Verse_Tutorial_Drone

October 31, 2022

1 Getting Started

A verse scenario is defined by a map, a set of agents and, if there exist multiple agents, a sensor.

In this section, we are going to look at how to create a simple scenario in Verse. In this scenario we will have a drone following a straight track and will avoid by moving upwards to avoid a static obstacle at known position in the track.

1.1 Instantiate map

The map of a scenario specifies the tracks that the agents can follow. In this example, we will only look at a simple map with two types of tracks: 1) type T0, which describe straight track that align with x-axis and 2) type TAvoidUp, which describes set of tracks with upward direction that the agent can follow while avoiding the obstacle. The types of tracks will also be referred as track modes in later sections. We will discuss in detail how to create maps in later sections. In this section we will import a pre-defined map.

[1]:
```python
from tutorial_map import M3

map1 = M3()
```

1.2 Creating agent

To create such a scenario in Verse, we first need to create an agent for Verse. An agent in Verse is defined by a set of tactical modes, a decision logic to determine the transition between tactical modes, and a flow function that defines continuous evolution. The agent's tactical mode and decision logic are provided as python code strings and the flow function is provided as a python function.

The tactical mode of the agents corresponds to an agent's decision. For example, in this drone avoidance example, the tactical mode for the agent can be Normal and AvoidUp. The decision logic also need to know the available track modes from the map. The tactical modes and track modes are provided as Enums to Verse.

[2]:
```python
from enum import Enum, auto

class CraftMode(Enum):
    Normal = auto()
    AvoidUp = auto()
```

```
class TrackMode(Enum):
    T0 = auto()
    TAvoidUp = auto()
```

We also require the user to provide the continuous and discrete variables of the agents together with the decision logic. The variables are provided inside class with name State. Variables end with _mode will be identify by verse as discrete variables. In the example below, craft_mode and track_mode are the discrete variables and x, y, z, vx, vy, vz are the continuous variables. The type hints for the discrete variables are necessary to associate discrete variables with the tactical modes and lane modes defined above

[3]:
```
class State:
    x: float
    y: float
    z: float
    vx: float
    vy: float
    vz: float
    craft_mode: CraftMode
    track_mode: TrackMode

    def __init__(self, x, y, z, vx, vy, vz, craft_mode, track_mode):
        pass
```

The decision logic describe for an agent takes as input its current state and the (observable) states of the other agents if there's any, and updates the tactical mode of the ego agent. In this example, the decision logic is traight forward: When the x position of the drone is close to the obstacle (20m), the drone will start moving upward. There's no other agents in this scenario. The decision logic of the agent can be written in an expressive subset of Python inside function decisionLogic.

[4]:
```
import copy
def decisionLogic(ego: State, track_map):
    next = copy.deepcopy(ego)
    if ego.craft_mode == CraftMode.Normal:
        if ego.x > 20 :
            next.craft_mode = CraftMode.AvoidUp
            next.track_mode = track_map.h(ego.track_mode, ego.craft_mode,␣
 ↪CraftMode.AvoidUp)
    return next
```

We incoperate the above definition of tactical modes and decision logic into code strings and combine it with an imported agent flow, we can then obtain the agent for this sceanrio.

[5]:
```
from tutorial_agent import DroneAgent
drone1 = DroneAgent('drone1', file_name="dl_sec1.py", t_v_pair=(1, 1),␣
 ↪box_side=[0.4]*3)
```

1.3 Creating scenario

With the agent and map defined, we can now define the scenario.

```
[6]: from verse.scenario import Scenario
     scenario = Scenario()
```

We can set the initial condition of the agent and add the agent

```
[7]: drone1.set_initial([[0, -0.5, -0.5, 0, 0, 0],[1, 0.5, 0.5, 0, 0, 0]],␣
      ↪(CraftMode.Normal, TrackMode.T0))
     scenario.add_agent(drone1)
```

and set the map for the sceanrio

```
[8]: scenario.set_map(map1)
```

Since we only have one agent in the scenario, we don't need to specify a sensor.

We can then compute simulation traces or reachable states for the scenario

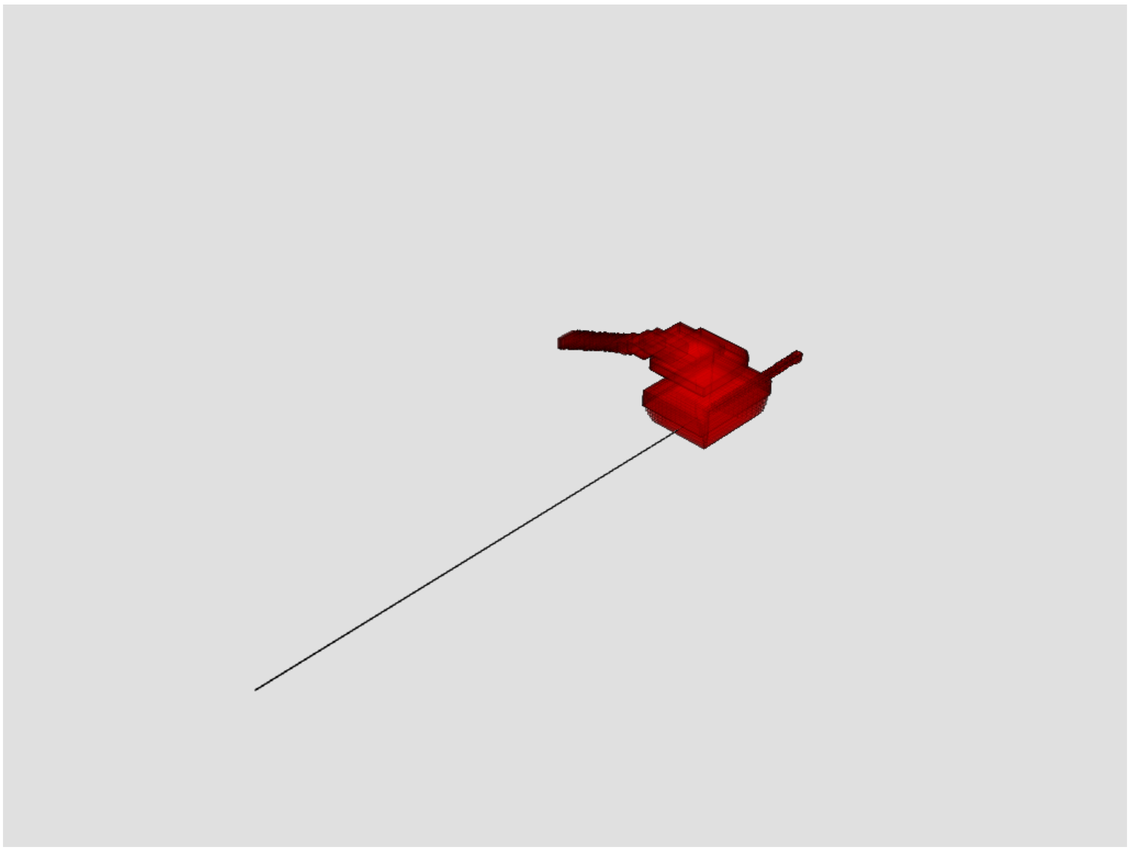
```
[9]: traces_simu = scenario.simulate(60, 0.2)
     traces_veri = scenario.verify(60, 0.2)
```

```
[[0.5825701068018891, 0.4141438327960614, 0.20845945426032098, 0.0, 0.0, 0.0]]
{'drone1': ['Normal', 'T0']}
{'drone1': ('AvoidUp', 'TAvoidUp')}
```

We can visualize the results using functions provided with Verse

```
[10]: from verse.plotter.plotter3D import *
      import pyvista as pv
      import warnings
      warnings.filterwarnings("ignore")

      pv.set_jupyter_backend(None)
      fig = pv.Plotter()
      fig = plot3dMap(map1, ax=fig)
      fig = plot3dReachtube(traces_veri, 'drone1', 1, 2, 3, color = 'r', ax=fig)
      fig.set_background('#e0e0e0')
      fig.show()
```



2 Adding multiple agents

In the previous section, we played around with a scenario with a single drone. In this example, we will create a scenario with multiple agents and explore the potential of Verse to handle multi-agent scenarios.

In this example, we will look at a two drone collision avoidance example. In this example, we have two drones running in a map with three parallel tracks arranged in vertical directions. The two drones are running in the same direction on the middle track. However, the later drone is moving faster than the drone in the front and it will perform a track switch while getting close to the front drone to avoid collision.

We will first setup a new scenario and add the map.

[11]:
```python
from verse.scenario import Scenario
from tutorial_map import M4
scenario = Scenario()
scenario.set_map(M4())
```

[12]:
```python
from enum import Enum, auto
```

```python
class CraftMode(Enum):
    Normal = auto()
    MoveUp = auto()
    MoveDown = auto()

class TrackMode(Enum):
    T0 = auto()
    T1 = auto()
    T2 = auto()
    M01 = auto()
    M10 = auto()
    M12 = auto()
    M21 = auto()

class State:
    x: float
    y: float
    z: float
    vx: float
    vy: float
    vz: float
    craft_mode: CraftMode
    track_mode: TrackMode

    def __init__(self, x, y, z, vx, vy, vz, craft_mode, track_mode):
        pass
```

Verse's decision logic support python any and all functions, which allows the decision logic to quantify over other agents in the scenario. This enable user to easily create multi-agent scenario. In addition, the support of user defined functions, such as the is_close function in example, enable user to write more complicated decision logic. In this case, the decision logic will take an additional argument others, which provides the states for other agents. Notice the type hint for both ego and others. The updated agent's decision logic looks like following.

[13]:
```python
from typing import List
import copy

def is_close(ego, other):
    res = ((other.x - ego.x < 10 and other.x-ego.x > 8) or\
        (other.y-ego.y < 10 and other.y-ego.y > 8) or\
        (other.z-ego.z < 10 and other.z-ego.z > 8))
    return res

def decisionLogic(ego: State, others: List[State], track_map):
    next = copy.deepcopy(ego)

    if ego.craft_mode == CraftMode.Normal:
        if any(ego.x<other.x and (is_close(ego, other) and ego.track_mode ==␣
↪other.track_mode) for other in others):
            if track_map.h_exist(ego.track_mode, ego.craft_mode, CraftMode.
↪MoveUp):
                next.craft_mode = CraftMode.MoveUp
                next.track_mode = track_map.h(
                    ego.track_mode, ego.craft_mode, CraftMode.MoveUp)
            if track_map.h_exist(ego.track_mode, ego.craft_mode, CraftMode.
↪MoveDown):
                next.craft_mode = CraftMode.MoveDown
                next.track_mode = track_map.h(
                    ego.track_mode, ego.craft_mode, CraftMode.MoveDown)

    if ego.craft_mode == CraftMode.MoveUp:
        if track_map.altitude(ego.track_mode)-ego.z > -1 and track_map.
↪altitude(ego.track_mode)-ego.z < 1:
            next.craft_mode = CraftMode.Normal
            if track_map.h_exist(ego.track_mode, ego.craft_mode, CraftMode.
↪Normal):
                next.track_mode = track_map.h(
                    ego.track_mode, ego.craft_mode, CraftMode.Normal)

    if ego.craft_mode == CraftMode.MoveDown:
        if track_map.altitude(ego.track_mode)-ego.z > -1 and track_map.
↪altitude(ego.track_mode)-ego.z < 1:
            next.craft_mode = CraftMode.Normal
            if track_map.h_exist(ego.track_mode, ego.craft_mode, CraftMode.
↪Normal):
                next.track_mode = track_map.h(
                    ego.track_mode, ego.craft_mode, CraftMode.Normal)

    return next
```

With the updated decision logic, we can now spawn the two agents with their initial conditions.

[14]:
```
from tutorial_agent import DroneAgent
drone1 = DroneAgent(
    'drone1', file_name="dl_sec2.py", t_v_pair=(1, 1), box_side=[0.4]*3)
drone1.set_initial(
    [[1.5, -0.5, -0.5, 0, 0, 0], [2.5, 0.5, 0.5, 0, 0, 0]],
    (CraftMode.Normal, TrackMode.T1)
)

drone2 = DroneAgent(
    'drone2', file_name="dl_sec2.py", t_v_pair=(1, 0.5), box_side=[0.4]*3)
drone2.set_initial(
    [[19.5, -0.5, -0.5, 0, 0, 0], [20.5, 0.5, 0.5, 0, 0, 0]],
    (CraftMode.Normal, TrackMode.T1)
)
```

We can then add both agents to the scenario

[15]:
```
scenario.add_agent(drone1)
scenario.add_agent(drone2)
```

With multiple agents in the scenario, we now need to add the sensor. The sensor defines how an agent is visible to other agents. In this example, we will use the default sensor function, which allows all agents to see all variables of other agents.

[16]:
```
from tutorial_sensor import DefaultSensor
scenario.set_sensor(DefaultSensor())
```

With the scenario fully constructed, we can now simulte or verify the scenario using the simulate/verify function provided by Verse.

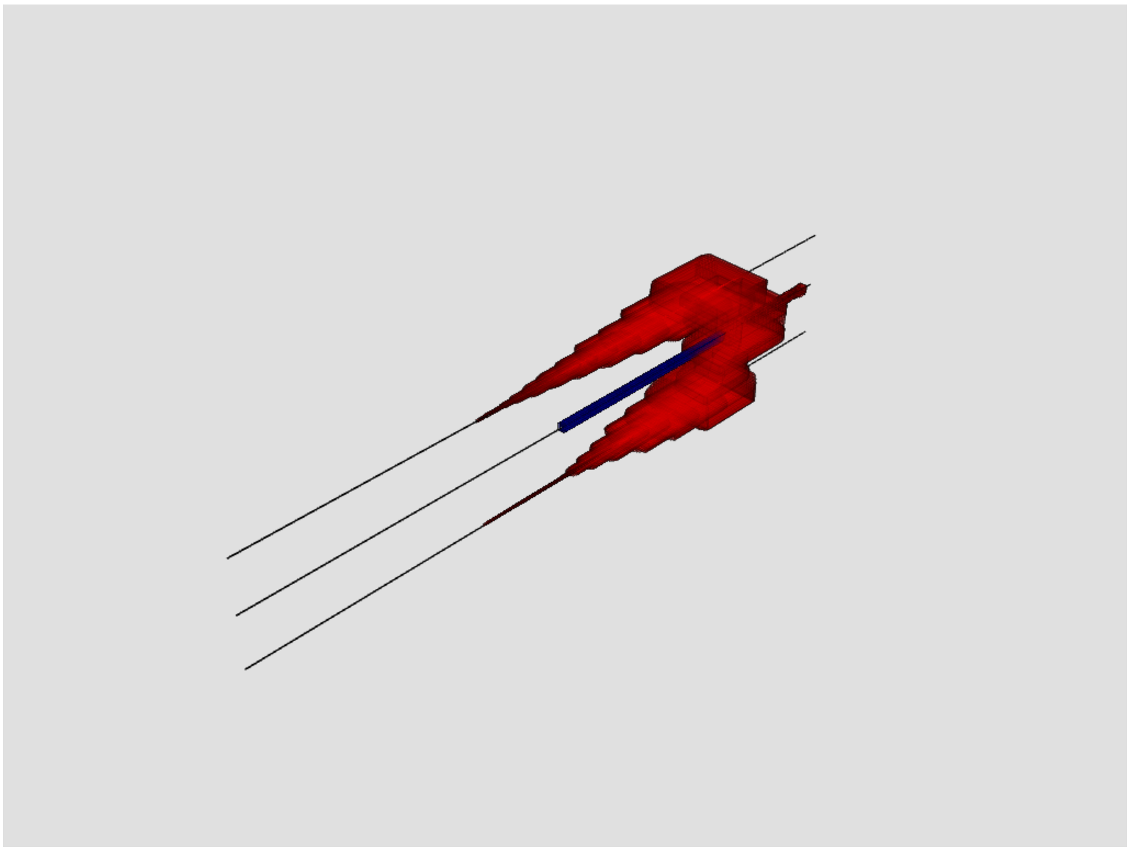
[17]:
```
traces_simu = scenario.simulate(60, 0.2)
traces_veri = scenario.verify(60, 0.2)
```

```
[[2.1280782128465505, 0.3059746320738689, 0.12784024912786485, 0.0, 0.0, 0.0],
[19.823844448840493, 0.21348489187574105, -0.3201274641900793, 0.0, 0.0, 0.0]]
{'drone1': ['Normal', 'T1'], 'drone2': ['Normal', 'T1']}
{'drone1': ('MoveDown', 'M12'), 'drone2': ['Normal', 'T1']}
{'drone1': ('MoveUp', 'M10'), 'drone2': ['Normal', 'T1']}
{'drone1': ('Normal', 'T2'), 'drone2': ['Normal', 'T1']}
{'drone1': ('Normal', 'T0'), 'drone2': ['Normal', 'T1']}
```

We can visualize the results using functions provided with Verse

[18]:
```
from verse.plotter.plotter3D import *
import pyvista as pv
import warnings
warnings.filterwarnings("ignore")

pv.set_jupyter_backend(None)
fig = pv.Plotter()
fig = plot3dMap(M4(), ax=fig)
fig = plot3dReachtube(traces_veri, 'drone1', 1, 2, 3, color = 'r', ax=fig)
fig = plot3dReachtube(traces_veri, 'drone2', 1, 2, 3, color = 'b', ax=fig)
fig.set_background('#e0e0e0')
fig.show()
```



3 Adding safety assertions

[19]:
```
from enum import Enum, auto

class CraftMode(Enum):
    Normal = auto()
    MoveUp = auto()
    MoveDown = auto()

class TrackMode(Enum):
    T0 = auto()
    T1 = auto()
    T2 = auto()
    M01 = auto()
    M10 = auto()
    M12 = auto()
    M21 = auto()

class State:
    x: float
    y: float
    z: float
    vx: float
    vy: float
    vz: float
    craft_mode: CraftMode
    track_mode: TrackMode

    def __init__(self, x, y, z, vx, vy, vz, craft_mode, track_mode):
        pass
```

Verse allow checking safety conditions while performing simulation and verification. The safety conditions in verse can be specified using Python assert statements in the decision logic. With the any and all functions, the user can also define safety conditions between different agents. In this example, we are going to add two safety conditions: 1) the distance between two agents in all x,y,z direction should always be greater than or equal to 1.0m and 2) both drones should never enter region $40 \leq x \leq 50$, $-5 \leq y \leq 5$, $-10 \leq z \leq -6$. The two safety asserts in the decisionLogic is shown below.

[20]:
```
from typing import List
import copy

def decisionLogic(ego: State, others: List[State], track_map):
    next = copy.deepcopy(ego)
    '''
    '''

    assert not any(ego.x-other.x < 1 and ego.x-other.x >-1 and \
        ego.y-other.y < 1 and ego.y-other.y > -1 and \
        ego.z-other.z < 1 and ego.z-other.z > -1 \
        for other in others),\
        "Safe Seperation"

    assert not (ego.x > 40 and ego.x<50 and\
        ego.y>-5 and ego.y<5 and\
        ego.z > -10 and ego.z<-6),\
        "Unsafe Region"

    return next
```

We can then construct the two agents and the scenario exactly as previous section.

[21]:
```
from verse.scenario import Scenario
from tutorial_map import M4
scenario = Scenario()
scenario.set_map(M4())

from tutorial_agent import DroneAgent
```

```
drone1 = DroneAgent(
    'drone1', file_name="dl_sec3.py", t_v_pair=(1, 1), box_side=[0.4]*3)
drone1.set_initial(
    [[1.5, -0.5, -0.5, 0, 0, 0], [2.5, 0.5, 0.5, 0, 0, 0]],
    (CraftMode.Normal, TrackMode.T1)
)
scenario.add_agent(drone1)

drone2 = DroneAgent(
    'drone2', file_name="dl_sec3.py", t_v_pair=(1, 0.5), box_side=[0.4]*3)
drone2.set_initial(
    [[19.5, -0.5, -0.5, 0, 0, 0], [20.5, 0.5, 0.5, 0, 0, 0]],
    (CraftMode.Normal, TrackMode.T1)
)
scenario.add_agent(drone2)

from tutorial_sensor import DefaultSensor
scenario.set_sensor(DefaultSensor())
```

We can then simulate and verify the scenario and visualize the result. We can see from the result that drone1 enters the unsafe region, which causes a safety condition violation.

[22]:
```
traces_simu = scenario.simulate(60, 0.2)
traces_veri = scenario.verify(60, 0.2)
```

```
[[1.7696354681685915, -0.4434928290377099, 0.25585978351279703, 0.0, 0.0, 0.0],
[19.526462854520915, 0.20568735262816873, 0.3399636492062529, 0.0, 0.0, 0.0]]
assert hit for drone1: "Unsafe Region" @ {'ego': State(x=40.01887935826787,
y=-0.030093720299303017, z=-7.944140319877006, vx=0.9842831331582708,
vy=-0.19685662663165182, vz=0.3999999999999826, craft_mode='Normal',
track_mode='T2'), 'others': [State(x=40.13735096608431, y=-0.010854984316650551,
z=0.05996425724163723, vx=0.19685662663165315, vy=-0.19685662663165315,
vz=-0.40000000000001057, craft_mode='Normal', track_mode='T1')], 'track_map':
<tutorial_map.M4 object at 0x7fee6b224cd0>}
{'drone1': ['Normal', 'T1'], 'drone2': ['Normal', 'T1']}
{'drone1': ('MoveDown', 'M12'), 'drone2': ['Normal', 'T1']}
{'drone1': ('MoveUp', 'M10'), 'drone2': ['Normal', 'T1']}
{'drone1': ('Normal', 'T2'), 'drone2': ['Normal', 'T1']}
assert hit for drone1: "Unsafe Region"
69
{'drone1': ('Normal', 'T0'), 'drone2': ['Normal', 'T1']}
```

[23]:
```
from verse.plotter.plotter3D import *
import pyvista as pv
import warnings
warnings.filterwarnings("ignore")
```

```python
pv.set_jupyter_backend(None)
fig = pv.Plotter()
fig = plot3dMap(M4(), ax=fig)
fig = plot3dReachtube(traces_veri, 'drone1', 1, 2, 3, color = 'r', ax=fig)
fig = plot3dReachtube(traces_veri, 'drone2', 1, 2, 3, color = 'b', ax=fig)
fig.set_background('#e0e0e0')
fig.show()
```