Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

Работа выполнена в СКБ «Промышленная робототехника»

СОГЛАСОВАНО

Начальник отдела ОНиПКРС

_____ Е.М. Димитриади
(подпись)
«10» 06 2024 г.

Декан ФЭУ

_____ А.С. Гудим
(подпись)
«10» 06 2024 г.

УТВЕРЖДАЮ

Проректор по научной работе,
д-р техн. наук. профессор

_____ А.В. Космынин
(подпись)
«10» 06 2024 г.

«Программа распознавания положения рук оператора коллаборативного робота для реализации следящей системы управления»

Комплект документации на управляющую программу для автоматизированной/роботизированной системы

Руководитель СКБ _____ 10.06.2024 С.И. Сухоруков
(подпись, дата)

Руководитель проекта _____ 10.06.24 *Ю.С. Иванов*
(подпись, дата)

Комсомольск-на-Амуре 2024

Карточка проекта

Название	Программа распознавания положения рук оператора коллаборативного робота для реализации следящей системы управления
Тип проекта	Тип проекта: науно-исследовательский проект (с дальнейшей публикацией РИНЦ, ВАК и т.д), другое
Исполнители	Студент ФЭУ _____ Д.М. Грабарь– 2УИм-1 Студент ФЭУ _____ М.А. Лямин– 2АУм-1
Срок реализации	02.2024-05.2024

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

ЗАДАНИЕ
на разработку

Название проекта: Программа распознавания положения рук оператора коллаборативного робота для реализации следящей системы управления

Назначение: Демонстрация возможностей коллаборативного роботизированного комплекса в рамках проводимых экскурсий и выставок

Область использования: Экскурсии и выставки, проводимые с целью профессиональной ориентационной работы в ФГБОУ ВО «КнАГУ»

Функциональное описание проекта: Получения информации с нескольких камер с дальнейшим распознаванием ключевых точек на руке оператора и построении трехмерной сцены на основе агрегированных данных

Техническое описание программы: Для работы системы необходим один вычислительный модуль (ЭВМ)

Требования: Вычислительное устройство под управлением Linux Ubuntu 20.04 или выше, предустановленный интерпретатор python3.6 или выше

План работ:

Наименование работ	Срок
Сбор, обработка данных с последующим обучением нейросетевых алгоритмов	02.2024
Проектирование системы обмена информации между отдельными программными модулями	03.2024
Разработка блока построения трехмерной сцены на базе полученных ключевых точек	04.2024
Тестирование полученных результатов и оформление отчета	05.2024

Комментарии:

Перечень графического материала:

1. Листинг управляющей программы
2. Результаты работы программы.

Руководитель проекта 10.06.24 Ю.С. Иванов

(подпись, дата)

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

ПАСПОРТ
Управляющей программы для автоматизированной/роботизированной системы

«Программа распознавания положения рук оператора коллаборативного робота для реализации следящей системы управления»

Руководитель проекта _____ Ю.С. Иванов
(подпись, дата)

Исполнители проекта _____ 10.06.24 Д.М. Грабарь
(подпись, дата)

_____ 10.06.2024 М.А. Лямин
(подпись, дата)

Комсомольск-на-Амуре 2024

Содержание

1 Общие положения

Настоящий паспорт является документом, предназначенным для ознакомления с основной структурой, особенностями и правилами эксплуатации управляющей программы «распознавание положения рук оператора коллаборативного робота для реализации следящей системы управления» (далее «программа»).

Паспорт входит в комплект поставки программы. Перед запуском программы внимательно изучите правила ее эксплуатации.

1.1 Наименование программы

Полное наименование программы – «распознавание положения рук оператора коллаборативного робота для реали-зации следящей системы управления».

1.2 Наименования документов, на основании которых ведется проектирование системы

Создание программы «распознавание положения рук оператора коллаборативного робота для реализации следящей системы управления» осуществляется на основании требований и положений следующих документов:

- задание на разработку.

1.3 Перечень организаций, участвующих в разработке системы

Заказчиком создания программы «распознавание положения рук оператора коллаборативного робота для реализации следящей системы управления» является Федеральное государственное бюджетное образовательное учреждение высшего образования «Комсомольский-на-Амуре государственный университет» (далее заказчик), находящийся по адресу: 681013, Хабаровский край, г. Комсомольск-на-Амуре, Ленина пр-кт., д. 27.

Исполнителями работ по созданию программы «распознавание положения рук оператора коллаборативного робота для реализации следящей

системы управления» являются Конструкторы студенческого конструкторского бюро (далее СКБ), студенты групп 2УИм-1, Грабарь Данниил Михайлович и 2АУм-1 Лямин Михаил Андреевич.

1.4 Сведения об использованных при проектировании нормативно-технических документах

При проектировании использованы следующие нормативно-технические документы:

ГОСТ 19.001-77. Единая система программной документации (ЕСПД). Общие положения.

ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения

ГОСТ 19.101-77. ЕСПД. Виды программ и программных документов.

ГОСТ 19.102-77. ЕСПД. Стадии разработки.

ГОСТ 19.401-78. ЕСПД. Текст программы. Требования к содержанию и оформлению.

ГОСТ 19.402-78. ЕСПД. Описание программы.

ГОСТ 19.404-79. ЕСПД. Пояснительная записка. Требования к содержанию и оформлению.

ГОСТ 19.505-79. ЕСПД. Руководство оператора. Требования к содержанию и оформлению.

2 Описание программы

2.1 Общие сведения

Существует коллаборативный роботизированный комплекс с интелектуальной системой жестового управления. В комплексе реализована система управления, осуществляющая распознование положения руки человека-оператора в пространсве, формирование управляющих команд для робота, передачи этих команд в контроллер коллаборативного робота. Комплекс построен на базе коллаборативного робота, внешнего устройства управленяи для обработки нейросетевых алгоритмов, комплекта web-камер и захвата установленного на фланце робота.

2.2 Функциональное назначение программы

Программа применяется для распознования ключевых точек руки человека-оператора с последующим построенимии трехмерной сцены.

2.3 Описание логической структуры

Система выполняет две основные задачи: распознование ключевых точек руки человека-оператора коллаборативного роботизированного процесса; построение трехмерной сцены для отслеживания положения руки оператора относительно рабочего пространсва робота. Задачи выполняются последовательно, сначало происходит распознование и на основе полученных данных происходит построение трехмерной сцены.

Текст программы приведен в Приложении А.

2.4 Используемые технические средства

Вычислительный модуль под управлением Ubuntu 22.04 -1 штука

WEB-камеры Logitech V-U0028 – 2 штуки

2.5 Вызов и загрузка

Программа предустановлена на вычислетельныом устройстве, для ее запуска необходимо в случае необходимости откоретировать файл config.yaml. С помощью команды python3 MultiprocessingHandDetection.py запустить основную программу.

3 Руководство оператора

3.1 Назначение программы;

Результат выполнения программы приведен в Приложении Б.

3.2 Условия выполнения программы;

Web-камера должна быть выставлена на штативе и отклиброванa шаблоном в виде шахмотной доски

3.3 Выполнение программы;

Программа заупскается через консоль операционной системы Linux Ubuntu 22.04. Программа работает в автоматическом режиме. Выход из программы производиться по нажатию клавиши «Q».

3.4 Сообщения оператору

Система вывод графическую информацию о расположении руки человека-оператора в двухмерном пространсве (непосредсвенно на изображении), а также в трехмерном пространсве.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг 1 – Конфигурационный файл системы

```
detection:
  min_detection_confidence: 0.8
  min_tracking_confidence: 0.5
  max_num_hands: 1 # Максимальное колличество распозноваемых рук

camera:
  ids: [0, 1] # Список камер, задается обязательно в квадратных скобках, через запятую.
Например [0, 1, 2],
  mode: "stream" # Доступные режимы: "stream", "video", "frame"
  width: 640 # Ширина
  height: 480 # Высота

fingers:
  wrist: [0]
  thumb: [4, 1]
  index: [8, 5]
  middle: [12, 9]
  ring: [16, 13]
  pinky: [20, 17]

calibration:
  img_count: 10
  time_out: 3
  mono_calibration_path: "./MonoCalibrationImages"
  stereo_calibration_path: "./StereoCalibrationImages"
  show_gui: 1
  rows: 6
  columns: 9
  type_calib: "chess"
  save: 0

reconstruction:
  inside_camera_parameters_path: "./InsideCameraParameters"
  thumb_connections: [[0, 1], [1, 2], [2, 3], [3, 4]]
  index_connections: [[0, 5], [5, 6], [6, 7], [7, 8]]
  middle_connections: [[0, 9], [9, 10], [10, 11], [11, 12]]
  ring_connections: [[0, 13], [13, 14], [14, 15], [15, 16]]
  pinkie_connections: [[0, 17], [17, 18], [18, 19], [19, 20]]
  fingers_colors: ['red', 'blue', 'green', 'black', 'orange']
  world_image: 1

arduino_arm:
  com: "COM6"
  baud: 1000000

```

Листинг 2 – Монокалибровка камеры

```
# python mono_calibration.py --stream-off - Запуск каллибровки без запуска камеры
# python mono_calibration.py - Запуск калибровки с запуском камеры

import os
```

```
import shutil
import argparse

from Camera import Camera
from config import CalibrationConfig, ReconstructionsConfig, CameraConfig
from Calibration import mono_calibration_camera, save_json

parser = argparse.ArgumentParser(description="Test file")
parser.add_argument("-so", "--stream-off", dest="stream",
                    type=bool, default=True, const=False, nargs="?", help="Using the cam-
era (default True)")

arguments = parser.parse_args()

if arguments.stream:
    try:
        shutil.rmtree(CalibrationConfig.mono_calibration_path)
    except FileNotFoundError:
        pass

    try:
        os.mkdir(CalibrationConfig.mono_calibration_path)
        os.mkdir(ReconstructionsConfig.inside_camera_parameters_path)
    except FileExistsError:
        pass


for camera_id in CameraConfig.ids:

    if arguments.stream:
        camera = Camera(device_id=camera_id,
                        mode="frame",
                        width=CameraConfig.width,
                        height=CameraConfig.height)

        camera.stream(img_count=CalibrationConfig.img_count,
                      time_out=CalibrationConfig.time_out,
                      path=CalibrationConfig.mono_calibration_path,
                      show_gui=CalibrationConfig.show_gui)

    ret, mtx, dist, rvecs, tvecs = mono_calibration_camera(
        path=f"{CalibrationConfig.mono_calibration_path}/Camera_{camera_id}_frame/",
        rows=CalibrationConfig.rows,
        columns=CalibrationConfig.columns,
        save=CalibrationConfig.save
    )

    rvecs = tuple(val.tolist() for val in rvecs)
    tvecs = tuple(val.tolist() for val in tvecs)

    inside_camera_params = {
        "RMSE": ret,
        "matrix": mtx.tolist(),
        "dist": dist.tolist(),
        "rotVecs": rvecs,
        "tvecs": tvecs
    }

    save_json(data=inside_camera_params,
              path=ReconstructionsConfig.inside_camera_parameters_path,
              name_file=f"camera_{camera_id}")


print("[!] Монокалибровка камер прошла успешно")

```

Листинг 3 – Стереокалибровка камеры

```
import os
import shutil
import argparse
import numpy as np

from Camera import Camera
from config import CalibrationConfig, ReconstructionsConfig, CameraConfig
from Calibration import stereo_camera_calibration, save_json, read_json

from multiprocessing import Process, Barrier

parser = argparse.ArgumentParser(description="Test file")
parser.add_argument("-so", "--stream-off", dest="stream",
                    type=bool, default=True, const=False, nargs="?", help="Using the cam-
era (default True)")

arguments = parser.parse_args()

if arguments.stream:
    try:
        shutil.rmtree(CalibrationConfig.stereo_calibration_path)
    except FileNotFoundError:
        pass

    try:
        os.mkdir(CalibrationConfig.stereo_calibration_path)
        os.mkdir(ReconstructionsConfig.inside_camera_parameters_path)
    except FileExistsError:
        pass


def main():

    barrier = Barrier(len(CameraConfig.ids))

    process = []

    for camera_id in CameraConfig.ids:

        if arguments.stream:
            camera = Camera(device_id=camera_id,
                            mode="frame",
                            width=CameraConfig.width,
                            height=CameraConfig.height)

            kwargs_cameras = {
                "img_count": CalibrationConfig.img_count,
                "time_out": CalibrationConfig.time_out,
                "path": CalibrationConfig.stereo_calibration_path,
                "show_gui": CalibrationConfig.show_gui,
                "barrier": barrier,
            }

            mp_process = Process(target=camera.stream, kwargs=kwargs_cameras,
name=f"Camera_{camera_id}")
            process.append(mp_process)

            mp_process.start()

    if arguments.stream:
        process[0].join()

```

```
    inside_camera_params_left = read_json(f"{ReconstructionsConfig.inside_camera_parameters_path}/camera_{CameraConfig.ids[0]}.json")
    inside_camera_params_right = read_json(f"{ReconstructionsConfig.inside_camera_parameters_path}/camera_{CameraConfig.ids[1]}.json")

    R, T = stereo_camera_calibration(matrix1=np.array(inside_camera_params_left["matrix"]),
                                     matrix2=np.array(inside_camera_params_right["matrix"]),
                                     dist1=np.array(inside_camera_params_left["dist"]),
                                     dist2=np.array(inside_camera_params_right["dist"]),
frames_folder1=f"{CalibrationConfig.stereo_calibration_path}/Camera_{CameraConfig.ids[0]}_frame",
frames_folder2=f"{CalibrationConfig.stereo_calibration_path}/Camera_{CameraConfig.ids[1]}_frame",
                                     save=CalibrationConfig.save)

    inside_stereo_params = {
        "camera_left": {
            "RMSE": inside_camera_params_left["RMSE"],
            "matrix": inside_camera_params_left["matrix"],
            "dist": inside_camera_params_left["dist"],
            "rotVecs": inside_camera_params_left["rotVecs"],
            "tvecs": inside_camera_params_left["tvecs"]
        },

        "camera_right": {
            "RMSE": inside_camera_params_right["RMSE"],
            "matrix": inside_camera_params_right["matrix"],
            "dist": inside_camera_params_right["dist"],
            "rotVecs": inside_camera_params_right["rotVecs"],
            "tvecs": inside_camera_params_right["tvecs"]
        },

        "R": R.tolist(),
        "T": T.tolist()

    }

    save_json(data=inside_stereo_params,
              path=ReconstructionsConfig.inside_camera_parameters_path,
              name_file=f"stereo_params")


if __name__ == "__main__":
    main()
```

Листинг 4 – Основной текст программы

```
import cv2
import time
import serial
import threading
import numpy as np
import mediapipe as mp
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D
from multiprocessing import Process, Barrier, Pipe
```

```
12.
13. from Camera import Camera
14. from Calibration import read_json
15. from HandRecognition import get_frame_keypoints, draw_landmarks, calculate_angle
16. from config import FingersID, CameraConfig, DetectionConfig, ReconstructionsConfig, Ardu-
inoArm
17.
18.
19. def __prediction(image: np.ndarray,
20.                    model: mp.solutions.hands.Hands,
21.                    fingers_info: dict,
22.                    ) -> tuple[np.ndarray, dict, list[tuple[int, int]]]:
23.     """
24.     Функция обертка для обобщения процесса детекции объекта
25.     :param image: Захваченный кадр
26.     :param model: Модель распознования
27.     :param fingers_info: Информация о пальцах
28.     :return: Захваченный кадр и информацию о пальцах
29.     """
30.
31.     rgb_frame = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
32.
33.     rgb_frame.flags.writeable = False
34.     prediction = model.process(rgb_frame)
35.     rgb_frame.flags.writeable = True
36.
37.     hand_landmarks = prediction.multi_hand_landmarks
38.
39.     key_points = [(-1, -1) for _ in range(len(mp.solutions.hands.HAND_CONNECTIONS))]
40.
41.     if hand_landmarks:
42.         # hand_label = prediction.multi_handedness[0].classification[0].label
43.         key_points = get_frame_keypoints(landmarks=hand_landmarks,
44.                                          img=image,
45.
count_points=len(mp.solutions.hands.HAND_CONNECTIONS))
46.
47.         image = draw_landmarks(image, key_points, mp.solutions.hands.HAND_CONNECTIONS)
48.
49.         for finger in fingers_info:
50.             angle = calculate_angle(
51.                 a=key_points[fingers_info[finger]["id"][0]],
52.                 b=key_points[fingers_info[finger]["id"][1]],
53.                 c=key_points[FingersID.wrist[0]]
54.             )
55.
56.             fingers_info[finger]["angle"] = angle
57.             fingers_info[finger]["close_finger"] = False
58.
59.             if angle <= 90:
60.                 fingers_info[finger]["close_finger"] = True
61.
62.             if finger == "Thumb" and angle <= 135:
63.                 fingers_info[finger]["close_finger"] = True
64.
65.             cv2.putText(image, str(round(angle, 2)),
66.                         key_points[fingers_info[finger]["id"][0]],
67.                         cv2.FONT_HERSHEY_PLAIN,
68.                         1,
69.                         (255, 0, 255))
70.
71.         for finger, i in zip(fingers_info, range(1, len(fingers_info) + 1)):
72.             cv2.putText(image,
73.                         f"{finger}: {fingers_info[finger]['close_finger']},
{round(fingers_info[finger]['angle'], 2)}",
74.                         (20, 20 * i),
75.                         cv2.FONT_HERSHEY_PLAIN,
```

```
                        1,
                        (255, 0, 0))

    return image, fingers_info, key_points


def DLT(P1: np.ndarray, P2: np.ndarray,
        point1: list, point2: list) -> np.ndarray:

    """
    Прямые линейные преобразования
    :param P1: Матрица проекций 1
    :param P2: Матрица проекций 2
    :param point1: Ключевая точка камеры 1
    :param point2: Ключевая точка камеры 2
    :return: вектор 3D точек
    """

    A = [point1[1] * P1[2, :] - P1[1, :],
         P1[0, :] - point1[0] * P1[2, :],
         point2[1] * P2[2, :] - P2[1, :],
         P2[0, :] - point2[0] * P2[2, :]
         ]

    A = np.array(A).reshape((4, 4))

    B = A.transpose() @ A
    U, s, Vh = np.linalg.svd(B, full_matrices=False)

    return Vh[3, 0:3] / Vh[3, 3]


def visualize_3d(barrier: Barrier,
                 receiver: Pipe):

    """
    Функция визуализации триангулируемых данных
    :param barrier: Барьер для ожидания других потоков
    :param receiver: Получение данных с потока
    :return:
    """

    barrier.wait()

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.elev = 35
    ax.azim = 70

    # Примените координатные повороты к точке по оси z как вверх
    Rz = np.array(([[0., -1., 0.],
                    [1., 0., 0.],
                    [0., 0., 1.]]))

    Rx = np.array(([[1., 0., 0.],
                    [0., -1., 0.],
                    [0., 0., -1.]]))

    # Информация по правильности построения пальцев представлена в config.yaml
    fingers = [ReconstructionsConfig.pinkie_connections,
               ReconstructionsConfig.ring_connections,
               ReconstructionsConfig.middle_connections,
               ReconstructionsConfig.index_connections,
               ReconstructionsConfig.thumb_connections]

    fingers_colors = ReconstructionsConfig.fingers_colors

```

```
    while True:
        key_points = receiver.recv()

        p3ds_rotated = []

        # Поворо точек относительно координант
        for kpt in key_points:
            kpt_rotated = Rz @ Rx @ kpt
            p3ds_rotated.append(kpt_rotated)

        p3ds_rotated = np.array(p3ds_rotated)

        for finger, finger_color in zip(fingers, fingers_colors):
            for _c in finger:
                ax.plot(xs=[p3ds_rotated[_c[0], 0], p3ds_rotated[_c[1], 0]],
                        ys=[p3ds_rotated[_c[0], 1], p3ds_rotated[_c[1], 1]],
                        zs=[p3ds_rotated[_c[0], 2], p3ds_rotated[_c[1], 2]],
                        linewidth=4, c=finger_color)

        # draw axes
        ax.plot(xs=[0, 5], ys=[0, 0], zs=[0, 0], linewidth=2, color='red')
        ax.plot(xs=[0, 0], ys=[0, 5], zs=[0, 0], linewidth=2, color='blue')
        ax.plot(xs=[0, 0], ys=[0, 0], zs=[0, 5], linewidth=2, color='black')

        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_zticks([])

        ax.set_xlim3d(-7, 7)
        ax.set_xlabel('x')
        ax.set_ylim3d(-7, 7)
        ax.set_ylabel('y')
        ax.set_zlim3d(-10, 10)
        ax.set_zlabel('z')

        plt.pause(0.0001)
        ax.cla()


def hand_detection(barrier: Barrier,
                   receiver_left_cam: Pipe = None,
                   receiver_right_cam: Pipe = None,
                   send_visualize: Pipe = None,
                   inside_cameras_parameters: dict = None):
    """
    Обобщенная функция по расспознованию ключевых точек на руке
    :param send_visualize:
    :param inside_cameras_parameters:
    :param barrier: Ожидание процессов
    :param receiver_left_cam: Полученние данных с камеры 1
    :param receiver_right_cam: Получение данных с камеры 2
    :return:
    """

    if inside_cameras_parameters is None:
        inside_cameras_parameters = {}

    # Информация о пальцах получаемых с камер
    fingers_info_left_camera = {
        "Thumb": {
            "close_finger": False,
            "angle": None,
            "id": FingersID.thumb
        },
        "Index": {
            "close_finger": False,
            "angle": None,
```

```
                "id": FingersID.index
            },
            "Middle": {
                "close_finger": False,
                "angle": None,
                "id": FingersID.middle
            },
            "Ring": {
                "close_finger": False,
                "angle": None,
                "id": FingersID.ring
            },
            "Pinky": {
                "close_finger": False,
                "angle": None,
                "id": FingersID.pinky
            }
        }
        fingers_info_right_camera = {
            "Thumb": {
                "close_finger": False,
                "angle": None,
                "id": FingersID.thumb
            },
            "Index": {
                "close_finger": False,
                "angle": None,
                "id": FingersID.index
            },
            "Middle": {
                "close_finger": False,
                "angle": None,
                "id": FingersID.middle
            },
            "Ring": {
                "close_finger": False,
                "angle": None,
                "id": FingersID.ring
            },
            "Pinky": {
                "close_finger": False,
                "angle": None,
                "id": FingersID.pinky
            }
        }

        # Ожидаем подключение всех паралельных процессов
        barrier.wait()

        # Инициализация базовых параметров распознования рук
        hands_left =
mp.solutions.hands.Hands(min_detection_confidence=DetectionConfig.detection_confidence,

min_tracking_confidence=DetectionConfig.tracking_confidence,
                                                max_num_hands=DetectionConfig.num_hands)

        hands_right =
mp.solutions.hands.Hands(min_detection_confidence=DetectionConfig.detection_confidence,

min_tracking_confidence=DetectionConfig.tracking_confidence,
                                                 max_num_hands=DetectionConfig.num_hands)

        # Расчёт триангуляции
        left_matrix = np.array(inside_cameras_parameters["camera_left"]["matrix"])
        right_matrix = np.array(inside_cameras_parameters["camera_right"]["matrix"])
        R = np.array(inside_cameras_parameters["R"])
        T = np.array(inside_cameras_parameters["T"])
```

```
273.
274.     # для триангуляции берем вектор поворотов радригеса с откалиброванного кадра (выбира-
ем сами), а также вектор
275.     # трансформаций
276.     left_R =
np.array(inside_cameras_parameters["camera_left"]["rotVecs"])[ReconstructionsConfig.world_imag
e]
277.     left_T =
np.array(inside_cameras_parameters["camera_left"]["tvecs"])[ReconstructionsConfig.world_image]
278.
279.     # Получение вторых параметров
280.     left_R, _ = cv2.Rodrigues(left_R)
281.     right_R = R @ left_R
282.     right_T = R @ left_T + T
283.
284.     # Расчёт матриц проекций
285.     P0 = left_matrix @ np.concatenate([left_R, left_T], axis=-1)
286.     P1 = right_matrix @ np.concatenate([right_R, right_T], axis=-1)
287.
288.     arduino = serial.Serial(port=ArduinoArm.com, baudrate=ArduinoArm.baud)
289.     arduino_code = 200
290.
291.     while True:
292.         key = cv2.waitKey(1) & 0xFF
293.
294.         # Получение данных с паралельных потоков камер
295.         frame_left_camera = receiver_left_cam.recv()
296.         frame_right_camera = receiver_right_cam.recv()
297.
298.         #  предсказание с дальнешим отображением данных
299.         frame_left_camera, fingers_info_left_camera, kps_left =
__prediction(frame_left_camera, hands_left,
300.                                                                                 fin-
gers_info_left_camera)
301.
302.         frame_right_camera, fingers_info_right_camera, kps_right =
__prediction(frame_right_camera, hands_right,
303.
fingers_info_right_camera)
304.
305.         # Установка общего значения пальцев
306.         fingers_info = {
307.             "Thumb": fingers_info_left_camera["Thumb"]["close_finger"] or fin-
gers_info_right_camera["Thumb"]["close_finger"],
308.             "Index": fingers_info_left_camera["Index"]["close_finger"] or fin-
gers_info_right_camera["Index"]["close_finger"],
309.             "Middle": fingers_info_left_camera["Middle"]["close_finger"] or fin-
gers_info_right_camera["Middle"]["close_finger"],
310.             "Ring": fingers_info_left_camera["Ring"]["close_finger"] or fin-
gers_info_right_camera["Ring"]["close_finger"],
311.             "Pinky": fingers_info_left_camera["Pinky"]["close_finger"] or fin-
gers_info_right_camera["Pinky"]["close_finger"]
312.         }
313.
314.         # Производим прямые линейные преобразования
315.         frame_p3ds = []
316.         for uv1, uv2 in zip(kps_left, kps_right):
317.             if uv1[0] == -1 or uv2[0] == -1:
318.                 _p3d = [-1, -1, -1]
319.             else:
320.                 _p3d = DLT(P0, P1, uv1, uv2)
321.             frame_p3ds.append(_p3d)
322.
323.         frame_p3ds =
np.array(frame_p3ds).reshape((len(mp.solutions.hands.HAND_CONNECTIONS), 3))
324.
325.         message = "$"
```

```
326.         for i in fingers_info:
327.             message += str(int(not fingers_info[i]))
328.         message += "$"
329.
330.         if arduino_code == 200:
331.             arduino.write(bytes(message, "utf-8"))
332.
333.         arduino_code = int(arduino.read(3).decode())
334.
335.         # Визуализируем полученные результаты
336.         send_visualize.send(frame_p3ds)
337.
338.         # Отображаем кадры с отрисовкой ключевых точек
339.         cv2.imshow("Camera Left", frame_left_camera)
340.         cv2.imshow("Camera Right", frame_right_camera)
341.
342.         if key == ord('q'):
343.             cv2.destroyAllWindows()
344.             hands_left.close()
345.             hands_right.close()
346.             break
347.
348.
349. def main():
350.     """
351.     Основная функция, которая запускает поток камер + отрисовку распознанных точек + 3D
реконструкцию
352.     :return:
353.     """
354.     web_cam1 = Camera(device_id=CameraConfig.ids[0],
355.                       mode=CameraConfig.mode,
356.                       width=CameraConfig.width,
357.                       height=CameraConfig.height)
358.
359.     web_cam2 = Camera(device_id=CameraConfig.ids[1],
360.                       mode=CameraConfig.mode,
361.                       width=CameraConfig.width,
362.                       height=CameraConfig.height)
363.
364.     # Загрузка информации о стерео параметрах камеры
365.     inside_stereo_parameters =
read_json(f"{ReconstructionsConfig.inside_camera_parameters_path}/stereo_params.json")
366.
367.     # Устанавлием барьер, который синхронихирует процессы между собой (дождется запуска 4
процессов)
368.     barrier = Barrier(4)
369.
370.     # Устанавливаем общение между потоками
371.     rec_web_cam1, send_web_cam1 = Pipe()  # Общение между камерой 1 и отрисовкой кон-
трольных точек
372.     rec_web_cam2, send_web_cam2 = Pipe()  # Общение между камерой 2 и отрисовкой кон-
трольных точек
373.     rec_visualize, send_visualize = Pipe()  # Общение между отрисовкой контрольных точек
и 3D реконструкцией
374.
375.     # Параметры камеры 1 (Информация в классе самой камеры)
376.     cam1_args = {
377.         "barrier": barrier,
378.         "show_gui": False,
379.         "sender": send_web_cam1,
380.     }
381.
382.     # Параметры камеры 2 (Информация в классе самой камеры)
383.     cam2_args = {
384.         "barrier": barrier,
385.         "show_gui": False,
386.         "sender": send_web_cam2,
```

```
387.      }
388.
389.      # Объявление процессов
390.      mp_web1 = Process(target=web_cam1.stream, kwargs=cam1_args, name="CameraLeft")
391.      mp_web2 = Process(target=web_cam2.stream, kwargs=cam2_args, name="CameraRight")
392.      visualize = Process(target=visualize_3d, args=(barrier, rec_visualize,),
name="Visualize 3D")
393.      res_cam = Process(target=hand_detection,
394.                        args=(barrier, rec_web_cam1, rec_web_cam2,
395.                              send_visualize, inside_stereo_parameters,),
396.                        name="HandDetection")
397.
398.      #  Запуск процессов
399.      mp_web1.start()
400.      mp_web2.start()
401.      visualize.start()
402.      res_cam.start()
403.
404.      # Дожидаемся завершения работы отрисовки ключевых точек
405.      res_cam.join()
406.
407.      # Убиваем остальные процессыт
408.      mp_web1.terminate()
409.      mp_web2.terminate()
410.      visualize.terminate()
411.
412.
413. if __name__ == "__main__":
414.      main()
415.
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Результаты выполнения программы

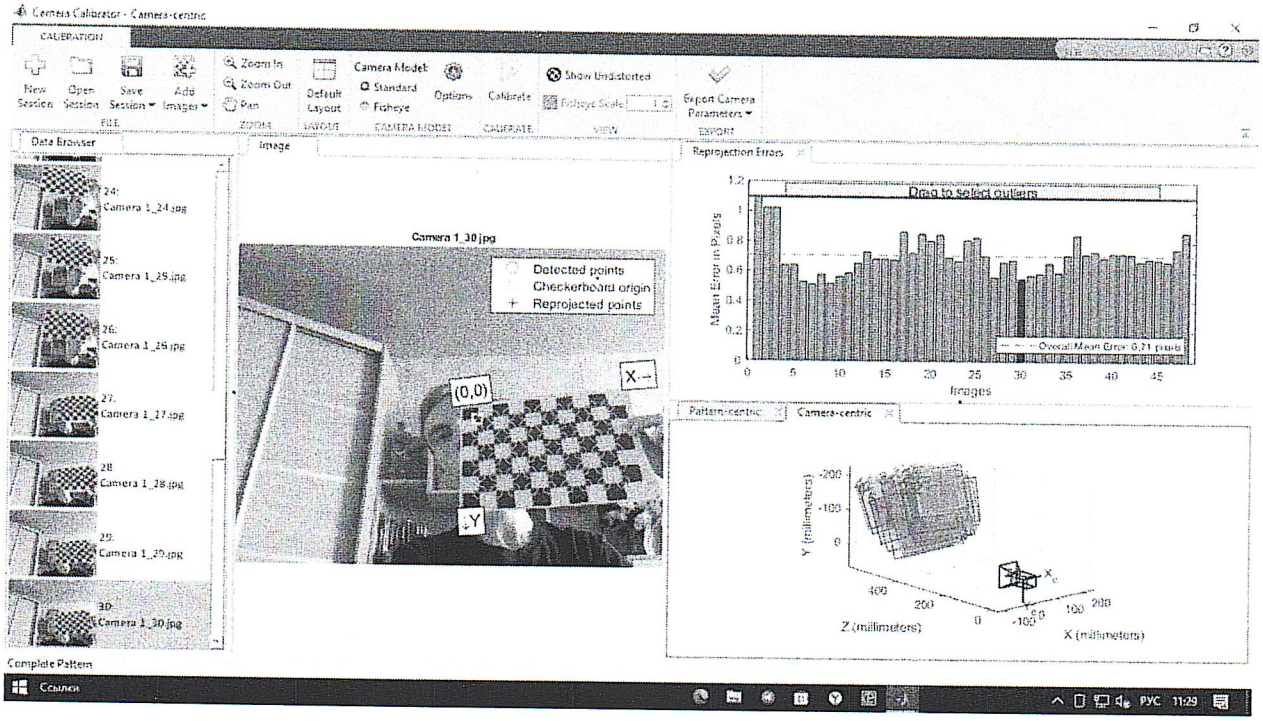

Рисунок Б.1 – Калибровка камеры

Рисунок Б.2 – Процесс сбора датасета

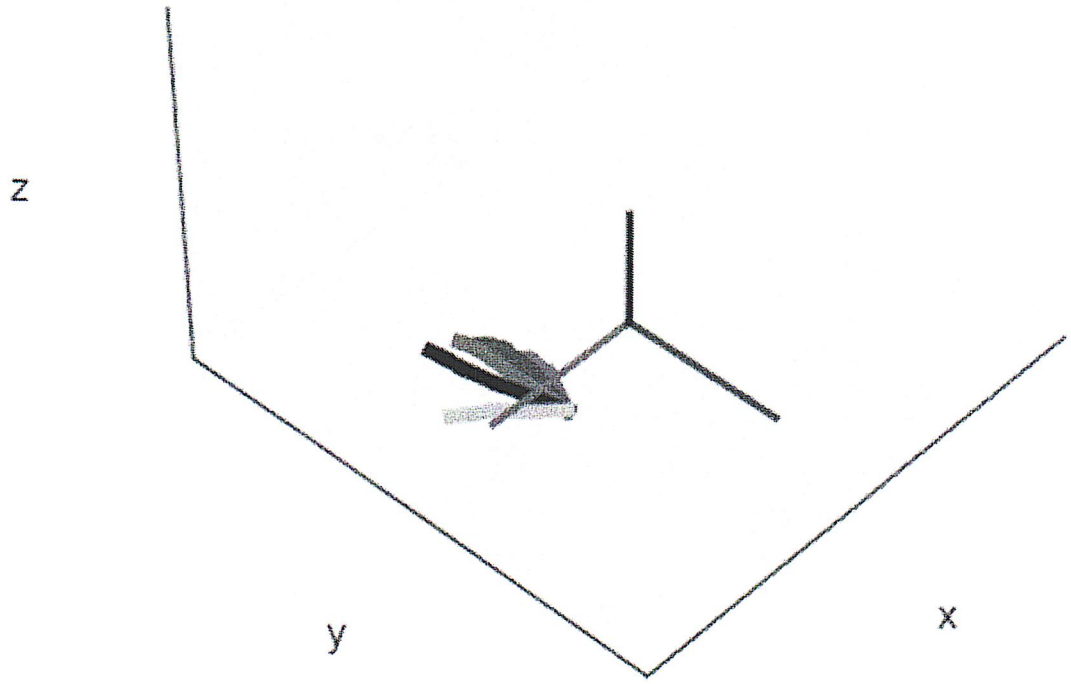

Рисунок Б.3 – Результат работы системы

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

СОГЛАСОВАНО

Начальник отдела ОНиПКРС

_____ Е.М. Димитриади
(подпись)
«10» 06 2024 г.

Декан ФЭУ

_____ А.С. Гудим
(подпись)
«10» 06 2024 г.

УТВЕРЖДАЮ

Проректор по научной работе,
д-р техн. наук. профессор

_____ А.В. Космынин
(подпись)
«10» 06 2024 г.

АКТ
о приемке в эксплуатацию проекта «Программа распознавания положения рук оператора коллаборативного робота для реализации следящей системы управления»

г. Комсомольск-на-Амуре «10» 06 2024 г.

Комиссия в составе представителей:

со стороны заказчика

- С.И. Сухоруков – руководитель СКБ,
- А.С. Гудим – декан ФЭУ

со стороны исполнителя

- Ю.С. Иванов– руководителя проекта,
- Д.М.Грабарь– группа,
- М.А. Лямин– группа.

составила акт о нижеследующем:

«Исполнитель» передает проект «Программа распознавания положения рук оператора коллаборативного робота для реализации следящей системы управления», в составе:

1. Вычислительный модуль
2. Управляющие программы
3. WEB-камеры Logitech V-U0028

Руководитель проекта _____ 10.06.24 Ю.С. Иванов
(подпись, дата)

Исполнители проекта _____ 10.06.24 Д.М. Грабарь
(подпись, дата)

_____ 10.06.2024 М.А. Лямин
(подпись, дата)