S.H.A.R.D. documentation

Afghan Goat

September 2026

Contents

1 Introduction

S.H.A.R.D. (Strike Hazard And Radius Designation) is a drone/rocket strike analysis tool which can help locating the exact landing point based on wind data and determining collateral damage and exclusion areas before strike.

2 Requirements and dependencies

The project requires Python with version >= 3.12 and the following libraries:

- matplotlib
- PyYAML
- plotly
- pylatex
- kaleido
- pandas
- shapely
- numpy
- rocketpy
- ipython

There are no other external simulation dependencies as the simulation logic is unique and is strengthtened by the RocketPy library.

Pdflatex is also needed on Linux. On Windows, you also need that too.

3 Technologies used

1. **Python version** >= 3.12 Which allowed us to make it cross platform.
2. **PyYAML** Which is a YAML parser and was used in reading the config values in a robust manner.
3. **plotly** Which is an open-source graphing library and was used for data plots such as the static margin plot over time.
4. **pylatex** Which was used for creating and compiling latex documents under python.
5. **kaleido** Which was used for generating static images of the flight scenarios and statistics.

6. **pandas** Is a fast, robust and open-source data analysis tool for python. This was used for the main data processing.
7. **shapely** Which was used for shape analysis of the landing points and the bounds of those.
8. **numpy** Which offers comprehensible math functions as extension.
9. **rocketpy** Which is used for the core per rocket flight simulation logic.
10. **ipython** Which can be used for robust shell command handling and offloading.
11. **Pdflatex** Which can be used to compile the generated latex report to a single PDF file.

4 Installation

We suggest that if possible, the Python packages should be installed to a brand new virtual environment.

For installing the required libraries, run:
python3 -m pip install -r requirements.txt

For the POI detection, some other libraries are also needed:
pip install numpy opencv-python-headless matplotlib PyYAML

Pdflatex and Texlive packages are also required, they can be installed by the following commands on a Linux system which uses aptitude:
sudo apt install texlive-full

After that, the program is usable and runnable.

5 Configuration

The Range Safety simulation framework relies on structured YAML configuration files to define environmental settings, geographical constraints, physical rocket characteristics, Monte Carlo execution parameters, and output visualizer settings. This section details the primary configuration files.

5.1 Environment Settings (`env_settings.yaml`)

The `env_settings.yaml` file serves as the root configuration for runtime file paths, global physical/geographical constants, site data sources, and safety region boundary definitions.

- **Global Execution & Template Paths:**
  - `current_config`: Specifies the primary vehicle configuration YAML file to load (e.g., `"config_new.yaml"`).

- `montecarlo_sim_path`: Output directory path where Monte Carlo trajectory simulation runs are stored (e.g., `"./montecarlo/"`).
- `latex_path`: Path to the directory containing LaTeX report preambles and section templates (e.g., `"./latex_template/"`).
- `wind_chart_path`: Location of the generated maximum allowable wind sectors configuration (e.g., `"./wind_max_output/wind_max_sectors.yaml"`).
- `merged_image_destination_path`: Target directory for merged safety boundary plots (e.g., `"./merger_results"`).

- **Physical Constants & Geo-location:**
  - `CENTER_LAT`, `CENTER_LON`: Launch site reference coordinates in decimal degrees (e.g., `47.118611`, `19.333889`).
  - `R`: Mean radius of the Earth in meters based on the WGS84 spheroid (`6378137.0` m).
  - `MIN_THRESHOLD`: Minimum numerical precision threshold to prevent floating-point division by zero in distance calculations (`1.0e-12`).
  - `MAX_EXPECTED_HEIGHT`: Upper bound ceiling altitude for spatial discretization (`3000` m).

- **Target Site Data Files:**
  - `MAP_IMAGE_DATA`: Path to the background aerial raster map image (e.g., `"./DATA/loter2.png"`).
  - `SITE_BOUNDARY_DATA`: Path to the CSV file defining outer range boundary polygon coordinates (e.g., `"./DATA/lat_lon_data.csv"`).

- **Custom Boundaries List (`boundaries`):** Defines explicit spherical/circular protection or target zones. Each entry contains:
  - `name`: Identifier string for the region (e.g., `"viewpoint"`).
  - `x`, `y`: Local Cartesian coordinates (in meters) relative to the launch origin.
  - `radius`: Protection zone radius in meters (e.g., `500.0` m).
  - `worth`: Strategic weighting score. A negative score (e.g., `-100`) indicates a critical safety hazard zone that triggers an immediate stop sequence if landing is detected within it.

5.2 POIs & Safety Boundaries (`boundaries_out.yaml`)

The `example_boundaries_out.yaml` file contains exported or generated lists of Points of Interest (POIs) and safety boundary definitions evaluated during range safety spatial analysis.

- `name`: Identifier of the point of interest (e.g., `important_POI_0`, `potential_POI_16`).
- `x`, `y`: 2D Cartesian coordinates in meters relative to the launch site center.
- `radius`: Safety keep-out or evaluation radius surrounding the asset in meters.
- `worth`: Numerical importance/weight score. Values above `100000` denote critical infrastructure assets, whereas lower positive scores (e.g., `7200`) signify secondary potential target points.

5.3 Main Rocket & Flight Configuration (`config.yaml`)

The `config.yaml` file is the core specification file governing rocket physical properties, solid rocket motor parameters, aerodynamic geometry, parachute recovery systems, flight launch conditions, Monte Carlo execution parameters, and output plot settings.

5.3.1 Basic Settings & Toggles (`basic_settings`)

Controls global simulation features and output generation flags:

- `main_parachute`, `drogue_parachute`: Boolean toggles (`True`/`False`) enabling recovery event simulations.
- `weather_custom`: Set to `True` for custom wind profiles, or `False` when importing global weather forecast datasets (e.g., GFS).
- `outputs`: Sub-dictionary enabling specific plotting outputs, including `rocket_draw`, `rocket_static_margin`, `flight_out_of_rail_conditions`, `flight_apogee_conditions`, `flight_trajectory_3d`, and `environment_info`.
- `distribution_config`: Path to the statistical dispersion distribution definition file (e.g., `distribution_1.yaml`).

5.3.2 Propulsion System (`motor`)

Defines solid rocket motor parameters used by RocketPy:

- `filename`: Path to the OpenRocket / RASAero `.eng` motor thrust curve file (e.g., `"DATA/TMK2_SFT_26_05_29.eng"`).
- `dry_mass`, `dry_inertia_i11`, `dry_inertia_i22`, `dry_inertia_i33`: Motor dry mass (kg) and principal moments of inertia ($\mathrm{kg} \cdot \mathrm{m}^2$).
- `burn_time`, `impulse`: Motor total burn duration (s) and total impulse (Ns).

- `grain_number`, `grain_density`, `grain_outer_radius`, `grain_initial_inner_radius`, `grain_initial_height`, `grain_separation`: Propellant grain geometric and physical specifications.
- `nozzle_radius`, `throat_radius`, `nozzle_position`, `center_of_dry_mass_position`, `grains_center_of_mass_position`: Position offsets and nozzle geometry relative to the motor reference frame.

5.3.3 Rocket Airframe Properties (`rocket`)

Specifies dry structural properties and aerodynamic drag profiles:

- `radius`, `mass`: Airframe outer radius (m) and total structural dry mass (kg).
- `inertia_i11`, `inertia_i22`, `inertia_i33`: Airframe rotational moments of inertia ($\text{kg} \cdot \text{m}^2$).
- `center_of_mass_without_motor`: Center of mass location of the dry airframe (m).
- `power_off_drag`, `power_on_drag`: Paths to CSV files containing Drag Coefficient (C_d) curves vs. Mach number under power-off and power-on flight phases.
- `coordinate_system_orientation`: Orientation reference convention (e.g., `"tail_to_nose"`).

5.3.4 Airframe Components (`button`, `nose`, `fin`, `tail`)

Geometric parameters for stability calculations:

- `button`: Launch rail button locations (`upper_button_position`, `lower_button_position`) and angular alignment (`angular_position`).
- `nose`: Nosecone profile parameters (`length`, `position`, `bluffness`, and shape `kind` such as `"Von Karman"`).
- `fin`: Fin set geometry including fin count `n`, `root_chord`, `tip_chord`, `span`, `position`, `cant_angle`, and `sweep_length`.
- `tail`: Boattail transition parameters (`top_radius`, `bottom_radius`, `length`, `position`).

5.3.5 Recovery Systems (`drogue` & `main`)

- `name`: Recovery stage label (`"Drogue"` / `"Main"`).
- `cd_s`: Effective drag area product C_dS (m^2).

- `trigger`: Event deployment trigger (`"apogee"` for drogue deployment, or numerical altitude in meters like `450` for main deployment).
- `lag`: Deployment delay time in seconds (e.g., `1.5`).

5.3.6 Flight & Launch Conditions (`flight`)

- `direction`: Launch rail azimuth heading in degrees from North ($0° - 360°$).
- `inclination`: Launch rail elevation angle in degrees relative to horizontal.
- `rail_length`: Launch guide rail length in meters (e.g., `12` m).
- `latitude`, `longitude`, `elevation`, `timezone`, `date_*`: Launch site coordinates and timestamp settings.
- `atmosphere_type`, `atmosphere_file`: Atmospheric profile source specification.
- `inclination_values`, `wind_direction_values`, `wind_velocity_values`: Parameter arrays for multi-run sensitivity and wind sector sweep analyses.

5.3.7 Monte Carlo Simulation Profiles (`nominal`, `ballistic`, `wcs`)

Configures simulation scenarios for dispersion analysis:

- `file_path`, `filename`, `description`: Storage locations and scenario labels for Nominal, Ballistic (un-parachuted), and Worst Case Stability (WCS) simulation runs.
- `simulated_points`: Number of Monte Carlo iterations to execute (e.g., `100` or `400`).
- `filenames`: Specific output names for generated summary CSV files (`summary_csv`) and scatter plots (`landing_points_plot`, `range_safety_plot`).

5.3.8 Global Enums & Plot Bounds (`filenames`, `plot_settings`)

- `filenames`: Suffixes and file paths for dispersion output text files, site CSV coordinates, and background maps.
- `plot_settings`: Defines display origins (`launch_point_x/y`, `view_point_x/y`) and axis limit ranges (`xlim_min`, `xlim_max`, `ylim_min`, `ylim_max`) for standardized visualizer rendering.

6 Usage

The following section covers the order of usage of the software to generate all parts.

6.1 Montecarlo simulation

First of all, run the predefined scenarios or if needed, make new scenarios. This can be achieved by using the old montecarlo files and example, copying them and modifying them. After the configuration is done, run the files, for example, this runs the GFS and our nominal wind model's WCS, Nominal and Ballistic scenarios:

- `python3 ./montecarlo_wcs.py`
- `python3 ./montecarlo_wcs_gfs.py`
- `python3 ./montecarlo_ballistic.py`
- `python3 ./montecarlo_ballistic_gfs.py`
- `python3 ./montecarlo_nominal.py`
- `python3 ./montecarlo_nominal_gfs.py`

After this, run the shape merger tool, which will generate a global boundary composite shape which includes all of the scenarios. Run `python3 ./mon_shape_merger_old.py`.

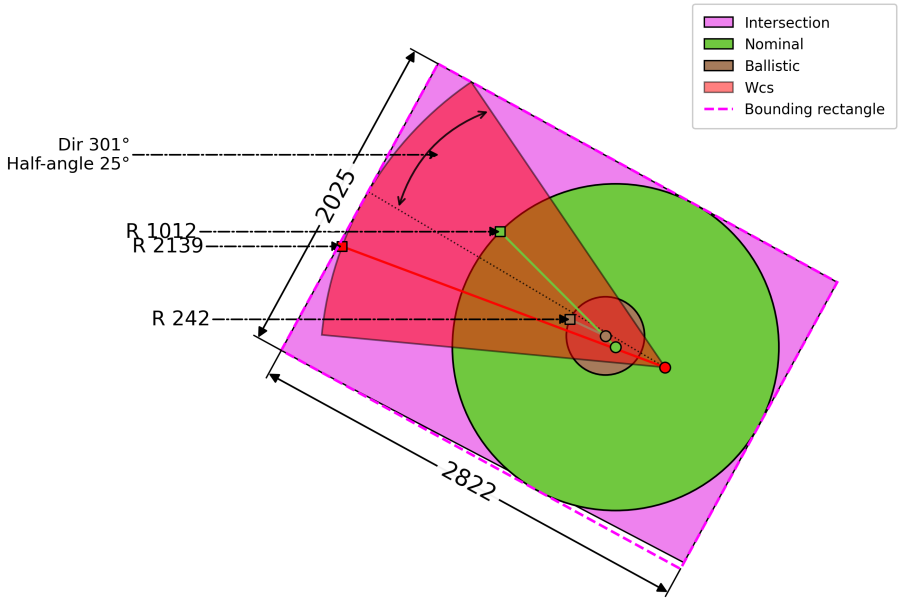

Figure 1: An example merged shape file. Will be used for further analysis and documentation purposes.

6.2 Wind maximalization

This consists of two parts. The first part checks the quasi-maximum wind velocity for each direction in the script using a multiplicative then additive continous montecarlo batching. To start this, run:
`python3 ./montecarlo_wind_maxxer.py`
NOTE: This may take hours to run through based on configuration. After this, a wind model can be generated by running `python3 ./wind_visualizer.py`.

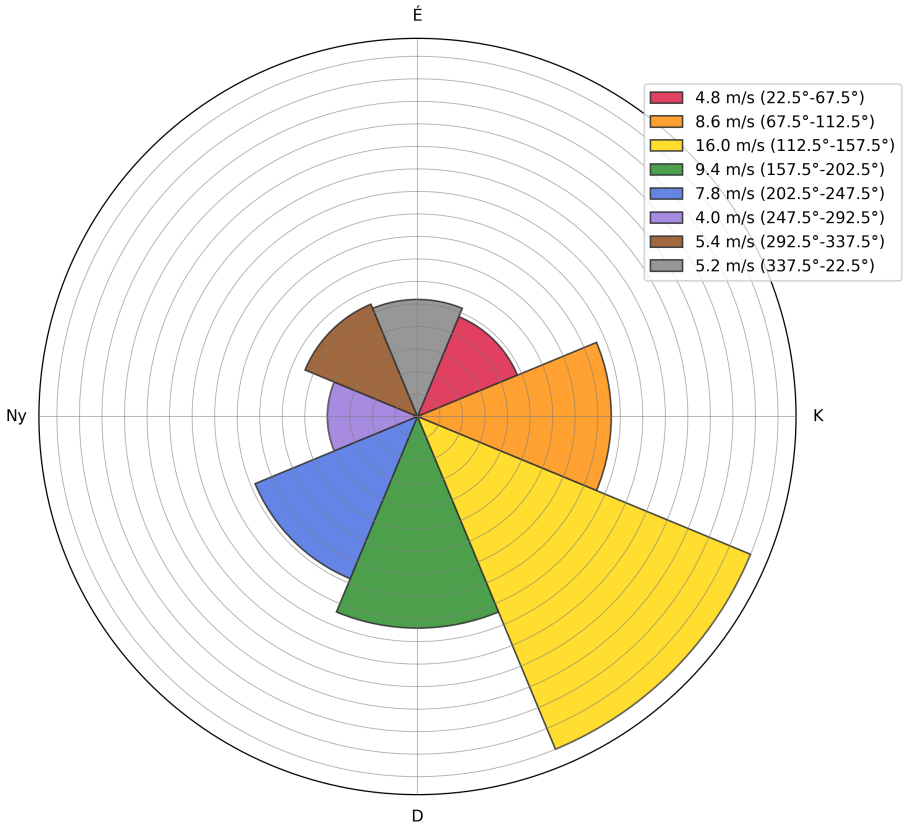

Figure 2: The wind direction model graph generated by the last script.

6.3 POI exclusion

The user can specify Point of Interests with score which can determine which areas are desirable as rocket landing and which are the exclusion zones. There is a system which can automatically identify POIs from satellite images and can

Figure 3: The program will also output a visual image analysis for users to determine the targets visually.

approximate the heights of the POIs.

POIs can be set manually in the `env_settings.yaml` under the boundaries section however, you can generate a `boundaries_out.yaml` which utilizes computer vision to identify critical infrastructure from satellite images. For this, run:

```
python detect_POS.py image.jpg --meters-per-pixel 0.15
```

Where the correct image and scaling needs to be set. This will output a boundaries_out.yaml file which you can copy into the boundaries section of the environment config file.

Now when running the wind maximalizer script, it will output a directional hazard and risk analysis into the wind_max_output folder. An example hazard analysis will be shown in the next section which will only contain the viewpoint as an exclusion zone.

6.4 Hazard and Boundary Scoring Analysis

6.4.1 Simulation Parameters

- **Optimal Wind Condition Tested:** 9.4 m/s @ 180°
- **Launch Coordinates:** 47.118611° N, 19.333889° E

- **Max Expected Height:** 3000 m
- **Simulated Points (N):** 50
- **Statistical Confidence Level:** 0 (Low Confidence / Preliminary)

Note: Zones assigned a negative worth represent restricted safety hazards. Any impact inside these boundaries triggers an instant mission safety failure. Positive worth denotes target zones.

Zone Name	Impacts Inside	Worth per Landing	Total Score
viewpoint	0	-100	0
Overall Simulation Score			**0**

6.5 Summary and analysis generation

If the documentation requires details about the ballistic object in question, run `python3 misc_exporter.py` because this generates the static margin plots, weight and other object related plots.

After all of the previous sections were ran, a summary script can be generated: `python3 doc_gen_new.py` which after some time, outputs the generated PDF into the generated_pdfs folder.

6.6 Variables Over Latex guide

VOL is a templating schema which can be used to add dynamic variables and paths to ordinary latex files. By default, you should edit the latex files located in the latex_template folder but this can be changed in the script file. VOL is a superset of latex and uses Pdflatex and Pylatex as latex engines. The main rationale for making a system such as this is to allow versioned file includes and easy csv,yaml,json key/value or row/column includes. VOL commands start and end with the |?| character sequence. Between that, a robust value import system is implemented. An example command is: $|?|montecarlo/ * nominal * /[LATEST]/montecarlo\_nominal.csv : max\_speed : mean : .3f|hu|?|$ Which means, that get the latest version of ANY nominal run's montecarlo_nominal.csv's max_speed row's mean value to 3 decimal points in Hungarian notation.

Table 1: Variables Over Latex (VOL) Command Summary

VOL Command Example	**Description / Output**
`|?|montecarlo/*/[LATEST]/data.csv:metric:mean:.2f|hu|?|`	CSV lookup with 2 decimals in Hungarian notation
`|?|config.yaml:latitude|?|`	Key lookup in specific YAML file
`|?|include_raw_tex:wind_images/wind_table.tex|?|`	Raw TeX snippet inclusion without placeholder parsing
`|?|montecarlo/*nominal*/[LATEST]/plot.png|?|`	Auto-generates `\includegraphics` for resolved image
`|?|path/to/image.png:path:V([0-9.]+)|?|`	Extracts regex pattern from resolved file path string
`|?|current_config|?|`	Direct key lookup in default main config
`|?|data.csv:metric:col:.3f|abs|hu|?|`	Takes absolute value, 3 decimals with Hungarian comma
`|?|settings.json:key:.2f|?|`	Formatted key lookup in JSON file