

PyMultiDIC Usage Manual

Detailed API inputs, native C++ local build, workflow composition, and output locations

1. Overview

PyMultiDIC exposes a Python API for the multi-view DIC workflow. Users call explicit functions named `pymultidic.<function_name>`. The API supports YAML-driven configuration and direct function inputs.

If an `MDICConfig` object is supplied, it has priority and later keyword inputs are ignored. If `config=None`, `case_root` is required and all other values come from defaults unless overridden.

2. Installing and Calling the Published Package

```
# Install the released package from PyPI
pip install pymultidic

# Configuration-file mode
import pymultidic
config = pymultidic.load_config("configs/MDIC.yaml")
report = pymultidic.run_pipeline(config)

# Direct-input mode
report = pymultidic.run_pipeline(
    case_root="case/CylinderDIC",
    steps=["validate", "sfm", "scale", "mask", "dic2d", "recon3d", "visualize3d"],
    subset_radius=25,
    subset_spacing=6,
    min_corrcoef=0.6,
)
```

Use this mode when the package has already been published to PyPI and you want a normal user installation. The package name is `pymultidic` and runtime calls are made with `import pymultidic`.

3. Native C++ Local Build without the Published pip Package

This section is about compiling the native C++ part of the project, not merely installing Python files. The top-level `native/CMakeLists.txt` is intended to build all native components in one configure/build pass: the `ncorr` native library, the `ncorr_cli` executable, and the `native_recon3d` `pybind11` Python extension.

Use this path when developing locally, changing files under `native/`, or validating native builds before publishing wheels. After the native build, install the Python package locally so `pymultidic` can call the compiled native artifacts.

```
# Clone source
git clone https://github.com/lbd-hfut/Multi-DIC.git
cd Multi-DIC

# WSL / Ubuntu / Debian native C++ dependencies
sudo apt-get update
sudo apt-get install -y build-essential cmake ninja-build python3-dev python3-pip
python3 -m pip install -U pybind11 scikit-build-core

# One native CMake configure/build pass from repository root
cmake -S native -B build/wsl-native -G Ninja -DPYBIND11_FINDPYTHON=ON -DPython_EXECUTABLE=/usr/bin/python3 -Dpybind11_DIR=$(python3 -m pybind11 --cmakedir)
cmake --build build/wsl-native

# Expected WSL/Linux outputs
# build/wsl-native/ncorr/libnative_ncorr.a
# build/wsl-native/ncorr/ncorr_cli
# build/wsl-native/recon3d/native_recon3d*.so

# Python local editable install after native build
python3 -m pip install -e .
python3 run.py --config configs/MDIC.yaml
```

Windows and macOS notes

```
# Windows example with CMake + Ninja from a Developer PowerShell
python -m pip install -U pybind11 scikit-build-core cmake ninja
cmake -S native -B build/windows-native -G Ninja -DPYBIND11_FINDPYTHON=ON
cmake --build build/windows-native

# Expected Windows outputs include
```

```
# build/windows-native/ncorr/ncorr_cli.exe
# build/windows-native/recon3d/native_recon3d*.pyd

# macOS/Linux is similar: install cmake, ninja, python-dev headers, pybind11,
# then run cmake -S native -B build/native -G Ninja and cmake --build build/native.
```

4. Input Model and Override Rules

The central object is MDICConfig. It stores project root, data layout, output layout, and raw workflow parameters. Create it using load_config from YAML or build_config from direct Python inputs.

Input group	Direct-mode parameters and defaults
Required case identity	case_root is required when config=None. project_name defaults to PyMultiDIC.
Data layout	output_root="results", speckle_dir="images", calibration_dir="calibrate_images", camera_glob="cam_**", reference_frame="001.bmp", deformed_frames=["002.bmp"].
SfM overrides	colmap_backend, colmap_workspace, reference_camera, camera_model, matcher, use_gpu, max_features, or colmap={...}.
Scale overrides	checkerboard_meta, square_size, square_size_unit, or scale_correction={...}.
DIC2D overrides	subset_radius, subset_spacing, seed_search_radius, rg_search_radius, cutoff_diffnorm, cutoff_iteration, num_threads, subset_truncation, units_per_pixel, cutoff_corrcoef, lenscoef, or dic2d={...}.
Recon3D overrides	recon_backend, min_views, min_corrcoef, max_reprojection_error_px, outlier_filter_enabled, position_mad_z, displacement_mad_z, or recon3d={...}.
Visualization overrides	visualization_output_dir, view_elev, view_azim, dpi, or visualization={...}.
Advanced nested overrides	raw_overrides={...} can update any nested configuration section directly.

5. Public API Reference

5.1 Summary Table

Function	Main role	Typical output
load_config	Load YAML into an MDICConfig object.	MDICConfig
build_config	Create MDICConfig from config or direct inputs.	MDICConfig
validate_project	Check case folders, cameras, images, and output folders.	dict report
run_validate	Pipeline-style alias for validate_project.	dict report
run_sfm	Run SfM/self-calibration and export camera/track data.	SfM report and files
run_scale	Estimate physical scale from checkerboard calibration images.	scale report and files
run_mask	Prepare ROI masks from user masks or automatic logic.	mask report and mask files
run_dic2d	Run 2D DIC for every camera and deformed frame.	DIC2D report and npz files
run_recon3d	Fuse DIC2D and SfM observations into 3D displacement results.	3D reports, npz, ply, pair surfaces
run_visualize3d	Render 3D morphology and displacement cloud maps.	figure paths and report
run_step	Run one named workflow step.	step report
run_pipeline	Run several workflow steps in order.	combined report

5.2 Detailed Function Reference

5.2.1 pymultidic.load_config(config_path, workspace_root=None)

Purpose: Loads a YAML configuration file and returns the MDICConfig object used by all workflow steps.

Parameter	Type	Meaning
config_path	str pathlib.Path	Required YAML path, for example configs/MDIC.yaml.
workspace_root	str pathlib.Path None	Optional base path for relative paths. Defaults to current working directory.

Output: MDICConfig. No computation is executed.

Example:

```
config = pymultidic.load_config("configs/MDIC.yaml")
```

5.2.2 pymultidic.build_config(config=None, *, case_root=None, project_name="PyMultiDIC", output_root="results", speckle_dir="images", calibration_dir="calibrate_images", camera_glob="cam_*", reference_frame="001.bmp", deformed_frames=None, image_extensions=None, workspace_root=None, raw_overrides=None, **overrides)

Purpose: Builds an MDICConfig directly in Python. If config is supplied, it is returned unchanged and all other inputs are ignored.

Parameter	Type	Meaning
config	MDICConfig None	Optional object with highest priority.
case_root	str Path None	Required when config=None. Case folder such as case/CylinderDIC.
project_name	str	Name written to reports.
output_root	str Path	Relative output folder under case_root. Default: results.
speckle_dir	str Path	Relative speckle image folder. Default: images.
calibration_dir	str Path	Relative calibration image folder. Default: calibrate_images.
camera_glob	str	Camera folder glob. Default: cam_*.
reference_frame	str	Reference image filename. Default: 001.bmp.
deformed_frames	list[str] tuple[str,...] None	Deformed image names. Default: ["002.bmp"].
image_extensions	list[str] tuple[str,...] None	Allowed image extensions.
workspace_root	str Path None	Base path for resolving relative paths.
raw_overrides	dict None	Nested configuration override dictionary.
**overrides	Any	Flat or section overrides such as subset_radius=25, min_corrcoef=0.6, dic2d={...}.

Output: MDICConfig ready for all run_* functions.

Example:

```
config = pymultidic.build_config(case_root="case/CylinderDIC", subset_radius=25, min_corrcoef=0.6)
```

5.2.3 pymultidic.validate_project(config=None, **kwargs)

Purpose: Validates the case folder, discovers cameras, checks images, reads image sizes, and creates output folders.

Parameter	Type	Meaning
config	MDICConfig None	Optional. If supplied, kwargs are ignored.
**kwargs	Any	Direct build_config inputs; case_root is required when config=None.

Output: dict with ok, project, case_root, result_root, camera_count, cameras, created_directories, warnings, errors.

Example:

```
report = pymultidic.validate_project(case_root="case/CylinderDIC")
```

5.2.4 pymultidic.run_validate(config=None, **kwargs)

Purpose: Alias for validate_project, keeping the run_* naming style for pipeline code.

Parameter	Type	Meaning
config	MDICConfig None	Optional.
**kwargs	Any	Direct build_config inputs when config=None.

Output: Same report as validate_project.

Example:

```
report = pymultidic.run_validate(case_root="case/CylinderDIC")
```

5.2.5 pymultidic.run_sfm(config=None, **kwargs)

Purpose: Runs SfM/self-calibration with the default pycolmap backend and exports cameras, sparse points, and observations for later steps.

Parameter	Type	Meaning
config	MDICConfig None	Optional.
case_root	str Path	Required in direct mode.
colmap_backend	str	Default: pycolmap.
colmap_workspace	str	Default: colmap.
reference_camera	str	Default: cam_0.
camera_model	str	Default: SIMPLE_RADIAL.
matcher	str	Default: exhaustive.
use_gpu	bool	Default: False.
max_features	int	Default: 8192.
colmap={...}	dict	Nested override for COLMAP/SfM settings.

Output: dict report plus results/logs/sfm_report.json and files under results/sfm/colmap, including cameras.npz, sparse_points.npz, observations.npz.

Example:

```
report = pymultidic.run_sfm(case_root="case/CylinderDIC", use_gpu=False, max_features=8192)
```

5.2.6 pymultidic.run_scale(config=None, **kwargs)

Purpose: Computes SfM-to-world scale using checkerboard calibration images and SfM camera geometry.

Parameter	Type	Meaning
config	MDICConfig None	Optional.
case_root	str Path	Required in direct mode.
checkerboard_meta	str Path	Default: calibrate_images/chessboard_meta.json.
square_size	float	Default: 10.0.
square_size_unit	str	Default: mm.
scale_correction={...}	dict	Nested scale settings.

Output: dict report plus results/logs/scale_report.json and results/scale/sfm2world_scale.json.

Example:

```
report = pymultidic.run_scale(case_root="case/CylinderDIC", square_size=10.0, square_size_unit="mm")
```

5.2.7 pymultidic.run_mask(config=None, **kwargs)

Purpose: Creates ROI masks from user-provided masks or automatic SfM/image texture logic.

Parameter	Type	Meaning
config	MDICConfig None	Optional.
case_root	str Path	Required in direct mode.
mask={...}	dict	Nested settings such as user_mask_dir, output_dir, external_threshold.
raw_overrides	dict	Alternative advanced nested override.

Output: dict report plus results/masks/mask, overlays, debug images, and results/logs/mask_report.json.

Example:

```
report = pymultidic.run_mask(case_root="case/CylinderDIC", mask={"use_user_mask_if_present": True})
```

5.2.8 pymultidic.run_dic2d(config=None, **kwargs)

Purpose: Runs per-camera 2D DIC. The default engine calls the native ncorr CLI and writes one npz per camera/frame.

Parameter	Type	Meaning
config	MDICConfig None	Optional.
case_root	str Path	Required in direct mode.
subset_radius	int	Default: 20.
subset_spacing	int	Default: 5.
seed_search_radius	int	Default: 50.
rg_search_radius	int	Default: 1.
cutoff_diffnorm	float	Default: 1.0e-6.
cutoff_iteration	int	Default: 50.
num_threads	int	Default: 1.
subset_truncation	bool	Default: False.
units_per_pixel	float	Default: 1.0.
cutoff_corrcoef	float	Default: 0.6.
lenscoef	float	Default: 0.0.
dic2d={...}	dict	Nested DIC2D override, including roi/seed/format/strain.

Output: dict report plus results/dic2d/dic2d_<camera>_<frame>.npz and results/logs/dic2d_report.json.

Example:

```
report = pymultidic.run_dic2d(case_root="case/CylinderDIC", subset_radius=25, subset_spacing=6, num_threads=4)
```

5.2.9 pymultidic.run_recon3d(config=None, **kwargs)

Purpose: Reconstructs sparse 3D displacement by fusing SfM observations and DIC2D displacement fields. It also writes pair surface meshes, post-processing results, QC plots, and PLY exports.

Parameter	Type	Meaning
config	MDICConfig None	Optional.
case_root	str Path	Required in direct mode.
recon_backend	str	Default: native.
min_views	int	Default: 2.
min_corrcoef	float	Default: 0.6.
max_reprojection_error_px	float	Default: 2.0.
outlier_filter_enabled	bool	Default: True.
position_mad_z	float	Default: 8.0.
displacement_mad_z	float	Default: 8.0.
recon3d={...}	dict	Nested reconstruction, pair, post3d, QC, export, and outlier settings.

Output: dict report plus results/recon3d/recon3d_<frame>.npz, .ply, pairs/<frame>, post/<frame>, QC outputs, and recon3d_report.json.

Example:

```
report = pymultidic.run_recon3d(case_root="case/CylinderDIC", min_views=2, min_corrcoef=0.6)
```

5.2.10 pymultidic.run_visualize3d(config=None, **kwargs)

Purpose: Renders visual outputs from reconstruction files, including morphology cloud map, total displacement cloud map, and Ux/Uy/Uz component cloud maps.

Parameter	Type	Meaning
config	MDICConfig None	Optional.
case_root	str Path	Required in direct mode.
visualization_output_dir	str	Default: figures.
view_elev	float	Default: 22.0.
view_azim	float	Default: -58.0.
dpi	int	Default: 180.
visualization={...}	dict	Nested visualization settings.

Output: dict report with outputs keys: surface_cloud_morphology, surface_cloud_displacement_total, surface_cloud_displacement_ux, surface_cloud_displacement_uy, surface_cloud_displacement_uz, and more.

Example:

```
report = pymultidic.run_visualize3d(case_root="case/CylinderDIC", view_elev=22, view_azim=-58)
```

5.2.11 pymultidic.run_step(config_or_step=None, step=None, **kwargs)

Purpose: Runs one named step. Useful for notebooks, scripts, and custom control loops.

Parameter	Type	Meaning
config_or_step	MDICConfig str None	Either a config or the step name when no config is supplied.
step	str None	Step name if first argument is a config.
**kwargs	Any	Direct build_config inputs when no config is supplied.

Output: dict report from the selected step.

Example:

```
report = pymultidic.run_step("validate", case_root="case/CylinderDIC")
```

5.2.12 pymultidic.run_pipeline(config=None, steps=None, *, stop_on_error=True, **kwargs)

Purpose: Runs multiple workflow steps in order and returns a combined report.

Parameter	Type	Meaning
config	MDICConfig None	Optional. If supplied, kwargs are ignored.
steps	list[str] tuple[str,...] None	Default: validate, sfm, scale, mask, dic2d, recon3d.
stop_on_error	bool	Default: True.
**kwargs	Any	Direct build_config inputs when config=None.

Output: dict with ok, steps, completed_steps, and reports keyed by step.

Example:

```
report = pymultidic.run_pipeline(case_root="case/CylinderDIC", steps=["validate", "sfm", "scale", "mask", "dic2d", "recon3d", "visualize3d"])
```

6. Composing a Complete Solving Workflow

A complete PyMultiDIC solution usually follows seven steps: validate, sfm, scale, mask, dic2d, recon3d, and visualize3d. The first six steps compute the physical result; visualize3d creates user-facing plots and cloud maps.

6.1 Configuration-file workflow

```
import pymultidic

config = pymultidic.load_config("configs/MDIC.yaml")
reports = {}
for step in ["validate", "sfm", "scale", "mask", "dic2d", "recon3d", "visualize3d"]:
    reports[step] = pymultidic.run_step(config, step)
    if not reports[step].get("ok"):
        raise RuntimeError(f"{step} failed: {reports[step]}")
print(reports["visualize3d"]["outputs"]["surface_cloud_displacement_total"])
```

This workflow is best for reproducible projects because the YAML file records paths and numerical parameters.

6.2 Direct-input workflow

```
import pymultidic

config = pymultidic.build_config(
    case_root="case/CylinderDIC",
    project_name="CylinderDIC",
    subset_radius=25,
    subset_spacing=6,
    min_corrcoef=0.6,
    outlier_filter_enabled=True,
)

pymultidic.run_validate(config)
pymultidic.run_sfm(config)
pymultidic.run_scale(config)
pymultidic.run_mask(config)
pymultidic.run_dic2d(config)
pymultidic.run_recon3d(config)
vis_report = pymultidic.run_visualize3d(config)

for key in ["surface_cloud_morphology", "surface_cloud_displacement_total",
            "surface_cloud_displacement_ux", "surface_cloud_displacement_uy",
            "surface_cloud_displacement_uz"]:
    print(key, vis_report["outputs"][key])
```

This workflow is best for notebooks and user scripts where the case path and a few numerical parameters are enough.

6.3 Partial reruns

After a full solve, rerun only the stages affected by a change. Visualization changes require only run_visualize3d. DIC2D parameter changes require run_dic2d, run_recon3d, and run_visualize3d. SfM or scale changes require all downstream stages.

7. Output Files and Result Locations

All default outputs are written under <case_root>/<output_root>. In the example this is case/CylinderDIC/results. Each step returns a Python dict report and generally writes a matching JSON report.

Location	Created by	Contents
<case_root>/<output_root>/logs	all steps	JSON reports such as validation_report.json, sfm_report.json, scale_report.json, mask_report.json, dic2d_report.json, recon3d_report.json, and pipeline_report.json.
<case_root>/<output_root>/sfm/colmap	run_sfm	COLMAP workspace, camera files, cameras.npz, cameras.json, sparse_points.npz, observations.npz, and diagnostic SfM outputs.
<case_root>/<output_root>/scale	run_scale	Scale correction JSON, checkerboard triangulation outputs, and detection overlays.
<case_root>/<output_root>/masks	run_mask	Binary ROI masks, numpy masks, overlays, debug images, and auto ROI metadata.
<case_root>/<output_root>/dic2d	run_dic2d	Per-camera/per-frame DIC2D npz files named dic2d_<camera>_<frame>.npz.
<case_root>/<output_root>/recon3d	run_recon3d	Global 3D reconstruction npz/ply files, QC plots, QC PLY files, pair surface meshes, and post3d data.

Location	Created by	Contents
<case_root>/<output_root>/recon3d/pairs/<frame>	run_recon3d	Pair surface npz/ply files with points, faces, valid_faces, correlation, reprojection error, and displacement fields.
<case_root>/<output_root>/recon3d/post/<frame>	run_recon3d	Post-processed pair data, including rigid-body-motion-removed displacement and face-level measures.
<case_root>/<output_root>/figures	run_visualize3d	General 3D visualization figures, reference point PLY, interpolated surface fields, and visualize3d report.
<case_root>/<output_root>/figures/surface_clouds	run_visualize3d	Morphology cloud map, total displacement cloud map, and Ux/Uy/Uz component cloud maps.

7.1 Reading important output paths from reports

```
vis_report = pymultidic.run_visualize3d(config)
outputs = vis_report["outputs"]

print(outputs["surface_cloud_morphology"])
print(outputs["surface_cloud_displacement_total"])
print(outputs["surface_cloud_displacement_ux"])
print(outputs["surface_cloud_displacement_uy"])
print(outputs["surface_cloud_displacement_uz"])
```

7.2 Files most users inspect first

For numerical 3D data, inspect results/recon3d/recon3d_<frame>.npz. For point-cloud viewing, open results/recon3d/recon3d_<frame>.ply. For final visual reporting, inspect results/figures/surface_clouds. For debugging, start from results/logs/recon3d_report.json and QC images under results/recon3d/qc/<frame>.

8. Practical Notes

PyPI versions are immutable: if a released version has to change, increment the version number and publish a new tag. Native C++ rebuilds are needed when files under native/ or CMake build configuration change; pure Python API and documentation changes do not require rebuilding native code.