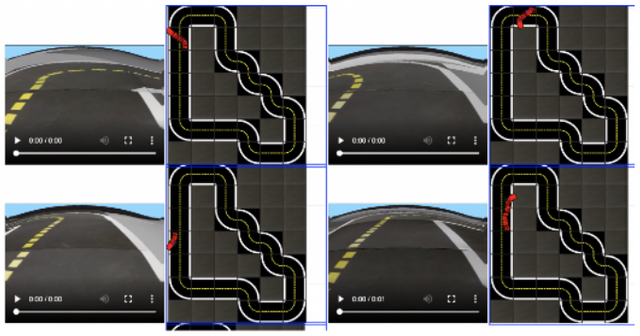
The AI Driving Olympics

with Duckietown

以人工智慧(深度學習、強化學習)實現自動駕駛的競賽！

AIDO I 於 NeuIPS 2018 舉辦 → ICRA 2019 International Conference on Robotics and Automation → AIDO II 於 ICRA 2019 舉辦 → AIDO III 於 NeuIPS 2019 舉辦



第一階段：將Solution(程式碼)上傳至AIDO，官方將以gym測試您的結果。

第二階段：將Solution在真實場地、執行於真正的機器人(Duckiebot)

不需準備器材！不需準備場地！不需到現場！只要上傳你的Solution！

如何開始?

應用所學!

Deep RL

Imitation Learning

Why AIDO? 將近期Deep Learning, Reinforcement Learning爆炸性的結果，應用於真實的自駕車，發揮所學，也能作為專題題目！

官方提供基礎範例、模板 (Baseline Solution, Template)

線上課程: https://vimeo.com/331891746

官方網站: https://www.duckietown.org/research/ai-driving-olympics

Duckiebook: http://docs.duckietown.org/DT19/AIDO/out/index.html

Bonus Lab: DDPG for Self-Driving Car in AI Driving Olympics

Presenter Chen-Lung Lu

If DDPG can do those, of course it can drive cars

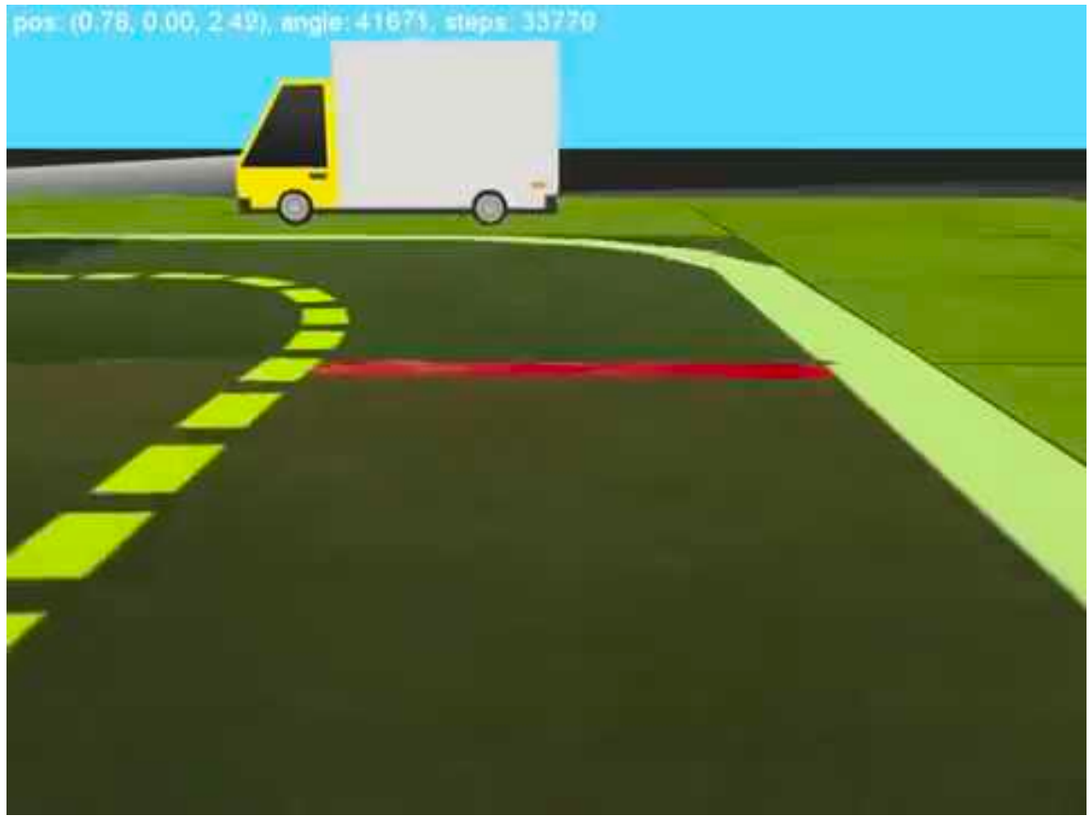
Search “DDPG Torcs”

Duckietown and gym-duckietown

A robotics research and education platform initiated in MIT CSAIL in 2016.

Everyone can play around robots and self-driving in a low cost environment.

[1]L. Paull, et al, “Duckietown: An open, inexpensive and flexible platform for autonomy education and research,” 2017 IEEE International Conference on Robotics and Automation (ICRA), May 2017.



A simulated environment we can train and test our DDPG algorithm.

And also other algorithm if you would like to.

DDPG: Game Environment - gym-duckietown

Your goal is try to make your "Duckiebot" stay in the middle of the right lane and travel as far as possible.

State:

RGB-image: 160*120*3 (0~255) (after wrapping)

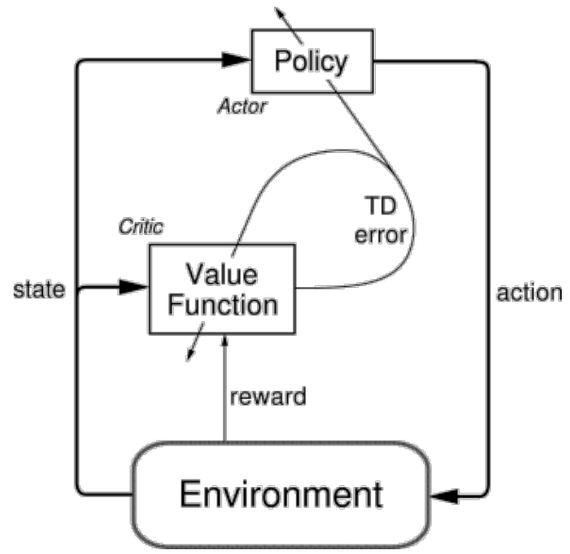
Actions:

Left and right wheels torques: [-1, 1]

Reward:

If in invalid step: -10 (after wrapping)

If reach maximum step: 0+4 (after wrapping)



DDPG: Run Baseline Solution

Requirement: **Python3.7** (You might need a update), pip3, github
Machine: You may NOT use ssh to a remote machine. You could use either local machines or remote desktop app (e.x. teamviewer).

```
Laptop $ git clone git://github.com/duckietown/challenge-aido_LF-baseline-RL-sim-pytorch.git
Laptop $ cd challenge-aido1_LF1-baseline-RL-sim-pytorch
```

```
Laptop $ pip3 install -e .
Laptop $ pip3 install -e git://github.com/duckietown/gym-duckietown.git#egg=gym-duckietown
```

Start training

```
Laptop $ cd duckietown_rl
Laptop $ python3 -m scripts.train_cnn.py --seed 123
```

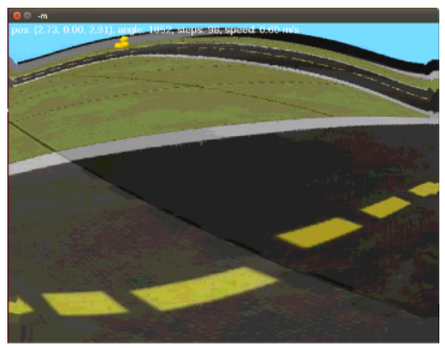
After training, your policies (models) will be saved at
duckietown_rl/pytorch_models
With name:
Actor model: DDPG_(seed number)_actor.pth
Critic model: DDPG_(seed number)_critic.pth

Please edit the file **scripts/test_cnn.py line 11**
seed = <YOUR_SEED_NUMBER> (in this example, 123)

Test your trained policy

```
Laptop $ python3 -m scripts.test_cnn.py
```

A screen should shows up demonstrate the result. The video might seems too fast. You can add a time pause after each step.



add time pause
At line 50, after `env.render()`, add following line

```
import time
Time.sleep(0.1)
```

DDPG: Code Explanation - train_cnn.py

In duckietown_rl/scripts/train_cnn.py

Wrappers: Customize your environments

```
# Wrappers
env = ResizeWrapper(env)
env = NormalizeWrapper(env)
env = ImgWrapper(env) # to make the images from 160x120x3 into 3x160x120
env = ActionWrapper(env)
env = DtRewardWrapper(env)
```

Initialize Policy and ReplayBuffer

```
# Initialize policy
policy = DDPG(state_dim, action_dim, max_action, net_type="cnn")

replay_buffer = ReplayBuffer()
```

DDPG Algorithm

```
while total_timesteps < args.max_timesteps:

    if done:

        if total_timesteps != 0:
            print(("Total T: %d Episode Num: %d Episode T: %d Reward: %f") % (
                total_timesteps, episode_num, episode_timesteps, episode_reward))
            policy.train(replay_buffer, episode_timesteps, args.batch_size, args.discount, args.tau)

        # Evaluate episode
        if timesteps_since_eval >= args.eval_freq:
            timesteps_since_eval %= args.eval_freq
            evaluations.append(evaluate_policy(env, policy))

            if args.save_models:
                policy.save(file_name, directory="./pytorch_models")
            np.savez("./results/{}.npz".format(file_name),evaluations)

        # Reset environment
        env_counter += 1
        obs = env.reset()
        done = False
        episode_reward = 0
        episode_timesteps = 0
        episode_num += 1

    # Select action randomly or according to policy
    if total_timesteps < args.start_timesteps:
        action = env.action_space.sample()
    else:
        action = policy.predict(np.array(obs))
        if args.expl_noise != 0:
            action = (action + np.random.normal(
                0,
                args.expl_noise,
                size=env.action_space.shape[0])
                      ).clip(env.action_space.low, env.action_space.high)

    # Perform action
    new_obs, reward, done, _ = env.step(action)
    if episode_timesteps >= args.env_timesteps:
        done = True

    done_bool = 0 if episode_timesteps + 1 == args.env_timesteps else float(done)
    episode_reward += reward

    # Store data in replay buffer
    replay_buffer.add(obs, new_obs, action, reward, done_bool)

    obs = new_obs

    episode_timesteps += 1
    total_timesteps += 1
    timesteps_since_eval += 1
```

DDPG: Code Explanation - wrappers.py

In duckietown_rl/wrappers.py

ResizeWrapper: resize your state (image) from 640*480 to 160*120

NormalizeWrapper: normalize the input data to 0~1

ImgWrapper: transform the image from 160*120*3 to 3*160*120

ActionWrapper: Make sure there's no extreme value in our output

****DtRewardWrapper**: Customize your reward

```
class DtRewardWrapper(gym.RewardWrapper):
    def __init__(self, env):
        super(DtRewardWrapper, self).__init__(env)

    def reward(self, reward):
        if reward == -1000:
            reward = -10
        elif reward > 0:
            reward += 10
        else:
            reward += 4

        return reward
```

DDPG: Code Explanation - ddpg.py

In duckietown_rl/ddpg.py

Here is where your agent (network) is

You can see what will be done if you call “policy.train()”, “policy.predict()”

Critic Network

```python
class CriticCNN(nn.Module):
    def __init__(self, action_dim):
        super(CriticCNN, self).__init__()

        flat_size = 32 * 9 * 14

        self.lr = nn.LeakyReLU()

        self.conv1 = nn.Conv2d(3, 32, 8, stride=2)
        self.conv2 = nn.Conv2d(32, 32, 4, stride=2)
        self.conv3 = nn.Conv2d(32, 32, 4, stride=2)
        self.conv4 = nn.Conv2d(32, 32, 4, stride=1)

        self.bn1 = nn.BatchNorm2d(32)
        self.bn2 = nn.BatchNorm2d(32)
        self.bn3 = nn.BatchNorm2d(32)
        self.bn4 = nn.BatchNorm2d(32)

        self.dropout = nn.Dropout(.5)

        self.lin1 = nn.Linear(flat_size, 256)
        self.lin2 = nn.Linear(256 + action_dim, 128)
        self.lin3 = nn.Linear(128, 1)

    def forward(self, states, actions):
        x = self.bn1(self.lr(self.conv1(states)))
        x = self.bn2(self.lr(self.conv2(x)))
        x = self.bn3(self.lr(self.conv3(x)))
        x = self.bn4(self.lr(self.conv4(x)))
        x = x.view(x.size(0), -1)  # flatten
        x = self.lr(self.lin1(x))
        x = self.lr(self.lin2(torch.cat([x, actions], 1)))  # c
        x = self.lin3(x)

        return x
```

Actor Network

```python
class ActorCNN(nn.Module):
    def __init__(self, action_dim, max_action):
        super(ActorCNN, self).__init__()

        # ONLY TRU IN CASE OF DUCKIETOWN:
        flat_size = 32 * 9 * 14

        self.lr = nn.LeakyReLU()
        self.tanh = nn.Tanh()
        self.sigm = nn.Sigmoid()

        self.conv1 = nn.Conv2d(3, 32, 8, stride=2)
        self.conv2 = nn.Conv2d(32, 32, 4, stride=2)
        self.conv3 = nn.Conv2d(32, 32, 4, stride=2)
        self.conv4 = nn.Conv2d(32, 32, 4, stride=1)

        self.bn1 = nn.BatchNorm2d(32)
        self.bn2 = nn.BatchNorm2d(32)
        self.bn3 = nn.BatchNorm2d(32)
        self.bn4 = nn.BatchNorm2d(32)

        self.dropout = nn.Dropout(.5)

        self.lin1 = nn.Linear(flat_size, 512)
        self.lin2 = nn.Linear(512, action_dim)

        self.max_action = max_action

    def forward(self, x):
        x = self.bn1(self.lr(self.conv1(x)))
        x = self.bn2(self.lr(self.conv2(x)))
        x = self.bn3(self.lr(self.conv3(x)))
        x = self.bn4(self.lr(self.conv4(x)))
        x = x.view(x.size(0), -1)  # flatten
        x = self.dropout(x)
        x = self.lr(self.lin1(x))

        # this is the vanilla implementation
        # but we're using a slightly different one
        # x = self.max_action * self.tanh(self.lin2(x))
        # because we don't want our duckie to go backwards
        x = self.lin2(x)
        x[:, 0] = self.max_action * self.sigm(x[:, 0])  # because
        x[:, 1] = self.tanh(x[:, 1])

        return x
```

DDPG: Rule of Thumb

- Don’t set replay buffer size too big. If it costs more memory than your RAM, training process would become very slow. See file duckietown_rl/args.py
- You actually don’t need that much time step. Set max_timestep to 5e5 or even lower. See file duckietown_rl/args.py
- Every evaluate frequency (in this case 5000), the code will save a model.

In duckietown_rl/args.py

```python
def get_ddpg_args_train():
    parser = argparse.ArgumentParser()
    parser.add_argument("--seed", default=0, type=int)  # Sets Gym, PyTorch and Numpy seeds
    parser.add_argument("--start_timesteps", default=1e4, type=int)  # How many time steps purely random policy is run for
    parser.add_argument("--eval_freq", default=5e3, type=float)  # How often (time steps) we evaluate
    parser.add_argument("--max_timesteps", default=1e6, type=float)  # Max time steps to run environment for
    parser.add_argument("--save_models", action="store_true", default=True)  # Whether or not models are saved
    parser.add_argument("--expl_noise", default=0.1, type=float)  # Std of Gaussian exploration noise
    parser.add_argument("--batch_size", default=32, type=int)  # Batch size for both actor and critic
    parser.add_argument("--discount", default=0.99, type=float)  # Discount factor
    parser.add_argument("--tau", default=0.005, type=float)  # Target network update rate
    parser.add_argument("--policy_noise", default=0.2, type=float)  # Noise added to target policy during critic update
    parser.add_argument("--noise_clip", default=0.5, type=float)  # Range to clip target policy noise
    parser.add_argument("--policy_freq", default=2, type=int)  # Frequency of delayed policy updates
    parser.add_argument("--env_timesteps", default=500, type=int)  # Frequency of delayed policy updates
    parser.add_argument("--replay_buffer_max_size", default=10000, type=int)  # Maximum number of steps to keep in the replay buffer
```

DDPG:How to make it better?

- Check out the duckietown_rl/wrapper.py, modify rewards in class DtRewardWrapper (set them higher or lower and see what happens)
- Check out src/gym-duckietown/gym_duckietown/simulator.py. See function compute_reward and try play around with it.
- Use other network ex: ResNet
- Cut off the horizon from the image (and correspondingly change the convnet parameters).
- Try making the observation image grayscale instead of color. And while you're at it, try stacking multiple images, like 4 monochrome images instead of 1 color image

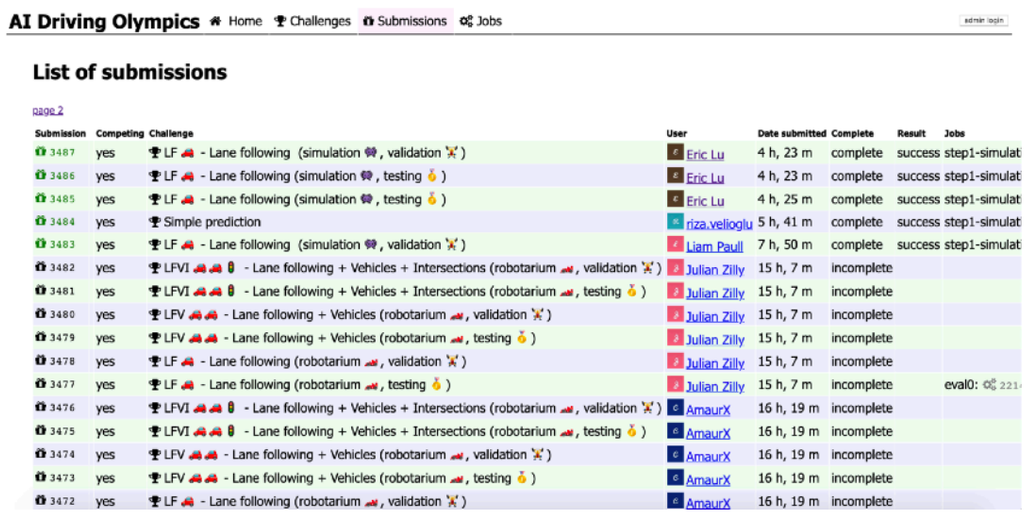
```python
def compute_reward(self, pos, angle, speed):
    # Compute the collision avoidance penalty
    col_penalty = self._proximity_penalty2(pos, angle)

    # Get the position relative to the right lane tangent
    try:
        lp = self.get_lane_pos2(pos, angle)
    except NotInLane:
        reward = 40 * col_penalty
    else:

        # Compute the reward
        reward = (
                +1.0 * speed * lp.dot_dir +
                -10 * np.abs(lp.dist) +
                +40 * col_penalty
        )
    return reward
```

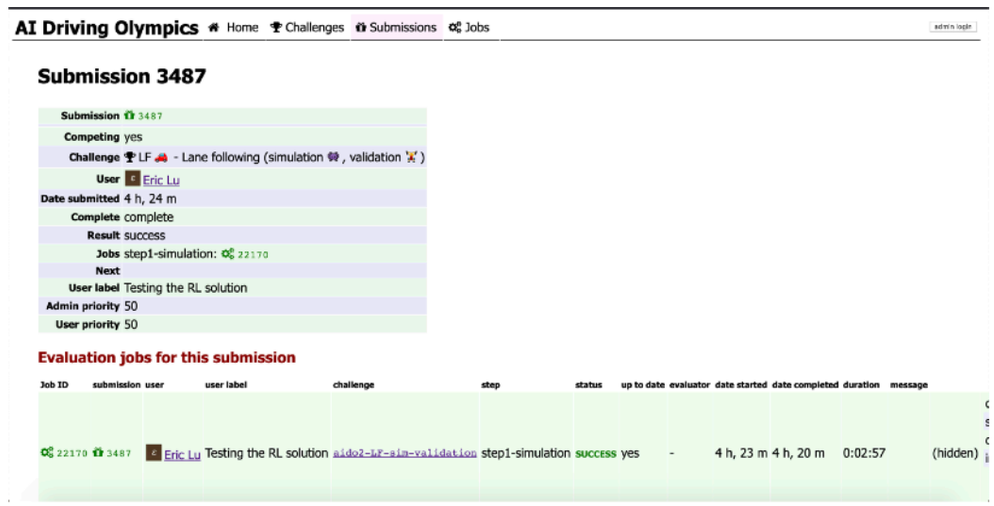
DDPG:Scoring Criteria

- Run baseline solution.
- Send your “wrapper.py”, “ddpg.py” and trained model (both actor and critic) in duckietown_rl/pytorch_models folder.
- Zip all file in one file and name it like “DLP_Duckietown_yourstudentID_name”

- Alternate way (before 5/21)
  - Submit your solution model to AIDO website. (see appendix tutorial in lab material)
  - Send screenshot of your submission

(screenshot this page)





Any Question?