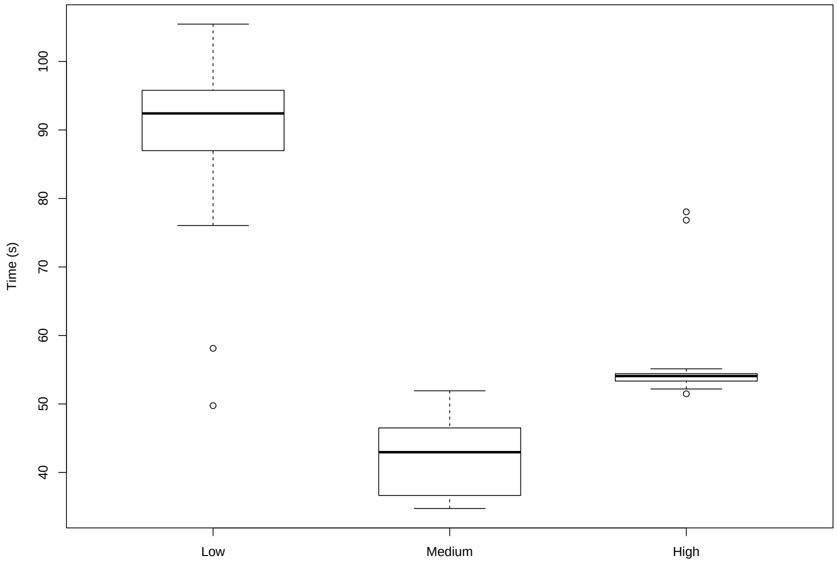


Figure 16. Recover time by process tolerance (Allen-Bradley primary PLC, GE redundant PLC).

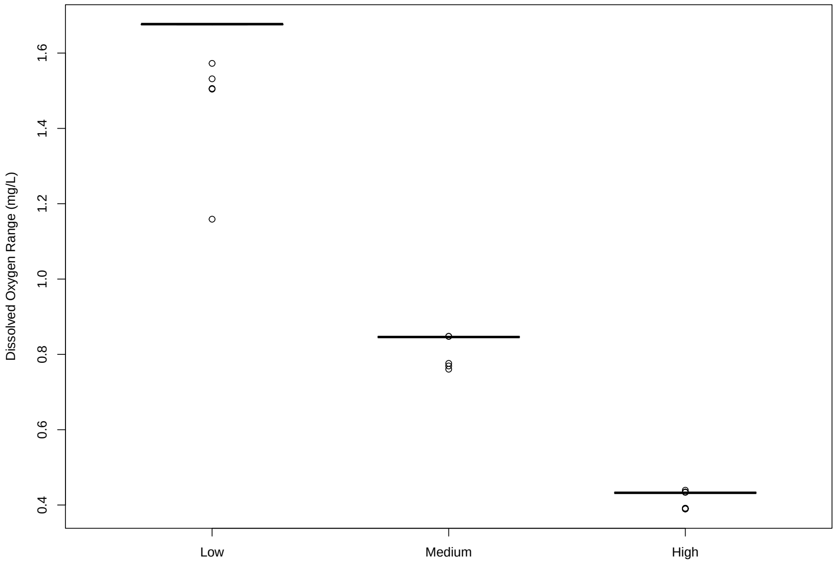


Figure 17. Performance degradation by process tolerance (Allen-Bradley primary PLC, GE redundant PLC).).

VII. Future Work

The active defense technique increased the resiliency of the aeration basin model, but it did so under the assumption that a cyber attack was already detected. An understanding of how to identify cyber attacks on ICS must be developed before the active defense technique should be considered feasible.

The active defense technique's return on investment as a security strategy should also be explored to determine its actual value in risk mitigation.

7.1 Identifying a Cyber Attack

A well-executed cyber attack may not be immediately apparent, thus the speed in which a switchover occurs will ultimately be driven by the time it takes to detect the cyber attack, not how fast the hardware can bring the redundant controller online.

A potential way to detect an attack is to look for abnormalities in process values. Engineers can use Shewhart control charts to identify variations in the process. Variations are typically either common cause or special cause [12]. Common cause variation refers to variation that is expected within the process where special cause refers to variation that is caused by some event [12]. Developing an understanding of both types of variations within the process leads to a more efficient process and potentially larger profits. Watching process data for trends such as a constant increase/decrease or consistent fluctuation above/below the set point can identify a process that is out of control [12]. Identifying these trends early on can minimize the damage caused by a loss of control. However, if the attacker is capable of causing the process to go out of control while convincing the operator and control software that the process is well within limits, detection may become even more difficult.

7.2 Determining the Cost Effectiveness of Active Defense

Ultimately, the decision to employ the proposed active defense technique comes down to its ability to mitigate risk to the ICS in an affordable manner. Security managers and ICS stakeholders need to see an appropriate return on investment for any security strategy. A risk assessment will help quantify the cost to secure the process and this cost will vary based on the type of process. A special purpose device could be designed to reduce implementation costs.

VIII. Conclusion

Through the use of diverse primary and redundant PLCs, the active defense technique proposed in this research reduces the likelihood of a cyber attack common cause failure. The active defense technique was shown to control an aeration basin with minimal deviation from the set point during a cyber attack in hot and cold redundancy configurations compared to a traditional redundancy approach. Due to the decreased process disruption during a cyber attack, the resiliency of the ICS that employed the active defense technique was improved. Improving resiliency through diverse redundancy will help protect our critical infrastructure and perhaps reduce disasters from common cause failures such as the fateful Challenger explosion. Note that the views expressed in this paper are those of the authors and do not reflect the official policy or position of the U.S. Air Force, U.S. Army, U.S. Department of Defense or U.S. Government.

Bibliography

1. Argonne National Laboratory, Moving Target Defense, Cyber Operations, Analysis, and Research, Lemont, Illinois (coar.risc.anl.gov/research/moving-target-defense), 2016.

2. G. Babineau, R. Jones and B. Horowitz, A system-aware cyber security method for shipboard control systems with a method described to evaluate cyber security solutions, *Proceedings of the IEEE International Conference on Technologies for Homeland Security*, pp. 99-104, 2012.

3. C. Davidson, J. Dawson, P. Carsten, M. Yampolskiy and T. Andel, Investigating the applicability of moving target defense for SCADA systems, *Proceedings of the Third International Symposium for ICS and SCADA Cyber Security Research*, pp. 107–110, 2015.

4. J. Downer, When Failure Is an Option: Redundancy, Reliability and Regulation in Complex Technical Systems, London School of Economics, London, United Kingdom (eprints.lse.ac.uk/36537/1/Disspaper53.pdf), 2009.

5. General Electric, PACSystems RX3i Mid-Range Controller, GE Intelligent Platforms Control Systems Solutions, Boston, Massachusetts, 2016.

6. O. Kleineberg, SCADA Security and Fault Tolerance – A Beautiful Pairing! Tofino Security, Lantzville, Canada (www.tofinosecurity.com/blog/scada-security-and-fault-tolerance-beautiful-pairing), 2012.

7. R. Lee, The Sliding Scale of Cyber Security, InfoSec Reading Room, SANS Institute, Bethesda, Maryland (www.sans.org/reading-room/whitepapers/analyst/sliding-scale-cyber-security-36240), 2015.

8. P. Manadhata and J. Wing, An attack surface metric, *IEEE Transactions on Software Engineering*, vol. 37(3), pp. 371–386, 2011.

9. Morphisec Cyber Security Blog, Moving Target Defense: Common Practices, Newton, Massachusetts (blog.morphisec.com/moving-target-defense-common-practices), 2016.

10. National Instruments, Redundant System Basic Concepts, Austin, Texas (www.ni.com/white-paper/6874/en), 2008.

11. Rockwell Automation, Allen-Bradley ControlLogix Redundancy User Manual, Rockwell Automation Publication 1756-UM535E-EN-P, Milwaukee, Wisconsin, 2016.

12. SkyMark, Control Charts, Pittsburgh, Pennsylvania (www.skymark.com/resources/tools/control_charts.asp), 2016.

13. U.S. House of Representatives (Ninety-Ninth Congress, Second Session), Investigation of the Challenger Accident, House Report no. 99–1016, Washington, DC (www.gpo.gov/fdsys/pkg/GPO-CRPT-99hrpt1016/pdf/GPO-CRPT-99hrpt1016.pdf), 1986.

14. G. Van Hermert, Water/wastewater: Achieving the three levels of redundancy, *Control Engineering*, Oak Brook, Illinois (www.controleng.com/single-article/waterwastewater-achieving-the-three-levels-of-redundancy/85beb274f0a4354c1c113f86f487accd.html), 2008.

15. P. Verissimo, A. Bessani, and M. Correia, Designing modular and redundant cyber architectures for process control: Lessons learned, *Proceedings of the Forty-Second Hawaii International Conference on System Sciences*, pp. 1–8, 2009.

16. M. von Sperling, *Biological Wastewater Treatment Series, Volume 2: Basic Principles of Wastewater Treatment*, IWA Publishing, London, United Kingdom, 2007.

17. J. Yoon, Framework for Evaluating the Readiness of Cyber First Responders for Industrial Control Systems, M.S. Thesis, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, 2016.

18. Q. Zhu, D. Wei and K. Ji, Hierarchical architectures of resilient control systems: Concepts, metrics and design principles, in *Cyber Security for Industrial Control Systems: From the Viewpoint of Close-Loop*, P. Cheng, H. Zhang and J. Chen (Eds.), CRC Press, Boca Raton, Florida, pp. 151–182, 2016.

19. R. Zhuang, S. DeLoach and X. Ou, Towards a theory of moving target defense, *Proceedings of the First ACM Workshop on Moving Target Defense*, pp. 31–40, 2014.

Appendix A. Python Experiment Code

This code was used to run the aeration basin model. This code simulates the aeration basin process and executes the experiment by automatically running trials with different experiment factors.

1.1 Y-Box Simulation Code

This code simulates the aeration basin process.

```
import time
import Ybox
import math
import pygbutton
import csv
import sys
import socket
import struct
import os
import string
import binascii
from ENIP import ENIP



#
   ################################################################
```

```
# Author: Evan Plumley and Andrew Chaves
#
# This program is the comminication engine for the Y-box and
    the GUI
# for monitoring and powering the physical wastewater system
#
#
#
   ################################################################



#############################################
# Chaves Initialization Code

#AI 0
#DI 1
#AO 2
#DO 3

#Fan 1 (3,0)
#Fan 0 (3,1)
#Low Left (3,3) ORP
#High Left (2,1) ORP
#Low Right (3,5)
```

```
#High Right (3,4)
#Top(3,6))
#DO 5
#ORP 6



class WwtSim:
def __init__(self):
self._running = True
self.ybox = Ybox.Ybox()



#Initialize Dissolved Oxygen and ORP and send to PLC
DO=2.0
ORP=-20.0

#Convert ORP (-50 to +50 scale) to something the ybox can use
    (0-4095)
ORP=ORP+150.0

#Initial Valve Positions
Aerobic_Valve=50.0
Anaerobic_Valve=50.0
```

```
#Scale for sending DO (Converts 0-7 and 0-100) to ybox values
Scale_DO=math.ceil(4095.0/7.0)
Scale_ORP=math.ceil(4095.0/700.0)
Scale_Valves=math.ceil(4095.0/100.0)

#Scales and casts as an int DO, ORP, and Valves
Send_DO=int(DO*Scale_DO)
Send_ORP=int(ORP*Scale_ORP)
Send_Aerobic_Valve=Scale_Valves*Aerobic_Valve
Send_Anaerobic_Valve=Scale_Valves*Anaerobic_Valve
Fan_1=0
Fan_2=0



#Let it fly
self.ybox.sendAnWrite(2,0,Send_DO)
self.ybox.sendAnWrite(2,1,Send_ORP)
self.ybox.sendAnWrite(2,5,Send_DO)
self.ybox.sendAnWrite(2,7,Send_ORP)
self.ybox.sendAnWrite(2,2,Send_Aerobic_Valve)
self.ybox.sendAnWrite(2,3,Send_Anaerobic_Valve)
self.ybox.sendWrite(3,0,Fan_1)
self.ybox.sendWrite(3,1,Fan_2)
self.ybox.sendWrite(3,2,0)
self.ybox.sendWrite(3,3,0)
self.ybox.sendWrite(3,4,0)
self.ybox.sendWrite(3,5,0)
self.ybox.sendWrite(3,6,0)
self.ybox.sendWrite(3,8,1)
self.ybox.sendWrite(3,9,1)


#What are we sending to the YBOX?
print('DO_Initial:', Send_DO)
print('\nORP_Initial:', Send_ORP)
print('\nSend_Aerobic_Valve_Initial:␣', Send_Aerobic_Valve)
print('\nSend_Anaerobic_Valve_Initial:␣',
    Send_Anaerobic_Valve)

#Tell the infinite loop below what the intialized values are
    ...(Set them as the starting values)
self.Initial_DO=Send_DO
self.Initial_ORP=Send_ORP
Initial_Aerobic_Valve=Send_Aerobic_Valve
Initial_Anaerobic_Valve=Send_Anaerobic_Valve



#method to monitor the PLC and chnage the display accordingly

def timedReads(self,Slope_Up, Slope_Down, Failure_Time,
    Run_Time, PLC, Single_Mode, Offline_Time, Cold_Redundancy)
    :


######NORMAL TIMED READS#########
past = int(round(time.time() * 1000)) #getting starting
    milisecond time to execute reads from the ybox


low=0
medium=0
high=0


if Slope_Up==.113:
low=Slope_Up


elif Slope_Up==.225:
medium=Slope_Up


elif Slope_Up==.45:
high=Slope_Up


if Cold_Redundancy==1 and PLC== 0 and Slope_Up== low:
with open('AB_Cold_Start_Low.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')
elif Cold_Redundancy==1 and PLC == 1 and Slope_Up== low:
with open('GE_Cold_Start_Low.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')

elif Single_Mode==1 and PLC== 0 and Slope_Up== low:
with open('AB_Single_Mode_Low.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')
elif Single_Mode==1 and PLC == 1 and Slope_Up== low:
with open('GE_Single_Mode_Low.csv', 'a') as outputs:
```

```
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')

elif Single_Mode==0 and PLC == 1 and Slope_Up== low:
with open('Dual_Mode_GE_First_Low.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')

elif Single_Mode==0 and PLC == 0 and Slope_Up== low:
with open('Dual_Mode_AB_First_Low.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')

if Cold_Redundancy==1 and PLC== 0 and Slope_Up== medium:
with open('AB_Cold_Start_Medium.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')
elif Cold_Redundancy==1 and PLC == 1 and Slope_Up== medium:
with open('GE_Cold_Start_Medium.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')
```

```
elif Single_Mode==1 and PLC== 0 and Slope_Up== medium:
with open('AB_Single_Mode_Medium.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')
elif Single_Mode==1 and PLC == 1 and Slope_Up== medium:
with open('GE_Single_Mode_Medium.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')

elif Single_Mode==0 and PLC == 1 and Slope_Up== medium:
with open('Dual_Mode_GE_First_Medium.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')

elif Single_Mode==0 and PLC == 0 and Slope_Up== medium:
with open('Dual_Mode_AB_First_Medium.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')

elif Cold_Redundancy==1 and PLC== 0 and Slope_Up== high:
with open('AB_Cold_Start_High.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')
elif Cold_Redundancy==1 and PLC == 1 and Slope_Up== high:
with open('GE_Cold_Start_High.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')

elif Single_Mode==1 and PLC== 0 and Slope_Up== high:
with open('AB_Single_Mode_High.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')
elif Single_Mode==1 and PLC == 1 and Slope_Up== high:
with open('GE_Single_Mode_High.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')

elif Single_Mode==0 and PLC == 1 and Slope_Up== high:
with open('Dual_Mode_GE_First_High.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')

elif Single_Mode==0 and PLC == 0 and Slope_Up== high:
with open('Dual_Mode_AB_First_High.csv', 'a') as outputs:
outputs.write(str('time'))
outputs.write(',')
outputs.write(str('DO'))
outputs.write(',')
outputs.write(str('Valve'))
outputs.write(',')
outputs.write(str('VFD'))
outputs.write('\n')

if PLC==1:
self.ybox.sendWrite(3,7,1)
else:
self.ybox.sendWrite(3,7,0)

#if PLC==0 and Cold_Redundancy==1:
#   self.ybox.sendWrite(3,8,1)
# self.ybox.sendWrite(3,9,0)

#if PLC==1 and Cold_Redundancy==1:
#   self.ybox.sendWrite(3,9,1)
#   self.ybox.sendWrite(3,8,0)

DEBUG=False
fail_bit=0
fail_bit2=0
fail_bit3=0
fail_bit4=0
fail_bit5=0
Run_Time=time.time() + Run_Time
Failure_Time=time.time() + Failure_Time
Offline_Time=Failure_Time + Offline_Time
while True:
```

```
present = int(round(time.time() * 1000)) #getting present
    time to comapre to past
#check to see if 100 milliseconds have  passed
present_display=time.time()
time_show=time.clock()
if present - past >= 100:
past = present

#Read the output (%) of the PIDs controlling the VFD and
    Valves
Valve_Aerobic_Percent_Output=self.ybox.sendRead(0,0) #valve
    percenatge
Valve_Anaerobic_Percent_Output=self.ybox.sendRead(0,1)
Frequency_Percent_Increase=self.ybox.sendRead(0,2)



#Scale valves and VFD speed values to 0-100, why is it 200?
zero_to_hun_scale=100/4095
Frequency_Scale=60/4095#100/4095

#Use the Scale
###############################################
#Display Frequecny sent to the VFP from the PLC
###############################################
Frequency_Percent_Increase_Scaled=(Frequency_Percent_Increase
    *Frequency_Scale) #0-60
```

```
print("PLC␣Freq\n", Frequency_Percent_Increase)

###################################
#Display Aerobic Valve updates
##################################
Valve_Aerobic_Percent_Output_Scaled=int(
    Valve_Aerobic_Percent_Output*zero_to_hun_scale) # Aerobic
    valve open percentage


###################################
#Display Anerobic Valve updates
##################################
Valve_Anaerobic_Percent_Output_Scaled=int(
    Valve_Anaerobic_Percent_Output*zero_to_hun_scale)# =
    Anerobic valve open percetage



####################
####################
####################

###########################
#Output to valves
Aerobic_Valve_Scaled_Display=Valve_Aerobic_Percent_Output
Anaerobic_Valve_Scaled_Display=Valve_Anaerobic_Percent_Output
```

```
self.ybox.sendAnWrite(2,3,Aerobic_Valve_Scaled_Display)
self.ybox.sendAnWrite(2,4,Anaerobic_Valve_Scaled_Display)


################Decrease DO AND ORP#################
#Subtract percentage of valves from 100 and use that to
    determine how much to lower DO and ORP
#The more anaerobic the water, the lower ORP and DO both are
DO_Subtract=(101-Valve_Aerobic_Percent_Output_Scaled)
ORP_Subtract=(101-Valve_Anaerobic_Percent_Output_Scaled)

#Subtract the values and divide by two to make the change
    less agressive (helps the PIDs)
DO_Update=int(self.Initial_DO-(DO_Subtract*Slope_Down))
ORP_Update=int(self.Initial_ORP-(ORP_Subtract*Slope_Down))

DO_Update_Display=DO_Update/585

#Update the previous oxygen value so next time through the
    loop we are subtracting from the most recent correct value
self.Initial_DO=DO_Update
self.Initial_ORP=ORP_Update
```

```
##############GET VFD TRUE SPEED ##################
#Get the VFD's direct output from the Ybox for its speed NOTE
    : the VFD must be running!!!!, not just on!!!
True_Frequency=self.ybox.sendRead(0,2)
#Convert the signal to 0-60 hertz
True_Frequency_Scaled=True_Frequency*(60/4095)



#Multiply the valve percentage by a slope, add this value to
    previous DO and ORP values



DO_Update=self.Initial_DO+(Slope_Up*
    Valve_Aerobic_Percent_Output_Scaled)
ORP_Update=self.Initial_ORP+(Slope_Up*
    Valve_Anaerobic_Percent_Output_Scaled)

################INCREASE DO AND ORP#################

#Increase DO and ORP if valves are opening according to
    equation
```

```
#Update the previous DO and ORP values so next time through
    the loop we are adding to the correct value
self.Initial_DO=DO_Update
self.Initial_ORP=ORP_Update

#Cast as an integer for the YBOX's sake
DO_Update=int(DO_Update)
ORP_Update=int(ORP_Update)



#Show what is being sent
print('\nDO_Update:␣', DO_Update)
print('\nORP_Update:␣', ORP_Update)



#let DO and ORP fly
self.ybox.sendAnWrite(2,0,DO_Update)
self.ybox.sendAnWrite(2,1,ORP_Update)
self.ybox.sendAnWrite(2,5,DO_Update)
self.ybox.sendAnWrite(2,7,ORP_Update)



if Cold_Redundancy==1 and PLC== 0 and Slope_Up== low:
with open('AB_Cold_Start_Low.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')

elif Cold_Redundancy==1 and PLC == 1 and Slope_Up== low:
with open('GE_Cold_Start_Low.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')

elif Single_Mode==1 and PLC== 0 and Slope_Up== low:
with open('AB_Single_Mode_Low.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')
elif Single_Mode==1 and PLC== 1 and Slope_Up== low:
with open('GE_Single_Mode_Low.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')
elif Single_Mode==0 and PLC== 0 and Slope_Up== low:
with open('Dual_Mode_AB_First_Low.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')

elif Single_Mode==0 and PLC== 1 and Slope_Up== low:
with open('Dual_Mode_GE_First_Low.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')

elif Cold_Redundancy==1 and PLC== 0 and Slope_Up== medium:
with open('AB_Cold_Start_Medium.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')
elif Cold_Redundancy==1 and PLC == 1 and Slope_Up== medium:
with open('GE_Cold_Start_Medium.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')

elif Single_Mode==1 and PLC== 0 and Slope_Up== medium:
with open('AB_Single_Mode_Medium.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')
elif Single_Mode==1 and PLC== 1 and Slope_Up== medium:
with open('GE_Single_Mode_Medium.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')
elif Single_Mode==0 and PLC== 0 and Slope_Up== medium:
with open('Dual_Mode_AB_First_Medium.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')

elif Single_Mode==0 and PLC== 1 and Slope_Up== medium:
with open('Dual_Mode_GE_First_Medium.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')

elif Cold_Redundancy==1 and PLC== 0 and Slope_Up== high:
with open('AB_Cold_Start_High.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')
elif Cold_Redundancy==1 and PLC == 1 and Slope_Up== high:
with open('GE_Cold_Start_High.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')

elif Single_Mode==1 and PLC== 0 and Slope_Up== high:
with open('AB_Single_Mode_High.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')
elif Single_Mode==1 and PLC== 1 and Slope_Up== high:
with open('GE_Single_Mode_High.csv', 'a') as outputs:
```

```
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')
elif Single_Mode==0 and PLC== 0 and Slope_Up== high:
with open('Dual_Mode_AB_First_High.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')

elif Single_Mode==0 and PLC== 1 and Slope_Up== high:
with open('Dual_Mode_GE_First_High.csv', 'a') as outputs:
outputs.write(str(time_show))
outputs.write(',')
outputs.write(str(DO_Update_Display))
outputs.write(',')
outputs.write(str(Valve_Aerobic_Percent_Output_Scaled))
outputs.write(',')
outputs.write(str(True_Frequency_Scaled))
outputs.write('\n')









print("True_Frequency_Raw", True_Frequency)





#Print the true VFD frequency
print('\nTrue_VFD_Frequency:␣', True_Frequency_Scaled)

#Start fans if the VFD is runnning
if True_Frequency_Scaled > 5:
self.ybox.sendWrite(3,0,1)
self.ybox.sendWrite(3,1,1)
else:
self.ybox.sendWrite(3,0,0)
self.ybox.sendWrite(3,1,0)

#Print divider for next time through loop
```

```
print('#########################################\n')

##########Obtain Alarms, check alarms, send to light if
    necessary#######
DO_High=self.ybox.sendRead(1,0)
if DO_High==1:
self.ybox.sendWrite(3,5,1)
else:
self.ybox.sendWrite(3,5,0)

DO_Low=self.ybox.sendRead(1,1)
if DO_Low==1:
self.ybox.sendWrite(3,4,1)
else:
self.ybox.sendWrite(3,4,0)



ORP_High=self.ybox.sendRead(1,5)
if ORP_High==1:
self.ybox.sendWrite(3,2,1)
else:
self.ybox.sendWrite(3,2,0)



ORP_Low=self.ybox.sendRead(1,3)
if ORP_Low==1:
```

```
self.ybox.sendWrite(3,3,1)
else:
self.ybox.sendWrite(3,3,0)



if present_display>Failure_Time:
if fail_bit==0 and Single_Mode==1 and PLC== 0 and
    Cold_Redundancy==0 and Slope_Up==low:
with open('AB_Single_Mode_Low.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==1 and PLC== 1 and
    Cold_Redundancy==0 and Slope_Up==low:
with open('GE_Single_Mode_Low.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==0 and PLC== 0 and
    Cold_Redundancy==0 and Slope_Up==low:
with open('Dual_Mode_AB_First_Low.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==0 and PLC== 1 and
    Cold_Redundancy==0 and Slope_Up==low:
with open('Dual_Mode_GE_First_Low.csv', 'a') as outputs:
```

```
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==0 and PLC== 1 and
    Cold_Redundancy==1 and Slope_Up==low:
with open('GE_Cold_Start_Low.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1
```

```
if fail_bit==0 and Single_Mode==0 and PLC== 0 and
    Cold_Redundancy==1 and Slope_Up==low:
with open('AB_Cold_Start_Low.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1



if fail_bit==0 and Single_Mode==1 and PLC== 0 and
    Cold_Redundancy==0 and Slope_Up==medium:
with open('AB_Single_Mode_Medium.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==1 and PLC== 1 and
    Cold_Redundancy==0 and Slope_Up==medium:
with open('GE_Single_Mode_Medium.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==0 and PLC== 0 and
    Cold_Redundancy==0 and Slope_Up==medium:
with open('Dual_Mode_AB_First_Medium.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==0 and PLC== 1 and
    Cold_Redundancy==0 and Slope_Up==medium:
with open('Dual_Mode_GE_First_Medium.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==0 and PLC== 1 and
    Cold_Redundancy==1 and Slope_Up==medium:
with open('GE_Cold_Start_Medium.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==0 and PLC== 0 and
    Cold_Redundancy==1 and Slope_Up==medium:
with open('AB_Cold_Start_Medium.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1



if fail_bit==0 and Single_Mode==1 and PLC== 0 and
    Cold_Redundancy==0 and Slope_Up==high:
```

```
with open('AB_Single_Mode_High.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==1 and PLC== 1 and
    Cold_Redundancy==0 and Slope_Up==high:
with open('GE_Single_Mode_High.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1
```

```
if fail_bit==0 and Single_Mode==0 and PLC== 0 and
    Cold_Redundancy==0 and Slope_Up==high:
with open('Dual_Mode_AB_First_High.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==0 and PLC== 1 and
    Cold_Redundancy==0 and Slope_Up==high:
with open('Dual_Mode_GE_First_High.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')
```

```
fail_bit=1

if fail_bit==0 and Single_Mode==0 and PLC== 1 and
    Cold_Redundancy==1 and Slope_Up==high:
with open('GE_Cold_Start_High.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')

fail_bit=1

if fail_bit==0 and Single_Mode==0 and PLC== 0 and
    Cold_Redundancy==1 and Slope_Up==high:
with open('AB_Cold_Start_High.csv', 'a') as outputs:
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write(',')
outputs.write(str('Fail/Switch'))
outputs.write('\n')


fail_bit=1


if Single_Mode==1 and fail_bit2==0:


####################GE DISABLE################
#Disable GE PLC
packets = []
with open('disable_packets.txt', 'r') as inp:
for line in inp:
li = line.strip()
li = binascii.unhexlify(li)


packets.append(li)


if not DEBUG:
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.108.208', 18245))


for p in packets:
if not DEBUG:
sock.send(p)
time.sleep(0.001)
else:
print(binascii.hexlify(li))
```

```
#print sock.recv(1000).encode('hex')
#time.sleep(0.05)

#####################AB DISABLE#################
e = ENIP()
e.connect('192.168.108.210', 44818, 1)

e.setToProg()

#e.setToRun()

e.close()

fail_bit2=1

if present_display > Offline_Time and fail_bit2==1 and
    fail_bit3==0:

######################GE ENABLE######################
packets = []
with open('enable_packets.txt', 'r') as inp:
for line in inp:
li = line.strip()
li = binascii.unhexlify(li)

packets.append(li)
```

```
if not DEBUG:
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.108.208', 18245))

for p in packets:
if not DEBUG:
sock.send(p)
time.sleep(0.001)
else:
print(binascii.hexlify(li))
#print sock.recv(1000).encode('hex')
#time.sleep(0.05)

########################AB ENABLE####################
e = ENIP()
e.connect('192.168.108.210', 44818, 1)

#e.setToProg()

e.setToRun()

e.close()

fail_bit3=1
```

```
if PLC==0 and Single_Mode==0 and Cold_Redundancy==0 and
    fail_bit4==0:

#Disable AB
e = ENIP()
e.connect('192.168.108.210', 44818, 1)

e.setToProg()

#e.setToRun()

e.close()

fail_bit4=1

#Switch to GE
self.ybox.sendWrite(3,7,1)




if PLC==1 and Single_Mode==0 and Cold_Redundancy==0 and
    fail_bit4==0:

#Disable GE PLC
```

```
packets = []
with open('disable_packets.txt', 'r') as inp:
for line in inp:
li = line.strip()
li = binascii.unhexlify(li)

packets.append(li)

if not DEBUG:
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.108.208', 18245))

for p in packets:
if not DEBUG:
sock.send(p)
time.sleep(0.001)
else:
print(binascii.hexlify(li))
#print sock.recv(1000).encode('hex')
#time.sleep(0.05)
sock.close()

#Switch to AB
self.ybox.sendWrite(3,7,0)
fail_bit4=1
```

```
if Cold_Redundancy==1 and PLC==0 and fail_bit5==0:

#Turn Off GE
self.ybox.sendWrite(3,9,0)

#Sleep

time.sleep(2.5)

#Disable AB
e = ENIP()
e.connect(’192.168.108.210’, 44818, 1)
e.setToProg()
#e.setToRun()
e.close()

#Switch to GE
self.ybox.sendWrite(3,7,1)

#Boot up GE
self.ybox.sendWrite(3,9,1)

fail_bit5=1

if Cold_Redundancy==1 and PLC==1 and fail_bit5==0:
```

```
self.ybox.sendWrite(3,8,0)

#Sleep

time.sleep(2.5)

packets = []
with open('disable_packets.txt', 'r') as inp:
for line in inp:
li = line.strip()
li = binascii.unhexlify(li)

packets.append(li)

if not DEBUG:
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.108.208', 18245))

for p in packets:
if not DEBUG:
sock.send(p)
time.sleep(0.001)
else:
print(binascii.hexlify(li))
#print sock.recv(1000).encode('hex')
```

```
#time.sleep(0.05)

self.ybox.sendWrite(3,7,0)
self.ybox.sendWrite(3,8,1)
sock.close()
fail_bit5=1

if present_display>Run_Time:
break

#Close the serial connection here
#self.ybox.close()
#wwtSim = WwtSim()

def Init_2(self):
DEBUG=False
DO=2.0
ORP=-20.0

#Convert ORP (-50 to +50 scale) to something the ybox can use
    (0-4095)
ORP=ORP+150.0

#Initial Valve Positions
Aerobic_Valve=50.0
Anaerobic_Valve=50.0
```

```
#Scale for sending DO (Converts 0-7 and 0-100) to ybox values
Scale_DO=math.ceil(4095.0/7.0)
Scale_ORP=math.ceil(4095.0/700.0)
Scale_Valves=math.ceil(4095.0/100.0)

#Scales and casts as an int DO, ORP, and Valves
Send_DO=int(DO*Scale_DO)
Send_ORP=int(ORP*Scale_ORP)
Send_Aerobic_Valve=Scale_Valves*Aerobic_Valve
Send_Anaerobic_Valve=Scale_Valves*Anaerobic_Valve
Fan_1=0
Fan_2=0



#Let it fly
self.ybox.sendAnWrite(2,0,Send_DO)
self.ybox.sendAnWrite(2,1,Send_ORP)
self.ybox.sendAnWrite(2,5,Send_DO)
self.ybox.sendAnWrite(2,7,Send_ORP)
self.ybox.sendAnWrite(2,2,Send_Aerobic_Valve)
self.ybox.sendAnWrite(2,3,Send_Anaerobic_Valve)
self.ybox.sendWrite(3,0,Fan_1)
self.ybox.sendWrite(3,1,Fan_2)
self.ybox.sendWrite(3,2,0)
self.ybox.sendWrite(3,3,0)
self.ybox.sendWrite(3,4,0)
self.ybox.sendWrite(3,5,0)
self.ybox.sendWrite(3,6,0)
self.ybox.sendWrite(3,8,1)
self.ybox.sendWrite(3,9,1)


#What are we sending to the YBOX?
print('DO_Initial:', Send_DO)
print('\nORP_Initial:', Send_ORP)
print('\nSend_Aerobic_Valve_Initial:␣', Send_Aerobic_Valve)
print('\nSend_Anaerobic_Valve_Initial:␣',
   Send_Anaerobic_Valve)

#Tell the infinite loop below what the intialized values are
   ...(Set them as the starting values)
self.Initial_DO=Send_DO
self.Initial_ORP=Send_ORP
Initial_Aerobic_Valve=Send_Aerobic_Valve
Initial_Anaerobic_Valve=Send_Anaerobic_Valve

#Disable GE


packets = []
with open('disable_packets.txt', 'r') as inp:
```

```
for line in inp:
li = line.strip()
li = binascii.unhexlify(li)
packets.append(li)

if not DEBUG:
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.108.208', 18245))
print("here")

for p in packets:
if not DEBUG:
sock.send(p)
time.sleep(0.001)
else:
print(binascii.hexlify(li))
#print sock.recv(1000).encode('hex')
#time.sleep(0.05)
time.sleep(2)

#Enable GE
packets = []
with open('enable_packets.txt', 'r') as inp:
for line in inp:
li = line.strip()
li = binascii.unhexlify(li)
```

```
packets.append(li)

if not DEBUG:
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((’192.168.108.208’, 18245))
print(”here”)

for p in packets:
if not DEBUG:
sock.send(p)
time.sleep(0.001)
else:
print(binascii.hexlify(li))
#print sock.recv(1000).encode(’hex’)
#time.sleep(0.05)



#Disable AB

e = ENIP()
e.connect(’192.168.108.210’, 44818, 1)
e.setToProg()
#e.setToRun()
e.close()

time.sleep(2)
```

```
#Enable AB

e = ENIP()
e.connect('192.168.108.210', 44818, 1)
#e.setToProg()
e.setToRun()
e.close()
```

1.2 Experiment Automation Script

This code runs the experiment, running trials defined by different experiment factors.

```
import time
import msvcrt
from pygame.locals import *
import serial
import csv
import Main_High_Speed_Three_Disable
import Ybox
import string

#
    #######################################################################################

# Author: Evan Plumley and Andrew Chaves
# Date: 6/20/2016
# version: 1.0
#
#The Script Caller class
# dependencies: pygame, pyserial
#
    ################################################################################

def call():
#GE=1
#AB=0
#def timedReads(self,Slope_Up, Slope_Down, Failure_Time,
    Run_Time, PLC, Single_Mode, Offline_Time, Cold_Redundancy)
    :

thang = Main_High_Speed_Three_Disable.WwtSim()



with open('Experiment_Inputs.txt', 'r') as inp:
for line in inp:
li = line.strip()
split=li.split("␣")
print (str(split))
current_time=time.clock()
with open('Experiment_Inputs_Track.txt', 'a') as out:
out.write(str(split))
out.write(str('␣'))
out.write(str(current_time))
out.write('\n')
Main_High_Speed_Three_Disable.Init_2(thang)
Main_High_Speed_Three_Disable.timedReads(thang, float(split
    [0]),float(split[1]),float(split[2]),int(split[3]),int(
    split[4]),int(split[5]),int(split[6]),int(split[7]))


#def timedReads(self,Slope_Up, Slope_Down, Failure_Time,
    Run_Time, PLC, Single_Mode, Offline_Time, Cold_Redundancy)
    :

#Main_High_Speed_Three.Init_2(thang)
#Main_High_Speed_Three.timedReads(thang
    ,.075,.1,25,60,1,1,10,0)

call()
```

1.3 Switchover Speed Test

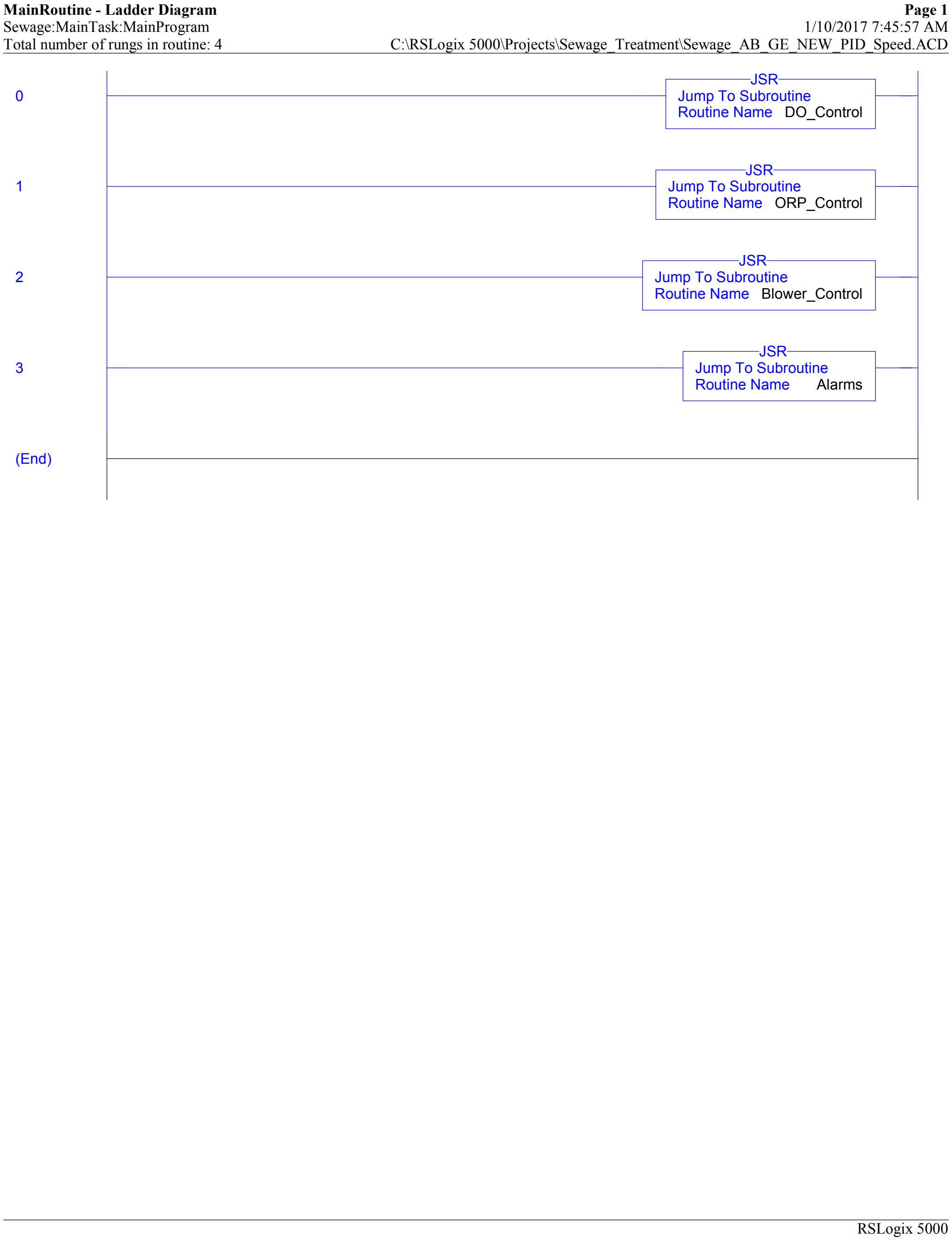
```
import time
import Ybox
import math
import pygbutton
import csv
import sys
import socket
import struct
import os
import string
import binascii
from ENIP import ENIP



#
    #######################################################################

# Author: Evan Plumley and Andrew Chaves
#
# This program is the comminication engine for the Y-box and
    the GUI
# for monitoring and powering the physical wastewater system
#
#
#
    #######################################################################



##################################################
# Chaves Initialization Code
```

```
#AI 0
#DI 1
#AO 2
#DO 3

#Fan 1 (3,0)
#Fan 0 (3,1)
#Low Left (3,3) ORP
#High Left (2,1) ORP
#Low Right (3,5)
#High Right (3,4)
#Top(3,6))
#DO 5
#ORP 6



class WwtSim:
def __init__(self):
self._running = True
self.ybox = Ybox.Ybox()

def timing(self):
i=0
while i<=30:
```

```
Switch=0
self.ybox.sendWrite(3,7,1)
start=time.time()
self.ybox.sendWrite(3,7,0)


while Switch!=1:
Switch=self.ybox.sendRead(1,3)

stop=time.time()
difference=stop-start
print (start)
print (stop)
print (difference)

with open('Timing.csv', 'a') as outputs:
outputs.write(str(difference))
outputs.write('\n')
i=i+1

andrew=WwtSim()
andrew.timing()
```

Appendix B. Allen-Bradley Ladder Logic

This logic is executed on the Allen-Bradley PLC, allowing it to control the aeration basin.

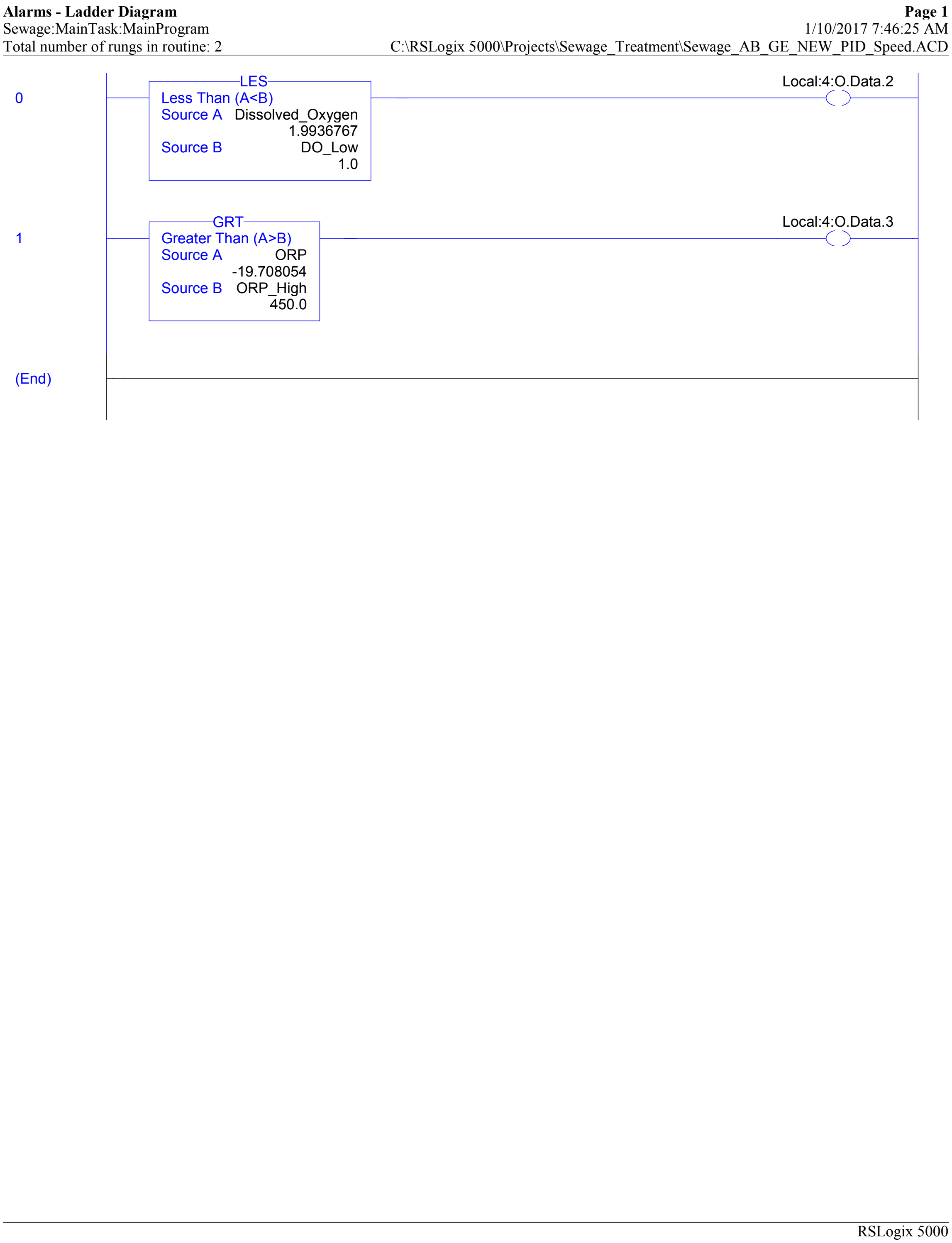


MainRoutine - Ladder Diagram
Sewage:MainTask:MainProgram
Total number of rungs in routine: 4

Page 1
1/10/2017 7:45:57 AM
C:\RSLogix 5000\Projects\Sewage_Treatment\Sewage_AB_GE_NEW_PID_Speed.ACD

RSLogix 5000

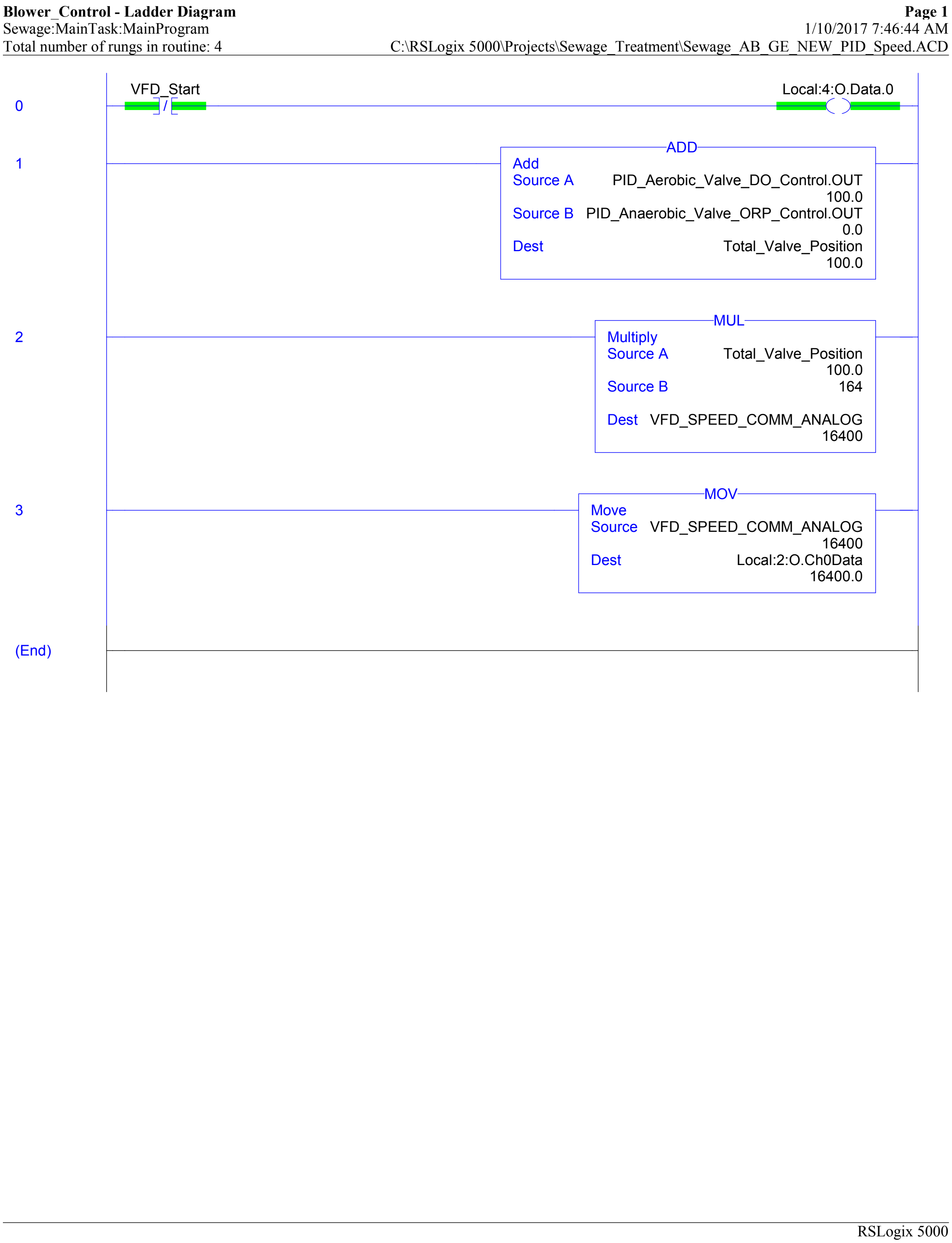
Figure 18. Main-calls all sub-functions for aeration basin.



Alarms - Ladder Diagram **Page 1**
Sewage:MainTask:MainProgram 1/10/2017 7:46:25 AM
Total number of rungs in routine: 2 C:\RSLogix 5000\Projects\Sewage_Treatment\Sewage_AB_GE_NEW_PID_Speed.ACD

RSLogix 5000

Figure 19. Alarms-sounds alarms if DO or ORP drop below or go above set point.

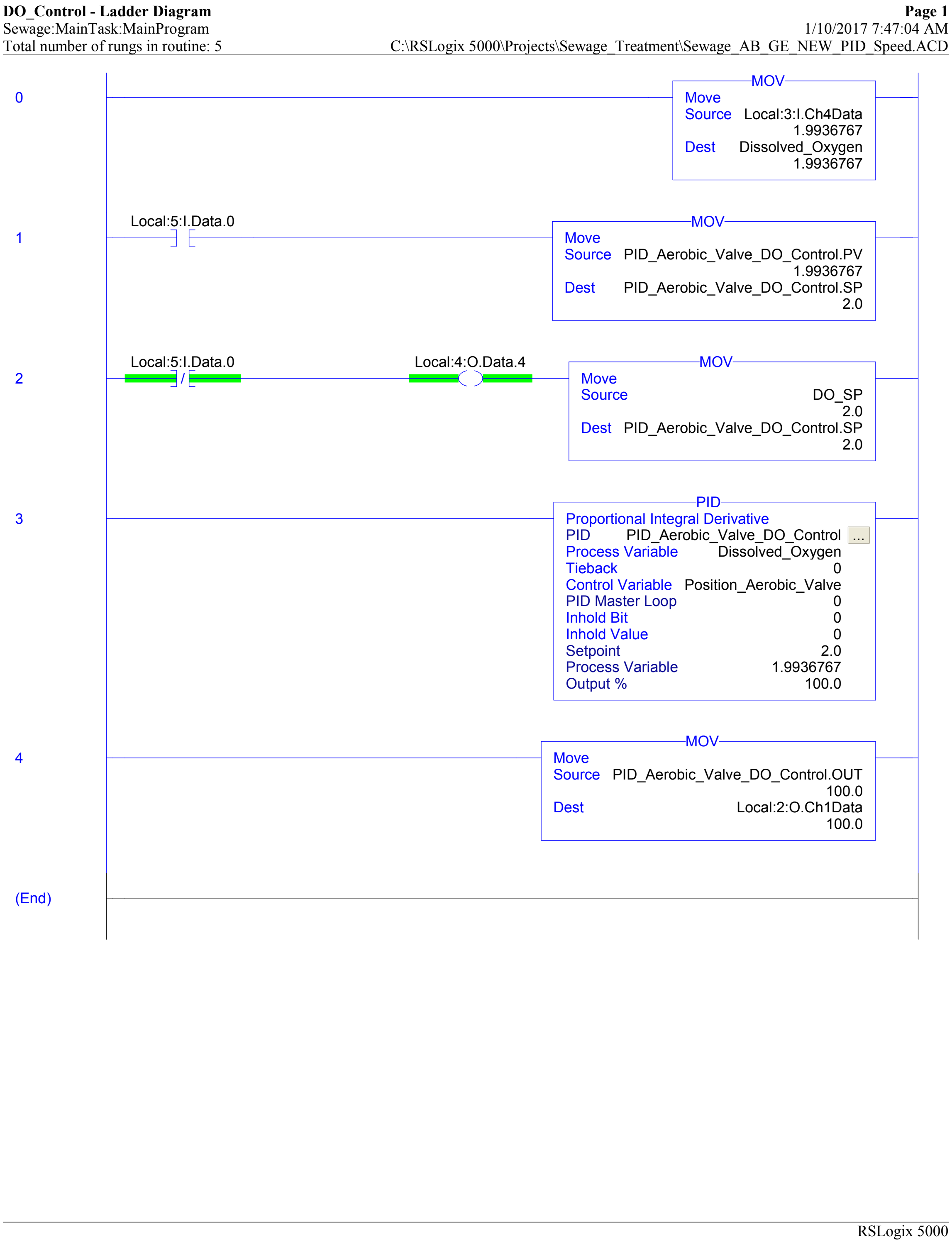


Blower_Control - Ladder Diagram **Page 1**
Sewage:MainTask:MainProgram 1/10/2017 7:46:44 AM
Total number of rungs in routine: 4 C:\RSLogix 5000\Projects\Sewage_Treatment\Sewage_AB_GE_NEW_PID_Speed.ACD

```
0      VFD_Start                                              Local:4:O.Data.0
       ]/[                                                          ( )

1                                       ADD
                          Add
                          Source A      PID_Aerobic_Valve_DO_Control.OUT
                                                                   100.0
                          Source B   PID_Anaerobic_Valve_ORP_Control.OUT
                                                                     0.0
                          Dest                      Total_Valve_Position
                                                                   100.0

2                                       MUL
                          Multiply
                          Source A                  Total_Valve_Position
                                                                   100.0
                          Source B                                   164

                          Dest                   VFD_SPEED_COMM_ANALOG
                                                                   16400

3                                       MOV
                          Move
                          Source                 VFD_SPEED_COMM_ANALOG
                                                                   16400
                          Dest                         Local:2:O.Ch0Data
                                                                 16400.0

(End)
```

RSLogix 5000

Figure 20. Blower-increases or decreases blower speed based on valve.

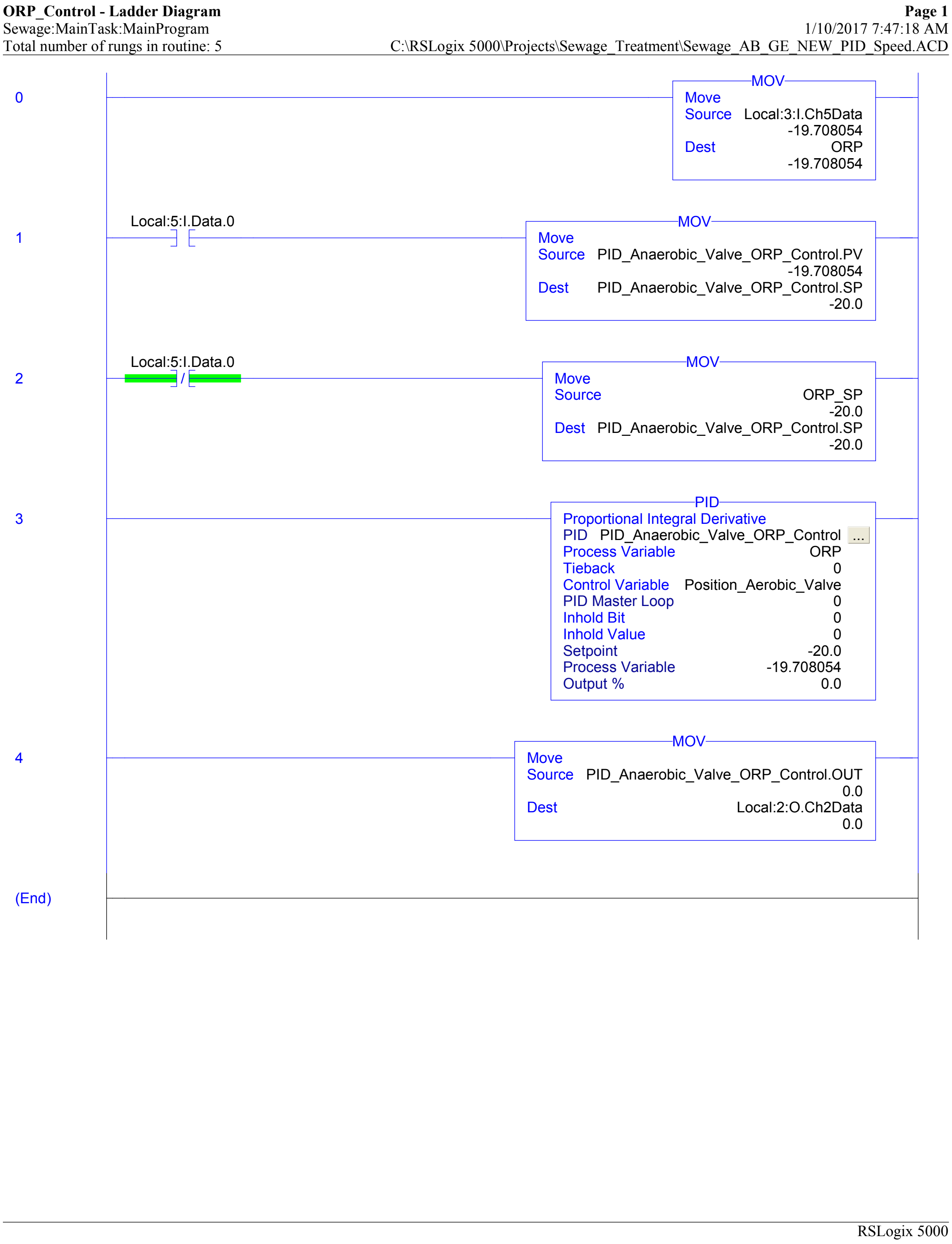


DO_Control - Ladder Diagram **Page 1**
Sewage:MainTask:MainProgram 1/10/2017 7:47:04 AM
Total number of rungs in routine: 5 C:\RSLogix 5000\Projects\Sewage_Treatment\Sewage_AB_GE_NEW_PID_Speed.ACD

```
0    MOV
     Move
     Source   Local:3:I.Ch4Data
                     1.9936767
     Dest     Dissolved_Oxygen
                     1.9936767

1    Local:5:I.Data.0
     ] [
     MOV
     Move
     Source   PID_Aerobic_Valve_DO_Control.PV
                     1.9936767
     Dest     PID_Aerobic_Valve_DO_Control.SP
                     2.0

2    Local:5:I.Data.0        Local:4:O.Data.4
     ]/[                     ( )
     MOV
     Move
     Source                  DO_SP
                               2.0
     Dest     PID_Aerobic_Valve_DO_Control.SP
                               2.0

3    PID
     Proportional Integral Derivative
     PID             PID_Aerobic_Valve_DO_Control  ...
     Process Variable            Dissolved_Oxygen
     Tieback                                    0
     Control Variable     Position_Aerobic_Valve
     PID Master Loop                            0
     Inhold Bit                                 0
     Inhold Value                               0
     Setpoint                                 2.0
     Process Variable                   1.9936767
     Output %                               100.0

4    MOV
     Move
     Source   PID_Aerobic_Valve_DO_Control.OUT
                                  100.0
     Dest               Local:2:O.Ch1Data
                                  100.0

(End)
```

RSLogix 5000

Figure 21. DO control-opens aerobic valve to increase or decrease DO and maintain set point.



ORP_Control - Ladder Diagram
Sewage:MainTask:MainProgram
Total number of rungs in routine: 5

Page 1
1/10/2017 7:47:18 AM
C:\RSLogix 5000\Projects\Sewage_Treatment\Sewage_AB_GE_NEW_PID_Speed.ACD

0
MOV
Move
Source Local:3:I.Ch5Data
-19.708054
Dest ORP
-19.708054

1
Local:5:I.Data.0
MOV
Move
Source PID_Anaerobic_Valve_ORP_Control.PV
-19.708054
Dest PID_Anaerobic_Valve_ORP_Control.SP
-20.0

2
Local:5:I.Data.0
/
MOV
Move
Source ORP_SP
-20.0
Dest PID_Anaerobic_Valve_ORP_Control.SP
-20.0

3
PID
Proportional Integral Derivative
PID PID_Anaerobic_Valve_ORP_Control ...
Process Variable ORP
Tieback 0
Control Variable Position_Aerobic_Valve
PID Master Loop 0
Inhold Bit 0
Inhold Value 0
Setpoint -20.0
Process Variable -19.708054
Output % 0.0

4
MOV
Move
Source PID_Anaerobic_Valve_ORP_Control.OUT
0.0
Dest Local:2:O.Ch2Data
0.0

(End)

RSLogix 5000

Figure 22. ORP control-opens anaerobic valve to increase or decrease ORP and maintain set point.

Appendix C. General Electric Ladder Logic

This logic is executed on the General Electric PLC, allowing it to control the aeration basin.

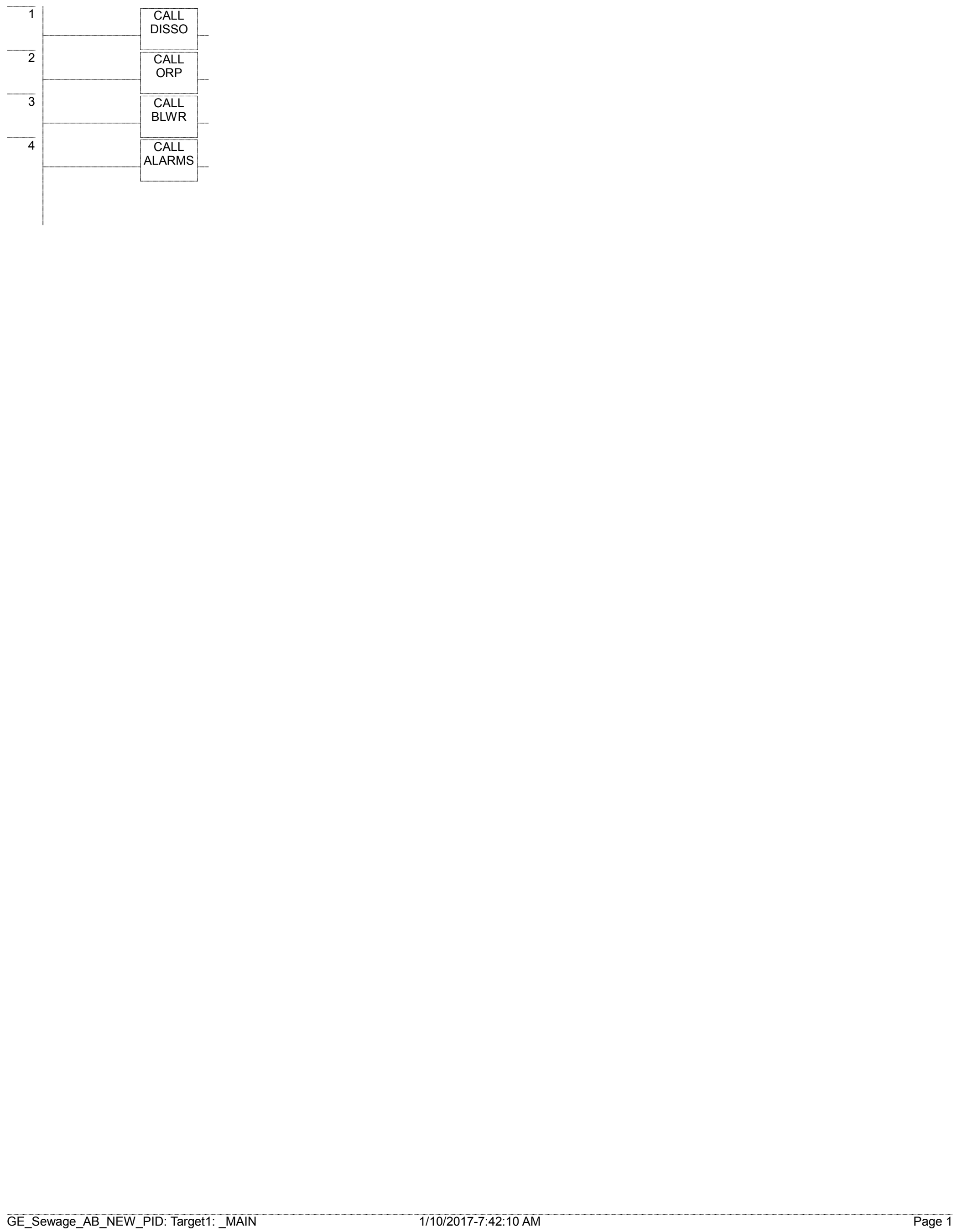


Figure 23. Main-calls all sub-functions for aeration basin.

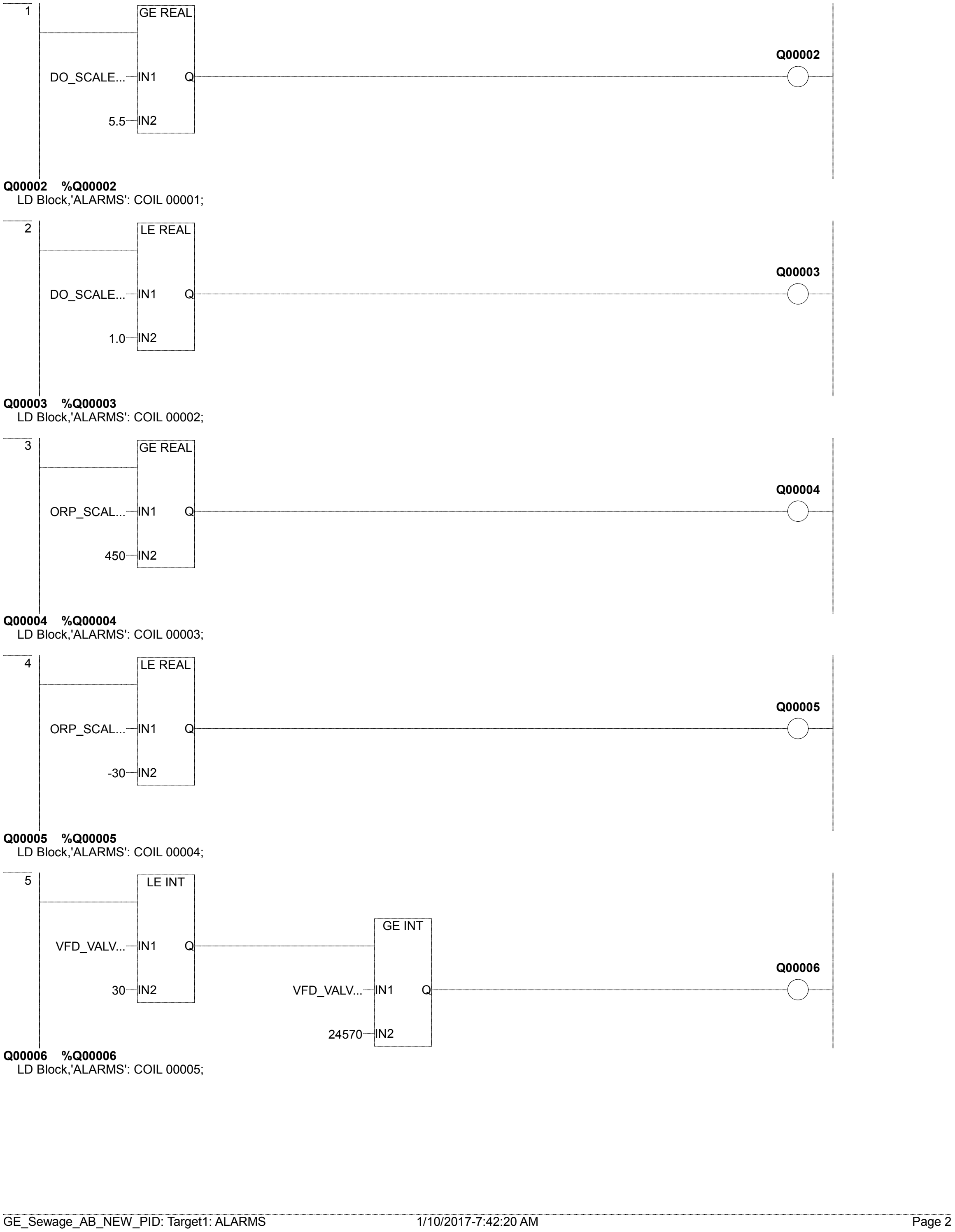
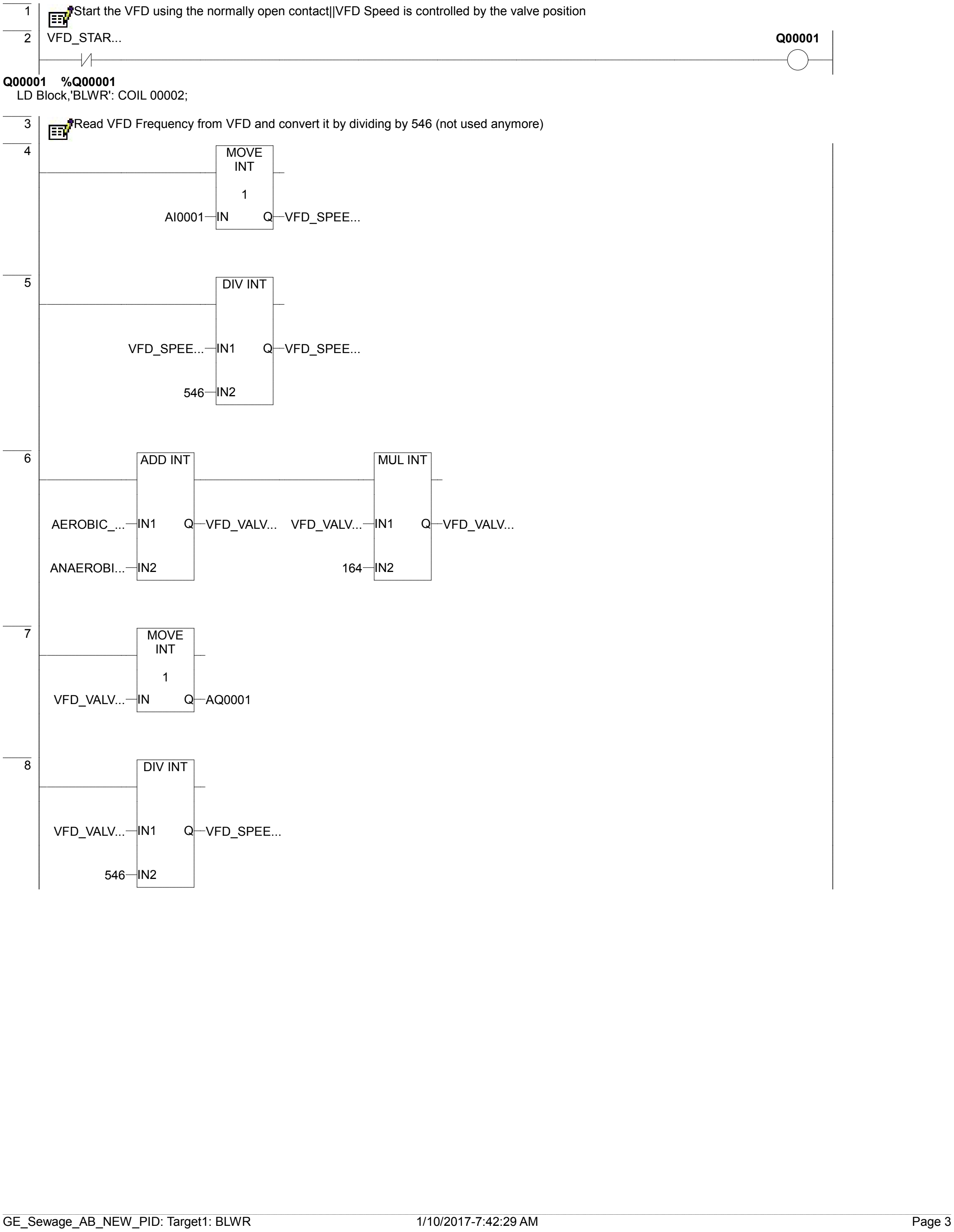


Figure 24. Alarms-sounds alarms if DO or ORP drop below or go above set point.



Start the VFD using the normally open contact||VFD Speed is controlled by the valve position

Q00001 %Q00001
LD Block,'BLWR': COIL 00002;

Read VFD Frequency from VFD and convert it by dividing by 546 (not used anymore)

GE_Sewage_AB_NEW_PID: Target1: BLWR 1/10/2017-7:42:29 AM Page 3

Figure 25. Blower-increases or decreases blower speed based on valve.

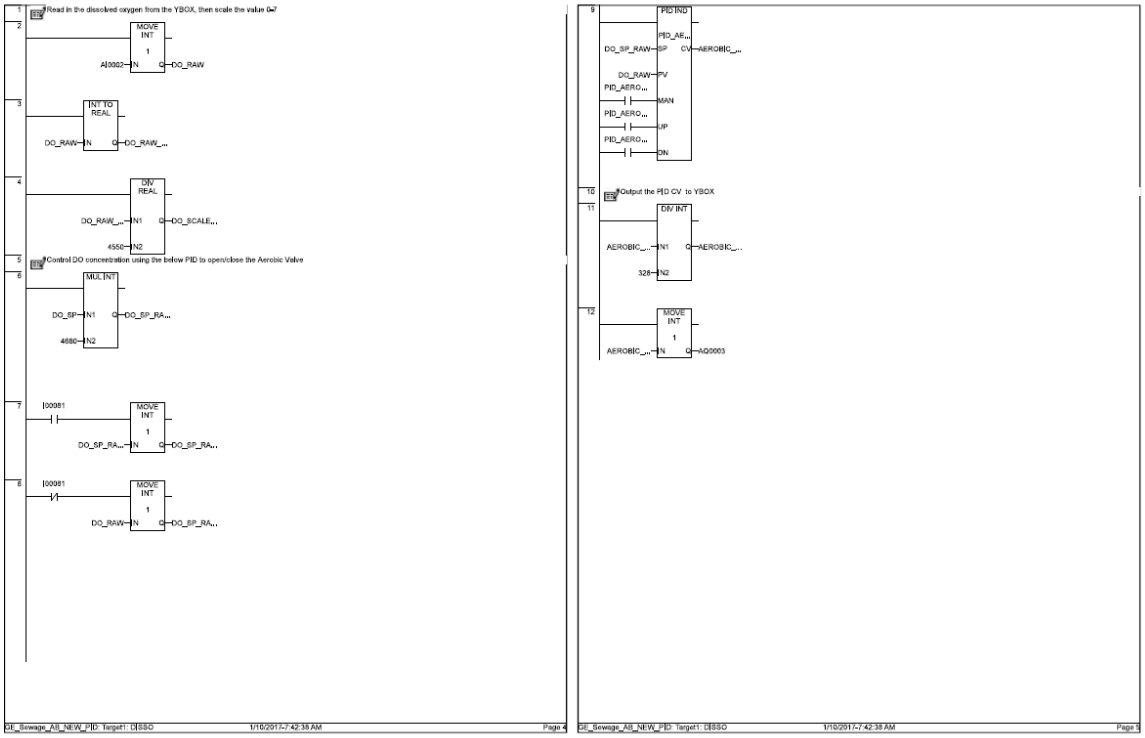
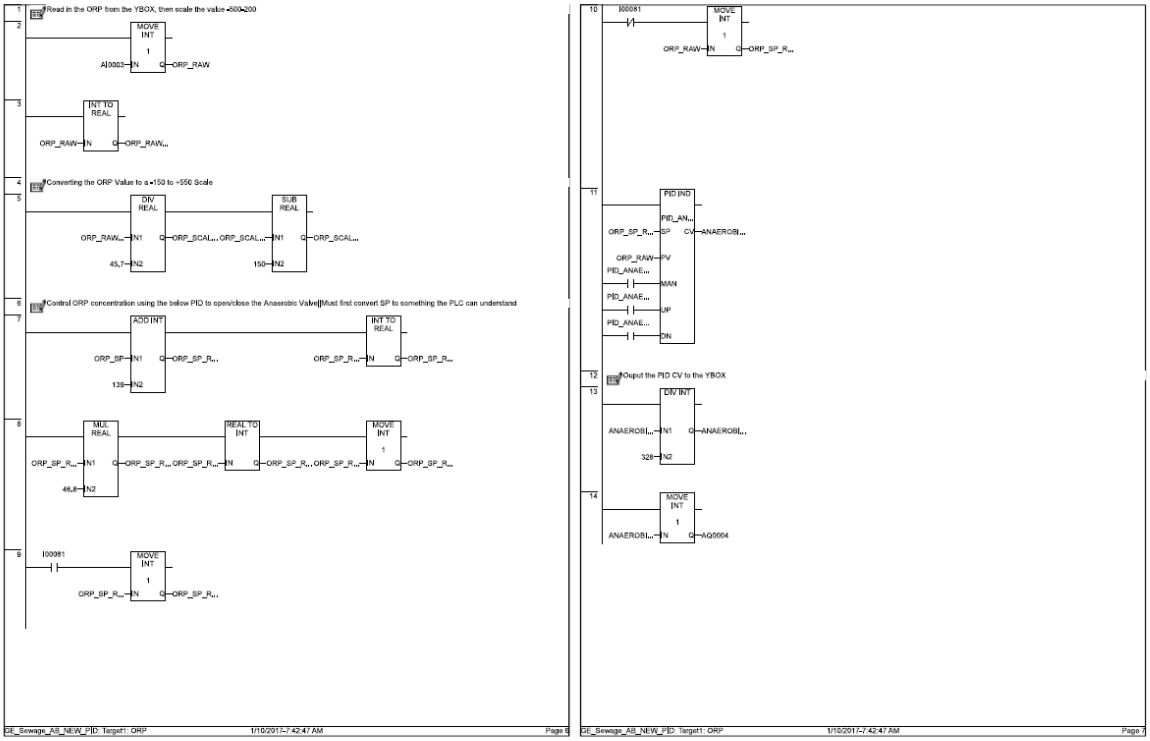


Figure 26. DO control-opens aerobic valve to increase or decrease DO and maintain set point.



Read in the ORP from the YBOX, then scale the value -500-200

Converting the ORP Value to a -150 to +550 Scale

Control ORP concentration using the below PID to open/close the Anaerobic Valve||Must first convert SP to something the PLC can understand

Ouput the PID CV to the YBOX

GE_Sewage_AB_NEW_PID: Target1: ORP 1/10/2017-7:42:47 AM Page 6

GE_Sewage_AB_NEW_PID: Target1: ORP 1/10/2017-7:42:47 AM Page 7

Figure 27. ORP control-opens anaerobic valve to increase or decrease ORP and maintain set point.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704–0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD–MM–YYYY)*
23–03–2017

2. REPORT TYPE
Master's Thesis

3. DATES COVERED *(From — To)*
June 2015 — Mar 2017

4. TITLE AND SUBTITLE
Increasing Cyber Resiliency of Industrial Control Systems

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

5d. PROJECT NUMBER
17G310

5e. TASK NUMBER

5f. WORK UNIT NUMBER

6. AUTHOR(S)
Chaves, Andrew J., 2nd LT, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT-ENG-MS-17-M-013

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of Homeland Security ICS-CERT
POC: Neil Hershfield, DHS ICS-CERT Technical Lead
ATTN: NPPD/CSC/NCSD/US-CERT Mailstop: 0635
245 Murray Lane, SW, Bldg 420, Washington, DC 20528
Email: ics-cert@dhs.gov phone: 1-877-776-7585

10. SPONSOR/MONITOR'S ACRONYM(S)
DHS ICS-CERT

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Industrial control systems (ICS) are designed to be resilient, capable of recovering from process faults and failures with limited impact to operations. Current ICS resiliency strategies use redundant PLCs. However, these redundant PLCs, being of similar make and model, can be exploited by the same cyber attack, defeating the ICS's resiliency strategy. This research proposes a resiliency strategy for ICS that employs an active defense technique to remove the cyber common cause failure. The resiliency of the active defense strategy is compared to traditional ICS resiliency by implementing both strategies in a semi-simulated wastewater treatment plant aeration basin that experiences a cyber attack. The active defense technique was shown to maintain effective treatment of the wastewater through the cyber attack where the traditional implementation allowed a process disruption that prevented the effective treatment of the wastewater.

15. SUBJECT TERMS
Industrial Control Systems, Resiliency, Active Defense

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Barry E. Mullins, AFIT/ENG
					19b. TELEPHONE NUMBER *(include area code)*
U	U	U	UU	132	(937) 255-3636 x7979; barry.mullins@afit.edu

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18