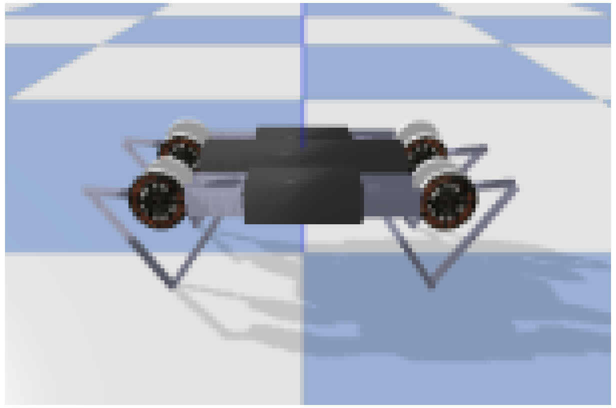
9강. DQN활용
(Kuka RL)

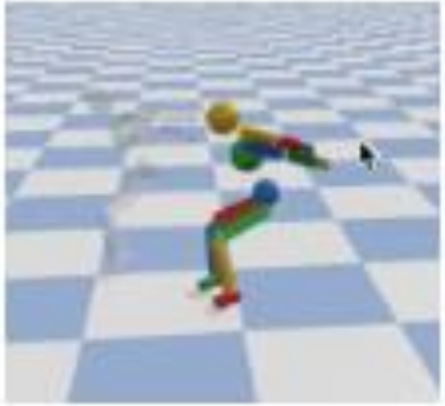
Pybullet

Pybullet

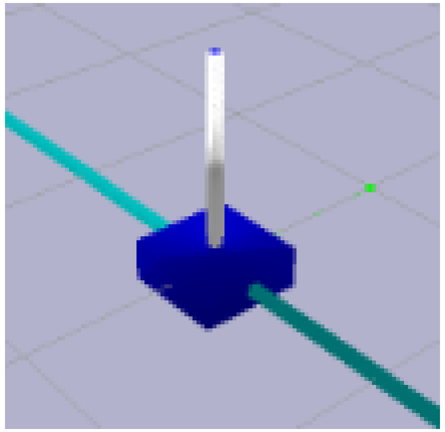
MinitaurBulletEnv-v0



HumanoidDeepMimic*BulletEnv-v1



CartPoleContinuousBulletEnv-v0

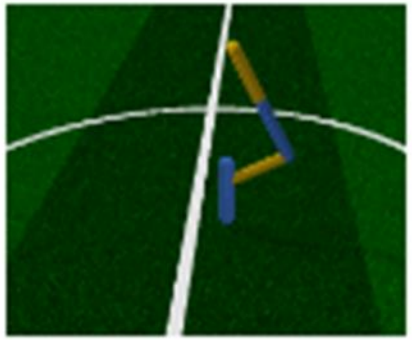


Pybullet

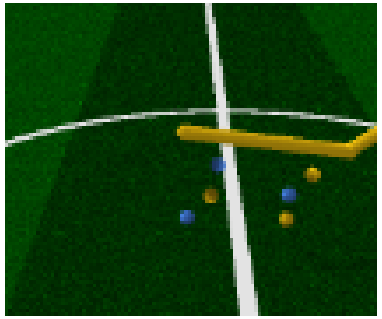
HumanoidBulletEnv-v0

AntBulletEnv-v0

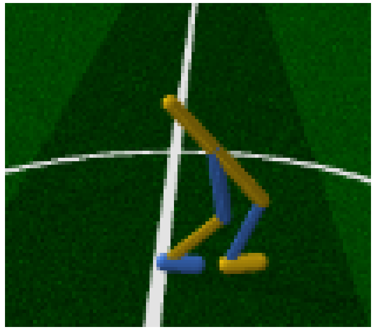
HopperBulletEnv-v0



HalfCheetahBulletEnv-v0

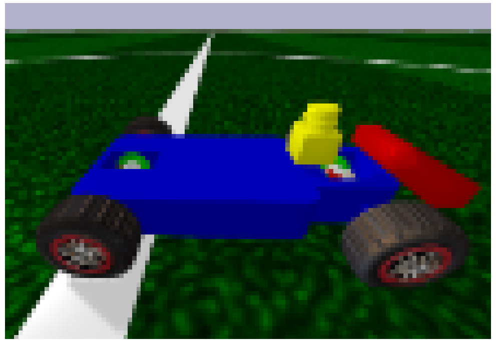


Walker2DBulletEnv-v0



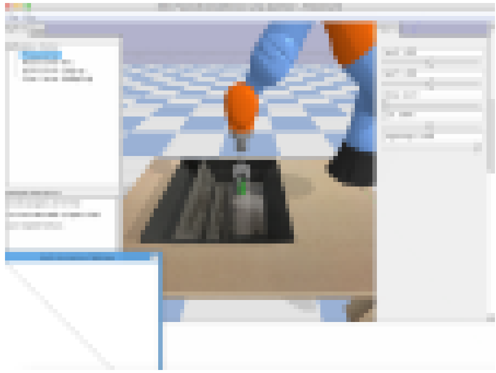
Pybullet

RacecarBulletEnv-v0

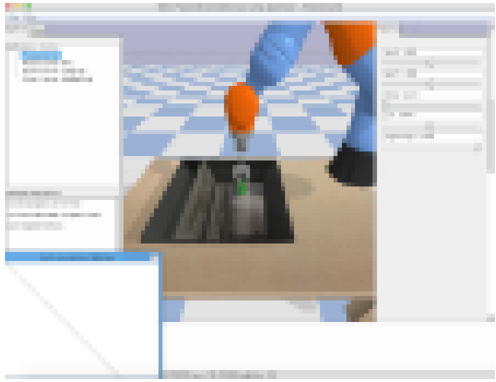


RacecarZedBulletEnv-v0

KukaBulletEnv-v0



KukaCamBulletEnv-v0



Pybullet

- A suite of RL Gym Environments are installed during "pip install pybullet"
- This includes PyBullet versions of the OpenAI Gym environments such as ant, hopper, humanoid and walker.
- There are also environments that apply in simulation as well as on real robots, such as the Ghost Robotics Minitaur quadruped, the MIT racecar and the KUKA robot arm grasping environments.

Pybullet

- The source code of pybullet, pybullet_envs, pybullet_data and the examples are here:
- https://github.com/bulletphysics/bullet3/tree/master/examples/pybullet/gym

- You can train the environments with RL training algorithms such as DQN, PPO, TRPO and DDPG.

Pybullet

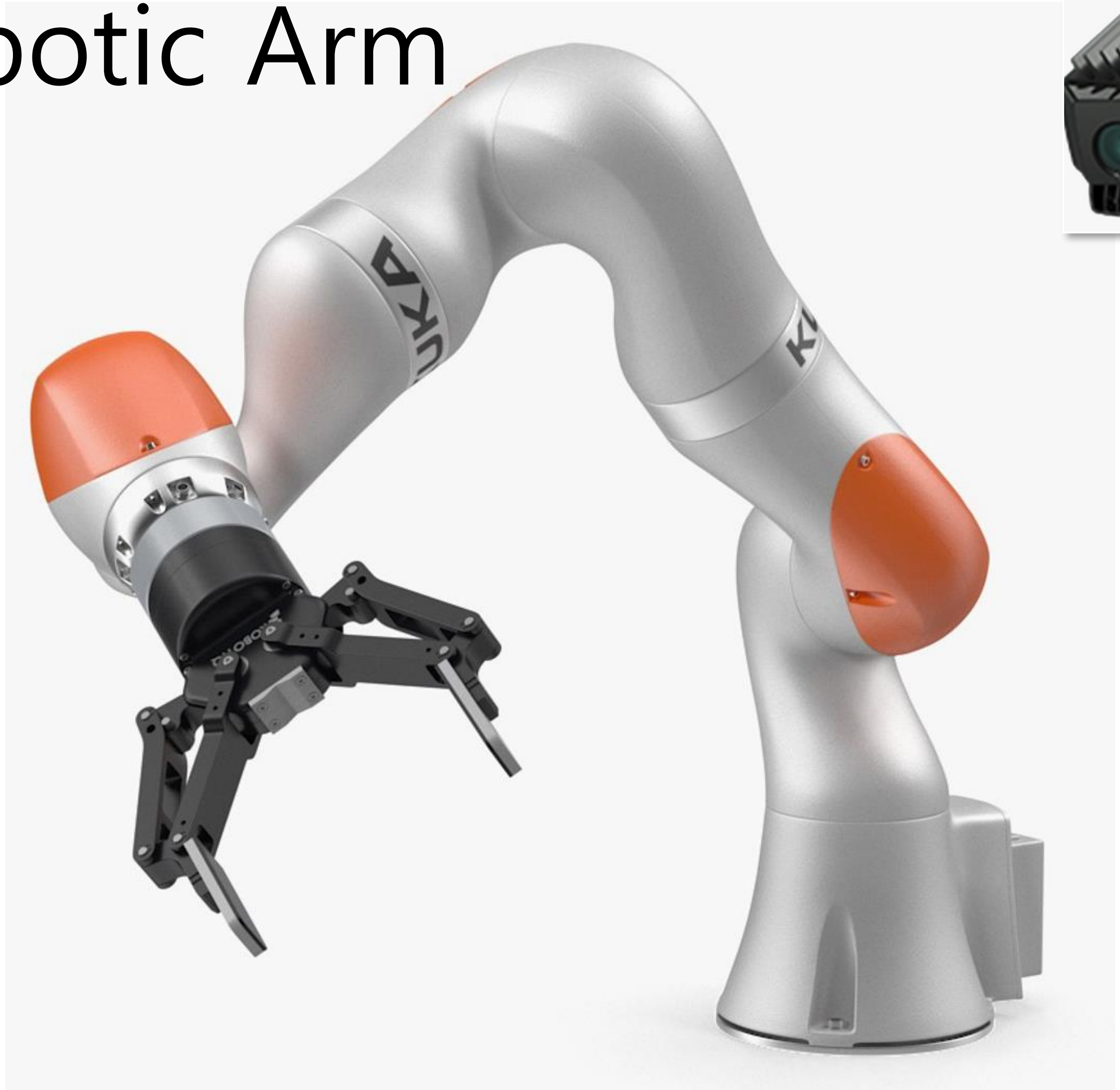
- Several pre-trained examples are available, you can enjoy them like this:
- pip install gym, pybullet, tensorflow, torch
- python -m pybullet_envs.examples.kukaGymEnvTest
- python -m pybullet_envs.examples.minitaur_gym_env_example

Pybullet

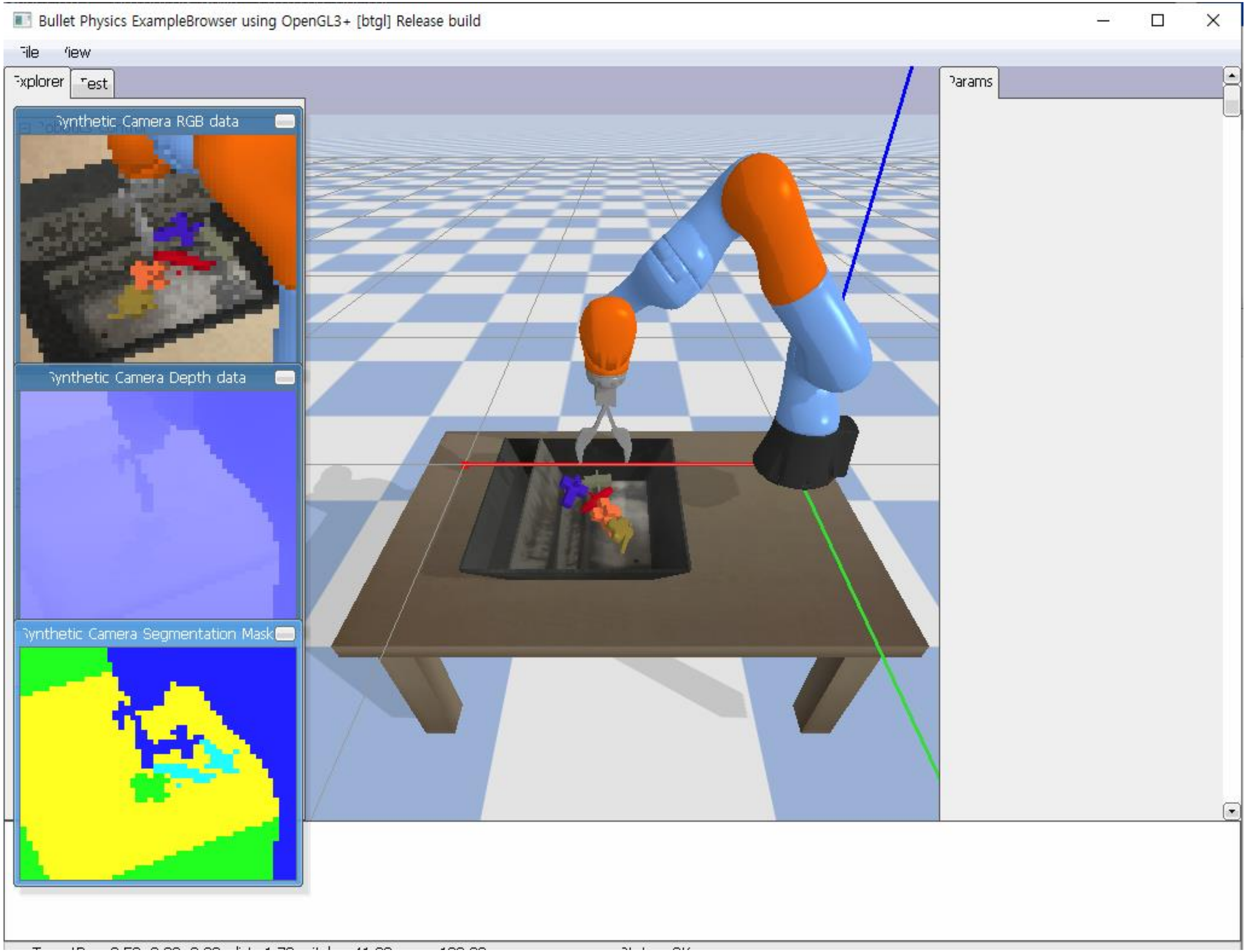
- Windows Gym Package Install:
  - conda install -c conda-forge box2d-py
  - pip install gym==0.23.1
  - **Pip install pybullet**
  - Pip install numpy==1.23.1
  - Pip install matplotlib

Kuka Robotic Arm

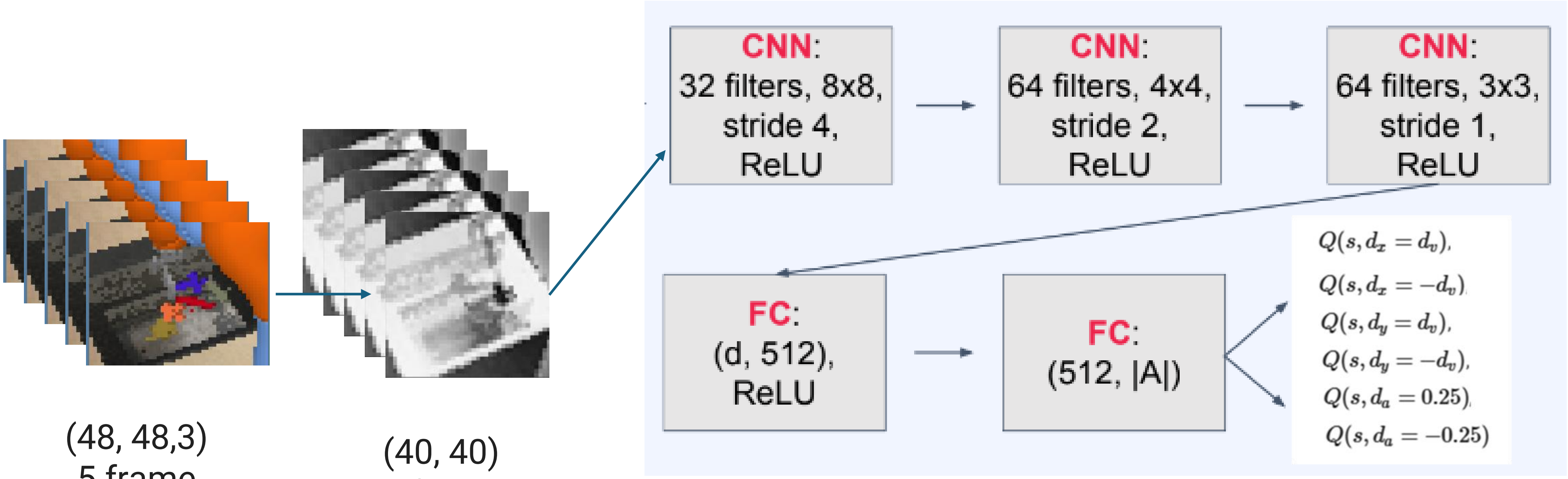
Kuka Robotic Arm



Kuka
Pybullet



Q-network



(48, 48,3)
5 frame
Stack

(40, 40)
5 frame
Stack

dv = 0.06 is the velocity in PyBullet
da = 0.25 is vertical angle offset for the gripper

States

- The input image signal is (48, 48, 3) RGB image and timestep t
  - Timestep, t is included in the state, since the policy must know how many steps remain in the episode to decide whether it must immediately move into a good grasping position.
- NN can solve the task purely by looking at the scene.
  - So, we'll use a stack of consecutive screens as an input.
  - In this way, we are hoping to capture the dynamics of the environment.

Actions

- The agent has to decide between 7 actions (0~6)
- In x (2) or y (2) direction, the manipulator moves
  - Actions correspond to changes in gripper pose (displacement)
  - Assumes that the velocity for each directions equal (±0.06)
  - Gripper automatically move down for each action (z: -0.06)

Actions

- Vertical angle offset (2) for the gripper :
  - The arm moves via position control of vertically- oriented gripper
  - We assume that the vertical angle offset for the gripper (± 0.25)
  - Gripper automatically closes if it moves below a fixed height threshold
- Not moving at all (1) : So that the manipulator can grasp an object

Reward

- The reward is binary and provided only at the last step.
- 1 if one of the objects is above height .2 (successful grasp)
- 0 for a failed grasp
- The arm has a fixed number of timesteps (T = 15) to find a good grasp, at which the episode ends

Setup Code Env.

- Git download: https://git-scm.com/downloads
- Windows Package설치:
  - 가상환경: Create: conda create –n [이름] python=3.8
  - Pytorch설치: https://pytorch.kr/get-started/locally/ 명령어라인 복사
  - `pip install pybullet`
  - `pip install tensorboardX`
  - pip install gym==0.23.1
  - Pip install numpy==1.23.1
  - Pip install matplotlib
- Code:
  - Colab:
    - https://colab.research.google.com/github/mahyaret/kuka_rl/blob/master/kuka_rl.ipynb#scrollTo=5B4fbx6MUnnE
  - Windows:
    - Train with 'bullet_kuka_train.py' (use 'policy_dqn.pt' after training)
    - Evaluate with 'bullet_kuka_eval.py'

Env.

```python
import gym
import math
import random
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from collections import namedtuple
import collections
from itertools import count
import timeit
from datetime import timedelta
from PIL import Image
from tensorboardX import SummaryWriter

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms as T
```

Image transform

```python
from pybullet_envs.bullet.kuka_diverse_object_gym_env import KukaDiverseObjectEn
from gym import spaces
import pybullet as p
```

pybullet

Env.

```
env = KukaDiverseObjectEnv(renders=False, isDiscrete=True, removeHeightHack=Fals
env.cid = p.connect(p.DIRECT)

# set up matplotlib
is_ipython = 'inline' in matplotlib.get_backend()
if is_ipython:
    from IPython import display

plt.ion()

# if gpu is to be used
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Gripper automatically move down for each action (z: -0.06

does Matplotlib backend support Jupyter notebook's line display?

```
preprocess = T.Compose([T.ToPILImage(),
                        T.Grayscale(num_output_channels=1),
                        T.Resize(40, interpolation=Image.CUBIC),
                        T.ToTensor()])
```

torchvision image transform:
Color: black&whilte
Resize :(40x40)
Interpolation: cubic function
Normalize: 0~1

Preprocess

```
preprocess = T.Compose([T.ToPILImage(),
                        T.Grayscale(num_output_channels=1),
                        T.Resize(40, interpolation=Image.CUBIC),
                        T.ToTensor()])
```

torchvision image transform:
Color: black&whilte
Resize :(40x40)
Interpolation: cubic function
Normalize: 0~1

```
def get_screen():
    global stacked_screens
```

```
    screen = env._get_observation().transpose((2, 0, 1))
    # Convert to float, rescale, convert to torch tensor
    # (this doesn't require a copy)

    screen = np.ascontiguousarray(screen, dtype=np.float32) / 255
    screen = torch.from_numpy(screen)
    # Resize, and add a batch dimension (BCHW)
    return preprocess(screen).unsqueeze(0).to(device)
```

Numpy:(48, 48, 3)->(3, 48, 48)

Store array as memory sequence to improve r/w time delay

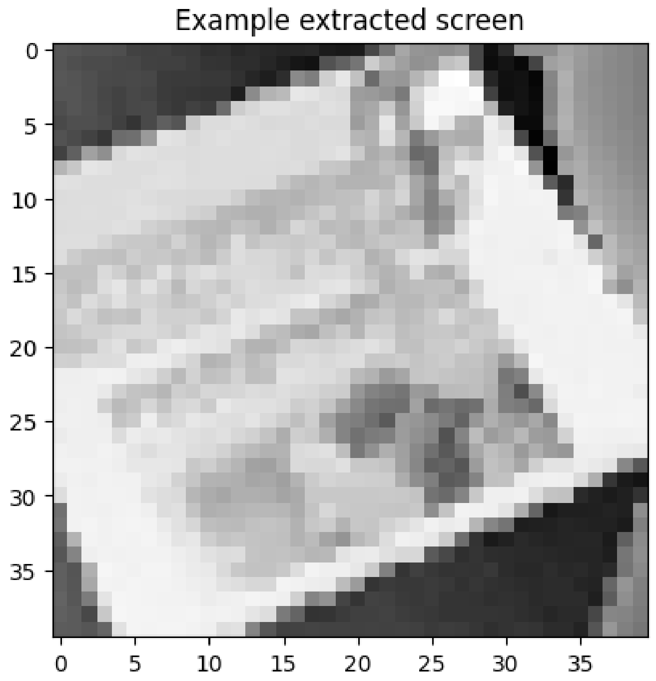
Numpy:(3, 48, 48)->(1, 40, 40)
-> Tensor:([1, 1, 40, 40])
-> Cuda tensor([1, 1, 40, 40])

Preprocess

```
env.reset()
plt.figure()
plt.imshow(get_screen().cpu().squeeze(0)[-1].numpy(),cmap='Greys',
           interpolation='none')
plt.title('Example extracted screen')
plt.show()
```

get_screen() returns Cuda tensor([1, 1, 40, 40]).
Change it to CPU tensor

To display with matplotlib, remove batch dim. and get the dim. :
->torch.Size([1, 40, 40])
->torch.Size([40, 40])



Replay buffer

```
Transition = namedtuple('Transition',
                        ('state', 'action', 'next_state', 'reward'))
```

namedtuple returns 'Transition' class
We can access Transition class

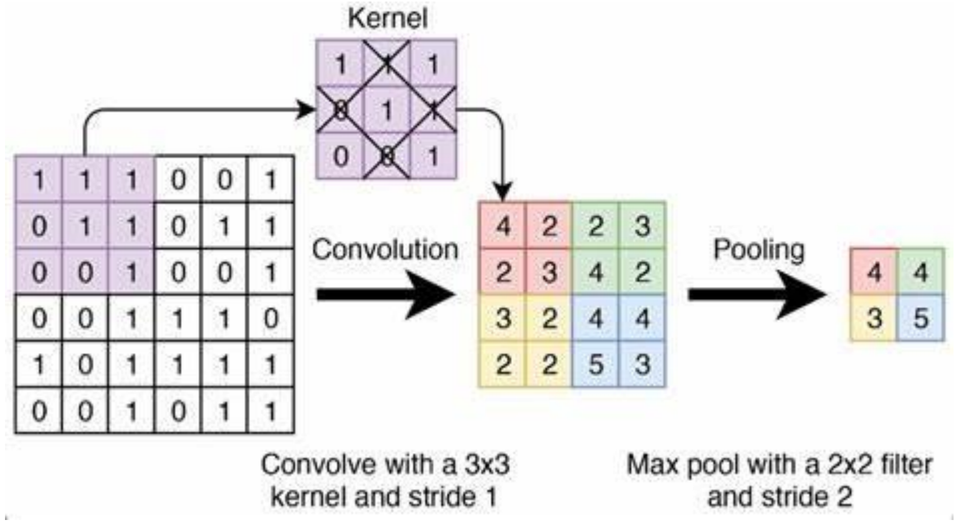
```
class ReplayMemory(object):

    def __init__(self, capacity):
        self.capacity = capacity
        self.memory = []
        self.position = 0

    def push(self, *args):
        """Saves a transition."""
        if len(self.memory) < self.capacity:
            self.memory.append(None)
        self.memory[self.position] = Transition(*args)
        self.position = (self.position + 1) % self.capacity

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

a cyclic buffer of bounded size

Uniform-random picking by the amount of batch_size

Q network

- How to calculate the output size of a convolution layer



$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features

n_{out}: number of output features

k: convolution kernel size

p: convolution padding size

s: convolution stride size

Q network

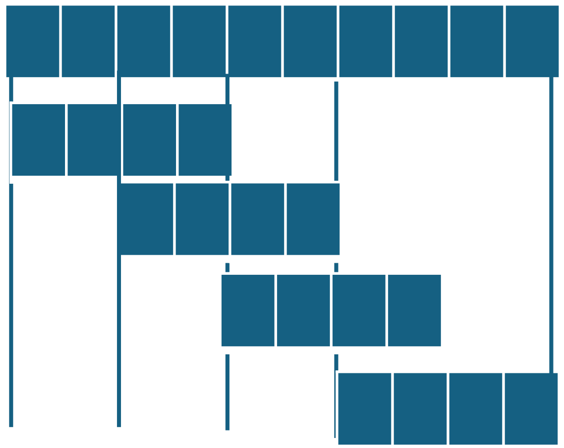
```
class DQN(nn.Module):
    def __init__(self, h, w, outputs):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(STACK_SIZE, 32, kernel_size=8, stride=4)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, stride=1)
```

```
        def conv2d_size_out(size, kernel_size = 5, stride = 2):
            return (size - (kernel_size - 1) - 1) // stride  + 1
        convw = conv2d_size_out(conv2d_size_out(conv2d_size_out(w,8,4),4,2),3,1)
        convh = conv2d_size_out(conv2d_size_out(conv2d_size_out(h,8,4),4,2),3,1)
        linear_input_size = convw * convh * 64
        self.linear = nn.Linear(linear_input_size, 512)
        self.head = nn.Linear(512, outputs)
```

```
STACK_SIZE = 5
```

torch.nn.Conv2d(*in_channels, out_channels, kernel_size, stride=1, padding=0,…*

```
e.g., when w=10
kernel=4 stride=2,
Output is (10-3-1)// 2
+ 1 = 4
```



Q network

```python
def forward(self, x):
    x = F.relu(self.bn1(self.conv1(x)))
    x = F.relu(self.bn2(self.conv2(x)))
    x = F.relu(self.conv3(x))
    x = F.relu(self.linear(x.view(x.size(0), -1)))
    return self.head(x)
```

Net() or net.forward()
Note: the function is called with either
1 element to determine next action,
Or a batch during optimization

Returns tensor([[Q(s, dx=dv),
Q(s, dx=−dv),
Q(s, dy=dv),
Q(s, dy=−dv),
Q(s, da=0.25),
Q(s, da=−0.25)]])

Q network

```
# Get screen size so that we can initialize layers correctly based on shape
# returned from pybullet (48, 48, 3).
init_screen = get_screen()
_, _, screen_height, screen_width = init_screen.shape

# Get number of actions from gym action space
n_actions = env.action_space.n

policy_net = DQN(screen_height, screen_width, n_actions).to(device)
target_net = DQN(screen_height, screen_width, n_actions).to(device)
target_net.load_state_dict(policy_net.state_dict())
target_net.eval()

optimizer = optim.Adam(policy_net.parameters(), lr=LEARNING_RATE)
memory = ReplayMemory(10000)
```

Output is torch.Size([1, 1, 40, 40])

Discrete(7)

Send network to GPU

Learning rate is 1e-4

Q network

```
eps_threshold = 0

def select_action(state, i_episode):
    global steps_done
    global eps_threshold
    sample = random.random()
    eps_threshold = max(EPS_END, EPS_START - i_episode / EPS_DECAY_LAST_FRAME)
    if sample > eps_threshold:
        with torch.no_grad():
            return policy_net(state).max(1)[1].view(1, 1)
    else:
        return torch.tensor([[random.randrange(n_actions)]], device=device, dtyp
```

EPS_START = 0.9
EPS_END = 0.1
EPS_DECAY = 200
EPS_DECAY_LAST_FRAME = 10**4
i_episode: 0~ 10**7

2nd column on max result is the max. action index where max element was found

Send action data GPU

Training

BATCH_SIZE = 32

*transitions: (s,a,s′,r), (s,a,s′,r),..
*zip: (s,s,..), (a,a,...), (s′,s;,..), (r,r,...)
batch: (state 32개(s,s,...),
action 32개(a,a,...),
next_state 32개(s′,s,...),
reward 32개(r,r,...))

```
def optimize_model():
    if len(memory) < BATCH_SIZE:
        return
    transitions = memory.sample(BATCH_SIZE)

    batch = Transition(*zip(*transitions))

    non_final_mask = torch.tensor(tuple(map(lambda s: s is not None,
                                          batch.next_state)), device=device, dtype=torch.bool)
    non_final_next_states = torch.cat([s for s in batch.next_state
                                                if s is not None])
```

torch.Size([32])
:tensor([T,T,..F,F,T])
is used as mask to retrieve non-None next-states

torch.Size([28, 5, 40, 40])
stores non-None next-states

tuple(map(f, tuple))
batch.next_state: 32개(s′,s′,...)

```
    state_batch = torch.cat(batch.state)
    action_batch = torch.cat(batch.action)
    reward_batch = torch.cat(batch.reward)

    state_action_values = policy_net(state_batch).gather(1, action_batch)
```

torch.Size([32, 5, 40, 40])
torch.Size([32, 1])
torch.Size([32])

Training

Compute Q(s_t, a): the model computes Q(s_t), then we select the columns of action taken

```
def optimize_model():
```

```
    state_action_values = policy_net(state_batch).gather(1, action_batch)
```

torch.gather(input, dim, index, ...) → gathers values along an axis specified by dim.
e.g.,

```
tensor([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
        [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
        [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34, 35, 36, 37, 38, 39]])
```

→

```
tensor([[ 3],
        [17],
        [24],
        [31]])
```

Compute V(s_t+1) for all next states. (either the expected state value or 0)

```
    next_state_values = torch.zeros(BATCH_SIZE, device=device)
    next_state_values[non_final_mask] = target_net(non_final_next_states).max(1)[0].detach()
    # Compute the expected Q values
    expected_state_action_values = (next_state_values * GAMMA) + reward_batch

    # Compute Huber loss
    loss = F.smooth_l1_loss(state_action_values, expected_state_action_values.unsqueeze(1))

    # Optimize the model
    optimizer.zero_grad()
    loss.backward()
    for param in policy_net.parameters():
        param.grad.data.clamp_(-1, 1)
    optimizer.step()
```

max(1)[0] selects the best return.

$$Q^{\pi}(s,a) = r + \gamma Q^{\pi}(s', \pi(s'))$$

Training loop

```
for i_episode in range(num_episodes):
    # Initialize the environment and state
    env.reset()
    state = get_screen()
    stacked_states = collections.deque(STACK_SIZE*[state],maxlen=STACK_SIZE)
    for t in count():
        stacked_states_t =  torch.cat(tuple(stacked_states),dim=1)
        # Select and perform an action
        action = select_action(stacked_states_t, i_episode)
        _, reward, done, _ = env.step(action.item())
        reward = torch.tensor([reward], device=device)

        # Observe new state
        next_state = get_screen()
        if not done:
            next_stacked_states = stacked_states
            next_stacked_states.append(next_state)
            next_stacked_states_t =  torch.cat(tuple(next_stacked_states),dim=1)
        else:
            next_stacked_states_t = None

        # Store the transition in memory
        memory.push(stacked_states_t, action, next_stacked_states_t, reward)
```

```
for i_episode in range(num_episodes):
        :
        stacked_states = next_stacked_states
        optimize_model()
```

function that performs a single step of the optimization

```
        if done:
            reward = reward.cpu().numpy().item()
            ten_rewards += reward
            total_rewards.append(reward)
            mean_reward = np.mean(total_rewards[-100:])*100
            writer.add_scalar("epsilon", eps_threshold, i_episode)
            if (best_mean_reward is None or best_mean_reward < mean_reward) and i_episode > 100:
                # For saving the model and possibly resuming training
                torch.save({
                        'policy_net_state_dict': policy_net.state_dict(),
                        'target_net_state_dict': target_net.state_dict(),
                        'optimizer_policy_net_state_dict': optimizer.state_dict()
                        }, PATH)
                if best_mean_reward is not None:
                    print("Best mean reward updated %.1f -> %.1f, model saved" % (best_mean_reward, mean_reward))
                best_mean_reward = mean_reward
            break
```

Store the best network parameters

```
    if i_episode%10 == 0:
            writer.add_scalar('ten episodes average rewards', ten_rewards/10.0, i_episode)
            ten_rewards = 0
```

Store the score for tensorboard

```
    if i_episode % TARGET_UPDATE == 0:
        target_net.load_state_dict(policy_net.state_dict())
```

Update target network

```
    if i_episode>=200 and mean_reward>50:
        print('Environment solved in {:d} episodes!₩tAverage Score: {:.2f}'.format(i_episode+1, mean_reward))
        break
```

Exit condition

evaluation

```
episode = 10
scores_window = collections.deque(maxlen=100)  # last 100 scores
env = KukaDiverseObjectEnv(renders=False, isDiscrete=True, removeHeightHack=False, maxSteps=20, isTest=True)
env.cid = p.connect(p.DIRECT)
# load the model
checkpoint = torch.load(PATH)
policy_net.load_state_dict(checkpoint['policy_net_state_dict'])

# evaluate the model
for i_episode in range(episode):
    env.reset()
    state = get_screen()
    stacked_states = collections.deque(STACK_SIZE*[state],maxlen=STACK_SIZE)
    for t in count():
        stacked_states_t =  torch.cat(tuple(stacked_states),dim=1)
        # Select and perform an action
        action = policy_net(stacked_states_t).max(1)[1].view(1, 1)
        _, reward, done, _ = env.step(action.item())
        # Observe new state
        next_state = get_screen()
        stacked_states.append(next_state)
        if done:
            break
    print("Episode: {0:d}, reward: {1}".format(i_episode+1, reward), end="₩n")
```

train

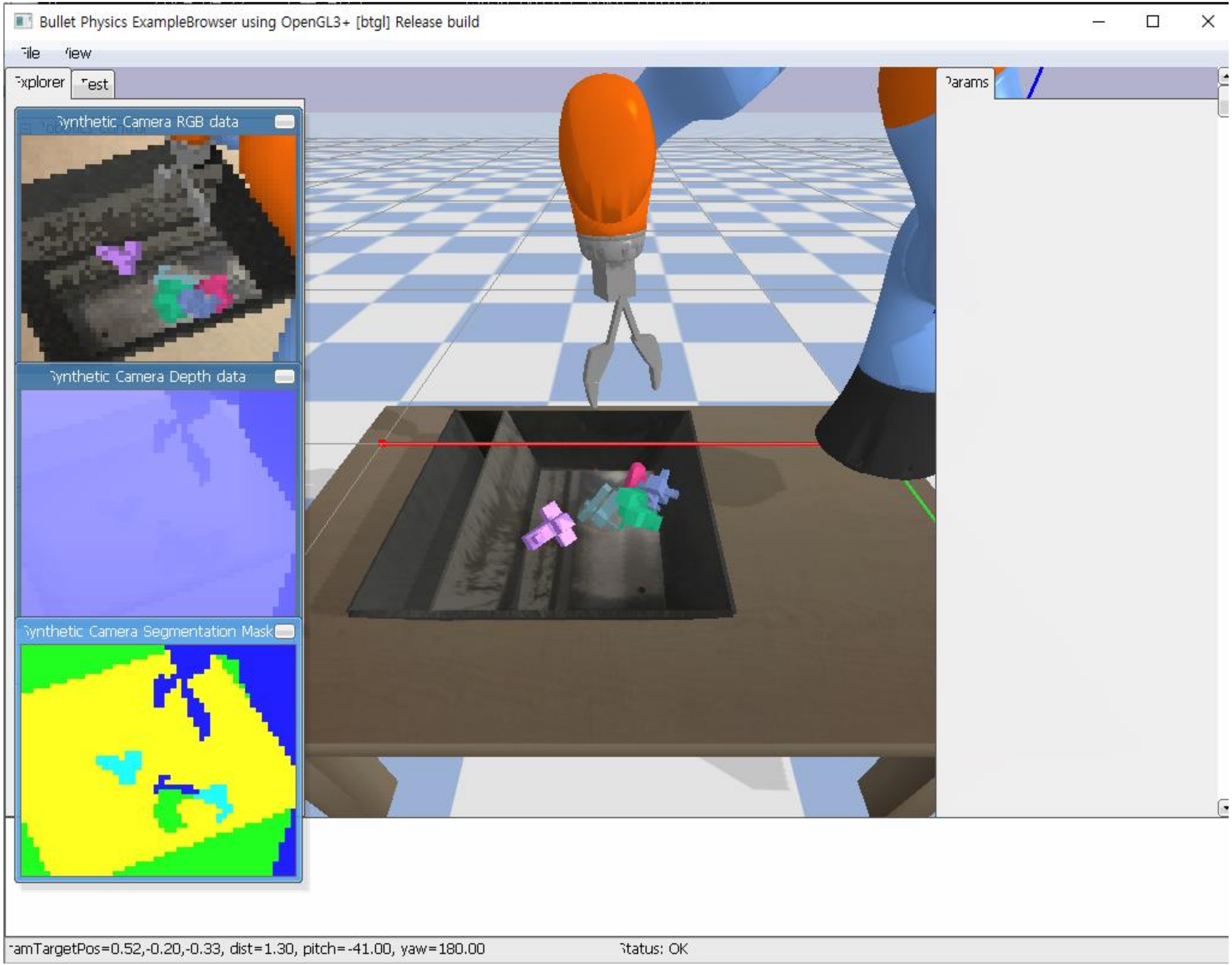
```
python .₩bullet_kuka_train.py
```

```
Best mean reward updated 36.0 -> 37.0, model saved
Best mean reward updated 37.0 -> 38.0, model saved
Best mean reward updated 38.0 -> 39.0, model saved
Best mean reward updated 39.0 -> 40.0, model saved
Best mean reward updated 40.0 -> 41.0, model saved
Best mean reward updated 41.0 -> 42.0, model saved
Best mean reward updated 42.0 -> 43.0, model saved
Best mean reward updated 43.0 -> 44.0, model saved
Best mean reward updated 44.0 -> 45.0, model saved
Best mean reward updated 45.0 -> 46.0, model saved
Best mean reward updated 46.0 -> 47.0, model saved
Best mean reward updated 47.0 -> 48.0, model saved
Best mean reward updated 48.0 -> 49.0, model saved
Best mean reward updated 49.0 -> 50.0, model saved
Best mean reward updated 50.0 -> 51.0, model saved
Environment solved in 24033 episodes!   Average Score: 51.00
Average Score: 51.00
Elapsed time: 17:43:47.831820
```

evaluation

```
python bullet_kuka_eval.py
```

```
Version = 4.6.0 NVIDIA 536.23
Vendor = NVIDIA Corporation
Renderer = NVIDIA GeForce RTX 3090/PCIe/SSE2
b3Printf: Selected demo: Physics Server
starting thread 0
started MotionThreads thread 0 with threadHandle 00000000000005E8
MotionThreadFunc thread started
Episode: 1, reward: 0
Episode: 2, reward: 0
Episode: 3, reward: 0
Episode: 4, reward: 1
Episode: 5, reward: 1
Episode: 6, reward: 1
Episode: 7, reward: 1
Episode: 8, reward: 1
Episode: 9, reward: 1
Episode: 10, reward: 1
numActiveThreads = 0
stopping threads
```


Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build
File
View
Explorer
Test
Synthetic Camera RGB data
Synthetic Camera Depth data
Synthetic Camera Segmentation Mask
Params
amTargetPos=0.52,-0.20,-0.33, dist=1.30, pitch=-41.00, yaw=180.00
Status: OK