Solar Tracking System with RasberryPi and Python

Lodovico Griccioli and Ethan Kern

June 13, 2018

Software

- PiMotor library from SB components which has been specially modified to work with these motors and for enhanced precision (included in project file).

Hardware

- SB components motor shield (sbcshop.com)
- 2 Nema 17 12V 0.4 (stepperonline.com)
- 3000mAH 12v TalentCell battery pack (amazon.com)
- 12V 3W 250mA SunnyTech solar cell (amazon.com)
- 3" Lazy Susan bearing (amazon.com)
- DIYmall DC0-25V voltage divider (amazon.com)
- Adafruit ADS-1015 12-bit ADC (class hardware)
- Custom wood and MDF frame

Description

Our project uses two stepper motors to pitch and rotate a solar cell so that it can be aimed automatically in the direction that generates the highest power possible. Then, after some time it automatically readjusts in case its environment has changed, and re-finds the orientation of maximum power. The voltage drop generated across the solar panel is directly used as feedback to determine where to point.

The script tracks power generation and motor movement by storing voltage readings from the ADC in a numpy array, and modifying a step-counting variable with every step taken by either of the motors. The voltage reading from the solar cell is reduced by a factor of 5 by the DC0 voltage divider as to not exceed the 3.3V limit of the Raspberry Pi, but this is accounted for in the code.

First the program will run an initial scan. The solar cell will be pitched down from zenith to 45°, then it will perform a whole rotation in the horizontal plane while measuring and storing the solar cell voltage every step (3.6°). By finding the index of the highest voltage in this array, we know how many steps from start the rotating motor took to get there (+1, accounting for off-by-one error). A step-counting variable is modified every time the motor takes a step, so the difference between this total and the index is how many steps the motor needs to move backward. The stepper motor is told the number of steps to take to return to this best position. After it stops at the optimum rotation angle, it needs to scan through the possible pitch angles to find the final orientation that produces the maximum power. The solar cell is pitched from 45° to zenith, taking measurements every 3.6°, and moving to the best pitch angle by the same method as described before. If we assume that all the incoming light from the sun is parallel to itself this should be sufficient to aim the solar cell perpendicular to the light when it initially orients itself.

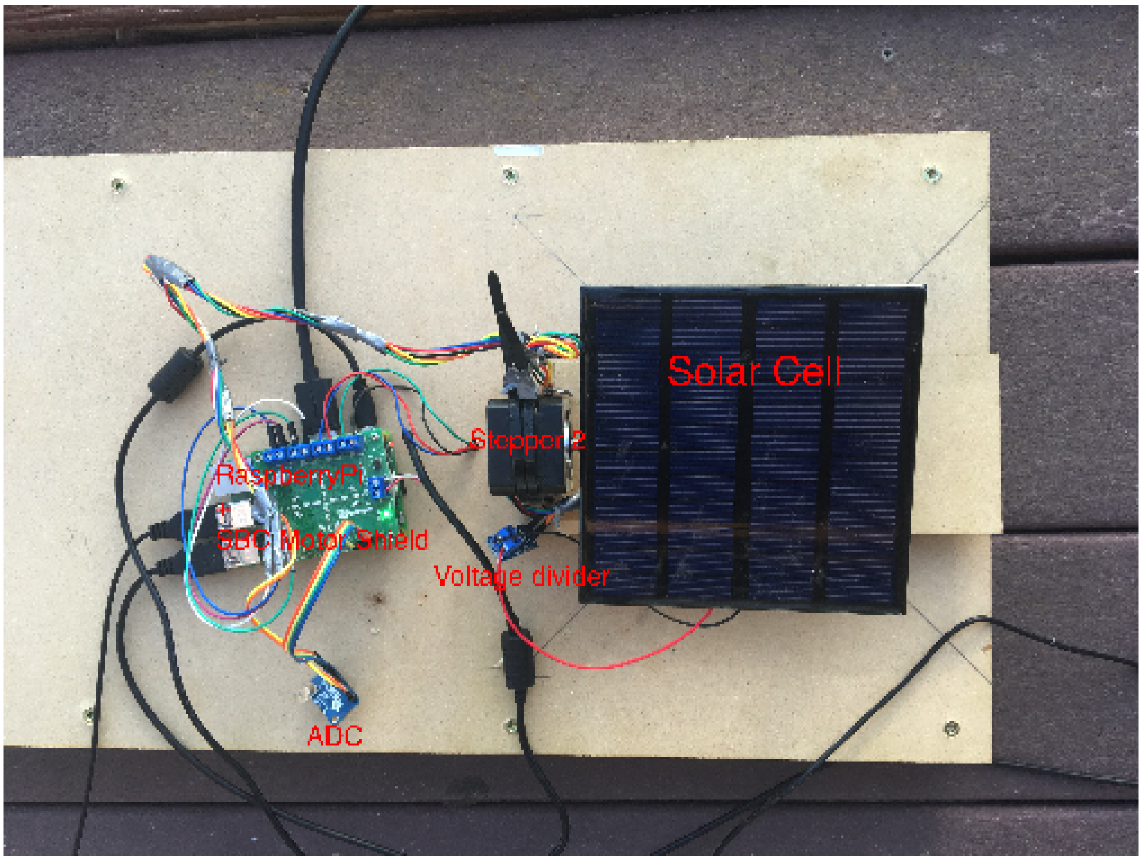

Figure 1: Aerial View of Hardware. Stepper 2 controls the pitch of the solar cell. The voltage divider reduces the input to the ADC (and Raspberry Pi) by a factor of 5. Given more time, everything would be encased and the wires would be redone and run underneath the board.

After the solar cell has found the best initial position, it stays there and prints its voltage drop to standard output periodically. We found an appropriate time to keep it stationary to be between thirty and sixty minutes. External light sources should not have changed much over this time period, and constantly trying to readjust would be a waste of power. For the purposes of our demo video, we made this stationary time sixty seconds.

After an appropriate amount of time, the the solar cell readjusts its position by scanning over its current general area. It scans through 30° of rotation in both directions and moves to the best rotation angle, and scans through 15° of pitch to move to the new best position. If it is already pitched all the way down to the ground, our code stops it from pitching itself down into the ground and damaging itself. The script is designed to keep running throughout the day in an infinite loop, readjusting itself and following the sun as a result.

Instructions

To run the hardware, the PiMotor.py library and run.py script must be downloaded in addition to the other libraries already present on our Physics 129L Raspberry Pis. The solar panel must be oriented straight up at the zenith, and then 12V of power given to the motors. Simply run the run.py script and the solar cell will find the optimum position, and adjust itself periodically in an infinite loop. This can be ended by interrupting the script.

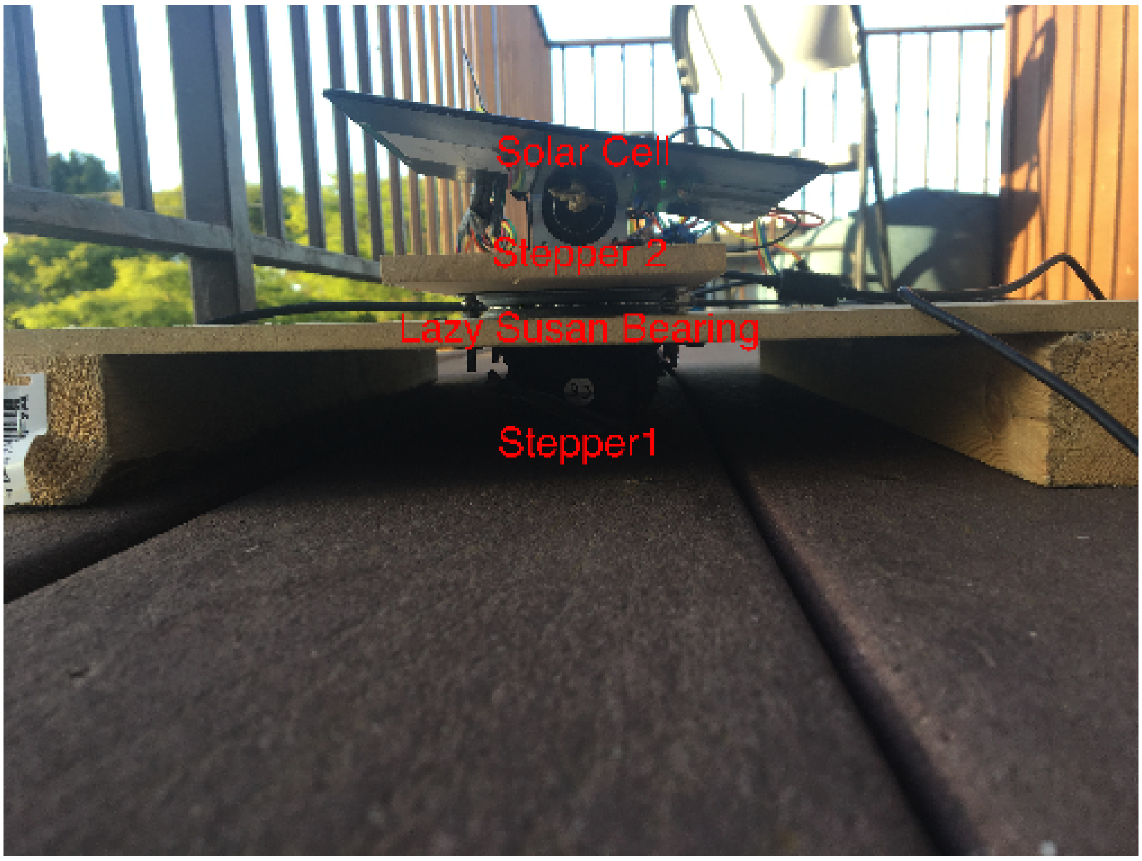

Figure 2: Horizontal View of Hardware. Stepper 1 controls the rotation of the solar cell. It directly rotates a flat bearing that the solar cell and Stepper 2 are fixed to the top of.

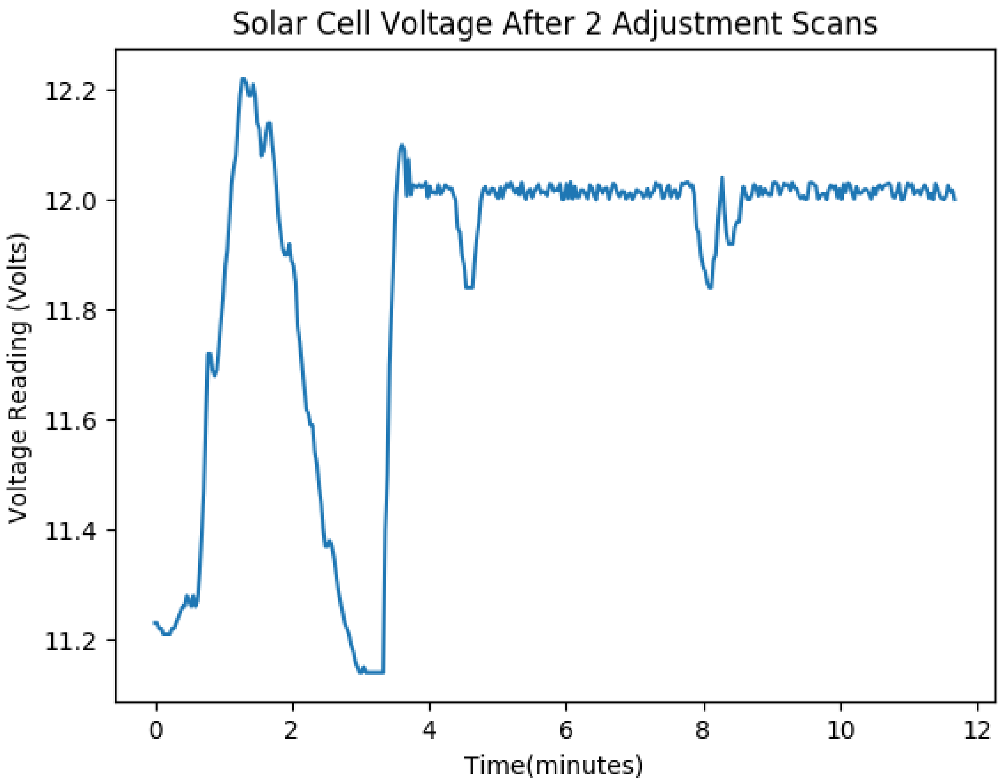

Figure 3: Voltage of solar cell read from initial scan, and through 2 automatic adjustment scans. The solar cell returns to a similar power level every time it scans its immediate area. We can see that the longer the program is running, the more efficient it becomes, as the cell will spend more of the overall time at the position of maximum power. The frequency it adjusts itself could also be greatly shortened, as external light sources shouldn't change much over a few minutes.

Results

The standard output in the video has been saved to the text file output.txt. Our project worked exactly as expected to within a reasonable accuracy. Due to some imperfections in soldering, a few of our voltage readings dropped to about zero, but this only resulted in the loss of a few data points out of 100. Their values are close together enough that this didn't affect performance, and given more time, these imperfections in the wiring would be fixed. There were slight changes in the voltage readings while stationary, but after an adjustment scan, the solar cell still returned to face exactly toward the direction of the sun. Although it did not return to exactly the same position as the initial (ideally best) position, it returned to a position generating the same power to within a third of a Volt. This level of accuracy is already beyond what this project was expected to have, as it already competently tracks the sun by directly using the amount of power generated.

Credits

- Ethan Kern: Writing run.py script, writing LaTeX, modifying PiMotor.py library
- Lodo Griccioli: Constructing hardware and wiring, modifying PiMotor.py library, writing LaTeX