VNIVERSIDAD DE LA HABANA

The Abdus Salam International Centre for Theoretical Physics

SCHOOL ON
COLLECTIVE ANIMAL BEHAVIOUR

yupi

A Python Library to Handle Trajectories

Alfredo Reyes González

La Habana, 2023

why to code?

- Molecular Dynamics

TheRoe, D.R., et.al., 2013. Ptraj and cpptraj: software forprocessing and analysis of molecular dynamics trajectory data.

- Human Mobility

Pappalardo, L.,et.al., 2019. scikit-mobility: A python library for the analysis, generation and risk assess-ment of mobility data.

- Geo-positional trajectories

Graser, A., 2019. Movingpandas: efficient structures for movement data inpython.

- Urban Trajectories

Shamal, A.D., et.al., S., 2019. An open source trajanalytics software for modeling, transformationand visualization of urban trajectory dat

- Generic (two dimensional) trajectories

Shenk, J.,et.al., 2021. Traja:A python toolbox for animal trajectory analysis. The Journal of OpenSource Software .

- Wide diversity of software to address specific trajectory-related task.
- Most of existing software for trajectory analysis only address trajectories limited to a fixed number of dimensions.
- Lack of integration between trajectory softwares.

outline

yupi : a python library to handle trajectory data

ELSEVIER

Environmental Modelling & Software

Volume 163, May 2023, 105679

Position Paper

yupi: Generation, tracking and analysis of trajectory data in Python

A. Reyes [a], G. Viera-López [b], J.J. Morgado-Vega [a], E. Altshuler [a]

Show more

+ Add to Mendeley   Share   Cite

https://doi.org/10.1016/j.envsoft.2023.105679

https://github.com/yupidevs/yupi

https://github.com/yupidevs/yupi_examples

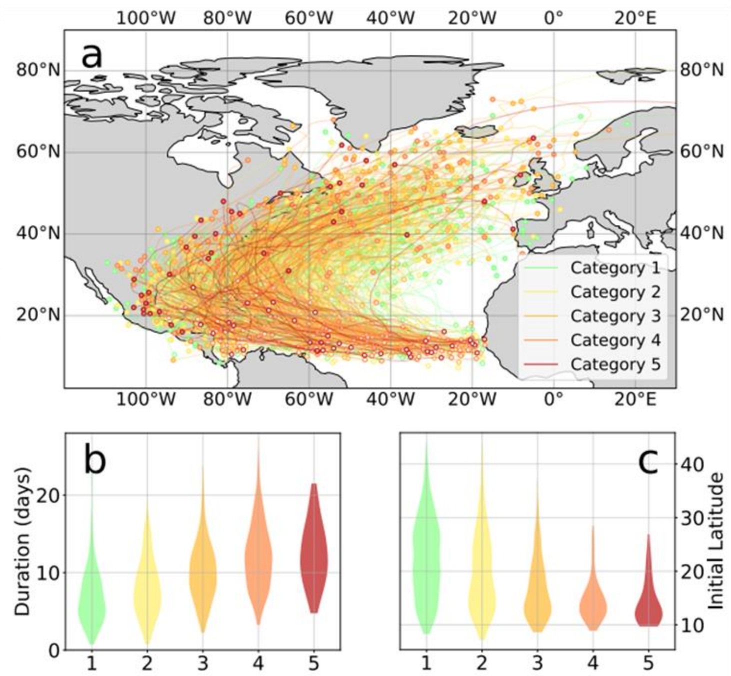


Figure 4: Visual inspection of the trajectories of hurricanes in the North Atlantic. (a) Trajectories displayed using a *yupi*'s spatial plot. (b) Distribution of hurricane duration for each category. (c) Distribution of hurricane's initial latitude for each category.

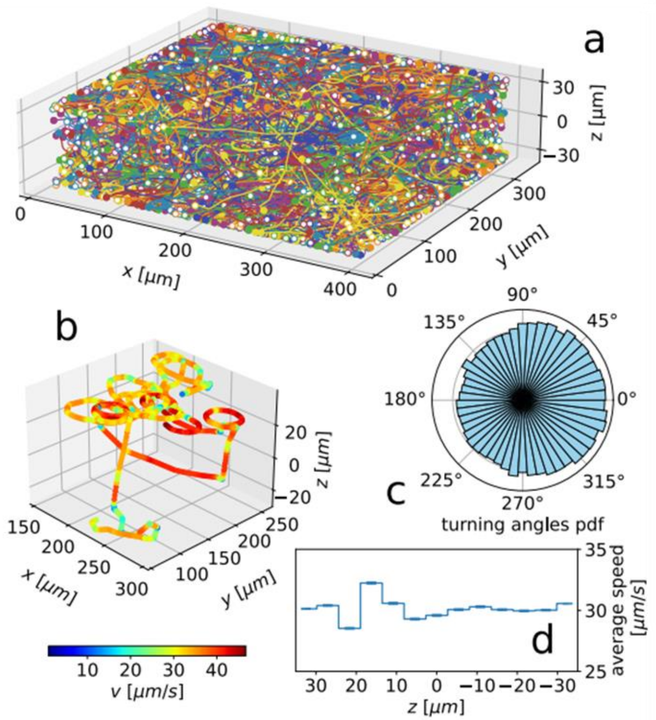
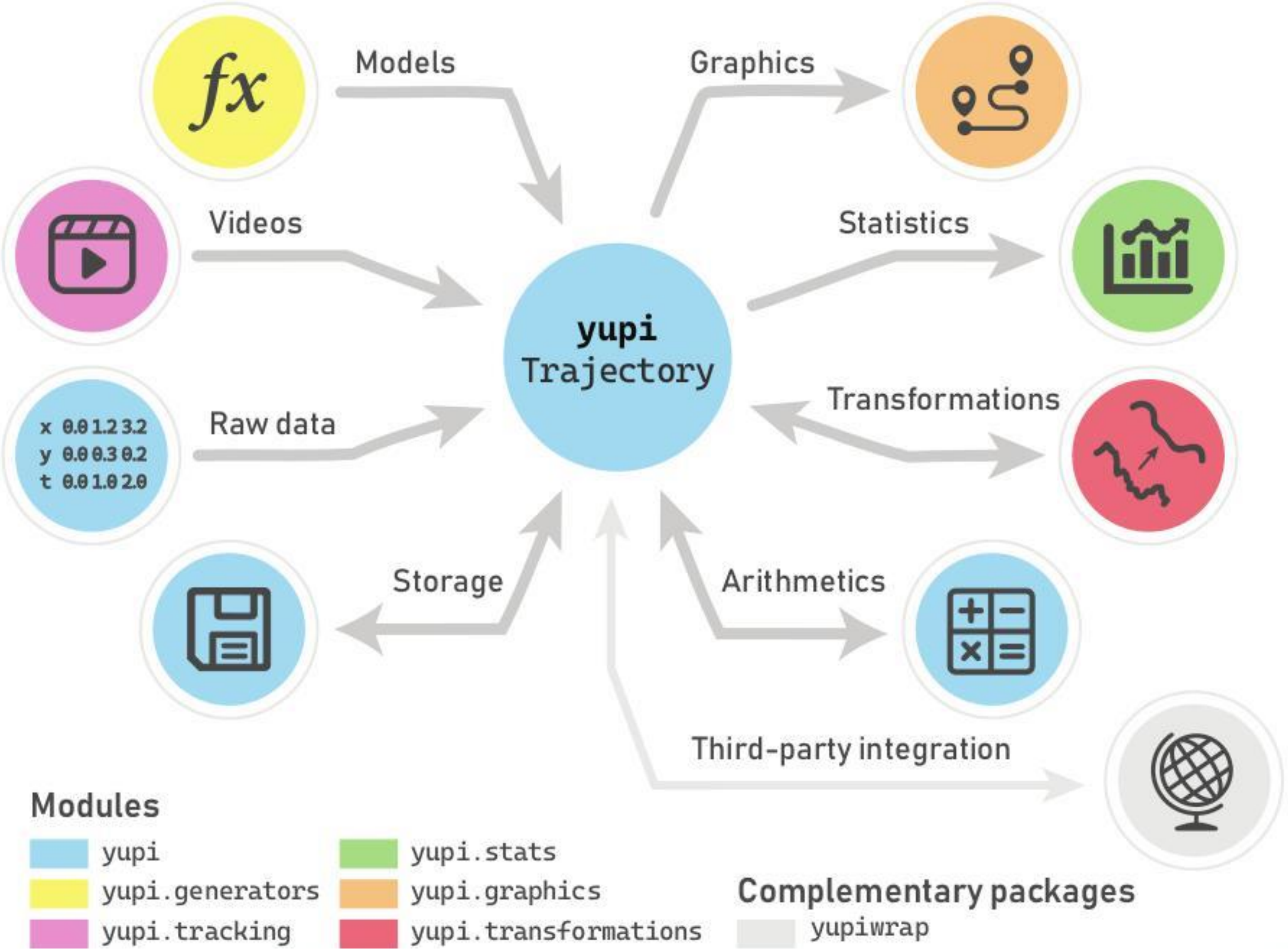


Figure 5: Reproduction of the analysis conducted in (Grognot and Taute, 2021). (a) Individual trajectories of an ensemble of $N = 2000$ bacteria. (b) Trajectory of a single bacteria where colors have been linked to speed values. Blue/red correspond to sections of low/high local speed. (c) Distribution of swimming direction. (d) Average speed and standard error of the mean as a function of height.

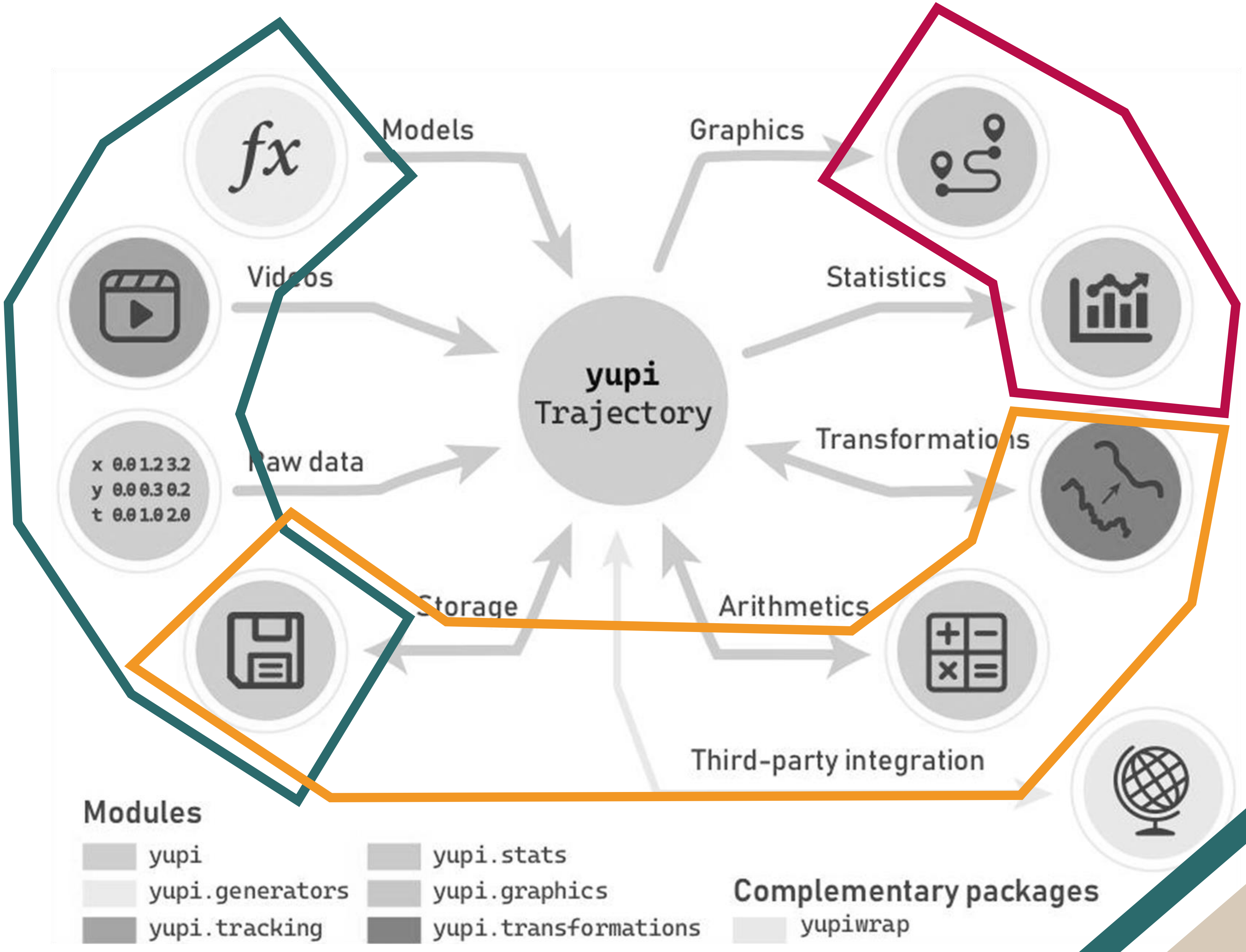
Yet Underused Python Instrument



trajectory acquisition

trajectory manipulation

getting information



yupi.core : vector and trajectory objects

Vector objects

```python
from yupi import Vector

vec = Vector([[3, 4], [2, 0], [8, 6]])

print(vec.x, vec.component(0))
# (Vector([3., 2., 8.]),
#  Vector([3., 2., 8.]))

print(vec.norm)
# Vector([ 5.,  2., 10.])
```

- The `Vector` class stores all time-evolving data in a trajectory.
- It was implemented by wrapping `ndarray` type from `numpy`.
- Information from a `Vector` instance can be extracted by accessing certain properties.

Trajectory objects

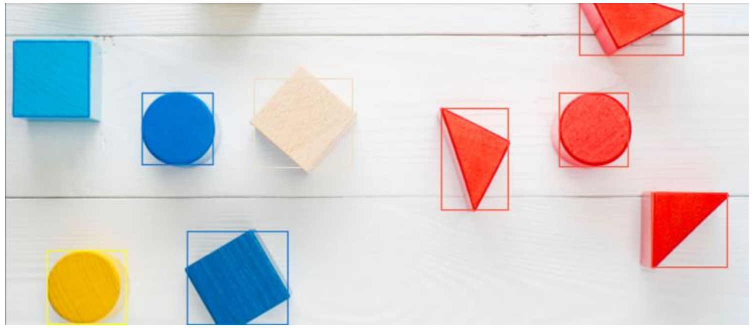
```python
from yupi import Trajectory

traj = Trajectory(
    x=[3, 5, 13], y=[4, 4, 10],
    dt=0.5, traj_id='my_traj')

print(traj.r, traj.r.x)
# (Vector([[ 3.,  4.],
#          [ 5.,  4.],
#          [13., 10.]]),
#  Vector([ 3.,  5., 13.]))

print(type(traj.r))
# yupi.vector.Vector
```

- A `Trajectory` object is the fundamental structure within `yupi`.
- Time, position, velocity and acceleration time series can be accessed through the attributes `t`, `r`, `v` and `a`.
- Other cool things:

```python
traj.v
traj.r.delta.norm
traj1 + traj2
traj[::2]
```

`yupi.tracking` : tracking algorithms

- Color Matching



- Template Matching

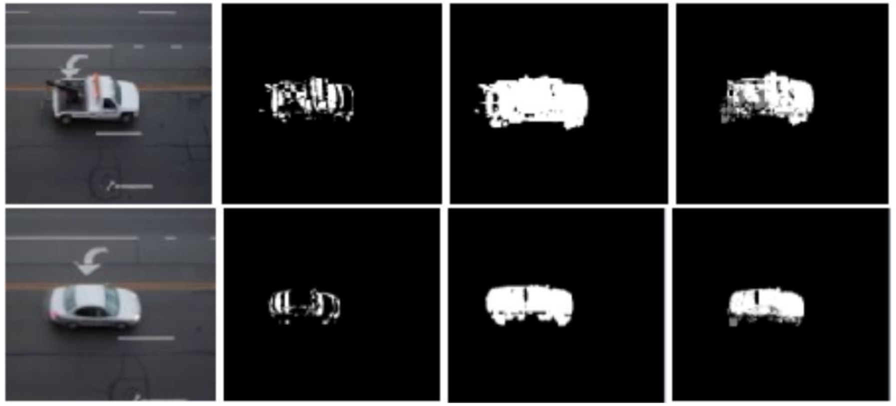
Template location

Template

- Background subtraction

- Optical Flow

- Frame Differencing



- The tracking workflow

1. algorithm → Algorithm
2. roi → ROI
3. tracker1 → ObjectTracker

   …

   trackerN → ObjectTracker
4. trackers.append({tracker})
5. scenario → TrackingScenario
6. scenario.track()

- Choose a tracking **algorithm**.
- Select a **region of interest**.
- Define **objects to track**.
- **Gather** all tracking objects.
- Create your **tracking scenario**.
- Run Forest, **run**!

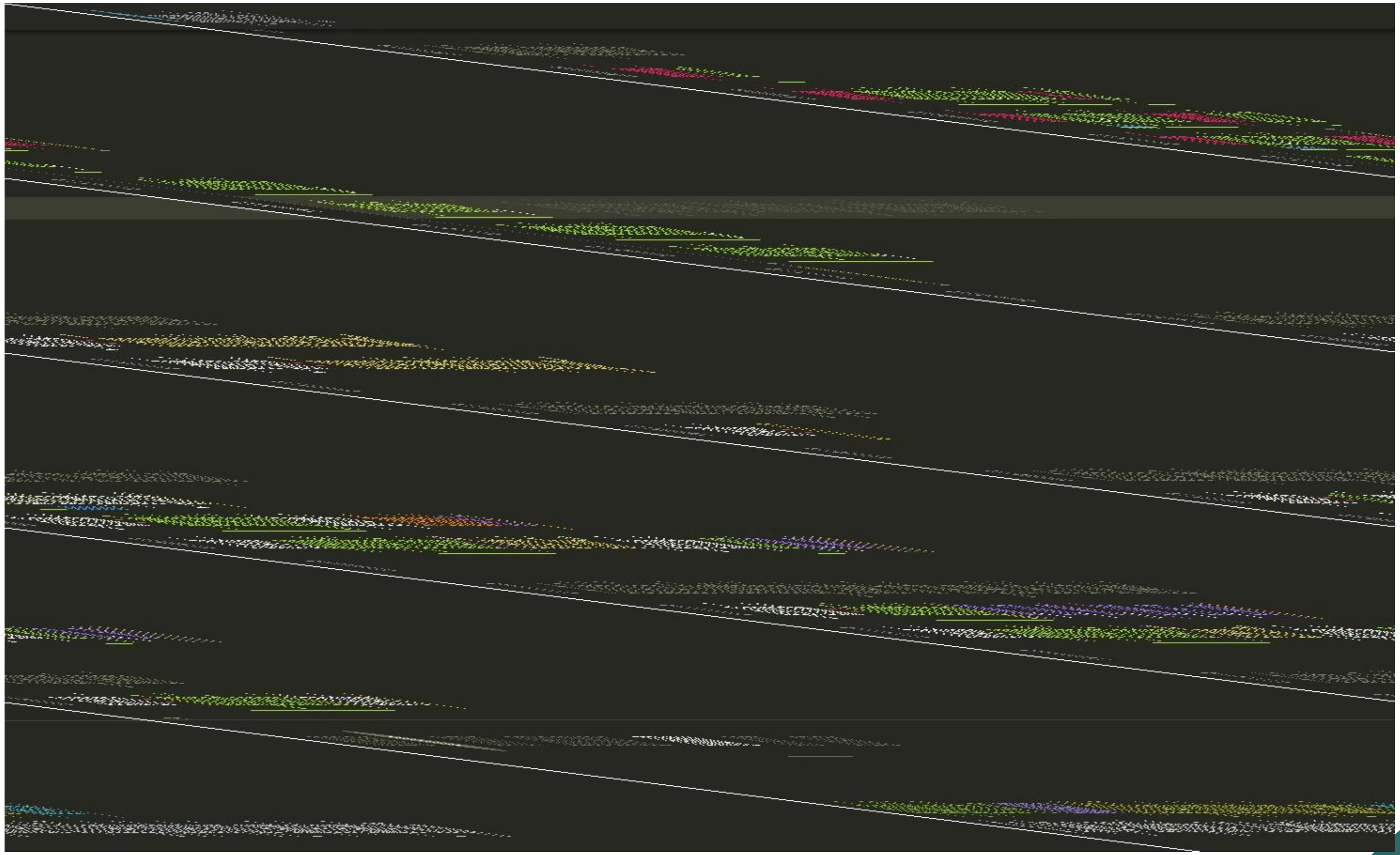
- Three Examples https://github.com/yupidevs/yupi_examples

yupi.tracking : example 1 (wheel at zero gravity)

```
template = cv2.imread(template_path)
algorithm1 = TemplateMatching(
    template)
tracker1 = ObjectTracker(
    "Central Pivot",
    algorithm1,
    ROI((80, 80)))
trackers.append(tracker1)
```

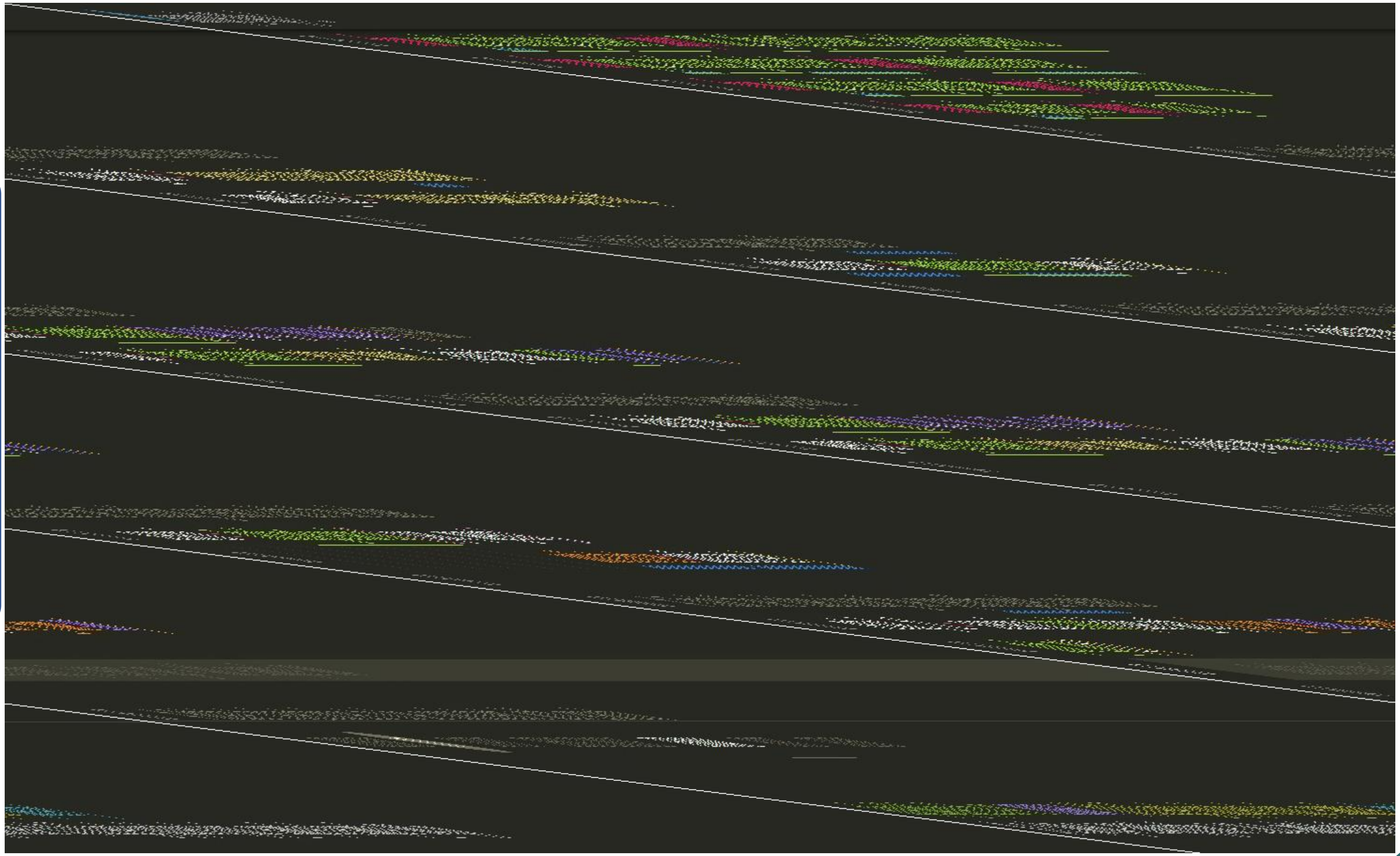
```
algorithm2 = ColorMatching(
    (80, 170, 90), (190, 255, 190))
tracker2 = ObjectTracker(
    "Green LED",
    algorithm2,
    ROI((50, 50)))
trackers.append(tracker2)
```

```
scenario = TrackingScenario(trackers)
scenario.track(video_path,
    pix_per_m=4441,
    start_frame=160)
```



- how to create a tracking scenario using Template and Color Matching algorithms

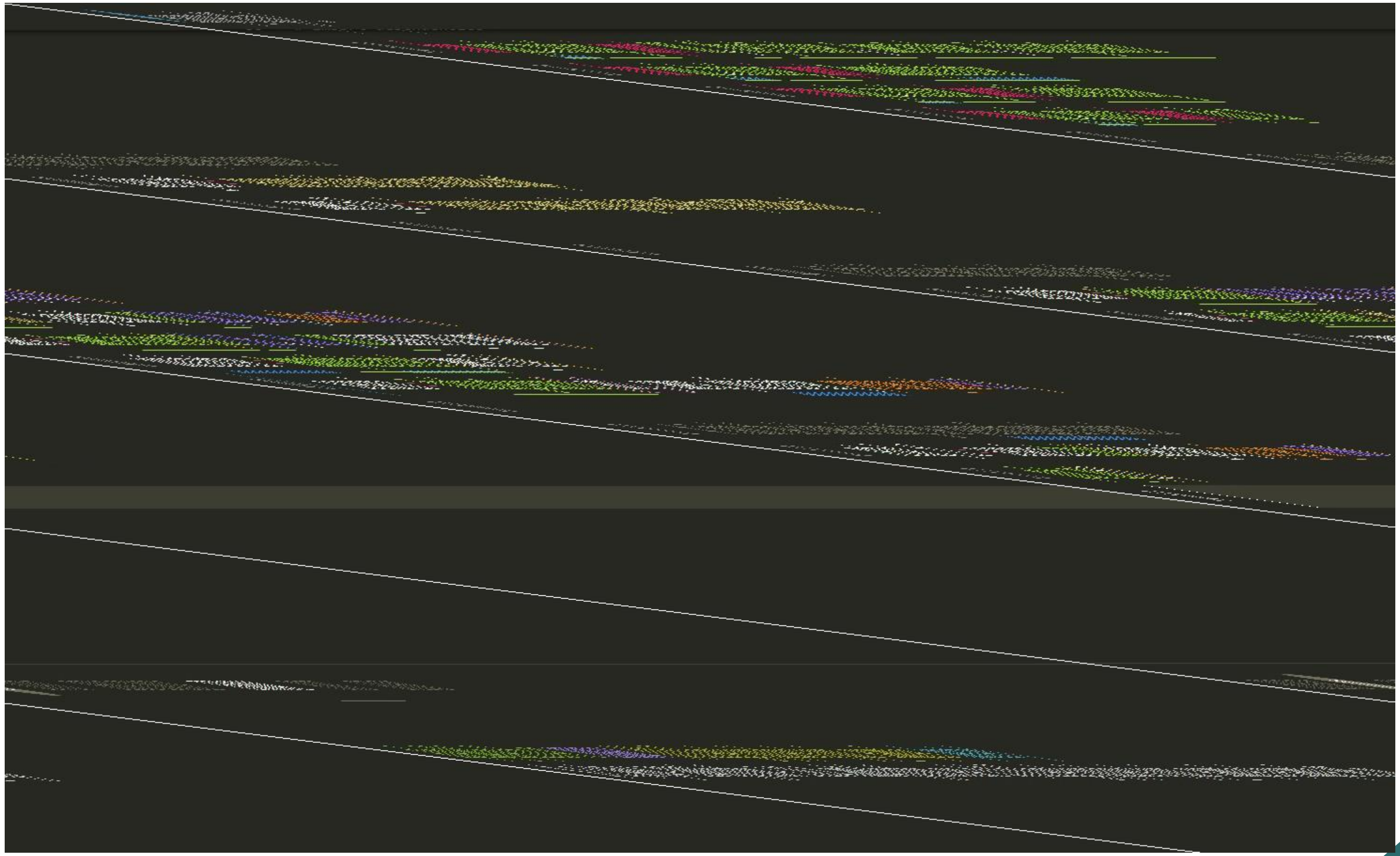
```
camera_file =
'resources/cameras/gph3+.npz'

undistorter = RmapUndistorter(camera_file)

scenario = TrackingScenario(
    trackers,
    undistorter=undistorter)
```



- how to correct camera distortions

yupi.tracking : example 3 (ant movement)

```
ant = ObjectTracker(
    'ant', algorithm,
    ROI((120, 120)))

camera = CameraTracker(
    ROI((.65, .65),
    ROI.CENTER_INIT_MODE))

scenario = TrackingScenario(
    [ant], camera,
    undistorter)
```



- how to track when the camera is also moving

Generator objects

- Langevin Model

$$\frac{d}{dt}\mathbf{v}(t) = -\gamma\mathbf{v}(t) + \sigma\boldsymbol{\xi}(t)$$

```
from yupi.generators import
LangevinGenerator

lg = LangevinGenerator(
    T, dim, N, dt, gamma, sigma)
trajs = lg.generate()
```

Observables

- Velocity Autocorrelation Function

$$\phi(\tau) = \frac{1}{T-\tau}\int_0^{T-\tau} \mathbf{v}(t')\cdot\mathbf{v}(t'+\tau)dt'$$

```
from yupi.stats import vacf

vacf_mean, vacf_std = vacf(
    trajs,
    time_avg=True,
    lag=lag_vacf
)
```

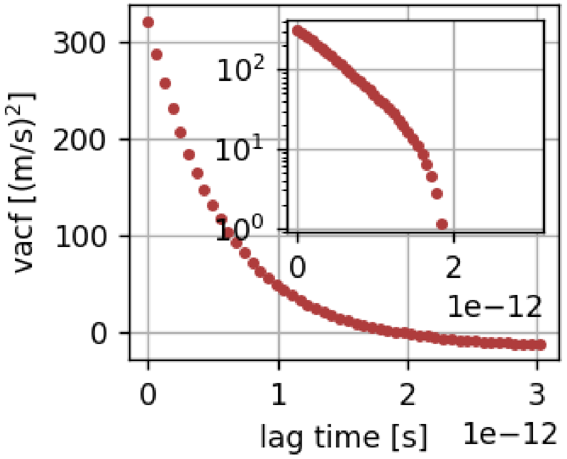
- Filters

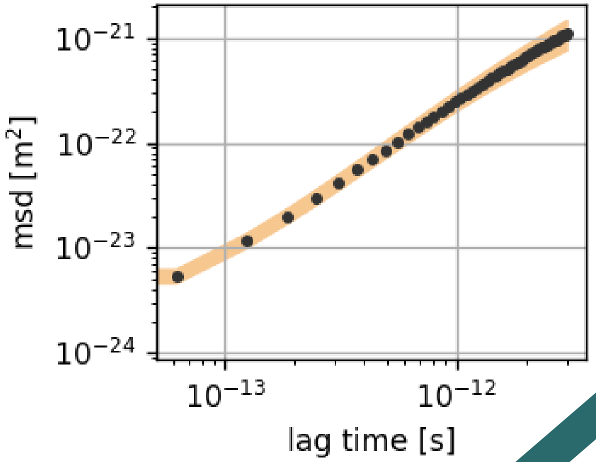
$$\mathbf{v}_s(t) = \Omega \int_0^t \mathbf{v}(t')e^{-\Omega(t-t')}dt'$$

```
from yupi.transformations import
   exp_convolutional_filter

smooth_traj = exp_convolutional_filter(
   traj, ommega=omega)
```

- Observables

```
from yupi.graphics import plot_vacf

plot_vacf(vacf, dt, lag_vacf, show=False)

plot_msd(msd, msd_std, dt, lag=lag_msd)
```





…and there is more

- Ways to instantiate Trajectory objects.
- Storage on disk
- Custom tracking algorithms (the abstract Algorithm class)
- Custom generators (the abstract Generator class)
- yupiwrap
- Marchine Learning perspectives

- J. J. Morgado Vega
- G. Viera López

Summary and Conclusions:

- yupi is an open-source Python library designed for comprehensive handling of trajectory data of any dimension.
- It integrates the processes of trajectory extraction, generation, transformation, storage, statistical analysis, and visualization.
- Methods are distributed across six modules, organized to operate on the same data structure.
- Trajectory objects are manipulated through instances of Vector objects, which inherit from ndarray from the numpy library.
- Integrate with other Python libraries dedicated to trajectory management.

Department of Computer Science,
Gran Sasso Science Institute, Italy

thanks