



Факультет Систем Управления и Робототехники

Разработка системы сборки датасета для диффузионной модели управления манипулятором

Аннотация – В работе создали установку для сбора датасета для обучения с учителем диффузионной модели управления манипулятором UR5e для сценария с толканием фигурки на плоскости в указанное место по данным с камер и телеметрии с робота.

Ключевые слова: манипулятор; датасет; телеоператор; UR5e

Практику проходил:
студент 2-го курса, 409578
Смирнов Алексей Владимирович

Руководитель:
Кандидат технических наук; доцент
Ведяков Алексей Алексеевич

Введение

Практика проходила в лаборатории при университете ИТМО в период с 17 по 30 июня.

Цель практики — исследование метода диффузионных политик для управления манипулятором и создание установки для сбора датасета для обучения модели.

Для достижения цели были сформулированы следующие задачи на практику:

- Инструктаж обучающегося
- Ознакомление со статьёй
- Настройка окружения
- Разработка системы телеоператора для манипулятора
- Разработка системы сохранения телеметрии

для сборки датасета с промышленного манипулятора UR5e, предназначенного для последующего обучения диффузионной модели.

1. Обзор источников

Рассмотрели статью Chi C. — Diffusion Policy [1]. В ней авторы предлагают использовать диффузионную модель для предсказания точек траектории движения манипулятора.

Изначальной целью проекта было повторить результат авторов, но ввиду ограниченного времени решили сузить цель до: Создать установку для сбора датасета для одного сценария.

В статье рассмотрены разные сценарии: толкание фигурки в указанное место, распределение соуса по пицце и другие. Остановились на сценарии с фигуркой.

Провели анализ датасета авторов: они использовали следующие данные:

- видео с 4-х ракурсов
- текущие скорости и положения суставов манипулятора
- текущие скорости и положение энд-эффектора манипулятора
- целевое положение энд-эффектора манипулятора
- этап сценария внутри эксперимента

2. Настройка окружения

Первой задачей была настройка рабочего окружения, включая симулятор Universal Robots UR Sim, библиотеку ur-rtde для языка Python.

2.1. Установка библиотеки RTDE

На этом этапе столкнулись с проблемой ошибки сборки пакета. Используемый пакет ur-rtde написан на языке C++ и включает интерфейсы для языка python. Решили проблему сменив главную версию системы сборки CMake с 4 на 3.

2.2. Настройка симулятора UR Sim

Производитель манипулятора Universal Robots предлагает использовать симулятор UR5e и распространяет его в виде Docker-образа. Настроили окружение для запуска симулятора при помощи инструмента docker-compose (Листинг 1) :

3. Разработка системы телеоператора для манипулятора

Так как используем метод обучения с учителем, необходимо создать датасет, в котором манипулятор выполняет задачу под управлением человека.

Для этого было два возможных варианта: кинестатика и телеоператор. Первый вариант значительно проще в реализации: человек руками двигает робота, но у него есть значительных недостаток: на видео всегда будет человек, из-за чего модель может обучиться хуже.

Поэтому выбрали второй способ: телеоператор с джойстика. Для реализации этого метода написали модуль программы, который считывает данных с джойстика в реальном времени с большой частотой (Листинг 2).

Проанализировали, какой функционал необходимо вынести на джойстик. В конечном варианте у телеоператора есть следующие возможности:

- Перемещение энд-эффектора по плоскости XY
- изменение скорости движения
- обозначение текущего этапа сценария
- начало и остановка записи сценария

3.1. Реализация управления манипулятором

Отклонения стика (нормализованное, с примененными метрвыми зонами) умножается на множитель скорости и задается как скорость энд-эффектора в плоскости XY:

```
103 left_stick = self.controller.leftStickPos()
104 sp = np.zeros(6)
105 sp[0] = -left_stick[0] * self.velocityMultiplier
106 sp[1] = -left_stick[1] * self.velocityMultiplier
107
108 self.rtde_c.speedL(sp, 1.0)
```

4. Сбор телеметрии

4.1. Данные с сенсоров

Использовали тот же формат, что и авторы статьи: zagf. Для сохранения телеметрии написали отдельный модуль (Листинг 3).

4.2. Видео

Для записи видео использовали несколько IP-камер. Считывали кадры при помощи библиотеки OpenCV. На этом этапе самым сложным было — синхронизация видео по времени (начало и сохранение записей с нескольких камер) одновременно.

Для решения этой проблемы использовали механизм многопоточности: за каждую камеру отвечает отдельный процесс (См. Листинг 4).

5. Основной модуль

Основной модуль программы (Листинг 5) объединяет все ранее упомянутые модули. Итерация основного цикла работы программы выглядит следующим образом:

```
def loop(self, target_time=.05):
    while True:
        dt = time.time()

        self.handleBtns()
        self.handleTelemetry()

        self.statusLine()

        left_stick = self.controller.leftStickPos()
        sp = np.zeros(6)
        sp[0] = -left_stick[0] * self.velocityMultiplier
        sp[1] = -left_stick[1] * self.velocityMultiplier

        self.rtdc_c.speedL(sp, 1.0)

        dt = time.time() - dt
        if dt > target_time:
            print("WARN cycle time too high")
        else:
            sleep_time = max(0, target_time - dt)
            time.sleep(sleep_time)
        self.epTick += 1
```

Заключение

В результате проекта получили рабочую установку для записи датасета на манипуляторе UR5e для сценария с толканием фигурки на плоскости. Исходный код проекта можно найти в онлайн-репозитории (См. Приложение).

Помимо этого, записали демонстрацию работы установки на видео (См. Приложение). В репозитории также можно найти записи с камер и телеметрию, записанные во время демонстрации.

Приложение

Демонстрация работы системы (видео):

https://drive.google.com/file/d/1Lrglu6aEk7I5UbF9WPFpyLqs9BmqRW-/view?usp=share_link

Исходный код проекта:

<https://github.com/Hanqnero/DiffusionPolicy-dataset-setup>

```
services:
  ursim:
    image: universalrobots/ursim_e-series
    container_name: ursim
    ports:
      - "51135:51135/udp" # Controller data
      - "5901:5900" # VNC
```

```

- "6080:6080" # Web VNC (browser GUI)
- "6666:6666"
- "29999:29999"
- "30001:30001" # Primary interface
- "30011:30011" # Primary interface
- "30002:30002" # Secondary interface (used by ur_rtde)
- "30012:30012" # Secondary interface (used by ur_rtde)
- "30003:30003" # Real-time interface
- "30013:30013" # Real-time interface
- "30004:30004" # RTDE interface
- "30020:30020"
- "50001:50001"
- "50002:50002"
- "50003:50003"
- "502:502"
volumes:
- ${HOME}/.ursim/urcaps:/urcaps
- ${HOME}/.ursim/programs:/ursim/programs
- /Users/hanqnero/Dev/SummerPractice:/workspace

```

Листинг 1. docker-compose.yml для образа симулятора

```

import hid
import time
import threading

JOYSTICK_VENDOR_ID = 0x46d
JOYSTICK_PRODUCT_ID = 0xc216

class ControllerHIDAdapter:
    def __init__(self, vid=JOYSTICK_VENDOR_ID, pid=JOYSTICK_PRODUCT_ID):
        self.d = hid.Device(vid, pid)

        self.lsx = 127
        self.lsy = 127
        self.rsx = 127
        self.rsy = 127
        self.btn = 8
        self.btnflags = 0

        self.deadzone = .05

        self.ioThread = None
        self.running = False
        self.lock = threading.Lock()

    def applyDeadzone(self, x, y):
        if x**2 + y**2 < self.deadzone**2:
            return 0, 0
        return x, y

    def setDeadzone(self, dz):
        self.deadzone = dz

# BYTES LAYOUT [64 bytes ber message]
# - LEFT STICK X [1 BYTE] <- Byte 0
# - LEFT STICK Y [1 BYTE] <- Byte 1
# - RIGHT STICK X [1 BYTE] <- Byte 2
# - RIGHT STICK Y [1 BYTE] <- Byte 3
# ...

```

```

def _ioLoop(self):
    # TODO: flush buffer
    while self.running:
        data = self.d.read(72)

        with self.lock:
            self.lsx = data[0]
            self.lsy = data[1]
            self.rsx = data[2]
            self.rsy = data[3]
            self.btn = data[4]
        self.btnflags = self.btn & 0b11110000 | self.btnflags

def lowerFlag(self, btn_n):
    if self.btnflags & (1 << btn_n):
        self.btnflags -= (1 << btn_n)

def leftStickPos(self):
    x, y = 0, 0
    with self.lock:
        x, y = self.lsx, self.lsy
    x = (x - 128) / 127
    y = -(y - 128) / 127

    x = min(max(x, -1.0), 1.0)
    y = min(max(y, -1.0), 1.0)

    x, y = self.applyDeadzone(x, y)

    return (x, y)

def rightStickPos(self):
    x, y = 0, 0
    with self.lock:
        x, y = self.rsx, self.rsy
    x = (x - 128) / 127
    y = -(y - 128) / 127

    x = min(max(x, -1), 1)
    y = min(max(y, -1), 1)

    x, y = self.applyDeadzone(x, y)

    return (x, y)

def btnState(self):
    with self.lock:
        btn = self.btn

    btn = (btn & 0b11110000) >> 4
    return [btn & (1 << i) != 0 for i in range(4)]

def createIoThread(self):
    thread = threading.Thread(target=self._ioLoop)
    return thread

def flush(self):
    self.d.nonblocking = True

    while True:

```

```

        data = self.d.read(64)
        if not data:
            break # Buffer is empty

    self.d.nonblocking = False

def start(self):
    self.flush()
    self.ioThread = self.createIoThread()
    self.running = True
    self.ioThread.start()

def stop(self):
    # Set sticks data to zero

    self.lsx = 127
    self.lsy = 127
    self.rsx = 127
    self.rsy = 127
    self.btn = 8

    self.running = False
    if self.ioThread:
        self.ioThread.join()
    self.d.close()

def waitForFirstData(self):
    self.d.read(64)

if __name__ == '__main__':
    c = ControllerHIDAdapter()
    time.sleep(1)
    c.start()
    try:
        while True:
            print( c.leftStickPos(), end='\t')
            print( c.rightStickPos(), end='\t')
            print( c.btnState(), end='\n')
            time.sleep(.04)
    except KeyboardInterrupt:
        c.stop()

```

Листинг 2. Модуль для чтения данных с джойстика

```

import zarr
import numpy as np
import os
import shutil
import time

DATA_ARRAY_SPECS = {
    'action': ((0, 6), 'float32'), # getTargetT
    'robot_eef_pose': ((0, 6), 'float32'), # getActualT
    'robot_eef_pose_vel': ((0, 6), 'float32'), # getActualTCPSpeed
    'robot_joint': ((0, 6), 'float32'), # getActualQ
    'robot_joint_vel': ((0, 6), 'float32'), # getActualQd
    'stage': ((0,), 'int8'),
    'timestamp': ((0,), 'int64')
}

```

```

ZARR_PATH = "new_replay_buffer.zarr"

class RealTimeZarrWriter:
    def __init__(self, zarr_path=ZARR_PATH, data_specs=DATA_ARRAY_SPECS,
overwrite=False):
        self.zarr_path = zarr_path

        if overwrite and os.path.exists(zarr_path):
            shutil.rmtree(zarr_path)

        self.root = zarr.open(zarr_path, mode='a') # Use append mode

        # Create main groups if they don't exist
        self.data_group = self.root.require_group('data')
        self.meta_group = self.root.require_group('meta')

        # Create data arrays based on the provided specifications
        for name, (shape, dtype) in data_specs.items():
            if name not in self.data_group:
                chunks = (100,) + shape[1:] if len(shape) > 1 else (100,)
                self.data_group.create_array(name, shape=shape, chunks=chunks,
dtype=dtype)

            # Create metadata arrays
            if 'episode_ends' not in self.meta_group:
                self.meta_group.create_array('episode_ends', shape=(0,), chunks=(100,),
dtype='i8')

        def append_data(self, timestep_data):
            for name, data in timestep_data.items():
                if name in self.data_group:
                    # print(f"Appending to `{name}` data {data}")
                    if isinstance(data, list):
                        data = np.array(data).reshape((1,6))
                    self.data_group[name].append(data)
                else:
                    print(f"Warning: Array '{name}' not found in data specifications.")

        def end_episode(self):
            """
            Marks the end of an episode by recording the current number of data points.
            """
            # Use the 'action' array's length as the reference for the total number
of timesteps
            current_len = self.data_group['action'].shape[0]
            self.meta_group['episode_ends'].append([current_len])

if __name__ == '__main__':
    ZARR_PATH = "new_replay_buffer_oop.zarr"

    # --- Example Usage ---
    print(f"Creating and populating Zarr dataset at: {ZARR_PATH}")
    writer = RealTimeZarrWriter(ZARR_PATH, DATA_ARRAY_SPECS, overwrite=False)

    print("Initial structure:")
    writer.root.tree()

    # Simulate logging two episodes in real-time
    for episode_idx in range(2):
        print(f"\n--- Logging Episode {episode_idx + 1} ---")
        episode_length = 50 + episode_idx * 10 # Make episodes different lengths

```



```

        for i in range(episode_length):
            # In a real application, this data would come from your robot
            sensors/controller
            timestep_data = {
                'action': np.random.rand(1, 6),
                'robot_eef_pose': np.random.rand(1, 6),
                'robot_eef_pose_vel': np.random.rand(1, 6),
                'robot_joint': np.random.rand(1, 6),
                'robot_joint_vel': np.random.rand(1, 6),
                'stage': np.array([episode_idx]),
                'timestamp': np.array([time.time()])
            }
            writer.append_data(timestep_data)

        # Mark the end of the episode
        writer.end_episode()
        print(f"Episode {episode_idx + 1} finished and marked.")

    print("\n--- Finished Logging ---")
    print("\nFinal dataset structure:")
    writer.root.tree()

    # Verification
    print("\nVerification:")
    final_episode_ends = list(writer.root['meta/episode_ends'])
    print(f"Episode end markers: {final_episode_ends}")
    assert final_episode_ends[0] == 50
    assert final_episode_ends[1] == 110
    print("Verification successful!")
    writer.root.tree()

```

Листинг 3. Модуль для сохранения телеметрии с сенсоров робота

```

import cv2
import os
import threading
import time
from pathlib import Path

class CameraThread(threading.Thread):
    def __init__(self, cam_id, cam_src, output_path, target_fps, target_resolution):
        super().__init__()

        self.cam_id = cam_id
        self.cam_src = cam_src
        self.output_path = output_path
        self.fps = target_fps
        self.resolution = target_resolution # (width, height)

        self.running = threading.Event()

        self.cap = None
        self.writer = None

        print(f"[Camera {self.cam_id}] Initializing camera thread.")
        self.cap = cv2.VideoCapture(self.cam_src)

        if not self.cap.isOpened():
            print(f"[Camera {self.cam_id}] Failed to open source: {self.cam_src}")

        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out_file = os.path.join(self.output_path, f"{self.cam_id}.mp4")
        self.writer = cv2.VideoWriter(out_file, fourcc, self.fps, self.resolution)

```

```

print(f"[Camera {self.cam_id}] Recording started.")
self.running.set()

def run(self):
    try:
        while self.running.is_set():
            ret, frame = self.cap.read()

            if not ret:
                print(f"[Camera {self.cam_id}] Failed to read frame.")
                break

            resized = cv2.resize(frame, self.resolution)
            self.writer.write(resized)
    except Exception as e:
        print(f"[Camera {self.cam_id}] Exception occurred: {e}")
    finally:
        self.cleanup()
        print(f"[Camera {self.cam_id}] Recording stopped and saved.")

def stop(self):
    self.running.clear()

def cleanup(self):
    if self.cap:
        self.cap.release()
    if self.writer:
        self.writer.release()

class CameraManager:
    def __init__(self, root_dir="recordings", target_fps=30,
target_resolution=(640, 480)):
        self.root_dir = Path(root_dir)
        self.cameras = {}

        self.recording_id = self._get_init_recording_id()

        self.threads = []
        self.target_fps = target_fps
        self.target_resolution = target_resolution # (width, height)

    def _get_init_recording_id(self):
        if not self.root_dir.exists():
            return 0
        recordings = [d for d in self.root_dir.iterdir() if d.is_dir()]
        if not recordings:
            return 0
        return len(recordings)

    def add_camera(self, cam_src):
        cam_id = len(self.cameras)
        self.cameras[cam_id] = cam_src
        print(f"[Manager] Added camera {cam_id}: {cam_src}")

    def _prepare_recording_folder(self):
        session_dir = self.root_dir / str(self.recording_id)
        session_dir.mkdir(parents=True, exist_ok=True)
        return session_dir

```

```

def start_recording(self):
    if not self.cameras:
        print("[Manager] No cameras to record.")
        return

    session_path = self._prepare_recording_folder()
    self.threads.clear()

    for cam_id, cam_src in self.cameras.items():
        thread = CameraThread(cam_id, cam_src, session_path, self.target_fps,
self.target_resolution)
        self.threads.append(thread)

    for thread in self.threads:
        thread.start()
        print(f"[Manager] Camera {thread.cam_id} thread started.")

    for thread in self.threads:
        if not thread.running.wait(timeout=5):
            print(f"[Manager] Camera {thread.cam_id} failed to start recording
in time. Skipping.")

    print(f"[Manager] Recording session {self.recording_id} started.")

def stop_recording(self):
    print(f"[Manager] Stopping all camera threads.")
    for thread in self.threads:
        thread.stop()
        print(f"[Manager] Camera {thread.cam_id} thread stopped.")
    print(f"[Manager] Recording session {self.recording_id} finished.")
    self.recording_id += 1

def set_root_dir(self, new_root):
    self.root_dir = Path(new_root)
    print(f"[Manager] Root directory set to: {self.root_dir.resolve()}")

if __name__ == "__main__":
    cam_manager = CameraManager(root_dir="recordings", target_fps=30,
target_resolution=(640, 480))

    cam_manager.add_camera("rtsp://root:admin@192.168.86.37/axis-media/media.amp")
    cam_manager.add_camera("rtsp://root:admin@192.168.86.39/axis-media/media.amp")
    # cam_manager.add_camera("rtsp://root:admin@192.186.86.40/axis-media/media.
amp")

    print("Starting recording...")
    cam_manager.start_recording()

    print("Stopping recording...")
    cam_manager.stop_recording()

    print("Starting recording...")
    cam_manager.start_recording()

    print("Stopping recording...")
    cam_manager.stop_recording()

```

Листинг 4. Модуль для сохранения видео с камер

```

import numpy as np
import time

import rtde_control

```

```

import rtde_receive

import controller
import zarr_logger
import camera

class ControllerTeleop:
    def __init__(self, IP):
        print('Connecting to robot at', IP)
        self.rtde_c = rtde_control.RTDEControlInterface(IP)
        self.rtde_r = rtde_receive.RTDEReceiveInterface(IP)

        self.velocityMultiplier = 0.1 # speed in meters per second with full
stick deflection
        self.velocityChangePerTick = 0.0005

        self.controller = controller.ControllerHIDAdapter()
        self.controller.start()

        self.epTick = 0
        self.zarrWriter = zarr_logger.RealTimeZarrWriter(overwrite=False)

        self.stageBtnFlag = False
        self.telemetryStage = 0

        self.cameraManager = camera.CameraManager(
            'recordings', target_fps=30, target_resolution=(640, 480))
        self.recording_running = False

        print('init finished')

    def stop(self):
        self.rtde_c.stopL()

        self.rtde_c.disconnect()
        self.rtde_r.disconnect()
        self.controller.stop()
        self.cameraManager.stop_recording()

    def statusLine(self):
        print(f"Stage: {self.telemetryStage}, Velocity: {
            self.velocityMultiplier:.4f}, Tick: {self.epTick:6d} ", end='\r')

    def handleBtns(self):
        if not self.controller.btnState()[0]:
            self.stageBtnFlag = False
        if not self.controller.btnState()[3]:
            self.endEpFlag = False

        if self.controller.btnState()[0]:
            if not self.stageBtnFlag:
                self.stageBtnFlag = True
                self.telemetryStage = (self.telemetryStage + 1) % 4

        elif self.controller.btnState()[1]:
            self.velocityMultiplier += self.velocityChangePerTick

        elif self.controller.btnState()[2]:
            self.velocityMultiplier -= self.velocityChangePerTick

        elif self.controller.btnState()[3]:
            if not self.endEpFlag:
                self.endEpFlag = True
                self.telemetryStage = 0

```

```

        self.epTick = 0
        self.zarrWriter.end_episode()

        if self.recording_running:
            self.cameraManager.stop_recording()
            self.recording_running = False
        else:
            self.cameraManager.start_recording()
            self.recording_running = True

    def handleTelemetry(self):
        telemetry = {
            'action': self.rtdc_r.getTargetTCPPOSE(),
            'robot_eef_pose': self.rtdc_r.getActualTCPPOSE(),
            'robot_eef_pose_vel': self.rtdc_r.getActualTCPSPD(),
            'robot_joint': self.rtdc_r.getActualQ(),
            'robot_joint_vel': self.rtdc_r.getActualQd(),
            'stage': np.array([self.telemetryStage]),
            'timestamp': np.array([time.time()])
        }

        if self.recording_running:
            self.zarrWriter.append_data(telemetry)

    def loop(self, target_time=.05):
        while True:
            dt = time.time()

            self.handleBttns()
            self.handleTelemetry()

            self.statusLine()

            left_stick = self.controller.leftStickPos()
            sp = np.zeros(6)
            sp[0] = -left_stick[0] * self.velocityMultiplier
            sp[1] = -left_stick[1] * self.velocityMultiplier

            self.rtdc_c.speedL(sp, 1.0)

            dt = time.time() - dt
            if dt > target_time:
                print("WARN cycle time too high")
            else:
                sleep_time = max(0, target_time - dt)
                time.sleep(sleep_time)
            self.epTick += 1

if __name__ == '__main__':
    ROBOT_IP = '192.168.86.5'
    # ROBOT_IP = 'localhost'
    t = ControllerTeleop(ROBOT_IP)

    t.cameraManager.add_camera(
        "rtsp://root:admin@192.168.86.37/axis-media/media.amp")
    t.cameraManager.add_camera(
        "rtsp://root:admin@192.168.86.39/axis-media/media.amp")

    try:
        t.loop()
    except KeyboardInterrupt:
        ...

```

```
finally:  
    t.stop()
```

Листинг 5. Основной модуль программы

Библиография

1. Chi C. и др. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion // The International Journal of Robotics Research. 2024.
2. Universal Robots RTDE C++ Interface [электронный ресурс]. 2025. URL: https://sdurobotics.gitlab.io/ur_rtde/.