APPENDICES

PYTHON CODE

FETCH_ENV.py

```
import numpy as np
from gym.envs.robotics import rotations, robot_env, utils
def goal_distance(goal_a, goal_b):
    assert goal_a.shape == goal_b.shape
    return np.linalg.norm(goal_a - goal_b, axis=-1)
class FetchEnv(robot_env.RobotEnv):
    def __init__(
        self, model_path, n_substeps, gripper_extra_height, block_gripper,
        has_object, target_in_the_air, target_offset, obj_range, target_range,
        distance_threshold, initial_qpos, reward_type,
    ):
        self.gripper_extra_height = gripper_extra_height
        self.block_gripper = block_gripper
        self.has_object = has_object
        self.target_in_the_air = target_in_the_air
```

```
        self.target_offset = target_offset
        self.obj_range = obj_range
        self.target_range = target_range
        self.distance_threshold = distance_threshold
        self.reward_type = reward_type

        super(FetchEnv, self).__init__(
            model_path=model_path, n_substeps=n_substeps, n_actions=4,
            initial_qpos=initial_qpos)
    def compute_reward(self, achieved_goal, goal, info):

        d = goal_distance(achieved_goal, goal)
        if self.reward_type == 'sparse':
            return -(d > self.distance_threshold).astype(np.float32)
        else:
            return -d
    def _step_callback(self):
        if self.block_gripper:
            self.sim.data.set_joint_qpos('robot0:l_gripper_finger_joint', 0.)
            self.sim.data.set_joint_qpos('robot0:r_gripper_finger_joint', 0.)
```

```
            self.sim.forward()

    def _set_action(self, action):
        assert action.shape == (4,)
        action = action.copy()
        pos_ctrl, gripper_ctrl = action[:3], action[3]
        pos_ctrl *= 0.05
        rot_ctrl = [1., 0., 1., 0.]
        gripper_ctrl = np.array([gripper_ctrl, gripper_ctrl])
        assert gripper_ctrl.shape == (2,)
        if self.block_gripper:
            gripper_ctrl = np.zeros_like(gripper_ctrl)
        action = np.concatenate([pos_ctrl, rot_ctrl, gripper_ctrl])
        utils.ctrl_set_action(self.sim, action)
        utils.mocap_set_action(self.sim, action)

    def _get_obs(self):
        grip_pos = self.sim.data.get_site_xpos('robot0:grip')
        dt = self.sim.nsubsteps * self.sim.model.opt.timestep
        grip_velp = self.sim.data.get_site_xvelp('robot0:grip') * dt
        robot_qpos, robot_qvel = utils.robot_get_obs(self.sim)
        if self.has_object:
```

```
        object_pos = self.sim.data.get_site_xpos('object0')
        object_rot = rotations.mat2euler(self.sim.data.get_site_xmat('object0'))
        object_velp = self.sim.data.get_site_xvelp('object0') * dt
        object_velr = self.sim.data.get_site_xvelr('object0') * dt
        object_rel_pos = object_pos - grip_pos
        object_velp -= grip_velp
    else:
        object_pos = object_rot = object_velp = object_velr = object_rel_pos = np.zeros(0)
    gripper_state = robot_qpos[-2:]
    gripper_vel = robot_qvel[-2:] * dt
    if not self.has_object:
        achieved_goal = grip_pos.copy()
    else:
        achieved_goal = np.squeeze(object_pos.copy())
    obs = np.concatenate([
        grip_pos, object_pos.ravel(), object_rel_pos.ravel(), gripper_state, object_rot.ravel(),
        object_velp.ravel(), object_velr.ravel(), grip_velp, gripper_vel,
    ])


    return {
```

```
            'observation': obs.copy(),
            'achieved_goal': achieved_goal.copy(),
            'desired_goal': self.goal.copy(),
        }
    def _viewer_setup(self):
        body_id = self.sim.model.body_name2id('robot0:gripper_link')
        lookat = self.sim.data.body_xpos[body_id]
        for idx, value in enumerate(lookat):
            self.viewer.cam.lookat[idx] = value
        self.viewer.cam.distance = 2.5
        self.viewer.cam.azimuth = 132.
        self.viewer.cam.elevation = -14.

    def _render_callback(self):

        sites_offset = (self.sim.data.site_xpos - self.sim.model.site_pos).copy()
        site_id = self.sim.model.site_name2id('target0')
        self.sim.model.site_pos[site_id] = self.goal - sites_offset[0]
        self.sim.forward()
```

```
    def _reset_sim(self):
        self.sim.set_state(self.initial_state)
        if self.has_object:
            object_xpos = self.initial_gripper_xpos[:2]
            while np.linalg.norm(object_xpos - self.initial_gripper_xpos[:2]) < 0.1:
                object_xpos = self.initial_gripper_xpos[:2] + self.np_random.uniform(-self.obj_range, self.obj_range, size=2)
            object_qpos = self.sim.data.get_joint_qpos('object0:joint')
            assert object_qpos.shape == (7,)
            object_qpos[:2] = object_xpos
            self.sim.data.set_joint_qpos('object0:joint', object_qpos)

        self.sim.forward()
        return True

    def _sample_goal(self):
        if self.has_object:
            goal = self.initial_gripper_xpos[:3] + self.np_random.uniform(-self.target_range, self.target_range, size=3)
            goal += self.target_offset
            goal[2] = self.height_offset
```

```
            if self.target_in_the_air and self.np_random.uniform() < 0.5:
                goal[2] += self.np_random.uniform(0, 0.45)
        else:
            goal = self.initial_gripper_xpos[:3] + self.np_random.uniform(-0.15, 0.15, size=3)
        return goal.copy()

    def _is_success(self, achieved_goal, desired_goal):
        d = goal_distance(achieved_goal, desired_goal)
        return (d < self.distance_threshold).astype(np.float32)

    def _env_setup(self, initial_qpos):
        for name, value in initial_qpos.items():
            self.sim.data.set_joint_qpos(name, value)
        utils.reset_mocap_welds(self.sim)
        self.sim.forward()
        gripper_target = np.array([-0.498, 0.005, -0.431 + self.gripper_extra_height]) + self.sim.data.get_site_xpos('robot0:grip')
        gripper_rotation = np.array([1., 0., 1., 0.])
        self.sim.data.set_mocap_pos('robot0:mocap', gripper_target)
        self.sim.data.set_mocap_quat('robot0:mocap', gripper_rotation)
        for _ in range(10):
            self.sim.step()
        self.initial_gripper_xpos = self.sim.data.get_site_xpos('robot0:grip').copy()
```

```
        if self.has_object:
            self.height_offset = self.sim.data.get_site_xpos('object0')[2]
```

PICK AND PLACE.py

```
import os
from gym import utils
from gym.envs.robotics import fetch_env
MODEL_XML_PATH = os.path.join('fetch',
'pick_and_place.xml')
class FetchPickAndPlaceEnv(fetch_env.FetchEnv,
utils.EzPickle):
    def__init_(self, reward_type='sparse'):
        initial_qpos = {
            'robot0:slide0': 0.405,
            'robot0:slide1': 0.48,
            'robot0:slide2': 0.0,
            'object0:joint': [1.25, 0.53, 0.4, 1., 0., 0., 0.],
        }
        fetch_env.FetchEnv.__init__(
```

```
            self, MODEL_XML_PATH, has_object=True,
block_gripper=False, n_substeps=20,
            gripper_extra_height=0.2, target_in_the_air=True,
target_offset=0.0,
            obj_range=0.15, target_range=0.15,
distance_threshold=0.05,
            initial_qpos=initial_qpos,
reward_type=reward_type)
        utils.EzPickle.__init__(self)
```

REACH.py

```
import os
from gym import utils
from gym.envs.robotics import fetch_env
MODEL_XML_PATH = os.path.join('fetch', 'reach.xml')
class FetchReachEnv(fetch_env.FetchEnv, utils.EzPickle):
    def __init__(self, reward_type='sparse'):
        initial_qpos = {
            'robot0:slide0': 0.4049,
            'robot0:slide1': 0.48,
            'robot0:slide2': 0.0,
```

```
        }
        fetch_env.FetchEnv.__init__(
            self, MODEL_XML_PATH, has_object=False,
block_gripper=True, n_substeps=20,
            gripper_extra_height=0.2, target_in_the_air=True,
target_offset=0.0,
            obj_range=0.15, target_range=0.15,
distance_threshold=0.05,
            initial_qpos=initial_qpos,
reward_type=reward_type)
        utils.EzPickle.__init__(self)
```

SLIDE.py

```
import os
import numpy as np
from gym import utils
from gym.envs.robotics import fetch_env
MODEL_XML_PATH = os.path.join('fetch', 'slide.xml')
class FetchSlideEnv(fetch_env.FetchEnv, utils.EzPickle):
    def __init_(self, reward_type='sparse'):
        initial_qpos = {
```

```
            'robot0:slide0': 0.05,
            'robot0:slide1': 0.48,
            'robot0:slide2': 0.0,
            'object0:joint': [1.7, 1.1, 0.4, 1., 0., 0., 0.],
        }
        fetch_env.FetchEnv.__init__(
            self, MODEL_XML_PATH, has_object=True,
block_gripper=True, n_substeps=20,
            gripper_extra_height=-0.02,
target_in_the_air=False, target_offset=np.array([0.4, 0.0, 0.0]),
            obj_range=0.1, target_range=0.3,
distance_threshold=0.05,
            initial_qpos=initial_qpos,
reward_type=reward_type)
        utils.EzPickle.__init__(self)
```

PUSH.py

```
import os
from gym import utils
from gym.envs.robotics import fetch_env
MODEL_XML_PATH = os.path.join('fetch', 'push.xml')
class FetchPushEnv(fetch_env.FetchEnv, utils.EzPickle):
```

```
    def __init__(self, reward_type='sparse'):
        initial_qpos = {
            'robot0:slide0': 0.405,
            'robot0:slide1': 0.48,
            'robot0:slide2': 0.0,
            'object0:joint': [1.25, 0.53, 0.4, 1., 0., 0., 0.],
        }
        fetch_env.FetchEnv.__init__(
            self, MODEL_XML_PATH, has_object=True,
block_gripper=True, n_substeps=20,
            gripper_extra_height=0.0, target_in_the_air=False,
target_offset=0.0,
            obj_range=0.15, target_range=0.15,
distance_threshold=0.05,
            initial_qpos=initial_qpos,
reward_type=reward_type)
        utils.EzPickle.__init__(self)
```