OEM Controls

Automated Angle Sensor Testing

by

Abdulmalik Suleiman, Dacheng Shen, Fernanda Hanashiro, Haoyu Zhang, Nidhish Yarlagadda, Nyamdemberel Munkhbat, Zijiang Zhao

COLLEGE OF ENGINEERING
UNIVERSITY OF CONNECTICUT

December 16, 2023

TABLE OF CONTENTS

CHAPTER 1

Introduction

1.1 Project Background

OEM Controls is a design and manufacturing company based in CT focused on customizing rugged joysticks and electronics for mobile equipment. They have developed the AS5, an Angle Sensor whose purpose is to provide accurate angle degrees and give feedback on the positions of a machine and its components. As it was recently manufactured, it must undergo testing on each angle degree in order to evaluate its accuracy – our team is assisting in the automation to facilitate this process. Due to our team's background in computer science and engineering, this project can provide us with a better understanding of real processes within the manufacturing engineering industry. Since our college education focuses on the academic setting of engineering, we have little exposure to real life work experience. Furthermore, automation is an essential part of computer science and is gradually becoming more essential as human availability decreases while demands for it increase. Due to this, having more experience with automation would contribute tremendously to our career as engineers.

Currently, OEM Control is using a hand test method as a workflow to testing their AS5 products. They have a specific person who is responsible for the testing, and the process of the testing is long and painful. Basically, the tester needs to level the clinometer to flat first, and deploy the AS5 which is going to be tested on a specific cube—which is hollow and 3D printed. Tester needs to rotate the cube to make all the six sides of the cube up, and calibrate each time. This is called six side calibration in the whole process. After the calibration, the tester will start a long and boring part. Tester will fix the cube and AS5 on the clinometer, and need to start testing the difference with the actual angle and the angle detected by AS5 from -80 degrees to 80 degrees every 5 degrees. The process of this part will be: First, rotate the handle at right side of the clinometer to make the degree fit -80 degree; Second, type in the actual degree which is -80 in the software in the computer at right side and click the button of the software to

Figure 1.1: Testing Cube, As5 and clinometer during test

record this test, it will record the angle tester typed in, the angle returned by AS5, and difference between this two data; Third, rotate the handle to -75 degree and keep this process of test till finished all 32 times repeating work. And finally, output the recorded data in the software to a CSS file in the file system, and finish this test.

1.2 Technical Challenges

In exploring the issue of compatibility, we need to delve further into analyzing the limitations and challenges of current automated angle tests. Although we have developed clear methods to implement these tests, practicality is still a distant goal, especially considering the various problems that may be encountered, which may affect not only the efficiency of the tests, but also the reliability and accuracy of the whole

system. Software and hardware compatibility is a major obstacle. Our systems may contain multiple hardware components from different vendors, each with its own specific software requirements and configurations. In such a diverse environment, ensuring that all parts work together seamlessly can be a daunting task. For example, specific hardware may require specific versions of drivers or operating systems, which may not be compatible with our software architecture.

As the test system continues to evolve, new software updates and hardware upgrades may introduce new compatibility issues. This means that we need to continuously monitor and test various parts of the system to ensure that they continue to work properly after updates or upgrades. This not only adds to the workload, but also increases the time it takes to complete the project.

In addition, physical limitations of the hardware can be an issue. For example, some sensors may not be able to accurately measure the desired angular range. This requires us to find alternatives or adjust the test parameters to accommodate these limitations. The scalability of the test system is also a consideration. As a project grows, new features or components may need to be added. These new additions must be compatible with the existing system, or they may cause instability in the entire system. So we need to address them adequately before the final product is completed to ensure that the work can be done with high quality.

1.3 Accuracy Control

When discussing accuracy control, the first thing we face is the technical challenge. As mentioned before, although we have not actually entered a test scenario yet, there are a number of technical challenges that we are bound to encounter during the implementation of an automated test system. One of the most common problems is the compatibility between software and hardware. Whether a program is good or bad, its most basic requirement is to be able to run normally. If even basic operation cannot be realized, then all other discussions are meaningless. This will be the first challenge we need to overcome in the implementation process.

Next, discuss the issue of accuracy control. At the beginning of the project, we will perform manual calibration. Even if there are some errors in the process, they

usually don't pose too much of a problem as long as we can catch them and change them in time. However, if there are problems with accuracy later in the project, fixing them becomes a huge undertaking. Considering that we may not have a lot of time to calibrate and make adjustments later in the project, we need to make good preparations in the early stages of the project.

At the beginning of the project, we need to conduct comprehensive testing and calibration of all hardware devices and software systems to ensure that they maintain a high level of accuracy throughout the testing process. This means that we need to meticulously check and adjust each component to ensure that they can maintain stable performance under different environments and conditions. We also need to regularly check and maintain the system to prevent any loss of accuracy due to prolonged use. This includes regular software updates, checking and replacing hardware components that may be subject to wear and tear, and re-calibrating sensors.

We also need to monitor various parameters during testing in real time to be able to make adjustments as soon as any deviations are detected. This will greatly improve our efficiency in adjusting accuracy at a later stage of the project, ensuring the accuracy and reliability of the entire testing process. Accuracy control is an important aspect of our automated test systems, requiring us to invest a great deal of effort and resources in the early stages of a project to ensure that a high level of accuracy and efficiency is maintained throughout the project. Through adequate preparation at an early stage and timely adjustment at a later stage, we can effectively solve possible accuracy problems and ensure the stability and reliability of the test system

1.4 Time Efficiency

When discussing manual operations, we are faced with the problem of time inefficiency due to the need for manual pre-calibration by the team. The testers would begin a long and tedious period of testing. The entire testing process was also lengthy because of the lengthy process of recording the data and the need to manually adjust the angles and record the data. This meant that we needed to repeat the trial and error process in the early stages of the project. Each trial and error is very time-consuming, which definitely lengthens the test cycle time significantly.

And all of this tedious work needs to be done manually, which is certainly a big test of the team's energy. In order to accomplish these tasks, the team must invest a lot of human resources. The long hours of concentration and constant physical exertion are a challenge for testers. During such repetitive and time-consuming tasks, testers may feel tired, which not only affects their productivity but also the accuracy of the test results.

In addition, the manual process requires extreme precision in every data recording and angle adjustment. This not only increases the workload, but also raises the risk of errors. Consistency and repeatability of each test is also difficult to ensure due to human intervention, which is a big problem for tests where accurate and consistent results are desired.

In addressing this issue, we can consider introducing more automation techniques to reduce the reliance on human intervention. For example, the use of automated data recording systems and angle adjustment devices can greatly reduce the time and errors associated with manual input. At the same time, this can also reduce the workload of testers and improve the efficiency of the whole testing process.

1.5 Human Error

It is important to realize that there is a high probability of significant physical and attentional exertion by team members during multiple manual operations, which can lead to errors in test results. If these errors are due to mistakes made by individual members, it can have a significant impact on the overall work process. We may need to spend a significant amount of time identifying these errors and repeating the previous steps to ensure that the correct values are tested. Distractions or miscalculations are common in the early stages of repetitive testing, so data accuracy is one of the limitations we face.

In such a work environment, testers may become fatigued due to prolonged periods of concentration and physical exertion, and this fatigue not only affects their productivity, but may also impair their judgment and operational accuracy. When performing a large amount of repetitive work, even the slightest negligence may lead to deviations in test results. For example, when manually adjusting the angle of a test device, even a small angular deviation may result in a significant error in the final data.

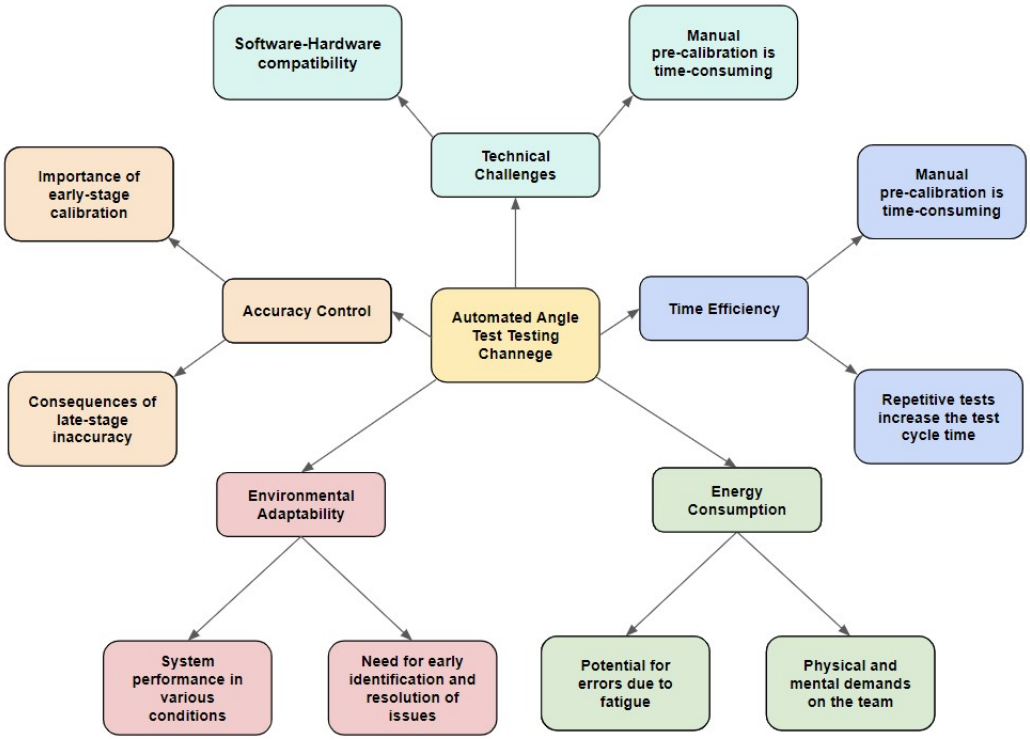
1.6 Environmental Adaptability

Although it may not be a problem we encounter often, it is important to consider the adaptability to different environmental factors (e.g., temperature, humidity, etc.) before finalizing a test system. The ability of an automation system to function and adapt to different environmental conditions is directly related to the reliability and usefulness of the system.

Different environmental conditions may have different effects on the components of the test system. For example, high or low temperatures may affect the operational efficiency of electronic equipment, and changes in humidity may affect the accuracy of sensors. Therefore, we need to conduct comprehensive environmental adaptability tests on the system to ensure that the system can still maintain stable and accurate performance.

As the application of the test system expands, it may need to be tested in different geographical environments and climatic conditions. This requires our hardware to be able to maintain efficient operation in different environments. For example, in high humidity or dusty environments, some parts of the system may need to be adjusted.

Even though we won't encounter it, there are strategies we can employ if it does happen. For example, improving the system's resistance to the environment by using more rugged materials or designing more flexible software algorithms to adapt to the effects of environmental changes on test results. At the same time, regular environmental adaptation testing and maintenance of the system is also an important measure to ensure the long-term stability of the system.



Our approach to automating the testing process of the AS5 is to have a robotic arm that'll perform all the previous tasks that were done by hand. We will be creating a Python desktop application that will control all parts of the automated testing involving the robotic arm through an API. Each stage of testing will be performed by their respective buttons. It begins with the AS5 being handed to the robotic arm, where the robotic arm will place the AS5 into the cube fixture to then do the six-side calibration. With the fixture at hand, the arm will rotate its clamp to a certain degree and test whether the AS5 is returning the same angle. The arm will iterate through multiple angles while logging the results which will be accessible through a CSV file that will be saved within the OEM's file system.

As technology advances and new devices are produced, the amount of repetitive tasks increases substantially – product testing, as in our project's case, can prove to be a time-inefficient and costly process. Due to this, there becomes a greater need to adopt automation practices. Our team's project proposes an efficient method to automate the monotonous task that a full-time engineer would be required to do, saving companies hundreds of dollars and time. In creating an efficient and faster testing process, we can aid other companies who have encountered a similar problem as well.

CHAPTER 2

Background

2.1 Related Works

Another group automated a torque angle sensor testing using SENT and LabVIEW. The SENT protocol is used in automotive applications for high-resolution sensor data transmission and LabVIEW is a graphical programming environment used to integrate programs often used in automation. The problem at hand was to use the SENT protocol and command a motor to rotate and then read an angle sensor. In order to automate the testing process, the group had to modify an existing program to adapt to multiple torque angle sensors and read the angular positions of the encoder and angle of the torque sensor while utilizing the SENT protocol. They also developed a LabVIEW program to act as a user interface for the test – this allowed the user to test parameters, communicate with the system, and graph/ store data to respective files (https://www.autosofttech.net/portfolio/automotive/118-sent-tester)

THINGS WE NEED TO ADD

- how is this group's process flawed and how is ours better?

2.2 Uses of Angle Sensor

THINGS WE NEED TO ADD

- what is the angle sensor used for
- what products does the angle sensor test

2.3 Robotic Arm Uses

THINGS WE NEED TO ADD

- background on robotic arms
- what are robotic arms usually used for

CHAPTER 3

Methodology

3.1 Current Process Progress

Our team was given the task of the Automating the Angle Test process for the OEM controls AS5 sensor. However, the original documentation detailing the specifics of the project left very little for us to work on. It gave a very brief description about the AS5 sensor: The sensor needed to provide accurate angle degrees to provide feedback on the position of a machine or components on the machine.We gathered from the document that the sensor needed to be tested, and that their current process was very tedious. We did not know what the sensor measured specifically, we did not know how it performed its measurements, and we knew nothing about the process of testing for automation. The document also gave very vague preferred technologies and clues to aid our team in solving it. It mentioned interfacing using the python programming language and it mentioned the LabVIEW software. At this point we understood that we needed to automate the testing process, but we had no knowledge of how the process was like, or what we needed to do to automate it. So our first step was to fully understand the problem given to us.

But before we set out on solving the project, we got familiar with all the members of our team to assess what skills and experiences we had. We had 5 computer science majors and 2 computer science and engineering majors. Our concentrations varied from Computational Data Analytics, to Algorithms and Theory to Software Development. We then set a regular meeting time, met with our advisor, and elected a team leader. We learned later on that our CSE team would work on this project in conjunction with an ECE team consisting of a Computer Engineering major and an Electrical Engineering major. So, we also got familiar with the skills and experience level of the other team and created regular meeting times for discussions and collaboration.

Once we solidified a schedule and familiarized ourselves with each other, we then took the next step towards solving the project – gathering more information about the

problem we needed to solve. In all our weekly team meetings, we discussed the problem presented to us and highlighted all the information that we had and those that needed to be obtained.

We knew the following:

- The angle sensor testing process was tedious and we would need to automate it.
- The ECE team mentioned that we would potentially be using a robot arm to automate the process. So we knew the solution involved automating a robot arm.
- And lastly, we were potentially going to use the LabVIEW software to interface with the angle sensor.

We needed to know the following:

- What was LabVIEW?
- What programming language did the project require?
- What type of arm would we be using?
- How does the manual testing process work?
- How is the testing process measured?
- How does the AS sensor work?
- What products do they test?

We had a lot of questions that only the sponsors could answer, so our next step was meeting with the sponsor of the project, OEM controls, to inquire about and clarify the project specifications. It was during this meeting that we got a lot more information on our project. We learnt what company they were and the work they did. They explained that they were manufacturers of rugged joysticks and electronic control modules for off highway machinery. They also explained the functionality and relevant details of the AS5 Sensor:

- It is a device that reads and reports the angle of a machine.
- It has 3 axes of accelerometer and 3 axes of gyroscope,
- It needs to be calibrated on 6 sides before usage.
- It uses a trusted angle table to determine its accuracy.
- It uses the software CanBUS and CanOPEN in its data communication.

They elaborated on why the testing process was tedious– one person had to do all of this(manually testing at 5 degree intervals). They proposed that a robot arm would expedite the process, and they needed us to automate this process with the arm.

We also got some information about the software LabVIEW. It is a diagnostics program good at checking inputs and outputs, and able to return pass/fail depending on the conditions met. They thought it would be a good option for our project.

We also discussed the possibility of moving resources to UConn, where we could more comfortably construct and test the solution. It would involve having a duplicate of the current test unit and all its parts, i.e the sensors and angle table.

And lastly, they insisted that some of our team visit them on site to gather more vital information that was needed for the project.

Following the meeting, some of our team members were not able to make the physical visit to the OEM controls site, but the team members that were able to attend were given a demonstration of the current tedious testing process, got more of our questions answered, and gathered pictures and information, which they relayed to the rest of the team. By the end of this visit and because we now understood the full scope of the problem, we had a concrete solution to solving the problem.

3.2 Project Design

3.2.1 Overview

We were in agreement that the project(the process of the automation) would consist of creating one software application that starts the testing process by calibrating the angle sensor, and tests it with the robot arm, then outputs all the results in a csv file saved on one of their computers. We also concluded that LabVIEW was going to be impractical for use. It cost upwards of about $6000 per desktop, a very costly decision considering we had 9 members on the team.

To be able to successfully automate the test, we first gained an understanding of the use cases of our application which is very simple. An employee will want to press a button that then runs the test. Then, after it is done, they will want to be able to

see the results and save them if desired. To be able to achieve this, we decided that a desktop application would be the best approach because the user interface allows any employee to control the execution of the automated test. Now that we had an approach to controlling the test, we had to create a method of executing the test.

Our solution to automating the test process goes as follows. We will build a desktop application using Python that controls a robot arm which will test the angle measurements of the test sensor against a reference sensor. After the test process has finished executing, the application will display the results to the user and allow them to save the results in the form of a CSV in a defined location in OEM's file system. (See figure 3-1).

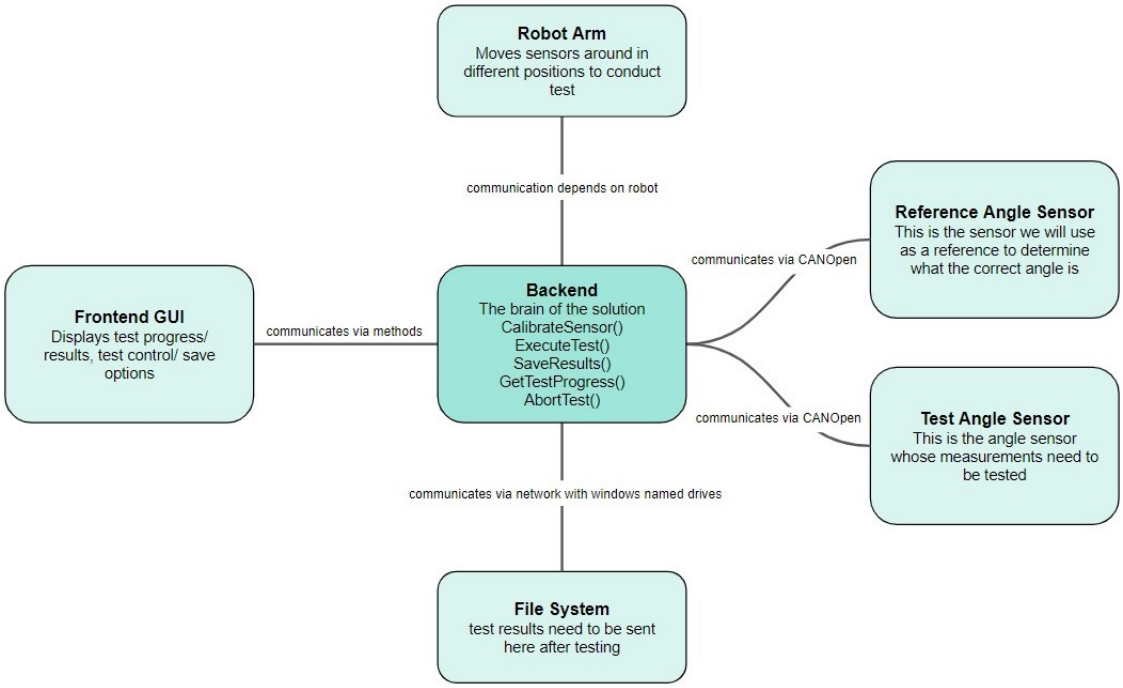


Figure 3-1. Project Overview Diagram.

This diagram gives an overview of the different components of the solution and how they communicate with each other. To successfully build the desktop application, we decided to split the development of the desktop application into different teams.

3.3 Delegation of Tasks

We are working with an ECE team that is responsible for the hardware aspect of the project. They are responsible for setting up the robot and converting the messages sent from the angle sensors from binary to nonbinary data that the backend team can use.

So at this juncture we understood the problem and had a possible solution, so what remained was splitting up the group to implement the solution. The ECE team would handle the hardware aspect of the robot. They would deal with the powering of the robot and the sensors, any possible mounts, and the hardware's interaction with the CANbus and CANopen software. The CSE team on the other hand would write the robot's program for the automated testing, and create a GUI(Graphical User Interface), as well as the corresponding testing programs to govern the entire testing process.

We split our CSE team into two– a backend team and a frontend team. With three members in the frontend and four members in the backend, our team was split according to experience, skill level, and interest in the tasks.

3.3.1 Frontend

The frontend application is a graphical user interface that allows an employee to initialize the automated test. It will display the test progress as the automated test executes and will display the results after the test has finished. Then, it will allow the user to save the test results in a designated location in OEM's file system. The frontend process involves using Figma to brainstorm a design and then will be built using the Tkinter library in Python and this application will communicate with the backend program through an API that we will design following object-oriented programming principles.

3.3.2 Backend

The backend team's primary responsibility is developing the test automation software, which involves succesfully controlling the robot arm and establishing communication with both the reference angle sensor and the test angle. These sensors com-

municate via CANOpen, thus the team will need to work on learning more about this communication protocol nd research on how to convert the byte messages into angles. The method of communication with the robot is dependent upon the specific robot we acquire, however, most robotic arms have an API that we can use to facilitate the actual control operations of the robot. Since we recently finalized on purchasing the UFACTORY xArm 6, we know it communicates via a Webocket API using a Standard Modbus TCP Protocol. Essentially, an API involves opening and closing connections between the receiver and sender while communicating. A Websocket API keeps one TCP connection open and uses that same one throughout its lifespan – this leads to real-time communication, efficiency, and reuced overhead, which are all important when communicating between software and hardware. A TCP break information into chunks, allowing them to send bits through different routes and make the sending of information more efficient. In order to successfully deliver our task, we must learn how to control the robotic arm with the API while adding failsafes to abort the test given any external factors (such as human interference) affect the execution of its task. Finally, the backend will also save the test results in a defined location in the OEM file system. Knowing this, choosing a robot was a key step in steering the direction of the implementation, so the Backend team worked closely with the ECE team to determine the optimal robotic arm. Now that we have determined that the UFACTORY xArm 6 is optimal for performing the task at hand, we will conduct more research on how to effectively communicate with OEM's file system in order to save test files.

Since the backend application functions as the brain of the general application, there are many methods we need to implement that could successfully communicate with the angle sensors, robotic arm, and the frontend to send test results. When structuring a script with these methods we must consider the need for synchronization of the angle sensors to ensure that the angles we are testing are being measured at the same time. Currently, a method to do this has not been achieved, but it will come to light after experimenting with the angle sensors and learning how they work and send messages. In order to achieve our task, we must take a deep dive into the API documentation and, fortunately, the XArm 6 has GitHub documentation of the robot's API which facilitates control over the robot arm. This API will be used to control the robot arm through the testing and calibrating subroutines to move the angle sensors into different positions.

Methods

The backend application has a lot of methods that need to be implemented to successfully automate the test. This section gives an overview of the functionality of each method.

CalibrateSensor() Before executing the test, each angle sensor has to be calibrated. This involves rotating the robot arm into 6 different positions and sending a message to the angle sensor to calibrate. We will be calibrating both angle sensors at the same time because they should give identical readings based on how they are oriented in the fixture (oriented in parallel).

ExecuteTest() Executes the angle sensor test. This rotates the arm from -80 to 80 degrees in increments of 5 degrees. This program takes in a parameter that determines how many axes to test. Once it completes one axis, it will rotate the arm to test the next axis and continue until the program terminates. Once complete, it returns the test results in the form of an object.

SaveResults() Saves the test results in a designated spot in OEM's file system. This step requires the computer running the application to have access to the file system, so permission exeptions must be handled here.

GetTestProgress() Gets the current axis and angle tested data from a test process that is currently executing. This is used by the frontend to display the progress of the test. In addition to this, it will also return a double that can be used to fill a progress bar on the GUI with the current test's execution status.

AbortTest() Aborts the test immediately and returns the robot to the neutral position if it is safe to do so.

These methods give an overview of the API that we will build to allow the frontend team to interact with the backend program. These methods represent the core functionality that the backend program will have to achieve. This list of methods is by no means extensive and will likely change as implementation begins.

Currently, we plan to build the backend program in Python so that our application can be imported as a class to the frontend application. However, if we are not able to communicate with the angle sensors and robot natively in Python, we might need

to use a different language which will make communication between our frontend and backend programs more difficult.

3.4 Choosing a Robotic Arm

After several discussions with the ECE team, we decided the following were the qualities that we concluded that was necessary for the robot:

- It had to have high accuracy Motors
- It had to be able to carry the load of the board and fixture conveniently
- Had to have 3 Axis or more
- Had to rotate around “wrist”
- Secure mountability, stability
- Able to be programmed with script(preferably python it was the language all the CSE team was familiar with.)
- Should be a COBOT. (A Collaborative Robot, capable of sharing a workspace with humans.
- Cable connection
- Budget: how much money our sponsors were willing to allocate to towards the project

Having this key information, we researched and came up with some potential robots:

- **MyPalletizer 260 M5Stack**

“MyPalletizer 260 M5Stack - The Most Compact 4-Axis Robotic Arm (Dual Screen Version)."

- 4 Axis Arm
- Fully Wrapped - Safe to Use
- Arduino and Python Interactable
- High precision magnetic encoder servo motor
- Communicates over USB-C
- Controlled by M5 Stack Basic : Coded with MicroPython

- **MyCobot 280 M5Stack 2023**

"MyCobot 280 M5Stack 2023- 6 DOF Collaborative Robot (Dual Screen Version)."

  - 6 Axis Arm
  - Payload weight of about 250g
  - Communicates over USB/Type-C
  - Controlled by the ESP32, 240 MHz Dual-core 520 KB SRAM Micro-processor
  - Arduino and myBlockly Interactable

- **UFACTORY xArm 6**

"UFACTORY xArm6 Robotic Arm."

  - 6 Degrees of Freedom
  - More expensive option
  - Built-in harmonic reducer, brushless servo, and 17-bit multi-turn absolute encoder make xArm joints durable and precise enough to work on long-term precise applications.
  - xArm SDK includes Python, ROS, and C++. Fully open-source software platforms offer more flexibility for your integration.

- **Universal Robots UR5e**

"UR5e."

  - Python based API
  - Multi-Access
  - Communicate over wifi, MODbus
  - Payload weight of about 5kg/11lbs

After deliberating and assessing all of these robot's qualities, we decided that the UFACTORY xArm 6 robot was the most adequate for our task. Apart from the fact that it matched all the criteria, its API was the most well-documented on Github. However, ultimately the OEM Controls Team was the one who had the final say, so we chose to present them with three of the options above: the xArm 6, the m5Stack 280, and the UR5e.

		OEM CONTROLS ROBOTIC ARM COMPARISON		
		ROBOTIC ARMS		
		MyCobot 280	xArm 6	UR5e
	Axis #	6	6	6
T	Repeatability	+/- 0.5	+/- 0.1	+/- 0.03
R	Harmonic Reducer	No	Yes	No
A	Open-Source SDK	M5Stack	xArm SDK	Urscript
I	Arm Reach	350mm	700mm	850mm
T	Payload Size	250g	5kg	5kg
S	Power Consumption	60W	~200 W	570W
	Cost	$1,448.00	$8,400 + Grabber	$35,000-$40,000

figure 3.1: comparison of the physical qualities of the three robotic arms

In the end, the xArm 6 was chosen to be used for automation as it satisfied our budget requirement, had an adequate payload size to sustain the weight of the AS5 angle sensor, had a good repeatability for our purpose, and had accessible and clear API documentation.

Because we have yet to complete this entire process, our methodology is still ongoing. Right now we have decided upon a robot arm, but we have yet to receive it. However, we have started implementing the frontend part of the program, we are currently researching the languages to interface with the robot we will need to program the robot. This is the current stage that we are at, but given that we understand the problem and project given to us we planned out a potential timeline of our work. Our projected course of action:

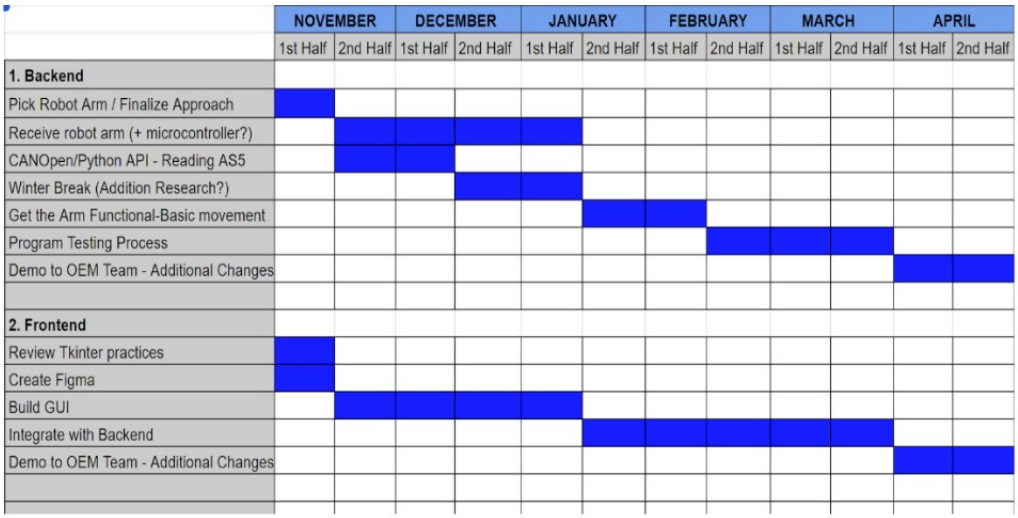


figure 3.2: our project's projected schedule throughout both semesters

CHAPTER 4

Preliminary Results

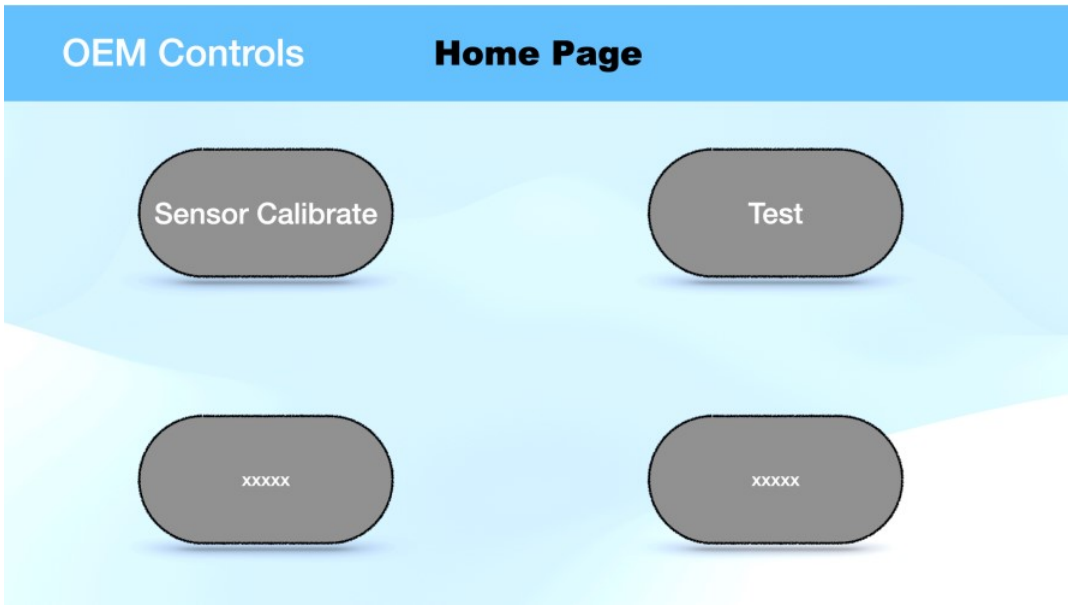


figure 4.1: the homepage allows the user to choose what action to perform

First, this screen is the main page of the software, which contains the "SENSOR CALIBRATE" and "TEST" buttons, which are designed to give the user the option of calibrating the sensor and starting the test. When the user clicks on "SENSOR CALIBRATE" it can be used to calibrate the sensor. In addition, there are two buttons labeled "XXXX" that can be used for other functional options, to make it easier to add new features to meet customer needs later in the development process. The initial result is that the user first clicks on "SENSOR CALIBRATE" to go to the Information page to calibrate the sensor. After calibration is complete, the user clicks on the "BACK" button to return to this page and then clicks on the "TEST" button to jump to the test screen for the next step.

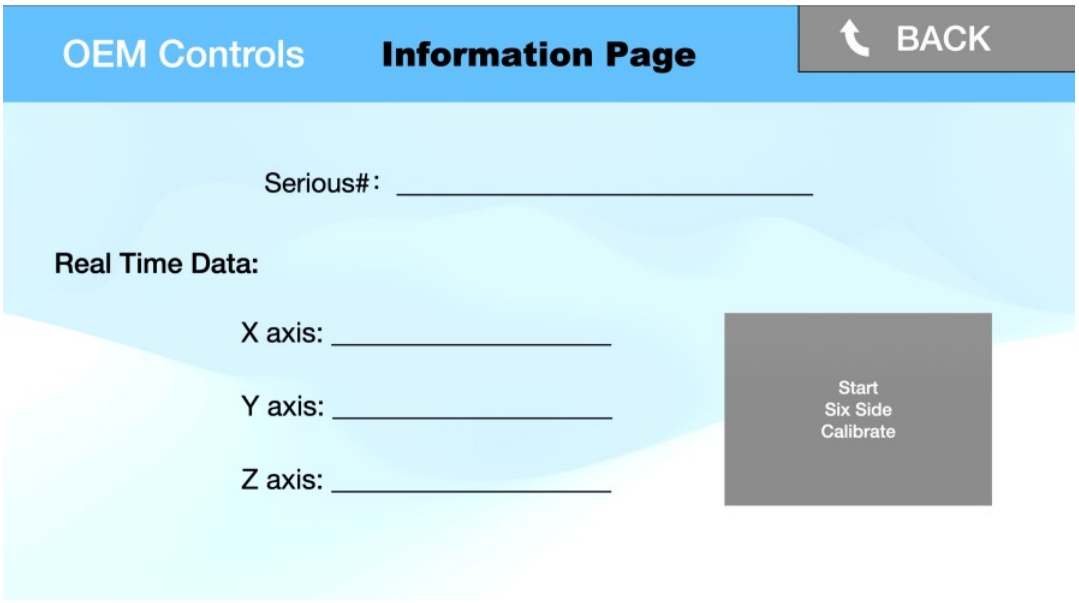


figure 4.2: the screen prompts the user for information on the calibration process

This screen displays the "Real time data" area, which contains the X-axis, Y-axis, and Z-axis is the real-time data coming from the three-axis sensors at the time of calibration. There is also a "Start Six Side Calibrate" button which is used to start the six-side calibration process, calibration is the approximate orientation of the six sides of the calibration. When the button is clicked during this calibration process, a fully automated calibration is initiated. Initial results are that on this screen, the user can enter or receive real-time data and initiate the calibration process with the click of a button.You can also see the "Serious" display area on the screen, which is intended to show the serial number of the Sensor. Each Sensor should have a different serial number to help the user differentiate between each test, and to help the user quickly find the corresponding Sensor and correct the problem after it occurs. The reason why this UI page is needed is that if the calibration work does not ensure the accuracy, then the later automation test will have to spend more time than the calibration work to correct the error if there is a problem. So this UI page is also indispensable.

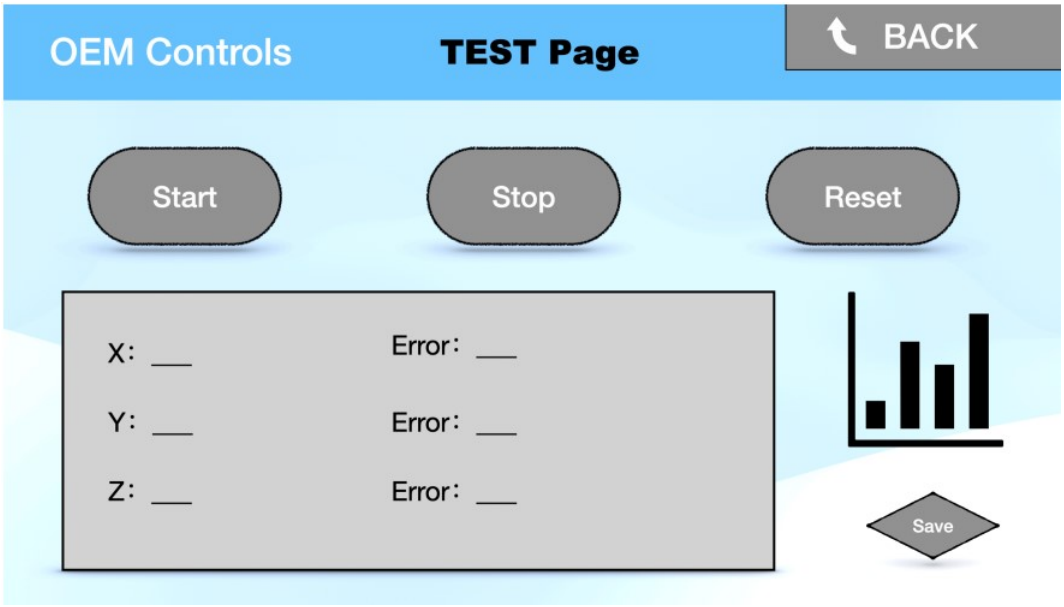


figure 4.3: screen where user can begin the testing process

This screen is the "Test" page and contains the "Start", "Stop" and "Reset" buttons, as well as the "Input" area for the X, Y, and Z axes. In the input area, the values to be measured are automatically entered into the program, and then the program returns each input and target value and error (a table is automatically generated at the end of the test) and displays the actual value, the input value, and the error to the screen. The "input" data corresponds to the X, Y, and Z axes respectively. "Save" button is located at the bottom of the page, when the user has completed the input and test can be carried out to save the data, to determine the saved data will automatically generate a table. Preliminary results, where the user can execute the test, enter data, return results, and choose to save the data after the test is completed. The "Back" button is for the user to return to the previous interface at any time.

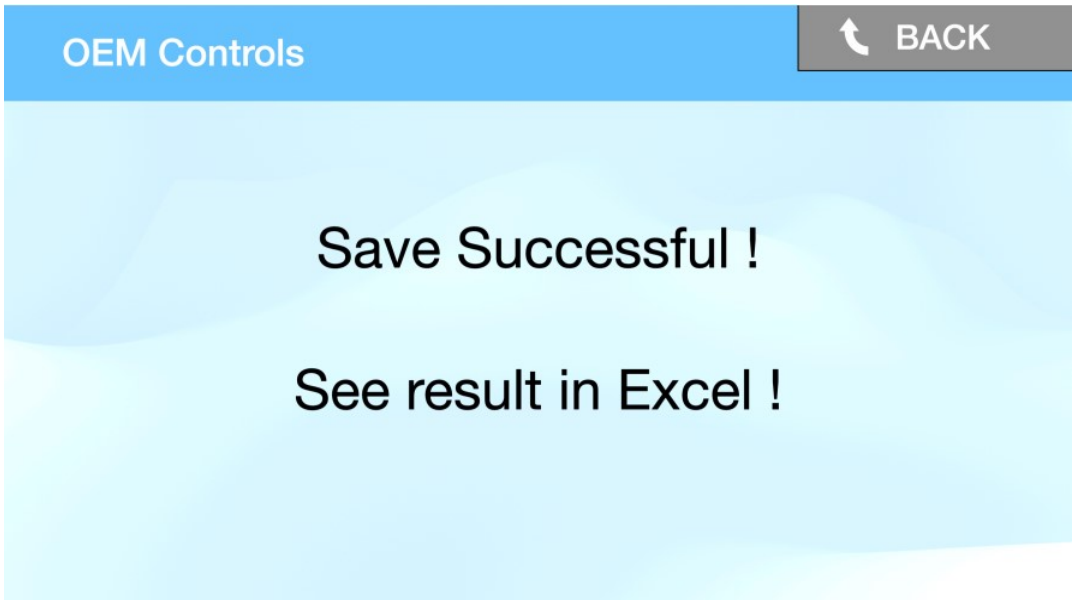


figure 4.4: save confirmation page

This page is the last page that will prompt the user with "Save Successful!" and "See result in Excel!". Preliminary Results This is the page where the user will get feedback on the success of the save once they have completed the test and confirmed that they have saved the data from the test, prompting the user they view the results of the test in the auto-generated Excel.I think the purpose of my addition to Excel is that it allows the user to see the results more clearly. Because if it is only displayed in the program it may make it more difficult to display and it will make the user waste a lot of time to find the data.

The UI has been designed to present the contents of the software to the customer intuitively and efficiently. The main interface provides a convenient entry point for calibrating sensors and performing tests by providing "SENSOR CALIBRATE" and "TEST" buttons. This design not only emphasizes the ease of operation for the user but also takes into account future scalability, such as the ability to add new functions to the "XXXX" button when more functions are required. On the "Information" page, the user can view the real-time data of the three-axis sensors and easily start the six-side calibration process with the "Start Six Side Calibrate" button. Here we want to automate the process to improve the efficiency and accuracy of the operation. On the "Test" page, the UI is integrated with the "Start", "Stop" and "Reset" buttons, as well as the X, Y, and Z axes. The UI provides the user with full control of the test by integrating 'Start', 'Stop', and 'Reset' buttons, as well as 'Input' fields for the X, Y, and Z axes, which are designed to test the accuracy of each small angle. The automatic data entry

and result feedback mechanism is designed not only to ensure the accuracy of the test but also to improve the efficiency of the data processing. Finally, the saved successful page provides clear feedback to the user, prompting them to view the test results in Excel, which allows the user to visualize the data more intuitively. To summarize, this software UI was designed with ease of use, automation, and feedback mechanisms in mind. It greatly improves the user experience and operation efficiency. Each page is tightly integrated with user needs and technical functions, making the whole software not only fully functional, but also will be very simple to use.

CHAPTER 5

Conclusion

5.1 Summary

In summary, our task is to automate an angle sensor testing process that is currently a tedious and time-consuming task performed by an engineer at OEM Controls. More specifically, we are automating a system to test OEM Controls' AS5 angle sensor, which is primarily used for joysticks and handles, which OEM Controls focuses on the most for their products, this includes markets such as aerial work platforms, agriculture, marine, and many others. The main issues encountered was the time inefficiency of the process, lack for accuracy control, and the proneness to human error, thus making our task to automate it a necessary one.

Initially, our task was vague and lacked a general structure to it, so we had a slower start process. Once our teams were able to physically visit the OEM Controls facility, we familiarized ourselves with their current angle sensor testing process and were able to design a structure to execute our automation task. Our design included creating a frontend application, a GUI built with the Tkinter python library, that allows the user to interact with the testing program and save the files to the OEM Controls system. It also consists of a backend program that communicates with a frontend GUI application using python methods, the robotic arm using its API, the AS5 and reference angle sensor using CANOpen, and the OEM Controls file system via network and drives.

After establishing a general structure to our automation task we began extensive research on finalizing a robotic arm to purchase. After mcuh research and discussion with OEM Controls, The UFACTORY xArm 6 proved to be an optimal choice for our task as it satisfied our budget requirements, had an appropriate payload, had an adequate repeatability, and was open-source. As this was relatively recent, we have yet to physically receive the arm, however we have begun the process of purchasing it. Now that this process of selecting an arm has been finalized, we can begin the programming and execution of the automation.

5.2 Future Work

Currently, our immediate future plans involve understanding how to build a GUI using the Tkinter library and exploring the xArm 6's documentation on GitHub. Due to it being open-source, we plan on researching its API documentation and better understand its functions. With this, we can create a script for programming the arm to perform the automation task, even before physically receiving the arm.

The general structure we will follow is as follows:

1. Calibrate arm
2. Loop for all six axis
   (a) Rotate grip to a -80 degree angle
   (b) In a loop until +80 degree angle:
      - Rotate arm plus 5 degrees
   (c) Switch to next axis

The task is simple – we essentially leverage the robotic arm to perform the angle testing while recording the results. As explained above, we must first calibrate the arm to be a precise as possible as our task involves ensuring an angle sensor is displaying accurate angles. Our procedure mirrors that of OEM's current testing process as they have an apparatus that holds the angle sensor and can rotate from -80 degrees to +80 degrees in different axis. The key benefits with our automation, as mentioned in our introduction, is the accuracy control and time efficiency, since we can program the robotic arm with repeatability of +0.01mm.

The Backend team's immediate future plans is to better understand the API Documentation and functions from GitHub, which will be followed by the development of a draft script to program the robotic arm to perform our automation task. Once physically receiving the arm, we will begin playing with the program and the robot functionalities.

Along with the ECE team, the backend team will also need to research on CANOpen and CANBus systems. Both operate using a physical communication system, which is a serial communication protocol used for connecting different electronic control units (ECUs).Essentially, it allows devices to communication with each other without the need for a host computer. These two systems will be essential in our task as they are necessary when communicating between backend and the angle sensors. Once we are

more familiar with this system we will be able to take an input angle from the sensors, read it, and translate it into information that our backend application can understand. Furthermore, this would allow us to synchronize the robot arm with the angle sensors' information as we want to ensure we are reading both sensors simultaneously and accurately.

As for the frontend team, we will be assigned to familiarize ourselves with the Tkinter python library. Essentially, Tkinter is a standard GUI library in Python used to create and develop applications with graphical interfaces. This will be a key step in developing our frontend application as Tkinter is the basis for the development of the GUI. Finalizing the frontent application would enable communication between the backend application and the user, allowing them to interact with the robotic arm to perform the automation process and save files to their system.

Lastly, the ECE team along with some of the CSE team will be tasked with 3D-printing a small structure that can hold both angle sensors simultaneously at the same level. This is crucial as our testing process relies on using a reliable angle sensor as a reference sensor to compare with the AS5, which we were assigned to test. This would allow us to accurately compare the angle provided by both sensors and deliver the most optimal results.

REFERENCES

[1] Mypalletizer 260 m5stack-the most compact 4-axis robotic arm (dual screen version). [Online]. Available: https://shop.elephantrobotics.com/products/mypalletizer

[2] Mycobot 280 m5stack 2023- 6 dof collaborative robot (dual screen version). [Online]. Available: https://shop.elephantrobotics.com/en-de/products/mycobot-worlds-smallest-and-lightest-six-axis-collaborative-robot

[3] Ufactory xarm6 robotic arm. [Online]. Available: https://top3dshop.com/product/ufactory-xarm-6-robotic-arm

[4] Ur5e. [Online]. Available: https://www.universal-robots.com/products/ur5-robot/

[5] Automotive torque angle sensor testing using sent and labview fpga. [Online]. Available: https://www.autosofttech.net/portfolio/automotive/118-sent-tester