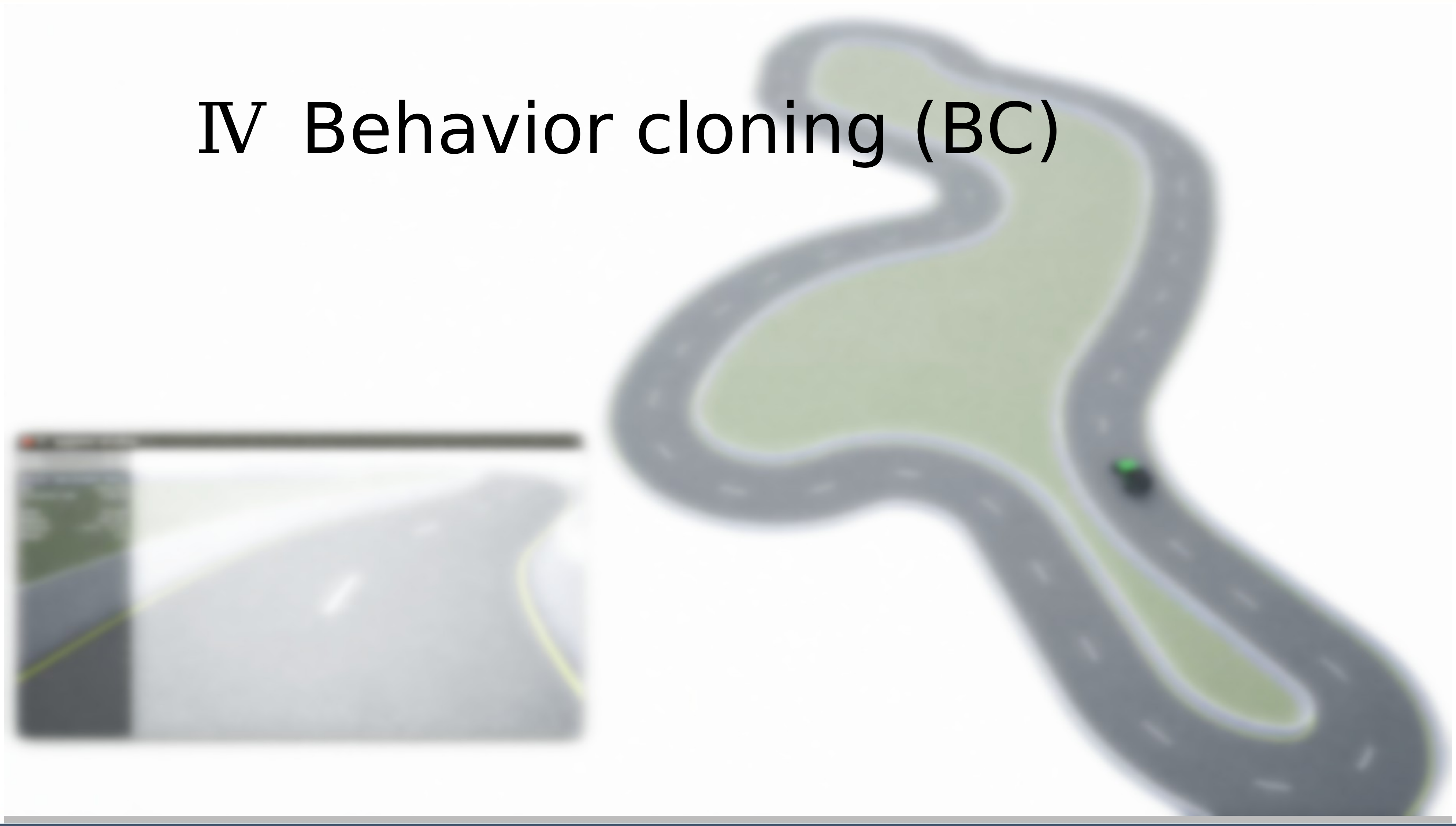
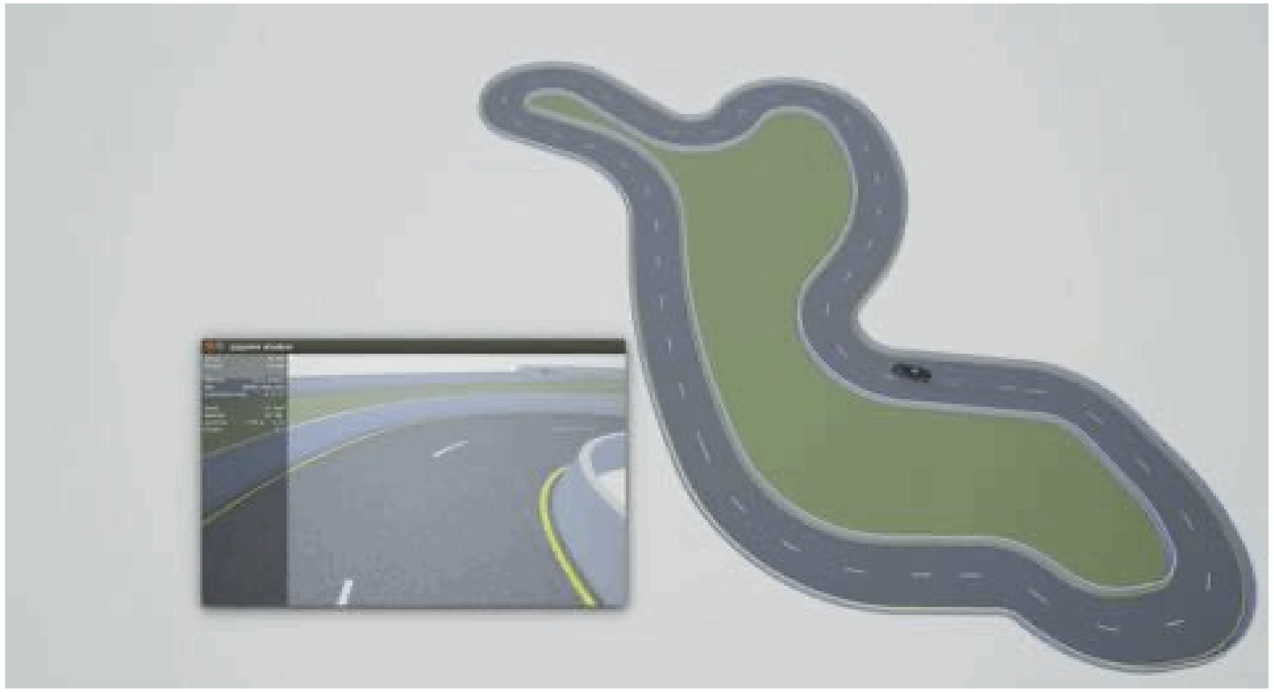
MoCAD: Carla-python Experimental Course



Ⅳ Behavior cloning (BC)

Outline



1. Imitation learning—Behavior Cloning
2. Collecting data: Autopilot or manual control
3. Training Model: Supervise learning
4. Applying the model to control vehicle

1. Imitation learning—Behavior Cloning

- **Imitation learning and why ?**

  An expert (typically a human) provides us with a set of demonstrations.
  The agent then tries to learn the optimal policy by following, **imitating the expert's decisions.**

Reinforcement learning (RL) reward **sparse or hard to design**

- **IL interacts with the environment**

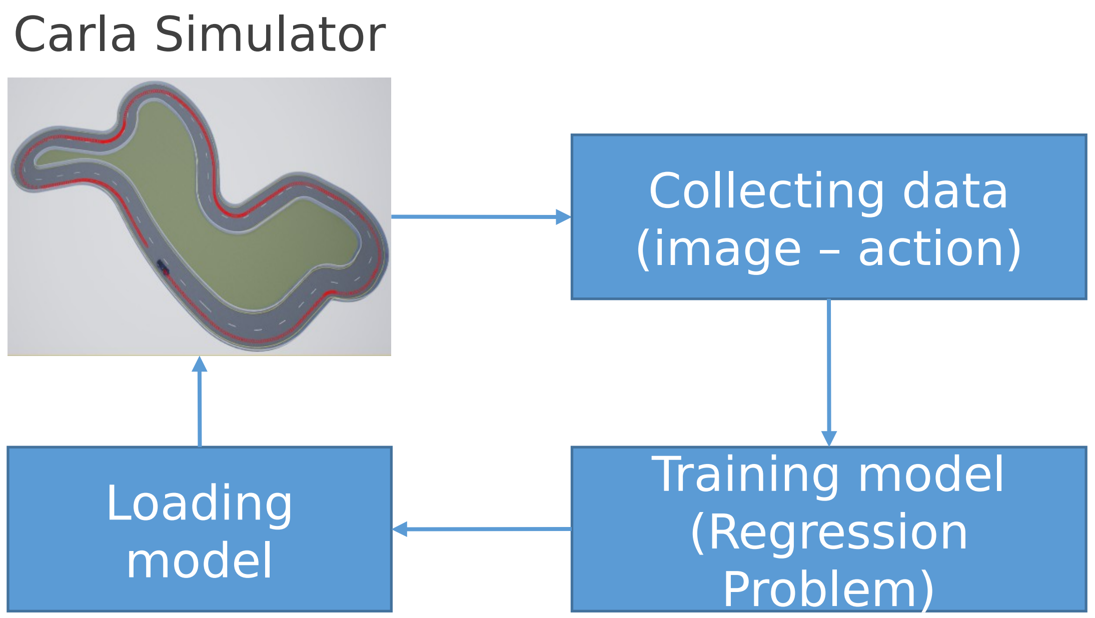
  Markov Decision Process (MDP)

  ① an A set of actions
  ② an S set of states
  ③ a P(s'|s,a) transition
  ④ an unknown R(s,a) reward function

The expert's demonstrations τ = (s0, a0, s1, a1, ...), actions are based on the expert's ("optimal") **π* policy**

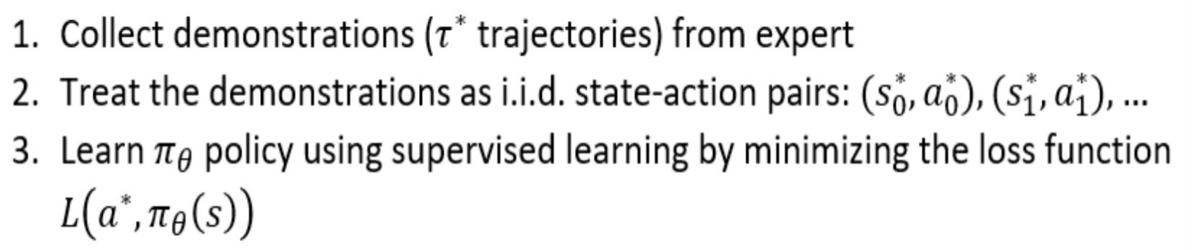
1. Imitation learning—Behavior Cloning

- **Imitation learning classification**
  - ① Behavior Cloning
  - ② Direct Policy Learning
  - ③ Inverse Reinforcement Learning


Carla Simulator
Collecting data (image – action)
Training model (Regression Problem)
Loading model

End to end learning

- **Behavior Cloning**
  learning the expert's policy using supervised learning.

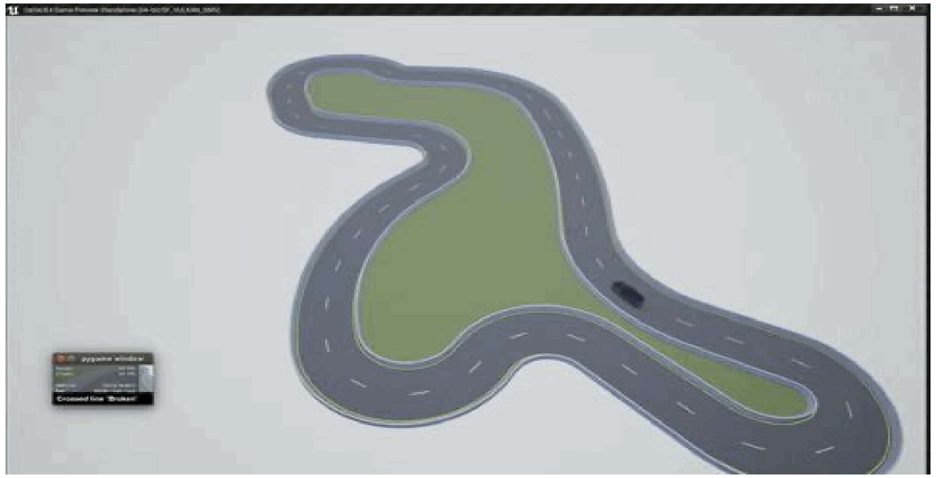


1. Collect demonstrations (τ^* trajectories) from expert
2. Treat the demonstrations as i.i.d. state-action pairs: $(s_0^*, a_0^*), (s_1^*, a_1^*), \ldots$
3. Learn π_θ policy using supervised learning by minimizing the loss function $L(a^*, \pi_\theta(s))$

2. Collecting data: Autopilot or manual control

- **Carla Simulator**

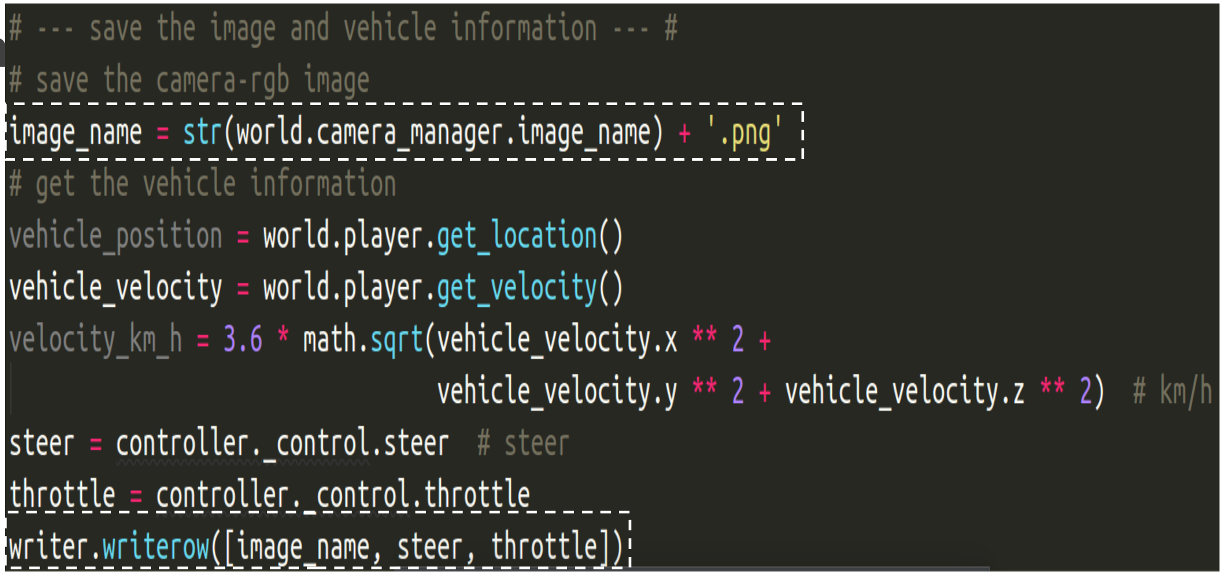
Mode: Synchronous or **Asynchronous**
Sensor: **RGB-image**， Depth-image， Lidar
Vehicle: Autopilot or **manual control**
Action： **Steer**， throttle， brake



- **Moving the vehicle and Collecting Data**

Control method:
Autopilot □ PID
Manual control □ Keyboard

Data:
RGB-image & Vehicle States



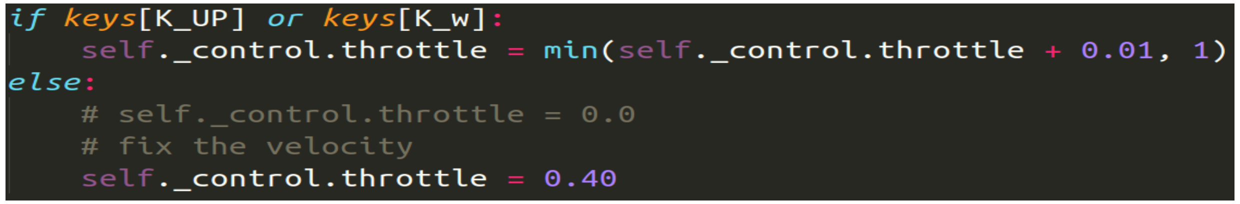
```
# --- save the image and vehicle information --- #
# save the camera-rgb image
image_name = str(world.camera_manager.image_name) + '.png'
# get the vehicle information
vehicle_position = world.player.get_location()
vehicle_velocity = world.player.get_velocity()
velocity_km_h = 3.6 * math.sqrt(vehicle_velocity.x ** 2 +
                                vehicle_velocity.y ** 2 + vehicle_velocity.z ** 2)  # km/h
steer = controller._control.steer  # steer
throttle = controller._control.throttle
writer.writerow([image_name, steer, throttle])
```

2. Collecting data: Autopilot or manual control

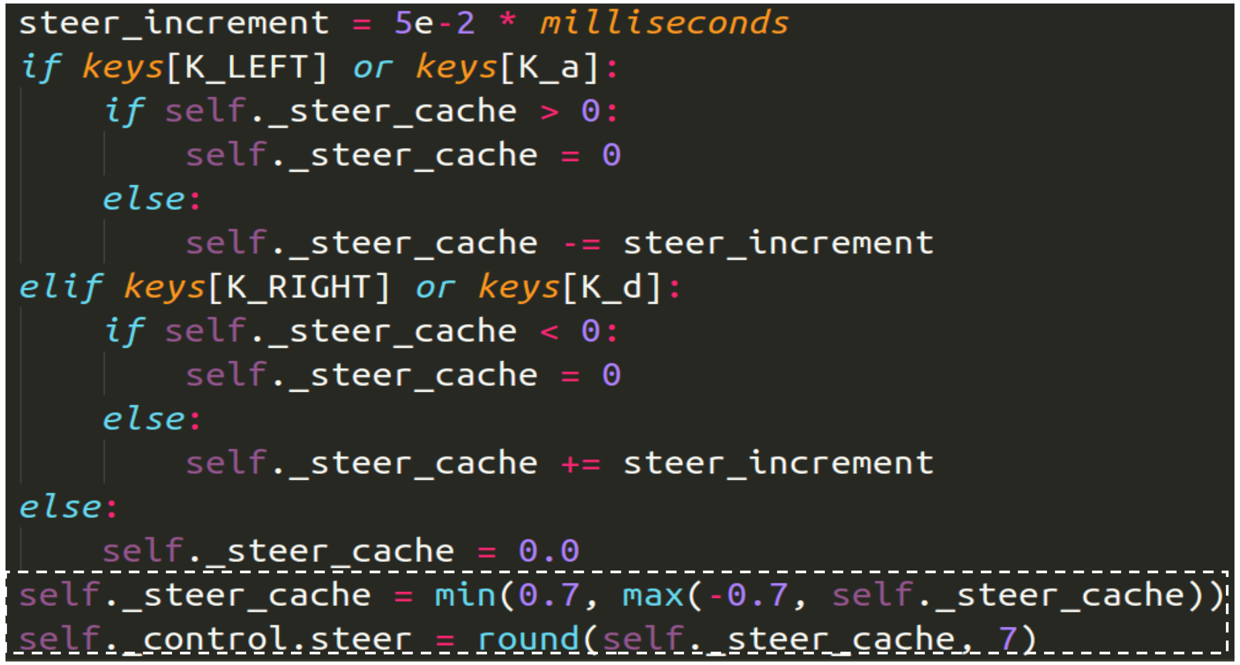
- **Data： image – action pair**

RGB-image： position， rotation， size

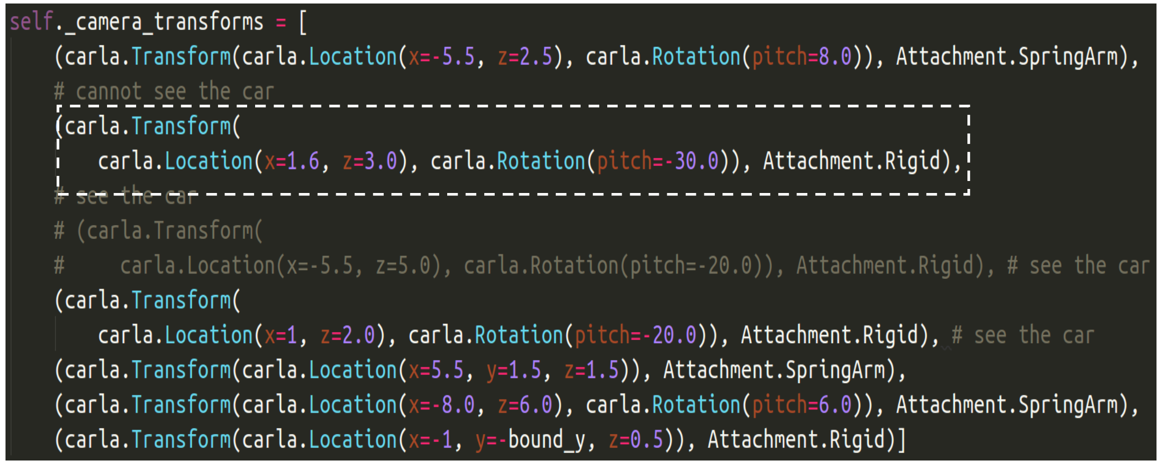
Action： Fixed throttle， steer



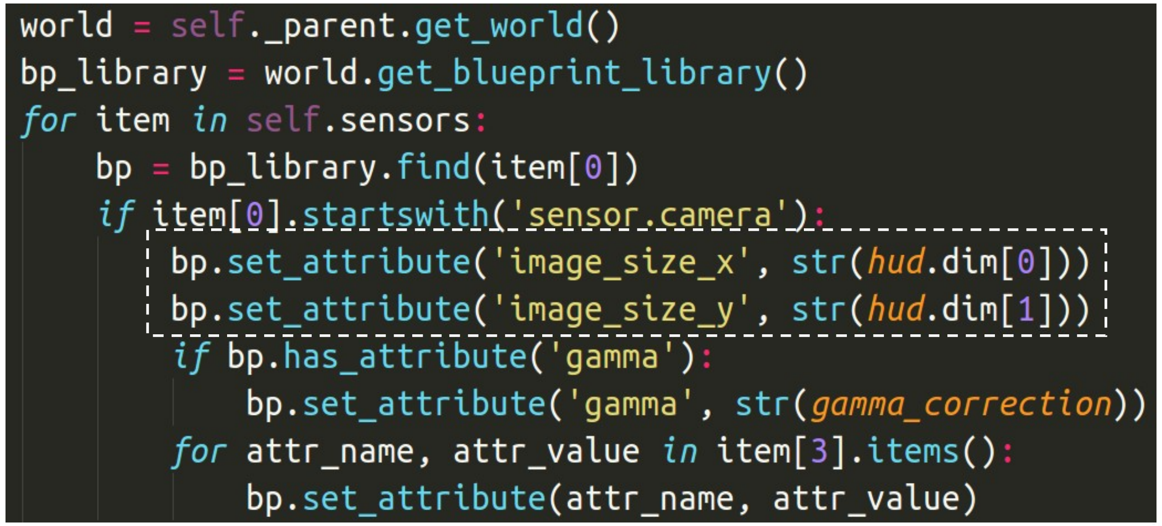
```
if keys[K_UP] or keys[K_w]:
    self._control.throttle = min(self._control.throttle + 0.01, 1)
else:
    # self._control.throttle = 0.0
    # fix the velocity
    self._control.throttle = 0.40
```



```
steer_increment = 5e-2 * milliseconds
if keys[K_LEFT] or keys[K_a]:
    if self._steer_cache > 0:
        self._steer_cache = 0
    else:
        self._steer_cache -= steer_increment
elif keys[K_RIGHT] or keys[K_d]:
    if self._steer_cache < 0:
        self._steer_cache = 0
    else:
        self._steer_cache += steer_increment
else:
    self._steer_cache = 0.0
self._steer_cache = min(0.7, max(-0.7, self._steer_cache))
self._control.steer = round(self._steer_cache, 7)
```



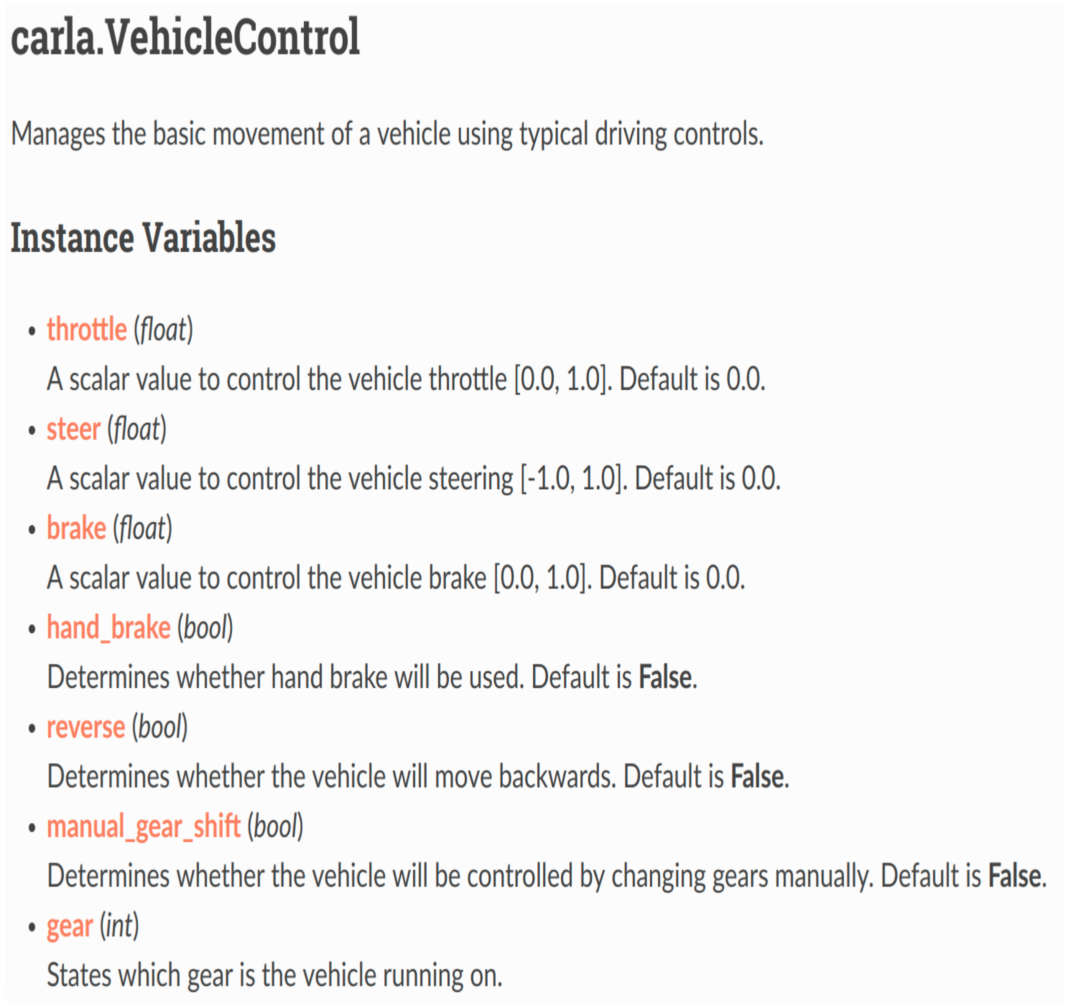
```
self._camera_transforms = [
    (carla.Transform(carla.Location(x=-5.5, z=2.5), carla.Rotation(pitch=8.0)), Attachment.SpringArm),
    # cannot see the car
    (carla.Transform(
        carla.Location(x=1.6, z=3.0), carla.Rotation(pitch=-30.0)), Attachment.Rigid),
    # see the car
    # (carla.Transform(
    #     carla.Location(x=-5.5, z=5.0), carla.Rotation(pitch=-20.0)), Attachment.Rigid), # see the car
    (carla.Transform(
        carla.Location(x=1, z=2.0), carla.Rotation(pitch=-20.0)), Attachment.Rigid), # see the car
    (carla.Transform(carla.Location(x=5.5, y=1.5, z=1.5)), Attachment.SpringArm),
    (carla.Transform(carla.Location(x=-8.0, z=6.0), carla.Rotation(pitch=6.0)), Attachment.SpringArm),
    (carla.Transform(carla.Location(x=-1, y=-bound_y, z=0.5)), Attachment.Rigid)]
```



```
world = self._parent.get_world()
bp_library = world.get_blueprint_library()
for item in self.sensors:
    bp = bp_library.find(item[0])
    if item[0].startswith('sensor.camera'):
        bp.set_attribute('image_size_x', str(hud.dim[0]))
        bp.set_attribute('image_size_y', str(hud.dim[1]))
        if bp.has_attribute('gamma'):
            bp.set_attribute('gamma', str(gamma_correction))
        for attr_name, attr_value in item[3].items():
            bp.set_attribute(attr_name, attr_value)
```

2. Collecting data: Autopilot or manual control

- Carla Class

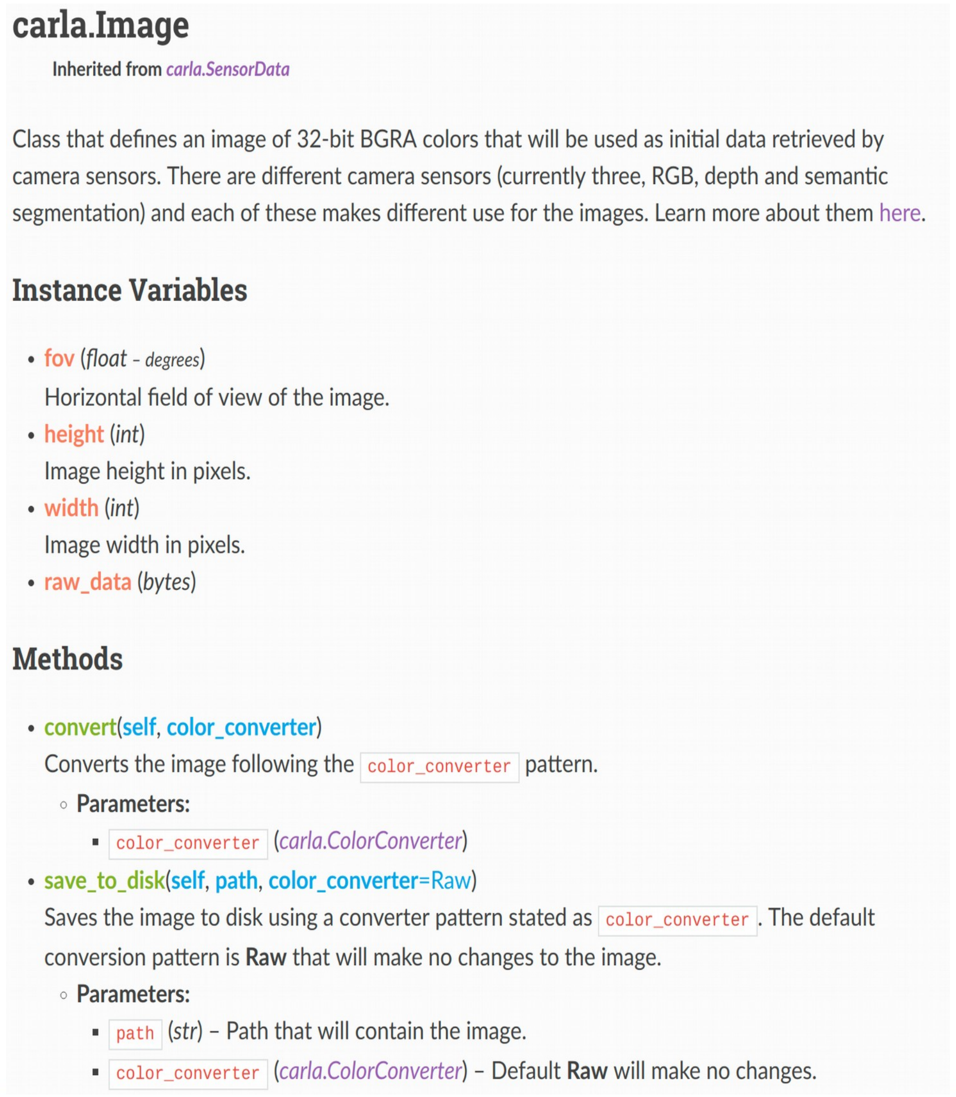


carla.VehicleControl

Manages the basic movement of a vehicle using typical driving controls.

Instance Variables

- throttle (*float*)
  A scalar value to control the vehicle throttle [0.0, 1.0]. Default is 0.0.
- steer (*float*)
  A scalar value to control the vehicle steering [-1.0, 1.0]. Default is 0.0.
- brake (*float*)
  A scalar value to control the vehicle brake [0.0, 1.0]. Default is 0.0.
- hand_brake (*bool*)
  Determines whether hand brake will be used. Default is **False**.
- reverse (*bool*)
  Determines whether the vehicle will move backwards. Default is **False**.
- manual_gear_shift (*bool*)
  Determines whether the vehicle will be controlled by changing gears manually. Default is **False**.
- gear (*int*)
  States which gear is the vehicle running on.



carla.Image

Inherited from *carla.SensorData*

Class that defines an image of 32-bit BGRA colors that will be used as initial data retrieved by camera sensors. There are different camera sensors (currently three, RGB, depth and semantic segmentation) and each of these makes different use for the images. Learn more about them here.

Instance Variables

- fov (*float - degrees*)
  Horizontal field of view of the image.
- height (*int*)
  Image height in pixels.
- width (*int*)
  Image width in pixels.
- raw_data (*bytes*)

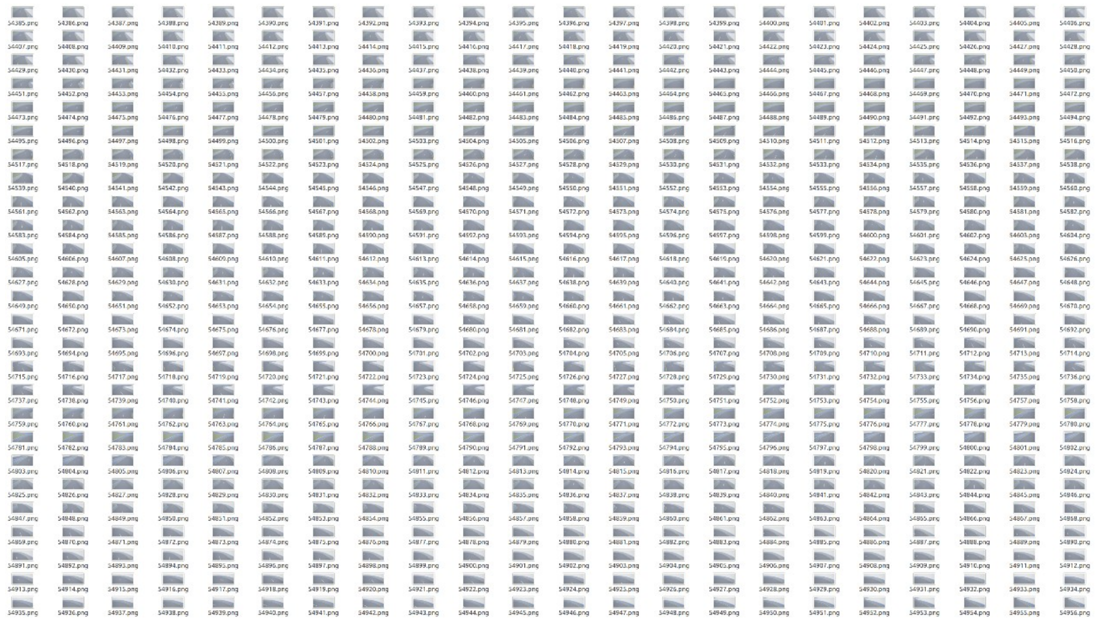
Methods

- convert(self, color_converter)
  Converts the image following the `color_converter` pattern.
  - **Parameters:**
    - `color_converter` (*carla.ColorConverter*)
- save_to_disk(self, path, color_converter=Raw)
  Saves the image to disk using a converter pattern stated as `color_converter`. The default conversion pattern is **Raw** that will make no changes to the image.
  - **Parameters:**
    - `path` (*str*) – Path that will contain the image.
    - `color_converter` (*carla.ColorConverter*) – Default **Raw** will make no changes.

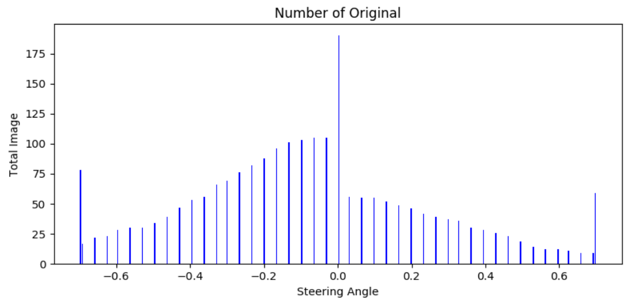
3. Training Model: Supervise learning

- Preprocess data：

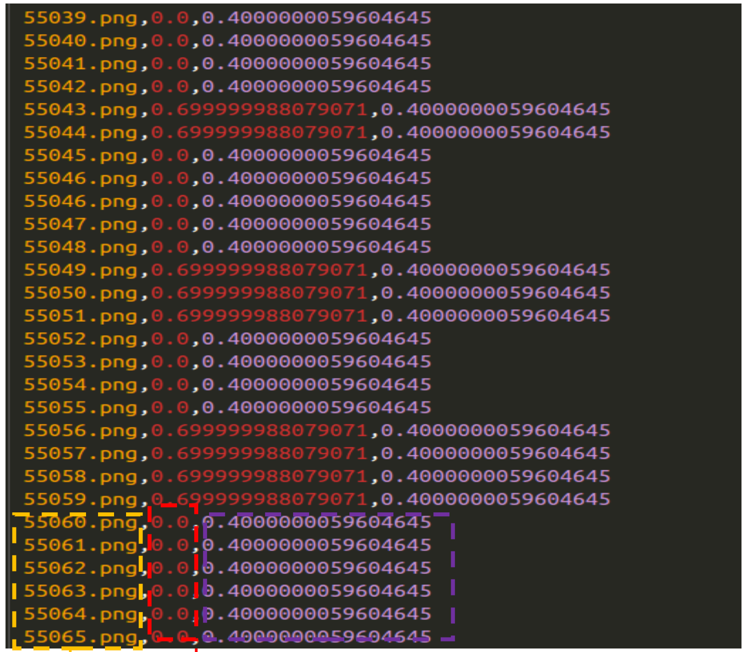
RGB-Image: x_train



① Resize;
② RGB-image to gray-image;
③ Crop;
④ Distribution;



Vehicle States: y_label



```
55039.png,0.0,0.4000000059604645
55040.png,0.0,0.4000000059604645
55041.png,0.0,0.4000000059604645
55042.png,0.0,0.4000000059604645
55043.png,0.699999988079071,0.4000000059604645
55044.png,0.699999988079071,0.4000000059604645
55045.png,0.0,0.4000000059604645
55046.png,0.0,0.4000000059604645
55046.png,0.0,0.4000000059604645
55047.png,0.0,0.4000000059604645
55048.png,0.0,0.4000000059604645
55049.png,0.699999988079071,0.4000000059604645
55050.png,0.699999988079071,0.4000000059604645
55051.png,0.699999988079071,0.4000000059604645
55052.png,0.0,0.4000000059604645
55053.png,0.0,0.4000000059604645
55054.png,0.0,0.4000000059604645
55055.png,0.0,0.4000000059604645
55056.png,0.699999988079071,0.4000000059604645
55057.png,0.699999988079071,0.4000000059604645
55058.png,0.699999988079071,0.4000000059604645
55059.png,0.699999988079071,0.4000000059604645
55060.png,0.0,0.4000000059604645
55061.png,0.0,0.4000000059604645
55062.png,0.0,0.4000000059604645
55063.png,0.0,0.4000000059604645
55064.png,0.0,0.4000000059604645
55065.png,0.0,0.4000000059604645
```

Timestamp

Y_label: steer

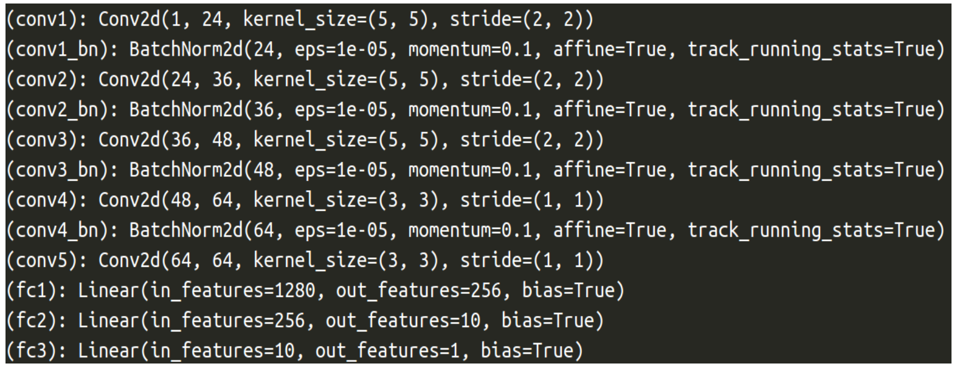
Throttle

3. Training Model: Supervise learning

- Training

X inputs

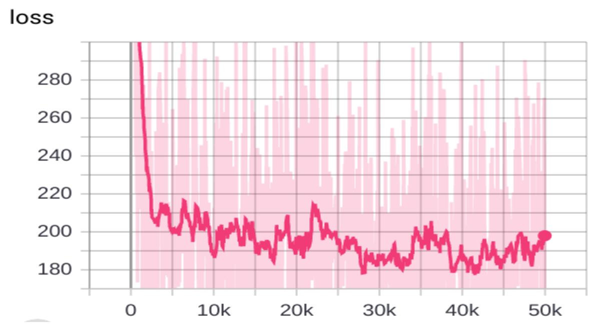
Model: CNN+BatchNormalization+Fully_connect



```
(conv1): Conv2d(1, 24, kernel_size=(5, 5), stride=(2, 2))
(conv1_bn): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(conv2): Conv2d(24, 36, kernel_size=(5, 5), stride=(2, 2))
(conv2_bn): BatchNorm2d(36, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(conv3): Conv2d(36, 48, kernel_size=(5, 5), stride=(2, 2))
(conv3_bn): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(conv4): Conv2d(48, 64, kernel_size=(3, 3), stride=(1, 1))
(conv4_bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(conv5): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
(fc1): Linear(in_features=1280, out_features=256, bias=True)
(fc2): Linear(in_features=256, out_features=10, bias=True)
(fc3): Linear(in_features=10, out_features=1, bias=True)
```

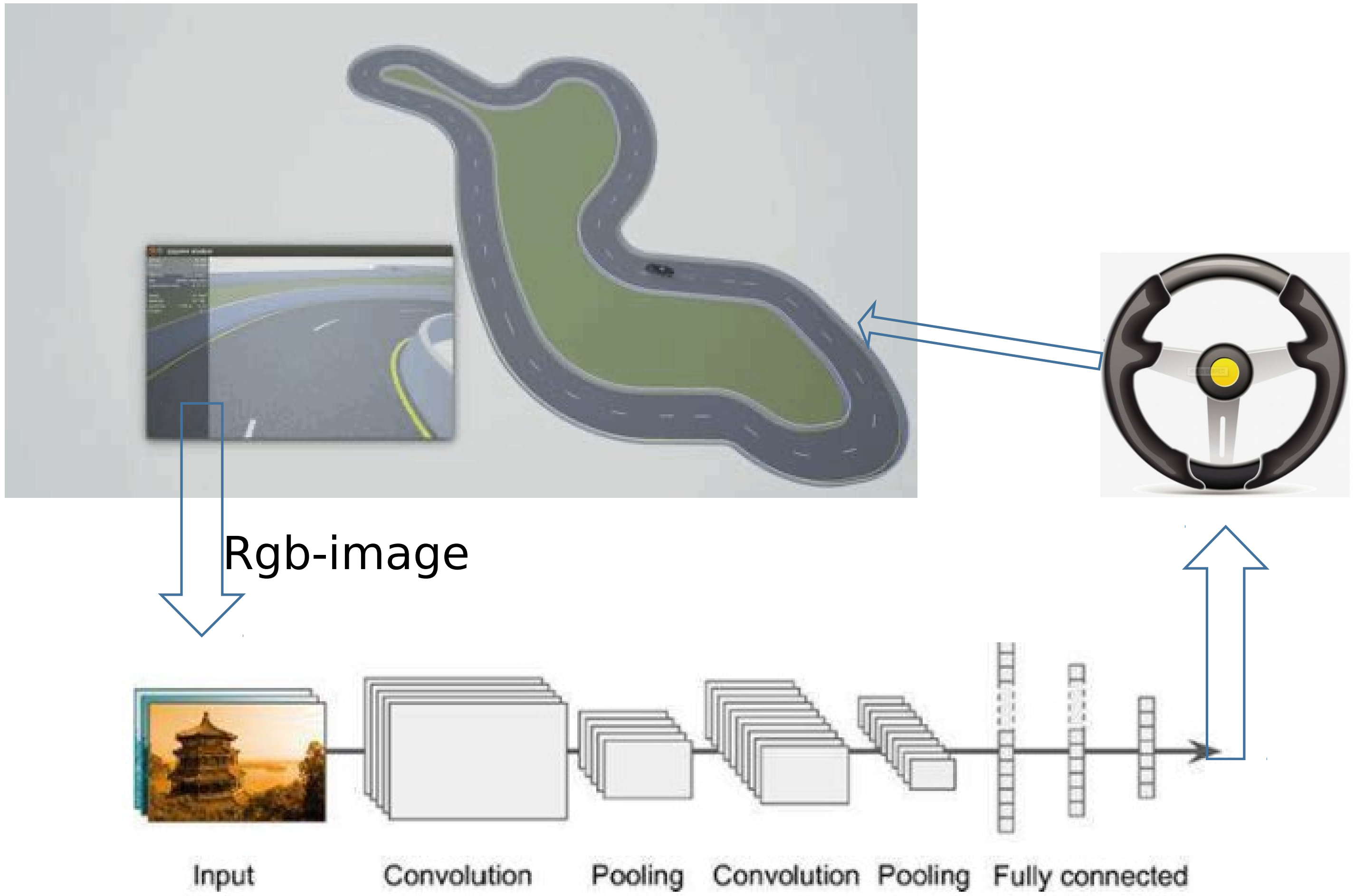
Loss function

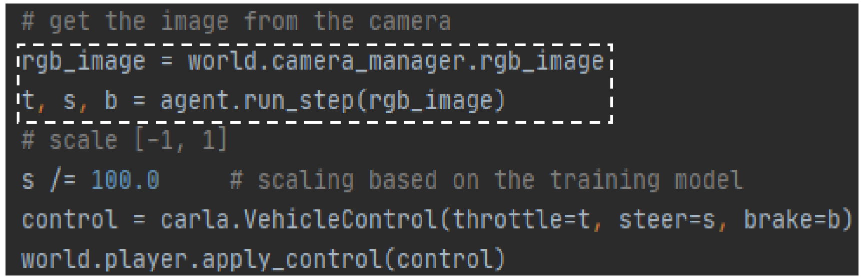
$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - f(x_i))^2$$



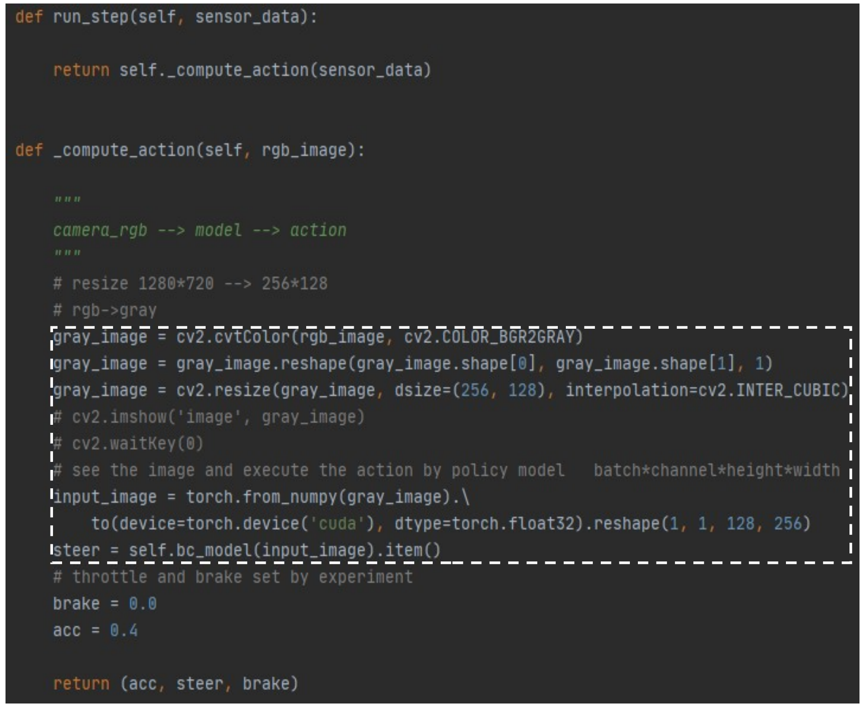
4. Control the vehicle

- Demo



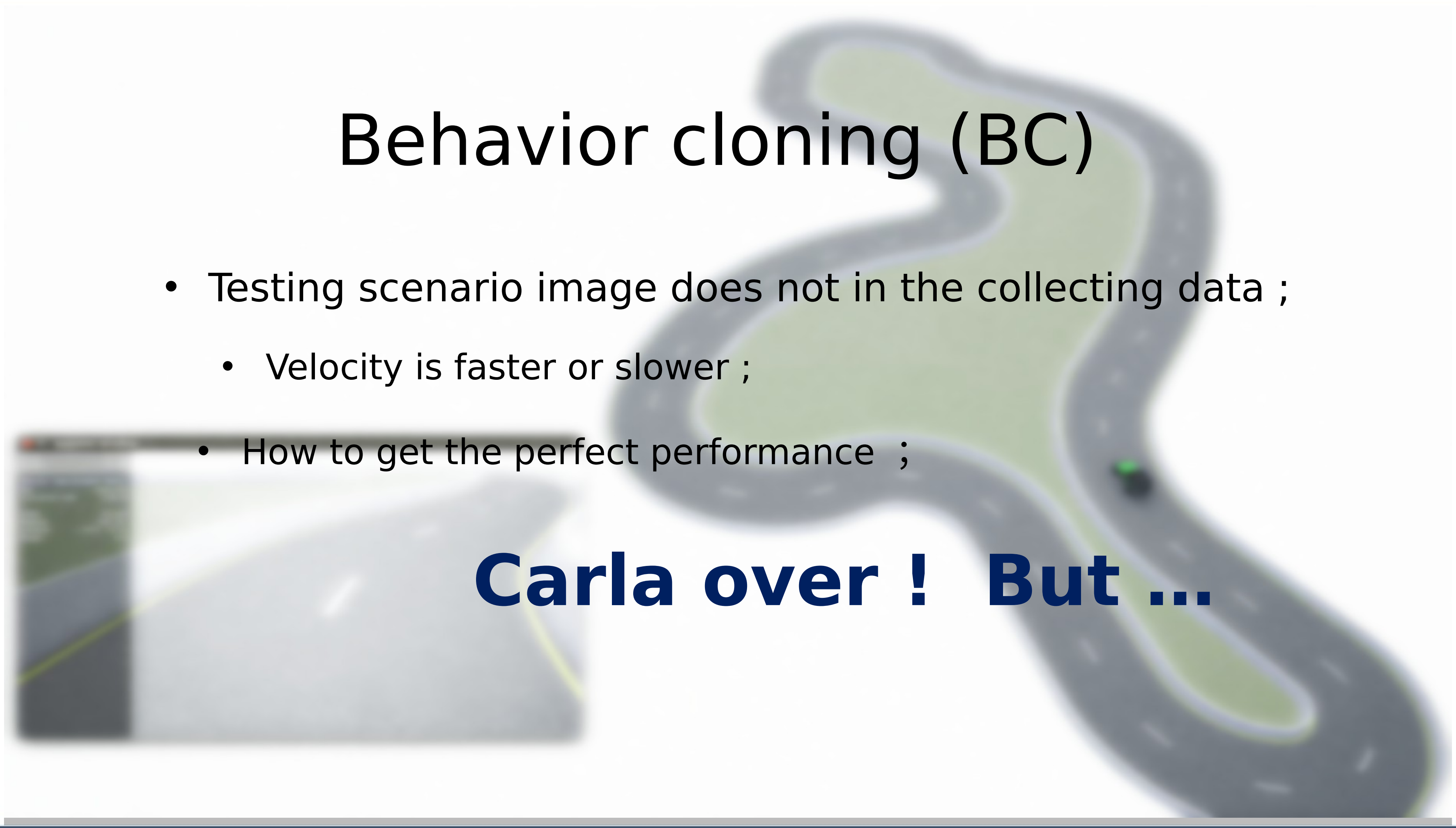


```python
# get the image from the camera
rgb_image = world.camera_manager.rgb_image
t, s, b = agent.run_step(rgb_image)
# scale [-1, 1]
s /= 100.0      # scaling based on the training model
control = carla.VehicleControl(throttle=t, steer=s, brake=b)
world.player.apply_control(control)
```



```python
def run_step(self, sensor_data):

    return self._compute_action(sensor_data)


def _compute_action(self, rgb_image):

    """
    camera_rgb --> model --> action
    """
    # resize 1280*720 --> 256*128
    # rgb->gray
    gray_image = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2GRAY)
    gray_image = gray_image.reshape(gray_image.shape[0], gray_image.shape[1], 1)
    gray_image = cv2.resize(gray_image, dsize=(256, 128), interpolation=cv2.INTER_CUBIC)
    # cv2.imshow('image', gray_image)
    # cv2.waitKey(0)
    # see the image and execute the action by policy model   batch*channel*height*width
    input_image = torch.from_numpy(gray_image).\
        to(device=torch.device('cuda'), dtype=torch.float32).reshape(1, 1, 128, 256)
    steer = self.bc_model(input_image).item()
    # throttle and brake set by experiment
    brake = 0.0
    acc = 0.4

    return (acc, steer, brake)
```

Behavior cloning (BC)



- Testing scenario image does not in the collecting data ;
  - Velocity is faster or slower ;
  - How to get the perfect performance ；

Carla over ! But …

Intermediate project: Leader-follower instance

Control a vehicle: Traditional or learning

Traditional: Keyboard and self-driving PID

Leader-follower instance: keyboard – PID

Tips:

① How to get the state information of leader ?

② What does the follower follow ?

③ How to control two vehicles ?