Improving Solar Panel Efficiency Using Reinforcement Learning

Abstract

Solar panels sustainably harvest energy from the sun. To improve performance, panels are often equipped with a tracking mechanism that computes the sun's position in the sky throughout the day. Based on the tracker's estimate of the sun's location, a controller orients the panel to minimize the angle of incidence between solar radiant energy and the photovoltaic cells on the surface of the panel, increasing total energy harvested. Prior work has developed efficient tracking algorithms that accurately compute the sun's location to facilitate solar tracking and control. However, always pointing a panel directly at the sun does not account for diffuse irradiance in the sky, reflected irradiance from the ground and surrounding surfaces, or changing weather conditions (such as cloud coverage), all of which are contributing factors to the total energy harvested by a solar panel. In this work, we show that a reinforcement-learning (RL) approach can increase the total energy harvested by solar panels by learning to dynamically account for such other factors. We advocate for the use of RL for solar panel control due to its *effectiveness*, *negligible cost*, and *versatility*. Our contribution is twofold: (1) an adaption of typical RL algorithms to the task of improving solar panel performance, and (2) an experimental validation in simulation based on typical solar and irradiance models for experimenting with solar panel control. We evaluate the utility of various RL approaches compared to an idealized controller, an efficient state-of-the-art direct tracking algorithm, and a fixed panel in our simulated environment. We experiment across different time scales, in different places on earth, and with dramatically different percepts (sun coordinates and raw images of the sky with and without clouds), consistently demonstrating that simple RL algorithms improve over existing baselines.

1 Introduction

Solar energy offers a pollution free and sustainable means of harvesting energy directly from the sun. Considerable effort has been directed toward maximizing the efficiency of end-to-end solar systems, including the design of photovoltaic cells [15, 26], engineering new photovoltaic architectures and materials [24], and solar tracking systems [4]. Solar tracking is especially important for maximizing performance of solar panels [8, 37, 21]. Given the proper sensors and hardware, a tracking algorithm can compute the relative location of the sun in the sky throughout the day, and a controller can orient the panel to point at the sun, illustrated in Figure 1. Its goal is to minimize the angle of incidence between incoming solar radiant energy and the grid of photovoltaic cells, as in Eke and Senturk [8], Benghanem [3], King *et al.* [21] and Kalogirou [17].

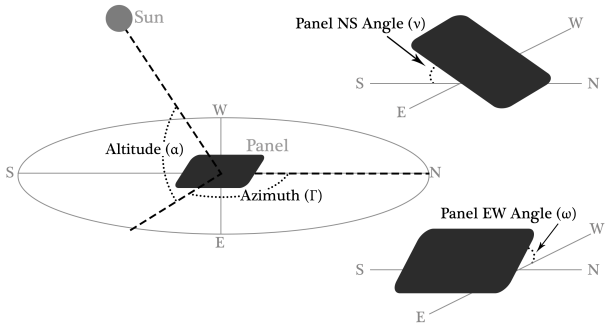


Figure 1: In the solar panel control problem, the panel changes its orientation over time to maximize total exposure to solar radiant energy.

Prior work has consistently demonstrated that panels using a tracking system increase the total energy by a substantial amount: Eke and Senturk [8] report that a dual-axis tracker yielded 71 kW/h, compared to a fixed panel's yield of 52 kW/h on the same day. They also report energy harvesting gains of dual-axis tracking systems over fixed systems varying from 15% to 40%, depending on the time of year. Mousazadeh *et al.* [32] report that gains from tracking can vary between 0% and 100%, while Clifford and Eastwood [5] report a gain of 23% due to tracking in simulation. Solar tracking and control result in non-trivial benefits in solar photovoltaic systems.

Recent work in solar tracking has focused on algorithms that are sufficiently accurate to inform control of panels,

building on the early work of Spencer [40], Walraven [45] and Michalsky [30]. The algorithm introduced by Reda and Andreas [36] computes the sun's location in the sky within $\pm 0.0003^\circ$ of accuracy, achieving the highest degree of accuracy of any known algorithm, but is computationally inefficient to the point of impracticality. Grena [12] overcomes these inefficiencies with a tracking algorithm that requires an order of magnitude fewer calculations while still achieving 0.0027° of accuracy.

However, prior literature suggests that a variety of factors contribute to the performance of a panel [21], and thus, pointing a panel directly at the sun is not always optimal behavior. Specifically, the total solar irradiance falling on a panel is a combination of *direct*, *reflective*, and *diffuse* irradiance [3]. The diffuse irradiance typically varies between 15% and 55% of direct irradiance depending on factors like cloud coverage and the time of day [33], while a case study by the Cold Climate Housing Research Center in Fairbanks, Alaska reports reflective irradiance varying from 5% to 25% of direct irradiance [6]. The reflective irradiance varies heavily based on the percentage of irradiance reflected off the surrounding ground surface: Typical values for this percentage given by McEvoy *et al.* [29] vary between 17% (soil), 25% (grass), 55% (concrete), and 90% (snow). Additionally, changing weather and atmospheric conditions can affect the optimal panel orientation [20]. Thus, optimal performance may involve prioritizing reflective or diffuse irradiance when direct sunlight is not available.

There are two additional shortcomings to the classical tracking approach. First, tracking algorithms take as input a variety of data that require additional hardware such as a barometer, thermometer, or GPS [13], increasing the total cost and system complexity.[1] Second, tracking algorithms are only accurate for a fixed window of time: The algorithm of Grena [12] is noted as accurate until 2023 AD (due to the subtle movements of the earth and sun), while the algorithms in Grena [13] are reported as accurate until 2110 AD.

In this work, we advocate for the use of RL to optimize solar panel performance. In this setting, a learned solar panel controller can account for weather change, cloud coverage, and diverse reflective indices of surroundings, offering an efficient yet adaptive solution that can optimize for the given availability of each type of solar irradiance without the need for complex hardware, regardless of the location or year. Our primary contribution is twofold:

1. The advancement of a highly relevant problem as an application area for RL, including a high fidelity simulation built using recently introduced models of solar irradiance.

2. The validation of the utility of RL approaches for solar panel control.

[1]The temperature and pressure are used to compute the refractive effects on the sun's perceived location, which our simulation ignores. Some tracking algorithms do not require the temperature and pressure but incur a cost of accuracy, such as Algorithm 1 from Grena [13].

Quantity	Variable	Range
latitude	L	$[-\pi, \pi]$
longitude	G	$\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$
azimuth	Γ	$[-\pi, \pi]$
altitude	α	$[-\pi, \pi]$
panel NS angle	ν	$[-\pi, \pi]$
panel EW angle	ω	$[-\pi, \pi]$
ground reflective index	ρ	$[0, 1]$

Figure 2: Relevant solar tracking and irradiance variables.

2 Background

We begin with some background on solar tracking and RL.

2.1 Solar Tracking

The amount of solar radiant energy contacting a surface on the earth's surface (per unit area, per unit time) is called *irradiance* [11]. We denote the total irradiance hitting a panel as R_t, which, per the models developed by Kamali *et al.* [19], is approximated by the sum of the *direct* irradiance, R_d, *diffuse* irradiance (light from the sky), R_f, and *reflective* irradiance, R_r (reflected off the ground or other surfaces). Each of these components is modified by a scalar, $\theta_d, \theta_f, \theta_r \in [0, 1]$, denoting the effect of the angle of incidence between oncoming solar rays and the panel's orientation, yielding the total:

$$R_t = R_d\theta_d + R_f\theta_f + R_r\theta_r \tag{1}$$

Additionally, the components R_d and R_f are known to be effected by cloud coverage [23, 34, 44]. We attend to these details in describing our simulation in Section 3.

A controller for a solar panel then seeks to maximize total irradiance, R_t, hitting the panel's surface. In the case of solar trackers, a running assumption is that it is near optimal to orient the panel such that its normal vector is pointing at the sun, and thus arises the necessity for accurate solar tracking algorithms. There are many types of tracking methods, only a few of which we discuss in this work; for an in depth survey of solar tracking techniques, see Mousazadeh *et al.* [32].

2.2 RL Background

Reinforcement learning is a computational learning paradigm in which an agent learns to make decisions that maximize an unknown reward function through repeated interaction with the agent's environment. In this work, we model the environment as a Markov Decision Process (MDP) [35]. An MDP is a five tuple, $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$, where:

1. $\mathcal{S}$ is a set of states,
2. $\mathcal{A}$ is a set of actions,
3. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \longmapsto \mathbb{R}$ is a reward function. Here, we assume the codomain is the real interval $[0, \text{RMAX}]$, WLOG, for some constant RMAX,
4. $\mathcal{T}(s' \mid s, a)$ is a probability distribution on next states given a state and action,

5. $\gamma \in (0, 1)$ is a discount factor, indicating how much the agent prefers immediate reward over future reward.

The solution to an MDP is called a *policy*, denoted $\pi : \mathcal{S} \longmapsto \mathcal{A}$. The goal of an agent is solve for a policy that maximizes long term expected reward, defined by the *value function*, $V^* : S \longmapsto \left[0, \frac{\text{RMAX}}{1-\gamma}\right]$, given by the classic Bellman Equation:

$$V^*(s) = \max_a \left(\mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s, a) V^*(s') \right). \quad (2)$$

Also of interest is the *action-value function*, $Q^* : \mathcal{S} \times \mathcal{A} \longmapsto \left[0, \frac{\text{RMAX}}{1-\gamma}\right]]$, which specifies the long term expected reward of executing an action in a state and behaving optimally thereafter:

$$Q^*(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{T}(s' \mid s, a) V^*(s'). \quad (3)$$

For further background on RL, see Sutton and Barto [41] or Kaelbling *et al.* [16].

Agents

We experiment with two relatively simple learning agents: Q-Learning and SARSA, each with a linear function approximator to estimate Q^*. To test the significance of modeling the sequential aspects of the problem, we also conduct experiments with Q-Learning with $\gamma = 0$ (so it only maximizes immediate return) and LinUCB [25], a standard approach to non-sequential Contextual Bandits. We chose each of the linear approximators to illustrate that online, efficient, and lightweight algorithms can be effective in the domain. We chose not to experiment with any Deep RL agents [31], as Deep RL typically requires more computational power (and often GPUs), which may be unavailable or limited in the real solar panel setting. However, this could be an area of further exploration.

Q-Learning, introduced by Watkins and Dayan [47], maintains an estimate of Q^* via updates after each experience $\langle s, a, r, s' \rangle$, updating according to the rule:

$$\hat{Q}(s, a) = (1 - \eta)\hat{Q}(s, a) + \eta(r + \gamma \max_{a'} \hat{Q}(s', a')), \quad (4)$$

where $\eta \in [0, 1]$ is a learning rate. The linear approximator extends tabular Q-Learning to domains where states are described by feature vectors, $\boldsymbol{s} = \begin{bmatrix} s_1 & s_2 & \dots & s_k \end{bmatrix}$. Here, $\hat{Q}$ is parameterized by a set of k-vectors w^a, where each vector corresponds to action a's parameters across the state variables, resulting in:

$$\hat{Q}_w(\boldsymbol{s}, a) = \sum_{i=1}^{k} w_i^a s_i. \quad (5)$$

The parameters are updated via the gradient update rule—given a single experience $\langle s, a, r, s' \rangle$:

$$w^a = w^a + \eta \left(r + \gamma \max_{a'} Q_w(s', a') - Q_w(s, a) \right). \quad (6)$$

SARSA [39] is similar, but makes its update using the chosen next action a', instead of the max action:

$$w^a = w^a + \eta \left(r + \gamma Q_w(s', a') - Q_w(s, a) \right). \quad (7)$$

We pair each of these algorithms with the naïve ε-greedy policy, which chooses actions according to:

$$\pi^\varepsilon(s) = \begin{cases} \text{argmax}_a \hat{Q}(s, a) & X = 0, X \sim \text{Bern}(\varepsilon), \\ a \sim \text{Unif}(\mathcal{A}) & otherwise, \end{cases} \quad (8)$$

where $\text{Unif}(\mathcal{A})$ is the uniform distribution on actions and $\text{Bern}(\varepsilon)$ is a Bernoulli distribution with parameter ε. For further details about these algorithms, see Geramifard *et al.* [9].

To increase the expressively of the linear function approximator, we introduce some non-linearity by applying a Gaussian radial basis function to each state variable. That is:

$$\phi(x) = e^{-(x^2)}, \quad (9)$$

and the agent receives states as:

$$\phi(s) = \begin{bmatrix} \phi(s_1) & \phi(s_2) & \dots & \phi(s_k) \end{bmatrix}. \quad (10)$$

Contextual Bandits

A simplification of the full RL problem is a non-sequential variant known as the Contextual Bandit introduced by Wang *et al.* [46], which extends the classic Multi-Armed Bandit problem [10] to include a *context matrix* $\boldsymbol{X}$ containing a feature vector for each action. That is, each column of $\boldsymbol{X}$ corresponds to each action's context: the entry $\boldsymbol{X}_{i,j}$ denotes the i-th feature of action a_j. We let $\boldsymbol{x}^a$ denote the context vector associated with action a. At each time step, the agent chooses an action $a \in \mathcal{A}$, and receives payoff according to an unknown, possibly stochastic function, $\mathcal{R}(\boldsymbol{x}, a)$. Here, agents still face the exploration–exploitation dilemma but do not need to learn their estimates from delayed reward.

We use the contextual bandit framework to assess the importance of modeling the solar panel control problem as sequential. The context vector for each action is the received percept from the environment (there is no difference in contexts across actions). Due to its simplicity and efficiency, we experiment with the LinUCB algorithm developed by Li *et al.* [25].

LinUCB adapts the core ideas of the UCB (Upper Confidence Bound) algorithm [2] to deal with contexts. At a high level, LinUCB assumes the underlying reward is determined by a linear payoff matrix $\boldsymbol{\theta}$ and maintains a running estimate $\hat{\boldsymbol{\theta}}$. The critical piece of the algorithm is its exploration strategy, which calculates a confidence interval on the difference between the agent's estimate of the expected reward and the actual return, which is factored into an overall *score* for each action. At each round, the agent then selects the action with maximal score according to:

$$\pi(s) = \underset{a \in \mathcal{A}}{\text{argmax}} \left(\sum_i \boldsymbol{x}_i^a \hat{\boldsymbol{w}}_i^a + \sigma^a \right), \quad (11)$$

where σ^a represents the confidence interval associated with action a. For the full details, see Li *et al.* [25].

We now turn to describing the details of our simulation.

3 Simulation

We introduce a high fidelity simulated environment to validate the use of RL for solar panel control. There are four basic stages to the simulation:

1. Computing the sun's location in the sky, relative to the panel.
2. Computing R_d, R_f, and R_r.
3. Computing θ_d, θ_f, and θ_r.
4. Generating percepts.

3.1 Sun's location in the sky

For a given latitude, longitude, year, month, day, and time, we simulate the relative positions of the sun to the specified location on earth. Our simulation computes the sun's altitude α (angle: degrees above the horizon) and azimuth Γ (angle: clockwise degrees along the horizon relative to North) using the highly accurate tracker algorithm from Reda and Andreas [36] implemented in the open source library `pysolar`.[2] Due to space constraints, we do not express the full computation. Note, however, that a quicker approximation (also available in `pysolar`) is given by:

$$\alpha = \arcsin(\cos L \cos\delta \cos H + \sin L \sin\delta), \tag{12}$$

$$\Gamma = \arcsin\left(\frac{\cos\delta \sin H}{\cos\alpha}\right). \tag{13}$$

For the full details, see Reda and Andreas [36].

3.2 Computing R_d, R_f, R_r

Given the sun's altitude α and azimuth Γ, we compute the R_d, R_f, and R_d from the models of Threlkeld and Jordan [43], Liu and Jordan [27] and Masters [28].[3]

$$R_d = Ae^{-km}, \tag{14}$$

$$R_f = C \cdot R_d, \tag{15}$$

$$R_r = \rho R_d(\sin\alpha + C), \tag{16}$$

where A is the apparent extraterrestrial flux, k is the optical depth, m is the air mass ratio, $\rho \in [0, 1)$ is a reflective index (albedo) denoting how reflective the ground is, and C is a sky diffusion factor, each given by the approximations:

$$m = \frac{1}{\sin\alpha}, \quad A = 1160 + \sin(0.99n - 271), \tag{17}$$

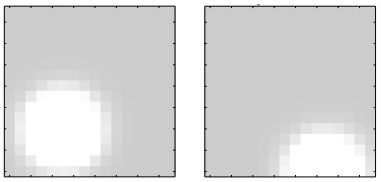
$$k = 0.174 + 0.035\sin(0.99n - 99), \tag{18}$$

$$C = 0.095 + 0.04\sin(0.99n - 99), \tag{19}$$

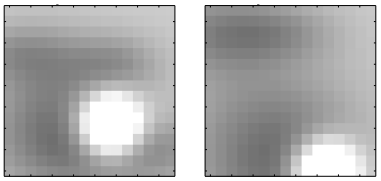
where $n \in [1 : 365]$ is a day of the year.

[2] `pysolar.org`

[3] Higher fidelity models are known to exist, such as those developed by Andersen [1], Klein [22] and Kamali *et al.* [19]. In particular, our estimates of the diffuse and reflective radiation are simple relative to the best known models. (This choice was made to make the simulation more efficient.)



(a) Sample Sky Percepts



(b) Sample Sky + Cloud Percepts

Figure 3: Example percepts given to the RL agent with no clouds (top) and simulated cloud coverage (bottom).

3.3 Computing $\theta_d, \theta_f, \theta_r$

Given the two angles describing the panel's orientation (ν: north-south tilt, ω: east-west tilt), we simulate the amount of total irradiance actually hitting the panel's surface. The models of Masters [28] compute θ_t in terms of the cos-similarity between the panel's normal vector, $\vec{p}$, and the sun's vector, $\vec{s}$, (with the panel as the origin):

$$\vec{p} = \begin{bmatrix}\sin(\nu)\cos(\omega) & \cos(\nu)\cos(\omega) & \cos(\nu)\cos(\omega)\end{bmatrix},$$

$$\vec{s} = \begin{bmatrix}\sin(\pi - \Gamma)\cos(\alpha) & \cos(\pi - \Gamma)\cos(\alpha) & \sin(\alpha)\end{bmatrix}.$$

We then compute θ_d as follows:

$$\theta_d = \frac{\vec{p}\cdot\vec{s}}{||\vec{p}||||\vec{s}||} = \sin(\nu)\cos(\omega)\sin(\pi - \Gamma)\cos(\alpha) + \cos(\nu)\cos(\omega)\cos(\pi - \Gamma)\cos(\alpha) + \cos(\nu)\cos(\omega)\sin(\alpha).$$

The diffuse irradiance incident angle θ_f is given by a simple approximation—the solar collector is exposed to the fraction of the sky it points to—while θ_r is given by the fraction of the ground the collector points to:

$$\theta_f = \frac{\cos\nu + \cos\omega}{2}, \quad \theta_r = \frac{2 - \cos\nu - \cos\omega}{2}. \tag{20}$$

3.4 Generating Percepts

The final step in the simulation is to generate percepts (state variables) for the learning algorithms.[4] Our most immediate plan for future work is to build a physical system to conduct experiments with RL outside of simulation. In the real setting, we plan on equipping each solar panel with a fish eye monocular camera to provide images of the sky as input for the RL algorithm. To approximate the real setting, our simulated environment supports percepts of three kinds:

1. The panel's orientation and two angles representing the sun's true position in the sky, relative to the panel (four state variables).

[4] The tracking algorithm always receives the same information: the year, month, day, hour, longitude, and latitude.

2. A 16×16 synthesized grayscale image of the clear sky (256 state variables, each in the range $[0, 1]$).
3. A 16×16 synthesized grayscale image of the sky with simulated cloud cover (256 state variables, each in the range $[0, 1]$).

For both image percepts, the perceived fraction of the sky changes as the panel moves to approximate having the camera mounted to the panel. In generating images, we ignore refractive effects on irradiance due to temperature and pressure.

Cloud cover is generated as Gaussian blobs in the synthesized images. The cloud conditions are randomized each morning at 4am, with the clouds moving from the left side of the image to the right (across the sky) throughout the course of the day (1 pixel per hour). Each day at 4am, we generate between 1 and 5 clouds uniformly at random, with each cloud is modeled as a Multivariate Gaussian. Parameters are randomized as:

$$\mu \sim \begin{bmatrix}\text{Unif}(0:N) & \text{Unif}(0:N)\end{bmatrix},$$
$$\Sigma \sim \begin{bmatrix}\text{Unif}(2:6) & 0.2 \\ 0.2 & \text{Unif}(2:4)\end{bmatrix}.$$

Where $\text{Unif}(x : y)$ denotes a uniform random sample from the natural interval $[x : y]$, and N is the dimension of the image (which we set to 16). The intensity of the cloud corresponds to the value of the Gaussian. Surprisingly, studies demonstrate that diffuse irradiance can be either magnified [38] or decreased [34] by cloud cover, depending on specific conditions. In light of these findings, our main interest in experimenting with clouds is to evaluate RL approaches on a rich state space. Direct irradiance (R_d), however, is almost always decreased by cloud cover. Thus, if a cloud sits between the panel and the sun, we reduce the direct irradiance hitting the panel's surface by a factor proportional to the cloud's intensity, but choose not to alter the diffuse irradiance, since there are conflicting empirical findings on its impact [38, 34]. In most experiments involving cloud cover, the clouds reduce the direct irradiance by anywhere from 0% to 40% at various points throughout the day.

4 Experiments

Our simulation is wrapped in an MDP using the open source library `simple-rl`.[5] In each experiment, evaluation is done *online*, in that the agent is learning while acting to better parallel the nature of solar panel control. The action space consists of five actions: $\mathcal{A} = \{$`tilt N`, `tilt E`, `tilt S`, `tilt W`, `nothing`$\}$. Executing the `nothing` action keeps the panel orientation fixed, while the other four each shift the panel 2° in their respective directions. Each decision step is equivalent to three minutes of time passing.

Our core algorithms are Q-Learning (`ql-lin`) and SARSA (`sarsa-lin`), each using a linear function approximator and a Gaussian radial basis function, where the input state features vary between percept types. In the first case, there are four state variables: the sun and panel angles (in radians). In the second two cases, the state variables are the pixel intensities of the image. We set exploration parameter $\varepsilon_0 = 0.3$ and learning rate $\eta_0 = 0.1$ with a standard annealing schedule, adopted from Darken and Moody [7]:

$$\eta_t = \frac{\eta_0}{\left(1.0 + \frac{t}{500}\right)}, \quad \varepsilon_t = \frac{\varepsilon_0}{\left(1.0 + \frac{t}{500}\right)}$$

Where t is a time step, and the update is performed every 500 time steps. For `q-lin` and `sarsa-lin` we chose to set $\gamma = 0.99$ to emphasize the long term consequences of behavior, contrasted with LinUCB and the short-sighted version of Q-Learning with $\gamma = 0.0$ (`q-lin,` $\gamma = 0.0$).

Our core benchmark algorithm is Algorithm 2 from Grena [13], an efficient but accurate solar tracking algorithm, coupled with a controller that always points perfectly at the tracker's estimate of the sun's location. We also provide results for a fixed panel to illustrate the importance of tracking (`fixed`), and a highly idealized controller that computes the perfect orientation at each decision step to illustrate an upper bound on possible performance (`optimal`), and to visualize the degree of sub-optimality of the other approaches.

In a separate experiment, we explore the performance difference between a single- and double-axis panel. For the single-axis panel, each agent can only tilt the panel along the East–West axis. For this experiment, we evaluated the core algorithms from our other experiment in New York City, NY during the first five days of August 2018.

All of our code for running experiments and for reproducing results is publicly available.[6]

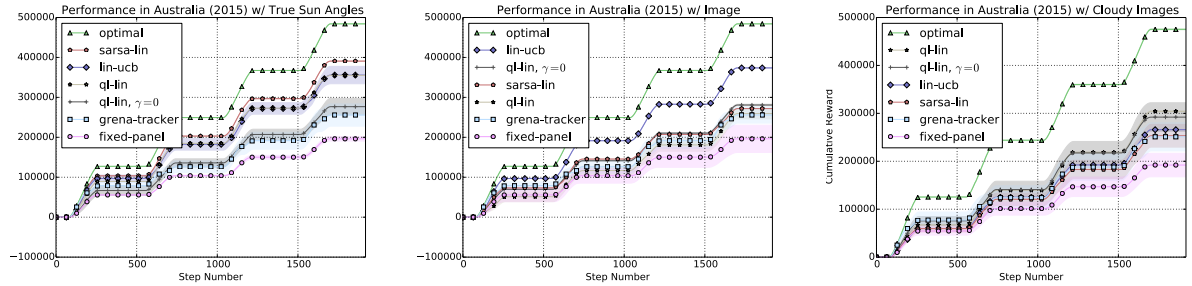
4.1 Results

Figure 4 illustrates the cumulative irradiance exposed to each panel in the Australia experiments (top) and the Iceland experiments (bottom). Notably, with the simple percept of the true sun angles, *all* of the learning algorithms outperform the baseline tracker and fixed panel in Australia, while in Iceland, `lin-ucb` performs worse than the `fixed` panel for the first two percepts. When just the image is provided in Australia, `lin-ucb` achieves by far the best performance; we hypothesize that this is due to LinUCB's informed approach to exploration compared to the ε-greedy used by `sarsa-lin` and both `ql-lin` and `ql-lin,` $\gamma = 0$. Additionally, when just the sun is in the image, there is little incentive to forecast expected future reward beyond the immediate next step, which may explain the success of `lin-ucb`. This is further corroborated by the fact that `ql-lin,` $\gamma = 0$ does quite well in the experiment as well.

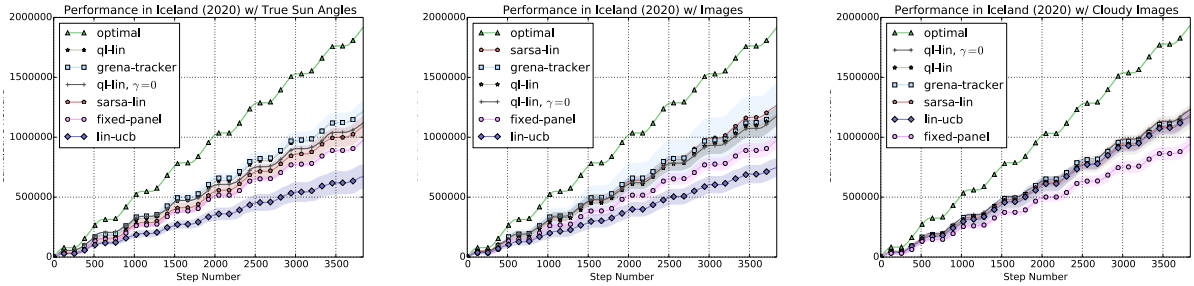
Conversely, we see that `lin-ucb` continues to struggle in Iceland. When clouds are present, `lin-ucb` performs comparably to the other learners. The cloudy image percepts pose a challenging RL problem, but still we see that the simple approaches achieve similar performance as the `grena-tracker`, and note the substantial room for improvement from further training or more complex learners. In Iceland, the results are largely the same as Australia, though we note that the `grena-tracker` does better. In all cases,

[5] `https://github.com/david-abel/simple_rl`

[6] *Link redacted during review.*



(a) True Sun Angles (b) Bitmap of the Sky (c) Bitmap of the Cloudy Sky



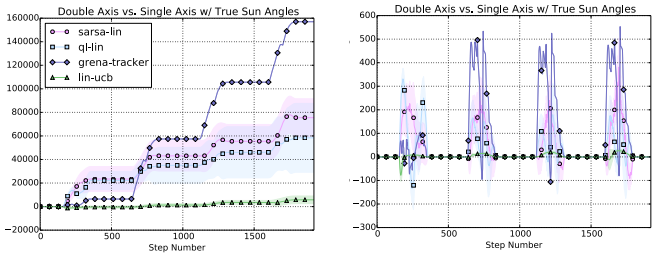
(d) True Sun Angles (e) Bitmap of the Sky (f) Bitmap of the Cloudy Sky

Figure 4: Cumulative irradiance falling on the panel's surface given different percepts over four days in Mildura, Australia in July of 2015

we note that there is room for improvement, suggesting that more sophisticated approaches may have more success on real panels than current techniques.

In all our simulations, the `grena-tracker` consistently outperforms the `fixed-panel` by around 25%, consistent with previously published results [8, 32, 5].

Figure 5 shows results for experiments comparing the *advantage* of the two-axis controller vs. a single-axis controller. In all cases except for `lin-ucb`, the presence of another axis immediately improves performance. Clearly the added problem complexity of the larger state-action space doesn't detract from the quality of the policy that the RL agents learn. We also observe that `lin-ucb` barely sees any performance gain from using the dual-axis controller.



(a) Cumulative (b) Average

Figure 5: The advantage of a dual-axis over a single-axis approach with the true sun percepts, shown as cumulative difference (left) and average difference (right). Positive values indicate the dual-axis version captured more irradiance.

5 Conclusion

We have here demonstrated the benefits of using a reinforcement-learning approach to improving the efficiency of solar panels over several established baselines. We introduced a high fidelity testbed for the problem of solar energy harvesting capable of simulating solar irradiance models anywhere on earth with a variety of generated percepts to approximate real world conditions. We take the end-to-end simulation, and the evaluation of RL on solar panel control, to be of independent interest to the broader AI and computational sustainability communities.

In the future, the natural next step is to implement a functioning RL controller on real solar panels. Additionally, there are novel algorithmic challenges posed by the solar panel setting. First, movement, observation, and computation all expend energy; incorporating these expenditures explicitly into the model poses both challenging planning and exploration questions. Second, Hsu *et al.* [14] demonstrate that simple RL approaches can optimize the problem of Maximum Power Point Tracking (MPTT) for solar panels. A system that jointly optimizes over these two criteria poses another difficult challenge. Lastly, we plan on applying RL to the analogous control problem presented by solar thermal energy [18].

References

[1] Pauli Andersen. Comments on calculations of monthly average insolation on tilted surfaces by sa klein. *Solar Energy*, 25(3):287, 1980.

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[3] M. Benghanem. Optimization of tilt angle for solar panel: Case study for Madinah, Saudi Arabia. *Applied Energy*, 88(4):1427–1433, 2011.

[4] Eduardo F Camacho and Manuel Berenguel. Control of solar energy systems1. *IFAC Proceedings Volumes*, 45(15):848–855, 2012.

[5] MJ Clifford and D Eastwood. Design of a novel passive solar tracker. *Solar Energy*, 77(3):269–280, 2004.

[6] Ryan Colgan, Nathan Wiltse, Michael Lilly, Ben LaRue, and Greg Egan. Performance of photovoltaic arrays. *Cold Climate Housing Research Center*, 2010.

[7] Christian Darken and John E Moody. Note on learning rate schedules for stochastic optimization. In *NIPS*, volume 91, pages 832–838, 1990.

[8] Rustu Eke and Ali Senturk. Performance comparison of a double-axis sun tracking versus fixed PV system. *Solar Energy*, 86(9):2665–2672, 2012.

[9] Alborz Geramifard, Thomas J Walsh, Stefanie Tellex, Girish Chowdhary, Nicholas Roy, Jonathan P How, et al. A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Foundations and Trends® in Machine Learning*, 6(4):375–451, 2013.

[10] John C Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 148–177, 1979.

[11] D Yogi Goswami, Frank Kreith, and Jan F Kreider. *Principles of solar engineering*. CRC Press, 2000.

[12] Roberto Grena. An algorithm for the computation of the solar position. *Solar Energy*, 82(5):462–470, 2008.

[13] Roberto Grena. Five new algorithms for the computation of sun position from 2010 to 2110. *Solar Energy*, 86(5):1323–1337, 2012.

[14] Roy Chaoming Hsu, Cheng-Ting Liu, Wen-Yen Chen, Hung-I Hsieh, Hao-Li Wang, Roy Chaoming Hsu, Cheng-Ting Liu, Wen-Yen Chen, Hung-I Hsieh, and Hao-Li Wang. A Reinforcement Learning-Based Maximum Power Point Tracking Method for Photovoltaic Array. *International Journal of Photoenergy*, 2015:1–12, 2015.

[15] J A Jervase, H Bourdoucen, and A Al-Lawati. Solar cell parameter extraction using genetic algorithms. *Measurement Science and Technology*, 12(11):1922–1925, 2001.

[16] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[17] Soteris A Kalogirou. Design and construction of a one-axis sun-tracking system. *Solar Energy*, 57(6):465–469, 1996.

[18] Soteris A Kalogirou. Solar thermal collectors and applications. *Progress in energy and combustion science*, 30(3):231–295, 2004.

[19] Gh A Kamali, I Moradi, and A Khalili. Estimating solar radiation on tilted surfaces with various orientations: a study case in karaj (iran). *Theoretical and applied climatology*, 84(4):235–241, 2006.

[20] Nelson A. Kelly and Thomas L. Gibson. Improved photovoltaic energy output for cloudy conditions with a solar tracking system. *Solar Energy*, 83(11):2092–2102, 2009.

[21] D L King, William E Boyson, and J A Kratochvil. Analysis of factors influencing the annual energy production of photovoltaic systems. *Conference Record of the Twenty-Ninth IEEE Photovoltaic Specialists Conference, 2002.*, pages 1356–1361, 2001.

[22] SA Klein. Calculation of monthly average insolation on tilted surfaces. *Solar energy*, 19(4):325–329, 1977.

[23] Danny HW Li, Chris CS Lau, and Joseph C Lam. Overcast sky conditions and luminance distribution in hong kong. *Building and Environment*, 39(1):101–108, 2004.

[24] Gang Li, Vishal Shrotriya, Jinsong Huang, Yan Yao, Tom Moriarty, Keith Emery, and Yang Yang. High-efficiency solution processable polymer photovoltaic cells by self-organization of polymer blends. *Nature materials*, 4(11):864–868, 2005.

[25] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM, 2010.

[26] Yongfang Li. Molecular design of photovoltaic materials for polymer solar cells: toward suitable electronic energy levels and broad absorption. *Accounts of Chemical Research*, 45(5):723–733, 2012.

[27] Benjamin Y.H. Liu and Richard C. Jordan. The interrelationship and characteristic distribution of direct, diffuse and total solar radiation. *Solar Energy*, 4(3):1–19, 1960.

[28] Gilbert M Masters. *Renewable and efficient electric power systems*. John Wiley & Sons, 2013.

[29] Augustin McEvoy, Tom Markvart, Luis Castañer, T Markvart, and Luis Castaner. *Practical handbook of photovoltaics: fundamentals and applications*. Elsevier, 2003.

[30] Joseph J Michalsky. The astronomical almanac's algorithm for approximate solar position (1950–2050). *Solar energy*, 40(3):227–235, 1988.

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[32] Hossein Mousazadeh, Alireza Keyhani, Arzhang Javadi, Hossein Mobli, Karen Abrinia, and Ahmad Sharifi. A review of principle and sun-tracking methods for maximizing solar systems output. *Renewable and sustainable energy reviews*, 13(8):1800–1818, 2009.

[33] William A Peterson and Inge Dirmhirn. The ratio of diffuse to direct solar irradiance (perpendicular to the sun's rays) with clear skiesa conserved quantity throughout the day. *Journal of Applied Meteorology*, 20(7):826–828, 1981.

[34] G Pfister, RL McKenzie, JB Liley, A Thomas, BW Forgan, and Charles N Long. Cloud coverage based on all-sky imaging and its impact on surface solar irradiance. *Journal of Applied Meteorology*, 42(10):1421–1434, 2003.

[35] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[36] Ibrahim Reda and Afshin Andreas. Solar position algorithm for solar radiation applications. *Solar energy*, 76(5):577–589, 2004.

[37] J Rizk and Y Chaiko. Solar Tracking System: More Efficient Use of Solar Panels. *Proceedings of World Academy of Science: Engineering & Technology*, 43:313–315, 2008.

[38] Nathan Robinson. Solar radiation. *Elsevier*, 1966.

[39] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.

[40] JW Spencer. Fourier series representation of the position of the sun. *Search*, 2(5):172–172, 1971.

[41] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[42] Matthew E. Taylor, Matthew E. Taylor, Peter Stone, and Peter Stone. Cross-domain transfer for reinforcement learning. *Proceedings of the 24th international conference on Machine learning - ICML '07*, pages 879–886, 2007.

[43] JL Threlkeld and RC Jordan. Direct solar radiation available on clear days. *Heat., Piping Air Cond.*, 29(12), 1957.

[44] P Tzoumanikas, E Nikitidou, AF Bais, and A Kazantzidis. The effect of clouds on surface solar irradiance, based on data from an all-sky imaging system. *Renewable Energy*, 95:314–322, 2016.

[45] Robert Walraven. Calculating the position of the sun. *Solar Energy*, 20(5):393–397, 1978.

[46] Chih-Chun Wang, Sanjeev R Kulkarni, and H Vincent Poor. Bandit problems with side observations. *IEEE Transactions on Automatic Control*, 50(3):338–355, 2005.

[47] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.