# MD_rigid

***Release 0.1***

**Andrea Silva**

**May 01, 2023**

# CONTENTS:

Compute the interlocking potential between a periodic substrate and a finite-size adsorbate, in the rigid approximation. The adsorbate is treated as a rigid body at a given orientation $theta$ and center of mass (CM) position $x_mathrm{cm}, y_mathrm{cm}$

CHAPTER

ONE

# SUBSTRATE

The substrate is defined as a periodic function resulting from either a monocromaitc superposition of plane waves or a potential well of a given shape repeated in space. The functions handling the substrate creation are in `tool_create_substrate.py`.

For a plane wave superposition, the substrate is defined by a suitable set of wave vectors, where the number of vectors defines the symmetry and length of vectors defines the spacing [1].

For a lattice of wells, the substrate is defined by the shape parameters of the well and the lattice vectors [2-5]. This substrate can be decorated with a lattice basis.

The parameters are specified in a JSON file. See Example/0-Substrate_types.ipynb for details.

CHAPTER

TWO

# CLUSTER

The cluster is defined as a collection of points (optionally decorate with a basis) belonging to a given lattice. For convenience, there are functions returning clusters in regular shapses, e.g. rectangles, hexagons, circles, etc. Using the Shapely package, clusters of arbitrary shapes (e.g. imaged in experiments) can be created. The functions handling the substrate creation are in `tool_create_substrate.py`.

See example/1-Cluster_creation.ipynb for details.

CHAPTER

# THREE

# STATIC MAPS

See example/2-Cluster_on_substrate.ipynb for details on the following functions.

## 3.1 Translations

To explore the energy landscape of an adsorbate over a substrate as a function of the CM at fixed orientation, see `static_trasl_map.py`

## 3.2 Rotations

To explore the energy landscape of an adsorbate over a substrate as a function of the imposed rotation $theta$, at fixed CM, see `static_roto_map.py`

## 3.3 Roto-translations

To search for the global minimum of an adosrbate, one needs to combine rotations and translation, and locate the energy minimum in the $(**x**_mathrm{cm}, **y**_mathrm{cm}, theta)$ space. See `static_rototrasl_map.py` for details.

CHAPTER

# FOUR

# DYNAMICS MAPS

To go beyond rigid maps, there are two essential tools: compute the minimum energy path between two minimum or perform a molecular dynamics calculation under given translational and rotational drives $(F_x, F_y, tau)$.

## 4.1 Barrier finding

The barrier between two points in the configurational space $(x_mathrm{cm}, y_mathrm{cm}) at fixed orientation can be estimated be the string algorithm [6], similar to the NEB methods. The ideal can be summarised like this: imagine the potential energy to be a hill landscape. Place a string between two points of the landscape and let it relax. The string would relax downhill until the gradient on the string vanishes, i.e. the string layes on the pass between the valleys and below the peaks.

See example/3-Barrier_from_stirng.ipynb

## 4.2 Molecular dynamics

The script `MD_rigid_rototrasl.py` solve the equation of motion for the center of mass and orientation of the cluster in the overdamped regime (no interial term).

See example/molecular_dynamics for an example of a system depinning under a constant force and torque.

### 4.2.1 Equations of motion

In the overdamped limit, the equation of motion are the following first order equations:

$$ gamma_{t} frac{dmathbf{r}}{dt} = (mathbf{F}_{ext} - nabla U) $$

$$ gamma_{r} frac{dtheta}{dt} = (tau_{ext} - frac{dU}{dtheta}) $$

The dissipation constants of the CM for a cluster of $N$ particles are linked the "particle-like" damping constant $gamma$ by $gamma_t = N gamma$ and $gamma_r = gamma sum_i r_i^2 $, where $r_i$ is the position of the $i$-th particle with respect to the center of mass.

In this picture energy is not conserved (fully dissipated in the Langevin bath between successive timesteps) and the value of the dissipation constant $gamma$ effectively sets how quickly the time flows. Thus by lowering $gamma$ one can "speed up" the simulations and match timescales similar to experiments.

CHAPTER

FIVE

# UNITS

The model can be ragarded as adimensional.

A coherent set of units useful to compare with experimental colloidal system is:

- energy in zJ
- length in $mumathrm{m}$
- mass in fKg

From which follows:

- force in fN
- torque in fN $cdot mu mathrm{m}$
- time in ms
- translational damping constant $gamma$ in fKg/ms

CHAPTER

SIX

# REFERENCES


1. Vanossi, Andrea, Nicola Manini, and Erio Tosatti. "Static and Dynamic Friction in Sliding Colloidal Monolayers." Proceedings of the National Academy of Sciences 109, no. 41 (October 9, 2012): 16429–33. https://doi.org/10.1073/pnas.1213930109.

2. Panizon, Emanuele, Andrea Silva, Xin Cao, Jin Wang, Clemens Bechinger, Andrea Vanossi, Erio Tosatti, and Nicola Manini. "Frictionless Nanohighways on Crystalline Surfaces." Nanoscale 15, no. 3 (2023): 1299–1316. https://doi.org/10.1039/D2NR04532J.

3. Cao, Xin, Andrea Silva, Emanuele Panizon, Andrea Vanossi, Nicola Manini, Erio Tosatti, and Clemens Bechinger. "Moiré-Pattern Evolution Couples Rotational and Translational Friction at Crystalline Interfaces." Physical Review X 12, no. 2 (June 15, 2022): 021059. https://doi.org/10.1103/PhysRevX.12.021059.

4. Cao, Xin, Emanuele Panizon, Andrea Vanossi, Nicola Manini, and Clemens Bechinger. "Orientational and Directional Locking of Colloidal Clusters Driven across Periodic Surfaces." Nature Physics 15, no. 8 (August 2019): 776–80. https://doi.org/10.1038/s41567-019-0515-7.

5. Cao, Xin, Emanuele Panizon, Andrea Vanossi, Nicola Manini, Erio Tosatti, and Clemens Bechinger. "Pervasive Orientational and Directional Locking at Geometrically Heterogeneous Sliding Interfaces." Physical Review E 103, no. 1 (January 13, 2021): 012606. https://doi.org/10.1103/PhysRevE.103.012606.

6. E, Weinan, Weiqing Ren, and Eric Vanden-Eijnden. "Simplified and Improved String Method for Computing the Minimum Energy Paths in Barrier-Crossing Events." The Journal of Chemical Physics 126, no. 16 (April 28, 2007): 164103. https://doi.org/10.1063/1.2720838.

CHAPTER

SEVEN

# CONTENTS

## 7.1 Modules

### 7.1.1 Tools

#### Substrate

tool_create_substrate.**calc_en_gaussian**(*pos*, *pos_torque*, *basis*, *a*, *b*, *sigma*, *epsilon*, *u*, *u_inv*)

Calculate energy, forces and torque on CM.

See corresponding particle function for details on parameters.

Return total energy (scalar) force (2d vector) torque (scalar).

tool_create_substrate.**calc_en_sin**(*pos*, *pos_torque*, *basis*, *ks*, *epsilon*)

Calculate energy, forces and torque on CM.

Substrate energy is modelled as sum of plane waves defined by the reciprocal vecotrs ks.

See corresponding particle function for details on parameters.

Return total energy (scalar) force (2d vector) torque (scalar).

tool_create_substrate.**calc_en_sin_tri**(*pos*, *pos_torque*, *basis*, *a*, *epsilon*, *u*, *u_inv*)

Calculate energy, forces and torque on CM.

!!! Coefficients are for triangular lattice only !!! Substrate energy is modelled as sum of three plane waves.

tool_create_substrate.**calc_en_tanh**(*pos*, *pos_torque*, *basis*, *a*, *b*, *ww*, *epsilon*, *u*, *u_inv*)

Calculate energy, forces and torque on CM.

Substrate energy is modelled as a lattice of tanh-shaped wells.

See corresponding particle function for details on parameters.

Return total energy (scalar) force (2d vector) torque (scalar).

tool_create_substrate.**calc_matrices_bvect**(*b1*, *b2*)

Metric matrices from primitive lattice vectors b1, b2.

Return 2x2 matrices to map to unit cell (u) and inverse (u_inv), back to real space.

tool_create_substrate.**calc_matrices_square**(*R*)

Metric matrices of square lattice of spacing R.

Return 2x2 matrices to map to unit cell (u) and inverse (u_inv), back to real space.

tool_create_substrate.**calc_matrices_triangle**(*R*)

Metric matrices of triangular lattice of spacing R.

Return 2x2 matrices to map to unit cell (u) and inverse (u_inv), back to real space.

tool_create_substrate.**gaussian**(*x*, *mu*, *sigma*)

tool_create_substrate.**get_ks**(*R*, *n*, *c_n*, *alpha_n*)

Compute wave vectors k of interfering plane waves

tool_create_substrate.**particle_en_gaussian**(*pos*, *pos_torque*, *basis*, *a*, *b*, *sigma*, *epsilon*, *u*, *u_inv*)

Calculate energy, forces and torque on each particle.

Substrate energy is modelled as a lattice of gaussian-shaped wells.

Inputs are: - pos: position of particles in rigid cluster, as (N,2) array. - pos_torque: reference point (1,2 array) to compute the torque (usually CM). - basis: list of position of the substrate basis (N,2 array).

Well specific inputs (usually passed as *en_input): - a, b: beginning and end of tempered region (W=0 for x>b). - sigma: width of the gaussian. - epsilon: depth of the well. - u, u_inv: matrices to map to subsrtate unit cell (see calc_matrices_bvect)

Returns (for each particle) energy (N array), force (N,2), torque (N)

tool_create_substrate.**particle_en_sin**(*pos*, *pos_torque*, *basis*, *ks*, *epsilon*)

Calculate energy, forces and torque on each particle.

Substrate energy is modelled as sum of plane waves defined by the reciprocal vecotrs ks.

Inputs are: - pos: position of particles in rigid cluster, as (N,2) array. - pos_torque: reference point (1,2 array) to compute the torque (usually CM). - basis: list of position of the substrate basis (N,2 array).

Well specific inputs (usually passed as *en_input): - ks: list of wavevector generating the potential. - epsilon: depth of the potential.

Returns (for each particle) energy (N array), force (N,2), torque (N)

tool_create_substrate.**particle_en_sin_tri**(*pos*, *pos_torque*, *basis*, *R*, *epsilon*, *u*, *u_inv*)

Calculate energy, forces and torque on each particle.

!!! Coefficients are for triangular lattice only !!! Substrate energy is modelled as sum of three plane waves.

tool_create_substrate.**particle_en_tanh**(*pos*, *pos_torque*, *basis*, *a*, *b*, *ww*, *epsilon*, *u*, *u_inv*)

Calculate energy, forces and torque on each particle.

Substrate energy is modelled as a lattice of tanh-shaped wells.

Inputs are: - pos: position of particles in rigid cluster, as (N,2) array. - pos_torque: reference point (1,2 array) to compute the torque (usually CM). - basis: list of position of the substrate basis (N,2 array).

Well specific inputs (usually passed as *en_input): - a, b: beginning and end of tanh well (W=-epsilon for r<a, W=0 for x>b). - ww: shape factor for the tanh potential. - epsilon: depth of the well. - u, u_inv: matrices to map to subsrtate unit cell (see calc_matrices_bvect)

Returns (for each particle) energy (N array), force (N,2), torque (N)

tool_create_substrate.**substrate_from_params**(*params*)

Initialise a substrate from a parameter dictionary (usually from JSON file)

Returns the functions computing the particle-wise and total energy, and the list of potential-specific paramters to pass to these functions.

### Cluster

tool_create_cluster.**add_basis**(*lat_pos*, *basis*)

Add a crystal basis to a simple Bravais lattice

tool_create_cluster.**calc_cluster_langevin**(*eta*, *pos*)

Compute the effective translational and rotational damping acting on a CM of a cluster of N particles

tool_create_cluster.**cluster_from_params**(*params*)

Create cluster from parameters in dictionary.

Return xy positions

tool_create_cluster.**cluster_inhex_Nl**(*N1*, *N2*, *a1=array([4.45*, *0. ])*, *a2=array([-2.225*, *3.85381305])*, *clgeom_fname='input_pos.hex'*, *cluster_f=<function create_cluster_circle>*, *X0=0*, *Y0=0*)

Create a cluster from lattice details (equivalent to .hex file) and specific shape function.

The .hex file will be created and removed using tempfile. Returns only xy positions.

Default values relate to Xin/EP colloids [Nat. Phys 2019, PRE 2021] and circular shape

tool_create_cluster.**cluster_poly**(*polygon*, *params*, *direction=0*)

Use a polygon (shapely object) to mask a lattice (defined in params).

Direction = 0: select the interior of the polygon (use bounds of polygon to define lattice big enough for mask)
Direction = 1: select the exterior of the polygon (up to the N1 N2 in params)

Return xy pos

tool_create_cluster.**create_cluster**(*input_cluster*, *angle=0*)

Create clusters taking as input the two primitive vectors a1 and a2 and the indices of lattice points. Put center of mass in zero.

tool_create_cluster.**create_cluster_circle**(*input_hex*, *outstream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *X0=0*, *Y0=0*)

Circle cluster

The 6 dimensions are for backward compatibility with EP.

tool_create_cluster.**create_cluster_hex**(*input_hex*, *outstream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *X0=0*, *Y0=0*)

Hexagonal cluster

The 6 dimensions are for backward compatibility with EP.

tool_create_cluster.**create_cluster_rect**(*input_hex*, *outstream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *X0=0*, *Y0=0*)

Rectangular cluster

The 6 dimensions are for backward compatibility with EP.

tool_create_cluster.**create_cluster_special**(*input_hex*, *outstream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *X0=0*, *Y0=0*)

Special-parall cluster. See paper on XXX.

The 6 dimensions are for backward compatibility with EP.

tool_create_cluster.**create_cluster_tri**(*input_hex*, *outstream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *X0=0*, *Y0=0*)

Triangular cluster

The 6 dimensions are for backward compatibility with EP.

tool_create_cluster.**create_input_hex**(*N1*, *N2*, *clgeom_fname='in.hex'*, *a1=array([4.45, 0.0])*, *a2=array([-2.225, 3.85381305])*)

Create input file in EP .hex format

Cluster is created from Bravais lattice a1 a2 with N1 repetition along a1 and N2 repetitions along N2. Default is Xin colloids: triangular with spacing 4.45

tool_create_cluster.**get_poly**(*points*, *scale=1*, *tho=0*, *c=[0, 0]*, *shift=0*, *cm=False*)

Get a polygon (Shapely obejct) from a set of points.

Optinally rotate, scale and shfit

tool_create_cluster.**get_rotomatr**(*angle*)

Get ACW rotation matrix of an angle [degree]

tool_create_cluster.**load_cluster**(*input_hex*, *angle=0*, *center=False*)

Load cluster form numpy file data. Optionally adjust CM and rotate.

tool_create_cluster.**load_input_hex**(*instream*)

Load .hex file defining lattice and size

tool_create_cluster.**params_from_ASE**(*ase_geom*, *cut_z=0*, *tol=0.9*)

Create JSON parameters to create clusters from ASE Atoms object.

Assume this is already a 2D surface oriented along z to extract 2x2 matrix: From:

a b 0 c d 0 0 0 1

**Take:**
a b c d

Positions are flattend out: (x,y) from (x,y,z)

Return parameters dictionary and list of z coordinates considered

tool_create_cluster.**params_from_poscar**(*poscar_fname*, *cut_z=0*)

Create JSON parameters to create clusters from POSCAR file.

Use ASE to read.

Return parameters file.

tool_create_cluster.**rotate**(*pos*, *angle*, *c=[0, 0]*)

Rotate positions pos of angle [degree] with respect to center c (default 0,0)

tool_create_cluster.**save_xyz**(*pos*, *outfname='cluster.xyz'*, *elem='X'*)

Save cluster as xyz in given file (default 'cluster.xyz')

#### String method

class string_method.**Path**(*a*, *b*, *pos*, *N*, *fix_ends=False*)

Bases: object

**eulerArc**(*potential*, *dt=1e-07*)

Euler method to integrate dynamics.

**reparametrizeArc**()

Equal arc parameterisation of the path.

class string_method.**PotentialPathAnalyt**(*path*, *en_f*, *en_in*)

Bases: object

**grad**()

Mathematical form of gradient (dV/dx, dV/dy) along the path gamma.

**total**()

Mathematical form of the total potential function V along the path gamma.

**update**(*path*)

### 7.1.2 Static

#### Static Translation

static_trasl_map.**static_traslmap**(*pos*, *inputs*, *calc_en_f*, *name=None*, *log_propagate=True*, *debug=False*)

Compute the energy of a rigid cluster as a function of CM position, at fixed orientation.

Inputs contains the details of the system, along with the unit cell of the substrate (S) and the fractional range of nbin translation to explore along each lattice direction (S[0], S[1]).

Return a Nx6 array with xcm, ycm, e_pot, forces[0], forces[1], torque

#### Static Rotation

static_roto_map.**static_rotomap**(*pos*, *inputs*, *calc_en_f*, *name=None*, *log_propagate=True*, *debug=False*)

Compute the energy of a rigid cluster as a function of orientation, at fixed position.

Inputs contains the details of the system, along with the range of N angles to explore.

Return a Nx5 array with theta, e_pot, forces[0], forces[1], torque

#### Transition state barrier

static_barrier_string.**static_barrier**(*pos*, *inputs*, *calc_en_f*, *name=None*, *log_propagate=True*, *debug=False*)

Compute the transition state barrier of a rigid cluster between two points of the substrate.

The barrier is computed using the string algorithm [DOI: 10.1063/1.2720838].

Inputs: - pos: position of particles in rigid cluster - inputs: dictionary defining the syste. Must contain the start (p0) and end (p1) of the initial path. Optionally, the number of points in the string (Npt=100) and number of iterations (Nsteps=3000)

Returns the positions of the relaxed path, along with energy, force and torque evaluated along it.

#### Static Rotation and Translation

static_rototrasl_map.**static_rototraslmap**(*pos*, *inputs*, *calc_en_f*, *name=None*, *log_propagate=True*, *debug=False*)

Compute the energy of a rigid cluster as a function of CM position and orientation theta.

Inputs contains the details of the system, along with the unit cell of the substrate (S) and the fractional range of nbin translation to explore along each lattice direction (S[0], S[1]), and range of orientations.

Return a Nx7 array with theta, xcm, ycm, e_pot, forces[0], forces[1], torque

### 7.1.3 Dynamics

#### Molecular Dynamics of rigid cluster

MD_rigid_rototrasl.**MD_rigid**(*inputs*, *outstream=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *name=None*, *log_propagate=False*, *debug=False*)

Overdamped (1st order differential equaltion) Langevin Molecular Dynamics of rigid cluster over a substrate.

Input parameters are passaed as directly as dictionary or read from a JSON file (if inputs is a string). Position of particles in cluster is given as .hex or .npy filename. Info about run are written in name+'info.json'

MD output is printed on sys stdout.

CHAPTER

EIGHT

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX

## S



## T



## U