NORTHERN ARIZONA UNIVERSITY

ASTRAEA

NAVY PRECISION OPTICAL INTERFEROMETER

Technology Feasibility Analysis

Team Members
Adam Schilperoort
Brandon Horner
Michael Partridge
Peter Kurtz
Trey Tangeman

Mentor
Isaac Shaffer

December 7, 2019

Contents

1 Introduction

Improving astrometry, or the measurement of position, motion, and magnitude of stars, is a never-ending goal of astronomers and astrophysicists around the world. By acquiring devices which can achieve higher precision astrometrics, scientists can improve upon previous models of stars and develop a more accurate representation of the cosmos. Breakthroughs in this area could lead to a better understanding of the universe, or potentially redefine the laws of physics altogether.

In the twentieth century, improvements in astrometry enabled physicists such as Albert Einstein and Stephen Hawking to make revolutionary contributions to the field of astrophysics. Following in these early innovator's footsteps, modern day astrophysicists are researching the properties of dark matter, modelling black holes, and searching for exoplanets. Improvements in observation technology could drastically accelerate research in these areas. Not only could this technology benefit scientists, but also the general public — for example, accurate star charts provide precise positioning for GPS satellites, which helps improve navigation around the planet.

From exploring foreign worlds to answering humanity's biggest questions, the impact of astrometry cannot be understated. Unfortunately, modern astrometry reached a point of diminishing returns, making it exponentially more expensive to improve on the existing technology. As a result, an alternative to the conventional telescope design needed to be developed. Navy Precision Optical Interferometer (NPOI) tackled this design problem by developing an optical telescope system. Their design is capable of providing the highest angular resolution on Earth at a significantly lower cost than equivalent conventional designs.

NPOI, an astronomical long-baseline optical interferometer has been in operation on Anderson Mesa, outside of Flagstaff, Arizona, since 1994. The 437m baseline array has a unique capacity for detecting and determining motions and orbits of binary systems, which is its current research focus. The site's unique precision is achieved through a high-tech mirror control system which simultaneously gathers light from three telescopes separated by some distance, and combines their respective images for analysis. NPOI's partners are Lowell Observatory, United States Naval Observatory (USNO), and Navy Research Laboratory (NRL), who each became involved with NPOI's development for separate reasons. NRL's goal was to develop a piece of technology able to measure the positioning of stars better than any other instrument on Earth to allow for precise navigation from stars, in the event GPS systems fail. USNO became involved to generate star charts for astronomy and astrophysics purposes, and Lowell joined due to the unique ability to study binary systems.

However, as the decades passed by, NPOI has been left with a system that has fundamental technological issues, which involve both hardware and software. These include the following:

- The hardware is out of date, virtually impossible to replace, and unreliable due to harsh environmental conditions.
- The hardware-to-software interface is overly complicated, poorly designed, and not well documented.

As a result, NPOI decided to start a capstone project which aims to solve some of their technological flaws. Currently, NPOI has two advisors working directly with the capstone project: Jim Clark, the current director of NPOI and an employee of NRL, and Teznie Pugh, the operations manager for Lowell Observatory. NPOI has commissioned the capstone team, Astraea, to develop a new technological solution that will simplify their existing system and provide a modern solution.

Specifically, NPOI's main issue involves the siderostats. These are mirrors responsible for tracking objects at the rate of Earth's rotation, and have a complex, dedicated computer system which handles many inputs and outputs. Those include stepper motors which turn the mirror, a camera system which reports positional offset from celestial objects, and optical limit switches which inform the mirror when it's reached its furthest range of motion. Additionally, a Narrow Angle Tracker (NAT) mirror controlled through piezoelectric actuators ensures light reflected down the vacuum tubes is centered, receiving positional offset feedback over an optical fiber network. At NPOI, there are 10 different siderostat stations each equipped with a computer rack which must handle input/output for both the siderostat and NAT. This design introduces additional problems to the system:

- Manually maintaining ten separate computer systems in remote locations is physically challenging and costly
- Remote locations expose hardware to more environmental hazards, making hardware more likely to break
- Troubleshooting and operating ten different machines is time-consuming

With these problems in mind, Astraea has developed a new and modern solution which involves developing a centralized computer to control each siderostat station. The centralized computer will communicate over the fiber-optic network at NPOI and issue signals to a motor controller, which controls the siderostat. Additionally, the new system will be able to handle the limit switches on the siderostat and feedback from the NAT using a microcontroller or microprocessor. In other words, Astraea plans to build a new centralized computer to send signals over the network to each siderostat station, replace the computer at each station with a microcontroller or microprocessor to handle the feedback loops, and control the stepper motors for each siderostat.

Using this centralized computer system, Astraea hopes to solve the problem of having ten separate computer systems in remote locations by reducing it down to one system. Additionally, the new computer will be relocated into a well-insulated laboratory which will significantly lower the chances of the machine failing due to environmental conditions. Lasty, because there is only one machine, the points-of-failure will be significantly reduced, allowing for easier troubleshooting.

2 Technology Challenges

This section will investigate the environmental requirements and the design decisions that have to be made in order to have a functional set of technology to create an end product. The environmental requirements details the set of requirements that constrain the design decisions. The design decisions briefly outline the technological challenges explored in this document.

Environmental Requirements
The environmental requirements for this project only relate to the inputs and outputs that the machinery and supporting hardware expect. Since much of the machinery in place is custom-built or specially made for NPOI, it is expected that the system can interact with the pieces already in place. The types of interactions expected are:

- Sending signals to stepper motors in order to control the siderostats.
- Assuring that when a physical limit is hit in the system that it is taken care of by the software.
- Handling two feedback loops in the system.

The team has decide that these are the design decisions that are critical to the development of a working and appropriate solution:

Interface Hardware
Because each piece of hardware has numerous inputs and outputs, the pieces that will be replaced will need to reproduce the necessary input/output in order to maintain proper system function. Additionally, signals from a central computer will need to be multiplexed.

Performance Hardware
This project will need a computer build with a CPU (Central Processing Unit) that can handle multithreading. The motherboard will have to be equipped with the required ports for everything that is needed by the system. It is possible that a prebuilt computer is the best solution for the client.

Development Environment
For the purposes of this project, the development environment will include: an operating system (OS), a programming language, a documentation tool, and a testing environment. Each component will be explored and discussed in detail, in the corresponding section.

Internal System Software
The architecture of the software is currently having each siderostat station have its own copy of the software. A potential alternative is centralizing the software to reduce redundant code. The decision needs to be made between keeping the software at each station or having a centralized system.

User-facing software
The user-facing software section will investigate user interface API software packages for graphical user interfaces. These user interfaces will be used to help display information about the mechanical systems and control software used at NPOI.

3 Technology Analysis

This section details the decisions made for each technical challenge. For each challenge, alternative solutions are compared, evaluation criteria are discussed, and the final decision is justified.

3.1 Hardware

Hardware encompasses all the physical components of a computer system, ranging from individual components which make up a PC, to interfaces and external devices. Within a standard PC, components include a central processing unit, motherboard, hard drive, random access memory (RAM), power supply, case, and cooling system. External devices include everything the computer is connected to, and these connections can be accomplished through a variety of interfaces. Device compatibility and system performance depends on the quality and capability of the hardware, similarly software capability and performance also depends on the hardware.

At NPOI the existing hardware is complex, outdated, and lacking sufficient documentation. As a result, designing a new system to interface with existing hardware is challenging. Input/Output feedback loops in the system requires interfacing with multiple stepper motors, an optical fiber network, USB devices, and Ethernet connections. Having the right hardware is important not only for interfacing with existing connections, but for being able to run the required software. The implemented hardware and software need to interact in such a way that they provide optimal performance when receiving, processing, and responding to input

feedback loops. Without adequate response times, the client's system will not be able to operate, so it is imperative to have hardware with sufficient speed to run the software solution. The sections below document and evaluate individual hardware components through concrete protocols for choosing the best hardware for our solution.

3.1.1 Interface Hardware

Interface hardware refers to the architecture that interconnects two devices. The current hardware uses PCI (Peripheral Component Interconnect) to fiber optic connections to communicate between the siderostat controller (SidCon) and the quad cell. Fiber optics are used for quick transmission of data over a large distance using light signals. This PCI to fiber connection can be replaced with USB to fiber. An alternative would be to keep the PCI slot to fiber that currently exists in the system. The stepper motors that move the siderostats and NATs should still be handled local to the siderostat stations. In addition, the limit switches for the siderostats and NATs could be handled by a simple microcontroller, microprocessor.

Fiber-Optic Networking
Optical fiber is a flexible, transparent fiber which is immune to electromagnetic interference and used to transmit data at high bandwidth. Optical fiber is currently NPOI's main method for communication across the site network; one of the feedback loops is only accessible through fiber. Thus, in building a new system, one of the most important aspects is how it interfaces with the fiber optic connection. Currently the site uses PCI at each station to connect with the optical fiber, but other solutions for communication are available. In this section optical fiber communication hardware will be evaluated to provide the fastest, most expandable networking solution.

To implement an interface, there are environmental factors relevant to hardware performance that must be considered. Site temperatures range from -20 °C during winter to 38 °C during summer, thus hardware must be able to perform in these conditions. Additionally, the budget is limited, so the lowest cost solution that works is ideal. Finally, data transfer rate must accommodate networking through an optical fiber network at a minimum of 0.3 MHz. The client has also expressed interest in having a long-term solution, and one that will be easy to fix in 20 years, so long-term availability of parts is key. This can be approximated through availability from other sellers and the existence of identical hardware alternatives.

To create the best solution for the client, evaluations will be made based on the following protocol:

- Temperature Tolerance: Range of Temperature Values, wider is better
- Cost: Monetary value, lower is better
- Data Transfer Rate: Measured in MB/s, higher is better
- Available From Other Sellers: Yes/No
- Identical Hardware Alternative: Yes/No

Fastcom FSCC/4-PCIe
This is the current fiber optic networking solution at NPOI. Fastcom cards are popular primarily due to their technical support and long-term availability, which separates their company from other PCI card manufacturers who regularly discontinue support for older products. These cards have proved to be reliable through years of temperature cycling at NPOI and continue to perform above the minimum data transfer limit. In addition to our client's personal experience with these cards, the manufacturer's website [1] details the following:

- Temperature Tolerance: 0 - 70°C
- Cost: $999
- Data Transfer Rate: 2.5 MB/s
- Available From Other Sellers: Yes
- Identical Hardware Alternative: No

Although temperature tolerance specs only go as low as 0 °C, site conditions drop as low at -20 °C. However, the cards currently installed at the site have survived years of this temperature cycling without failure, attesting to their reliability. In addition, the planned solution may be in a temperature-controlled location, making temperature tolerance less of an issue. For at least 10 years Fastcom has listed and updated this product on their webpage, and this product can be found from other sellers, so although future availability is unproven, it seems likely the company will continue support for this product. However, because this part is proprietary and non-standard, an identical hardware alternative cannot be found, thus making this solution less attractive.

Icron Ranger 2324 - 4 Port USB 2.0 Fiber Optic Extender
Because USB connections specialize in input/output and are standard on every computer build, it may be important to consider USB to fiber as a networking solution. A USB connection would remove the constraint created by using PCI, thus allowing the solution to be built on server racks or other computers which lack PCI ports. Plug-and-play capability also

drastically improves the usability of this device, saving astronomers time and effort downloading drivers. Additionally, one of NPOI's partners, Lowell Observatory, has experience using this particular interface. The manufacturer's website [2] details the Icon Ranger 2324 - 4 Port USB 2.0 Fiber Optic Extender:

- Operating Temperature: 0 to 50 °C
- Cost: $999
- Data Transfer Rate: 60 MB/s
- Available From Other Sellers: Yes
- Identical Hardware Alternative: Yes

Because optical fiber networks are becoming more common, and Icron is a fairly large company, this part should be available from the manufacturer for at least 20 years, if not longer. If no longer available through Icron, other USB to fiber converters, functionally identical to this device, would be readily available from other manufacturers and sellers. Although operating temperature does not meet the specifications for remote locations, the implemented solution will be inside a temperature controlled room. The most attractive aspect of this choice is the high data transfer rate.

The client has experience using both the Fastcom FSCC/4-PCIe and the Icron Ranger 2324 - 4 Port USB 2.0 Fiber Optic Extender, and a codebase already exists which uses either of these. This means both options represent an easier learning curve for implementing software than another alternative. Price for both systems is identical and temperature tolerances are similar. Thus, the main distinguishing factor between the Fastcom PCIe card and the Icron USB Fiber Optic Extender is data transfer rate and the long-term availability and usage (represented by availability from other sellers and identical hardware alternative). While both transfer rates are fast enough, the USB system would provide a large margin for inefficiencies or expanded functionality. Additionally, USB would remove an unnecessary constraint added by choosing a PCI-based system, allowing the solution to be compatible with a much larger selection of motherboards, thus our recommendation is the Icron Ranger 2324 - 4 Port USB 2.0 Fiber Optic Extender.

Stepper Motor and Limit Switch

Each station will require a microcontroller or microcomputer interface which interprets commands from the central brain, sends signals to a motor controller, and receives feedback from limit switches. The current solution is a custom motherboard with embedded code, making it virtually impossible to replace. Thus, to create a replaceable or better maintainable system, some alternative solutions should be explored.

While stepper motors and limit switches are standardized, motor controllers vary drastically in their capabilities. Because the motor controllers will be expected to operate the siderostat

flawlessly, choosing the correct device is paramount. Due to the large range of options, however, the only devices which will be considered are ones the client has expressed interest in. Each device will be evaluated on the following:

- Cost
- Wiki Availability
- External Libraries
- Operating Speed
- Output Resolution
- Active Damping

	Raspberry Pi 3	Arduino Uno w/ Ethernet Shield	Zeta Microstepping Drive
Cost/Unit	$35	$45	$600
Wiki Availability	Yes	Yes	No
External Libraries	Yes	Yes	No
Output Speed	700 MHz	16 MHz	2 MHz
Output Resolution	4096	4096	50800
Active Damping	No	No	Yes

Table 1: Motor controller Comparison [3][4][5][6]

While the Zeta Microstepping costs more, has no wiki availability or external libraries, has lower output speed than either the Raspberry Pi or Arduino, the active damping and superior output resolution translate to a more precise system. However, the cost associated with the device (upwards of $600) and its complexity makes it unattractive. Each station will require an interfacing solution, thus the fully-implemented solution will require 10 systems, costing $6,000 for the Zeta solution while only costing $350-$450 for an equivalent Raspberry Pi or Arduino Solution. Thus, because the Raspberry Pi's output speed is much higher than the Arduino and is slightly less expensive, the recommendation is for the Raspberry Pi.

3.1.2 Conclusion

In conclusion, the best choice for the fiber-optic interface is the Icron Ranger 2324 because it achieves a higher data transfer rate than the Fastcom FSCC, will be easier for the client to use, and won't limit the choice of motherboard. The motor controller recommendation is the Raspberry Pi, primarily due to its low cost and higher output rate. It should be noted that the Zeta will drive the stepper motors more precisely, so if the Raspberry Pi is unable to achieve the desired angular resolution during testing, the Zeta solution should then be explored.

3.1.3 Performance Hardware

The hardware that is most important to the centralized computer is the CPU and the motherboard. A motherboard The CPU should be able to handle multiple threads, but more cores come at a cost. The motherboard should be able to handle all of the inputs and outputs to multiple siderostat stations. There would be benefits to a motherboard that could accommodate future solutions as well as the current technology for testing purposes.

Processor
There are many CPUs that could be chosen for this project. Since a multi-threaded system is desired, ideally they would have one core dedicated to each siderostat station. The client has expressed that the design should be able to handle future upgrades to the site, involving 10 total siderostat stations. At a minimum the CPU should handle 10 threads, for this would perform marginally better than a dual or quad core processor. In the previous section, it was determined that the USB to fiber connection was the most optimal for our system, however, that device is only compatible with Intel chipsets, therefore only Intel processors will be considered. Intel i5 processors will not be considered because they lack hyperthreading. Each CPU will be compared based on the following:

- Number of cores
- Clock Speed
- Cost

The Intel Core i9-9820X Skylake X LGA 2066 165W BX80673I99820X would be optimal for operations of the system, being able to run 10 processes in parallel [7].

- Number of cores: 10 cores
- Clock Speed: 3.3 GHz
- Cost: $550

The Intel Core i7-6800K Broadwell-E LGA 2011-v3 would be a good choice of CPU for operations of the system [8].

- Number of cores: 6 cores
- Clock Speed: 3.4 GHz
- Cost: $ 332

The i9 processor could be overkill for this project. It is possible that this system would not benefit significantly from having 10 cores, and there are hidden costs in cooling the processor that will need to be explored further. Instead of 10 cores, the i7 processor could handle 12 threads over 6 cores. Although it has less cores than the i9 processor, each core has hyper-threading so handling up to 12 threads will still be very fast.

Motherboard

The motherboards listed here are examples that have a compatible socket for a CPU listed above. Each one will be at a minimum compatible with the system that needs to connect to it. The motherboards will be compared based on a variety of hardware slots available. While USB-C options are nice for future-proofing the computer, they will not be considered required. Ethernet slots are required to connect to the current network. Two USB slots are for the USB to fiber connection and at least a keyboard for operation are required. Specifications that will be looked at include:

- Number of USB 2.0 and 3.0 slots
- Number of USB 3.1 slots
- Number of USB-C slots
- Number of PCI-e Slots
- Number of Ethernet Slots
- Cost

The EVGA X299 Dark 151-SX-E299-KR Intel Motherboard has an LGA 2066 socket to pair with the Intel Core i9 processor [9]. This socket is supported by the Intel X299 chipset so a similar board with this chipset would suffice.

- Number of USB 2.0 and 3.0 slots: 4 x USB 2.0 and 2 x USB 3.0
- Number of USB 3.1 Slots: 1 x USB 3.1 [Gen 2 Type A], 6 x USB 3.1 [Gen 1]
- Number of USB-C slots: 1
- Number of PCI-e Slots: 5 x PCI-e x16, 1 x PCI-e x4
- Number of Ethernet Slots: 2
- Cost: $470

The MSI X99A Gaming Pro Carbon ATX Intel Motherboard has an LGA 2011-v3 socket to pair with the Intel Core i7 processor [10]. This socket is supported by the Intel X99 chipset so a similar board with this chipset would suffice.

- Number of USB 2.0 and 3.0 slots: 2 x USB 2.0
- Number of USB 3.1 Slots: 4 x USB 3.1 [Gen1], 1 x
- USB 3.1 [Gen2]
- Number of USB-C slots: 1
- Number of PCI-e Slots: 4 x PCI-e x16, 1 x PCI-e x4, 1 x PCI-e x1
- Number of Ethernet Slots: 1
- Cost: $322

3.1.4 Conclusion

	MSI motherboard & Intel i7	EVGA motherboard & Intel i9
Thread handling	Yes	Yes
Motherboard slots	Yes	Yes
Cost	$654	$1076

Table 2: CPU and Motherboard Comparison [7][8][9][10]

Since the Intel i9 processor will be likely be overkill, the MSI motherboard will suffice for the system. The combination of the MSI motherboard with the i7 processor, will be considerably cheaper than the EVGA motherboard combined with the i9 processor with a savings of $422 (see table 2). The MSI motherboard has a variety of USB connections as well as PCI slots allowing for implementation of a variety of solutions. In the condition that more USB connections are needed for the motherboard, they can be added into the available PCI slots on the MSI motherboard.

The design decision to go with a 6-core CPU with a motherboard to match comes with a high confidence level. The cost of going with more than a 6-core is too high. To justify buying a CPU with more cores, the system would need to be using more than 12 threads, which according to the client, is not going to be required in the near future. The motherboard must have some USB slots and an Ethernet cable, which are fairly common for motherboards to have. Although the decision was made to go with a cheaper build, there will not be any bottlenecks to the system.

As requested by the client, an off-the-shelf build that meets the required specifications is the preferred option. In terms of ordering new parts and assembling them, there is a lot of overhead with a custom built desktop. The client has expressed that the cost of operating the interferometer is very expensive and if there are many parts to order, it could take many days to collect and assemble them all. Buying one complete computer build would be easier and effective in the case of buying a replacement computer.

3.2 Development Environment

A development environment (DE) is constituted by the tools used for program development. Development of working software requires tools for writing, documenting, executing, and debugging that software. The language chosen to write the software is in itself a tool. Deeper still, the operating system on which all of these tools run is a tool of a development environment.

The tools used to get the job done can determine the quality of the final product, and timeline in which it was completed. Picking the right tools for the job will multiply the effort put forth. Picking the wrong tools will detract from the effort put forth. This section will detail the tools that will be chosen, what makes them the right tools, and how they will be tested.

3.2.1 Operating System

The operating system can be thought of as the platform on which all the tools run. It is the bridge between the computer hardware and the software behind the tools used.

GNU/Linux, often just Linux, is an open source operating system built from a combination of tools from the GNU and Linux projects. Being that it is open source, there is no associated licensing fee or monetary cost to use the system. There are many different distributions (distros) of GNU/Linux to choose from. A distribution is essentially a bundle of packages chosen to form the system. Different distributions have different packages, sizes, user-bases, release schedules, and intended use-cases. For this project, the chosen Linux distro must:

- Provide basic functionality out of the box.
- Provide simple means of updating the system.
- Remain stable for up to a year at a time between updates.
- Be backed by good documentation.
- Have a large user-base.

	Debian	Arch Linux	Fedora
Basic Functionality	✓	✗	✓
Simple Updating	✓	✓	✓
Stable for 1 Year	✓	✗	✗
Documentation	✓	✓	✓
Large user-base	✓	✓	✓
Total(✓)	5	3	4

Table 3: GNU/Linux Distribution Comparison [11][12][13][14]

Debian is the best fit operating system for this project. When compared to the other top distribution options, Arch Linux and Fedora, Debian scores highest in terms of out-of-the-box functionality, simple updating process, stability, documentation, and size of user-base size (see table 3). Debian provides all the tools needed to use the system upon installation and provides simple commands to keep the system up to date and maintained. Its stability is an attribute of its release schedule.

The Debian release schedule is dependent on its testing process. The overall process includes three stages: unstable, testing, and stable in that order. New versions of Debian (combinations of updated packages) enter two separate testing stages: unstable and testing. New and updated packages enter the unstable stage. Here they must prove to be "bug-free" for a number of days in order to graduate to the testing stage. At each stage there are real people using and testing the packages, which means bugs are found and fixed quickly. Once

packages in testing are deemed stable enough by those people, the testing stage is frozen, meaning it closes its doors to new, updated packages and is tested as is until it is ready to be called the next stable release. This results in a robust system whose packages have been thoroughly tested and proven to work together for many people, on many different sets of hardware, without updating the packages. This translates to a solid offline system, that can go a year without the need to update packages for bug-fixes.

Debian's large user-base means that most common problems have been solved and their solutions are readily available online; so while it does not have an extensive documentation resource like the Arch Linux wiki, solutions are well documented across the web. Having a large user-base means that Debian is tested on many machines and different hardware combinations so it can almost be guaranteed someone has got it to work on the system (and has probably documented how). Being that Debian is one of the most widely-used Linux distributions, when 3rd party software vendors make a Linux version of their software, it is often only for Debian.

3.2.2 Language

Each programming language has certain tasks that they are best suited for - use-cases they were designed for. Abstracting each local brain into one centralized brain or command center requires that the language be able to:

- Interact with computer hardware at the low-level easily.
- Work on Linux out-of-the-box.
- Send commands to the Linux command-line easily for external hardware input/output.
- Be compatible with 3rd party hardware controller libraries.

	C/C++	Java
Easy Low-level Control	✓	✗
Out-of-the-box Linux Support	✓	✗
Easy Command-Line Interaction	✓	✗
Library Compatibility	✓	✓
Total(✓)	4	3

Table 4: Language Comparison [15]

C/C++ is the best language choice for this project, scoring higher than Java in terms of low-level control, out-of-the-box Linux support, interaction with the Linux command line, and compatibility with 3rd party, open source libraries (see table 4).

C/C++ give low-level control by putting the programmer in control of memory management. This means the programmer can allocate and clear memory as needed within the program, and even pass in memory addresses as parameters. In Java, the programmer is limited to pass by value and subject to the discretion of Java's automatic garbage collection implementation. C/C++ provides access to low-level procedures such as unsigned arithmetic, which Java covers up. This is a useful toolkit to have considering manipulating bitwise hardware input/output is a real possibility for this project.

C/C++ compiles and runs out-of-the-box on Linux. Debian comes pre-installed with most needed C/C++ tools, meaning one can compile and run C/C++ source code on a fresh install without installing additional packages. This is because most of the Debian operating system and its component tools are written in C. For Java compilation and execution, additional tools must be installed. Choosing a language that works out-of-the-box becomes more important when considering that the system will be offline for most of its life, making the need for installation of additional tools a restriction. Being that downtime is expensive, the language should be fully supported by the chosen system (Debian). C/C++ provides means to talk to the Debian command line which is useful for sending and receiving output through the systems ports. While it can be done in Java, it is a more streamlined process in C/C++.

A large amount of open source libraries that may prove useful for this project are written in C/C++ including stepper motor control and Arduino and Raspberry Pi libraries. This allows for the library code to be integrated into project without having to translate it first. The existing source code that runs the current system at NPOI is written in C/C++ so software migration and recycling will be an easier process.

As a bonus, C/C++ runs as native machine code, where Java runs on a Java Virtual Machine (JVM). This translates to: C runs faster and lighter than Java, using less hardware resources. While this improvement in weight and speed is sometimes insignificant, its benefits will likely show considering the base level Raspberry Pi has only 1 GB of RAM.

3.2.3 Documentation

Documentation without code tells the reader what the code should do, but has no way of doing it. Code without documentation may do what it is supposed to, but provide no explanation or context behind what it is doing and why. Good program documentation is as important as good code for this reason: all software must be maintained. If the person who is maintaining the code does not know what the code is supposed to do, or why a piece of the code is where it is, they may change it for lack of context, and break functionality. Having good documentation in place gives the reader answers to what, why, and how, and provides

foundation for reasoning for or against changing what is in place. In addition to maintaining functionality, another important connection to documentation is time. By leaving undocumented code behind, one is wasting the time of the person whose job it is to maintain it and the money of the company who hired them.

Having documentation throughout the source code is essential, but even more convenient is having a separate document to reference and share. For C/C++ there are tools that generate such documentation based on the comments in the code, automatically creating an HTML document that can be viewed by and shared with anyone. The most popular documentation generators are Doxygen and NaturalDocs. Doxygen and NaturalDocs are both documentation generators, but they differ in terms of:

- Easy installation.
- Easy configuration.
- Full C/C++ language support.
- Simple syntax.
- Multiple output formats.

	Doxygen	NaturalDocs
Easy Installation	✓	✗
Easy Configuration	✓	✗
Full C/C++ Support	✓	✗
Simple Syntax	✓	✗
Multiple Output Formats	✓	✗
Total(✓)	5	0

Table 5: Documentation Generator Comparison [16][17]

Doxygen is the best fit documentation generator for this project. When compared to NaturalDocs, Doxygen scores highest in terms of installation, configuration, C/C++ support, simple syntax, and supported output formats (see table 5).

Doxygen is easy to install, and is available for Linux, Windows, and Mac in both binary and source formats. Doxygen creates documentation based on the rules set in the configuration file. The configuration file can be automatically generated and reused on multiple projects; a

much more elegant solution when compared to NaturalDocs requiring command line configuration for each project. According to Doxygen's website it is "...the de facto standard tool for generating documentation from annotated C++ sources". In other words it was designed for use with C/C++, whereas NaturalDocs only offers partial support for C/C++. When it comes to syntax, Doxygen supports and interprets the Javadoc comment format (/** */), which is much less verbose than the NaturalDocs comment format. Javadoc formatted comments happen to be used in some of the existing source code for this project, meaning existing comments can be used as is, with little to no reformatting. Doxygen supports multiple document output formats including HTML, LaTex, and man page, whereas NaturalDocs can only produce HTML output.

3.2.4 Unit Testing

Testing code is an essential part of programming. Implementing insufficiently tested code poses the risk of system failure, hardware damage, and in critical applications, the loss of life. Testing code is such a crucial step in the programming process, that some programmers write the tests before they write the code. Testing code manually with print statements might work, but often makes the code very messy, complicates output, and is usually removed after use. By using a unit testing framework, the programmer ensures that the output and existing code remain organized and readable, and that the test code remains ready for use at all stages of development. The two prominent unit testing frameworks available for use with C/C++ source code are Google Test and CppUnit. When comparing the two these looked for:

- Easy installation.
- Short learning curve.
- Simple syntax.
- Time saving features.

	Google Test	CppUnit
Easy Installation	✗	✗
Short Learning Curve	✗	✗
Simple Syntax	✗	✗
Time Saving Features	✓	✗
Total(✓)	1	0

Table 6: Unit Test Framework Comparison [18][19]

It has been decided that Google Test will be the unit testing framework. Google Test does not separate itself from CppUnit by much; therefore, the decision to go with Google Test over CppUnit is not strong and may not hold (see table 6). Installation for both Google Test and CppUnit is not simple for the average user. Further, the documentation for installation can be hard to track down for both. Google Test and CppUnit have steep learning curves, with hard to find documentation on explicit use; the user is required to locate and walk through sample code in order to gain partial footing. The two frameworks break even on documentation, CppUnit has better and easier to find installation documentation, Google Test has better feature documentation.

That being said, where Google Test does gain some separation is in features. Google Test allows for performing tests independently of each other, which means that if one test fails, other tests can continue to run. With CppUnit if one test fails the system stops, which increases the time spent debugging. The other area of separation is in syntax. Google Test allows for running all tests with one command without repeating the names of each test to run; the writing of the tests themselves are also less verbose. Google Test is narrowly clearer in their examples when showing which header files need to be included, compared to CppUnit.

3.2.5 Conclusion

The development environment for this project includes: an operating system, a programming language, a documentation generator, and a testing tool. The final decision for each component is as follows:

- **Operating System**: Debian, a GNU/Linux distribution
- **Programming Language**: C/C++
- **Documentation Generator**: Doxygen
- **Unit Testing**: Google Test

3.3 Internal System Software

Internal system software is the software that a system runs and the way that software is designed can make updating or even just operating the system incredibly easy or almost impossible. SidCon is the software that each siderostat currently runs and is approximately 33,000 lines of code, including operations such as getting feedback from a quad-cell to move the narrow angle tracker, getting orders from the observers to update the position of a siderostat system, and doing the calculations based on input on how much electricity needs to be pulsed to move the motors directly on the siderostat. However there are multiple siderostat stations operational every time the NPOI is conducting observations.

Each siderostat station has its own copy of the code locally and any software updates require recompiling all of the code at each station. With the potential changes to hardware the issue arises regarding where to have the majority of the code. Either most of the code is centralized and communicates orders with each siderostat station or continue to have each siderostat station with a copy of the entire code base. With a centralized system, the stations would only have the part of the software that receives an input that tells the system how much to move the mirror. All the calculations for how much the mirror should move would be handled by the centralized system

There are multiple factors at play when choosing whether to have a centralized section that communicates with smaller subsections or having each siderostat station having the entire code base. These include:

- Easy to update
- Easy to debug errors
- Easy to navigate code

These factors are important for the client for ease of operations and for maintainabilty.

3.3.1 Centralized Software

Centralizing the software reduces the amount of code at each siderostat station while having a new computer that communicates with each station telling them to do commands. However, since the code at each station would only be the software that converts signals on how much the motors should move, the only time the software at the siderostat stations would need to be updated would be when motors where changed, and that does not happen very often. The centralized portion of the code could then be updated at one place making it easy to push updates, although there is a slight downside that if the motors were to be updated then each siderostat station would have to receive a software update.

As for debugging errors, in a centralized system there is more communication between computers[20] which in turn can cause networking related bugs, making the search for a bug more costly. But, once a bug is found and fixed, it only has to be updated at the centralized computer if that is where the bug originated. Networking bugs can be hard to find, however fixing the bug on one computer takes less time than multiple computers.

Finally, in a centralized system there will be fewer copies of the same code, but there may be more overhead code needed to have the computers networked. Since more of the code is original, fewer lines are copied, the code becomes easier to navigate.

3.3.2 Independent Systems

Having each siderostat system having its own copy of the software, running independently of each other siderostat system, requires that any and all updates to the software be done to each station, requiring manually going to different locations to install the same update multiple times, and if one of the siderostat stations is forgotten, time and money could be wasted looking for a nonexistent bug in the code when a station just needed to be updated.

Since each siderostat system has a copy of the entire software[20], if one of the siderostat systems has an error there is no indication on whether the bug is related to receiving input, moving the mirror, or sending output to the mirror. Basically the bug could have originated from anywhere in the code base, making debugging a nightmare because potentially to debug the system the entire code base would have to be sifted through to find the location of the bug. Then after the bug is found, fix and then push the updates to all the systems which as discussed previously is not ideal.

For code readability, there is only one set of code between all of the siderostat stations so a developer can read the software from one siderostat and know what all the rest of the siderostat systems have in terms of software, therefore code readability is an aspect of independent systems[20].

3.3.3 Conclusion

	Centralized Software	Independent Software
Easy to update	✓	✗
Easy to debug errors	✓	✗
Easy to navigate code	✓	✓
Total(✓)	3	1

Table 7: Internal Software Comparison [20]

Centralized system had the highest score out of the two explored options, along with a desire from the client to have a centralized system instead of the current independent systems, the project will proceed to convert the existing code that is on each siderostat system and centralize the software. This implementation will involve communications between a centralized computer and each siderostat system.

3.4 User-facing Software

User-facing software, or more specifically a user interface, is a piece of software that is meant to interact with an underlying program or system, while making it convenient and accessible for the user to manipulate this underlying system. In the case of this project, JavaCon, the user interface used by observers at NPOI, achieves the job of interacting with the underlying system, SidCon, but fails to do so in a convenient or accessible manner.

JavaCon has a convoluted codebase and a lack of usability. The convoluted codebase makes it difficult to update and manage the software because it is hard to navigate and there is very little to no documentation on it. Since JavaCon is not a user-friendly interface, observers must painstakingly go through a manual to look up kernel commands that then need to be inputted manually into the computer.

On top of the software being difficult to manage and use, the operations it provides are a small set of basic commands that could easily be displayed on a graphical or terminal-based user interface. If a new interface is implemented for the observers, this would improve the usability of the interface, as a manual would no longer be needed, and it would allow a developer to add additional features.

This introduces the solution to this problem. The plan is to either modify the existing JavaCon code into a more usable interface or create a new user interface – graphical or terminal-based – altogether. Whichever solution is chosen will require a redesign of the user interface, which requires that an appropriate software is chosen to facilitate creating a product that is convenient and accessible to the user. The user interface software choices will be documented in this section.

3.4.1 User Interface API

There are several software packages out there that would allow a developer to create a user interface. A package such as this is considered an application programming interface (API). The two categories of software APIs that will be considered are graphical-based and terminal-based.

Since striving for a user interface that is mainly accessible for the observers at NPOI, it needs to be considered whether the user interface is graphical and whether it has the required functionality. However, since the project needs to be completed in a constrained window of time, the software API must also be easy to use and lightweight.

Here is how each criteria will be categorized:

- Graphical: Yes or no
- Functionality:
  - Create basic input and outputs: Yes or no (required)
  - Run on a graphical back-end: Yes or no (optional)
  - Draw diagrams: Yes or no (optional)
- Documentation: Yes or no
- Lightweight: Yes or no
- Development environment: Programming language the API supports

There will be four options considered for the user interface API. Three of the options involve replacing JavaCon with a new program altogether, while one simply involves updating JavaCon and using the environment it was built with, Java. Because it was decided that the development language for this project will be C/C++, the only replacements that will be considered are options in this language. The candidates are:

Updating JavaCon and using Java's built-in API
This first option would provide a head start, as there would be no need to rewrite the interface between SidCon and the user interface. However, as mentioned in the introduction to this section, JavaCon is a messy and convoluted codebase, especially when considering its functionality, causing a lot of developer friction. Additionally, Java's built-in graphical API is not the greatest product when it comes to usability and ability to design coherent user interfaces.

The Java API was tested by creating a basic interface with a few buttons and dialog boxes. Overall, the testing process went smoothly, as the documentation was easy to navigate and understand. However, the API was found to be a bit janky to use and fairly limited in its ability to design interfaces. Also, since the system uses the Java environment, it was a heavy weight user interface requiring a lot of unnecessary bloat in order to run the program.

Here is how this option was scored:

- Graphical: Yes
- Functionality:
  - Create basic input and outputs: Yes
  - Run on a graphical back-end: No
  - Draw diagrams: No
- Documentation: Yes
- Lightweight: No
- Development environment: Java

Dear ImGui
Dear ImGui is a lightweight, extensible, and easy-to-use graphical user interface API. It can be implemented into a project simply by including a C/C++ header file and then making API calls. One can use a plethora of graphical APIs as the back-end, such as OpenGL, which allows the programmer to draw graphics alongside the user interface, making for a very flexible design. Contrary to most other graphical user interface APIs out there, ImGui is an immediate mode instead of a retained mode interface, making it a stateless and functional style programming interface.

This option was evaluated by using example code provided by the Dear ImGui repository and experimenting with the API. The immediate mode interface was found to be pleasant to work with and the most lightweight out of all the options, as all is required is it to include a single header file to run the program. It was found that the API offers many different types of interfaces, such as the ability to make histograms or add color wheels, which would allow a developer to make very detailed designs. The underlying OpenGL back-end was also easy to work with and would expand the ability to draw diagrams or other images if need be.

Overall, Dear ImGui scored well, here are the results:

- Graphical: Yes
- Functionality:
  - Create basic input and outputs: Yes
  - Run on a graphical back-end: Yes
  - Draw diagrams: Yes
- Documentation: Yes
- Lightweight: Yes
- Development environment: C++

Qt GUI

Similar to Dear ImGui, Qt GUI, or just Qt, is a C/C++ graphical user interface API. However, compared to Dear ImGui, Qt is much more heavyweight and is essentially a framework on its own. The API works in a similar fashion as the Java graphical API, where one creates objects and manages the state. Qt also supports interfacing with OpenGL, but in a less flexible way. Compared to the other APIs, Qt has a large learning curve and thus requires a large and extensive set of documentation.

The Qt GUI framework was evaluated by reading through its extensive documentation and by creating an example application to test with. The first impression of this API was that the setup was complex and difficult to start a simple application with it. Additionally, it took a chunk of time to figure out how to integrate the GUI into a larger application, rather than just a basic example application. However, once as Qt was configured and working, it was found to be more usable than the Java API and offered a better set of design choices. The results for this option were as follows:

- Graphical: Yes
- Functionality:
  - Create basic input and outputs: Yes
  - Run on a graphical back-end: Yes
  - Draw diagrams: No
- Documentation: Yes
- Lightweight: No
- Development environment: C++

ncurses
The other options involved replacing JavaCon with a graphical user interface, but there also exists the option of using a terminal based interface. The advantages of a terminal based interface over a graphical interface is that the learning curve is much lower, and it is much easier to integrate into an application. However, the trade-off is that the application becomes a lot less usable and more difficult to design a good interface with. The ncurses software is exactly this: an API that allows the user to create a terminal based interface in C/C++.

The ncurses software was evaluated by first examining existing ncurses applications to see if it would fit the needs of the project. Since the set of functions that is required for the user interface is minimal, it was chosen as a viable option. Then, a basic application was created to test the usability of the software. The API was easy to interact with, but it lacked much of the functionality that the other APIs had. Additionally, it was difficult to design an interface that was more usable than basic terminal output. This API was graded accordingly:

- Graphical: No
- Functionality:
  - Create basic input and outputs: Yes
  - Run on a graphical back-end: No
  - Draw diagrams: No
- Documentation: Yes
- Lightweight: Yes
- Development environment: C++

3.4.2 Conclusion

	Java API	Dear ImGui	Qt GUI	ncurses
Graphical	✓	✓	✓	✗
Functionality	✓	✓	✓	✓
Documentation	✓	✓	✓	✓
Lightweight	✗	✓	✗	✓
Total (✓)	3	4	3	3

Table 8: User interface API comparison

After evaluating all four candidates, it was decided to use Dear ImGui as the graphical user interface API. Dear ImGui provides the best usability and designability out of all the options, and it is also the easiest to work with from the tests. On top of that, it allows for the freedom to expand on the existing functionality by allowing it to interface with OpenGL to draw diagrams or other imagery for the observers to see visually.

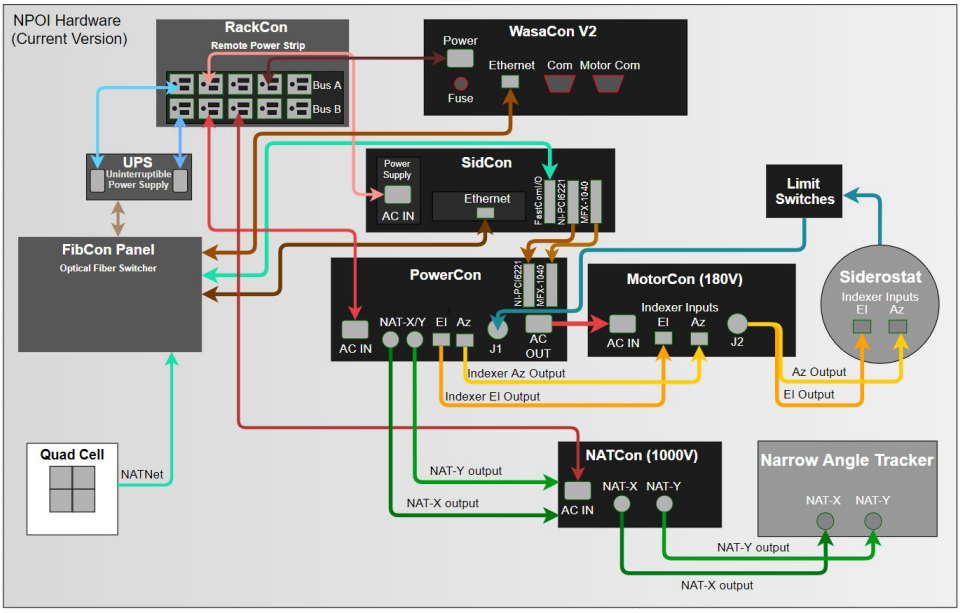
The other options also provide the level of usability and designability that was needed. However, since there are time constraints, it was decided that the most workable solution should be used, which also happens to be Dear ImGui. It is the most lightweight API and does not require navigating through pages of documentation like Qt does, for example.

While communication interaction between SidCon and the user interface will have to be rewritten, this trade-off is reduced by the fact that there will have to be rewrites of the communication code in the SidCon for the centralized controller system, in the first place.

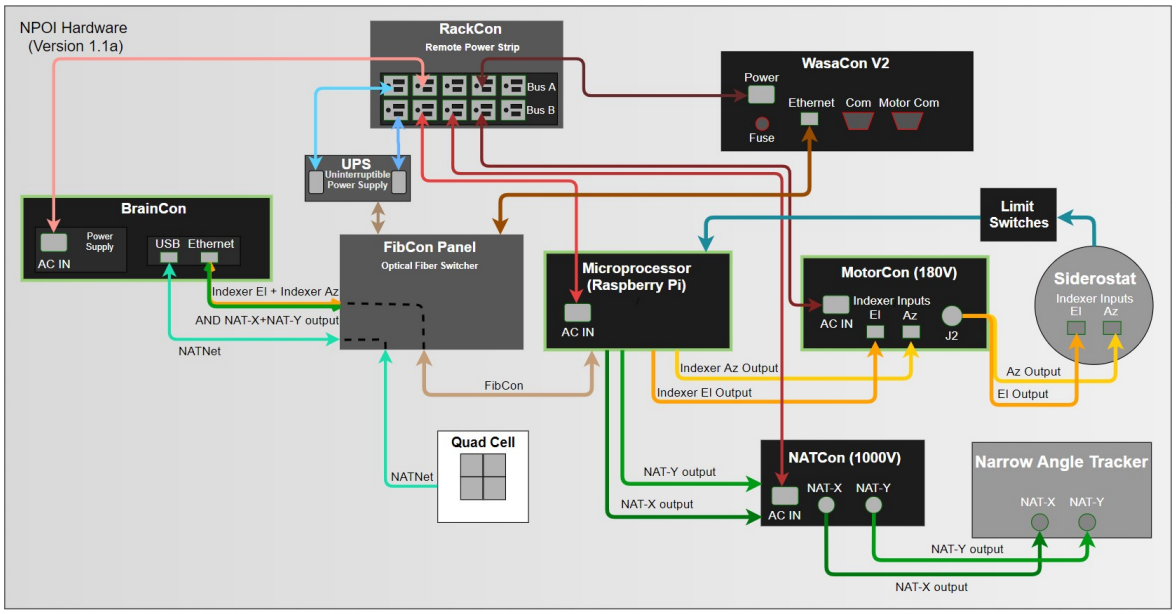
Overall, the confidence level for Dear ImGui being the correct design decision is high. It provides the functionality that is needed, and some more, while also being a very lightweight and user-friendly API to work with. Lastly, based on the evaluations and testing, it will cause the least friction and be the most cost effective option in terms of developer time.

4 Technology Integration

It is difficult to imagine all of the hardware pieces integrated into one system, so to show integration of the technology covered in this document, this section shows a few diagrams to help visually understand the desired changes to be made.



In the figure above, the current hardware of a single siderostat station is displayed. The SidCon notably has three PCI cards and an Ethernet cable connecting to NATNet, the network used at NPOI for the NAT and quad cell feedback. All of these cards could be replaced if PowerCon was replaced with a Raspberry Pi, as seen in the figure below. The SidCon is the main issue to address, as there are few motherboards left to replace them and they currently operate in a one-to-one fashion as opposed to one-to-many.



As seen in the figure above, all of the functionality of PowerCon has been replaced by the Raspberry Pi microprocessor. The power output from PowerCon to MotorCon has been moved to the RackCon (a remote controlled power strip). This solution utilizes the fiber optic network to communicate between the BrainCon (replacement for SidCon) and the microprocessor, instead of a local, direct connection to PowerCon. This way, a single BrainCon can operate multiple Raspberry Pis over the network whereas before a single SidCon could only connect to a single PowerCon. Given that the CPU will be able to handle 12 threads, we could operate 12 siderostat stations nearly in parallel. The limit switch feedback will also be interpreted by the microprocessor. The NAT-X and NAT-Y output to NATCon as well as the indexer elevation and azimuth to MotorCon will need to be reproduced by the Raspberry Pi.

Overall, this solution looks to be the best way to create a parallel system. As it stands, the PowerCon documentation does not exist at NPOI. PowerCon is a growing threat to the system; if all of them fail, there would be a devastating downtime for the system. There is a similar urgency of replacing SidCon, as the custom motherboards they use are running out.There are many ways to update this system, but ridding the system of custom built embedded systems is a good way to ensure that the system can be easily maintained by future engineers.

5 Conclusion

The goal of the project is to update several components at NPOI, some hardware related, and some software related. The current system in place at NPOI is outdated, as hardware is constantly breaking down, and the existing system is overly complex. As a result, team Astraea has attempted to document a solution for their problem. One that will reduce the complexity of the system, reduce the rate of failure, and stay robust so that NPOI can continue to operate as hardware changes present themselves in the market.

The design decisions detailed in this document cover the technology that the team considers critical to the development of a centralized computer system, which can control stepper motors, handle the limit switch and NAT feedback, and communicate to multiple stations over a fiber-optic network. To summarize:

- The team decided to use Icron Ranger as the fiber-optic network interface.
- The team decided to go with a custom-built computer which includes a processor that is capable of programming with a high number of threads.
- The team decided to use a Raspberry Pi to manage the feedback between the limit switch and control the stepper motors.
- The team decided on an appropriate development environment, which includes the Debian distribution of GNU/Linux, the C/C++ programming language, Doxygen for documentation, and Google Test for unit testing.
- The team decided to create a software that depends on a centralized system.
- The team decided to use ImGui for graphics user interface applications.

To go along with each of these decisions, here is a table that indicates the team's level of confidence for each piece of technology decided upon:

Challenge	**Potential Solution**	**Confidence Score**
Fiber-Optics	Icron Ranger 2324	4
Motor and Limit Switch Controller	Raspberry Pi	3
Central Processing Unit	Intel Core i7-6800K	5
Motherboard	MSI X99A	5
Operating System	Debian	5
Programming Language	C/C++	5
Documentation Generator	Doxygen	3
Unit Testing Framework	Google Test	3
Software Location	Centralized	5
GUI	Dear ImGui	5

Table 9: Technological Analysis Choice Conclusion

To conclude this document, the team would like to emphasize their excitement and desire to succeed on this project. Not only will the team be saving NPOI in the financial department by reducing the amount of failures in the system, but the system will be significantly less complex and easier to manage. Going forward, the team hopes NPOI can continue to contribute to the field of astrometry and doso efficiently.

6 References

1. https://fastcomproducts.com/products/fastcom-fscc4-pcie/
2. http://www.icron.com/products/icron-brand/usb-extenders/fiber/usb-2-0-ranger-2324/
3. https://www.adafruit.com/product/3055?src=raspberrypi
4. https://store.arduino.cc/usa/arduino-uno-rev3
5. https://store.arduino.cc/usa/arduino-ethernet-shield-2
6. http://divapps.parker.com/divapps/emn/prior_version_compumotor /english/pgs224_242_zeta.pdf
7. https://www.amazon.com/Intel-i9-9820X-Processor-Unlocked-LGA2066/dp/B07KCCH7JL/ ref=sr_1_26?m=ANNSJU9W28Y9J&marketplaceID=ATVPDKIKX0DER&qid=1573182391 &refinements=p usepackage_4%3AIntel&s=merchant-items&sr=1-26
8. https://www.newegg.com/intel-core-i7-6th-gen-core-i7-6800k/p/N82E16819117649? Item=9SIA4YU9MC6119&Description=Intel%20Core%20i7-6800K%20Broadwell-E% 20LGA%202011
9. https://www.evga.com/products/product.aspx?pn=151-SX-E299-KR
10. https://www.google.com/shopping/product/1?safe=off&rlz=1C1CHBF_enUS758US758& sxsrf=ACYBGNSWfIV_5Q9_2z9gWGC0smOkjUSyrw:1573183023446& q=MSI+X99A+GAMING+PRO+CARBON+LGA+2011-v3&biw=1057 &bih=1022&prds=epd:9290765876892994957,oid:9290765876892994957, pid:9290765876892994957,prmr:3&sa=X &ved=0ahUKEwj01uPQ1NnlAhVDIKwKHU7p80Q1sEDCE0
11. https://www.debian.org/doc/manuals/debian-faq/ch-ftparchives#s-frozen
12. https://www.debian.org/intro/about
13. https://wiki.archlinux.org/index.php/Arch_Linux
14. https://docs.fedoraproject.org/en-US/project/
15. https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B
16. http://www.doxygen.nl/manual/index.html
17. https://www.naturaldocs.org/getting_started/
18. https://freedesktop.org/wiki/Software/cppunit/
19. https://github.com/google/googletest

20. https://www.researchgate.net/profile/Robert_St_Louis/publication/220427171_Centralization_vs_Decentralization_of_Application_Software/links/54612d8a0cf2748710526e8d.pdf