SolarPACES 2022, 28th International Conference on Concentrating Solar Power and Chemical Energy Systems
Operations, Maintenance, and Component Reliability
https://doi.org/10.52825/solarpaces.v1i.719

Published: 02 Feb. 2024

HelioSoil: A Python Library for Heliostat Soiling Analysis and Cleaning Optimization

Giovanni Picotti[1], Michael E. Cholette[1], Ye Wang[2], Cody B. Anderson[1,3], Theodore A. Steinberg[1], John Pye[2], and Giampaolo Manzolini[3]

[1] Faculty of Engineering, Queensland University of Technology, Brisbane, Australia

[2] School of Engineering, The Australian National University, Canberra, Australia

[3] Dipartimento di Energia, Politecnico di Milano, Milan, Italy

Abstract. Soiling losses and their mitigation via cleaning operations represent important challenges for Solar Tower (ST) plants. Yet soiling losses are not well considered in existing CSP software, likely due to the lack of tools for soiling estimation and cleaning optimization. In this paper, a Python-based heliostat soiling library, called HelioSoil, is introduced which allows for the assessment of heliostats' soiling state and the optimization of the solar field cleaning schedule to maximize plant profit. The library is freely available on GitHub under a LGPL license, which enables extensions via other Python APIs (e.g. CoPylot) and integration with other CSP plant simulation packages to consider soiling losses. This latter capability is demonstrated in this study through an LCOE assessment and cleaning optimization of a hypothetical Australian ST plant with SolarTherm. Hence, HelioSoil provides the CSP community with a package for soiling assessment and cleaning resource optimization, which can be integrated with available software for high-level, long-term simulations. HelioSoil facilitates the inclusion of soiling and cleaning costs in CSP economics and ultimately aim to de-risk the deployment of ST plants.

Keywords: Soiling, CSP, Reflectance, Cleaning, Heliostats, Python, Open-Source

1. Introduction

The performance of Solar Tower (ST) power plants is strongly affected by the overall optical efficiency of the solar field, which can be significantly degraded by soiling of heliostats. Studies addressing and investigating the soiling process are available in literature, however, soiling-induced reflectance losses are not yet properly accounted for in commonly adopted software for CSP plant design and lifetime cost assessments [1], and only limited capabilities are available for PV technologies [2]. Although models have been recently developed to estimate the impact of soiling and to optimize cleaning regimes in CSP [3], [4], [5], [6], [7] there is currently no available software for estimating soiling losses and/or optimizing cleaning for a given CSP plant. The Heliostat Soiling (HelioSoil) library is presented in this study, which is based on the authors' previous work [3], [8]. The library enables the assessment of reflectance losses due to soiling in a solar field and their impact on the performance of the plant, which is subsequently exploited to optimize the cleaning regime. HelioSoil has been developed entirely in Python and is available on GitHub, which enables extension/interaction with other available Python libraries and Application Programmable Interfaces (APIs). The LGPL license enables its use with other commercial and open-source CSP software.

2. Library Overview

HelioSoil is available on GitHub at https://github.com/cholette/HelioSoil, under an open source LGPL-2.1 license. The HelioSoil library is composed of a soiling_model package, three notebooks that are used to demonstrate the functionalities of the library, and a Python environment file, which specifies the required Python libraries and can be used to create an environment via the conda package manager. The soiling_model package contains three main modules, namely base_models.py, cleaning_optimization.py, and utilities.py.

The base_models.py module is made of multiple classes that describe the main components of the soiling model presented in [8], define functions to import the required input data, characterize appropriately local environmental properties, define heliostats fields and their sectorization, compute optical efficiencies for each sector and assess their movement based on the coordinates, analyze reflectance data, define main constants, and predict reflectance losses for a given location and plant design. The core of the module is represented by the base_model class, which simulates the steps that describe the overall soiling process, from airborne dust measurements to expected reflectance losses, as described in [8]. The field_model and the fitting_experiment classes are subclasses of base_model, which serve two of the main purposes of the library: 1) simulating the reflectance losses for a whole solar field and 2) analyzing field data to fit the free parameter of the soiling model. A more detailed description of these classes and their methods will be given in the following. Other relevant classes in the base_models.py module are those used to instantiate and collect all the functions required to import and define the environmental inputs or a given subset (class simulation_inputs), dust characteristics (class dust), sun apparent movement (class sun), heliostat characteristics and solar field sectors (class helios), plant design (class plant), constants (class constants), and reflectance measurements and their subsets (class reflectance_measurements).

The cleaning_optimization.py module deals exclusively with the definition of the optimization problem (class optimization_problem) and the evaluation of a periodic cleaning schedule, whose final output is the Total Cleaning Cost (TCC) due to costs incurred for cleaning and the revenue lost because of soiled heliostats [3]. The utilities.py module is mostly made of functions that are required to treat and plot input or experimental data, and an important class used to define the airborne dust size distribution (class DustDistribution).

2.1 Input Files

The modules take as input three Excel data sheets (.xlsx): one for the basic model parameters, one for input data, and one for the solar field layout (that can also be given as a Comma Separated Value file). One EnergyPlus Weather (.epw) input file is also required to define the climate and the geographical coordinates of the simulated plant.

The input data workbook must have the following two sheets:

- “Dust” which is required to provide some properties and constants (e.g. density and Hamaker constant) that depend on the composition of airborne dust, and to describe the airborne dust size distribution (usually assumed from literature, e.g. [9]).
- “Weather” which has columns with the following headers:
    - Time (required), a datetime in dd/mm/yyyy HH:MM format;
    - AirTemp (required), a float of the air temperature. Units are °C;
    - WindSpeed (required), a float of the wind speed. Units are m/s;
    - TSP/PM_X (required), a float of airborne dust concentration as TSP or PM_X. Units are $\mu g/m^3$;
    - DNI (optional), a float representing the Direct Normal Irradiation. Units are W/m^2;
    - RainIntensity (optional), a float representing the rain intensity. Units are mm/hr.

The next sheets are required for the fitting_experiment class only:

- "Tilts" which has n+1 columns for n mirrors. The first column is Time in the same datetime format as the "Weather" tab and columns 2 to n+1 are with headers Mirror_x, with x=1,2,...,n which contain the tilts in degrees (repeated identically for each time step). This sheet is required for the fitting_experiment class, since the field_model class computes the tilts via the tracking requirement in the helios_angles method.
- "Reflectance_Average": the first column is again Time in datetime format as above, followed by one column for each mirror with the average of reflectance measurements at each time. Time in this tab corresponds to the reflectance measurement events.
- "Reflectance_Sigma": the tab has the same structure of Reflectance_Average but reports the standard deviation of the reflectance measurements at each time.

The model parameters file contains values for parameters related to the site, the plant design, the heliostat characteristics, and a few constants. However, only a limited subset of these values is required for the fitting_experiments class, and hence a tailored Excel sheet is provided for the related notebook. The solar field file needs to be a .xslx or a .csv document, where the heliostats are identified by a x-coordinate and y-coordinate. The column headers are Loc. X and Loc. Y, respectively. Examples of the format of these sheets can be found in the data/public folder.

3. Main Functionalities

The library is developed to perform three main tasks: 1) simulate heliostats soiling and compute reflectance losses; 2) analyze reflectance data from experiments and fit the soiling model's lone parameter; and 3) assess and optimize cleaning strategies to minimize the TCC (sum of direct cleaning costs and soiling-induced revenue losses [3]). Three notebooks have been created to guide the user through the main functionalities of the library, and they will be described in detail in this section through ad-hoc case studies.

3.1 Solar Field Soiling Simulation

The first case study is designed to import all the required parameters, compute optical efficiency and reflectance losses for each sector of the solar field, and test a simple cleaning strategy on an annual simulation. Since this case study spans some of the main objectives of the library, the script developed to achieve such tasks will be described in detail below, including the most relevant lines of the code. The first lines define the data folder, the files to be imported, and four parameters required to decide the number of sector (n_az and n_rad, representative of the number of azimuthal and radial sectorizations, respectively), the number of cleaning trucks (n_trucks*)* and the number of annual cleans of the whole solar field (n_cleans). Moreover, the Python modules described in Section 2 are imported in the notebook assigning to each of them an appropriate shortcut. The following lines instantiate the model imodel, which also reads the parameter file, the solar field one, and performs the sectorization. The input data (airborne dust concentration and weather) are also subsequently imported in the sim_data module instance. A plant instance is created to import the plant design parameters. Eventually, the sector_plot function (not shown) is used to plot the solar field and its sectors, with the corresponding representative heliostats, as depicted in Figure 1, left.

```
import soiling_model.base_models as smb
import soiling_model.utilities as smu
imodel = smb.field_model(file_params,file_SF,num_sectors=(n_az,n_rad))
sim_data = smb.simulation_inputs(file_weather,dust_types="PM10")
plant = smb.plant()
plant.import_plant(file_params)
```

Next, the relevant sun angles (azimuth and zenith) that are required to evaluate the movement of the heliostats are computed (depending on plant design and time of the day) via the sun_angles and helios_angles functions. The following lines of the script apply the steps that define the soiling process as described in [8]: computing the amount of dust falling towards the mirrors, assessing the balance between adhesive and removal forces for each particle diameter (assuming that particles that would be removed at night when mirrors are stowed would immediately be removed). Eventually, once the amount of dust particles adhering on the surface of the heliostats has been computed, the second-to-last line computes the area of the heliostats that is affected by the adhering dust particles.

```
imodel.deposition_flux(sim_data)
imodel.adhesion_removal(sim_data)
imodel.calculate_delta_soiled_area(sim_data)
imodel.plot_area_flux(file_weather,airT,windS)
```

A novel feature of the current library is the possibility to plot the deposition flux area for a given pair of air temperature and wind speed, through the last line of code reported above, whose output is depicted in Figure 1, right. The user can modify wind speed and air temperature to simulate different environmental conditions, but also modify the assumed airborne dust size distribution in the weather file to obtain results for different scenarios.

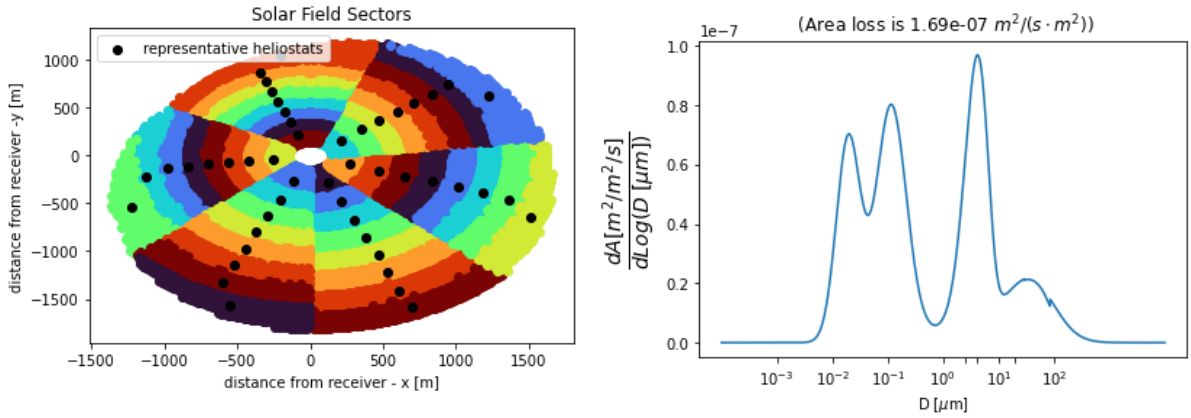

Figure 1. Solar field sectors and representative heliostats (left) ; Area loss rate for given airborne dust size distribution at wind speed = 10 m/s and air temperature = 20°C (right).

Once the amount of dust covering the surface of the heliostats has been computed, a simple cleaning schedule specified by the previously assigned n_trucks and n_cleans is applied following the approach described in [3]. The notebook then initiates a cleaning schedule model and subsequently computes the reflectance losses due to the defined cleaning regime. Eventually, the soiling factor $f_{soil}(t)$, defined as the ratio between actual (current at time t) and nominal (in clean conditions) reflectance of the heliostats, is computed for each sector, and an overall solar field average is provided. The visual representation of the average soiling factor is depicted in Figure 2. It is remarkable to observe the high-frequency oscillations due to the angular dependence of soiling losses throughout the day, as described in a previous work [8]: the relative movement of sun and heliostats causes a continuous variation of both the shade that dust particles cast on the heliostats' surface and the area where sun beams are reflected but subsequently blocked by impact with dust particles. These effects are usually larger during mornings and evenings, causing the oscillations observable in Figure 2.

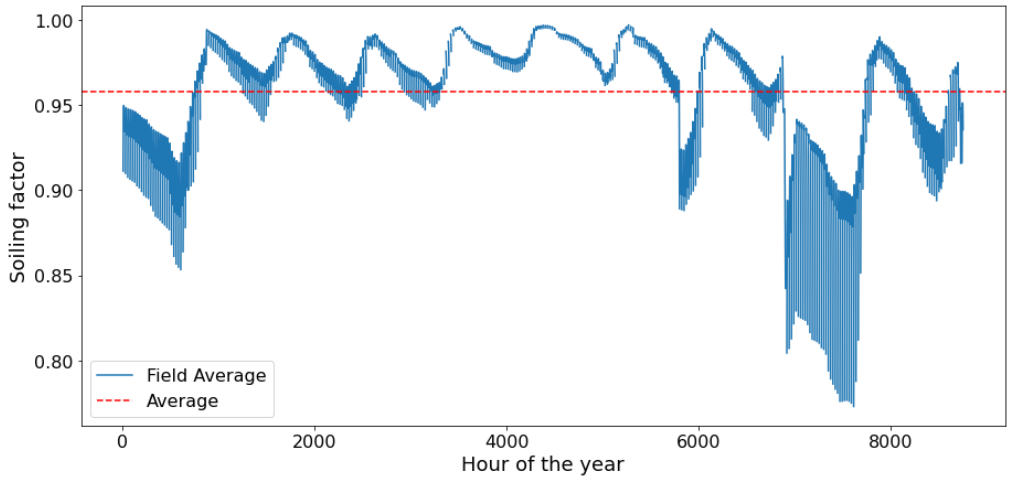

Figure 2. Average soiling factor with 10 annual cleanings and 4 trucks.

Of course, the heliostats of a solar field move differently according to their position with respect to the tower and their aiming point, thus affecting their optical efficiency, which is further affected by their location around the field. To compute the optical efficiency of the heliostats, the HelioSoil library exploits SolarPILOT's in-built algorithms through its Python API CoPylot [10]. The optical_efficiency function of the field_model class in the base_models Python module assigns the parameters required by CoPylot to set the plant simulation. A look-up table is created with an assigned number of solar azimuth (n_az) and elevation (n_el) discretization, through which the script computes the optical efficiency for each heliostat at each time. Sector-averaged values are then obtained, and then used to compute a value for the whole solar field. Combining the calculated soiling factors and optical efficiency it is possible to compute the optical efficiency of the whole solar field and its sectors in soiled conditions. Figure 3, left, shows the computed optical efficiency for the whole year, both with an always-clean field (blue lines) and including the effects of soiling (orange lines). Figure 3, right, depicts the daily behavior of the optical efficiency, whose variation throughout the day is emphasized by the impact of blocking and shading due to dust particles adhering on the heliostats' surface [3], [7], [8].

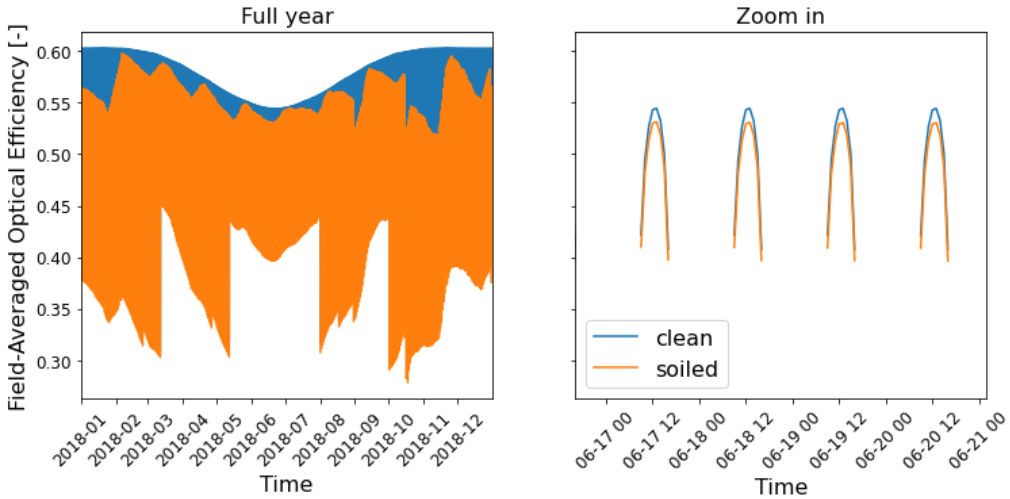

Figure 3. Comparison of solar field optical efficiency in clean and soiled conditions.

3.2 Model Fitting

One of the main inputs declared in the `parameters` file is the value of hrz0. It is the only free parameter of the model that needs to be tuned for accurate soiling predictions [8]. A notebook is provided to guide the user through the fitting procedure from experimental data. The script

first reads the input reflectance data collected and the main experimental parameters. Some of these are specific to the devices (e.g. k-factor and incidence angle) used to collect the measurements, both for reflectance and weather data, and the user should take care when performing analysis on data that are not already provided within the library folders. The data are then divided between training data and test data. Different to the previous case study, the script creates an instance of the fitting_experiment module and reshapes the input data in the required format, through the following lines of code.

```
imodel = smb.fitting_experiment(parameter_file)
hrz0_multi,sse_multi =
imodel.fit_hrz0_least_squares(sim_data_train,reflect_data_train)
```

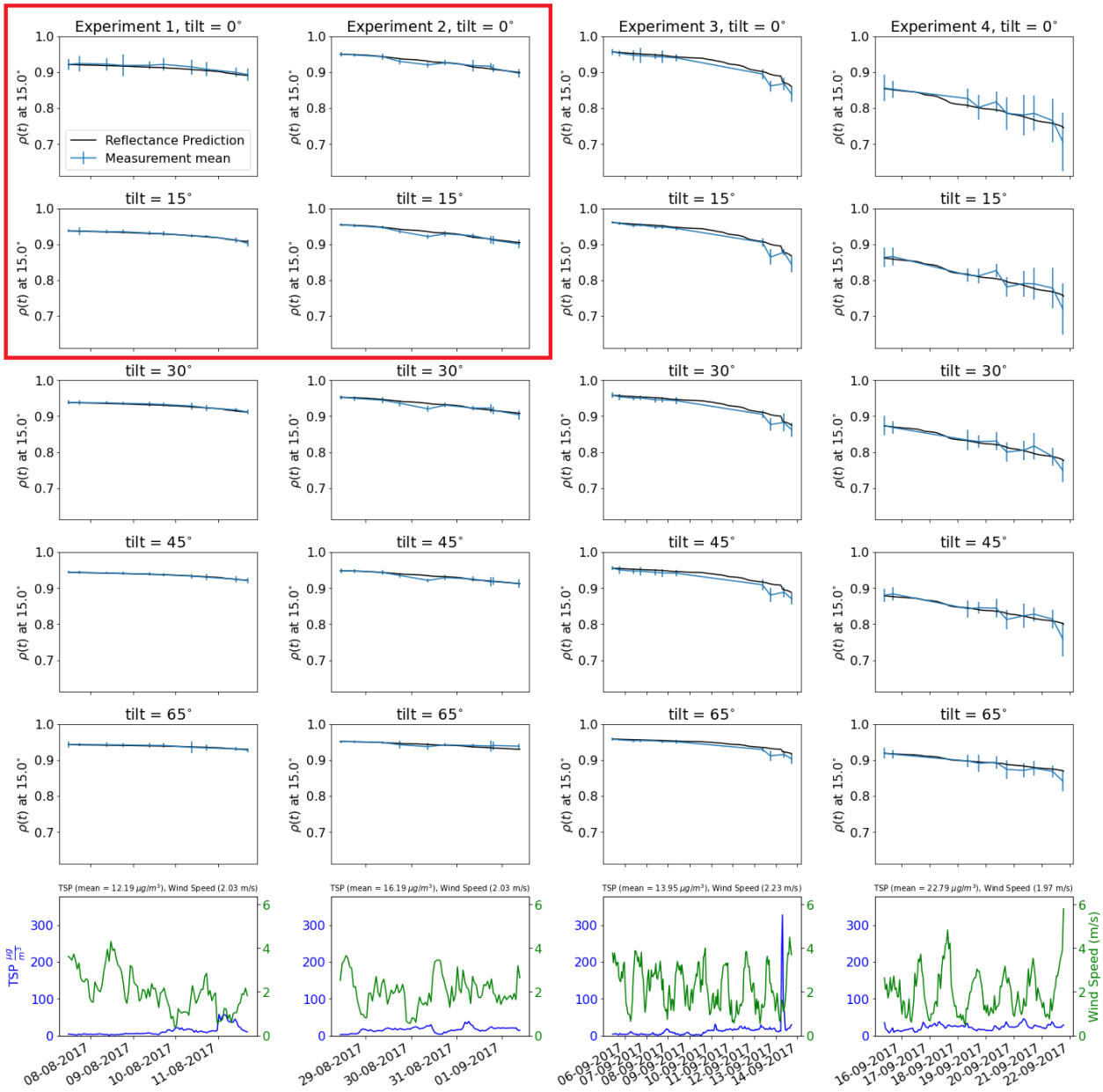

Figure 4. Reflectance prediction after fitting procedure. Training data are shown inside the red rectangle.

Once the data is imported, the fit_hrz0_least_squares function is exploited to identify the values of hrz0 that best fit the training data, highlighted in the red rectangle in Figure 4. The outcomes of the performed analysis suggest that one or two experimental campaigns suffice to provide enough inputs for the fitting procedure and guarantee good agreement with the

measured values. The least tilted mirrors are exploited for the fitting, as they suffer the highest losses and offer the most reliable assessment of dust deposition in the area.

Eventually, the model is updated with the identified value of hrz0 and the reflectance trends are predicted for the whole set of experiments. Figure 4 shows the results obtained for the data collected on the roof of a building at QUT (Queensland University of Technology, Brisbane, Australia), previously discussed in [8].

3.3 Cleaning Optimization and Integration with SolarTherm

The last case study deals with the optimization of the cleaning schedule and resources used for the given solar field and simulated soiling-induced reflectance losses. After the usual definition of data files and main parameters, the script instantiates the optimization problem through the dedicated class in the cleaning_optimization.py module. Subsequently, the model requires manual input (default values are already given) of expected costs and electricity prices to properly compute revenues and expenses for each scenario. The script then performs a grid search on number of trucks and annual cleans to identify the optimal cleaning resources and frequency. Eventually, the TCC for each combination of number of trucks and number of annual cleans is provided, allowing the user to choose the most convenient one. Figure 5, left, shows the test case used in the notebook, where the optimal is given by 2 trucks cleaning 15 times per year. However, the “bumpy” behavior of the curve suggests that this is highly dependent on the timing of high-dust events (or storms) with respect to the cleaning schedule. This is a known issue in cleaning optimization [11], and the limited cost reduction due to a lower number of trucks to be purchased may not compensate the higher risk.

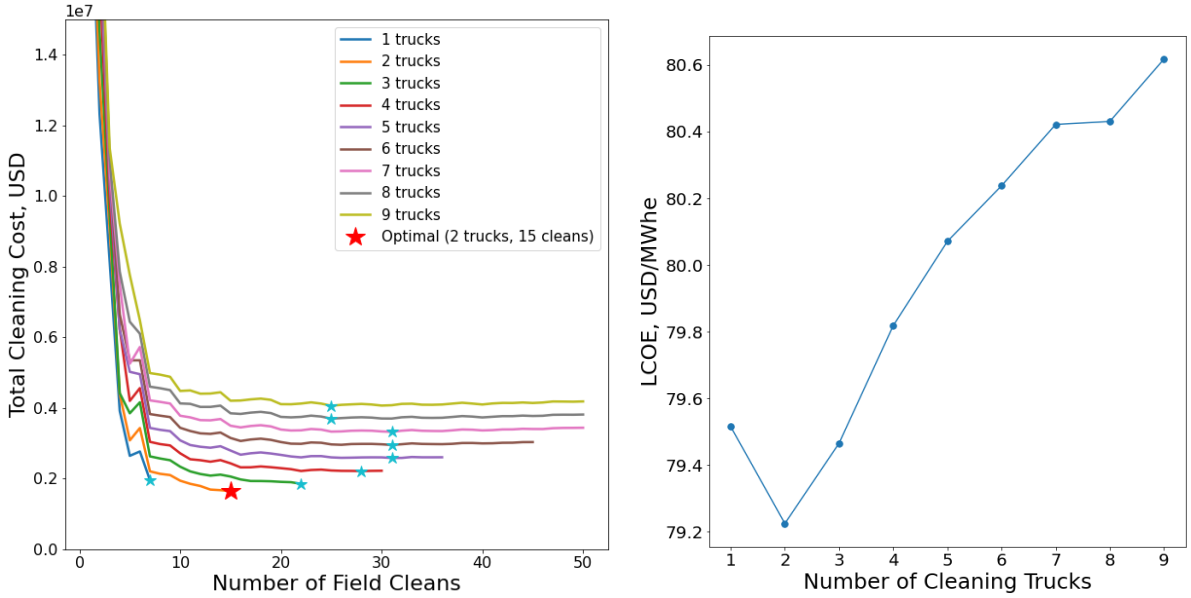

Figure 5. Optimal cleaning schedule and resource assessment (left); Impact of cleaning schedule on LCOE of a CSP system (right).

To further evaluate the costs and benefits of mirror cleaning strategies on a whole CSP system basis, HelioSoil is integrated with the Na-PB-SCO2 package in SolarTherm [12]. The hour-by-hour average soiling factor is loaded into the SolarTherm model and becomes an extra multiplier next to the field optical efficiency. The annual energy output is obtained by a full-year simulation with a detailed control strategy that coordinates the system operation, including receiver start-up, heliostat aiming strategy and defocus, safety considerations, power block start-up and shut-down, energy storage and dispatch. The system model is also augmented with a calculation for the cost of cleaning. The results of the levelized cost of electricity (LCOE) for the cases of 1–9 trucks with the previously identified optimal number of field cleans are shown

in Figure 5, right. In this system analysis model the optimal cleaning strategy is again 2 trucks, however, 3 trucks would be preferred to 1 truck only (which is different from the analysis performed with HelioSoil considering only cleaning-related costs). This shows how the integration between the two packages could provide more informed decisions for CSP plant operators and will be the objective of future work.

4. Conclusion

This paper describes the newly developed and publicly available Python library HelioSoil (https://github.com/cholette/HelioSoil, LGPL license). The library enables the assessment of soiling-induced reflectance losses for a given heliostat(s) or an entire solar field (divided in a finite number of sectors), and subsequently establishes an optimal cleaning scheduling and resourcing (i.e., number of cleaning trucks required and cleanings per annum). The library is also equipped with a module that can import experimentally collected reflectance data with related weather data and fit the free parameter of the soiling model for accurate predictions. Thus, the model can be calibrated for a known location with a limited amount of experimental data, and subsequently exploited for whole plant simulations and economics estimates (e.g. exploiting available packages like SolarTherm). Future work will aim at making the system integration more automatic and robust, and applicable to a wide range of CSP configurations. The HelioSoil library is the first complete package available open source for assessment of soiling impact on heliostats and subsequent cleaning optimization. Its licensing and scripting language choices make it reusable and integrable with any higher level CSP simulation software, thus providing the CSP community with a reliable tool that can promote more accurate O&M cost estimations and hence facilitate the financing and deployment of solar tower plants.

Data availability statement

Data used in this paper are available at:
https://github.com/cholette/HelioSoil/tree/main/data/public

Author contributions

G. Picotti: Conceptualization, Methodology, Software, Writing; M.E. Cholette: Conceptualization, Methodology, Software, Writing, Supervision; Y. Wang: Software, Writing; C.B. Anderson: Methodology, Software; T.A. Steinberg: Resources, Supervision; J. Pye: Resources, Supervision; G. Manzolini: Writing, Supervision.

Competing interests

The authors declare no competing interests.

Funding

The authors acknowledge the support of the Australian Government for this study, through the Australian Renewable Energy Agency (ARENA) within the framework of the Australian Solar Thermal Research Institute (ASTRI – Projects ID P54 and P61)

References

1. M., Blair, N., Diorio, N., Freeman, J., Gilman, P., Janzou, S., Neises, T., Wagner, M., 2018. System Advisor Model (SAM) General Description System Advisor Model.
2. F. Holmgren, W., W. Hansen, C., A. Mikofski, M., 2018. Pvlib Python: a Python

Package for Modeling Solar Energy Systems. J. Open Source Softw. 3, 884. https://doi.org/10.21105/joss.00884
3. Picotti, G., Moretti, L., Cholette, M.E., Binotti, M., Simonetti, R., Martelli, E., Steinberg, T., Manzolini, G., 2020. Optimization of cleaning strategies for heliostat fields in solar tower plants. Sol. Energy 204, 501–514.https://doi.org/10.1016/j.solener.2020.04.032.
4. Truong-Ba, H., Cholette, M.E., Picotti, G., Steinberg, T.A., Manzolini, G., 2020. Sectorial reflectance-based cleaning policy of heliostats for Solar Tower power plants. Renew. Energy 166, 176–189. https://doi.org/10.1016/j.renene.2020.11.129
5. Wales, J.G., Zolan, A.J., Newman, A.M., Wagner, M.J., 2021. Optimizing vehicle fleet and assignment for concentrating solar power plant heliostat washing. IISE Trans. 0, 1–13. https://doi.org/10.1080/24725854.2021.1966858.
6. Wolfertstetter, F., Wilbert, S., Dersch, J., Dieckmann, S., Pitz-Paal, R., Ghennioui, A., 2018. Integration of Soiling-Rate Measurements and Cleaning Strategies in Yield Analysis of Parabolic Trough Plants. J. Sol. Energy Eng. 140. https://doi.org/10.1115/1.4039631
7. Picotti, G., Binotti, M., Cholette, M.E., Borghesani, P., Manzolini, G., Steinberg, T., 2019. Modelling the soiling of heliostats: Assessment of the optical efficiency and impact of cleaning operations. AIP Conf. Proc. 2126. https://doi.org/10.1063/1.5117555.
8. Picotti, G., Borghesani, P., Manzolini, G., Cholette, M.E., Wang, R., 2018. Development and Experimental Validation of a Physical Model for the Soiling of Mirrors for CSP Industry Applications. Sol. Energy 173, 1287–1305. https://doi.org/10.1016/j.solener.2018.08.066.
9. Seinfield, J.H., Spyros, P.N., 2016. Atmospheric Chemistry and Physics: From Air Pollution to Climate Change, 3rd ed. John Wiley & Sons, Ltd. https://doi.org/10.1080/00139157.1999.10544295.
10. Hamilton, W.T., Wagner, M.J., Zolan, A.J., 2021. Demonstrating SolarPILOT's python API through heliostat optimal aimpoint strategy use case. Proc. ASME 2021 15th Int. Conf. Energy Sustain. ES 2021. https://doi.org/10.1115/ES2021-60502.
11. Zolan, A., Mehos, M., 2020. Wash Vehicle Fleet Sizing for Contingency Planning Against Dust Storms. Sol. Paces 2020. https://doi.org/10.1063/5.0085675.
12. Scott, P., Alonso, A.D.L.C., Hinkley, J.T., Pye, J., 2017. SolarTherm: A flexible Modelica-based simulator for CSP systems. AIP Conf. Proc. 1850. https://doi.org/10.1063/1.4984560