Adaptive Solar Tree

Ahmed Syalabi Seet
Institute of System Sciences, National University of Singapore, Singapore
syalabi.seet@hotmail.com

***Abstract*—Modern solar panel systems have achieved full solar tracking capabilities but lacks the ability to handle power loss due to shadowing. In this work, we present a novel and intuitive method of increasing solar outputs by leveraging reinforcement learning and a 5-DOF robotic arm manipulator design. We utilized in-game cameras to emulate shadow ratios of solar modules derived from current-voltage sensor readings. The reinforcement agent was trained in a simulated environment equipped with geographically-accurate sun positioning and shadow casting using ray-tracing. Our approach showed promising results in simplifying and integrating solar tracking and shadow detection. Project page: https://github.com/syalabi-seet/adaptive-solar-tree**

***Index Terms*—Reinforcement Learning, 3D Simulation, Solar Panels, Shadow Detection, Renewable Energy**

I. Introduction

Solar panels have been around for decades but to date, has not been deployed as hoped for. In the context of Singapore, some of those reasons are cloud cover, aesthetics, and space constraints [1]. While complete cloud cover cannot be avoided for such a cloudy climate like in Singapore, partial cover, bad aesthetics and space constraints could be alleviated by design. As such, solar tree designs have been in the rise due to its simple design and pleasing aesthetics but was expensive [2]. This is due to the structures usually being built as large metal structures. Similar to standard solar panel farms, their orientations are fixed, meaning that their solar panels are oriented in a direction that would be most optimal throughout the year taking all 365 days into account. While this reduces the overall power harnessed per day, on top of that, potential shadow loss due to cloud cover or canopy cover, could further reduce the power outputs from the solar panels. This approach was motivated by the natural mechanism called etiolation which is seen in sunflowers where plants grow towards sunlight to increase survivability [4].

II. Preliminaries

A. Solar Modelling

Tracking Frequencies

1) Horizontal (surface tilt = 0, surface azimuth = 0)
2) Every 24 hours
3) Every 4 hours
4) Every 2 hours
5) Every 1 hour
6) Every 1/2 hour
7) Every minute

Using the optimal angle algorithm stated in [3], the various tracking frequencies were compared over 5 years (2022-2027) to reduce variance through time. It can be seen in Figure 1 that tracking frequency does not affect diffuse irradiance, which is the surface irradiance generated through the ground reflection and water molecule refraction, and can be considered constant.

Algorithm 1 Optimal angle algorithm

Require: latitude, longitude, date, time
for $minute = 1, 2, ..., 1440$ **do**
  Calculate solar altitude, solar azimuth
end for
$irrad_{max} = 0$
$angle_{max} = 0$
for $angle = 0, 5, ..., 90$ **do**
  for $timestep = 1, 2, ..., 59$ **do**
    Calculate irradiance, insolation
    if $irrad_{angle} > irrad_{max}$ **then**
      $irrad_{max} = irrad_{angle}$
      $angle_{max} = angle$

Post-tracking is the pre-computing of the minutely solar geometric information to obtain the optimal angles between each tracking step. Insolation is a measure of solar energy that is incident on a specified area over a period of time. Post-tracking improves insolation as tracking frequency increases, this can be seen in cases 4 and 5 in Figure 2. As for cases 10 and 11, tracking every half hour generates similar results to without post-tracking.

As post-tracking is a trial-and-error approach of obtaining the optimal angles, despite being the standard in most tracking systems, it is inefficient with inefficiencies proportional to the degree of accuracy. For example, in Step 2 in the algorithm, for minutely-accurate geometric data (solar altitude and azimuth), this step alone has to be computed 1440 times per day.

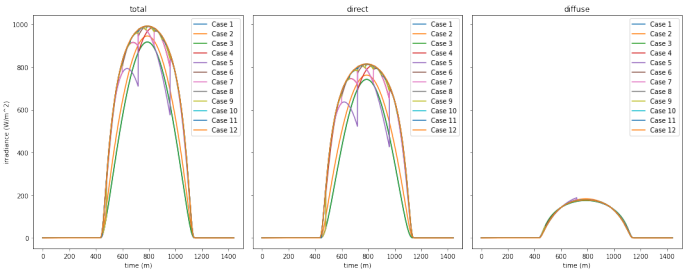

Fig. 1. Irradiance over time: *(i) total irradiance, (ii) direct irradiance, (iii) diffuse irradiance for different tracking frequencies*

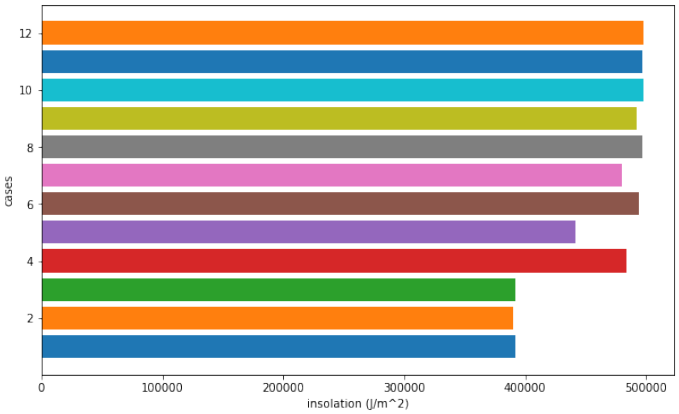

Fig. 2. Average monthly insolation: *the total insolation per square area averaged monthly for different tracking frequencies from January 2022 to January 2027*

B. Shadow ratio

The hurdle for any reinforcement learning environment is bridging the simulation to the actual environment and vice versa. It would be trivial to train a robotic arm manipulator while providing all global positional vectors of all its limbs and the position of the sun but this is not the case in the actual environment. We had to find a means to describe the solar state at any given time with minimal sensory.

In [5], there was an attempt to use visual observations using RGB cameras to track the solar positions of the sun. While successful to a certain extent, this method was found to be energy-intensive and without solar shielding on the camera, the direct solar rays could damage its camera lens over time.

Piezoelectric sensors or light sensors was also inadequate as, on top of requiring additional power to operate, these sensors did not provide necessary information as to the shadows being cast onto the solar panels.

Therefore, we decided to use the solar panels itself by coupling them with current-voltage sensors to obtain the power outputs. These solar panels were fitted in parallel and its power outputs measured individually. As the solar intensity is relative to the time of day and also the environment conditions, the maximum output of the panels was taken as the basis of the derivation.

Assuming an array E_{array} consisting n-elements representing the electrical power outputs $(P = IV)$ of each solar panel with negligible differences in electrical efficiency (η_{cell}), and that the solar irradiation (H) acting on all panels are equal given the close distance between.

$$E_{array} = \begin{bmatrix} E_1 & E_2 & E_3 \\ E_4 & E_5 & E_6 \end{bmatrix} \tag{1}$$

$$\begin{aligned} E_{solar} &= H \cdot S_{module} \cdot A_{module} \\ &= max(E_{array}) \cdot S_{module} \end{aligned} \tag{2}$$

$$\begin{aligned} E_{module} &= \eta_{cell} \cdot E_{solar} \\ &= \eta_{cell} \cdot max(E_{array}) \cdot S_{module} \end{aligned} \tag{3}$$

$$S_{module} = \frac{E_{module}}{max(E_{array})} \tag{4}$$

where S_{module} is the shadow ratio of the module, P_{module} is the power output reading of the module, H is the solar irradiation per unit surface area, A_{module} is the surface area of the module, E_{module} is the electrical energy generated by the module, E_{solar} is the solar energy generated by the module, η_{cell} is the electrical efficiency of the module [7].

The shadow ratio of each module can be derived as the ratio between the module's electrical output and the output of the maximum.

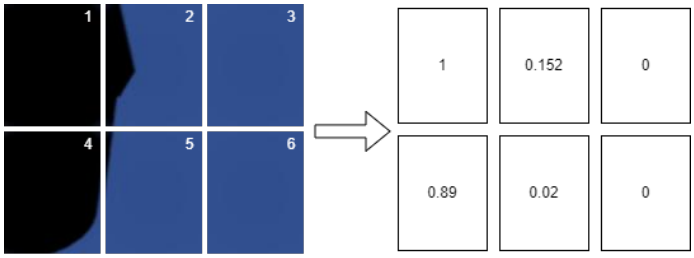

Fig. 3. 6-segment solar panel array: *(i) left shows shadow maps generated from the simulated environment, (ii) right shows the shadow ratio values of each panel*

III. Environment Design

A. Environment

TABLE I
Environment Parameters

Parameter	Value
Randomize	True
Shadow ratio limit	0.1
Incidence angle limit	15
Joint speed	300
Joint stiffness	100,000
Joint damping	100
Number of obstacles	15
Number of panels	6

a) Sun Controller: To emulate a geographically-accurate sun in the simulation, the PSA solar position algorithm was developed in C# based of their provided C++ codes [7]. To speed up training time, the sunrise and sunset timings were calculated using NOAA sunrise/sunset formula [8], simulation times were then randomized between solar hours only at the start of each episode.

b) Spawn Manager: To instantiate and randomize the shape, distance and rotational position of the obstacle around the tree module to partially block the solar panels from the sun.

c) Environment Controller: Equatorial locations [9] between latitudes $-30°$ and $30°$ and longitudes $-160°$ and $160°$ were chosen at random to initialize the simulation location at the start of each episode.

d) Module Controller: To control the joint states of the robotic arm and compute the shadow ratios of the end effector

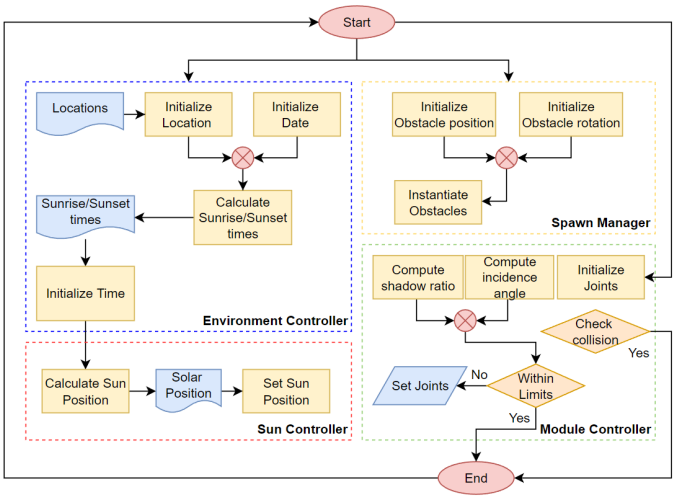

Fig. 4. Episodic flow

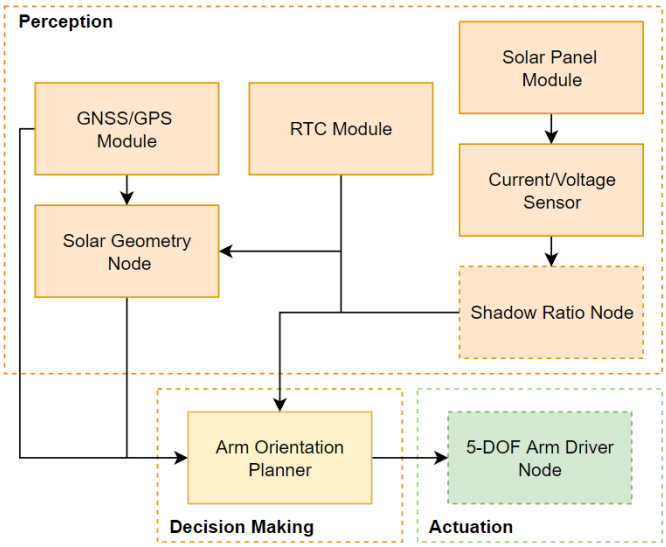

Fig. 5. System Architecture

B. Module

a) Robotic arm: A 5-DOF robotic arm design was inherited from the RX160 industrial robot designed by Stäubli Robotics [10], the robot is equipped with 5 revolute joints.

b) Shadow ratio sensors: To emulate shadow ratios on each panel, in-game cameras were panned onto each panel surface. The P_{rgb} pixels of the camera output were then extracted and converted into weighted grayscale P_{gray} [11]. The pixels with zero magnitude (N_{black}) were counted and divided over the total number of pixels (N_{total}) to obtain the shadow ratio.

$$P_{gray} = 0.299P_{rgb,r} + 0.587P_{rgb,g} + 0.114P_{rgb,b} \quad (5)$$

$$\begin{aligned} N_{total} &= \sum_{i=1}^{x}\sum_{j=1}^{y} P(i,j)_{gray} \\ N_{black} &= \sum_{i=1}^{x}\sum_{j=1}^{y} (P(i,j)_{gray} = 0) \\ S_{module} &= \frac{N_{black}}{N_{total}} \end{aligned} \quad (6)$$

where x is the width of the camera output, y is the height of the camera output, P is the pixel magnitude.

c) Incidence angle sensors: The incidence angles of each solar panel were calculated using its relative rotational vectors and solar positional vectors. These are only used as reward signals and are not known to the agent during training.

C. Agent

a) Observation states: 15 curated elements representing, (i) the location's latitude and longitude, (ii) solar altitude and azimuth, (iii) all panels' shadow ratios and (iv) all motor angle states in degrees

b) Action space: 5 continuous values representing each motor rotational target in degrees

c) Reward: (i) $+1$ when all solar panels are within incidence angle limit of $< 15°$ and shadow ratio limit of < 0.1, (ii) -0.001 for every timestep, (iii) -1 when there is collision

To keep the observation states simple, feature engineering was performed to transform original states like location's latitude and longitude and the current time and date to higher value information like solar altitude and azimuth.

These states could then be obtained using electronic modules like a GNSS/GPS module for the latitude and longitude and an RTC module to obtain the current time and date.

Designing the reward function was the most difficult as it was a trial-and-error effort. A sparse reward structure worked best where the agent was only rewarded when the goal was met. Sub-goals also tend to mislead the agent to a sub-optimal state.

As the module did not have any angular limits in-place for collision avoidance, the agent was made to learnt its own range of motion during training by penalizing the agent with a -1 reward when any collision occurred.

During training, the episode ends when the goal has been achieved by the agent to reduce repetitions in the agent buffers and also reduce training time. Episodes were also designed to last for only 3000 steps which translates to a minute in game time to motivate the agent to reach its target before the episode ends.

IV. Experiments

The agent was trained with the Proximal Policy Optimization (PPO) algorithm [6] using the Unity MLAgents toolkit. The purpose was to test the train-ability of the agent using minimal observation states.

A total of 12 agents were trained simultaneously for 1,700,000 steps concluding to about 0.995 in average reward per episode and an average episode length of 3.87 steps. There were some episodes where the randomization of the

TABLE II
HYPERPARAMETERS

Hyperparameter	Value
Batch size	2,048
Buffer size	20,480
Learning rate	0.0003
Beta	0.005
Epsilon	0.2
Lambda	0.95
Epochs	3
Hidden units	256
Number of layers	3
Gamma	0.995
Max. training steps	30,000,000
Time horizon	1,000
Max. episode steps	3,000
Stacked vectors	20
Decision period	10

shadow obstacles possibly resulted in an impossible setting thus causing the discrepancy of 0.005 in average reward.

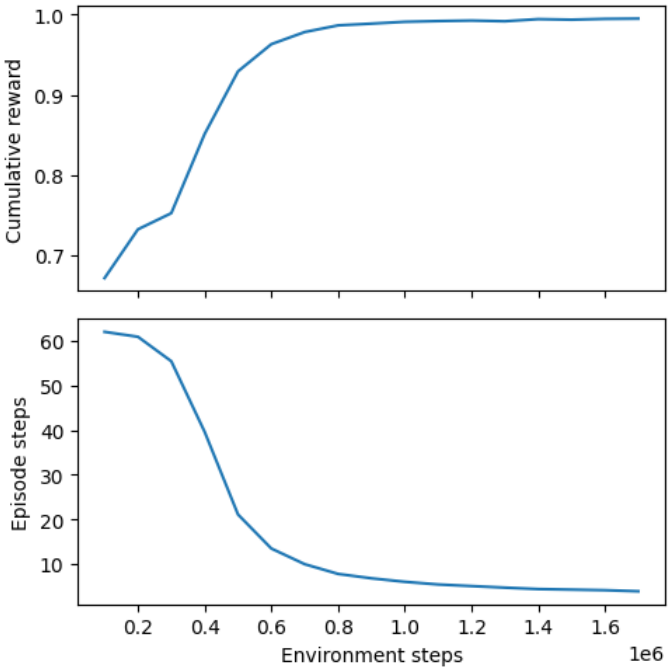

Fig. 6. Training outcome: *(i) top shows the average cumulative results during the course of training, (ii) bottom shows the average episode length in steps*

V. CONCLUSION

Despite only using 15 observation states, the agent showed promising results in learning on how to manoeuvre away from shadows and aligning itself towards the sun's direction, mimicking etiolation in plants. Unfortunately, the manoeuvring could be more concise while using lesser steps to achieve its target orientation, sometimes overshotting its target. This could be due to the lack of observation states to describe the shadowing being casted onto the solar panels.

In our attempt, we utilized a 6-segment solar panel array that were fitted in parallel and did not use any visual observation of the shadow mappings and instead used the shadow ratio. The shadow ratios were not indicative of how the shadows were being casted onto the panels therefore the agent did not know how big the shadow was and how much it had to veer away to avoid the shadow. This could be solved by increasing the number of solar panel segments in the array. With more segments, the observation states would be more informative in terms of which segment is being shadowed and how the shadow is being divided amongst the segments.

VI. FUTURE WORKS

Due to the lack of time and expertise, a custom manipulator could not be designed and built during the course of the project. As the main objective of this project was to test the feasability of a reinforcement-based solar tracking system, a simulation of the artefact was built instead of an actual one.

With this, a manipulator equipped with stepper motors and self-locking mechanism using worm gears would be required to translate the simulation to a real life solar tree as usual motors built into robotic arms tend to be servo motors that have low detent torque, meaning that it does not withstand force or retain its position when unpowered.

As for the brain, there were plans to use Raspberry Pi Pico microcontroller to run the reinforcement agent model using Tensorflow-Lite and Pico C++ SDK. The advantage of Raspberry Pi Pico was its low power usage and its in-built low-power dormant sleep mode that was reported to only run on 0.8mAh [12].

To further improve the agent, it is could be worthwhile to explore training multiple behaviours for the agent. The task could be split into 2 behaviours, a behaviour could be specialized in just positioning itself into the sun while another behaviour could be for manoeuvring away from shadows.

Another suggestion would be to train a swarm of these tree modules using Particle Swarm Optimization (PSO) as the objective was to work as a team to maximize power output.

REFERENCES

[1] P. Andrews-Speed, (2021, February 7) Commentary: Why hasn't solar energy in Singapore taken off in a big way after so long? Channel News Asia. https://www.channelnewsasia.com/commentary/solar-energy-singapore-panels-cloudy-unpredictable-electricity-1882996

[2] M. Almadhhachi, I. Seres, I. Farkas, "Significance of solar trees: Configuration, operation, types and technology commercialization", Energy Reports, vol. 8, pp. 6729 - 6743, 2022.

[3] Matius, M. E., Ismail, M. A., Farm, Y. Y., Amaludin, A. E., Radzali, M. A., Fazlizan, A., Muzammil, W. K. (2021). On the Optimal Tilt Angle and Orientation of an On-Site Solar Photovoltaic Energy Generation System for Sabah's Rural Electrification. Sustainability, 13(10), 5730. https://doi.org/10.3390/su13105730

[4] Solymosi, K., Vitanyi, B., Hideg, E., amp; Boddi, B. (2007). Etiolation symptoms in sunflower (helianthus annuus) cotyledons partially covered by the pericarp of the achene. Annals of Botany, 99(5), 857–867. https://doi.org/10.1093/aob/mcm034

[5] Abel David, Emily Reif and Michael L. Littman. "Improving Solar Panel Efficiency Using Reinforcement Learning." (2017).

[6] S. John, W. Filip, D. Prafulla, R.Alec and K. Oleg, "Proximal Policy Optimization Algorithms", In: *arXiv preprint arXiv:1707.06347* (2017).

[7] M. Blanco-Muriel, Alarcón-Padilla, D. C., López-Moratalla, T., and Lara-Coira, M. Í., "Computing the solar vector", Solar Energy, vol. 70, pp. 431 - 441, 2001.

[8] ESRL Global Monitoring Laboratory - Global Radiation and Aerosols. (n.d.). Retrieved November 4, 2022, from https://gml.noaa.gov/grad/solcalc/

[9] countries.csv — Dataset Publishing Language —. (n.d.). Google Developers. Retrieved November 4, 2022, from https://developers.google.com/public-data/docs/canonical/countries_csv
[10] RX-160 Industrial robot. (n.d.). Automate. Retrieved November 4, 2022, from https://www.automate.org/products/staubli-robotics/rx-160-industrial-robot
[11] Image Processing 101 Chapter 1.3: Color Space Conversion. (2019, May 24). Dynamsoft Blog. https://www.dynamsoft.com/blog/insights/image-processing/image-processing-101-color-space-conversion/
[12] Raspberry Pi Documentation - The C/C++ SDK. (n.d.). Retrieved November 4, 2022, from https://www.raspberrypi.com/documentation/microcontrollers/c_sdk.html