State Estimation Observation Model Setup

Observation Model Setup

Having defined the link ends, you can now define and create the observation models. Below the general workflow for this is discussed.

Defining observation settings

which models?

Tudat supports a diverse set of observation types, (see Observation models for a comprehensive list). The creation of an observation model is done in a similar manner as models used for the numerical propagation: an object defining the settings of each observation model is created, which is then processed to create the actual observation model.

mention two-step approach?

A basic observation is defined by a combination of its type, and a link definition. Most observation types may (or must) have additional settings, such as light-time corrections, biases, etc.

put this in a 'note' block?

Below is a basic example of creating settings for two observation models.s

why not re-use link ends example?

as a kind of step-by-step approach

```
# Define link ends
one_way_nno_mex_link_ends = dict( );
one_way_nno_mex_link_ends[ transmitter ] =
estimation_setup.observation.body_reference_point_link_end_id( "Earth", "NNO"
);
one_way_nno_mex_link_ends[ receiver ] =
estimation_setup.observation.body_origin_link_end_id( "MeX" );
one_way_nno_mex_link_definition = estimation_setup.link_definition(
one_way_nno_mex_link_ends )


# Create list of observation settings
observation_settings_list = list()
observation_settings_list.append( observation_setup.one_way_range(
one_way_nno_mex_link_ends ) )
observation_settings_list.append( observation_setup.one_way_open_loop_doppler(
one_way_nno_mex_link_ends ) )
```

again mention ('note'?)

that

put before code?

This defines a one-way range and one-way Doppler (open-loop) observable, each with the New Norcia ESTRACK station/Mars Express as transmitter/receiver (see Link ends setup). These settings are put into the `observation_settings_list` list. Note that this list of observation model settings can be extended with any number of entries, with any number of link ends. The only limitation is that you may not have duplicate entries of link ends *and* observable type (as this would essentially define an identical type of observation).

one links end def. for a obs.

When defining observation models, you can for most types of models define settings for:

sometimes it's tudat; sometimes tudatPy?

- **Biases:** A bias in TudatPy is applied to the observable after its 'ideal' value computed from the environment is computed. You can find a list of settings for observation biases in our API documentation
- **Light-time corrections:** When using an observable that involves the observation of one point/body in space by another (including any observable that involves the exchange of elecromagnetic signals), it is automatically assumed that the signal travels at the speed of light, and the associated light-time is determined when calcialting the observable. Deviations from the signal's ideal trajectory (straight line at speed of light) may be defind by adding light-time correction settings, as listed in our API documentation

calculating?

- **Light-time convergence settings:** Calculating the light time between two link ends requires the iterative solution of the light-time equation. Default settings for convergence criteria for this solution are implemented, but a user may modify these settings if so desired. The associated settings object can be created using the `light_time_convergence_settings()` function.

The above options are added to the calls of the observation model settings factory functions. Below is an example

```
# Define link ends
one_way_nno_mex_link_ends = dict( );
one_way_nno_mex_link_ends[ transmitter ] =
estimation_setup.observation.body_reference_point_link_end_id( "Earth", "NNO"
);
one_way_nno_mex_link_ends[ receiver ] =
estimation_setup.observation.body_origin_link_end_id( "MeX" );
one_way_nno_mex_link_definition = estimation_setup.link_definition(
one_way_nno_mex_link_ends )

# Define settings for light-time calculations
light_time_correction_settings = [
observation_setup.first_order_relativistic_correction( [ 'Sun' ] )]

# Define settings for range bias
range_bias_settings = observation_setup.absolute_bias( 0.01 )

# Create list of observation settings
observation_settings_list = list()
observation_settings_list.append( observation_setup.one_way_range(
    one_way_nno_mex_link_ends
    light_time_correction_settings = light_time_correction_settings,
    bias_settings = range_bias_settings ) )
observation_settings_list.append( observation_setup.one_way_open_loop_doppler(
    one_way_nno_mex_link_ends,
    light_time_correction_settings = light_time_correction_settings ) )
```

– with – correction?

where we have defined that, for both observation models for which settings are created, the light-time calculation will take into account the first-order relativistic correction of the Sun, by using the `first_order_relativistic_correction()` function. For the range observable, we have defined an absolute bias of 1 cm (0.01 m) using the `absolute_bias()`, while leaving the Doppler observable unbiased.

Creating the models

Depending on the type of simulation you are using, you can use one of two manners in which to create the observation simulators from the observation settings:

- Create a set of observation simulators directly, using the `create_observation_simulators()` function:

remove or expand level of detail

a

```
# Create physical environment (as set of physical bodies)
bodies = ...

# Create settings for observation models
observation_settings_list = list( )
...

# Create observation simulators
observation_simulators = create_observation_simulators(
observation_settings_list, bodies )
```

extremely vague; either

- Create an : `Estimator` object (discussed further here), which creates the observation simulators automatically

```
~~# Create physical environment (as set of physical bodies)~~
estimator = Estimator(...)

# Extract observation simulators
observation_simulators = estimator.observation_simulators
```

like what happens with?

In either case, the `observation_simulators` variable is a list of objects derived from `ObservationSimulator`, with a single object responsible for the simulation of a single *type* of observable (*e.g.* one-way range, one-way Doppler, *etc.*). ~~The `observation_simulators` list of simulators can then be used for Observation Simulation.~~

too much observ. sim.! list of ObsSim objects

For 'manual' simulation of observations, you can extract an `ObservationModel` object from the `ObservationSimulator` (TODO example). Whereas the `ObservationSimulator` is responsible for *all* observations of a given kind, the `ObservationModel` simulates observations of a single kind, for a single set of link ends (e.g. one-way range observations between a given ground station and a single spacecraft). Details on the associated options can be found in the API documentation.

when to use? mention!

better highlight already above; not entirely clear there!