Link to published site:

https://ese5180.github.io/fp-f24-report-website-t03-skyscan/

Review your IoT Venture Pitch assignment. Identify what changed throughout the project, including:

Target Market & Demographics

The primary target market for SkyScan includes solar business owners who operate in the U.S. solar energy market, which has a total addressable market (TAM) of $160.3 billion. Within this, the serviceable addressable market (SAM) is estimated at $32.06 billion, targeting the 11,000 solar business owners who would benefit from optimized solar panel efficiency. The serviceable obtainable market (SOM) is calculated as $77.5 million, reflecting a 22% capture of the SAM. These business owners are motivated to reduce costs and maximize renewable energy output, making SkyScan's real-time optimization an attractive solution. Secondary markets include residential solar users and community-based renewable energy projects, particularly in regions with consistent sunlight. Although there were not any large changes with the market or demographics, we were able to specify more about who would purchase our product and the secondary markets. Our target customer includes solar business owners who operate in the U.S. solar energy market, which includes Nextera Energy Resources, Cypress Creek Renewables, Recurrent Energy.

Security, Hardware, & Software

SkyScan’s security features include encrypted wireless communication using the LoRa protocol, which ensures low-power, long-range data transmission with reliable security.

Hardware Requirements:

- Servos: Dynamically adjust solar panel tilt for optimal sun exposure.
- Microcontrollers (MCUs): nrf7002 DK and sparkfun lora. Act as the gateway, processing data from sensors and transmitting commands.
- Sensors: Solar panels monitor sunlight intensity and each given angle and send back this information. Later features of this may include sensing of panel temperature and other environmental factors.
- Solar Panels

The only hardware changes that were made throughout the project were the battery charging circuit (see below for more information on that), and the servos we switched from the 180 degree ones to the 360 degree ones

Software Requirements:

- Memfault
- Web Scraping: Utilizes tools to extract sun exposure angles from public sources like Google’s Sunroof API. Deployed costs are $5 per user per month.
- Amazon EC2: Server infrastructure for web scraping and data analysis.
- Amazon S3: Data storage for real-time and historical sun exposure data.

Throughout the project we had many different software iterations and additional features. We started with the code for the google sunroof API/data and then adding our code to adjust the panels according to this data. In section C below, there is a figure that maps out a highview of our system. The next step was to integrate that system together. The next step was involving the Amazon endpoint (the data to which in the code was removed as the IoT core endpoints are not to be posted on a possibly publicly accessible system).

Together, these components create a seamless ecosystem that supports dynamic adjustments and secure data processing.

Security Requirements:

To get the main functionality working before the end of the sem, we did not have time to flush. Despite my (Chris) slipup in the presentation, security is actually not an afterthought. However, I still believe this is sligtly acceptable in this project because there are not clear incentives on creating malicious behaviour for our devices. However, before publicly releasing, we would be sure to add verifification of the integrity and authenticity of messages.

Product Function & Components

SkyScan is a solar optimization system designed to maximize energy capture through dynamic panel adjustments and send back accurate data on the position/sun exposure. The core of the system is a centralized microcontroller unit (MCU), which acts as the gateway for processing data and managing commands. The system starts with solar irradiance sensors that collect real-time sunlight data, which is transmitted via LoRa wireless communication modules. The LoRa 1 module relays this data to the MCU through a UART 2 channel, enabling the processing of sensor inputs.

Once processed, the MCU calculates the optimal panel tilt and sends commands back to the LoRa 2 module, which communicates with servo motors attached to multiple solar panels. These servos adjust the panel angles in real-time to maximize exposure to the sun. Data from the MCU can also be accessed via a USB (UART 0) connection to a laptop, allowing for monitoring and troubleshooting using tools like a serial terminal or Python scripts (via PySerial). The product's ecosystem ensures seamless interaction between its components. The multiple panel and servo array enables scalability, while the secure and energy-efficient LoRa protocol ensures reliable long-range communication. With real-time control and monitoring capabilities, SkyScan offers users an integrated solution for optimizing solar energy generation. A quick change was that we weren't able to use the UART 1 likely due to cross overs on the board (met with Miles to learn more about this). Another change was the fact that we weren't sure if we could

The system starts with integrated sensors that monitor sunlight intensity, panel temperature, and environmental factors to collect real-time data. This data is transmitted via LoRa wireless modules to a central microcontroller unit (MCU), which processes the inputs and issues commands to servo motors. These motors reposition the solar panels to achieve optimal tilt based on the sun's current position. Additionally, the system incorporates a secure online web interface, enabling users to monitor solar performance metrics, analyze historical data, and make adjustments as needed. SkyScan's seamless integration of hardware and software components ensures that users can maintain peak solar efficiency without manual intervention.

Power & Cost Budgeting

The SkyScan system is designed with both cost efficiency and sustainability in mind. The estimated manufacturing cost (COGS) for each device is $129.31, which includes $47.81 for the solar panel, $13.46 for the microcontroller, $5.25 for servo motors, $2.52 for wireless modules, and $32.27 for additional sensors and components. Assembly and other miscellaneous costs add another $30 to the total. With a 20% markup, the final selling price is set at $155.17 per device. For software, a one-time hardware fee of $241.50 covers initial deployment, while a monthly subscription fee of $15 per user ensures access to cloud-based services, including real-time data processing, updates, and storage through Amazon EC2 and S3. Additionally, SkyScan is designed to operate on minimal energy by leveraging solar power, making it not only cost-effective but also sustainable. This pricing strategy balances affordability for customers with profitability, ensuring a scalable and accessible solution for the renewable energy market.

What parts of your project would you consider a success? Why?

Our project had several successes that we are proud of. Setting up the Selenium server to scrape sun position data was our first achievement, as it allowed us to efficiently collect external data to input for our system. Additionally, outlining our specifications early on provided a clear roadmap and kept the team aligned throughout the development process. We were also able to successfully establish UART communication between the nRF and the LoRa machine. This was a significant milestone, as reliable data exchange between the computer, the nrf, and the gateway lora was the backbone of our system. Despite its use in our final product, setting up the API endpoint for fetching data was another success, as it enabled real-time data retrieval for future iterations. Configuring and testing motor controls was a highlight of the project. We managed to implement precise movements and effectively interface with the hardware. Moreover, designing the test bench for the system was another point of pride. The design ensured the developing product could be easily worked on by giving the parts structure while remaining open to adjust and tinker components. Finally, setting up basic LoRa communication provided a functional wireless communication layer, an essential feature for the system's overall functionality.

- Setting up selenium server to scrape sun position data
- Outlining our specifications
- Getting UART communication working successfully between the nrf and LoRa machine
- Setting up the API endpoint for fetching the data
- Configuring and Testing motor controls
- Designing test bench
- Setting up basic LoRa communication

What parts of your project didn’t go well? Why?

Despite these successes, there were also challenges that hindered some aspects of our progress. Integrating Memfault into our codebase proved to be a difficult task due to compatibility and dependency conflicts with our local machines. We also faced significant challenges when interfacing with multiple UART ports, as configuration nuances, particularly with device tree settings, created obstacles during development. Understanding how zephyr truly integrated as a build system, nrf port manager, and OS was difficult– but in the end it allowed us to grasp a deeper understanding of how to deploy production-grade code.

Another notable issue was Windows disallowing two read connections to the same serial port. This forced us to migrate to using the Python serial library in our server code, adding unanticipated complexity to the software design. Additionally, hosting our server code on a separate machine and having the nRF use WiFi to fetch data introduced reliability issues, which required further debugging and refinement. Following test-driven development (TDD) was another area where we struggled. While the approach is highly effective in theory, adhering to it consistently proved challenging due to time constraints and the iterative nature of our development process. These struggles served as valuable learning experiences, highlighting areas for future improvement in our planning and execution.

- Integration of memfault into our codebase
- Interfacing with multiple UART ports
    - Challenges with configurations → device tree nuances
    - Windows disallowing two read connections to the same port (forces us to migrate to using the python serial port in our server code)
- Hosting our server code on a separate machine, and having the NRF use wifi to fetch it
- Following Test driven development

If you had to do it again, how might you change your development approach given the finite time and money resources?

- Given another chance, we probably would focus on getting software running even if most of our hardware specifications haven't been built yet at all. This is because the software was an important part of our project, especially in this mostly software based class. Chris spent a lot of time on the hardware circuitry, servos, and panel mounts. It was just very hard to have everybody working on software simultaneously; however, we definitely should have done just software at first.
- Additionally, as stated before, we definitely should have leveraged TDD more. It was not important for the goals of the MVP; however, it is nonetheless best practice and would have been beneficial

Would you change your system design after this development cycle?

- I definitely would choose a board that could use lora and wifi (if there is one). A lot of the pain of our project has been figuring out how to send data between our devices, so being able to combine two boards would have been really helpful. I also would look into the servos mor
- Additionally, for the MVP at least, I implemented battery charging differently: instead of having the same panels both measure and charge, I would have some panels just for charging. That way, the circuitry would be greatly simplified.

Project Media

MVP Pics

Field device:

Gateway device:

DEMO LINK:

https://www.dropbox.com/scl/fi/pjv16b0ebjuh4miqluv8f/12-9-24-1-46-57.mov?rlkey=sufy07ei8y44om1euldi0zu07&st=nkn57wia&dl=0

Memfault: