Solar Array Analysis Suite

Honors Capstone

Ian Stewart - icstewar@umich.edu

Advised by Professor A Harvey Bell, IV

Last Updated: 12/09/21

Table of Contents

Background

In 1987, the South Australian government hosted the first ever World Solar Challenge, a race across the Australian Outback in which electric vehicles powered only by the sun would compete to complete the race in the shortest time. The winning car, Sunraycer, had been built by engineers from the American General Motors Company, who took such a liking to the idea that they decided to host a collegiate race for solar-powered electric vehicles in the United States three years later in 1990, with the top three teams winning sponsorship to travel to Australia to compete in the next iteration of the World Solar Challenge, held later that same year.

The University of Michigan Solar Car Team (UMSCT) was founded in 1989 to compete in the aforementioned GM Sunrayce. The team's first car, Sunrunner, took first place and the team was provided with the opportunity to fly their car to Australia to compete in the 1990 World Solar Challenge, where they took third place. This kickstarted the most successful project team at the University of Michigan, and in the 32 years since its founding the team has built 15 solar-powered electric race cars, taken nine national championships, one international championship, and earned a podium finish in the World Solar Challenge seven times.

Although it had initially occurred every three years, in 1999 the World Solar Challenge instead began running every two years. This streak would only be broken in 2021, when concerns over the COVID-19 virus led the South Australian government to cancel the event for that year. Thus, the University of Michigan Solar Car Team was forced to determine what they would do with the designs for a car they had been working on since the completion of the 2019 World Solar Challenge two years prior. Ultimately, it was decided that the team would attempt a completely solar-powered Cannonball Run from coast to coast across the United States of America.

As the car the team had been working on had originally been designed for the rules and regulations of the now defunct 2021 World Solar Challenge, the team needed to re-run all of its analysis for a route east to west across the United States instead of the originally planned route from north to south across Australia. Due to the increased distance and presence of mountains, this route will put even more importance on the performance of the solar array than the World Solar Challenge would have. Furthermore, the sun will now be largely on one side of the car rather than mostly behind it, which will increase the amount of energy lost to shading and cause light distribution across the array to be uneven.

It is these changes in circumstance that pushed me to make the Solar Array Analysis Suite, which can take a route file and a weather file generated by the Strategy division and an array designed by the Array division and analyze the data to predict the total system performance of the array under the given conditions.

Initial Requirements

The software suite I designed was made specifically to integrate with the current workflow of our Array and Strategy divisions. As such, it reads in data prepared by the Strategy division through the usage of their own pre-existing tools, as well as CAD data exported directly from the Array division's prior work in Siemens NX. This ensures easy adoption of the suite is possible without forcing either division to change what they already do every day.

It has also been designed to include as many different sources of efficiency loss as possible. In particular, the suite currently allows for analysis to be done while incorporating efficiency losses due to the base efficiency of the cells, the efficiency of cell encapsulant, the angle of the sun, the temperature of the cells, any shading caused by the car itself, and the maximum power point trackers (MPPTs), each of which will be explained in greater detail below. This far outstrips our previous spreadsheet to estimate array efficiency and our other spreadsheet to estimate MPPT efficiency, resulting in far more accurate comparisons between competing array designs and far more accurate estimates of absolute system efficiency, with the potential for future improvements to make the system model an even better predictor of real-world performance..

Lastly, the software suite has been designed in such a way as to be as intuitive as possible whilst also working within the strict time constraints imposed by the design cycle. Important information is displayed through text readouts and graphically in a way that should be easy for future users to understand, and should help point them in the right direction regarding decisions about array design, both in a comparative and in an absolute sense.

Initial Designs

This software suite is not, as I would hope should be obvious, the team's first time using math to help inform and justify array design decisions. Our modern models of array performance began in 2018, when Array Lead Eric Brown decided to create a spreadsheet to consolidate the results of loss factor calculations the team had been using to estimate our array's practical efficiency.

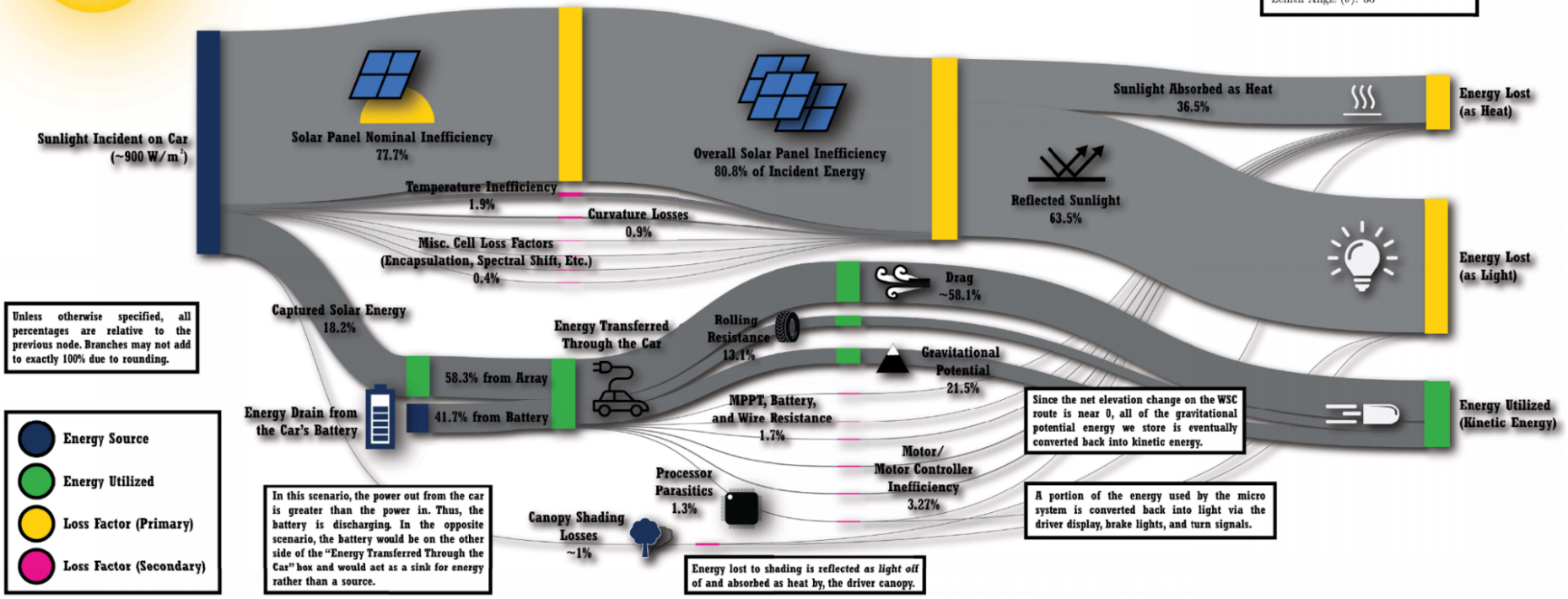
The chart on the next page shows a high-level overview of the power flow in a solar car. On the left, energy comes into the car via the sun and the car's internal battery, and on the right that energy leaves the car either as kinetic energy helping to propel the car or as waste energy in the form of light or heat. It is all of the losses acting on the sunlight incident on the car that this project hopes to capture, and so secondary loss factors operating on energy transferred through the car will not be considered in this software suite.

ENGINEERING HONORS PROGRAM
UNIVERSITY OF MICHIGAN

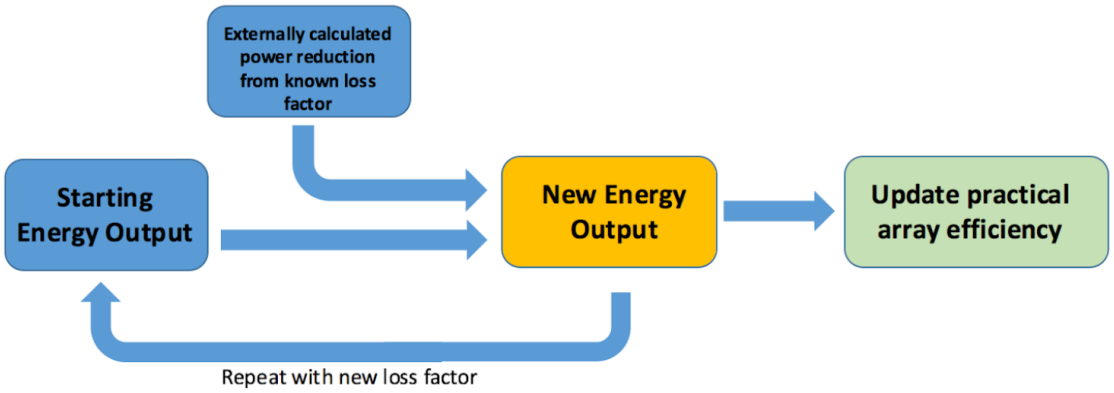
University of Michigan solarcar

Energy Flow in a Silicon Solar Car

Based on Aurum, University of Michigan Solar Car's 2015 Vehicle



The original spreadsheet idea was to take a predicted energy output of the array under lossless conditions, and then use external calculations (done via hand or via MATLAB) to act upon that predicted output and correct for the expected losses to produce a new energy output prediction, which could then be used to calculate the practical efficiency of the array.



This would allow data to be easily stored in one centralized place for different car and array designs, allowing easy tweaking of numbers and allowing some calculations that had previously been performed by hand to be automated by the spreadsheet.

How to Interpret this Sheet:
Each block moving from left to right across the sheet applies the effects of a different loss factor to the expected power output of the array
The *"P expected"* and *"Total Energy"* tabs furthest to the right take all currently modelled loss factors into account. Most power reduction factors are estimated/calculated externally to the spreadsheet via matlab

Summary

**All analysis performed assuming AM1.5 spectrum*

Nominal %eff	26	
Active Area/Cell	0.000855	*Pointing*
Total Area	3.56	*Pointing*
Total Cells	4163.74269	*Pointing*
Voc		*Driving*
Isc		*Driving*
Vmpp	0.97	*Driving*
Impp	0.229	*Driving*
Pcell [W]	0.22213	*Driving*
Nominal P_array [W]	924.8921637	*Driving*
Practical P_array [W]	883.7851342	*Driving*
Total Losses	0.0444	*Driving*
Practical array eff %	24.84442445	*Driving*
		Pointing
		Pointing

Modelled Losses	
Thermal	0.0232
Spectral Shift	0
Curvature	0
Shading	0.0212
Reflection	0

*very rough approximation, basedon scaling of Novum loss

Shading Loss
0.0212

***strat

Base Input Power

Time	Irradiance [W/m2]	Avail.Power [W]	P expected [W]
6:00 AM	0	0	0
7:00 AM	325	1157	30082
8:00 AM	766	2726.96	70900.96
9:00 AM	536	1908.16	49612.16
10:00 AM	752	2677.12	69605.12
11:00 AM	916	3260.96	84784.96
12:00 PM	1013	3606.28	93763.28
1:00 PM	1035	3684.6	95799.6
2:00 PM	981	3492.36	90801.36
3:00 PM	856	3047.36	79231.36
4:00 PM	670	2385.2	62015.2
5:00 PM	437	1555.72	40448.72
6:00 PM	644	2292.64	59608.64
7:00 PM	0	0	0
	Total Energy [J]	*114459696*	~~*2975952096*~~

*Irradiance data for Oct. 10, 2019 in Alice Springs AU
*Irradiance is horizontal, direct+diffuse

Thermal Losses

*refer to matlab script SCT19_ThermalLosses(in drive) for array efficiency reduction as a function of temperature

Array Temp	Power Loss	P expected [W]
75	0.0450	0
75	0.0450	28728.31
75	0.0450	67710.4168
45.0	0.0180	48719.14112
45.0	0.0180	68352.22784
45.0	0.0180	83258.83072
45.0	0.0180	92075.54096
45.0	0.0180	94075.2072
45.0	0.0180	89166.93552
45.0	0.0180	77805.19552
45.0	0.0180	60898.9264
45.0	0.0180	39720.64304
75	0.0450	56926.2512
75	0.0450	0
	Total Energy [J]	2906775455

Spectral Shift efficiency	Power Loss	P expected [W]
0.15	0.5161	0
0.24	0.2258	23289.29032
0.305	0.0161	69757.39613
0.31	0	49612.16
0.31	0	69605.12
0.31	0	84784.96
0.31	0	93763.28
0.31	0	95799.6
0.31	0	90801.36
0.31	0	79231.36
0.31	0	62015.2
0.305	0.0161	39796.32129
0.24	0.2258	46148.62452
0.15	0.5161	0
	Total Energy[J]	2896576820

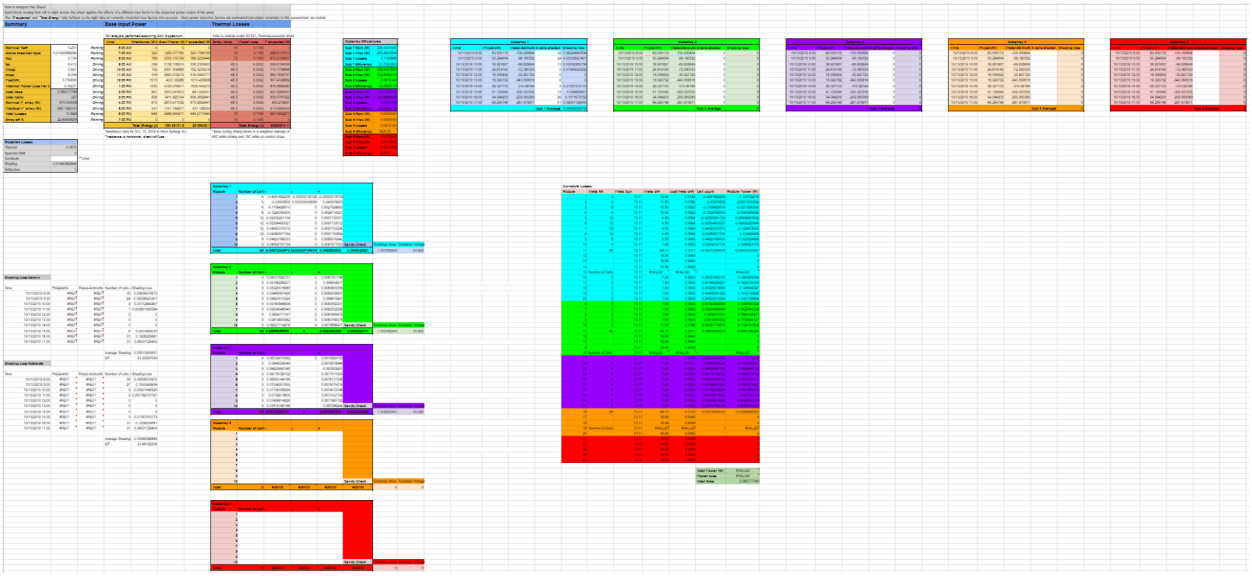
*Temp during driving times is a weighted average of 45C while driving and 75C while pointing at two 30min checkpoints

0.0232452133

Curvature Losses

Module	Theta NX	Theta Sun	Theta diff	cos(Theta diff)	Cell count	Module Power [W]
1	7.06	73.11	9.83	0.9853	46	47.20869896
2	7.06	73.11	9.83	0.9853	46	47.20869896
3	6.82	73.11	10.07	0.9846	46	47.17405686
4	6.44	73.11	10.45	0.9834	46	47.11751605
5	1.5221	73.11	15.3679	0.9643	42	42.18260856
6	1.5221	73.11	15.3679	0.9643	42	42.18260856
7	1.5221	73.11	15.3679	0.9643	42	42.18260856
8	-0.206	73.11	17.096	0.9559	42	41.81418399
9	-0.206	73.11	17.096	0.9559	42	41.81418399
10	-0.206	73.11	17.096	0.9559	42	41.81418399
11	-0.9	73.11	17.79	0.9522	42	41.65551212
12	-0.9	73.11	17.79	0.9522	42	41.65551212
13	-0.9	73.11	17.79	0.9522	42	41.65551212
14	-0.194	73.11	17.084	0.9559	42	41.81687376
15	-3.2339	73.11	20.1239	0.9390	45	44.01134522
16	-3.2339	73.11	20.1239	0.9390	45	44.01134522
17	-3.2339	73.11	20.1239	0.9390	45	44.01134522
18	-5.86	73.11	22.75	0.9223	45	43.22705958
19	-1.94	73.11	18.83	0.9465	45	44.36390437
20	-10.85	73.11	27.74	0.8852	45	41.48839062
					Total Power [W]	868.5961488
					Power Loss	0.04583054524

This would later be expanded upon by myself to provide more calculations natively in the sheet and to allow analysis on the level of each individual subarray rather than only the entire array, consisting of all of the individual subarrays in parallel. As each subarray is really its own self-contained set of solar cells whose performance does not depend on the performance of the other subarrays that make up the entire car's array, this allowed us to identify any specific areas that were particularly weak when designing an array.



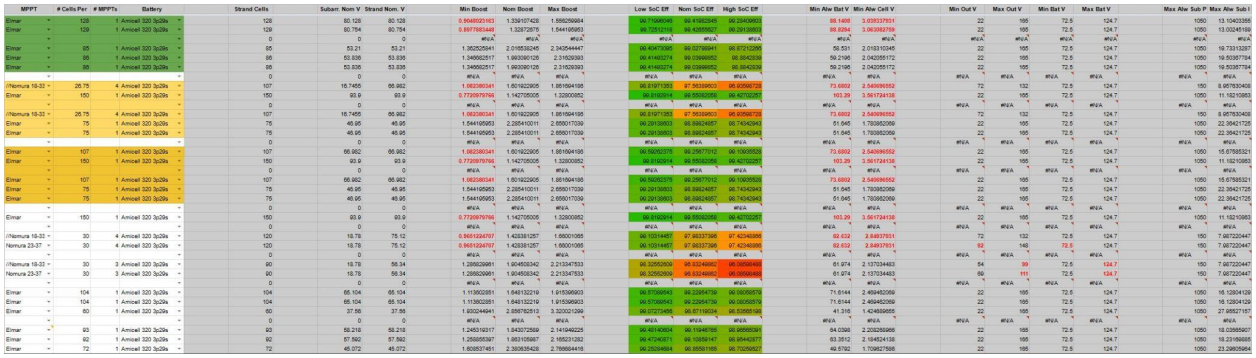
One loss factor that this spreadsheet did not predict were losses due to the efficiency of the team's maximum power point trackers, or MPPTs. These devices are what allow a set of photovoltaic solar cells to actually operate as the power source for a car. An MPPT has an internal algorithm that allows it to track what voltage a string of cells should be at to produce the greatest possible power output, and then holds it at that particular voltage. In order to use this power to charge the car's battery or power its motors, the MPPTs must also act as a voltage converter, boosting the power coming in from the array to the voltage of the battery.

The operation of an MPPT is not 100% efficient - as will be discussed in detail later, their efficiency depends on three fundamental factors: input voltage, input current, and output voltage. However, without very specific testing data for a certain model of MPPT, it is exceptionally difficult to accurately predict the efficiency a MPPT will operate at. As such, our first attempt at this task was a spreadsheet that attempted to predict which arrangement of MPPTs would result in the best MPPT efficiency for a given array setup.

First, then, we needed to pick MPPTs to compare. After some research, we narrowed the field down to two choices - using a different model by the same MPPT manufacturer that had helped power our 2019 car to third place in the World Solar Challenge, or swap to a Japanese manufacturer whose innovative control software allowed multiple MPPTs to be connected in series to help overcome curvature losses.

Once these MPPTs had been identified, I created a spreadsheet that fit a curve to attempt to predict the efficiency of each kind of MPPT under given conditions. In particular, the efficiency was predicted based on the “Boost Ratio” the MPPT was expected to require, which is merely the ratio between the voltage the MPPT will output (dictated by the voltage of the battery) and the voltage of the maximum power point of the array as enforced by the MPPT (essentially the input voltage to the MPPT from the array). Although this was not perfect, it was the best I could do in a spreadsheet format, and allowed decent approximation under most operating conditions.

The rest of the spreadsheet was designed to allow rapid analysis of various array configurations, with the user merely needing to select from a predetermined MPPT type, battery type, and then input the number of cells per MPPT and the number of MPPTs in parallel. From this information, the spreadsheet would report the expected voltages of each subarray and battery, the minimum, maximum, and nominal boost ratios, the minimum and maximum operating conditions of the battery and each subarray and whether any of them would be violated, and of course the predicted MPPT efficiency. In order to add a new MPPT or battery type, all a user needed to do was punch its data and curve fit parameters into a dedicated page on the spreadsheet. The new MPPT or battery would be automatically added as an option able to be selected on the main calculation page.



	Array Min V	Array Max V	Array Max I	Array Max P	Min Boost Ratio	Max Boost Ratio	Output Min V	Output Max V	Supply Power	MPP Eff 2	MPP Eff 1	MPP Eff 0
Elmar	20	150	7	1050	1.1	3.5	22	165	0.24	0.101322	-0.91842	100.468
//Nomura 18-33	10	30	7	150	1.1	3.5	18	33	0.054	0	-2.41716	101.436
Nomura 23-37	10	30	7	150	1.1	3.5	23	37	0.054	0	-2.41716	101.436

	P	S	Min V (Cell)	Nom V (Cell)	Max V (Cell)	Min V (Bat)	Nom V (Bat)	Max V (Bat)	Max Ic (Cell)	Max Id (Cell)	Max Ic (Bat)	Max Id (Bat)	Res (Cell)	Res (Bat)
Amicell 320 3p29s	3	29	2.5	3.7	4.3	72.5	107.3	124.7	10	28.8	30	86.4	0.005	0.04833333333
Amicell 320 4p22s	4	22	2.5	3.7	4.3	55	81.4	94.6	10	28.8	40	115.2	0.005	0.0275
Amprius 410 21p29s	21	29	2.75	3.6	4.39	79.75	104.4	127.31	1.85	1.85	38.85	38.85	0.017	0.02347619048
Amprius 410 32p19s	32	19	2.75	3.6	4.39	52.25	68.4	83.41	1.85	1.85	59.2	59.2	0.017	0.01009375
Amprius 410 29p21s	29	21	2.75	3.6	4.39	57.75	75.6	92.19	1.85	1.85	53.65	53.65	0.017	0.01231034483
Amprius 405 17p24s	17	24	2.75	3.63	4.39	66	87.12	105.36	2.7	2.7	45.9	45.9	0.007	0.009882352941

Although this was helpful for comparative array design, its ability to predict accurate absolute efficiency numbers was lacking, and so a project was needed to combine the strengths of both of these spreadsheets into one tool that could operate with much greater accuracy while retaining as much of the simplicity of these spreadsheets as possible. Thus, the Solar Array Analysis Suite project was launched, hoping to combine all of these calculations into one package that integrates directly with the existing workflow of the Strategy and Array divisions.

Software Suite Overview

The Solar Array Analysis Suite consists of four MATLAB modules and one Unity executable. It operates upon data provided by the operations already performed by our Strategy and Array divisions, enabling easy integration into their current workflow.

The first MATLAB module is the CAD Handler, which takes a STL file generated by the Array division from an array design they have produced in Siemens NX. This module imports the car CAD to MATLAB, analyzes the location of every individual cell, and outputs two files - one containing all of the cell normals for cell-by-cell analysis of the cosine losses, and one containing all of the cell corners for the Unity Shading Simulator.

The Unity executable is the Unity Shading Simulator, which reads in a file with the location of the corners of each cell, generates the best fit rectangle on the best fit plane for each cell, and then determines under which conditions each cell will be shaded. For the time being, it only gives results one subarray at a time; however, with only slight tweaking it should be able to give per-cell shading data for future analysis. This is the only part of this project that I did not code myself - rather, it was coded by Ian Bertram (ianbtr@umich.edu).

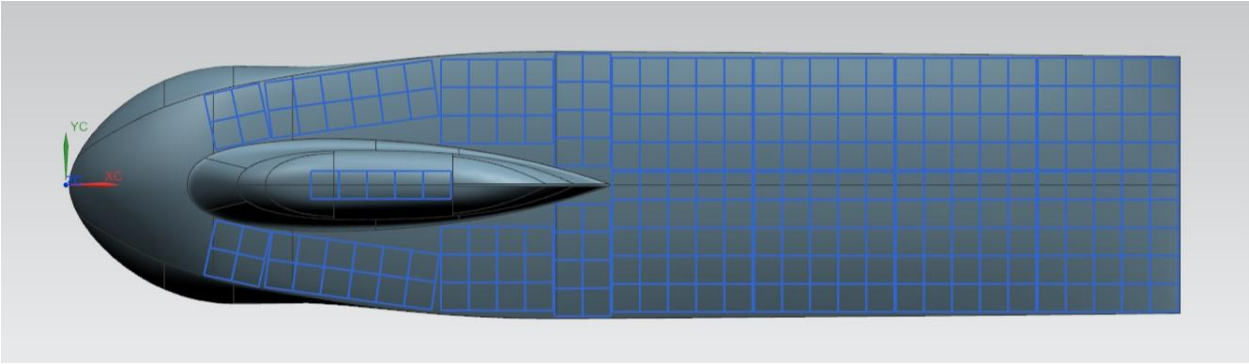
The second MATLAB module is the Shading Data Analyzer, which takes in the data from the Unity Shading Simulator and uses a cubic spline interpolant to combine it with data from a race file produced by the Strategy division to prepare it for final analysis.

The third MATLAB module is the MPPT Data Analyzer, which takes MPPT test data and determines the best 4D hyperplane to fit the data points, allowing accurate prediction of the MPPT efficiency under a given set of conditions. The choice of model parameters is informed via the Akaike Information Criterion (AIC) and the Coefficient of Variation of the Mean Absolute Error (CVMAE).

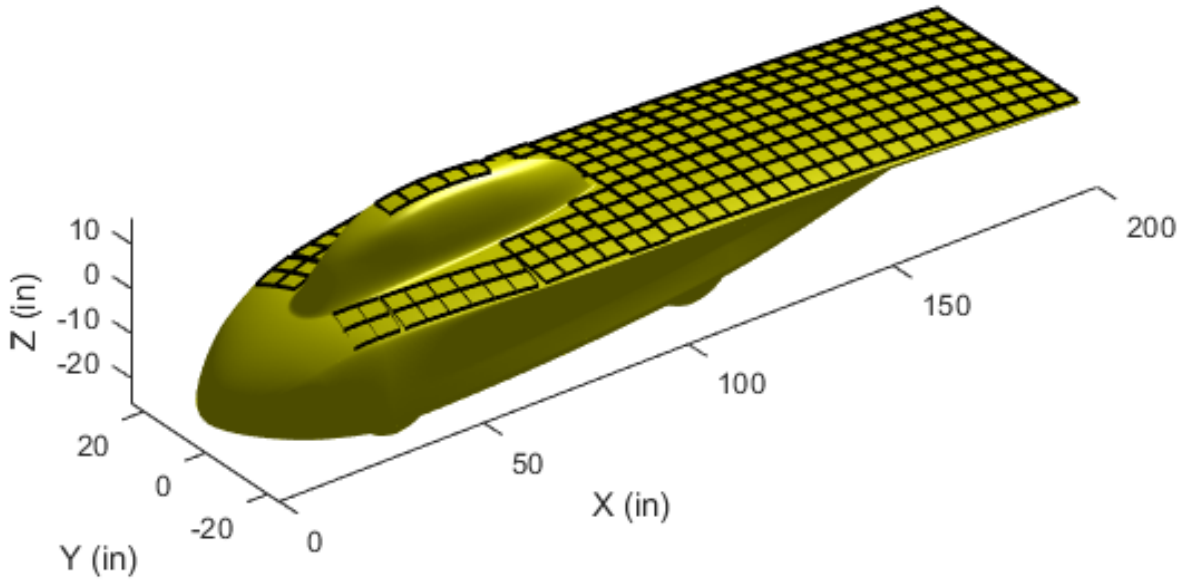
The fourth and final MATLAB module is the Cell Loss Analyzer, which takes the data provided by the other three MATLAB modules and determines the total efficiency of the entire system for the provided race file. Ultimately, this module is what provides the data that will allow us to make more informed array design decisions in the future.

CAD Handler

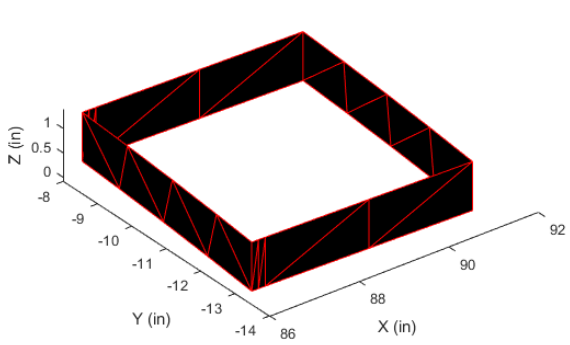
The Array division normally does their design work in Siemens NX, creating an array layout as a series of lines that are then projected onto the aerobody:

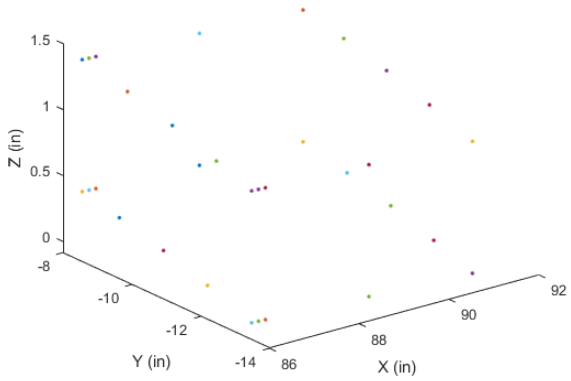


Because these lines have no physical area, it is impossible to act directly upon them; however, by extruding faces upwards by a bit, it is possible to generate STL files with the CAD for both the car's aerobody and each subarray, which can then be imported into MATLAB:

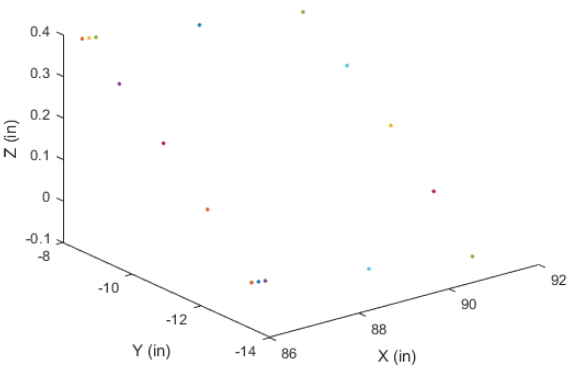


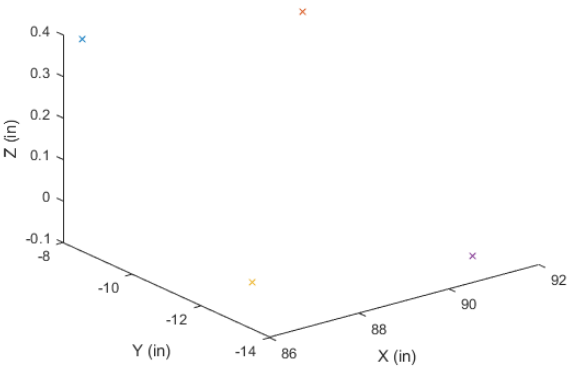
Each face of a STL file is made up of a number of triangles, which, when put together, make up the overall surface of each extruded cell. It is possible to extract the points from the vertices of these individual triangles, as shown in the following images:





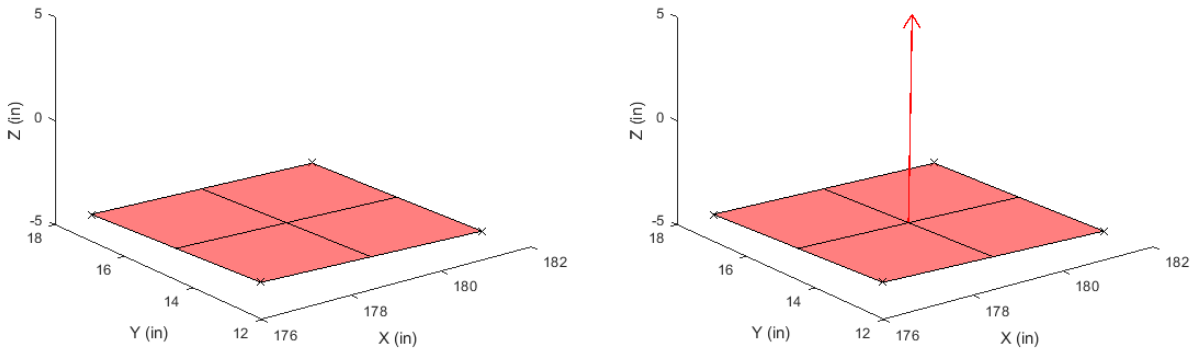
After finding the vertices of each of these triangles, we can discard all of the points whose Z values are above the average Z value of all of the points (provided our extrusion was by enough to ensure none of the points from the top end up below the average, or vice versa). This leaves us with only the points on the bottom edge of the surface, and we can then find the corners by finding the four points with the greatest distance from a point at the center of all of the points on the bottom edge.



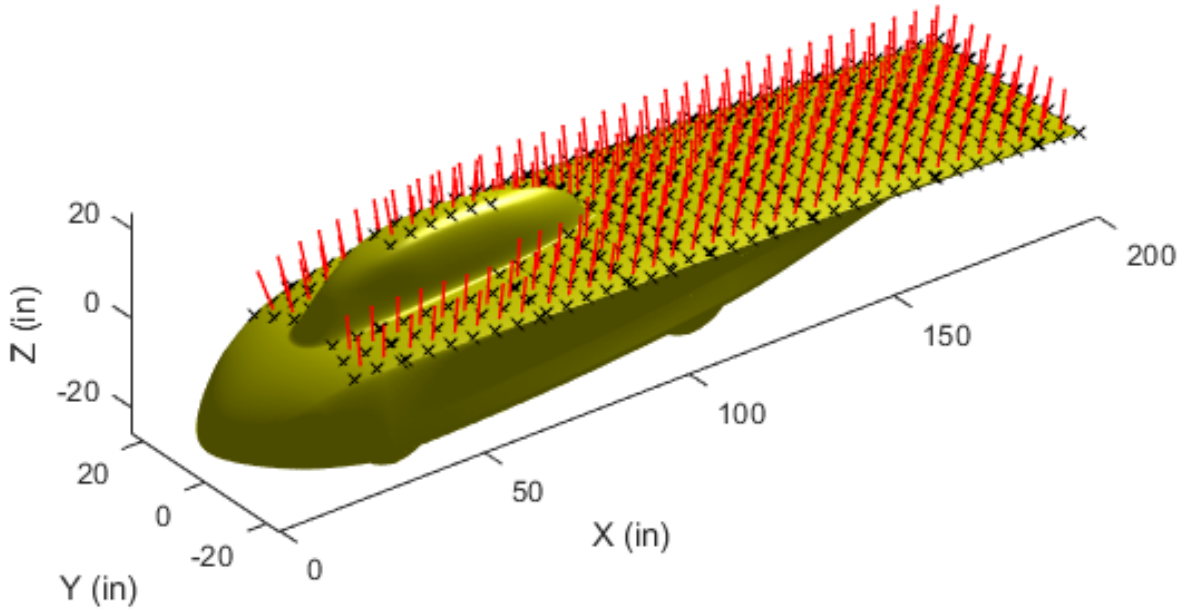


These points are then exported to the file “subarray_rectangles.txt” which is then imported into the Unity Shading Simulator.

We then need to find the plane of best fit through each of these four corner points, which is accomplished by minimizing the sum of the squares of the normal distances to the plane for all four points. From this plane, we can determine the normal vector of the cell. If this process finds a normal vector whose tip is below its tail, we know we are facing the wrong direction and can simply negate all of its values. These normal vectors are then saved to the file “subarray_normals.csv” which is then imported into the Cell Loss Analyzer.



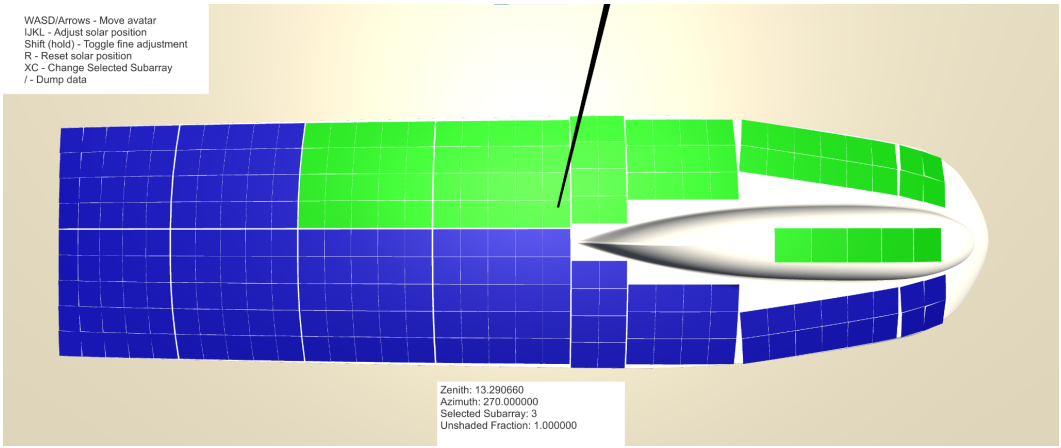
These points and the normal vectors are then re-applied to the original aerobody CAD for a visual check, to ensure the process has been completed successfully.



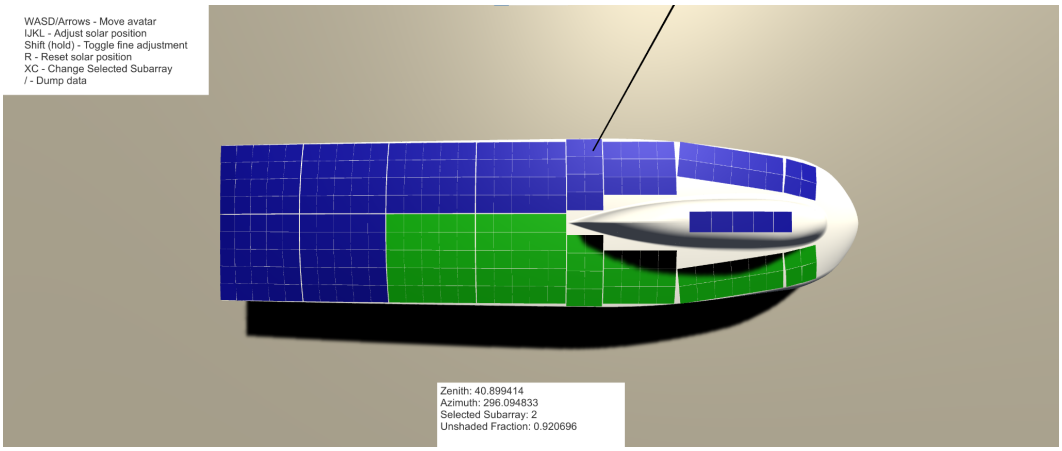
Unity Shading Simulator

The Unity Shading Simulator uses the Unity Real-Time Development Platform to create an interactive environment in which the aerobody and cells of the car may be analyzed in the presence of ray-traced sunlight. The Unity Shading Simulator allows a user to manually manipulate the location of the sun relative to the car to inspect the resulting shade on the vehicle's array, and is also capable of performing an automated sweep and dumping the data to the "subarray_shading_table_X.csv" files, which are imported into the Shading Data Analyzer.

The Unity Shading Simulator must be provided with the CAD of the aerobody as "car.obj" and the points of the vertices of each array cell in "subarray_rectangles.txt" in order to function properly. It then does a very similar plane fit to that of the CAD Handler, except after finding this plane it then finds the rectangle of best fit in the plane for the four corner points. It then generates a flat surface over this rectangle, which it registers as active solar cell area. All of these surfaces are then added to the model, grouped together by subarray.

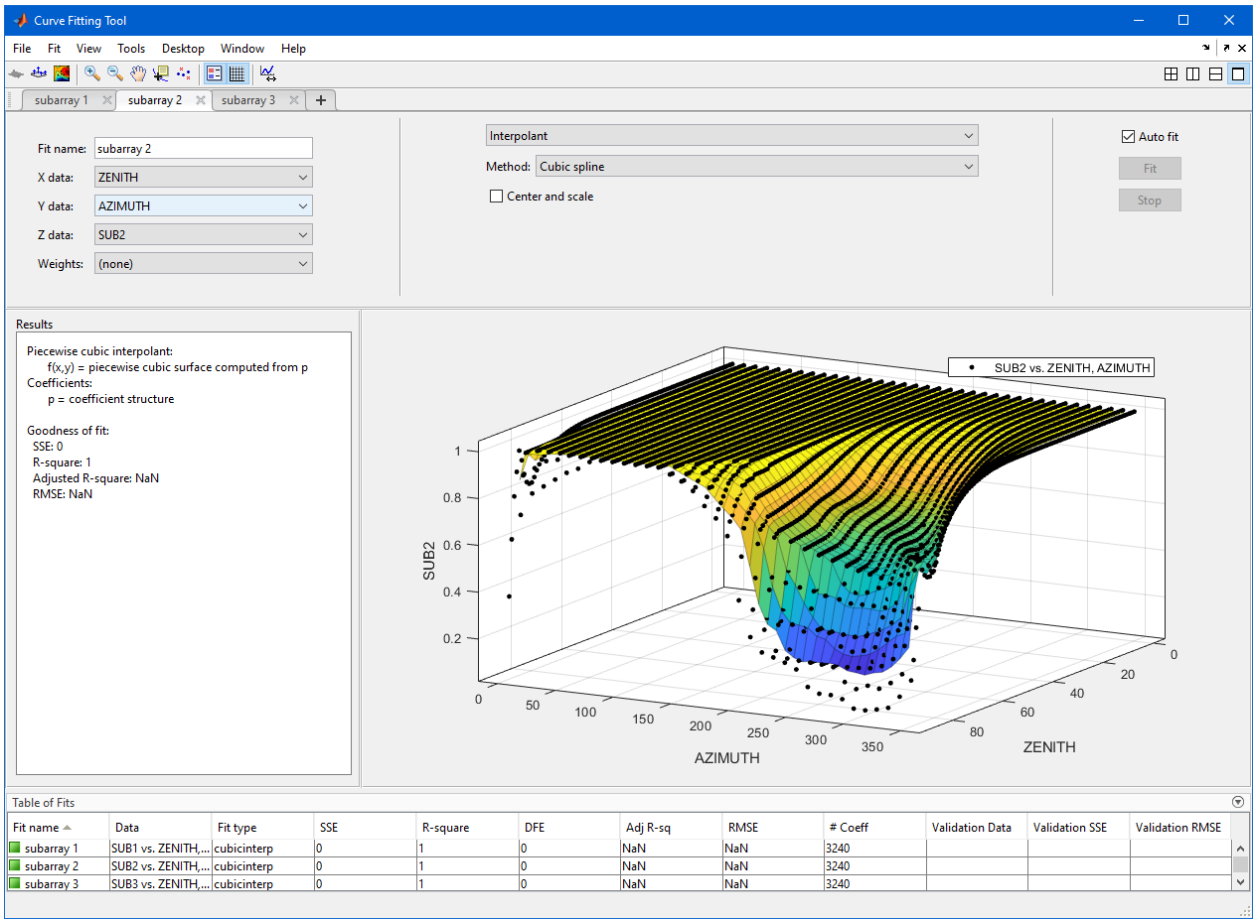


As the sun is moved by the user, the shadow that will be cast onto the car for a given relative zenith and azimuth angle can be seen in real-time.

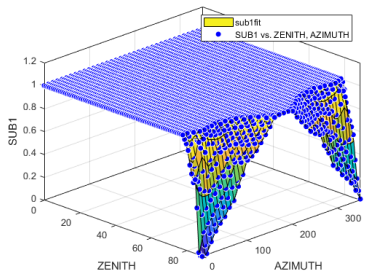


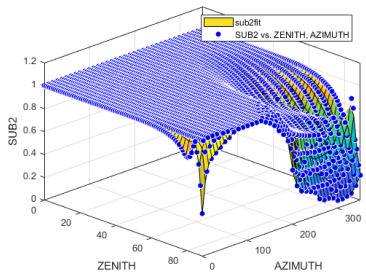
Shading Data Analyzer

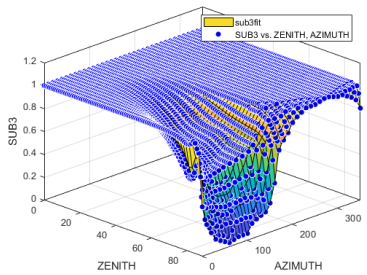
The data output by an automated sweep in the Unity Shading Simulator is then imported into the Shading Data Analyzer module, where the createShadingFits script loads all of the data from the sweep into MATLAB and enables easy usage of the MATLAB Curve Fitting Tool.



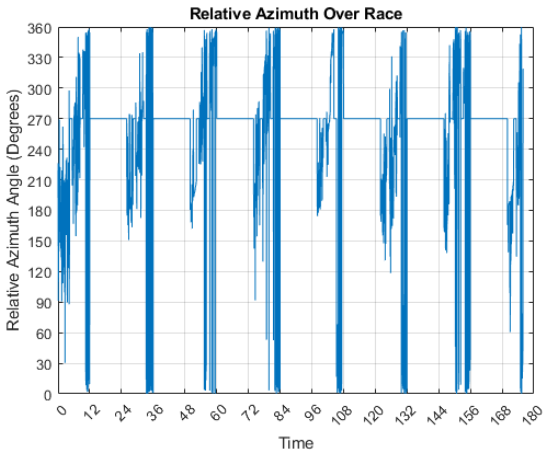
Code is then auto-generated based on these cubic spline interpolants, providing the ability to plug a set of zenith and azimuth values into the generated function and get back a predicted fraction of each subarray that would be shaded under those conditions.

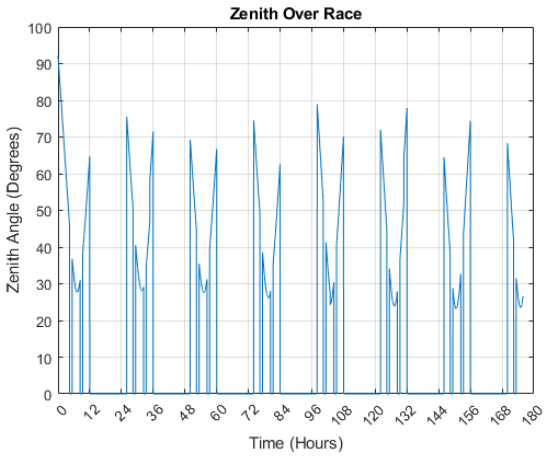




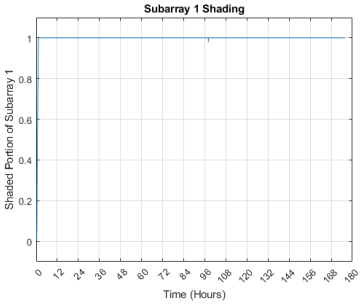


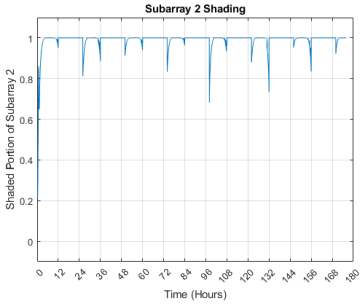
The script then takes data from the race file provided by the Strategy division and calculates the zenith and azimuth angles the sun will be at relative to the angle of the car over the course of the race, with time measured in hours since the start of the race:

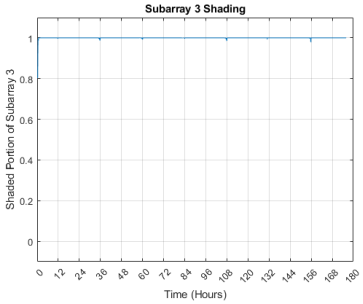




This then allows us to simply plug the relative azimuth and zenith values for each point in time into the cubic spline interpolants we developed earlier, and get the resulting shaded fractions of each array as a function of time:



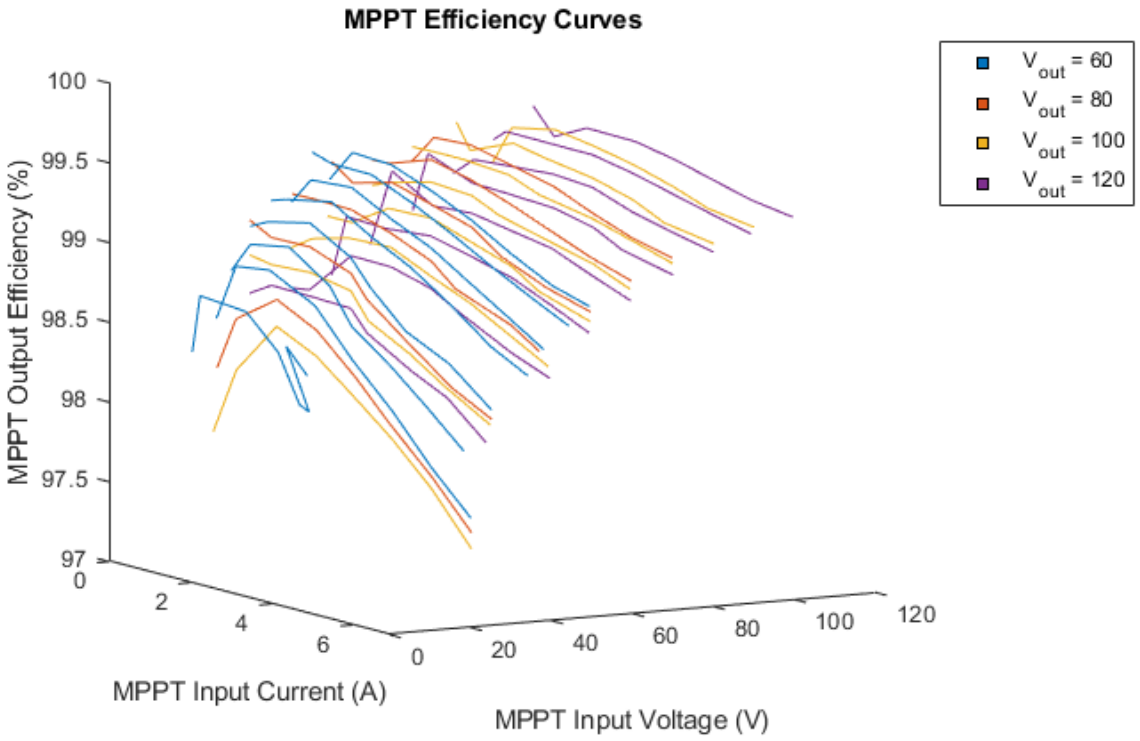




In the future, it is planned to modify the Unity Shading Simulator to provide shading data for every individual cell rather than just the entire subarray. So long as it is possible to automate the curve fit performed by the Curve Fitting Tool, this would allow array shading to be analyzed on a per-cell basis in the same manner as cosine losses from the relative angle of the sun to each cell, which could enable us to get much more accurate absolute shading losses in the Cell Loss Analyzer. For now, however, a per-subarray analysis is still sufficient for comparative purposes, and provides accurate enough data where we will be better able to predict the real-world performance of the array with the tool than with our old spreadsheets.

MPPT Data Analyzer

In order to create a good model for the efficiency of our MPPTs, we first need to methodically test them under a variety of conditions. Once we have gathered this data we can plot it:

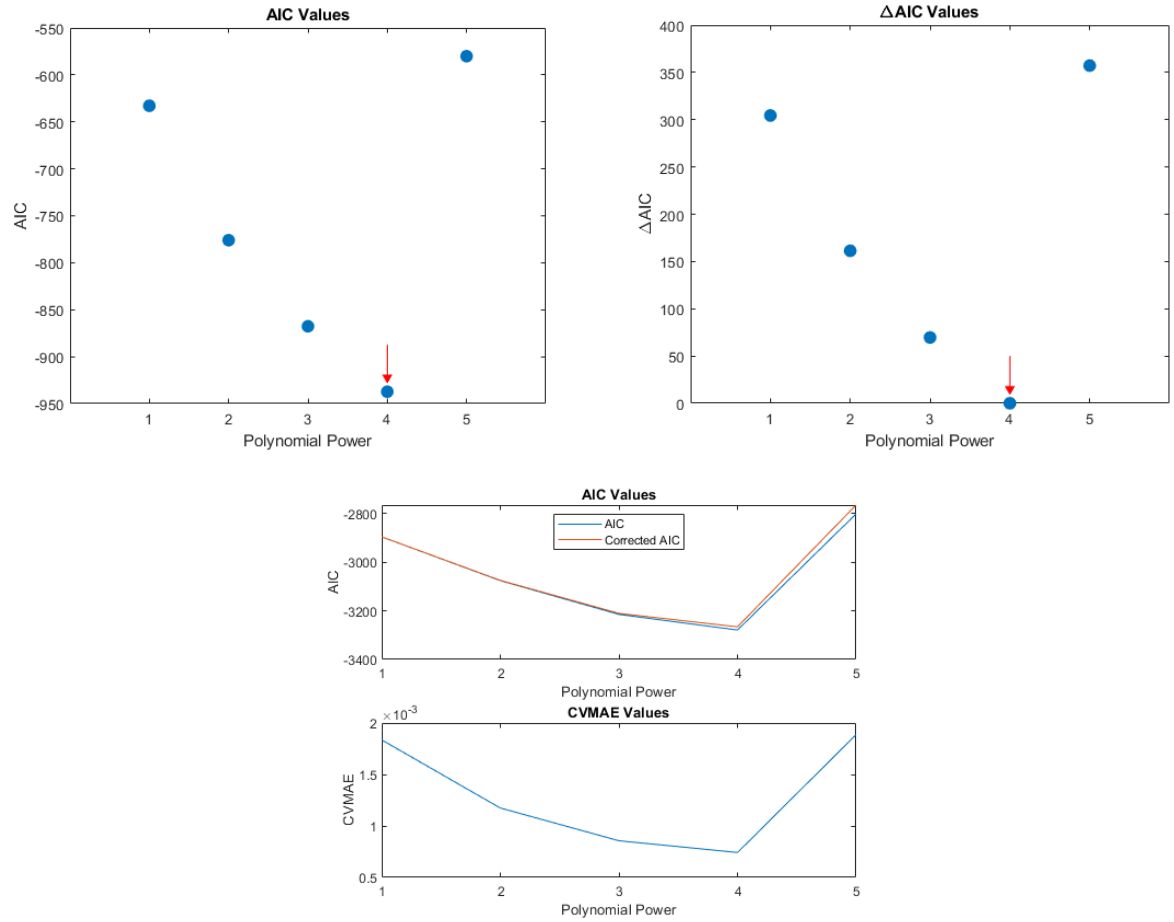


As can be seen, the efficiency of the MPPTs depends on the input current, the input voltage, and also the output voltage (represented by the various colors on the chart - note how each color makes its own 3D curve). Thus, in order to facilitate accurate prediction of the efficiency of each MPPT, we need to use a 4D hyperplane fit to these data points.

However, in order to accomplish this, we must first consider that it is very difficult to determine current in without the aforementioned per-cell shading data, and therefore hard to determine the expected voltage value at the subarray's maximum power point. However, it is very easy to apply our efficiencies to the total power available to the array and use that to determine the expected power in to the MPPT. Fortunately, the voltage-current product law tells us that the current into the MPPT will simply be the power in divided by the voltage in, and so it is trivial to estimate the current in so long as we have a value for the voltage of the subarray. In this case, with no better estimate available, we simply assume that each cell in the subarray is at the maximum power point voltage (V_{mpp}) listed on the specification sheet for the cells we are using. When per-cell shading data output is added to the Unity Shading Simulator, we can use test data of the IV curves of our cells to predict a V_{mpp} from our determined current, but for now this will have to do.

As such, our hypersurface fit is currently configured to just take a power in value rather than convert every power in value to a predicted current in value. Thus, we use the Curve_Fit_Power_Picker script to aid in selecting the best hypersurface model.

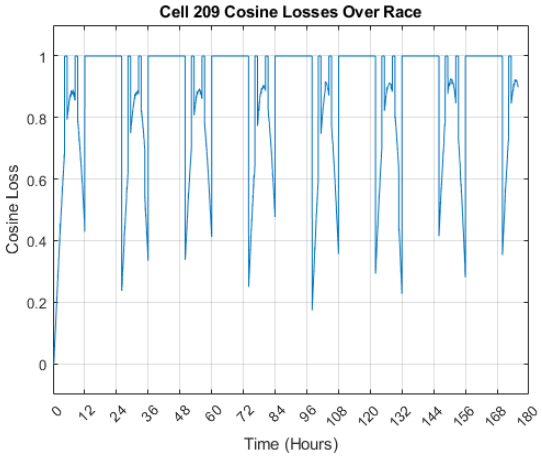
In particular, we use the Akaike Information Criterion and attempt to select the model with the lowest (most negative) AIC value. Should the sample size be determined to be too small, the program will instead recommend using the corrected AIC, which further penalizes additional parameters to help prevent overfitting. The user can then use the Coefficient of Variation of the Mean Absolute Error as calculated by the hyperplane fit function to verify the choice made by the AIC with another method of model selection.

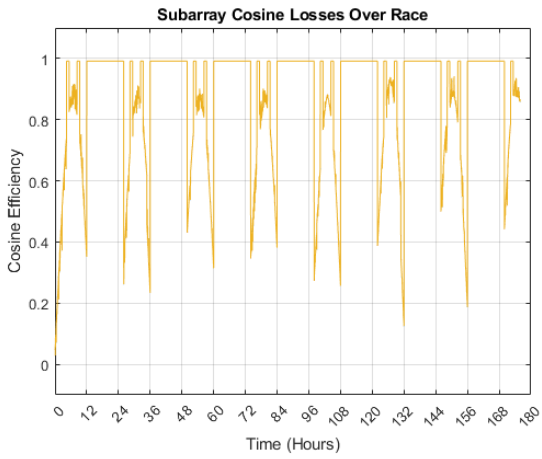


Care must be taken while analyzing the potential models to ensure that there is no rank deficiency in the hyperplane fit function. Should this occur, it points to unidentifiability in the model for the given number of parameters, and so while the results should still be reasonably close, one or more parameters are effectively arbitrary and may skew the results in a way that a model with a lower power (and therefore fewer parameters) may not. Furthermore, the R^2 value can be inspected by hand to ensure a good fit.

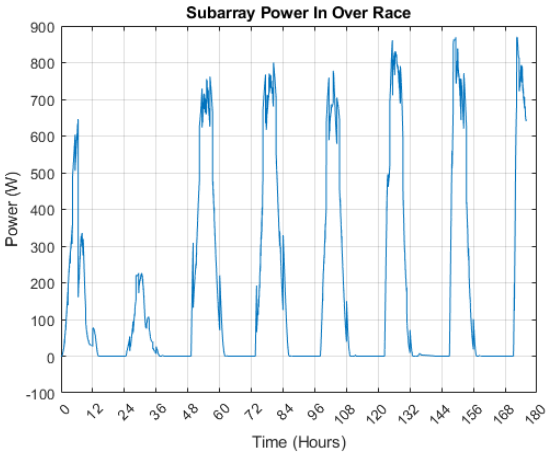
Cell Loss Analyzer

The final script, cellLosses, takes the output from the CAD Handler and the Shading Data Analyzer as well as the function generated by the MPPT Data Analyzer and uses all of this data to estimate the practical performance of the array over the course of our race. First, every cell's normal vector compared to the direction of the sun and only the component of the sunlight that is perpendicular to the cell is considered to be absorbed, with the remainder lost to what is known as "cosine losses". A single cell (any particular cell can be chosen) and all three subarrays are plotted for visual inspection.



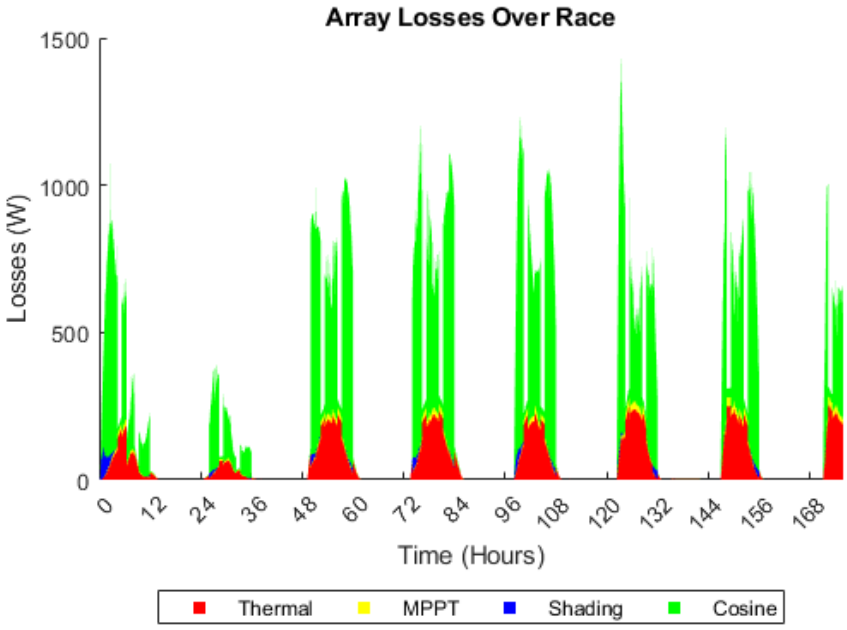


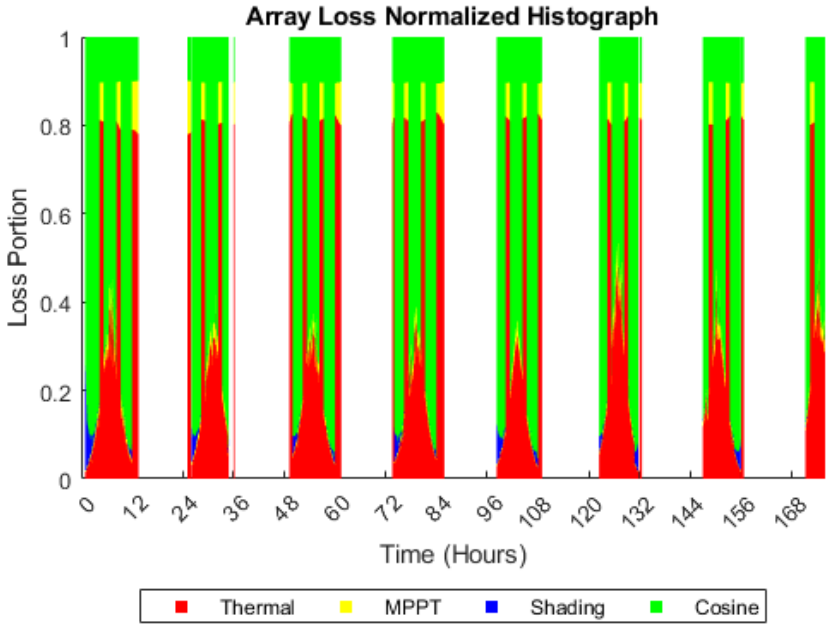
The shading data as calculated in the Shading Data Analyzer and the thermal efficiency as calculated in the original loss factor estimate spreadsheet are then multiplied by the cosine efficiency, providing data for the total subarray losses over the course of the race. The irradiance data from the race file is used to determine how much light will be in the atmosphere at each point of the race, and the total amount of light energy available to each subarray is calculated based on their area. Multiplying this by the total subarray efficiency and the base cell efficiency gives the total power in that is expected over the course of the race.



This allows us to calculate the total system efficiency, which is simply the integral of the aforementioned power in curve divided by the integral of the irradiance curve - in other words, it is the total amount of energy delivered to the car's battery divided by the total amount of energy ever available to the car's array. This number is simply printed out to the user in the terminal - in the case of the array I've been using for my initial round of testing, this number is 18.61525559%.

Lastly, in order to help visualize the losses incurred by the array system, I included the following two plots, which help to break down the power losses experienced by the car into their component categories in a way that is easy to understand.





Preliminary Results

Thus far, the members of the Array division have been trained in the usage of this tool, and used it to create the final array layout for our upcoming 2022 vehicle, as it has allowed them to directly compare several potential array layouts with even slight changes and determine which will perform best on the car. Even fractions of a percentage point can make a massive difference in a highly efficient solar-powered vehicle over a multi-day race, and this tool will enable us to make the right decisions in less time to maximize car efficiency. I cannot wait to see what this tool will do when used from the very beginning of a cycle to analyze not only small changes in cell placement, but also radically different car shapes and sizes.

Future Improvements

There are several aspects of this software suite that can still be improved upon by future work.

Currently, the thermal efficiency calculations are still a weighted average of the temperature of the array over the course of a race, and so it does not take into account the temperature increase when the car is not moving (due to the lack of wind over the array), such as when it is pointing its array at the beginning and end of each day. As such, it is likely underpredicting the energy that will be lost to thermal inefficiency during these times.

It would be much more accurate to calculate the current from each subarray in order to capture the effect of shading cells on the rest of the subarray (essentially limiting a subarray to its worst-performing cell). This will require re-tooling the MPPT fit to work on current rather than power (which is easy) and then dropping the current linearly from I_{mpp} to 0 as different percentages of the cell are shaded. In order to do this, however, the Unity Shading Simulator needs to support outputting shading data on a per-cell basis. Then we need to add some sort of method to segment the array so we can consider bypass diodes, and we will also need to add some way to automate generating shading curve fits (it is unlikely that anyone would manually generate a curve fit for each individual cell - that's just a waste of time).

In an ideal world, this is converted from four MATLAB modules and one Unity executable into a single file, perhaps with a GUI for ease of use. Right now, files must be manually copied between modules, and it would be much faster if everything was automated and controlled from one single place.

In the extreme long term, it would be ideal to also model the rest of the electrical systems in the car, turning this from an array-focused analysis tool into a system-wide electrical analysis tool, removing its dependence on the strategy division's other tools and race files and allowing the software packages to work in parallel to essentially verify each others' results.

Special Thanks

Ian Bertram - Unity programming and design, data gathering

Ibrahim Syed - Projected race file creation

Dominik Kerschbaum - Siemens NX array design

Gabriella Teodoru - Siemens NX array design

Bennett Mejia - Siemens NX array design

Samantha Romano - Siemens NX aerobody design lead

UMSCT Aerodynamics Division - Siemens NX aerobody design

UMSCT Strategy Division - Data gathering, race file assistance

Jeff Dunworth - Assistance with mathematical modeling

Connor Boermann - Consulting

Amal Bansode - Consulting

Andrew Olejniczak - Consulting

Rishabh Goel - Consulting