Overview of the Code

ECE/MAE 5340 Planning for Mobile Robotics
Dr. Greg Droge

A Sim is an Object with Variables and Four Functions

```python
class Sim(Protocol[StateType]):
    """Basic class formulation for simulating

    Attributes:
        data(Data[StateType]): The simulation data
        params(SimParameters[StateType]): Simulation parameters
        lock(Lock): Lock for thread safe plotting
        stop(Event): The sim should stop when the event is true

    """
    data: Data[StateType] # The simulation data
    params: SimParameters[StateType] # Simulation parameters
    lock: Lock # Lock for updating the current state
    stop: Event # The sim should stop when the event is true

    def update(self) -> None:
        """Calls all of the update functions"""

    def post_process(self) -> None:
        """Process the results"""

    def update_plot(self) -> None:
        """Plot the current values and state. Should be done with
        lock to avoid simultaneous access of state.
        """

    def store_data_slice(self, sim_slice: Slice[StateType]) -> None:
        """Stores data after update"""
```

Variables

Stores the current state, time, measurements, and any other data produced by the sim

Stores how the simulation will run (frequency of update, plotting, size of simulation time step, end time)

Used to lock the thread for accessing the "data" object for safe parallel processing

Used to tell the sim to stop running

Functions

Called whenever the sim state should be updated

Called after sim is finished to process results

Called when plotting

Used to store the latest data

Note: Any class that has all of these attributes and functions is a sim (Duck Typing)

Data Stored in the Data Object

```
class Data(Generic[StateType]):
    """Stores the changing simulation information

    Attributes:
        current(Slice): Stores the current slice of data to be read
        next(Slice): Stores the next data to be created
        state_traj(NDArray[Any]): Each column corresponds to a
        trajectory data
        time_traj(NDArray[Any]): vector Each element is the time for
        the state in question
        traj_index_latest(int): Index into the state and time
        trajectory of the latest data
        control_traj(NDArray[Any]): Each column corresponds to a
        control input vector
        range_bearing_latest(RangeBearingMeasurements): Stores the
        latest data received for range-bearing measurements
    """
```

State, time, and control input data at single point in time

State, time, and control input data over the entirety of the simulation

Sensor data at current point in time

Using the *start_sim()* Function

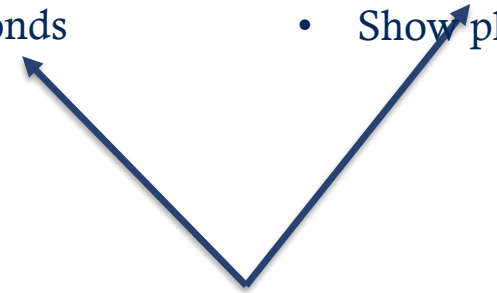
- Basic function that uses Python's *asyncio* library to call two functions asynchronously

run_simulation(…)

- **While** sim time is valid and stop not set
  - Copy over latest data
  - Call simulation *update(…)*
  - Call simulation *store_data_slice(…)*
  - Wait for *sim_update_period* seconds
- Store final data
- Call simulation *post_process(…)*

continuous_plotting(…)

- Create the plot
- **While** stop not set
  - Call simulation *update_plot(…)*
  - Wait for *sim_plot_period* seconds
- Show plots until closed



These *wait* statements allow the other loop to run

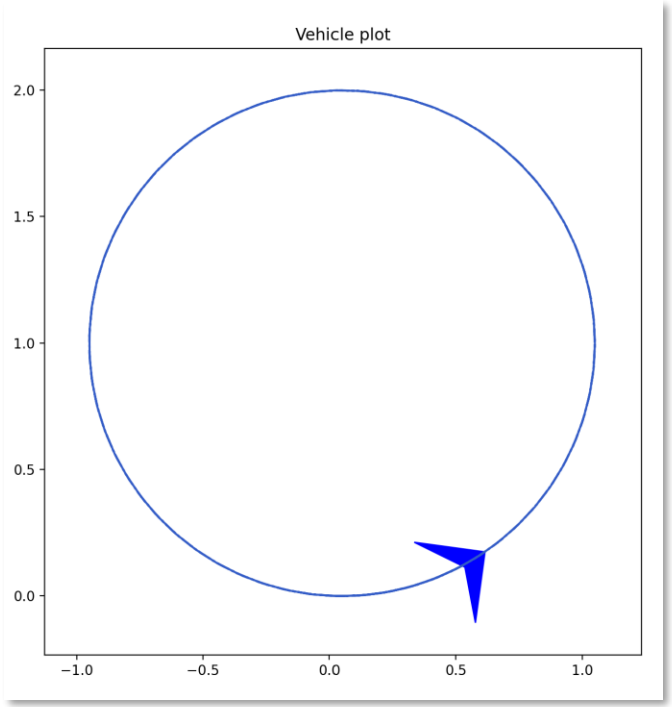
Note, the asyncio call will run the two functions on the same thread

Example using the *start_sim()* Function

- Simple sim (*python .\py_sim\launch\simple_sim.py*) has a vehicle move in a circular motion while plotting its state and position over time

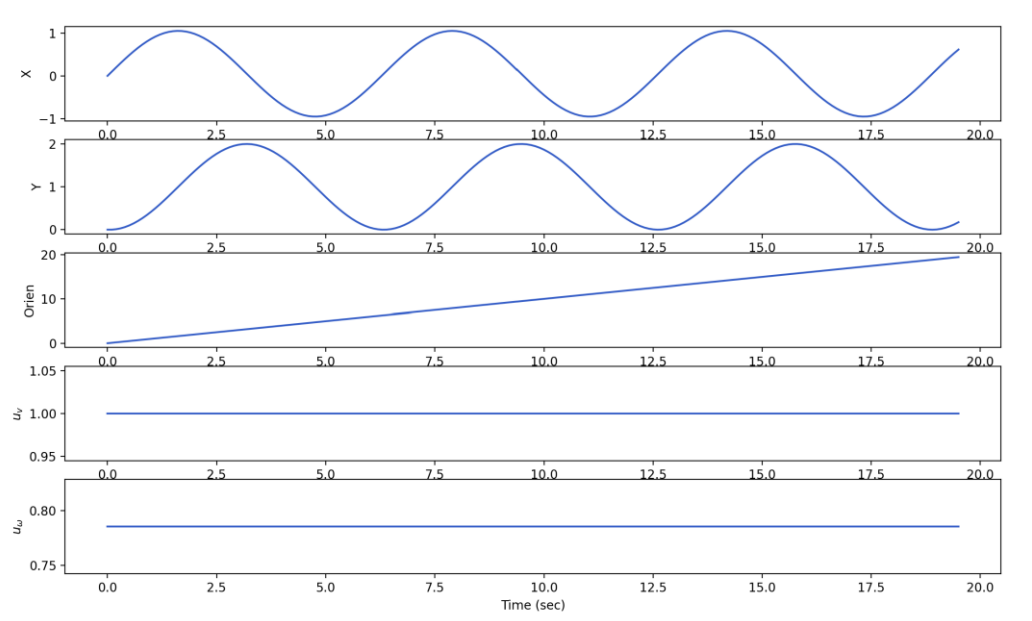
run_simulation(…)

- Updates the state based on a unicycle motion model to move in circle
- Stores the state and trajectory information



continuous_plotting(…)

- Plots the state, 2D trajectory, and the individual state values over time



SingleAgentSim – The Major Sim Implemented

```python
class SingleAgentSim(Generic[StateType]):
    """Implements the main functions for a single agent simulation

    Attributes:
        params(SimParameters): parameters for running the simulation
        data(Data): Stores the current and next slice of information
        lock(Lock): Lock used for writing to the data
        stop(Event): Event used to indicate that the simulator should be stopped
        figs(list[Figure]): Stores the figures that are used for plotting
        axes(dict[Axes, Figure]): Stores the axes used for plotting and their corresponding
        figures
        state_plots(list[StatePlot[StateType]]): Plots depending solely on state
        data_plots(list[DataPlot[StateType]]): Plots that depend on the data
    """
    def __init__(self, ···

    def update(self) -> None:
        """Performs all the required updates"""
        raise NotImplementedError("Update function must be implemented")

    def update_plot(self) -> None: ···

    def store_data_slice(self, sim_slice: Slice[StateType]) -> None: ···

    def post_process(self) -> None: ···

    def initialize_data_storage(self, n_inputs: int) -> None: ···
```

Note: A data-driven design is used in the sim. All data is defined as sim attributes and other functions just update those attributes as needed

Main sim object attributes

Parameters for plotting

Main sim functions

- *update(…)* is left for children classes
- *update_plot(…)* loops through all plotters and draws the figure
- *store_data_slice(…)* stores the current data info into the trajectory variables
- *post_process(…)* prints final state (kind of pointless)

Initialize the storage values for the entirety of the sim – called from the constructor

The Different Simulations – *SimpleSim* and *VectorFollower*

- The simulations all inherit the *SingleAgentSim*, differing in additional attributes and the *update(…)* function implementation

SimpleSim

Attributes
- *dynamics:* Function used to update the state
- *controller:* Function used to calculate the controller
- *dynamic_params:* parameters for the dynamics
- *control_params:* parameters for the controller

update(…)
- Calculates the controller given the state
- Updates the state given the control input
- Updates the time

VectorFollower

Attributes
- *dynamics:* Function used to update the state
- *controller:* Vector-based controller
- *dynamic_params:* parameters for the dynamics
- *control_params:* parameters for the controller
- *vector_field:* Function to calculate vector based on state

update(…)
- Calculates a vector to be followed
- Calculates the controller given the state and vector
- Updates the state given the control input
- Updates the time

Green is used to show the differences from the SimpleSim

NavFieldFollower and *NavVectorFollower*

NavFieldFollower

Attributes

- *dynamics:* Function used to update the state
- *controller:* Vector-based controller
- *dynamic_params:* parameters for the dynamics
- *control_params:* parameters for the controller
- *vector_field:* Function to calculate vector based on state
- *world:* Polygon world in which navigation occurs
- *sensor:* Range sensor used to detect obstacles (note that sensors aren't actually used)

update(…)

- Updates sensor measurements
- Calculates a vector to be followed from state and time
- Calculates the controller given the state and vector
- Updates the state given the control input
- Updates the time

NavVectorFollower

Attributes

- *dynamics:* Function used to update the state
- *controller:* Vector-based controller
- *dynamic_params:* parameters for the dynamics
- *control_params:* parameters for the controller
- *vector_field:* Go-to-goal vector field
- *world:* Polygon world in which navigation occurs
- *sensor:* Range sensor used to detect obstacles
- *carrot:* Object used to calculate a goal point

update(…)

- Updates sensor measurement
- Updates carrot goal point
- Calculates a vector to be followed from state, goal, and obstacles
- Calculates the controller given the state and vector
- Updates the state given the control input
- Updates the time

Green is used to show the differences from the SimpleSim

8

Plotting in the *SingleAgentSim*

- The *SingleAgentSim* (and, thus, all of its children) require a plot manifest

```
class PlotManifest(Generic[StateType]):
    """Defines data necessary for plotting

    Attributes:
        figs(list[Figure]): Figures created for plotting
        axes(dict[str, Axes]): Mapping of axis name to the axis
        state_plots(list[StatePlot[StateType]]): List of all the state plots
            (plots that only depend on the state)
        data_plots(list[DataPlot[StateType]]): List of all the data plots
            (plots that depend on many data elements)
    """
```

The state and data plots have a *plot(…)* function that is called within the *SingleAgentSim.update_plots(…) function*

- The *PlotManifest* can be created using *create_plot_manifest()* – The following are some plots that can be created
  - State/input time series
  - Position (as triangle or dot)
  - Occupancy grid
  - Polygon world
  - 2D Position trajectory
  - Range measurement location and/or lines
  - Plan being followed
  - Graph of points
  - Goal, carrot points

Plot Example with Most Plot Options Enabled

Example created from “python .\py_sim\00_homework\02_Point_following.py”