BUREAU OF RECLAMATION

Development and Refinement of Rotor Turning Device for Safer and More Efficient Maintenance and Diagnostic Tasks

Science and Technology Program
Research and Development Office
Final Report No. ST-2023-21006-01

U.S. Department of the Interior

January 2023

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-31-2024	Research	December, 2021 – January, 2024

4. TITLE AND SUBTITLE
Development and Refinement of Rotor Turning Device for Safer and More Efficient Maintenance and Diagnostic Tasks

5a. CONTRACT NUMBER
RR4888FARD2101601/F406A

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
1541 (S&T)

6. AUTHOR(S)
Jacob Lapenna, Electrical Engineer

5d. PROJECT NUMBER
Final Report ST-2023-21006-01

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Hydropower Diagnostics and SCADA Group
Technical Service Center
Bureau of Reclamation
U.S. Department of the Interior
Denver Federal Center
PO Box 25007, Denver, CO 80225-0007

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Science and Technology Program
Research and Development Office
Bureau of Reclamation
U.S. Department of the Interior
Denver Federal Center
PO Box 25007, Denver, CO 80225-0007

10. SPONSOR/MONITOR'S ACRONYM(S)
Reclamation

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Final Report ST-2023-21006-01

12. DISTRIBUTION/AVAILABILITY STATEMENT
Final Report may be downloaded from https://www.usbr.gov/research/projects/index.html

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A modular system, known as the Rotor Turning Test Suite (RTTS) was designed, developed, and tested under this research. This system allows for turning the rotors of rotating machines within Reclamation's fleet in a safe, precise, and efficient manner. The RTTS also allows for future expansion by incorporating slow-roll diagnostic procedures into the system software for automated and efficient diagnostic testing, tracking, and trending. Full automation of the turning process is also possible in the future. This report summarizes the basics of this system, and discusses future work to complete and expand the RTTS.

15. SUBJECT TERMS
rotor turning, rotating machine, rotating machine diagnostics, automation, cyber-physical system, system design

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	THIS PAGE			Jacob Lapenna
U	U	U			19b. TELEPHONE NUMBER *(Include area code)* (303) 445-2829

Mission Statements

The Department of the Interior (DOI) conserves and manages the Nation's natural resources and cultural heritage for the benefit and enjoyment of the American people, provides scientific and other information about natural resources and natural hazards to address societal challenges and create opportunities for the American people, and honors the Nation's trust responsibilities or special commitments to American Indians, Alaska Natives, and affiliated island communities to help them prosper.

The mission of the Bureau of Reclamation is to manage, develop, and protect water and related resources in an environmentally and economically sound manner in the interest of the American public.

Disclaimer

Information in this report may not be used for advertising or promotional purposes. The data and findings should not be construed as an endorsement of any product or firm by the Bureau of Reclamation, Department of Interior, or Federal Government. The products evaluated in the report were evaluated for purposes specific to the Bureau of Reclamation mission. Reclamation gives no warranties or guarantees, expressed or implied, for the products evaluated in this report, including merchantability or fitness for a particular purpose.

Acknowledgements

The Science and Technology Program, Bureau of Reclamation, sponsored this research.

Development and Refinement of Rotor Turning Device for Safer and More Efficient Maintenance and Diagnostic Tasks

Final Report No. ST-2023-21006-01

prepared by

Technical Service Center
Jacob Lapenna, Electrical Engineer

Contents

Page

Mission Statements ..... **iii**
Disclaimer ..... **iii**
Acknowledgements ..... **iii**
Peer Review ..... **v**
Executive Summary ..... **vii**
1. System Design **1**
1.1 Background 1
1.2 System Requirements 2
1.3 Systems Analysis 3
1.3.1 User Research 3
1.3.2 Turning Resolution 4
1.3.2.1 Absolute vs Relative Encoder 5
1.3.2.2 Absolute Encoder Accuracy 6
1.3.3 Modular Design 8
2. Rotor Turning Test Suite (RTTS) **9**
2.1 Hardware 9
2.2 Software 14
2.2.1 Distributed Software Architecture 14
2.2.2 User Interface 1
2.2.3 Mapping and Automated Turning 3
2.2.4 RTTS Application Deployment 5
3. Future Advancement **6**
3.1 Automation 6
3.1 Air Gap Measurement 7
3.2 Partial Discharge Mapping 7
3.2 Circumference Tool 7
4. Conclusions **10**
References **12**
Appendix A – Distribution Box **13**
Appendix B – DC Supply Box **15**
Appendix C – Rotor Control Box **17**
Appendix D – Stator Control Box **18**

Peer Review

Bureau of Reclamation
Research and Development Office
Science and Technology Program

Final Report ST-2023-21006-01

Development and Refinement of Rotor Turning Device for Safer and More Efficient Maintenance and Diagnostic Tasks

JACOB LAPENNA

Digitally signed by JACOB LAPENNA
Date: 2024.01.29 11:38:21 -07'00'

Prepared by: Jacob Lapenna
Electrical Engineer, Bureau of Reclamation, Technical Service Center

LYLE BROUWER

Digitally signed by LYLE BROUWER
Date: 2024.01.29 12:35:03 -07'00'

Peer Review by: Lyle Brouwer
Electrical Engineer, Bureau of Reclamation, Technical Service Center

Executive Summary

The Bureau of Reclamation (Reclamation) has many large rotating machines within its hydropower fleet. From maintenance tasks like machine alignment, to diagnostic tasks like air gap measurement and rotor mounted partial discharge mapping, the need to turn these machines in a slow, controlled, and precise manner is ubiquitous across these assets.

In practice, this turning has always been performed by personnel getting inside the machine and pushing on the rotor. In cases where this is not possible, rotation is often facilitated by complex crane maneuvers. In all cases, however, manually turning a machine's rotor is dangerous, inefficient, and imprecise. This research has developed a system to perform this turning with the click of a button, without personnel inside the machine, while monitoring the rotor's rotational position with very high precision and accuracy. Called the Rotor Turning Test Suite (RTTS), it is a cyber-physical system that was designed to control rotor turning via a local, real-time browser application. The browser application is local in that there is no internet connection, for security reasons, and the application is served over the local area network. The application is real-time in that it controls the rotor and displays data to the user as it happens.

The RTTS is modular in both hardware and software. The hardware is modular to be field deployable and to scale to the user's needs across Reclamation's fleet and various machine turning tasks. The software is modular to allow for future advancements in a controlled and efficient manner. Intended future advancements include incorporating air gap measurement and rotor mounted partial discharge mapping into the real time browser application, as well as full automation of the turning process.

The overall system architecture was designed to minimize user burden while increasing capability. Specifically, the networked architecture allows for the modular nature of the system. With control via a browser application, the user does not need to install any software on their device. The user can simply connect to the RTTS local network, navigate a browser to the application, and start interacting with the controller. Software updates need only be applied to the RTTS itself for the user to gain access to new features and advancements, and the user does not need to concern themselves with information technology policy compliance, as no software is installed on their device.

This research determined the RTTS system architecture through systems analysis and user research. User research was carried out via surveys. Systems analysis determined the need for the RTTS to utilize a rotary encoder to measure rotor position during turning, and a once-per-revolution (OPR) indicator to track precise rotation count. These devices are needed to meet the precision and accuracy in position monitoring that users require. While many respondents of the user surveys indicated a technological squeamishness and a desire for physical controls (analog knobs and switches) instead of computer controls, systems analysis determined that the needs of the users could not be fully met without computer controls. In the face of the conflicting requirements, compromises were made toward computer controls in favor of increased functionality and future enhancements.

The resultant system is the first iteration of the RTTS and represents years of system engineering. This version of RTTS can be immediately deployed for basic and efficient rotor turning and is poised to be the test suite of the future for all maintenance and diagnostics tasks requiring controlled, precise turning of a rotor. While the immediate return on investment (ROI) is clear in increased safety and efficiency, this ROI will only increase as future funding creates more enhancements to the RTTS. These future enhancements will help asset managers collect, maintain, and interpret the needed diagnostic data to keep their fleets running smoothly and make high-dollar decisions quickly.

1. System Design

1.1 Background

Controlled, precise rotation of a generator or pump is a constant need during routine maintenance and diagnostic procedures within Reclamation's hydropower fleet. The problem is that these rotors (and associated shafts and turbines) are very large and very heavy. Previously, the only way to turn these rotors was by hand, with personnel getting inside the machine and pushing. This is very dangerous, as the mass makes it difficult and straining to get the rotor to turn in the first place, and, once it is turning, it takes a lot of effort to stop it. Even when turning very slowly, the mass could easily crush a person inside. When a rotor is too large, or turning friction is too high, personnel cannot turn the rotor manually by pushing, and they often resort to using cranes or hoisting equipment to pull on parts of the rotor with chains and rigging. All these methods of turning are dangerous to personnel and take significant effort and potentially days to coordinate and carry out. In addition, turning forces can be applied nonuniformly to the rotor, which can push the turning shaft against one of its bearings. This off-center forcing can affect sensitive measurements during maintenance and diagnostic tasks. For these reasons, previous work developed a prototype for turning rotors via relatively low power electromagnetic forcing.

This prototype magnetizes the rotor poles by passing a direct current through the field winding. A direct current is then also applied to one phase of the stator winding, magnetizing it. Like a stepper motor, magnetic forcing between the rotor and stator fields applies a torque to the rotor shaft in the direction of field alignment (opposite magnetic poles attract, like poles repel). If the magnetic fields create sufficient torque, the rotor will turn. By switching the current in the stator, torque on the rotor can be controlled. Torque is minimized when the magnetic fields align and is maximized when the fields are fully unaligned. Field alignment can occur in two ways: a stable alignment occurs when opposite magnetic poles are together, and an unstable alignment occurs when like poles are together. For small movement away from these alignment positions, with stator current applied, torque is toward the stable position and away from the unstable one.

The prototype requires a knowledgeable operator to stand above the turning rotor and time the stator switching with the magnetic alignment (and varying torque) of the fields to maintain rotation. The stator current is switched off just before the rotor reaches stable alignment with the stator's magnetic field. The momentum of the turning rotor allows it to turn past this alignment point, and current in the stator can be reversed and switched back on just after the rotor passes this alignment point. The reversed current changes the stable alignment to an unstable one, creating a torque to continue rotation away from this position. The rotor will continue rotation until it approaches a stable alignment for the reversed current direction. This process is repeated to continue rotation as desired.

This prototype works well, though requires extensive knowledge (or practical experience) of the varying magnetic forces and torque described above to reliably control rotor turning. In addition, even when reliable turning could be mastered with the prototype, precision was lacking, and, for example, precisely turning the rotor in 90-degree increments is still difficult. Because precise turning

tasks need to be performed by a diverse range of personnel across crafts and disciplines, it was clear that a more advanced turning system needed to be developed. Such a turning system needed to track and report rotor position to a fraction of a degree. In addition, automation is desirable, so that any operator can simply make the necessary electrical connections to the rotor and stator windings, click a button, and the rotor would automatically rotate to a predetermined position and stop. Such a system upgrade would require the design of a cyber-physical system, with sensors and actuators to interact with physical processes, and software to monitor these sensors, make decisions, and control these actuators. The design of this new cyber-physical system was the work of this research.

1.2 System Requirements

The following system requirements were determined to be minimally necessary for a fully capable rotor turning system.

1. Maximum voltage peak protection on all asset connected circuits.
2. Protection for switching inductive loads on all asset connected circuits.
3. Switching of stator current to control rotation.
4. Switching of stator current to either of two connected phases of a three-phase machine.
    a. If one phase is aligned (zero torque), the other phase will not be, and torque can still be imparted on the rotor without physically changing circuit connections.
    b. High-voltage testing can be carried out on the third phase not connected to the stator switching device.
5. All source circuits will have Underwriters Laboratory (UL) listed protection products in place for safety and equipment protection.
6. Other than source circuit voltage, all system voltages should be less than 60 V for human safety.
7. Relative rotational position (i.e., rotor position relative to its starting position) has sufficient resolution, accuracy, and precision to determine rotor position within one slot width (i.e., the arc length of a single stator winding slot).
8. Relative rotational position can be displayed to the operator in real time.
9. The system is physically robust for shipping to various field sites.
10. The entire system can safely and reliably fit on a single shipping pallet.
11. The system can easily incorporate future automation of typical turning tasks such as air gap measurement and partial discharge mapping.
12. The system can store historical data (such as any needed machine parameters or diagnostic test results) for later recall and use.
13. The system has a means of access control.
14. The system should be safe for both personnel and the assets it is connected to.

It is important to note that the above requirements say nothing about how the system will achieve this functionality but rather only what the system needs to be capable of doing. The actual way in which these requirements are accomplished is the realm of system engineering and systems analysis, which is discussed below.

1.3 Systems Analysis

1.3.1 User Research

The first task of this project was an attempt to gain insight into what a potential user of this system wants. This was done by sending a survey of questions to prospective users throughout Reclamation. This user research helped shape the system requirements above, most notably the requirement of position resolution. This position resolution was determined by noting user needs during the tasks they were most likely to carry out when performing rotor turning. In addition to system requirements, some insight into the desired user interface was gained from these surveys as well.

For the survey questions related to the user interface, it was clear that a technological squeamishness was present. Comments from many respondents indicated a lot of this squeamishness resulted from a desire to reduce user burden in maintaining related software on their own laptops and devices. In other words, all the software should be fully contained and maintained within the system devices. However, part of this technicalogical squeamisheness also arose from a fundamental desire to have only mechincal and analog components and controls, and no software at all. However, the system requirements cannot be met reliably without some level of software control, and so this desire for a purely physical analog system cannot be entertained. It was therefore determined that a browser-based application was the most effective way to achieve the overall system requirements while maintaining fidelity to commonly accepted system design principles and facilitating user needs and desires. The main reasons for this determination are as follows:

1. Software is required. Any purely physical user interface would simply be interacting with this software in the background and would represent an added layer of complexity and abstraction. The user interface itself should skip such added complexity and be software based itself.
2. Almost everyone in modern society has used a browser and browser application at some point, which makes the user interface more intuitive and lowers training time and burden.
3. To access a browser application, the user only needs to have a device with network connectivity and a browser. Most government furnished laptops and iPhones have this capability built in. The user does not need to install any specialized software onto these devices or do anything outside the typical device maintenance as prescribed by their information technology department.
4. Connection to the system is through a web browser. Mainstream browsers are developed and distributed by large corporations with extensive budgets and represent some of the most used pieces of software in the world. This means browsers are already highly optimized for speed and reliability and are extensively tested to minimize bugs and increase security.
5. Browser applications support development via modern programming languages like Python, Hypertext Markup Language (HTML), Cascading Stylesheets (CSS), and JavaScript. Like the browser, such languages are also extensively optimized and tested. These languages also support software schema that can follow commonly adopted system design principles like separation of concerns, encapsulation, abstraction, scalability, performance, and fault tolerance.
6. Programming languages like Python already have many free and open-source libraries, packages, and application programming interfaces (API) that can help achieve system

requirements in a more efficient way. For example, Python has libraries and packages for multiprocessing, in-memory data caching, event handling, and distributed computing. There are also many data acquisition systems readily available with published Python APIs for interfacing with them.

7. HTML and CSS were designed to create incredibly rich visual content and JavaScript was designed to make that content dynamic and interactive. These languages are purpose built for delivering visual content in a highly efficient and abstracted way.

In short, browser applications have been used for decades to deliver rich, dynamic, and interactive content and data to humanity at an unparalleled scale with unparalleled resources devoted to optimizing and testing these technologies. Such technologies represent the best possible framework to develop a dynamic and interactive control interface which can present complex data to the user in an intuitive and visual way. For all these reasons and more, a browser application software architecture was chosen for the control interface. The chosen tech stack was Python, HTML, CSS, and JavaScript, with more specifics of the software architecture discussed in greater detail later.

1.3.2 Turning Resolution

Without any external sensors, the only way to determine rotor position is by correlation to current and voltage signals on the control circuits connected to the rotor and stator windings. There are theoretically several ways of doing this. One way would be to correlate rotor position with impedance, as the current circuit's impedance will change as the rotor comes in and out of alignment with the stator phase under control. Another way would be to correlate rotor position with the voltage induced on the stator phase terminals. Specifically, as the magnetized rotor turns, the change in flux within the stator winding will induce a sinewave voltage on the stator terminals, and this sinewave's oscillation corresponds to the rotor coming in and out of alignment with the stator winding phases. However, in both cases, the best possible position resolution would be approximately the width of a rotor pole face. In addition, these methods would require extensive signal conditioning and real time signal analysis to carry out. A far simpler and more reliable method would be to adopt an external sensor that monitors rotor position during turning.

Rotary encoders are typically used for just this purpose throughout many industrial automation applications. An encoder, fitted with a wheel of precise circumference, could be pressed against the rotor shaft or slip ring, causing the encoder to turn in proportion to rotor rotation. The encoder (and once-per-revolution (OPR) sensor) setup developed in this research project is shown in Figure *1*. This encoder position can be known to a high level of accuracy and precision. The problem then becomes mapping encoder position to rotor position. This is a matter of knowing the drive ratio, or the circumference of the drive surface (e.g., the slip ring in Figure *1*) divided by the circumference of the encoder wheel being driven (i.e., the black wheel on top of the encoder in Figure *1*).

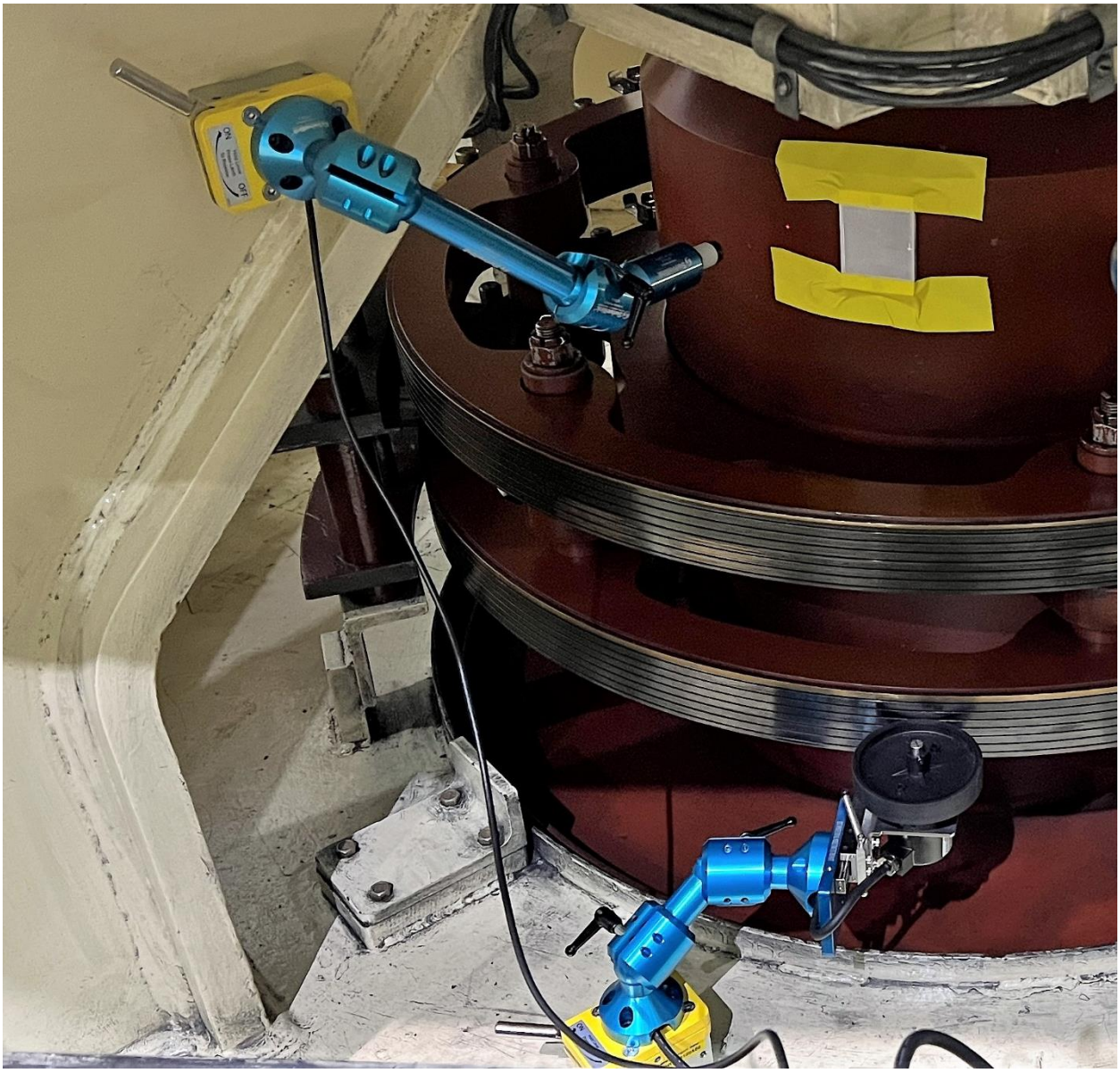


Figure 1 Photo of an encoder being driven by a rotor slip ring (bottom) and a once per revolution (OPR) sensor mounted to detect full rotation from optically sensing the retroreflector (top).

1.3.2.1 Absolute vs Relative Encoder

There are two main types of encoders, absolute and relative. Relative encoders provide a pulse signal output, and encoder position is determined by counting the pulses encountered as the encoder turns. An absolute encoder outputs a binary value for a discrete position. For example, an 8-bit absolute encoder would output an 8-bit binary value, with 2^8=256 possible unique position values in one revolution of the encoder.

The relative encoder suffers from present position only being known if the previous position is known. In other words, if the software or device monitoring the encoder loses count of the pulses (e.g., due to signal loss or a processing interruption), encoder position (and therefore rotor position)

becomes inaccurate, with increased inaccuracy for longer interruptions (or faster turning speeds with the same interruption duration).

In contrast to relative encoders, an absolute encoder's signal can be lost or not monitored for a short period of time, and, when the signal is regained, exact encoder position can still be known by simply reading the binary value output when the signal or processing capability is regained. This makes the absolute encoder a much more fault tolerant method of determining rotor position and was the method of choice for the system developed here.

For large drive ratios, an encoder turns several times for each rotor revolution. This means that even an absolute encoder's total revolutions must still be counted, however, in practice, this is much less process intensive than the counting of a relative encoder pulse. Specifically, signal loss and processing interruptions usually do not last as long as a full encoder revolution for a large range of drive ratios and common turning speeds. In addition, fault checking can be built into the software for this case by referencing rotational speed, rotational direction, and comparing the last absolute encoder value with the presently measured one.

1.3.2.2 Absolute Encoder Accuracy

One drawback of using an encoder for monitoring rotor position is that the drive ratio must be known to very high accuracy and precision. In addition, this method needs to work on all rotating machines within Reclamation's fleet. Each facility, and often each machine within a facility, is unique. This means that, for a constant encoder wheel circumference, the drive ratio is likely different on every machine within Reclamation's fleet. Moreover, any error in measuring the drive ratio (i.e., error in the measured drive surface's circumference) propagates to the error of the calculated rotor position.

This relative rotor position, in degrees from the initial starting position, as a function of encoder position, is calculated using the following equation:

$$\theta_R(n) = 360\frac{C_e}{C_d}\left(k + \frac{n}{N}\right)$$

Where C_e is the circumference of the encoder wheel, C_d is the circumference of the drive surface (C_e/C_d is the inverse of the drive ratio), N is the encoder's number of points per revolution, n is the present point output by the encoder (encoder position), k is the count of full encoder revolutions from its starting position, and θ_R is the angular position of the rotor (in degrees) relative to its starting position. k and n can be given a sign convention for rotational direction as needed (e.g., starting at zero, counting down for counterclockwise rotation and up for clockwise rotation).

Note that in the above equation for rotor position, we must measure the drive surface and encoder wheel circumferences C_d and C_e. In practice, this is a difficult task to carry out accurately and precisely, and any error in these circumference measurements would propagate as increased uncertainty in rotor position. For this reason, it is advisable to use another sensor to generate a once-per-revolution (OPR) signal. The OPR signal is generated by a stationary laser that outputs a pulse when it reflects from a retroreflector device. If this retroreflector is attached to the rotating shaft of the rotor, exactly one revolution of the rotor can be determined (i.e., the first pulse marks the start of the revolution and the second pulse marks one revolution). We can use this OPR signal

to determine one revolution of the rotor while monitoring the encoder to precisely determine the drive ratio. In this way, the rotor turning system operator would not have to measure any circumferences and could simply use the system itself to calculate the drive ratio. The system would then have everything it needs to determine rotor position as accurately and precisely as feasibly possible. The equation for rotor position in terms of the drive ratio R_d is:

$$360\frac{1}{R_d}\left(k + \frac{n}{N}\right) = \theta_R$$

The drive ratio, in terms of reliably measurable quantities, after exactly one revolution (360 degrees) of the rotor is:

$$k + \frac{n}{N} = R_d$$

The uncertainty in the rotor position θ_R depends on the uncertainty in the values used to calculate it. N is a constant, and k is countable, therefore there is no uncertainty associated with these values. If we assume the uncertainty in the OPR signal (e.g., time delay in signal output, measurement, and processing) is very small for reasonable rotor turning speeds, we only need to consider the error in the encoder position. Typical uncertainty for an absolute encoder's position is half of the encoder's smallest increment. For an 8-bit encoder, the smallest increment would be $360/2^8$ or 1.4 degrees and therefore the uncertainty in encoder position δ_n would be 0.7 encoder degrees. Moreover, because encoder uncertainty is constant, it will have a very small contribution to rotor position for large drive ratios. In practice, typical drive ratios are around 10 for an encoder wheel that is one foot in circumference (the encoder turns 10 full revolutions for each one rotor revolution), and encoder uncertainty would be 0.03% of the drive ratio in this case. For this reason, we neglect the uncertainty in drive ratio when calculated from the OPR and encoder readings. This means all values except for n in the equation for relative rotor position can be treated as constants (i.e., no uncertainty associated with these values).

Using the propagation of uncertainty (Taylor, 1997) in a single variable function we can approximate the uncertainty in the calculated drive ratio δ_R as:

$$\delta_R \leq \frac{360}{R_d N}\delta_n$$

This uncertainty is important to characterize, as system requirements specifically require knowing the rotor position to within one stator slot width. Regardless of machine size, stator core tooth width (and therefore slot width) is optimized for magnetic flux. This size is roughly similar across machines and is an optimization between enough iron (core teeth) to prevent magnetic saturation, and enough copper (stator slot) to efficiently carry operational circuit current through the stator winding. This results in a slot size that is on the order of one inch in arc length. However, this also means that for larger diameter machines, slot count goes up, and system resolution must accommodate these larger machines. This means that for Reclamation's highest slot count machines, rotor position needs to be known to withing 0.48 degrees. Therefore, the required resolution in rotor position needs to be 0.48±0.24 degrees or better. With this resolution and accuracy, the system could place an arbitrary point of the rotor in front of an arbitrary slot on the stator to within half of

a slot width. For machines with fewer slots, this resolution would only increase precision of locating a point on the rotor to within a smaller fraction of a slot width.

For an 8-bit encoder, any drive ratio above 2.93 would allow for an encoder resolution of 0.48 degrees or better. In other words, a drive ratio of 2.93 or higher would mean 2.93 encoder revolutions for each rotor revolution, or 750 encoder points per rotor revolution. This is 0.48 rotor degrees per encoder point. However, if we solve for δ_R in the inequality for rotor position uncertainty, we note that the drive ratio must be at least 4.12 to achieve an uncertainty of 0.24 degrees or less. As the drive ratio increases, the uncertainty decreases. This means the drive surface diameter (e.g., the slip ring or rotor shaft) must be 1.31 feet or greater for an encoder wheel circumference of 1 foot. In practice, this is easily achieved within Reclamation's fleet. This analysis also suggests that an 8-bit encoder is sufficient.

1.3.3 Modular Design

Separation of concerns is a fundamental design principle of system engineering. This principle focuses on modular design where each module represents a specific functionality within the overall system. Reliance between modules is minimized, and cohesion within a module is maximized. In the case of the rotor turning system, modularity in the physical sense requires self-contained devices that perform specific functions and can be incorporated into the overall system as needed. The overall system does not need each module to function (low reliance), and each module contains the minimum required hardware and software to perform its specific task (high cohesion). This modularity is also incorporated into the control software as well. This design philosophy facilitates efficient system maintenance, repair, scalability, upgrades, and enhancements.

For the rotor turning system, the most basic functionality (modularity) is as follows, and each entry in the list below represents a basic device (module) within the rotor turning system:

- Power distribution: getting power from the wall plug to each device within the system.
- Rotor controller: control of the rotor field current and rotor position measurement.
- Stator controller: control of the stator field current.
- Current supply: the current supplies that the rotor and stator controllers interact with.
- Future enhancements (not implemented within this research):
    - Air gap measurement: measure and record the distance between the rotor and stator with sensors mounted on the rotor and stator. Close communication with the rotor and stator controller is required to obtain this data over the entire circumference of the machine as the rotor turns.
    - Partial discharge mapping: remotely sense partial discharge in each stator slot from sensors attached to the rotor. Close communication with the rotor and stator controller is required to obtain this data and map it to each stator slot over the entire circumference of the machine as the rotor turns (Lapenna, 2021).

These modules/devices form the foundation of the Rotor Turning Test Suite (RTTS). The RTTS not only meets the initial basic requirement of precise control over rotor turning, but it also opens the door to a whole host of capabilities in slow-roll diagnostic testing and maintenance tasks.

2. Rotor Turning Test Suite (RTTS)

The system designed and constructed under this research has been designated the Rotor Turning Test Suite (RTTS), which is also synonymous with the Rotor Turning Test Set. It is intended to be easily adaptable and expandable with future capability to automate all diagnostic testing or tasks that require rotor turning. Hence the use of "suite" within the RTTS name indicating a group of related components intended to be used together. The fundamental design of the RTTS is described here.

2.1 Hardware

To achieve a physical design that is modular and easily deployed, the entire system was engineered to fit inside multiple Pelican iM2620 Storm cases. These cases are durable, easily acquired, and have an exterior footprint (when laid on their backs) of 21.20x16.00 inches. This makes them easily stackable, with four cases to a stack layer, on a 40x48 inch shipping pallet, with each stack layer being 10.60 inches. The entire system can be contained in no more than sixteen of these cases, for a total stacking height of 42.20 inches on the pallet. With no case weighing more than 40 pounds, the entire system is easily deployed by a single individual from the pallet stack, and the stacked pallet can be shipped to any Reclamation facility as needed. The principle of separation of concerns was followed in the design of each case and its contents (referred to as a box) with each case containing only the minimum required hardware to carry out its required functionality. When deployed, the boxes are connected as depicted in Figure *2*, where green is alternating current (AC) power, blue is the inter-integrated circuit (I2C) control network, orange is the local area network, purple is sensor connections, and red and black is the high-current DC control signal. Each box within Figure *2* is discussed in greater detail below.

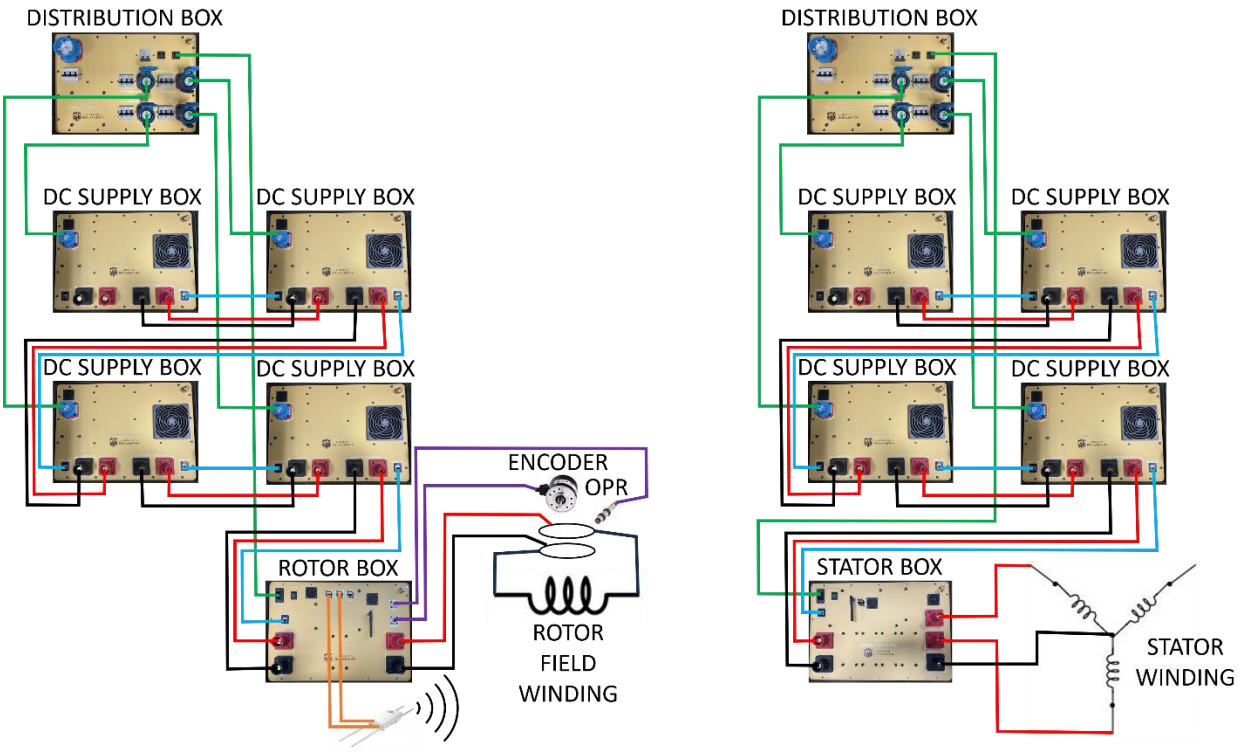


Figure 2 RTTS connection diagram

The system is powered via distribution boxes that distribute three-phase 480 V, 60 A AC power from a wall receptacle to the needed location within the system. The system is broken into two main categories: rotor functionality and stator functionality. Because the rotor field connection can be physically far away from the stator field connection, these functionalities need to be physically independent. For this reason, there are two distribution boxes, one for the stator functionality and one for the rotor functionality. Each distribution box contains four 480 V, 20 A circuit branches as well as a step-down transformer that feeds two 120 V, 3 A circuit branches. In addition, all circuit branches and the main input are protected via circuit breakers. All of this is mounted on a custom faceplate bolted to the iM2620 case for a fully enclosed box as shown in Appendix A – Distribution Box. The two distribution boxes are identical and therefore interchangeable.

Powered by the distribution boxes, there are a total of eight direct current (DC) power supply boxes. Each of these boxes contains a 480 V, 20 A input receptacle and an XP Power HPT5K0TS048 power supply rated at a nominal 5 kW output. There is also a fan to help cool the XP power supply, a smaller 24 V power supply to run the fan (and power any future functionality within this box). In addition, there are ethernet connectors for parallel control communication, and 400 A rated cam-lock connectors for parallel connection to other DC power supply boxes. The XP power supply within each DC power supply box is capable of a maximum output of 113 A at 48 V. The distribution box allows up to four DC power supply boxes to be connected in parallel to supply 452 A at 48 V. With eight DC power supply boxes, the RTTS has the capability to apply 452 A to the stator winding and 452 A to the rotor field winding simultaneously.

Each XP power supply within each DC power supply box is controlled via the I2C protocol. This means each XP supply needs a unique I2C address. This address is determined by selectively

grounding three pins on the back of the supply to control three bits within the supply's eight-bit address. Because three bits are being selectively controlled, eight unique addresses are possible, and this is why the RTTS system has been designed with eight total DC power supply boxes. According to the datasheet, a total of five XP power supplies can be connected in parallel, and we could have designed the RTTS to have a total of ten DC power supply boxes (five for the rotor and five for the stator). However, limiting the RTTS to eight DC power supply boxes makes these boxes within the RTTS interchangeable without the operator needing to worry about I2C address conflicts. If ten boxes were used, then there would be two repeated I2C addresses, and the RTTS operator would have to take care not to connect conflicting addresses on the same parallel circuit. By using eight, these are interchangeable throughout the RTTS system, and the operator does not need to worry about what addresses are connected on which parallel circuit (rotor or stator). These DC power supply boxes are directly controlled via the rotor and stator control boxes. The schematics and custom faceplate for the DC power supply boxes can be seen in Appendix B – DC Supply Box. The faceplates have been designed to accommodate the XP Power HPT5K0TS048-L, which is the same power supply but in a different physical form factor than the HPT5K0TS048. If the HPT5K0TS048 has limited stock, repair of existing systems or new systems can utilize the HPT5K0TS048-L instead.

The rotor control box controls rotor functionality and hosts the communication network hardware, orchestrates all tasks throughout the RTTS system, serves the user interface and handles user requests, and directly controls the current to the rotor field winding. The rotor control box is considered the keystone of the RTTS system. This was a design choice and was made to minimize latency in the communication of rotor position to the RTTS operator. Specifically, because the encoder and OPR signals are most reliably taken from the rotor slip ring, which is physically close to the rotor current supply input, it made sense for the rotor control box to monitor these signals. Because the rotor control box monitored these signals, it also made sense for it to host and serve the user interface, as this would eliminate the need to transfer raw sensor data over the network, which reduces overall latency in position monitoring and reduces network load. Similarly, because almost all other automation logic follows from rotor position (i.e., stator current switching, air gap measurements, and partial discharge must all be mapped to rotor position), it made sense to have the rotor control box be the main controller of the RTTS system, orchestrating the software tasks of any other control box in the system via distributed computing. Finally, because the rotor control box is the main controller, it made sense to also be the main network node and house the network router and be the domain name server (DNS) of the RTTS system.

The rotor control box achieves its functionality mainly via custom software running on a Raspberry Pi CM4. This CM4 is Wi-Fi capable and has 8 GB of memory. The operating system and software run from a 128 GB microSD (SD) card, which allows for easy field repair with an identically flashed SD card should the original become corrupted or fail. The CM4 is mounted to a Raspberry Pi CM4 IO Board. The IO Board facilitates connection of two Measurement Computing MCC 152 digital data acquisition (DAQ) hardware attached on top (HAT) devices, or DAQ HAT for short. One DAQ HAT to monitor the encoder, and another to monitor the OPR signal. There is another HAT to facilitate connection to the CM4's various general-purpose input/output (GPIO) pins via screw terminal blocks. This is all powered from a 24 V, 10 A DC power supply, which is in turn powered from the box's 120 V input. This 120 V input is intended to be plugged into the 120 V, 3 A output of the power distribution box discussed earlier.

Memory corruption within the CM4's operating system could result if the main power was interrupted while writing to memory. Therefore, an uninterruptable power supply (UPS) is needed.

However, most UPS devices utilize internal batteries to maintain power when input power is interrupted. Such a device would require maintenance to change these batteries and is not ideal. To get around the need for UPS batteries, the final HAT on the IO Board is a Juice4Halt J4H-HV-TRM, which utilizes large capacitors to power the CM4 for approximately 60 more seconds in the event of power interruption. In addition to maintaining power, the UPS HAT communicates with the CM4 to run a shutdown script when it detects an interruption to the input power. With this UPS, the operator can simply unplug or switch off the rotor control box and trigger a clean shutdown of the CM4.

The IO Board is mounted on a sheet of 1/8 inch thick ultra-high molecular weight (UHMW) polyethylene plastic which is mounted to an aluminum angle bracket that mounts to the box's custom faceplate. This custom UHMW plastic mounting bracket provides a flexible mounting surface to reduce shock to the electronic components during shipping. The power converter for the Wi-Fi router is mounted on the back of this flexible sheet. The power converter takes 120 V input and outputs power over ethernet (PoE) to the Wi-Fi router. This PoE connection node can also take an ethernet input from an external modem for internet connectivity as needed for software development, installation, and updates.

The router is an Advantech EKI-6333AC-2G-A router, which can function as a bridge, client, or access point as desired. If needed, more of these routers can be deployed as bridge repeaters to increase network reach through a facility. There are three ethernet through-ports on the custom faceplate to facilitate this network. The first port delivers PoE power to the router from the power converter in the box, through the faceplate, to the router mounted on the inside of the case lid. The second port connects the router through the faceplate to the CM4's network port and is designated the local area network (LAN). The third port can take in an ethernet internet connection from a modem through the faceplate to the power converter, which can then connect the RTTS network to the internet via the PoE connection. This port is designated the wide area network (WAN). In practice, the WAN is only used during software dependency installation and updating from cloud sources as needed. During operation, the WAN connection is not typically used. There is also an ethernet port to output I2C control communication to the DC power supply boxes.

In addition to the above components, the rotor control box also takes in the paralleled output from the DC power supply boxes, runs it across a freewheel diode, and outputs this current to the rotor field. The freewheel diode circulates current and prevents large voltage spikes when switching an inductive load, as most rotor field windings have high inductance. Appendix C – Rotor Control Box shows the rotor control box design schematics.

The stator control box is like the rotor control box in that it houses a similar CM4 setup. This CM4 is also equipped with a capacitor-based UPS, though it does not utilize any DAQ HATs, as it does not monitor any sensors. This CM4 uses a Wi-Fi antenna to connect to the RTTS local area network and listens for instructions over the network from the rotor control box. These instructions determine switching and direction of the current through one of two phases in the stator winding.

Per system requirement 4, the RTTS must be able to control current in one of two connected stator phases of a three-phase machine. This is because aligned magnetic fields produce zero torque, and rotation from a stationary aligned position is not possible. When one phase is aligned with the rotor, the other two phases are not. The system can simply route current through one of the unaligned phases to produce torque and start rotation. In addition to routing current through one of two

phases, the current direction through the selected phase must also be controlled. To do this, the stator control box contains a silicon-controlled rectifier (SCR) circuit and a custom printed circuit board (PCB). This SCR circuit is shown in Figure *2*.

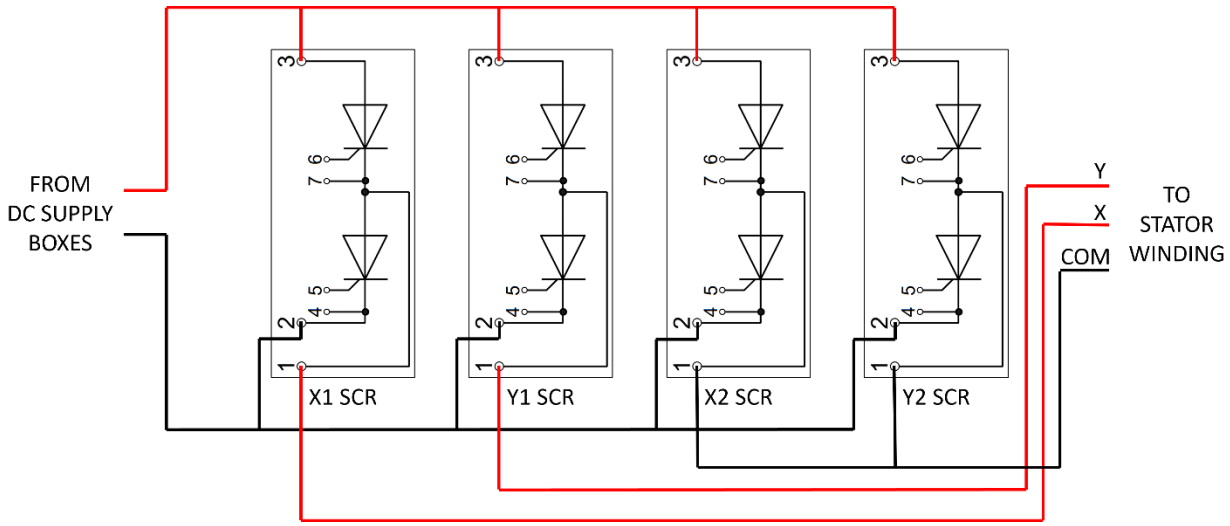


Figure 3 SCR circuit diagram

Figure *2* shows four Infineon ETT630N18P60HPSA1 SCR modules, each containing two individual SCRs and their gates. The gates, and therefore the SCR conduction, are controlled via the SCR module terminals 4 and 5, and 7 and 6. These gates are controlled by the custom PCB (see Appendix D – Stator Control Box). When selectively fired via this PCB, current direction and output terminal can be controlled for forward or reverse current out of the X or Y lead. By connecting the X and Y leads to different phases of the machine, and the COM lead to the machine's wye connection, two phases can be selectively controlled in either current direction.

Figure *4* shows the four current output configurations. In each image, the green trace is the current path through the SCR circuit and stator winding, with the green arrows along the current paths indicating direction through the stator winding. The outlined arrows in each image indicate which SCRs must be triggered to allow the current path shown. Which current path is output is simply a matter of which SCR gates are fired, which is ultimately controlled by the software. The hardware design documents for the stator control box can be seen in Appendix D – Stator Control Box.

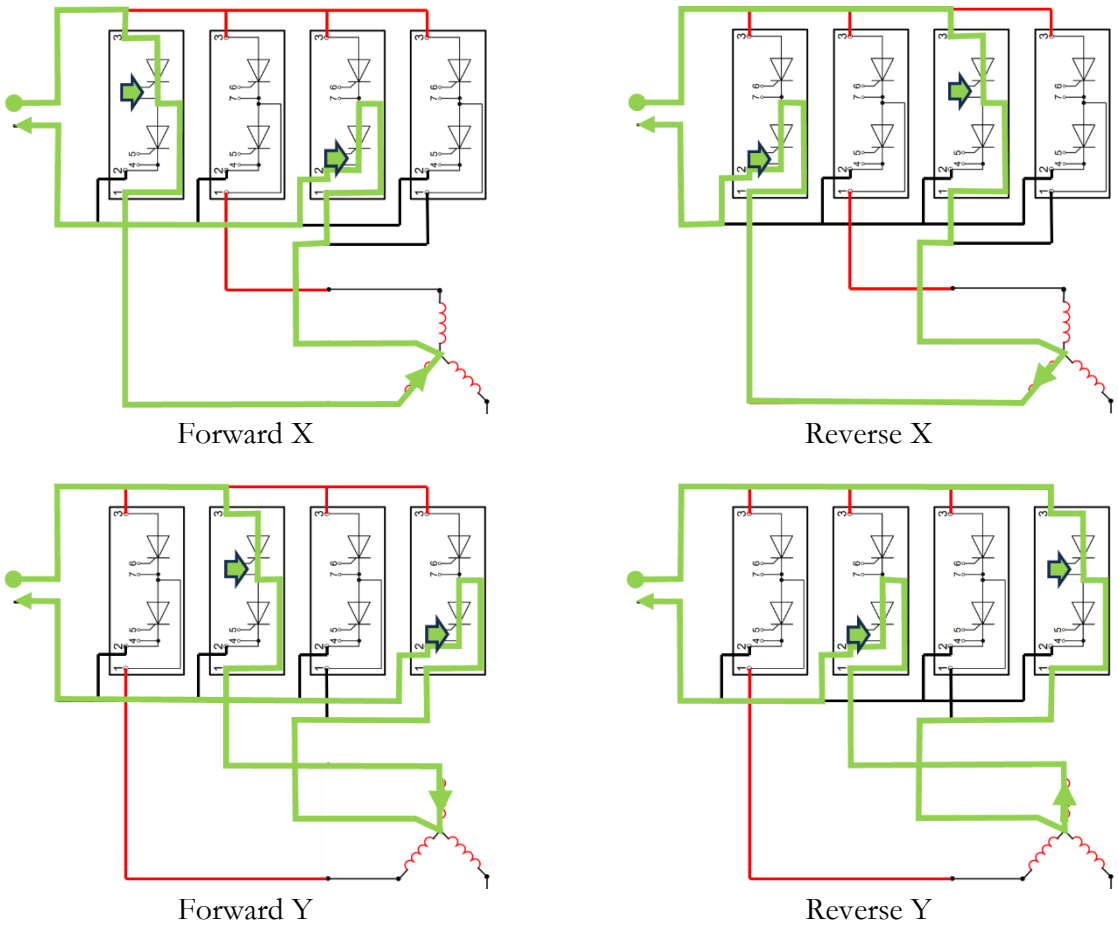


Figure 4 Four current configurations from selective gating of SCRs

2.2 Software

2.2.1 Distributed Software Architecture

The hardware discussed above is controlled via custom software. Figure *5* shows a more detailed depiction of the hardware interfaces involved. Within Figure *5*, the HATs are directly plugged into the respective CM4's 40-pin general purpose GPIO header, one on top of the other, in the order shown. The CM4 inside the rotor control box is the central command and control hardware which runs the RTTS software that orchestrates all tasks across the system by means of distributed computing. The RTTS software package is comprised of multiple subpackages, one for each control box within the RTTS. The rotor's subpackage includes a Flask-SocketIO application which is the user interface. The software for the user interface takes input from the RTTS operator via a browser application, interprets this input, and performs an action based on this interpretation. These actions may be triggered within the rotor box's hardware itself or distributed to other hardware within other control boxes connected to the same network. This is distributed computing.

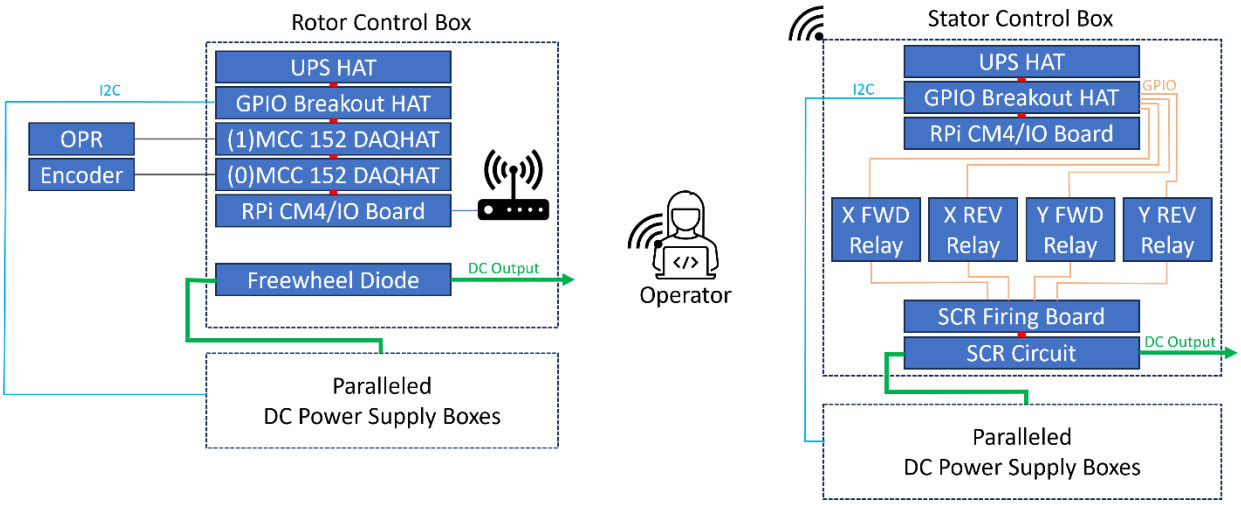


Figure 5 Hardware control block diagram

The distributed computing architecture is achieved via the Celery Python library, which distributes tasks across the CM4's throughout the RTTS via a message broker. The message broker of choice is Redis, which is an advanced in-memory key-value store. The in-memory storage and key-value structure of Redis means that it is very fast. Redis storage is within the CM4's random access memory (RAM) rather than the file system and key-value hash table lookups are of order one computational complexity. In addition to using Redis as a message broker, the RTTS also leverages the in-memory storage provided by Redis for caching and retrieving data used between multiple tasks and Python processes. Redis also allows for persistent storage dumps of in-memory data (i.e., storage in the file system). This persistence feature is used for storing test results and other data such as rotating machine parameters for future reference.

This distributed architecture means that each control box (e.g., rotor, stator, and eventually air gap and partial discharge mapping control boxes) has a CM4, and each CM4 runs Celery worker processes which process computing tasks queued from the message broker. The rotor control box's CM4 is unique in that it is also running two Redis instances and serving the user interface as a web application in addition to its Celery worker processes. In all cases, each of these pieces of software have been made to run as a Linux systemd service. Systemd is a system and service manager for Linux which helps increase fault tolerance and allows graceful shutdown and service start on boot.

When the rotor control box's CM4 boots up, systemd starts these services from their configuration files. First, two instances of Redis are started as services after the `network.target` unit is reached during boot (i.e., all network related services have started). The first Redis instance serves as the message broker for Celery, the user interface's WebSocket communication, and caching data between tasks and processes. The second Redis instance is only for persistent storage dumps to the file system of important data. Once the Redis services are started, the Celery service starts.

Every CM4 within the RTTS is set up to run a Celery service. The following service configuration is an example of the Celery service on the rotor control box's CM4. The only difference between the service file on the rotor CM4 and the service files on the other boxes' CM4s is that the rotor Celery service waits for the Redis services to be available before it is started (see lines 3 and 4). On the other boxes, there is no Redis service running, and the Celery service does not need to wait for these

targets. These other boxes connect to the rotor control box's Redis instance over the LAN via port 6379, the default Redis port. The bulk of the service file then relies on a Celery configuration file, which is different depending on the CM4 running the service.

```
1  [Unit]
2  Description=Celery Service
3  After=network.target redis-server.service redis-data-server.service
4  Wants=redis-server.service
5
6  [Service]
7  Type=simple
8  User=pi
9  Group=pi
10 EnvironmentFile=/etc/celery/celery.conf
11 WorkingDirectory=/home/pi/rtts/
12 ExecStart=/bin/sh -c '${CELERY_BIN} \
13     --app $CELERY_APP worker \
14     --queues=$CELERY_QUEUE \
15     --pidfile=${CELERYD_PID_FILE} \
16     --logfile=${CELERYD_LOG_FILE} \
17     --loglevel="${CELERYD_LOG_LEVEL}"'
18 ExecStop=/bin/sh -c '${CELERY_BIN} \
19     --app $CELERY_APP control shutdown'
20 ExecReload=/bin/sh -c '${CELERY_BIN} \
21     --app $CELERY_APP worker restart \
22     --queues=$CELERY_QUEUE \
23     --pidfile=${CELERYD_PID_FILE} \
24     --logfile=${CELERYD_LOG_FILE} \
25     --loglevel="${CELERYD_LOG_LEVEL}"'
26 Restart=always
27
28 [Install]
29 WantedBy=multi-user.target
```

An example of the Celery configuration file is shown in the code snippet below. The configuration file specifies parameters unique to the box running the celery service. The key parameters unique to each CM4 are the Celery application instance (line 8) and the queue the Celery workers listen to (line 11). The queue parameter defines which task queue the respective CM4's worker processes will listen to. These workers will only process tasks received in the queue specified. When the process receives a new task on the queue, the task name is referenced against the task names registered to the Celery application instance within the respective box's subpackage. By creating specific task queues for each control box within the RTTS, the rotor control box simply adds tasks to the respective box's queue, and that box's CM4 processes the tasks with the specialized hardware associated with that box. For example, only the stator control box's CM4 has tasks associated with controlling its SCR circuit. These tasks are defined within the stator subpackage and registered to its Celery application instance. Other boxes that do not have this SCR circuit hardware and capability

ignore these tasks because they listen to their own queue associated with their own tasks related to their own specialized hardware and functionality within the RTTS.

```
# Name of nodes to start
CELERYD_NODES="w1"

# Absolute path to the 'celery' command:
CELERY_BIN="/home/pi/rtts/.env/bin/celery"

# App instance to use
CELERY_APP="rtts.stator.cel"

# Queue to use
CELERY_QUEUE="stator_queue"

# - %n will be replaced with the first part of the nodename.
# - %I will be replaced with the current child process index
#   and is important when using the prefork pool to avoid race conditions.
CELERYD_PID_FILE="/home/pi/rtts/rtts/stator/os/celery/celery.pid"
CELERYD_LOG_FILE="/var/log/celery/%n%I.log"

# For development and diagnostics
# CELERYD_LOG_LEVEL="DEBUG"
# For production
CELERYD_LOG_LEVEL="INFO"
```

In addition to the Celery service configuration seen above, each subpackage's Celery application instance has its own Python configuration. Because the rotor control box's CM4 is the only invoker of tasks, the task queues are defined in the rotor's Celery application instance configuration as shown below. Specifically, the `task_routes` setting defines the queue routing. For example, any task name starting with `rtts.stator` will be routed to the stator queue. The configuration of the stator subpackage registers these task names (and the code associated with them) to the Celery application instance on the stator box's CM4. This means that when the Celery worker processes running on the stator box's CM4 see one of these task names on the `stator_queue` it will run the code associated with this task. The rotor box's CM4 does not need any knowledge of the actual code that is to be run on the stator box's CM4, instead, it only needs the task name associated with that code.

Future RTTS functionality can be easily gained by defining another box's task queue in this configuration file, and defining which user interface events should trigger the calling of those tasks. Then any code associated with those tasks can be defined within that box's subpackage. The configuration file shows an example of defining an `air_gap_queue` in the comments. These comments are intended to guide future software engineers in incorporating future functionality such as air gap measurement and partial discharge mapping. All other boxes have a similar Python configuration for their Celery application instance except no task route is defined (as these CM4s do not invoke/route tasks). Thes other boxes register task code associated with their boxes

functionality (e.g., `imports = ["rtts.stator.tasks"]` for the stator box's subpackage instead of `imports = ["rtts.rotor.tasks"]`).

```
from pathlib import Path

accept_content = ["pickle", "json"]
task_routes = {
    "rtts.rotor.*": {"queue": "rotor_queue"},
    "rtts.stator.*": {"queue": "stator_queue"},
    # future queues here (e.g. air gap control, patch antenna control)
    # e.g.: "rtts.air_gap.*": {"queue": "air_gap_queue"} air gap tasks
}
imports = ["rtts.rotor.tasks"]
broker_connection_retry_on_startup = True
pidfile = str(Path().resolve() / 'os/celery/celery.pid')
worker_cancel_long_running_tasks_on_connection_loss = True
```

An example of the associated task file that is imported from the `imports` configuration mentioned above is shown below. Specifically, this is a single task (of several) from the stator subpackage's task file. All subpackages have task files that work the same way, except that they would contain code specific to their box's hardware and functionality within the RTTS. The task shown allows the RTTS operator to scan the I2C network connected to the stator box to determine what DC power supply boxes are connected and ready to operate.

The code below imports the main package's `bus` object, which gives access to all the custom I2C-related methods needed to work with the I2C bus and communicate with connected power supply boxes. The subpackage's Celery application instance and SocketIO objects are also imported. The Celery application instance is used as a decorator to the actual task function, which sets the task name and registers the function code to its instance. After the I2C scan, the `socketio` object emits the scan results to the frontend via the WebSocket protocol for the RTTS operator to view.

```python
from rtts import bus
from rtts.stator import cel, socketio

@cel.task(name="rtts.stator.scan_pmbus")
def scan_pmbus():
    current_limits = {}

    response = bus.scan()

    if bus.detected_addresses:
        bus.broadcast(bus.WRITE_PROTECT, 0x00)

    for address, current_limit in response.items():
        if current_limit[0] is not None:
            current_limits[f"0x{address:02X}"] = current_limit

    socketio.emit("stator_currents_response", current_limits)
```

The above task to scan the stator box's I2C bus is invoked from the rotor subpackage from within the events monitoring code. The specific event for the stator I2C scan is shown below, where the `socketio` decorator is the Flask-SocketIO application instance running the user interface. This decorator registers the event function code to the specific event. When the frontend sends the `"request_stator_i2c_scan"` event, the decorated function is called. This function causes the rotor's Celery application instance to queue the `rtts.stator.scan_pmbus` task, which is sent over the network to the stator box for distributed processing.

```python
from rtts.rotor import cel, socketio

@socketio.on("request_stator_i2c_scan")
def request_stator_i2c_scan():
    cel.send_task(
        "rtts.stator.scan_pmbus",
    )
```

The frontend sends the `"request_stator_i2c_scan"` event via the below JavaScript code. Specifically, when the appropriate button on the browser page (rendered via a different HTML file) is clicked, the frontend's SocketIO object emits the event to the server's backend Python code.

```javascript
var socket = io.connect(location.origin);
var stator_i2c_scan_button = document.getElementById("stator_i2c_scan_button");

stator_i2c_scan_button.onclick = function() {
    socket.emit("request_stator_i2c_scan");
}
```

A screen shot for the user interface portion of this scanning feature appears in Figure 6. On page load, the "Scan I2C Bus" button is the only element showing. After clicking this button, if DC power supply boxes are connected to the bus and powered on, the stator task returns the information shown below the button. This includes a table of all connected boxes with their power supply's I2C address and current limit setting, with a total output capability totaled at the bottom of the table (if multiple power supply boxes are connected).

An interactive slider is also shown below the table. The user can click and drag the green slider handle to set all connected power supply current limits to a value within their maximum output range. Like the scan button, when the user releases the slider handle, the frontend JavaScript code emits an event to the backend event code running on the rotor control box's CM4. This event code parses the specific event and queues a task to perform this action. The stator box's CM4 listens to this queue and runs the code associated with this task to set all connected power supply current limits. This is how all user input works. A user action on the page emits an event over the network to the rotor box, which parses the event and queues the task. This task may be carried out on the rotor box or sent back over the network to the appropriate box within the RTTS to be carried out on its hardware. In many cases, the tasks also contain code to send information regarding task completion over the network back to the frontend for parsing and display by the client-side code.

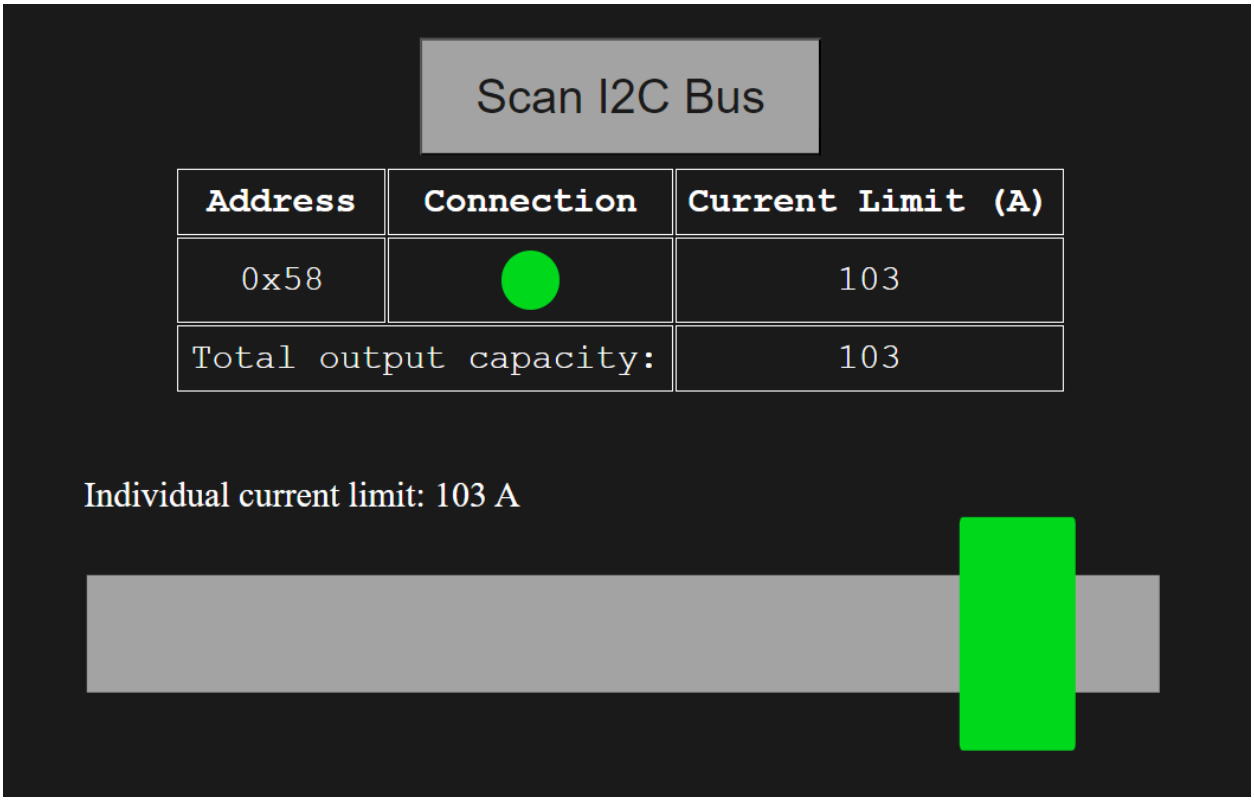


Figure 6 View of I2C bus scan results.

2.2.2 User Interface

The user views and their routes (i.e., the browser application pages and their relative uniform resource locators (URL)) are defined in the usual way using the Python Flask library. Specifically, the

Flask application instance registers each route with a decorator. The decorated function then becomes the code that is run when the user navigates to the view route from their browser. In addition to processing form submissions and performing various backend logic tasks, this function code ultimately specifies which HTML file to render to the user. This HTML file determines what the user sees in their browser and links to the JavaScript code that determines how the user can interact with what they see. Each route and its associated view within the RTTS application serves a unique purpose. As of this writing, the following URL routes, relative to the root URL, and their associated views, are defined within the RTTS application:

1. /
    a. The base index route with a view showing a warning disclaimer. This disclaimer must be acknowledged to access any other view.
2. /menu
    a. This view is presented after acknowledging the index view's warning disclaimer. This view is the main menu from which the user can access all the functionality of the RTTS application. As of this writing, the following selections are possible:
        i. Setup
        ii. Manual Turn
        iii. Map Machine
        iv. Smart Turning
        v. Air Gap Testing
3. /setup
    a. This view allows the user to scan the I2C bus from both the rotor control box and the stator control box. Once scanning is complete and the connected power supply boxes are listed, the user can also set current limits from this view.
4. /manual_turn
    a. This view allows the user to control the stator field current. The user can select which output port (X or Y) to send current through. The user can then click and hold a "FORWARD" or "REVERSE" button to output forward and reverse current respectively through the selected output port. Current is output for as long as the user holds the button.
5. /collect_metadata
    a. The user is redirected to this view when they select "Map Machine" from the main menu. This view is a form where the user can enter machine related data required for machine mapping. The following data is required to be entered by the user:
        i. Facility name
        ii. Asset name
        iii. Encoder drive surface measured circumference (feet)
            1. This is the approximate circumference of the encoder drive surface, as measured by the user using a tape measure or other means.
        iv. Number of rotor poles
            1. Allows the RTTS software to convert electrical degrees to mechanical degrees or vice versa.
        v. Number of stator slots
            1. Allows the RTTS software to determine relative stator position from a known starting position.
        vi. Direction encoder is mounted (up or down)
6. /map

   a. This route always redirects to the base index view unless it is accessed via form submission from the /collect_metadata view. This view shows an oscilloscope which reports rotor position in real time in graphical form. The intent is to turn the rotor while the software monitors the OPR and encoder signals to measure the exact circumference of the encoder drive surface and determine the drive ratio.
7. /load_map
   a. This view is a form to load a previously stored machine map from the facility and asset names.
8. /smart_turning
   a. This view is not yet fully functional, as is intended to provide all the automated turning functionality of the RTTS system. It can only be accessed through the /load_map form submission.
9. /air_gap
   a. This view is a placeholder view for implementing air gap measurement functionality in the future.

2.2.3 Mapping and Automated Turning

Machine mapping is a necessary step to automated turning of the rotor. A machine map provides the information needed to apply stator current based on encoder signal alone. In other words, if encoder position is known, the RTTS software should be able to determine the required current output state after referencing the machine map. While further development is needed to fully automate turning, the RTTS map functionality has been mostly worked out within this research.

At a minimum the machine map would contain rotor pole count and the precise encoder drive surface circumference as measured from the OPR and encoder itself. If these parameters are known, theoretically, automated turning would proceed as follows:

1. The user enters useful machine parameters, including the number of rotor poles, and initiates the mapping function within the RTTS application.
2. The rotor must turn at least one full revolution. This step tracks the number of encoder positions for exactly one rotor revolution, as determined from the OPR signal. Turning the rotor for one revolution can be achieved in this step in one of two ways:
   a. The user manually turns the rotor through one revolution from the /manual_turn view within the RTTS application.
   b. The RTTS automatically turns the rotor through one revolution based on an approximate machine map generated from approximate encoder drive surface circumference as measured by the user with a tape measure or other means.
3. The map information (e.g., at a minimum, the drive ratio and rotor pole count) is stored persistently for later recall.
4. The rotor must start from an aligned reference location.
   a. The RTTS applies forward current to the Y lead.
      i. If rotor rotation is detected, current output is held until rotor rotation stops. The rotor is now aligned with the stator phase connected to the Y lead.
      ii. Otherwise, the RTTS applies forward current to the X lead.
         1. If rotor rotation is detected, current output is held until rotor rotation stops. The rotor is now aligned with the stator phase connected to the X lead.

    2. Otherwise, rotor rotation is not possible with the current configuration. Increase DC power output or assess issues that may be causing excess friction in rotor turning.
5. The RTTS zeros the encoder position. All rotor rotation is now measured relative to the position obtained in step 5. This is a position of known alignment and is 120 electrical degrees from the phase connected to the lead that was not aligned in step 4.
6. A map is calculated from the information obtained in step 3 which maps encoder position to needed current output state. For example, if the rotor was aligned to the stator phase connected to the Y lead in step 5, then the needed current output state to get forward rotation is forward current out of lead X. The machine map would calculate a state (e.g., position 0 maps to X-forward) for every encoder position and store them in a lookup table for automated rotation.
7. The RTTS checks encoder position.
8. The RTTS looks up what state is needed to turn the rotor in the user defined manner from the lookup table created in step 6.
9. The RTTS applies the output state determined in step 8.
10. The rotor turns, changing the encoder position.
11. The process is repeated until the desired rotor position is reached or the desired testing is completed.

Note that in the above process for automated turning, the map only determines basic output states and not current quantities. The current quantity applied during a calculated output state is what determines the magnitude of torque applied. This magnitude will need to be determined on a case-by-case basis and requires significant feedback control. This is the major area of research and development that still needs to be carried out. For example, if the user needs the rotor to only turn a very small, predefined amount, applying too much current for too long in the calculated output state would cause the rotor to overshoot the target position. Because the RTTS software monitors encoder position, it could automatically detect the overshoot and automatically correct. However, if feedback processing is insufficient, the RTTS may overshoot in both directions and never reach the desired rotor position. For this reason, automatically turning the rotor in precise increments may not be feasible. However, precise rotor location while manually turning the rotor is already achievable, and any user that needs rotor rotation in precise increments may need to settle for manually turning the rotor (timing the click of the forward and reverse button) while manually monitoring rotor position from the RTTS application. Such a need is common during machine inspection and bearing alignment.

A simpler automated turning case would be constant rotation. In this case, the automated procedure above could be applied with full current in each output state. The RTTS software could monitor rotation speed from the change in encoder position per unit time. If rotation speed increases beyond some predefined threshold, the automated process would stop applying current, letting the rotor coast. Once the rotation speed decreased below another predefined lower threshold, the automated process could start again. In this way, the RTTS could maintain constant rotation (though not at constant speed). This type of turning would be acceptable for several slow-roll diagnostic procedures, including air gap measurement and partial discharge mapping.

Diagnostic procedures that rely on constant turning often require mounting a sensor on a rotor pole face and mapping readings from this sensor to stator slot position (or core tooth position). Because the RTTS software gains access to stator slot count during the machine mapping procedure (it is

entered by the user before mapping can take place), the RTTS can potentially track sensor position relative to slot number. Such a feature could be guided via the view control logic. Any functionality that requires tracking a sensor on a rotor pole face would only allow access to its related views through another web form. This form would ask the user to record which slot the sensor is in front of just after the alignment and encoder zeroing stage in the automated procedure described above. Constant, automated rotation could then track which slot (or core tooth) the sensor is in front of based on encoder position and slot count from that point forward.

This automated slot tracking feature would be very useful, as presently these tests use proximity probes and post processing of the voltage recoded from these proximity probes to determine slot position. For example, air gap measurement data is presently mapped to stator position from the data itself. This data is post processed to find local extrema within the proximity probe's voltage signal. These extrema correspond to slots and core teeth, and once found in post processing, they must be counted to track sensor location from a given reference point. With the RTTS software, this post processing step would not need to be carried out, and the RTTS being able to directly determine sensor position from encoder position would allow real time air gap measurements to be displayed to the user. For partial discharge mapping, the benefits are even greater. The complexity of the partial discharge mapping system could be greatly reduced given this RTTS position tracking functionality.

2.2.4 RTTS Application Deployment

Because of the modular nature of the RTTS system, the software package must be installed on multiple CM4s. This is done through Git, which is primarily a version control software toolset. In addition to version control change tracking within the codebase, Git provides many features for continuous integration. Specifically, we have utilized server-side hooks on each CM4 to facilitate deployment of the software codebase to each box's hardware. For efficiency, development and maintenance of the codebase is not performed on the individual CM4s. Rather, the codebase is modified and maintained from a centralized machine that can have a more feature rich environment (e.g., a graphical user interface, optimized code editors, language linters, etc.) that the CM4s do not need. The proper dissemination of the codebase (i.e., deployment) is then taken care of automatically using Git hooks.

To perform this continuous integration, each CM4 contains a remote repository. These remote repositories are treated as origin repositories and are therefore created bare. Within the same directory as these bare repositories, a clone is made, which becomes the running codebase on each CM4. The bare repositories on each CM4 contains a post-receive hook. Git runs the bash script within each CM4's post-receive hook whenever changes are pushed to that CM4's bare repository.

The development machine has a client-side post-commit hook that pushes to all remote origins after each commit. In this way, every time a change is committed to the codebase by the software engineer, the change is automatically pushed to each CM4 within the RTTS and each CM4 automatically processes the change to update the running software on the CM4 and restart the RTTS application services.

Each CM4's post-receive hook is unique to that CM4's software requirements. For example, the rotor control box is the central network node of the RTTS, and thus also serves as the domain name server (DNS) for the LAN network. Therefore, part of the script within the post-receive hook on

the rotor control box's CM4 is dedicated to moving the DNS related files to their appropriate location within the files system upon receiving changes.

In addition to moving specific files to specific locations depending on the CM4 the hook is running on, these hooks also set up the software dependencies on their respective hosts. The Python standard for specifying package metadata is the pyproject.toml file within the root directory of the codebase. This file lists important information, including package dependencies like Celery and Flask. Most of these dependencies are listed on the Python Package Index (PyPI) and can easily be downloaded and installed from a Python pip command. However, the daqhats package, which is used to interact with the MCC DAQ HATs hardware, is a relative dependency.

The codebase for the daqhats packages is cloned from GitHub and stored along with the RTTS codebase. This means that to install it along with the codebase, Python pip must know the relative path to the cloned repository within the codebase itself. However, because the relative path may be different on each CM4, the relative path listed within the pyproject.toml file's dependency list must also update based on the CM4 running the software. This is also done with the post-receive hooks and a custom toml_mod.py file. Specifically, the post-receive hook script runs the toml_mod.py Python script, which in turn updates the dependencies in the pyproject.toml file on the respective CM4. In this way, each CM4's pyproject.toml file contains the dependencies with the correct relative path to the daqhats package. This allows the whole RTTS software package to be installed with a single Python pip command. This command can also be run from the post-receive hook to install the RTTS package after updating the relative dependencies.

This method of continuous integration means that, once reliable hooks have been coded and installed on all the CM4s, the software engineer need only commit changes to the codebase and the hooks automatically update these changes to all devices within the RTTS. Without this continuous integration, the software engineer would have to start remote secure shells (SSH) into each CM4 and manually perform many different commands to properly update all the software throughout each CM4's operating system. In development, this would have to happen in order to test even the smallest changes to the codebase.

3. Future Advancement

As discussed earlier, one of the major design intents when engineering the RTTS software was to allow future functionality and enhancements to be easily incorporated. Two enhancements that could be almost immediately incorporated into the RTTS are air gap measurement and partial discharge mapping, both of which require rotor rotation to be carried out. In addition to these new features, a tool was also invented, and its feasibility studied for incorporating into the RTTS system as an accessory to measure the approximate circumference of the encoder drive surface.

3.1 Automation

As mentioned earlier, automation of rotor turning is likely possible. While the RTTS software has been engineered to facilitate exploration and development of this automation more easily, full

automation has not yet been realized. The RTTS software resulting from this research greatly increases the ease with which automation logic can be tried, measured for effectiveness, and iterated upon. The RTTS produced in this research project will be very useful and routinely deployed into the field for turning procedures. Its ease of deployment and modular design will allow the researcher to try different automation methods and quickly iterate on their findings, advancing and eventually implementing fully automated capability within the software in the near future.

3.1 Air Gap Measurement

The air gap measurement box would have its own CM4 and MCC DAQ HAT. The DAQ HAT would measure voltages from proximity probes mounted on the rotor and stator. The air gap CM4 would have tasks defined within its subpackage to collect and process these voltages, correlating them to distance between the rotor and stator (i.e., air gap). By communicating with the rotor control box (which tracks rotor position), these distance measurements can be mapped to precise locations of the individual rotor pole faces or stator core teeth. These mapped measurements can also be analyzed to determine metrics such as machine circularity and concentricity, and these metrics can be displayed in real time to the user. These metrics could also be sent back to the rotor control box's CM4 for persistent storage for trending across future air gap measurements.

3.2 Partial Discharge Mapping

Mapping partial discharge has been investigated in previous research (Lapenna, 2021). This diagnostic method would require another box with its own CM4 and its own tasks. These tasks would allow the CM4 to process signal from a near-field communication (NFC) antenna connected through specialized partial discharge monitoring hardware. In this way, partial discharge events can be detected across the air gap with the NFC antenna, mounted on a rotor pole face, moving past every slot within the stator winding as the rotor is turned. By communicating with the rotor control box to determine rotor position, these partial discharge events can be mapped to the slot from which they originate. When this mapping is trended over years, deep insight into machine condition and aging can be gleaned by analyzing how the partial discharge changes throughout the machine over time.

3.2 Circumference Tool

Another area of RTTS system enhancement would be in the way the user could measure the approximate circumference of the encoder drive surface. Recall that, if the mapping procedure is to be fully automated, the RTTS software needs to reference the approximate encoder drive circumference as measured by the RTTS operator. This approximate circumference would be used to generate an initial, approximate machine map used to automatically obtain the more precise actual machine map.

The ideal location to drive the encoder is off one of the field winding slip rings. This is because the machine's slip rings are typically very clean machine surfaces with defined tolerances for roundness. This means they provide a reliable and mechanically stable drive surface with good traction. In

addition, the slip rings are located physically close to the rotor control box, as the rotor current supply output must be connected to the brush rigging to inject current into the field winding. Because of this brush rigging, it is often cumbersome for the user to measure slip ring circumference, as all the brushes get in the way of a tape measure. To overcome this burden, a tool has been investigated which could closely approximate encoder drive surface circumference while only accessing a small portion of the surface (i.e., the user does not need to string a tape measure over the entire circumference).

An illustration of the invented tool is shown in Figure 7 where the geometry used to calculate circumference is also shown. The invented tool is shown in solid colors within Figure 7. It is constructed of a sturdy bracket (solid grey) of precise geometry. Specifically, the distance between the points where the bracket contacts the circle's circumference is known to very high precision and accuracy. Mounted to this bracket is a sensitive laser distance sensor (purple) which precisely measures the distance from its face to objects in front of it.

This distance sensor would be mounted to the bracket on a precision slider such that it can travel back and forth over the midpoint of the bracket (as indicated by the arrow) while measuring the distance to the circumference of the circle. The shortest recorded distance along the sensors travel path is then the point of the circle's circumference that is exactly in the middle of the bracket contact points. Note that for this to be the case, as the sensor travels across the midpoint, it cannot have any radial travel toward or away from the circumference surface. Then, if the distance between the sensor face and the bracket contact points is known, the circumferential bulge at the midpoint between the bracket contact points can be calculated. This creates the geometry shown in Figure 7, where the red lines show the known values D (the distance between bracket contact points), and l (the maximum circumferential bulge at the midpoint of D). These known values provide all the information needed to calculate the circle's circumference.

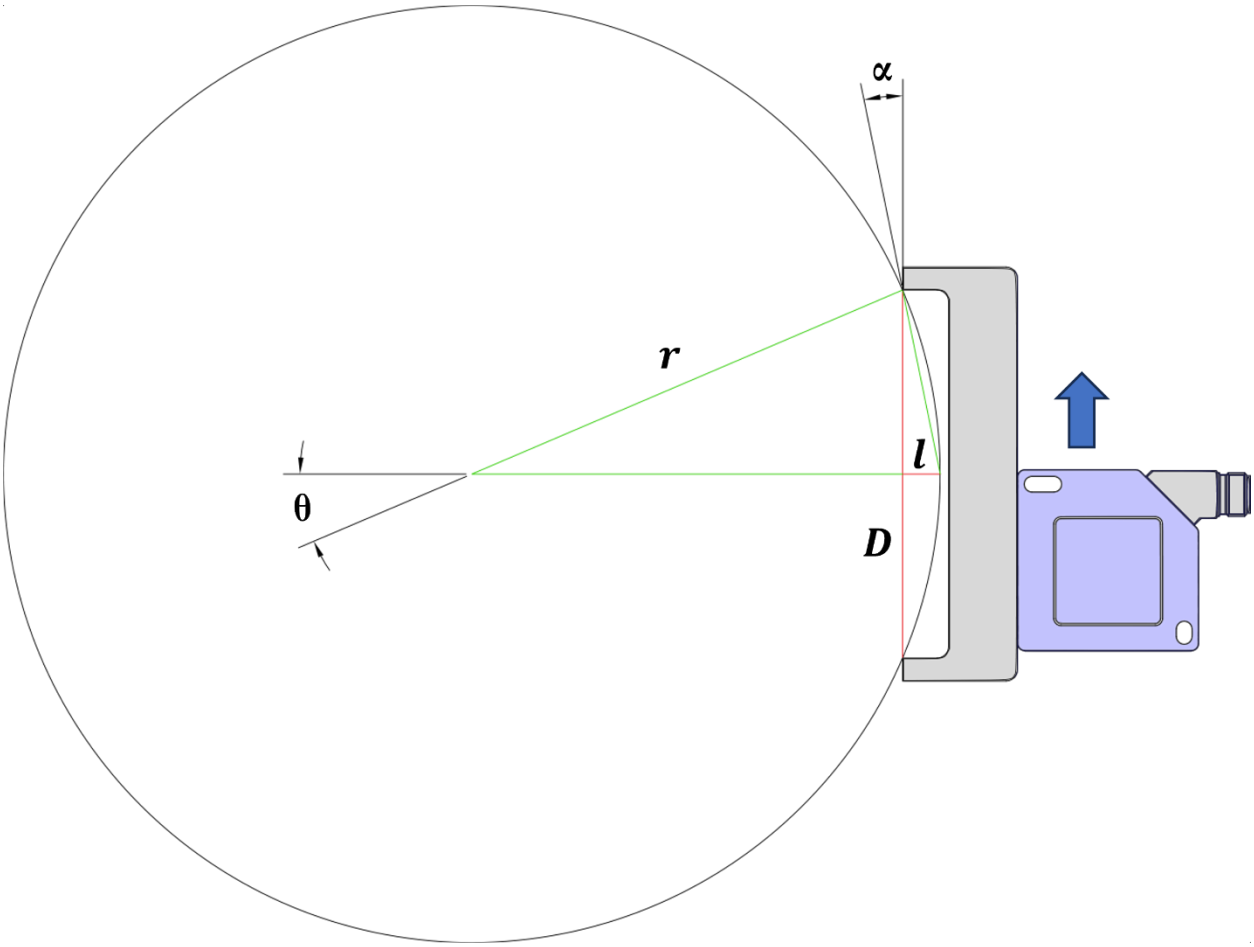


Figure 7 Invented circumference tool.

To calculate the circle's circumference from D and l, first note the isosceles triangle formed by the circle's radii shown in green within Figure 7. The apex of this isosceles triangle is formed by angle θ. Moreover, by noting that, for $\theta < 90°$ (our case), $\propto = \theta/2$, the sine of the angles can be written in terms of known values:

$$\sin\left(\frac{\theta}{2}\right) = \sin(\alpha) = \frac{l}{\sqrt{\left(\frac{D}{2}\right)^2 + l^2}}$$

The trigonometric half-angle formula for the sine function, for $0° < \theta < 90°$, then gives:

$$\frac{l}{\sqrt{\left(\frac{D}{2}\right)^2 + l^2}} = \sqrt{\frac{1 - \cos(\theta)}{2}}$$

The final step to solve this equation is to write $\cos(\theta)$ in terms of r and l.

$$\cos(\theta) = \frac{r - l}{r}$$

Substituting this into the equation, solving for r, and multiplying by 2π to get the circumference of the circle yields the final solution:

$$C = \frac{\pi\left(\left(\frac{D}{2}\right)^2 + l^2\right)}{l}$$

Note that this gives an infinite circumference for a flat surface ($l = 0$). This makes sense, as the circumference gets very large (approaches infinity), the bulge between the bracket contact points approaches zero. Further note that if the tool were to be applied to a concave surface, the sensor would measure a depression, and the calculation would be the same but calculating from the inside diameter of the circumference. The smallest circumference that can be measured is determined by the depth of the bracket between the contact points. This bracket depth dimension must be less than $D/2$, otherwise the circumference surface would bottom out on the bracket without touching both contact points. Also note that the bracket contact edges would be machined to be 90-degree edges. If these edges are radiused, they would contact the circumference tangential to their radius and D would vary in an unknown way with varying circle circumference.

It is also important to note that any error in the bulge measurement would propagate to the circumference calculation. For this reason, a potentially more accurate and simple design of the tool could do without the laser distance sensor and use a depth dial indicator that is manually slid along a window at the back of the bracket. The user could record the minimum depth and then type this value into a form within the RTTS application. The RTTS application could convert this depth value to l from the known dimensions of the bracket, and then use l to calculate the circumference automatically within the software.

4. Conclusions

The RTTS represents a breakthrough in rotating machine diagnostics. Where, previously, test engineers were beholden to disparate diagnostic tools across different manufacturers to carry out limited diagnostics, the RTTS now makes it possible to incorporate these tests into one easy to use system while extending the diagnostics in previously impossible ways. The RTTS operator needs no specialized training or skillset. If they have used a web browser, they can operate the RTTS. This puts all slow-roll maintenance and diagnostic procedures in the hands of crafts personnel at the facilities where the tasks are carried out. While these maintenance and diagnostic procedures have been made more efficient to carry out, the greatest benefit will be realized in increased frequency of testing. Because the RTTS is so easy to deploy, and because the RTTS can be operated by facility personnel instead of flying in a specialized test engineer, this testing can be incorporated into annual routines at each facility. This will produce more frequent, higher quality diagnostic data, which will directly result in greater insight into machine condition and aging trends. In turn, this greater insight

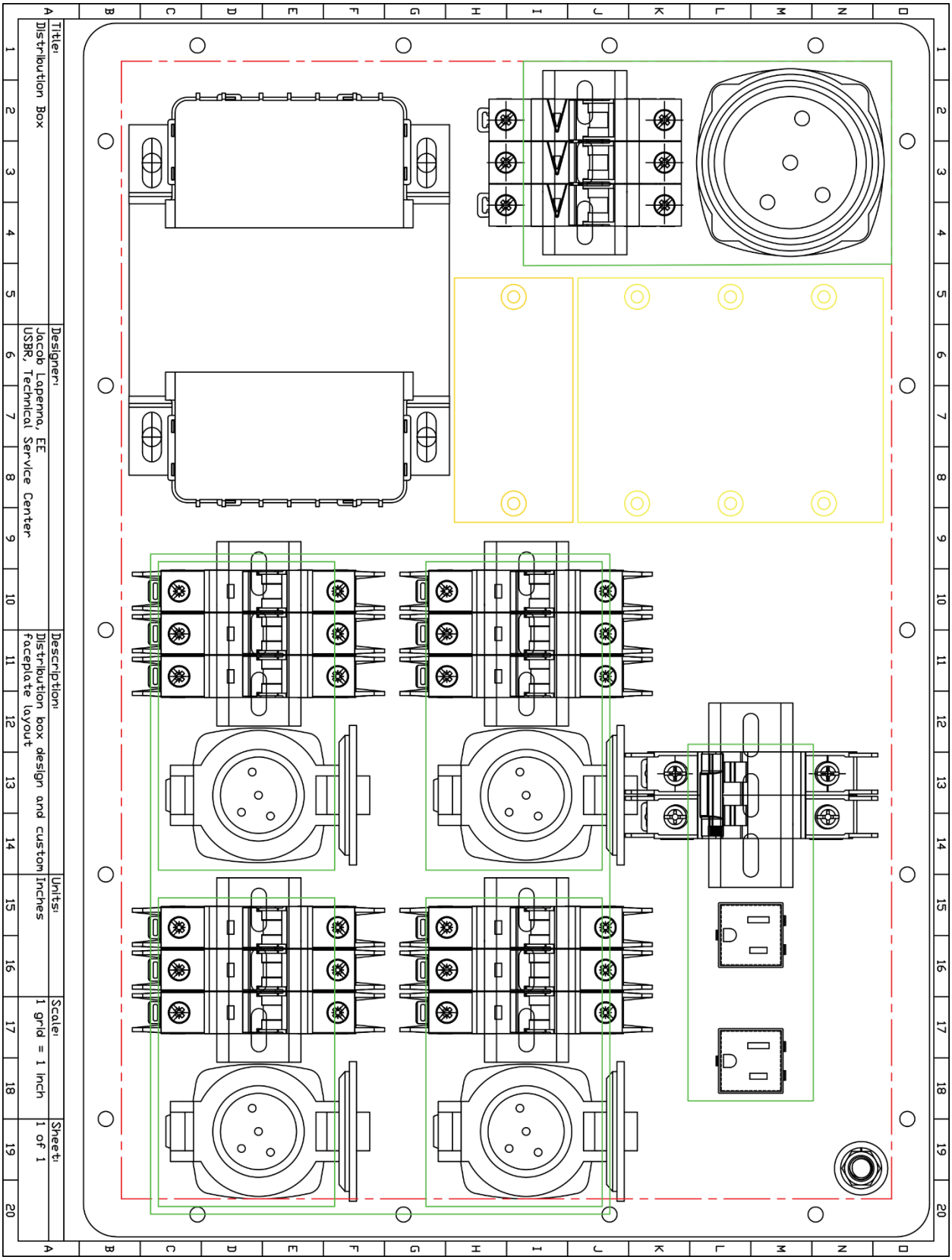
will ultimately allow Reclamation to carry out its core mission in a more efficient manner than previously possible without the RTTS. In addition, the RTTS has set a precedent in the advancement of traditional diagnostic and maintenance procedures. The RTTS is not simply a rotor turning device, it is a system that will allow the collection, analysis, reporting, and maintenance of specific data sources in ways that were not previously possible. This project has opened the doors to our internal capabilities in engineering advanced systems to bring together data across disciplines and throughout Reclamation.

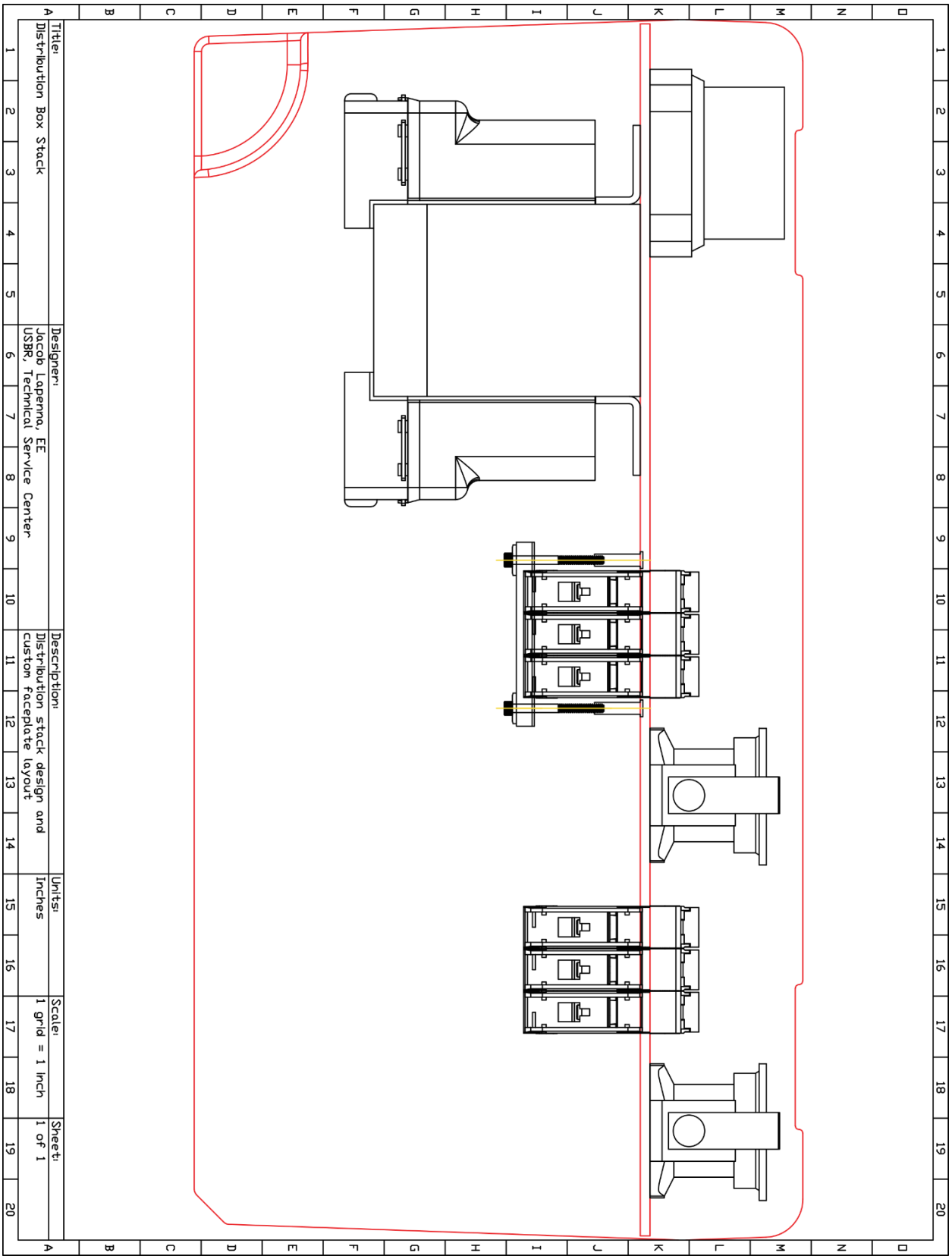
References

Lapenna, J. 2021. Rotor Installed Corona Mapping of Sator Windings within Large Diameter Hydro Generators. https://usbr.gov/research/projects/detail.cfm?id=19078. 11/16/2023.

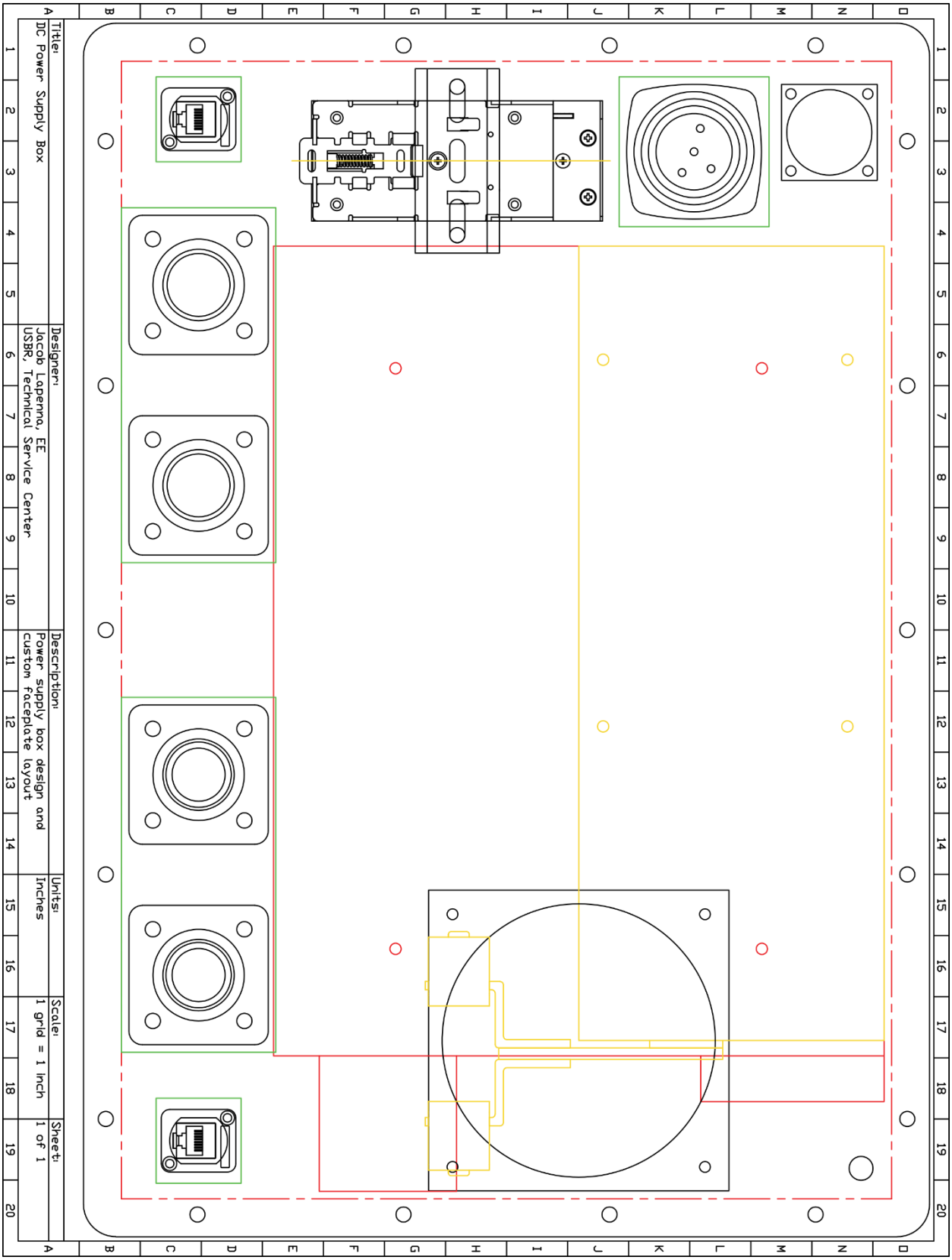
Taylor, J. R. (1997). An Introduction to Error Analysis, the Study of Uncertainties in Physical Measurements (2nd ed., pp. 73-75). University Science Books.

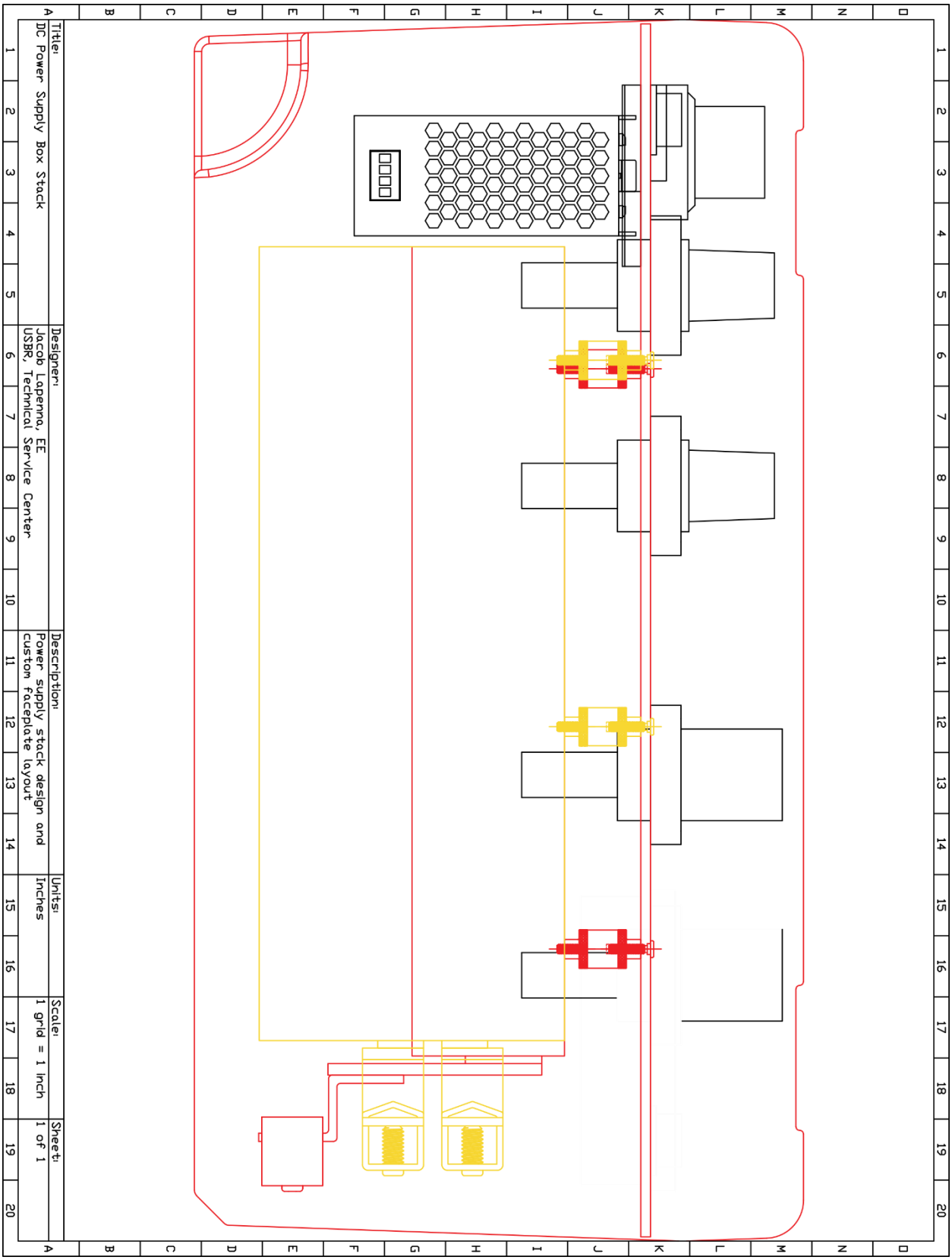
Appendix A – Distribution Box



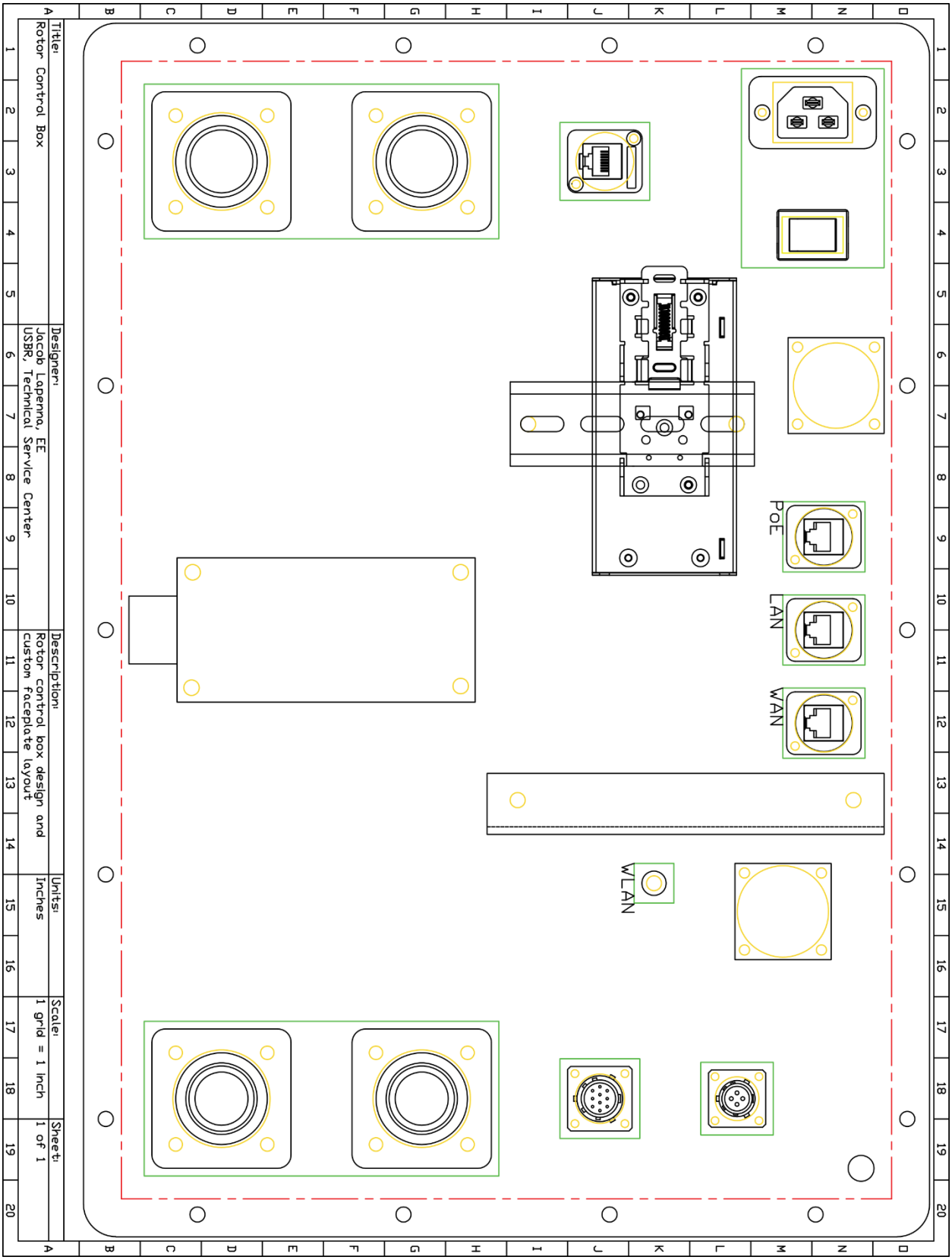

Title:
Distribution Box Stack
Designer:
Jacob Laperna, EE
USBR, Technical Service Center
Description:
Distribution stack design and custom faceplate layout
Units:
Inches
Scale:
1 grid = 1 inch
Sheet:
1 of 1

Appendix B – DC Supply Box

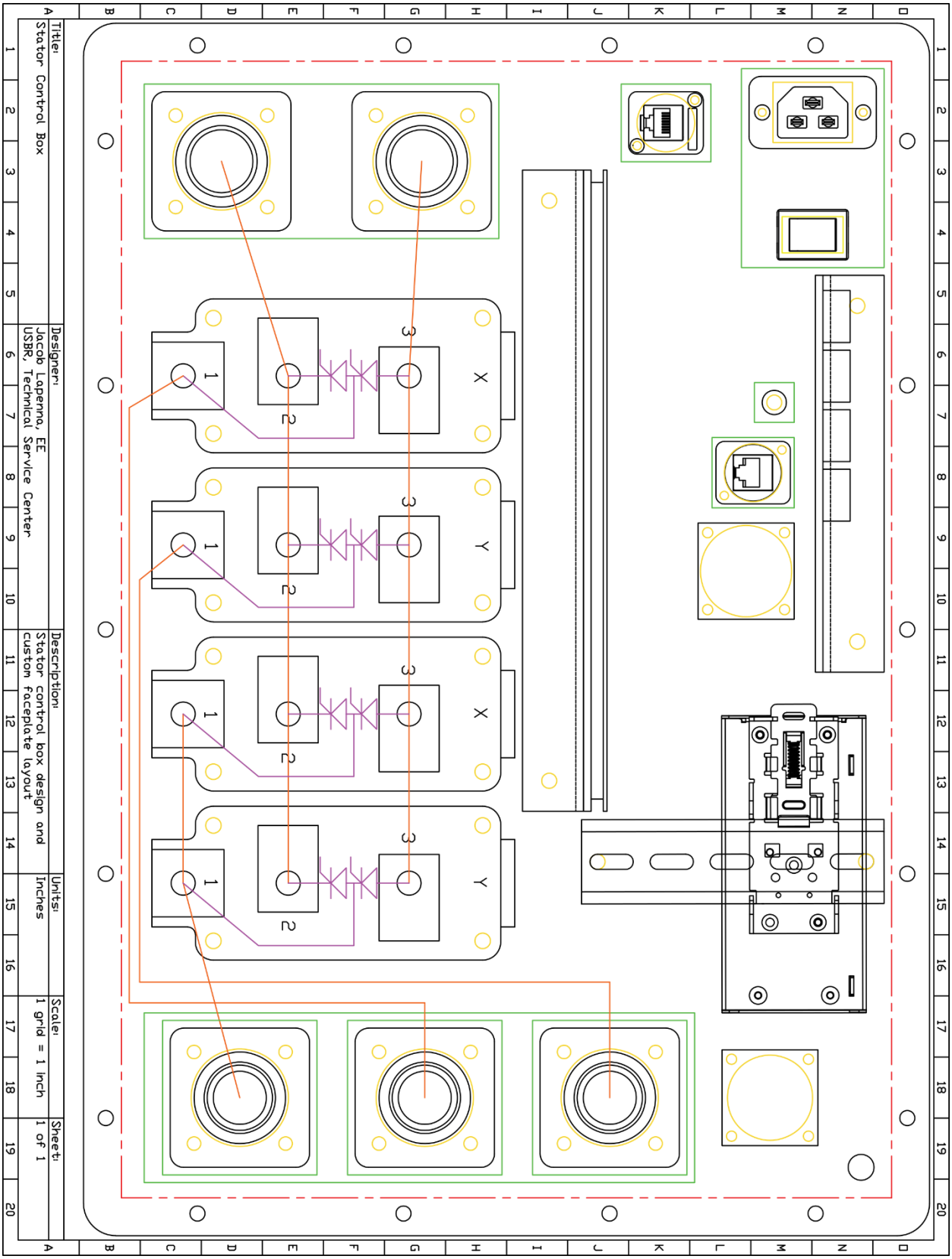


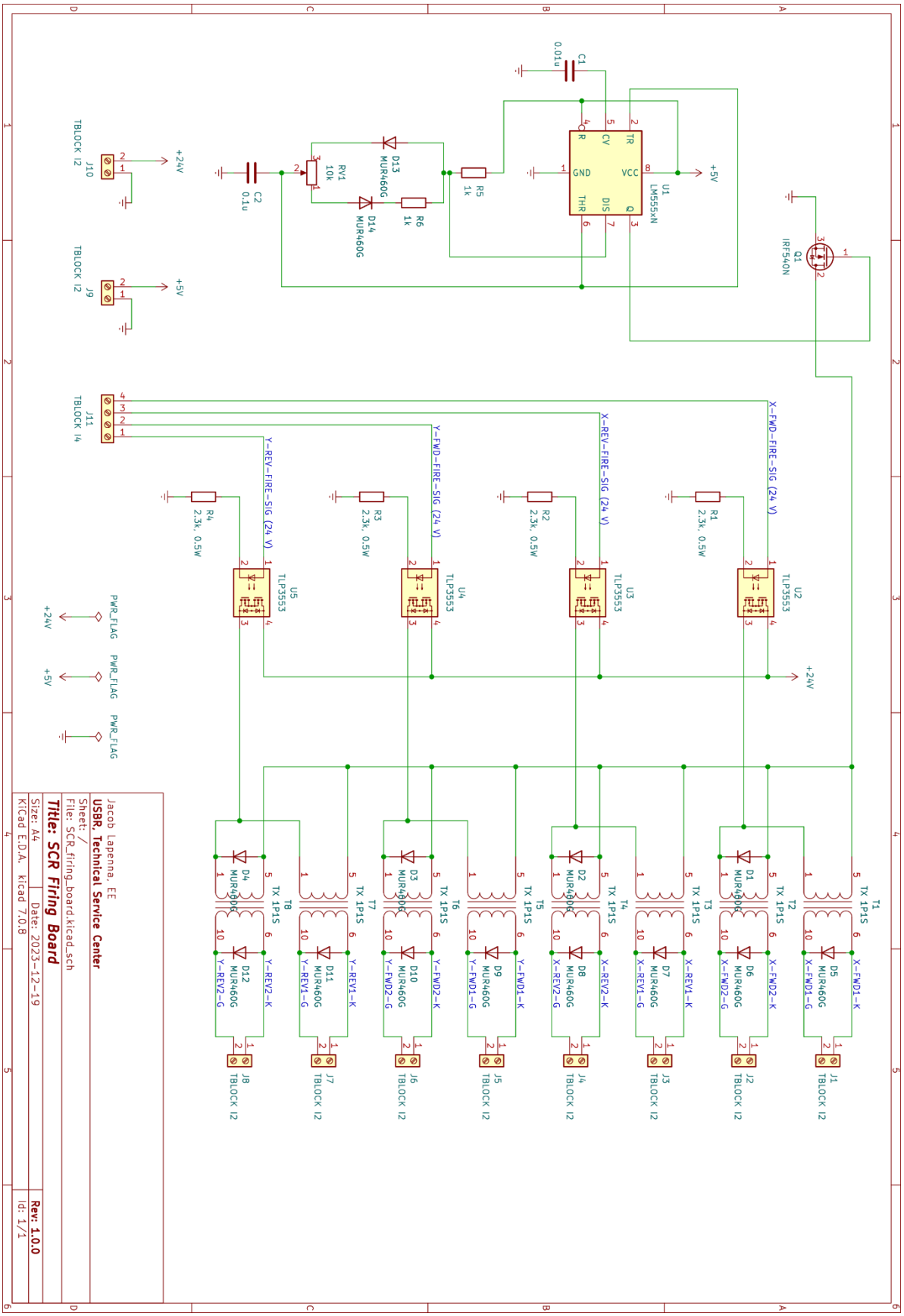

Title:
DC Power Supply Box Stack
Designer:
Jacob Lapenna, EE
USBR, Technical Service Center
Description:
Power supply stack design and custom faceplate layout
Units:
Inches
Scale:
1 grid = 1 inch
Sheet:
1 of 1

Appendix C – Rotor Control Box



Appendix D – Stator Control Box




Jacob Lapenna, EE
USBR, Technical Service Center
Sheet: /
File: SCR_firing_board.kicad_sch
Title: SCR Firing Board
Size: A4
Date: 2023-12-19
KiCad E.D.A. kicad 7.0.8
Rev: 1.0.0
Id: 1/1