# По проекту Newman было сделано следующее:

1. На рабочем компьютере была создана виртуальная машина VMware Workstation Pro
2. На виртуальную машину установлен дистрибутив ubuntu 20.04
3. В соответствии с рекомендациями по установке на официальном сайте был установлен программный продукт “ROS Noetic”
4. Были установлены необходимые для работы с проектов программные компоненты в ОС ubuntu и ROS
5. Работа ROS была протестирована на разных программных пакетах, работающих под управлением ROS (turtlesim, rviz)
6. Для изучения ROS и добавление в Newman возможности двигаться к ROS был подключен манипулятор с хватом. На манипуляторе отрабатывались возможности работы с суставами в ROS и моделирование этого процесса.
7. Для калибровки серводвигателей в системе и задания максимальных и минимальных углов движения в суставах, была написана прошивка для Arduino и подключен потенциометр для задания калибровочных значений.
8. Для манипулятора написан URDF файл и разработана прошивка, произведена калибровка положения и перемещения
9. Прошивка манипулятора была адаптирована для движения шеи Newman
10. Для разработки алгоритмов распознавания лица и слежением за лицами в ROS был добавлена и настроена программный код для работы с камерой intel real sense D435. Камера подключена и проверена.
11. Создано и визуализировано в ROS представление Newman (Цифровой двойник)
12. Разработана программная часть, реализующая распознавание лица
13. Разработана программная часть, осуществляющая слежение за лицом
14. Для захвата изображения с камеры, установленной в глазу Newman на микрокомпьютер установлена операционная система ubuntu и установлен программный пакет ROS
15. Для захвата изображения с камеры в глазу был собран из исходного кода и установлен программный пакет на микрокомпьютере под управлением ROS
16. Было настроено сетевое взаимодействие между двумя компьютерами под управлением ROS
17. Был написан скрипт для быстрой установки и настройки ROS и программных компонентов Ubuntu
18. Был создан и настроен git сервер внутри лаборатории 116
19. Для повышения производительности работы и портативности с помощью установочного скрипта и git репозитория проект был перенесен с виртуальной машины на ноутбук
20. Для еще большего повышения производительности программных алгоритмов на ноутбуке был установлен, настроен и протестирован инструментарий для вычислений на графическом процессоре Nvidia CUDA. Текущая версия программных алгоритмов разработанных для Newman не поддерживает CUDA, но в дальнейшем планируется добавить ее поддержку
21. Для повышения удобства разработки проекта настроен ssh протокол обмена между ноутбуком с ROS и рабочим компьютером на Windows. Настроен процесс удаленной разработки и отладки между этими компьютерами, настроен проброс графической подсистемы на удаленный компьютер через ssh подключение.
22. Для работы Newman в сети ROS без DNS и DHCP были соответствующим образом настроены ноутбук и микрокомпьютер
23. Для правильного слежения за лицом откалиброваны углы перемещения головы робота и начальное положение его цифрового двойника
24. Произведены изменения в коммутации цепей питания и управления Newman
25. Подключен и настроен внешний микрофон, встроенный в туловище Newman
26. Начата разработка и конфигурирование механизмов речевого взаимодействия с роботом на основе ПО PocketSphinx, микрофона, звуковой карты и усилителя НЧ.

# Установленные программные компоненты

Пароль пользователя ноутбука “116”

Для работы над проектом были установлены на ноутбук следующие программные продукты:

ROS noetic нужно устанавливать по инструкции <http://wiki.ros.org/noetic/Installation/Ubuntu>

Ubuntu 20.04 LTS

Terminator – многооконный терминал

Git – программа для работы с репозиториями

VScode – среда разработки

Внутри vscode установлены ROS (microsoft), CMake(twxs), Python, C++

net-tools- инструментарий для работы с сетью из консоли

cuda external pkg, libglfw3, libglfw3-dev, – зависимости для куда

nvidia drivers – последние драйвера

nvidia-utils-535

nvidia-driver-535

cuda для того чтобы установить cuda нужно следовать инструкции <https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=Ubuntu&target_version=20.04&target_type=deb_local>

Для работы с распознаванием речи были установлены следующие компоненты

GStreamer и его зависимости libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev libgstreamer-plugins-bad1.0-dev gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly gstreamer1.0-libav gstreamer1.0-doc gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-alsa gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio

bison swig

alsa-utils libasound2-dev

pocketsphinx

python3-pocketsphinx

pocketsphinx-en-us

ros-noetic-audio-common - пакет поддержки аудио для ROS

На микрокомпьютере Raspbery PI были установлены следующие компоненты:

Ubuntu 20.04 server

ros noetic base

raspicam – пакет для захвата видео с камер, который нужно собрать по инструкции <https://github.com/cedricve/raspicam>

# Цифровой двойник

Внешний вид цифрового двойника представлен на рисунке 2

Диаграмма URDF описания изображена на рисунке 3

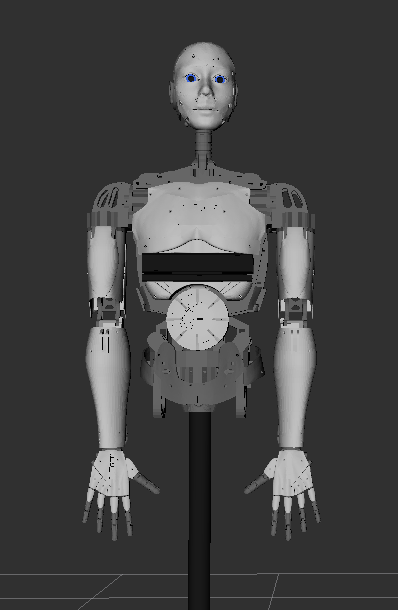


Рисунок 1 Цифровой двойник

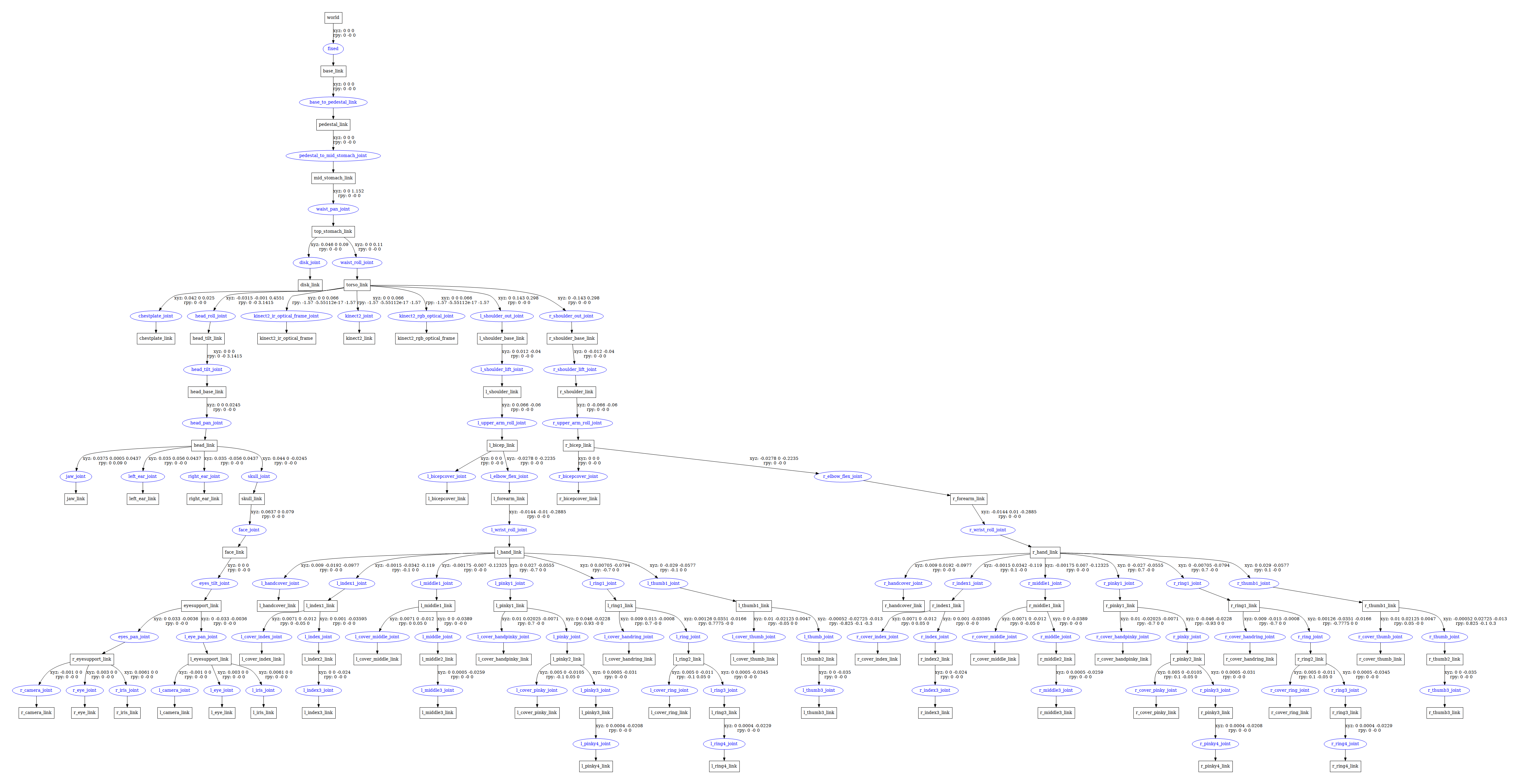


Рисунок 2 диаграмма URDF описания

# Структуры системы трекинга лица

Для работы алгоритмов трекинга лица требуется источник видео. Источником является видеокамера, которая установлена в правом глазу робота. Захват и обработку данных с камеры занимается микрокомпьютер Raspbery PI 3B+. На микрокомпьютере установлен Linux Ubuntu и ROS noetic.

Raspberry PI 3B+ управляет суставами шеи через подключенный по средствам USB микроконтроллер Arduino Mega а также с помощью ШИМ контроллера на микросхеме PCA9685. Передача данных между Arduino и Raspberry PI осуществляется по протоколу ros serial, между Arduino и PCA9685 осуществляется по протоколу I2C, сервоприводы имеют прямое подключение к контроллеру на PCA9685. Все вычисления, контроль и управление в системе осуществляет ноутбук под управлением ROS noetic. Ноутбук и микрокомпьютер соединяются по средствам проводной сети Ethernet.

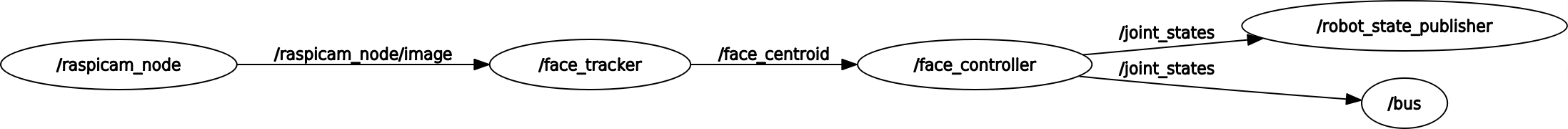


Рисунок 3 Структура системы трекинга лица в ROS

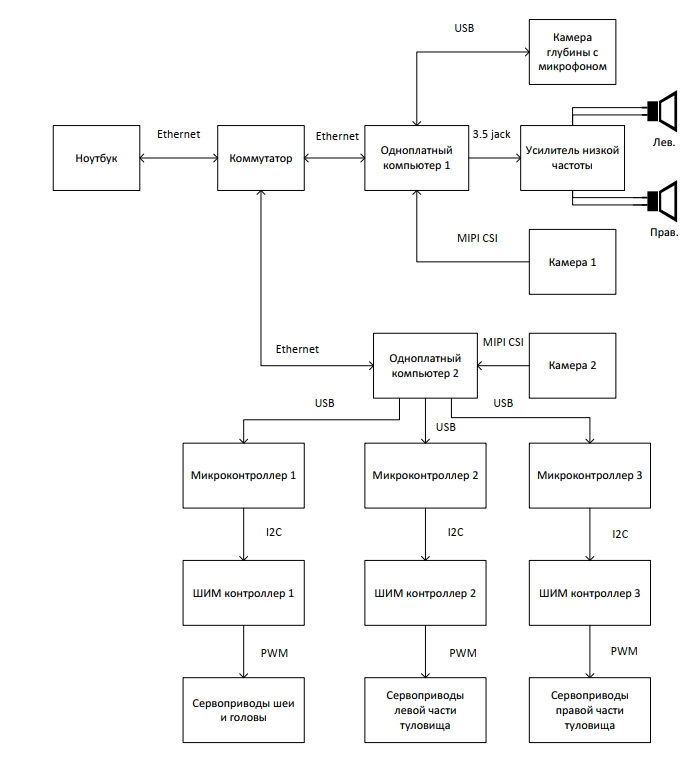


Рисунок 4 Планируемая структурная-схема проекта

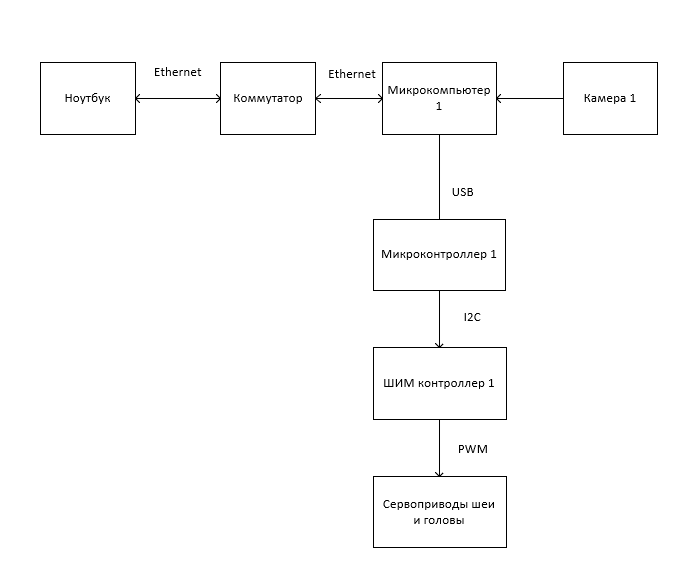


Рисунок 5 Реализованная структурная-схема проекта

Структура программной части робота представлена на рисунке 3. Сигнал с видеокамеры поступает на микрокомпьютер, который работает под управлением среды ROS. В ROS на микрокомпьютере запущен узел “raspicame\_node”. Этот узел принимает на вход видео с камеры преобразует видео в последовательный вид, упаковывает в пакеты и отправляет в теме “raspicam\_node/image” по сети Ethernet ноутбуку. На ноутбуке сообщение принимает узел “face\_tracker”. В этом узле на изображении обнаруживаются и обводятся окружностью лица, центы окружностей отправляются в сообщении “face\_centroid” узлу “face\_controller”. Узел “face\_controller” размечает изображение на участки, при попадании на которые центров окружностей требуется повернуть голову в лево, в право или ничего не делать. Этот узел отправляет в случае необходимости в теме joint\_states сообщения для поворота головы в нужную сторону таким образом, чтобы в центральном участке изображения находился центр окружности лица. Сообщения “Joint states” передаются узлу “robot\_state\_publisher” которое при помощи URDF файла и узла RVIZ (на рисунке не показан) создает цифрового двойника робота, который точно повторяет все движения реального. Также сообщения “Joint\_states” через локальную сеть поступают на Raspberry pi а из нее на узел “bus”, который выполняется на Arduino mega. На этом узле происходит непосредственное управление сервоприводами через драйвер на микросхеме PCA9685.

К началу выставки планировалось реализовать Newman в соответствии с Рисунком 4, но удалось реализовать в том виде, который изображен на рисунке 5.

# Структура программного репозитория

Установочный скрипт хранится в корневой папке репозитория Newman\_ws/rosinstall.sh

Структура директорий репозитория Newman\_ws/src

├── **face\_tracker\_control\_Newman** пакет ros который производит трекинг лица

│ ├── config

│ ├── msg

│ └── src

├── **face\_tracker\_Newman** – пакет ros в котором происходит определение лица на видео

│ ├── config

| | └── track.yaml файл параметров ROS, имеющий такие параметры, как путь к файлу Haar, topic входного изображения, topic выходного изображения и флаги для включения и отключения отслеживания лица.

│ ├── data

| | └── face.xml - классификатор лица на основе признаков Хаара из пакета OpenCV

│ ├── launch

│ └── src

**├── inmoov\_description** - в этом пакете хранится URDF описание робота

│ ├── config

│ ├── launch

│ ├── robots

│ └── urdf

├── **inmoov\_meshes** – храняться 3D модели деталей робота

│ └── meshes

├── **raspicam\_node** -скомпилированный код пакета raspicam, который работает на RaspberryPi

├── **robot\_control\_arduino** -скетч Arduino для контроллера, который управляет суставами

├── **servo\_test** – скетч Arduino для калибровки серводвигателей

**└── sound\_Newman** – метапакет для распознавания голоса и голосового ответа

├── sound\_Newman

└── **speech\_to\_text\_Newman** – пакет преобразования речи в текст

├── config

│ └── test

├── launch

├── model

│ └── zero\_ru\_cont\_8k\_v3

│ ├── zero\_ru.cd\_cont\_4000

│ ├── zero\_ru.cd\_ptm\_4000

│ └── zero\_ru.cd\_semi\_4000

└── scripts

# Описание кода пакета трекера лица

Нужно включить ros/ros.h в каждый узел ROS C ++; в противном случае исходный код не будет компилироваться. Остальные три заголовка являются заголовками транспорта изображения, которые имеют функции для публикации и подписки на сообщения изображения с низкой пропускной способностью. Заголовок cv\_bridge имеет функции для преобразования между типами данных OpenCV ROS. Заголовок image\_encoding.h имеет формат кодирования изображения, используемый во время преобразования ROS-OpenCV:

#include <ros/ros.h>

#include <image\_transport/image\_transport.h>

#include <cv\_bridge/cv\_bridge.h>

#include <sensor\_msgs/image\_encodings.h>

Следующий набор заголовков для OpenCV. Заголовок imgproc состоит из функций обработки изображений, highgui имеет функции, связанные с GUI, а objdetect.hpp имеет API для обнаружения объектов, такие как классификатор Haar:

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>

#include "opencv2/objdetect.hpp"

Последний заголовочный файл предназначен для доступа к пользовательскому сообщению с именем centroid. Определение сообщения центроида имеет два поля, int32x ,а также Int32y, он может содержать координаты центра распознанного лица. Можно проверить это определение сообщения из папки face\_tracker\_pkg/msg/centroid.msg

#include "face\_tracker\_control\_Newman/centroid.h"

Следующие строки кода дают имя окну необработанного изображения и окну обнаружения лица:

static const std::string OPENCV\_WINDOW = "raw\_image\_window";

static const std::string OPENCV\_WINDOW\_1 = "face\_detector";

Следующие строки кода создают класс C ++ для детектора лиц. Фрагмент кода создает дескрипторы NodeHandle, который является обязательным дескриптором для узла ROS; image\_transport, который помогает отправлять сообщения ROS Image по графу вычислений ROS; и издатель для лицевого центроида, который может публиковать значения центроидов, используя файл centroid.msg. Остальные определения предназначены для обработки значений параметров из файла параметров track.yaml:

class Face\_Detector

{

  ros::NodeHandle nh\_;

  image\_transport::ImageTransport it\_;

  image\_transport::Subscriber image\_sub\_;

  image\_transport::Subscriber image\_sub\_;

  image\_transport::Publisher image\_pub\_;

  ros::Publisher face\_centroid\_pub;

  face\_tracker\_control\_Newman::centroid face\_centroid;

  string input\_image\_topic, output\_image\_topic, haar\_file\_face;

  int face\_tracking, display\_original\_image, display\_tracking\_image, center\_offset, screenmaxx;

Ниже приведен код для извлечения параметров ROS из файла track.yaml. Преимущество использования параметров ROS состоит в том, что мы можем избежать жесткого кодирования этих значений внутри программы и изменять значения без перекомпиляции кода:

try{

  nh\_.getParam("image\_input\_topic", input\_image\_topic);

  nh\_.getParam("face\_detected\_image\_topic", output\_image\_topic);

  nh\_.getParam("haar\_file\_face", haar\_file\_face);

  nh\_.getParam("face\_tracking", face\_tracking);

  nh\_.getParam("display\_original\_image", display\_original\_image);

  nh\_.getParam("display\_tracking\_image", display\_tracking\_image);

  nh\_.getParam("center\_offset", center\_offset);

  nh\_.getParam("screenmaxx", screenmaxx);

  ROS\_INFO("Successfully Loaded tracking parameters");

  }

Следующий код создает подписчика для topic входного изображения и издателя для изображения с обнаруженным лицом. Всякий раз, когда изображение попадает в topic входного изображения, оно вызывает функцию imageCb. Названия тем извлекаются из параметров ROS. Создаем другого издателя для публикации значения центроида, который является последней строкой фрагмента кода:

}

    image\_sub\_ = it\_.subscribe(input\_image\_topic, 1,

      &Face\_Detector::imageCb, this);

    image\_pub\_ = it\_.advertise(output\_image\_topic, 1);

    face\_centroid\_pub = nh\_.advertise<face\_tracker\_control\_Newman::centroid>("/face\_centroid",10);

  }

Следующий фрагмент кода - это определение imageCb, который является обратным вызовом для input\_image\_topic, что он в основном делает, это конвертирует sensor\_msgs/Image данные в cv::Mat Тип данных OpenCV. cv\_bridge::CvImagePtr cv\_ptr буфер выделяется для хранения образа OpenCV после выполнения преобразования ROS-OpenCV с использованием функции cv\_bridge::toCvCopy:

 void imageCb(const sensor\_msgs::ImageConstPtr& msg)

  {

    cv\_bridge::CvImagePtr cv\_ptr;

    namespace enc = sensor\_msgs::image\_encodings;

    try

    {

      cv\_ptr = cv\_bridge::toCvCopy(msg, sensor\_msgs::image\_encodings::BGR8);

    }

    catch (cv\_bridge::Exception& e)

    {

      ROS\_ERROR("cv\_bridge exception: %s", e.what());

      return;

    }

      string cascadeName = haar\_file\_face;

            CascadeClassifier cascade;

      if( !cascade.load( cascadeName ) )

      {

    cerr << "ERROR: Could not load classifier cascade" << endl;

      }

Переходим к основной части программы, которая заключается в обнаружении лица, выполненного для преобразованного типа данных изображения OpenCV, из сообщения ROS Image. Ниже приведен вызов функции detectAndDraw(), который выполняет обнаружение лица, и в последней строке можно увидеть опубликованную в topic выходного изображения. С помощью cv\_ptr->image мы можем получить cv::Mat тип данных, и в следующей строке cv\_ptr->toImageMsg() может преобразовать это в ROSImage. Аргументы функции

detectAndDraw() - это изображение OpenCV и каскадные переменные:

detectAndDraw( cv\_ptr->image, cascade );

image\_pub\_.publish(cv\_ptr->toImageMsg());

Функция detectAndDraw() взята из примера кода OpenCV для обнаружения лица: аргументами функции являются входное изображение и каскадный объект. Следующий фрагмент кода сначала преобразует изображение в градации серого и выровняет гистограмму с помощью API-интерфейсов OpenCV. Это своего рода предварительная обработка перед обнаружением лица по изображению. Для этого используется функция cascade.detectMultiScale()

<http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html>

 Mat gray, smallImg;

    cvtColor( img, gray, COLOR\_BGR2GRAY );

    double fx = 1 / scale ;

    resize( gray, smallImg, Size(), fx, fx, INTER\_LINEAR );

    equalizeHist( smallImg, smallImg );

    t = (double)cvGetTickCount();

    cascade.detectMultiScale( smallImg, faces,

        1.1, 15, 0

        |CASCADE\_SCALE\_IMAGE,

        Size(30, 30) );

Следующий цикл будет повторяться для каждого лица, обнаруженного с помощью функции detectMultiScale(). Для каждого лица он находит центроид и публикует /face\_centroid topic:

    for ( size\_t i = 0; i < faces.size(); i++ )

    {

        Rect r = faces[i];

        Mat smallImgROI;

        vector<Rect> nestedObjects;

        Point center;

        Scalar color = colors[i%8];

        int radius;

        double aspect\_ratio = (double)r.width/r.height;

        if( 0.75 < aspect\_ratio && aspect\_ratio < 1.3 )

        {

            center.x = cvRound((r.x + r.width\*0.5)\*scale);

            center.y = cvRound((r.y + r.height\*0.5)\*scale);

            radius = cvRound((r.width + r.height)\*0.25\*scale);

            circle( img, center, radius, color, 3, 8, 0 );

        face\_centroid.x = center.x;

        face\_centroid.y = center.y;

            //Publishing centroid of detected face

        face\_centroid\_pub.publish(face\_centroid);

        }

Чтобы сделать окно выходного изображения более интерактивным, есть текст и строки для оповещения о лице пользователя слева или справа или в центре. Для этого используется API OpenCV. Вот код для отображения текста, такого как Лево, Право и Центр, на экране:

    putText(img, "left", cvPoint(50,240), FONT\_HERSHEY\_SIMPLEX, 1, cvScalar(255,0,0), 2, LINE\_AA);

    putText(img, "center", cvPoint(280,240), FONT\_HERSHEY\_SIMPLEX, 1, cvScalar(0,0,255), 2, LINE\_AA);

    putText(img, "right", cvPoint(480,240), FONT\_HERSHEY\_SIMPLEX, 1, cvScalar(255,0,0), 2, LINE\_AA);

# Файл track.yaml

Файл track.yaml содержит параметры ROS, которые требуются face\_tracker\_node, Вот содержимое track.yaml:

image\_input\_topic: "/raspicam\_node/image"

face\_detected\_image\_topic: "/face\_detector/raw\_image"

haar\_file\_face: "/home/robot/Newman\_ws/src/face\_tracker\_Newman/data/face.xml"

face\_tracking: 1

display\_original\_image: 0

display\_tracking\_image: 1

Можно изменить все параметры в соответствии с требованиями. В частности, может потребоваться изменить haar\_file\_face, который является путем к файлу лица Haar. Если установить face\_tracking: 1, он включит отслеживание лица, в противном случае нет. Кроме того, если нужно отобразить исходное изображение и изображение для отслеживания лица, можно установить здесь флаги.

# Файл конфигурации параметров сервопривода servo\_param.yaml

Файл servo\_param.yaml содержит конфигурацию, такую, как предельные углы поворота контроллера и шага движения; Кроме того, он имеет параметры экрана, такие как максимальное разрешение изображения с камеры и смещение от центра изображения. Смещение используется для определения области вокруг фактического центра изображения:

servomaxx: 0.5   #max degree servo horizontal (x) can turn

servomin: -0.5   # Min degree servo horizontal (x) can turn

screenmaxx: 640   #max screen horizontal (x)resolution

center\_offset: 50  #offset pixels from actual center to right and left

step\_distancex: 0.03 #x servo rotation steps

# Узел контроллера трекера лица

Узел контроллера трекера лица отвечает за управление сервоприводом шеи робота в соответствии с положением центроида лица. Код трекера находится по пути face\_tracker\_control\_Newman/src/face\_tracker\_controller.cpp.

Заголовки, используемые в узле следующие

#include "ros/ros.h"

#include <iostream>

#include <sensor\_msgs/JointState.h>

#include "face\_tracker\_control\_Newman/centroid.h"

Следующие переменные содержат значения параметров из servo\_param.yaml:

int servomaxx, servomin,screenmaxx, center\_offset, center\_left, center\_right;

float servo\_step\_distancex, current\_pos\_x;

Следующие заголовки сообщения sensor\_msgs::JointState предназначено для хранения позы робота

Это обработчик издателя для публикации команд положения в контроллере:

ros::Publisher dynamixel\_control;

в функции main() Первая строка - это подписчик /face\_centroid, у которого есть значение centroid, и когда значение приходит в topic, оно вызывает функцию face\_callback ():

ros::Subscriber number\_subscriber = node\_obj.subscribe("/face\_centroid",10,face\_callback);

Следующая строка инициализирует дескриптор издателя, в котором значения будут опубликованы в разделе joint\_states:

dynamixel\_control = node\_obj.advertise<sensor\_msgs::JointState>("joint\_states", 1);

Следующий код создает ограничения вокруг фактического центра изображения. Это полезно для получения приблизительной центральной точки изображения:

center\_left = (screenmaxx / 2) - center\_offset;

center\_right = (screenmaxx / 2) + center\_offset;

Вот функция обратного вызова, выполняемая при получении значения центроида, поступающего через раздел /face\_centroid. Этот обратный вызов также имеет логику для перемещения Dynamixel для каждого значения центроида.

В первом разделе значение x в центроиде сравнивается с center\_left, а если оно слева, оно просто увеличивает положение сервоконтроллера. Он будет публиковать текущее значение, только если текущая позиция находится в пределах лимита. Если он находится в пределе, то он опубликует текущую позицию в контроллере. Логика одинакова для правой стороны: если грань находится на правой стороне изображения, это уменьшит положение контроллера.

Когда камера достигает центра изображения, она останавливается и ничего не делает. Этот цикл повторяется, и мы получим непрерывное отслеживание:

void track\_face(int x,int y)

{

    //Find out if the X component of the face is to the left of the middle of the screen.

    if(x < (center\_left)){

    ROS\_INFO("Face is at Left");

    current\_pos\_x += servo\_step\_distancex;

    current\_pose = current\_pos\_x;

    ROS\_INFO("servo controller current\_position = %f",current\_pos\_x);

        if(current\_pos\_x < servomaxx and current\_pos\_x > servomin ){

        joint\_state.header.stamp = ros::Time::now();

        joint\_state.name[0] ="waist\_pan\_joint";

        joint\_state.position[0] = 0;

        joint\_state.name[1] ="head\_roll\_joint";

        joint\_state.position[1] = 0;

        joint\_state.name[2] ="head\_tilt\_joint";

        joint\_state.position[2] = 0;

        joint\_state.name[3] ="head\_pan\_joint";

        joint\_state.position[3] = current\_pose;

        joint\_state.name[4] ="jaw\_joint";

        joint\_state.position[4] = 0;

        dynamixel\_control.publish(joint\_state);

    }

    }

    //Find out if the X component of the face is to the right of the middle of the screen.

    else if(x > center\_right){

    ROS\_INFO("Face is at Right");

    current\_pos\_x -= servo\_step\_distancex;

    current\_pose = current\_pos\_x;

# Контроллер сервопривода

Сервопривод, который поворачивает голову робота для отслеживания лиц людей, управляется ШИМ сигналом. Управляющий сигнал генерирует микросхема PCA9685. Микросхема в свою очередь управляется контролером Arduino Mega. Контроллер через порт usb подключен к микрокомпютеру Raspberry Pi 3b+. Связь ROS с Arduino происходит через программный интерфейс последовательного порта с помощью утилиту rosserial.

Код узла ROS, который исполняется на Arduino приведен ниже по тексту. В проекте используются стандартные библиотеки без изменений.

Adafruit\_PWMServoDriver -v3.01

#include <Wire.h>

#include <Adafruit\_PWMServoDriver.h>

#include <ros.h>

#include <std\_msgs/Int32.h>

#include <sensor\_msgs/JointState.h>

#define FREQUENCY 50

uint8\_t servonum = 0;

ros::NodeHandle node\_handle;

Adafruit\_PWMServoDriver robot\_servos;

//Servo robot\_servos[5];

int input;

int MIN\_PULSE\_WIDTH [5] ={640, 640, 525, 880, 640};

int MAX\_PULSE\_WIDTH [5]={2400, 2400, 2450, 2450, 2400

int mid\_positions[5] = {90, 90, 90, 90, 90};

int SERVO\_CURRENT\_POSITIONS[5];

float TARGET\_JOINT\_POSITIONS[5] = {0,0,0,0,0};

// Convert the joint state values to degrees, adjust for the center and write to the servo

void writeServos() {

  for (int j = 0; j < 5; j++) {

    int target\_angle;

    if (j == 2) {

      // Due to difference in mounting directions

      target\_angle = - TARGET\_JOINT\_POSITIONS[j]\*(180/3.14) + mid\_positions[j];

    } else {

      target\_angle = TARGET\_JOINT\_POSITIONS[j]\*(180/3.14) + mid\_positions[j];

    }

    robot\_servos.setPWM(j, 0, pulseWidth(target\_angle, j));

    SERVO\_CURRENT\_POSITIONS[j] = target\_angle;

  }

  node\_handle.spinOnce();

}

void servoControlSubscriberCallbackJointState(const sensor\_msgs::JointState& msg) {

  TARGET\_JOINT\_POSITIONS[0] = msg.position[0];

  TARGET\_JOINT\_POSITIONS[1] = msg.position[1];

  TARGET\_JOINT\_POSITIONS[2] = msg.position[2];

  TARGET\_JOINT\_POSITIONS[3] = msg.position[3];

  TARGET\_JOINT\_POSITIONS[4] = msg.position[4];

  writeServos();//-----------------------------------------------------------------------------------------------------

}

ros::Subscriber<sensor\_msgs::JointState> servo\_control\_subscriber\_joint\_state("joint\_states", &servoControlSubscriberCallbackJointState);

void setup() {

robot\_servos.begin();

robot\_servos.setPWMFreq(FREQUENCY);

  // Initial the servo motor connections and initialize them at home position

  for (unsigned int i = 0; i < 5; i++) {

    //robot\_servos[i].attach(servo\_pins[i]);//----------------------------------------------------------

    //robot\_servos[i].write(mid\_positions[i]);//--------------------------------------------------------написать свою функцию

    robot\_servos.setPWM(i, 0, pulseWidth(mid\_positions[i], i ));

    SERVO\_CURRENT\_POSITIONS[i] = mid\_positions[i];//-------------------------------------------------

  }

  // Set the communication BaudRate and start the node

  node\_handle.getHardware()->setBaud(115200);

  node\_handle.initNode();

  node\_handle.subscribe(servo\_control\_subscriber\_joint\_state);

}

void loop() {//ок

  // Keep calling the spinOnce() method in this infinite loop to stay tightly coupled with the ROS Serial

  node\_handle.spinOnce();//ок

  delay(1);//ок

}

int pulseWidth(int angle, int input)//---------------------------------

{

  int pulse\_wide, analog\_value;

  //int MIN\_PULSE\_WIDTH [6] ={1269, 718, 500, 446, 580, 525};

 // int MAX\_PULSE\_WIDTH [6]={2196, 2409, 2597, 2551, 2597, 1597};

  pulse\_wide   = map(angle, 0, 180, MIN\_PULSE\_WIDTH[input], MAX\_PULSE\_WIDTH[input]);

  analog\_value = int(float(pulse\_wide) / 1000000 \* FREQUENCY \* 4096);

  //Serial.println(analog\_value);

  return analog\_value;

}

# Соединение всех узлов вместе

Далее рассмотрим окончательный файл запуска face\_tracker\_Newman/start\_automatic\_tracking.launch, Этот файл запускает захват изображения с камеры, распознавание лиц и отслеживание с помощью серводвигателей:

<launch>

   <arg name="enable\_raw" default="True"/>

   <arg name="enable\_imv" default="false"/>

   <arg name="camera\_id" default="0"/>

   <arg name="camera\_frame\_id" default="raspicam"/>

   <arg name="camera\_name" default="camerav2\_1280x960"/>

   <arg name="model" />

   <arg name="gui" default="true" />

    <machine name="robot" address="192.168.1.110" env-loader="/opt/ros/noetic/env\_robot.sh" user="ubuntu" password="123456789" />

    <node machine="robot" type="raspicam\_node" pkg="raspicam\_node" name="raspicam\_node" output="screen">

    <param name="private\_topics" value="true"/>

    <param name="camera\_frame\_id" value="$(arg camera\_frame\_id)"/>

    <param name="enable\_raw" value="$(arg enable\_raw)"/>

    <param name="enable\_imv" value="$(arg enable\_imv)"/>

    <param name="camera\_id" value="$(arg camera\_id)"/>

    <param name="camera\_info\_url" value="package://raspicam\_node/camera\_info/camerav2\_1280x960.yaml"/>

    <param name="camera\_name" value="$(arg camera\_name)"/>

    <param name="width" value="640"/>

    <param name="height" value="480"/>

    <param name="framerate" value="30"/>

  </node>

    <node machine="robot" pkg="rosserial\_python" type="serial\_node.py" name="bus" respawn="true">

       <param name="port" value="/dev/ttyUSB0"/>

       <param name="baud" value="115200"/>

    </node>

   <param name="robot\_description" textfile="$(find inmoov\_description)/robots/inmoov.urdf" />

   <node name="robot\_state\_publisher" pkg="robot\_state\_publisher" type="robot\_state\_publisher" />

   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find inmoov\_description)/urdf.rviz" required="true" />

     <rosparam file="$(find face\_tracker\_Newman)/config/track.yaml" command="load"/>

     <node name="face\_tracker" pkg="face\_tracker\_Newman" type="face\_tracker\_node" output="screen" />

     <rosparam file="$(find face\_tracker\_control\_Newman)/config/servo\_param.yaml" command="load"/>

     <node name="face\_controller" pkg="face\_tracker\_control\_Newman" type="face\_tracker\_controller" output="screen" />

</launch>

# Финальный запуск

Для запуска проекта требуется в командной строке ros выполнить:

**$ roslaunch face\_tracker\_Newman start\_automatic\_tracking.launch**

На рисунке 6 изображены окно программы отслеживания лица, цифровой двойник и терминал. На терминале отображается сообщение что лицо в центре. Это значит то, что поворот головы не требуется.

# Выводы

Была реализованы алгоритмы распознавания и слежения за лицом на базе Newman, созданы и протестирован цифровой двойник, запущенна мультикомпьютерная сеть ROS, Проект разрабатывался с использованием механизма контроля версий, был создан сервер для хранения проекта. Налажена возможность мультиплатформенной удаленной разработки проекта

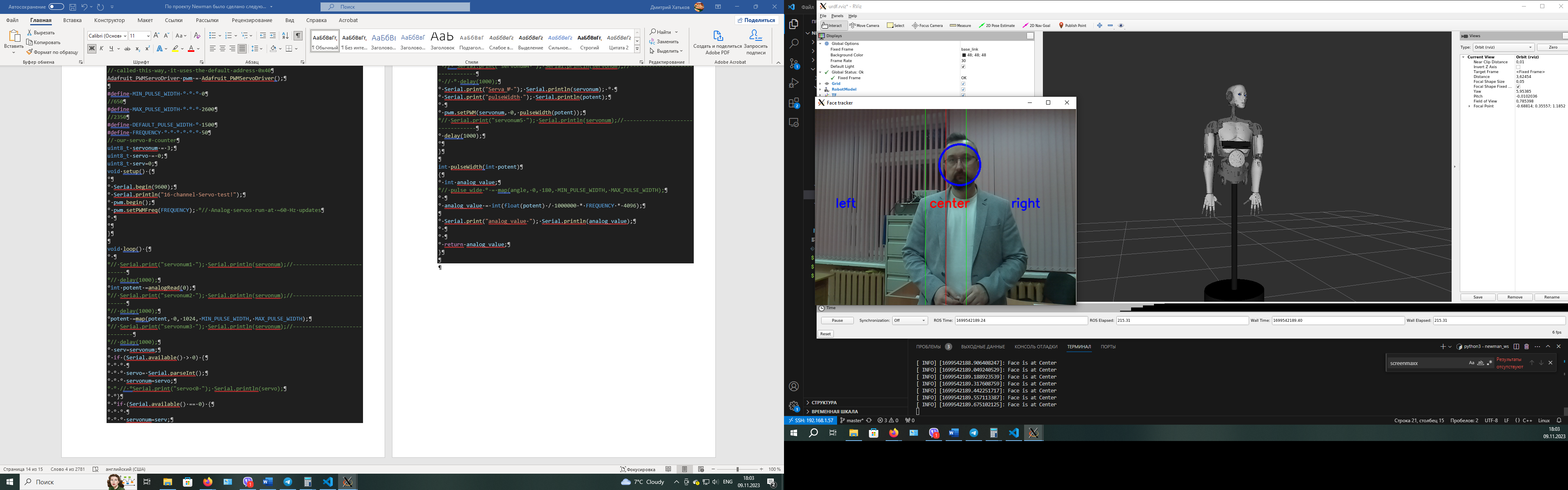
В ходе выполнения проекта был реализован механизм распознавания и трекинга лиц на базе модели антропоморфного робота Newman, в основу механизма были положены алгоритмы распознавания лиц на основе каскадного фильтра Хаара и алгоритмы по предобработке изображений из библиотеки OpenCV. Для Newman на базе ROS был создан цифровой двойник и запущенна мультикомпьютерная сеть ROS. Данный метод реализации механизма распознавания и трекинга лиц, показал субъективно неплохую вероятность распознавания лиц за исключением людей в очках. К недостатку алгоритма можно отнести высокую нагрузку на процессор.

# Список текущих багов и недоработок

Механизм распознавания и трекинга лиц использует до 70% ресурсов процессора. Начат процесс по переносу алгоритма распознавания лиц на GPU но не завершен. Ожидается прирост производительности процессора за счет переключения части расчетов на GPU

Возникают проблемы с трекингом 2х и более лиц. Проблема связана с тем, что алгоритм разработан на трекинг одного лица в кадре. Проблема проявляется в переключении с одного лица на другое с поворотом головы. Возможные пути решения — это поменять алгоритм таким образом, чтобы только одно изображение обводилось геометрической фигурой и соответственно передавались координаты центра только одной этой фигуры

Проблема в прекращении отслеживания лиц в одном из крайних положениях шеи. Возникает из-за постоянного массового появления лиц по краям изображения. Проявляется при отслеживании лиц в толпе. Проблема возникает из-за декремента или инкремента расчетного угла поворота головы, который выходит за значения ±1.57 рад Возможный путь решения — это задать пределы углов поворота головы, которые рассчитываются узлом отслеживания лица.

Рисунок 6 Отслеживание лица

# Литература

Для создания программы отслеживания лица были использованы материалы из следующих источников:

1 Lentin Joseph\\ ROS Robotics Projects chapter 2

2 <https://maker.pro/arduino/tutorial/how-to-control-a-robot-arm-with-ros-and-arduino>

3 <https://maker.pro/ros/tutorial/robotic-arm-simulation-in-robot-operating-system-ros>

4 <https://maker.pro/arduino/tutorial/how-to-use-arduino-with-robot-operating-system-ros>