Titerra Documentation

John Harwell

Apr 28, 2023

CONTENTS:

CHAPTER

ONE

SYNOPSIS

TITERRA is the SIERRA projects, plugins, and extensions that I'm using on my PhD thesis.

Contains the following sub-projects:

1. FORDYCA (foraging with dynamic caches)
2. PRISM (graph manipulation for construction, deconstruction)

1.1 TITERRA Quickstart

1. Follow the SIERRA quickstart guide.
2. Install required python packages:

```
pip3 install -r requirements/common.txt
```

3. Select one of the TITERRA projects with `--project=<project>` when invoking SIERRA.
4. That's it!

1.2 Extensions to SIERRA Platforms

1.2.1 ARGoS Platform Extensions

Batch Criteria

The following Batch Criteria are defined which can be used with any Project in TITERRA (in addition to the general SIERRA batch criteria).

- *Sensor and Actuator Noise*

Sensor and Actuator Noise

Inject sensor and/or actuator noise into the swarm.

Cmdline Syntax

`saa_noise.{category}.C{cardinality}[.Z{population}]`

- `category` - [sensors,actuators,all]
  - `sensors` - Apply noise to robot sensors only. The `sensors` dictionary must be present and non-empty in the `main.yaml`.
  - `actuators` - Apply noise to robot actuators only. The `actuators` dictionary must be present and non-empty in `main.yaml`.
  - `all` - Apply noise to robot sensors AND actuators. [`sensors`, `actuators`] dictionaries both optional in `main.yaml`.
- `cardinality` - The # of different noise levels to test with between the min and max specified in the config file for each sensor/actuator which defines the cardinality of the batch experiment.
- `population` - The static swarm size to use (optional).

Examples

- `saa_noise.sensors.C4.Z16`: 4 levels of noise applied to all sensors in a swarm of size 16.
- `saa_noise.actuators.C3.Z32`: 3 levels of noise applied to all actuators in a swarm of size 32.
- `saa_noise.all.C10`: 10 levels of noise applied to both sensors and actuators; swarm size not modified.

The values for the min, max noise levels for each sensor which are used along with `cardinality` to define the set of noise ranges to test are set via the main YAML configuration file (not an easy way to specify ranges in a single batch criteria definition string). The relevant section is shown below. If the min, max level for a sensor/actuator is not specified in the YAML file, no XML changes will be generated for it.

Important: In order to use this batch criteria, you **MUST** have the version of ARGoS from Swarm Robotics Research. The version accessible on the ARGoS website does not have a consistent noise injection interface, making usage with this criteria impossible.

The following sensors can be affected (dependent on your chosen robot's capabilities in ARGoS):

- light
- proximity
- ground
- steering
- position

The following actuators can be affected (dependent on your chosen robot's capabilities in ARGoS):

- steering

YAML Config

For all sensors and actuators to which noise should be applied, the noise model and dependent parameters must be specified (i.e. if a given sensor or sensor is present in the config, all config items for it are mandatory).

The appropriate `ticks_range` attribute is required, as there is no way to calculate in general what the correct range of X values for generated graphs should be, because some sensors/actuators may have different assumptions/requirements about noise application than others. For example, the differential steering actuator `noise_factor` has a default value of 1.0 rather than 0.0, due to its implementation model in ARGoS, so the same range of noise applied to it and, say, the ground sensor, will have different XML changes generated, and so you can't just average the ranges for all sensors/actuators to compute what the ticks should be for a given experiment.

```yaml
perf:
  ...
  robustness:
    # For ``uniform`` models, the ``uniform_ticks_range`` attributes are
    # required.
    uniform_ticks_range: [0.0, 0.1]

    # For ``gaussian`` models, the ``gaussian_ticks_stddev_range`` and
    # ``gaussian_ticks_mean_range`` attributes are required.
    gaussian_ticks_mean_range: [0.0, 0.1]
    gaussian_ticks_stddev_range: [0.0, 0.0]

    # For ``gaussian`` models, the ``gaussian_labels_show``,
    # ``gaussian_ticks_src`` attributes are required, and control what is
    # shown for the xticks/xlabels: the mean or stddev values.
    gaussian_ticks_src: stddev
    gaussian_labels_show: stddev

    # The sensors to inject noise into. All shown sensors are optional. If
    # omitted, they will not be affected by noise injection.
    sensors:
      light:
        model: uniform

        # For a ``uniform`` model, the ``range`` attribute is required, and
        # defines the -[level, level] distribution that injected noise will
        # be drawn from.
        range: [0.0, 0.4]

      proximity:
        model: gaussian
        stddev_range: [0.0, 0.1]
        mean_range: [0.0, 0.0]
      ground:
        model: gaussian
        stddev_range: [0.0, 0.1]
        mean_range: [0.0, 0.0]
      steering: # applied to [vel_noise, dist_noise]
        model: uniform
        range: [0.0, 0.1]
      position:
```

(continues on next page)

(continued from previous page)

```
      model: uniform
      range: [0.0, 0.1]

    # The actuators to inject noise into. All shown actuators are
    # optional. If omitted, they will not be affected by noise injection.
    actuators:
      steering: # applied to [noise_factor]
        model: uniform
        range: [0.95, 1.05]
```

Uniform Noise Injection Examples

- `range: [0.0,0.1]` with `cardinality=1` will result in two experiments with uniform noise distributions of `[0.0, 0.0]`, and `[-0.1, 0.1]`.

Gaussian Noise Injection Examples

- `stddev_range: [0.0,1.0]` and `mean_range: [0.0, 0.0]` with `cardinality=2` will result in two experiments with Guassian noise distributions of `Gaussian(0,0)`, `Gaussian(0, 0.5)`, and `Gaussian(0, 1.0)`.

Experiment Definitions

- exp0 - Ideal conditions, in which noise will be applied to the specified sensors and/or actuators at the lower bound of the specified ranges for each.
- exp1-expN - Increasing levels of noise, using the cardinality specified on the command line and the distribution type specified in YAML configuration.

1.3 Projects

1.3.1 SIERRA Projects within TITERRA

FORDYCA

FORDYCA is the SIERRA projects plugin for running large scale swarm experiments with foraging robots which can dynamically create, exploit, and deplete intermediate drop/pickup sites called caches.

Batch Criteria

- *Block Density*
- *Swarm Population Dynamics*
- *Block Quantity*
- *Block Motion Dynamics*
- *Oracle*
- *Task Allocation Policy*
- *Temporal Variance*

Swarm Population Dynamics

Cmdline Syntax

```
population_dynamics.C{cardinality}.F{Factor}[.{dynamics_type}{prob}[...]]
```

- `cardinality` - The # of different values of each of the specified dynamics types to to test with (starting with the one on the cmdline). This defines the cardinality of the batched experiment.
- `Factor` - The factor by which the starting value of all dynamics types specified on the cmdline are increased for each each experiment (i.e., value in last experiment in batch will be `<start value> + cardinality`; a linear increase).
- `dynamics_type` - [B,D,M,R]
  - `B` - Adds birth dynamics to the population. Has no effect by itself, as it specifies a pure birth process with $\lambda = \infty$, μ_b (a queue with an infinite # of robots in it which robots periodically leave from), resulting in dynamic swarm sizes which will increase from N...N over time. Can be specified with along with `D|M|R`, in which case swarm sizes will increase according to the birth rate up until N, given N robots at the start of simulation.
  - `D` - Adds death dynamics to the population. By itself, it specifies a pure death process with $\lambda_d = prob$, and $\mu_d = \infty$ (a queue which robots enter but never leave), resulting in dynamic swarm sizes which will decrease from N...1 over time. Can be specified along with `B|M|R`.
  - `M|R` - Adds malfunction/repair dynamics to the population. If `M` dynamics specified, `R` dynamics must also be specified, and vice versa. Together they specify the dynamics of the swarm as robots temporarily fail, and removed from simulation, and then later are re-introduced after they have been repaired (a queue with λ_m arrival rate and μ_r repair rate). Can be specified along with `B|D`.

Important: The specified λ or μ are the rate parameters of the exponential distribution used to distribute the event times of the Poisson process governing swarm sizes, *NOT* Poisson process parameter, which is their mean; e.g., $\lambda = \frac{1}{\lambda_d}$ for death dynamics.

Examples:

- `population_dynamics.C10.F2p0.B0p001`: 10 levels of population variability applied using a pure birth process with a 0.001 parameter, which will be linearly varied in [0.001,0.001*2.0*10]. For all experiments, the initial swarm is not controlled directly; the value in template input file will be used if swarm size is not set by another variable.

- `population_dynamics.C4.F3p0.D0p001`: 4 levels of population variability applied using a pure death process with a 0.001 parameter, which will be linearly varied in [0.001,0.001*3.0*4]. For all experiments, the initial swarm size is not controlled directly; the value in template input file will be used if swarm size is not set by another variable.
- `population_dynamics.C8.F4p0.B0p001.D0p005`: 8 levels of population variability applied using a birth-death process with a 0.001 parameter for birth and a 0.005 parameter for death, which will be linearly varied in [0.001,0.001*4.0*8] and [0.005, 0.005*4.0*8] respectively. For all experiments, the initial swarm is not controlled directly; the value in the template input file will be used if swarm size is is not set by another variable.
- `population_dynamics.C2.F1p5.M0p001.R0p005`: 2 levels of population variability applied using a malfunction-repair process with a 0.001 parameter for malfunction and a 0.005 parameter for repair which will be linearly varied in [0.001, 0.001*1.5*2] and [0.005, 0.005*1.5*2] respectively. For all experiments, the initial swarm size is not controlled directly; the value in the template input file will be used if swarm size is not set by another variable.

Block Quantity

Cmdline Syntax

```
block_quantity.{block_type}.{increment_type}{N}
```

- `block_type` - `cube` or `ramp`, depending on what type of blocks you want to control the count of.
- `increment_type` - {Log,Linear}. If `Log`, then swarm sizes for each experiment are distributed 1...N by powers of 2. If `Linear` then block counts for each experiment are distributed linearly between 1...N, split evenly into 10 different sizes.
- `N` - The maximum block count.

Examples:

- `block_quantity.cube.Log1024`: Cube block counts 1...1024
- `block_quantity.ramp.Linear1000`: Ramp block counts 100...1000

Block Density

Cmdline Syntax

```
block_density.CD{density}.I{Arena Size Increment}.C{cardinality}
```

- `density` - <integer>p<integer> (i.e. 5p0 for 5.0)
- **`Arena Size Increment` - Size in meters that the X and Y dimensions should** increase by in between experiments. Larger values here will result in larger arenas and more blocks. Must be an integer.
- `cardinality` How many experiments should be generated?

Examples:

- `block_density.CD1p0.I16.C4`: Constant density of 1.0. Arena dimensions will increase by 16 in both X and Y for each experiment in the batch (4 total).

Block Motion Dynamics

Cmdline Syntax

```
block_motion_dynamics.C{cardinality}.F{Factor}.{dynamics_type}{prob}
```

- `cardinality` - The # of different values of each of the specified dynamics types to to test with (starting with the one on the cmdline). This defines the cardinality of the batched experiment.
- `Factor` - The factor by which the starting value of all dynamics types specified on the cmdline are increased for each each experiment (i.e., value in last experiment in batch will be `<start value> + cardinality`; a linear increase).
- `dynamics_type` - [RW]
  - `RW` - Adds random walk dynamics to the arena. Free blocks will execute a random walk with a specified probability each timestep.

Examples:

- `block_motion_dynamics.C10.F2p0.RW0p001`: 10 levels of block motion variability applied using a random walk with a 0.001 probability for each block each timestep, which will be linearly varied in [0.001,0.001*2.0*10]. For all experiments, the initial swarm is not controlled directly; the value in template input file will be used if swarm size is not set by another variable.

Oracle

Cmdline Syntax

```
oracle.{oracle_name}[.Z{population}]
```

- `oracle_name` - {entities, tasks}
  - `entities` - Inject perfect information about locations about entities in the arena, such as blocks and caches.
  - `tasks` - Inject perfect information about task execution and interface times.
- `population` - Static size of the swarm to use (optional).

Examples:

- `oracle.entities.Z16` - All permutations of oracular information about entities in the arena, run with swarms of size 16.
- `oracle.tasks.Z8` - All permutations of oracular information about tasks in the arena, run with swarms of size 8.
- `oracle.entities` - All permuntations of oracular information of entities in the arena (swarm size is not modified).

Task Allocation Policy

Cmdline Syntax

`ta_policy_set.all[.Z{population}]`

`population` - The swarm size to use (optional)

Examples:

- `ta_policy_set.all.Z16`: All possible task allocation policies with swarms of size 16.
- `ta_policy_set.all`: All possible task allocation policies; swarm size not modified.

Temporal Variance

Injecting waveforms into the swarm's environment which affect the individual robot behavior to simulate changing outdoor conditions, changing object sizes/weights, etc.

Note: The graphs generated from this criteria exclude exp0.

Warning: Some of the temporal variance config is very FORDYCA specific; hopefully this will change in the future, or be pushed down to a project-specific extension of a base flexibility class.

Cmdline Syntax

`temporal_variance.{variance_type}{waveform_type}[step_time][.Z{population}]`

- `variance_type` - [BC,BM,M].
  - `BC` - Apply motion throttling to robot speed when it is carrying a block according to the specified waveform.
  - `BM` - Apply the specified waveform when calculating robot block manipulation penalties (pickup, drop, etc.).
  - `M` - Apply the specified waveform to robot motion unconditionally.
- `waveform_type` - {Sine,Square,Sawtooth,Step{U,D},Constant}.
- `step_time` - Timestep the step function should switch (optional).
- `population` - The static swarm size to use (optional).

Examples:

- `temporal_variance.BCSine.Z16` - Block carry sinusoidal variance in a swarm of size 16.
- `temporal_variance.BCStep50000.Z32` - Block carry step variance switch at 50000 timesteps in a swarm of size 32.
- `temporal_variance.BCStep50000` - Block carry step variance switching at 50000 timesteps; swarm size not modified.

The frequency, amplitude, offset, and phase of the waveforms is set via the `main.yaml` configuration file for a project (not an easy way to specify ranges in a single batch criteria definition string). The relevant section is shown below.

For the {Sine,Square,Sawtooth} waveforms, the cardinality of the batched experiment is determined by: (Size of Hz list -1) * (Size of BC_amp/BM_amp list - 1).

YAML Config

```yaml
perf:
  ...
  flexibility:
    # The range of Hz to use for generated waveforms. Applies to Sine,
    # Sawtooth, Square waves. There is no limit for the length of the list.
    hz:
      - frequency1
      - frequency2
      - frequency3
      - ...
    # The range of block manipulation penalties to use if that is the type of
    # applied temporal variance (BM). Specified in timesteps. There is no
    # limit for the length of the list.
    BM_amp:
      - penalty1
      - penalty2
      - penalty3
      - ...
   # The range of block carry penalties to use if that is the type of applied
   # temporal variance (BC). Specified as percent slowdown: [0.0, 1.0]. There
   #is no limit for the length of the list.
   BC_amp:
      - percent1
      - percent2
      - percent3
      - ...

   # The range of motion throttle penalties to use if that is the type of
   # applied temporal variance (M). Specified as percent slowdown: [0.0,
   # 1.0]. There is no limit for the length of the list.
   M_amp:
      - percent1
      - percent2
      - percent3
      - ...
```

Experiment Definitions

- exp0 - Ideal conditions, which is a `Constant` waveform with amplitude `BC_amp[0]`, `BM_amp[0]`, `M_amp[0]` depending.
- exp1-expN
  - Cardinality of `|hz|` * `|BM_amp|` if the variance type is `BM` and the waveform type is Sine, Square, or Sawtooth.
  - Cardinality of `|hz|` * `|BC_amp|` if the variance type is `BC` and the waveform type is Sine, Square, or Sawtooth.
  - Cardinality of `|hz|` * `|M_amp|` if the variance type is `M` and the waveform type is Sine, Square, or Sawtooth.
  - Cardinality of `|BM_amp|` if the variance type is `BM` and the waveform type is StepU, StepD.

- Cardinality of `|BC_amp|` if the variance type is `BC` and the waveform type is StepU, StepD.
- Cardinality of `|M_amp|` if the variance type is `M` and the waveform type is StepU, StepD.

SIERRA Command Line Extensions

General Options

```
usage: sierra-cli [--template-input-file filepath] [--exp-overwrite]
                  [--sierra-root dirpath]
                  [--batch-criteria [<category>.<definition>,...]
                  [[<category>.<definition>,...] ...]]
                  [--pipeline [PIPELINE ...]] [--exp-range EXP_RANGE]
                  [--platform-vc] [--processing-serial] [--n-runs N_RUNS]
                  [--skip-collate] [--plot-log-xscale]
                  [--plot-enumerated-xscale] [--plot-log-yscale]
                  [--plot-regression-lines PLOT_REGRESSION_LINES]
                  [--plot-primary-axis PLOT_PRIMARY_AXIS] [--plot-large-text]
                  [--plot-transpose-graphs] [--scenario <block dist>.AxBxC]
                  [--controller {d0, d1, d2}.<controller>]
```

Multi-stage options

Options which are used in multiple pipeline stages

--template-input-file The template `.xml` input file for the batch experiment. Beyond the requirements for the specific `--platform`, the content of the file can be any valid XML, with the exception of the SIERRA requirements detailed in ln-sierra-tutorials-project-template-input-file.

Tip: Used by stage {1,2,3,4}; can be omitted otherwise. If omitted: N/A.

--exp-overwrite When SIERRA calculates the batch experiment root (or any child path in the batch experiment root) during stage {1, 2}, if the calculated path already exists it is treated as a fatal error and no modifications to the filesystem are performed. This flag overwrides the default behavior. Provided to avoid accidentally overwrite input/output files for an experiment, forcing the user to be explicit with potentially dangerous actions.

Tip: Used by stage {1,2}; can be omitted otherwise. If omitted: N/A.

Default: False

--sierra-root Root directory for all SIERRA generated and created files.

Subdirectories for controllers, scenarios, experiment/experimental run inputs/outputs will be created in this directory as needed. Can persist between invocations of SIERRA.

Tip: Used by stage {1,2,3,4,5}; can be omitted otherwise. If omitted: N/A.

Default: "<home directory>/exp"

--batch-criteria

Definition of criteria(s) to use to define the experiment.

Specified as a list of 0 or 1 space separated strings, each with the following general structure:

`<category>.<definition>`

`<category>` must be a filename from the `core/variables/` or from a `--project <project>/variables/` directory, and `<definition>` must be a parsable namne (according to the requirements of the criteria defined by the parser for `<category>`).

Tip: Used by stage {1,2,3,4,5}; can be omitted otherwise. If omitted: N/A.

Default: []

--pipeline

Define which stages of the experimental pipeline to run:

- Stage1 - Generate the experiment definition from the template input file, batch criteria, and other command line options. Part of default pipeline.
- Stage2 - Run a previously generated experiment. Part of default pipeline.
- Stage3 - Post-process experimental results after running the batch experiment; some parts of this can be done in parallel. Part of default pipeline.
- Stage4 - Perform deliverable generation after processing results for a batch experiment, which can include shiny graphs and videos. Part of default pipeline.
- Stage5 - Perform graph generation for comparing controllers AFTER graph generation for batch experiments has been run. Not part of default pipeline.

Default: [1, 2, 3, 4]

--exp-range

Set the experiment numbers from the batch to run, average, generate intra-experiment graphs from, or generate inter-experiment graphs from (0 based). Specified in the form `min_exp_num:max_exp_num` (closed interval/inclusive). If omitted, runs, averages, and generates intra-experiment and inter-experiment performance measure graphs for all experiments in the batch (default behavior).

This is useful to re-run part of a batch experiment in HPC environments if SIERRA gets killed before it finishes running all experiments in the batch, or to redo a single experiment with real robots which failed for some reason.

Tip: Used by stage {2,3,4}; can be omitted otherwise. If omitted: N/A.

--platform-vc

For applicable `--platforms`, enable visual capturing of run-time data during stage 2. This data can be frames (i.e., .png files), or rendering videos, depending on the platform. If the captured data was frames, then SIERRA can render the captured frames into videos during stage 4. If the selected `--platform` does not support visual capture, then this option has no effect. See ln-sierra-usage-rendering-platform for full details.

Tip: Used by stage {1,4}; can be omitted otherwise. If omitted: N/A.

Default: False

--processing-serial

If TRUE, then results processing/graph generation will be performed serially, rather than using parallellism where possible.

Tip: Used by stage {3,4}; can be omitted otherwise. If omitted: N/A.

Default: False

--n-runs

The # of experimental runs that will be executed and their results processed to form the result of a single experiment within a batch.

If `--platform` is a simulator and `--exec-env` is something other than `hpc.local` then this may be be used to determine the concurrency of experimental runs.

Tip: Used by stage {1,2}; can be omitted otherwise. If omitted: N/A.

--skip-collate

Specify that no collation of data across experiments within a batch (stage 4) or across runs within an experiment (stage 3) should be performed. Useful if collation takes a long time and multiple types of stage 4 outputs are desired. Collation is generally idempotent unless you change the stage3 options (YMMV).

Tip: Used by stage {3,4}; can be omitted otherwise. If omitted: N/A.

Default: False

--plot-log-xscale

Place the set of X values used to generate intra- and inter-experiment graphs into the logarithmic space. Mainly useful when the batch criteria involves large system sizes, so that the plots are more readable.

Tip: Applicable graphs:

- `SummaryLineGraph`

Tip: Used by stage {4,5}; can be omitted otherwise. If omitted: N/A.

Default: False

--plot-enumerated-xscale Instead of using the values generated by a given batch criteria for the X values, use an enumerated list[0, ..., len(X value) - 1]. Mainly useful when the batch criteria involves large system sizes, so that the plots are more readable.

Tip: Applicable graphs:

- `SummaryLineGraph`

Tip: Used by stage {4,5}; can be omitted otherwise. If omitted: N/A.

Default: False

--plot-log-yscale Place the set of Y values used to generate intra - and inter-experiment graphs into the logarithmic space. Mainly useful when the batch criteria involves large system sizes, so that the plots are more readable.

Tip: Applicable graphs:

- `SummaryLineGraph`
- `StackedLineGraph`

Tip: Used by stage {4,5}; can be omitted otherwise. If omitted: N/A.

Default: False

--plot-regression-lines For all 2D generated scatterplots, plot a linear regression line and the equation of the line to the legend.

Tip: Applicable graphs:

- `SummaryLineGraph`

Tip: Used by stage {4,5}; can be omitted otherwise. If omitted: N/A.

--plot-primary-axis This option allows you to override the primary axis, which is normally is computed based on the batch criteria.

For example, in a bivariate batch criteria composed of

- ln-sierra-platform-argos-bc-population-size on the X axis (rows)
- Another batch criteria which does not affect system size (columns)

Metrics will be calculated by *computing* across .csv rows and *projecting* down the columns by default, since system size will only vary within a row. Passing a value of 1 to this option will override this calculation, which can be useful in bivariate batch criteria in which you are interested in the effect of the OTHER non-size criteria on various performance measures.

0=criteria of interest varies across *rows*.

1=criteria of interest varies across *columns*.

This option only affects generating graphs from bivariate batch criteria.

Tip: Applicable graphs:

- `Heatmap`
- `StackedLineGraph`

Tip: Used by stage {4,5}; can be omitted otherwise. If omitted: N/A.

--plot-large-text

This option specifies that the title, X/Y axis labels/tick labels should be larger than the SIERRA default. This is useful when generating graphs suitable for two column paper format where the default text size for rendered graphs will be too small to see easily. The SIERRA defaults are generally fine for the one column/journal paper format.

Tip: Used by stage {4,5}; can be omitted otherwise. If omitted: N/A.

Default: False

--plot-transpose-graphs

Transpose the X, Y axes in generated graphs. Useful as a general way to tweak graphs for best use of space within a paper.

Changed in version 1.2.20: Renamed from `--transpose-graphs` to make its relation to other plotting options clearer.

Tip: Applicable graphs:

- `Heatmap`

Tip: Used by stage {4,5}; can be omitted otherwise. If omitted: N/A.

Default: False

--scenario

Which scenario the swarm comprised of robots running the controller specified via `--controller` should be run in.

A scenario is defined as: block distribution type + arena dimensions. This is somewhat tied to foraging and other similar applications for the moment, but this may be modified in a future version of TITERRA.

Valid block distribution types:

- `RN` - Random
- `SS` - Single source
- `DS` - Dual source
- `QS` - Quad source

- PL - Power law

A,B,C are the scenario X,Y,Z dimensions respectively (which can be any postive INTEGER values). All dimensions are required.

Tip: Used by stage {1,2,3,4}; can be omitted otherwise. If omitted: N/A.

--controller Possible choices: d0.CRW, d0.DPO, d0.ODPO, d0.MDPO, d0.OMDPO, d1.BITD_DPO, d1.BITD_ODPO, d1.BITD_OMDPO, d2.BIRTD_DPO, d2.BIRTD_ODPO, d2.BIRTD_OMDPO

Which controller robots will use in the foraging experiment. All robots use the same controller (homogeneous swarms).

Head over to the FORDYCA docs for the descriptions of these controllers.

Tip: Used by stage {1,2,3,4,5}; can be omitted otherwise. Only required for stage 5 if `--scenario-comp` is passed.

Stage1: Generating Experiments

```
usage: sierra-cli [--preserve-seeds PRESERVE_SEEDS] [--no-preserve-seeds]
                  [--n-blocks N_BLOCKS]
                  [--static-cache-blocks STATIC_CACHE_BLOCKS]
```

Stage1: General options for generating experiments

--preserve-seeds Preserve previously generated random seeds for experiments (the default). Useful for regenerating experiments when you change parameters/python code that don't affects experimental outputs (e.g., paths). Preserving/overwriting random seeds is not affected by `--exp-overwrite`.

Default: True

--no-preserve-seeds Opposite of `--preserve-seeds`.

Default: True

--n-blocks # blocks that should be used in the simulation. Can be used to override batch criteria, or to supplement experiments that do not set it so that manual modification of input file is unneccesary.

Tip: Used by stage {1}; can be omitted otherwise. If omitted: N/A.

--static-cache-blocks # of blocks used when the static cache is # respawned (d1 controllers only).

Tip: Used by stage {1}; can be omitted otherwise. If omitted: N/A.

PRISM

PRISM is the SIERRA projects plugin for running large scale swarm experiments with robots constructing or deconstructing large 3D structures composed of heterogeneous blocks.

Batch Criteria

- *Construction Target Specifications*

Construction Target Specifications

Define a construction target to be built during simulation. Multiple targets can be specified.

Cmdline Syntax

```
ct_specs.{class}.{AxBxC}@{D,E,F}
```

- `class`
  - `rectprism` - A rectangular prism defined by the bounding box `AxBxC` with an anchor (lower left corner) at `(D,E,F)`.
  - `ramp` - A sloped structure that robots can drive up, defined by the bounding box `AxBxC` with an anchor (lower left corner) at `(D,E,F)`.

Examples:

- `ct_specs.ramp.8x8x4@4,4,0`: A square ramp structure 4 units tall anchored on its lower left corner at (4,4,0).

SIERRA Command Line Extensions

General Options

Stage1: Generating Experiments

1.3.2 Extra Project Configuration

TITERRA defines some additional configuration options besides those available/required by SIERRA, as shown below.

Main Configuration Extensions

Summary Performance Measures

Within the pointed-to .yaml file for `perf` configuration, the structure is:

```yaml
perf:

  # The title that graphs of raw swarm performance should have (cannot be
  # known a priori for all possible projects during stage 4). This key is
  # required for all batch criteria which for which 'raw' performance graphs
  # should be generated.
  raw_perf_title: 'Swarm Blocks Collected'

  # The Y label for graphs of raw swarm performance (cannot be known a
  # priori for all possible projects during stage 4). This key is required
  # for all batch criteria for which 'raw' performance graphs should be
  # generated.
  raw_perf_ylabel: '\# Blocks'

  # The name of the collated CSV containing overall performance measures
  # for each experiment in the batch (1 per experiment) which should be used
  # for generating performance measures. This key is mandatory.
  inter_perf_csv: 'blocks-transported-cum.csv'

  # The name of the collated CSV containing the count of the average #
  # of robots experiencing inter-robot interference for each experiment in
  # the batch (1 per experiment) which is used in generating performance
  # measures. Mandatory for all batch criteria for which 'self-organization'
  # performance graphs should be generated.
  interference_count_csv: 'interference-in-cum-avg.csv'

  # The name of the collated CSV containing the count of the average
  # duration of a robot experiencing inter-robot interference for each
  # experiment in the # batch (1 per experiment) which should be used for
  # generating performance measures. Mandatory for all batch criteria for
  # which 'self-organization' graphs should be generated.
  interference_duration_csv: 'interference-duration-cum-avg.csv'

  # The CSV file under ``sim_metrics_leaf`` for each experiment
  # which contains the applied environmental variances. This key is
  # required for all batch criteria which for which 'flexibility' performance
  # graphs should be generated.
  tv_environment_csv: 'tv-environment.csv'

  # The CSVfile under ``sim_metrics_leaf`` for each experiment which
  # contains information about temporally fluctuating populations. This key is
  # required for all batch criteria which for which 'flexibility' performance
  # graphs should be generated.
  tv_population_csv: 'tv-population.csv'
```

`perf.robustness` sub-dictionary

See *SAA noise config*.

1.4 TITERRA General Usage Tips

1. Look at scripts under `scripts/`, which are scripts I've used before on MSI (they might no longer work, but they do give you some idea of how to invoke SIERRA).
2. If you are running the FORDYCA project and using a `quad_source` block distribution, the arena should be at least 16x16 (smaller arenas don't leave enough space for caches and often cause segfaults).
3. For `SS,DS` distributions a rectangular 2x1 arena is required. That is, an arena where the X dimension is twice the Y dimension. If you try to run with anything else, you will get an error.
4. For `QS,PL,RN` distributions a square arena is required (if you try to run with anything else, you will get an error).

1.5 Running On The Minnesota Supercomputing Institute (MSI)

1.5.1 MSI Setup

Important: Prior to executing these steps you should have:

1. Completed the ln-sierra-tutorials-hpc-local-setup guide.
2. Read through ln-sierra-usage.
3. Gotten **CORRECT** results on some small scale experiments of interest on your local machine.

You really, *really*, **really**, don't want to be trying to do development/debugging on MSI.

Workflow

1. Get an MSI account (you will need to talk to Maria Gini, my advisor), and verify that you can login to `mesabi` via the following commands, run from your laptop on a UMN computer/UMN wifi (will not work from outside UMN campus without a VPN):

   ```
   ssh <x500>@mesabi.umn.edu
   ```

   Where `<x500>` is your umn x500. If the commands are successful, you have logged into a `mesabi` login node (this is different than a `mesabi` compute node).

   A similar check for `mangi` via the following commands, run from your laptop on a UMN computer/UMN wifi (will not work from outside UMN campus without a VPN):

   ```
   ssh <x500>@mangi.umn.edu
   ```

   If the commands are successful, you have logged into a `mangi` login node (this is different than a `mangi` compute node).

2. Once you can login, you can begin the setup by sourcing the environment definitions:

```
. /home/gini/shared/swarm/bin/msi-env-setup.sh
```

Important: ANYTIME you log into an MSI node (login or compute) to build/run ANYTHING you MUST source this script otherwise things might not work. This includes if you ran the script on a login node and then started an interactive session via job submission with `-p interactive` (the environment is NOT inherited).

3. On an MSI login node (can be any type, as the filesystem is shared across all clusters), install the same python dependencies as in ln-usage, but user local (you obviously don't have admin priveleges on the cluster):

```
pip3 install --user -r requirements/msi.txt
```

This is a one time step. Must be done on a login node, as compute nodes do not always have internet access (apparently?).

4. On an MSI login node, get an interactive job session so you can build your selected project and its dependencies natively to the cluster you will be running on (mangi/mesabi) for maximum speed:

```
srun -N 1 --ntasks-per-node=4  --mem-per-cpu=1gb -t 1:00:00 -p interactive --pty␣
↪bash
```

The above command, when it returns, will give you 1 hour of time on an actual compute node. You know you are running/building on a compute node rather than a login node on mangi/mesabi when the hostname is `cnXXXX` rather than `lnXXXX`.

5. In your interactive session clone the bootstrap repo and follow the instructions on the bootstrap README:

```
git clone https://github.com/swarm-robotics/bootstrap.git
```

In general, the only argument you should need for the script is `--msi`.

1.5.2 MSI Runtime

Important: Prior to executing these steps you should have:

1. Already successfully completed the *MSI Setup* steps. If not–shoo! Go do that first.

Workflow

1. Read the documentation for SLURM scripts and MSI job submission to a partition on your chosen cluster:
   - https://www.msi.umn.edu/partitions https://www.msi.umn.edu/content/job-submission-and-scheduling-slurm

   Seriously–**READ THEM**.

2. Now that you have read the MSI docs, copy and modify `scripts/example.sh` script in the TITERRA repo for your experiment/batch experiment, and modify as needed:

   SLURM parameters you **DO** need to change:

   - The email (I don't want to get emails about **YOUR** jobs).

   SLURM parameters you **MIGHT** need to change:

   - The number of requested nodes.

Other things you **MIGHT** need to change:

- The location of the FORDYCA and SIERRA repos in the script, if you did not clone them to the same location as specified earlier in these steps.

3. Have your jobs script reviewed before submission (will likely save you a *LOTS* of time fighting with the job submission system).

4. Submit your job via:

```
sbatch your-script.sh
```

Note the job number that the job submission will print when you run this command–it is important to track job progress and to figure out what happened when (not if) things go wrong.

It is also generally a good idea to get in the habit of giving your PBS scripts (fairly) unique and descriptive names when submitting, in order to make tracking down what stderr/stdout file belongs to what job when once you've run a few dozen jobs.

You will get an email when your job starts, aborts because of error, and finishes. To view the in-progress stdout of the job, look in your home directory for output/error files:

- `$HOME/R-your-jobname.1234.out` where `1234` is the job number of your job. MSI will create this file in the directory you submit the job from, and direct your job's stdout to it.
- `$HOME/R-your-jobname.1234.err` where `1234` is the job number of your job. MSI will create this file in the directory you submit the job from, and direct your job's stderr to it.

5. Reap the rewards of research!

1.6 API Reference

1.6.1 titerra.projects

titerra.projects.common

titerra.projects.common.cmdline

Command line parsing and validation for the TITAN project.

titerra.projects.common.generators

titerra.projects.common.generators.argos

Extensions to `PlatformExpDefGenerator` common to all TITAN scenarios which use ARGoS.

- *BaseScenarioGenerator*: Init the object.
- *ForagingScenarioGenerator*: Init the object.
- *ForagingSSGenerator*: Generates XML changes for single source foraging.
- *ForagingDSGenerator*: Generates XML changes for dual source foraging.
- *ForagingQSGenerator*: Generates XML changes for quad source foraging.
- *ForagingPLGenerator*: Generates XML changes for powerlaw source foraging.

- *ForagingRNGenerator*: Generates XML changes for random foraging.

class titerra.projects.common.generators.argos.**BaseScenarioGenerator**(**args*, ***kwargs*)

Inheritance

__doc__ = None

__init__(**args*, ***kwargs*) → None

__module__ = 'titerra.projects.common.generators.argos'

generate_arena_map(*exp_def: sierra.core.experiment.definition.XMLExpDef*, *the_arena: titerra.platform.argos.variables.arena.RectangularArena*) → None
Generate XML changes for the specified arena map configuration.

Writes generated changes to the simulation definition pickle file.

generate_block_count(*exp_def: sierra.core.experiment.definition.XMLExpDef*) → None
Generates XML changes for # blocks in the simulation. If specified on the cmdline, that quantity is used (split evenly between ramp and cube blocks).

Writes generated changes to the simulation definition pickle file.

static generate_block_dist(*exp_def: sierra.core.experiment.definition.XMLExpDef*, *block_dist: titerra.platform.argos.variables.block_distribution.BaseDistribution*) → None
Generate XML changes for the specified block distribution.

Does not write generated changes to the simulation definition pickle file.

generate_convergence(*exp_def: sierra.core.experiment.definition.XMLExpDef*)
Generate XML changes for calculating swarm convergence.

Does not write generated changes to the simulation definition pickle file.

class titerra.projects.common.generators.argos.**ForagingScenarioGenerator**(**args*, ***kwargs*)

Inheritance

__doc__ = None

__init__(**args*, ***kwargs*) → None

__module__ = 'titerra.projects.common.generators.argos'

generate() → sierra.core.experiment.definition.XMLExpDef
Generate XML modifications common to all ARGoS experiments.

class titerra.projects.common.generators.argos.**ForagingSSGenerator**(**args*, ***kwargs*)
Generates XML changes for single source foraging.

This includes:

- Rectangular 2x1 arena
- Single source block distribution
- One nest

Inheritance

__doc__ = '\n Generates XML changes for single source foraging.\n\n This includes:\n\n - Rectangular 2x1 arena\n - Single source block distribution\n - One nest\n '

__init__(**args*, ***kwargs*) → None

__module__ = 'titerra.projects.common.generators.argos'

generate()
Generate XML modifications common to all ARGoS experiments.

class titerra.projects.common.generators.argos.**ForagingDSGenerator**(**args*, ***kwargs*)
Generates XML changes for dual source foraging.

This includes:

- Rectangular 2x1 arena
- Dual source block distribution
- One nest

Inheritance

__doc__ = '\n Generates XML changes for dual source foraging.\n\n This includes:\n\n - Rectangular 2x1 arena\n - Dual source block distribution\n - One nest\n '

__init__(**args*, ***kwargs*) → None

__module__ = 'titerra.projects.common.generators.argos'

generate()
Generate XML modifications common to all ARGoS experiments.

class titerra.projects.common.generators.argos.**ForagingQSGenerator**(**args*, ***kwargs*)
Generates XML changes for quad source foraging.

This includes:

- Square arena
- Quad source block distribution
- One nest

Inheritance

__doc__ = '\n Generates XML changes for quad source foraging.\n\n This includes:\n\n - Square arena\n - Quad source block distribution\n - One nest\n '

__init__(**args*, ***kwargs*) → None

__module__ = 'titerra.projects.common.generators.argos'

generate()
Generate XML modifications common to all ARGoS experiments.

class titerra.projects.common.generators.argos.**ForagingPLGenerator**(**args*, ***kwargs*)
Generates XML changes for powerlaw source foraging.

This includes:

- Square arena
- Powerlaw block distribution
- One nest

Inheritance

__doc__ = '\n Generates XML changes for powerlaw source foraging.\n\n This includes:\n\n - Square arena\n - Powerlaw block distribution\n - One nest\n '

__init__(**args*, ***kwargs*) → None

__module__ = 'titerra.projects.common.generators.argos'

generate()
Generate XML modifications common to all ARGoS experiments.

class titerra.projects.common.generators.argos.**ForagingRNGenerator**(**args*, ***kwargs*)
Generates XML changes for random foraging.

This includes:

- Square arena
- Random block distribution
- One nest

Inheritance

__doc__ = '\n Generates XML changes for random foraging.\n\n This includes:\n\n - Square arena\n - Random block distribution\n - One nest\n '

__init__(**args*, ***kwargs*) → None

__module__ = 'titerra.projects.common.generators.argos'

generate()
Generate XML modifications common to all ARGoS experiments.

titerra.projects.common.generators.ros1gazebo

Extensions to PlatformExpDefGenerator common to all TITAN scenarios which use ROS with real robots.

- *BaseScenarioGenerator*: Init the object.
- *ForagingScenarioGenerator*: Generates XML changes for foraging. Because we are working with real robots,

class titerra.projects.common.generators.ros1gazebo.**BaseScenarioGenerator**(**args*, ***kwargs*)

Inheritance

__doc__ = None

__init__(**args*, ***kwargs*) → None

__module__ = 'titerra.projects.common.generators.ros1gazebo'

class titerra.projects.common.generators.ros1gazebo.**ForagingScenarioGenerator**(**args*, ***kwargs*)

Generates XML changes for foraging. Because we are working with real robots, there is no arena setup to do with SIERRA (i.e., you have to manually setup the environment).

Inheritance

__doc__ = '\n Generates XML changes for foraging. Because we are working with real robots,\n there is no arena setup to do with SIERRA (i.e., you have to manually setup\n the environment).\n '

__init__(**args*, ***kwargs*) → None

__module__ = 'titerra.projects.common.generators.ros1gazebo'

generate() → sierra.core.experiment.definition.XMLExpDef

titerra.projects.common.generators.ros1robot

Extensions to PlatformExpDefGenerator common to all TITAN scenarios which use ROS with real robots.

- *BaseScenarioGenerator*: Init the object.
- *ForagingScenarioGenerator*: Generates XML changes for foraging. Because we are working with real robots,

class titerra.projects.common.generators.ros1robot.**BaseScenarioGenerator**(**args*, ***kwargs*)

Inheritance

__doc__ = None

__init__(**args*, ***kwargs*) → None

__module__ = 'titerra.projects.common.generators.ros1robot'

class titerra.projects.common.generators.ros1robot.**ForagingScenarioGenerator**(**args*, ***kwargs*)

Generates XML changes for foraging. Because we are working with real robots, there is no arena setup to do with SIERRA (i.e., you have to manually setup the environment).

Inheritance

__doc__ = '\n Generates XML changes for foraging. Because we are working with real robots,\n there is no arena setup to do with SIERRA (i.e., you have to manually setup\n the environment).\n '

__init__(**args*, ***kwargs*) → None

__module__ = 'titerra.projects.common.generators.ros1robot'

generate() → sierra.core.experiment.definition.XMLExpDef

titerra.projects.common.generators.scenario_generator_parser

titerra.projects.common.generators.utils

Utils to support generator extensions for the the TITAN project.

1.6.2 titerra.variables

titerra.variables.batch_criteria

Extensions to SIERRA batch criteria classes for the TITAN project.

- *IConcreteBatchCriteria*: 'Final' interface for user-visible batch criteria.
- *IPMQueryableBatchCriteria*: Mixin interface for batch criteria which can be queried during stage
- *BivarBatchCriteria*: Combination of the definition of two separate batch criteria.

class titerra.variables.batch_criteria.**IConcreteBatchCriteria**
'Final' interface for user-visible batch criteria.

Inheritance

__doc__ = "\n 'Final' interface for user-visible batch criteria.\n "

__module__ = 'sierra.core.variables.batch_criteria'

graph_xlabel(*cmdopts: Dict[str, Any]*) → str
Get the X-label for a graph.

Returns The X-label that should be used for the graphs of various performance measures across batch criteria.

graph_xticklabels(*cmdopts: Dict[str, Any]*, *exp_names: Optional[List[str]] = None*) → List[str]
Calculate X axis tick labels for graph generation.

Parameters

- **cmdopts** – Dictionary of parsed command line options.
- **exp_names** – If not None, then these directories will be used to calculate the labels, rather than the results of gen_exp_names().

graph_xticks(*cmdopts: Dict[str, Any]*, *exp_names: Optional[List[str]] = None*) → List[float]
Calculate X axis ticks for graph generation.

Parameters

- **cmdopts** – Dictionary of parsed command line options.
- **exp_names** – If not None, then this list of directories will be used to calculate the ticks, rather than the results of gen_exp_names().

class titerra.variables.batch_criteria.**IPMQueryableBatchCriteria**
Mixin interface for batch criteria which can be queried during stage 4 to generate performance measures.

Inheritance

__doc__ = 'Mixin interface for batch criteria which can be queried during stage\n 4 to generate performance measures.\n '

__module__ = 'titerra.variables.batch_criteria'

pm_query(*pm: str*) → bool

Parameters **pm** – A possible performance measure to generate from the results of the batch experiment defined by this object.

Returns *True* if the specified pm should be generated for the current object, and *False* otherwise.

class titerra.variables.batch_criteria.**BivarBatchCriteria**(**args*, ***kwargs*)
Combination of the definition of two separate batch criteria.

Inheritance

__doc__ = '\n Combination of the definition of two separate batch criteria.\n '

__init__(*args, **kwargs) → None

__module__ = 'titerra.variables.batch_criteria'

pm_query(*pm: str*) → bool

1.6.3 titerra.platform

1.6.4 titerra.tools

1.7 C++ Code For Projects

- FORDYCA
- PRISM

1.8 Other Projects: (in descending probability of interest)

- SIERRA
- COSM
- RCPPSW
- RCSW
- LIBRA

PYTHON MODULE INDEX

t

INDEX

Symbols

B

T