**Task-4**

**Agent Pendulum**

**Reinforcement Learning:**

It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience

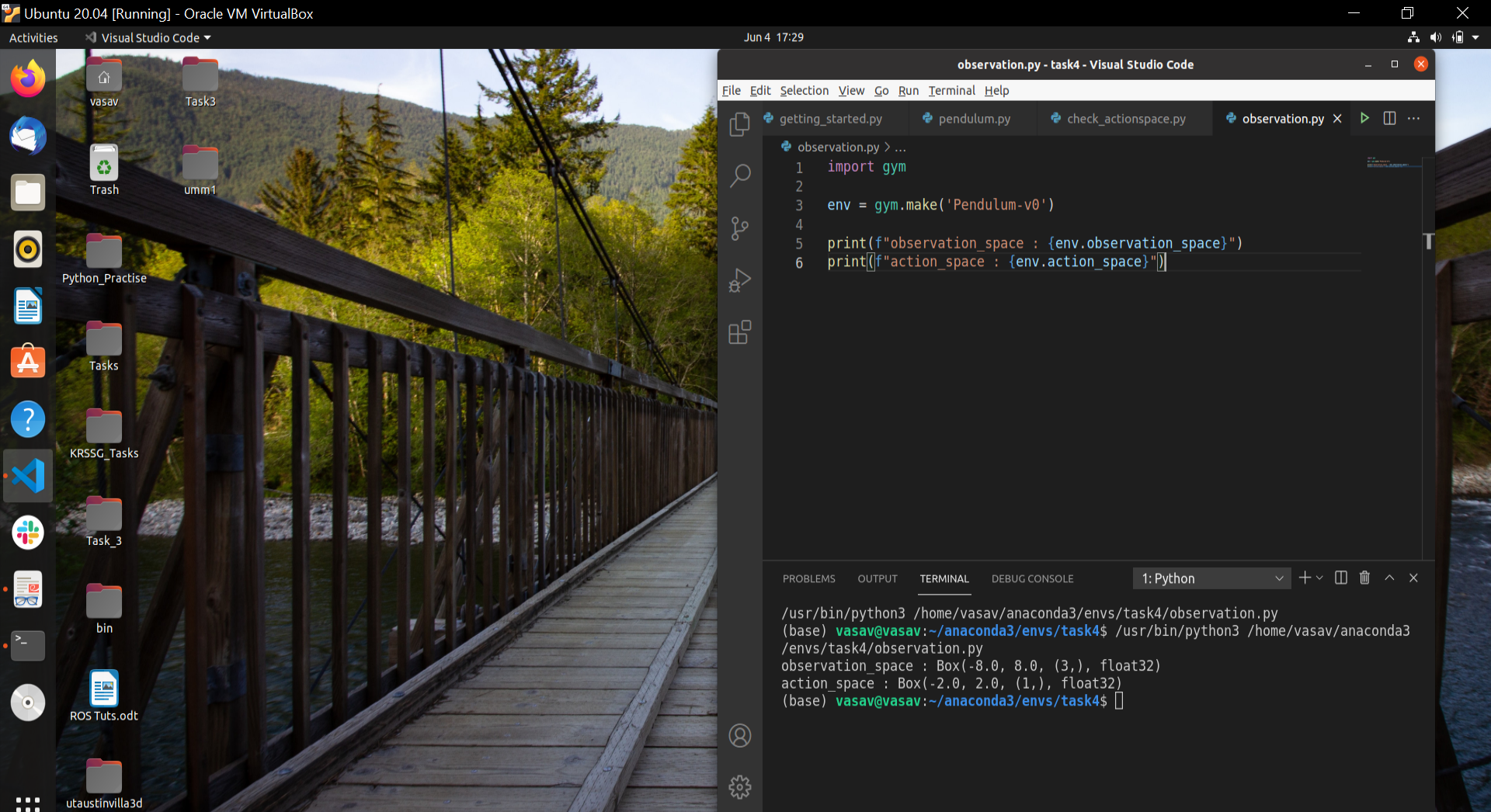
**Pendulum-v0**

Inverted Pendulum swing-up problem

In this version of the problem, the frictionless pendulum starts in a random position, and the goal is to swing it up so that it stays upright.

**1->You should inspect the observations dimensions and action dimensions by printing env.observation\_space and env.action\_space.**

Every environment comes with an action\_space and an observation\_space. These attributes are of type [Space](https://github.com/openai/gym/blob/master/gym/core.py), and they describe the format of valid actions and observations:



Here we get our output in the Box space

The [Box](https://github.com/openai/gym/blob/master/gym/spaces/box.py) space represents an n-dimensional box, where each coordinate lies

Between bound defined by [low, high], which are the first two elements of the Box, if we talk about the observation\_space “low = -8” and “high = 8”

And the 3rd element is a tuple containing a single element, which tells about the number of parameters in this space and the last, i.e, the 4th element is the datatype.

**2->Look at the environment definition and understand what the observations mean: gym/gym/envs/classic\_control/pendulum.py ○ in particular, look at the step function, line 32, and where the observations come from (get\_obs function, line 57)**

The step function in general returns four values: observation, reward, done, info. So here also our step function returns \_\_getobs(), -costs, False and {}

* Where \_\_getobs() is the observation and contains the value of th and thdot, here th is the angle of a pendulum with the vertical and thdot is the angular velocity of the pendulum.
* -costs will give us the value of reward
* False is value for the done(tells whether it’s time to reset the environment again.)
* {} is the empty dictionary for the info

So basically our step function is updating the value of th and thdot, calculating the dost, calculating the new value of th and thdot.

And \_\_getobs() returns a numpy array which has three values first two are the sin(th) and co(th) and the third one is the thdot

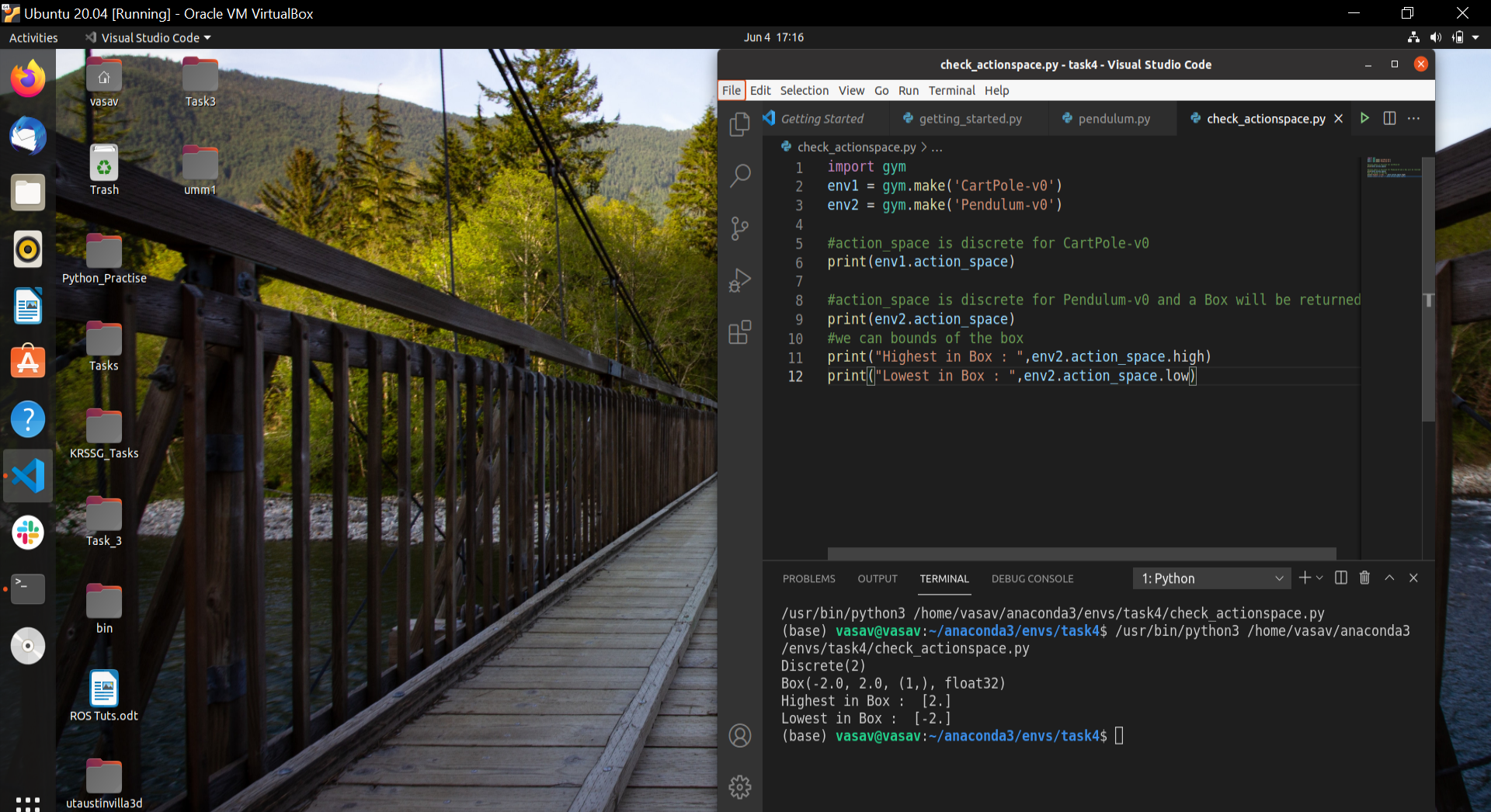
**3->What do the different components in the observations stand for? What’s self.state[1]/thetadot in the environment file, what does it mean?**

Observation is an environment-specific object which represents the observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game.

Different components of the observation will tell us about the state of the environment

The self.state[1] or thetadot is the angular velocity of the pendulum

**4->Is this discrete action (“turn left/right”) or continuous action (“turn left/right at [0-1]x speed”)?**

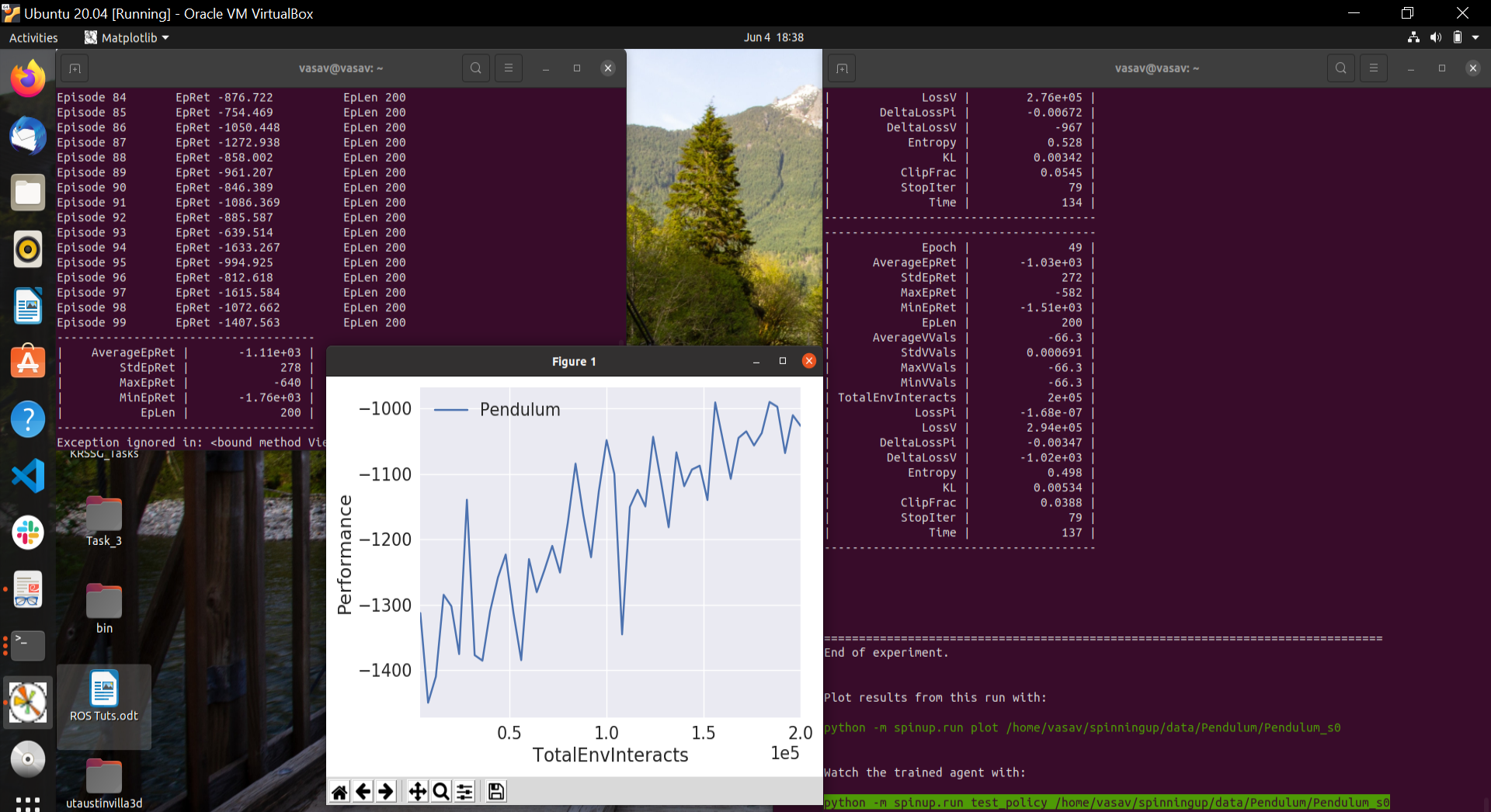


For our case, the action is continuous as printing action space for our environment returns a Box which represents an n-dimensional box, where each coordinate is bound between [low, high]

But in the case of CarPole-v0 has a discrete action space

Discrete space means space consists of n distinct points each mapped to an integer in the value [0, n-1]

**● At this point, look at the trained PPO policy**

****

**5->The reward calculation (“costs”, line 42), has different components. What do they stand for and how are they weighted?**

Reward:

Function of theta, thetadot (omega) and action (torque)

th is theta

thdot is the angular velocity

u is torque

Reward = -[(theta^2) + 0.1\*(theta\_dot^2) + 0.001\*(action^2)]

Theta is normalized between -pi and pi. Therefore, the lowest reward is -(pi^2 + 0.1\*8^2 + 0.001\*2^2) = -16.2736044, and the highest reward is 0. In essence, the goal is to remain at zero angle (vertical), with the least rotational velocity, and the least effort.