ИНСТРУКЦИЯ ИСПОЛЬЗОВАНИЯ БИБЛИОТЕКИ

Предварительно подготовить на рабочем месте:

1. Установка библиотеки pthread

Библиотека Pthread предоставляет API-интерфейс для создания и управления потоками в приложении. Библиотека Pthread основана на стандартизированном интерфейсе программирования, который был определен комитетом по выпуску стандартов IEEE в стандарте POSIX 1003.1c. Сторонние фирмы-изготовители придерживаются стандарта POSIX в реализациях, которые именуются библиотеками потоков Pthread или POSIX.

1. Потоки

Компиляция компилятором gcc с ключом: -pthread

Подключение библиотеки:

#include <pthread.h>

Вначале создается один поток, исполняющий функцию main. Этот поток может создавать дочерние потоки, которые начинают выполняться параллельно. Те, в свою очередь, также могут создавать свои дочерние потоки. Все потоки имеют доступ ко всем глобальным переменным программы.

Каждый поток должен быть описан в виде отдельной функции вида:

void *func(void *);

Для каждого потока описывается:

pthread_t pid; // дескриптор потока

pthread_attr_t attr; // атрибуты потока

Инициализация атрибутов потока:

pthread_attr_init(&attr);

pthread_attr_setscope(&attr,PTHREAD_SCOPE_SYSTEM);

PTHREAD_SCOPE_SYSTEM – поток конкурирует за процессор со всеми потоками системы

PTHREAD_SCOPE_PROCESS – поток конкурирует за процессор с потоками, созданными родительским потоком

Создание потока:

pthread_create(&pid,&attr,func,arg);

arg – параметр, передаваемый функции func при запуске

Если поток создан успешно, функция возвращает нуль.

Завершение потока:

pthread_exit(value);

value – возвращаемое значение или NULL

Функция вызывается неявно, если поток просто завершил выполнение.

Ожидание завершения дочернего потока родительским:

pthread_join(pid,value_ptr);

value_ptr – адрес переменной для возвращаемого потоком значения (или NULL).

2. Семафоры

Семафор – особый тип разделяемой переменной, которая обрабатывается двумя неделимыми операциями: P и V. Значения семафора – неотрицательные целые числа. Операция V сигнализирует, что событие произошло: она увеличивает значение семафора на 1. Операция P приостанавливает процесс до момента, когда событие произойдет: она ждет, когда значение семафора станет положительным, и уменьшает его на 1.

Подключение библиотеки:

#include<semaphore.h>

Описание семафора:

sem_t lock;

Инициализация семафора:
sem_init(&lock,SHARED,init_value);
SHARED = 1 – семафор может быть разделяемым между процессами
0 – семафор используют потоки только одного процесса
init_value – начальное значение
Операции над семафором:
sem_wait(&lock); // Операция P: { while (lock==0); lock--; }
sem_post(&lock); // Операция V: { lock++; }
Пример: производитель-потребитель

```
sem_t empty,full;
int data;
int main()
{ ...
  sem_init(&empty,SHARED,1);
  sem_init(&full,SHARED,0);
  ...
}
void *Producer(void *arg)
{ ...
  for (...)
  { sem_wait(&empty);
    data=Expression();
    sem_post(&full);
  }
  ...
}
void *Consumer(void *arg)
{ ...
  for (...)
  { sem_wait(&full);
    Use(data);
    sem_post(&empty);
  }
  ...
}
```

3. Блокировки (мьютексы)

Мьютексы используются для выделения критических интервалов.
Описание мьютекса:
pthread_mutex_t mutex;
Инициализация мьютекса (с атрибутами по умолчанию):
pthread_mutex_init(&mutex,NULL);
Операции над мьютексами:
pthread_mutex_lock(&mutex); // блокирование мьютекса
pthread_mutex_unlock(&mutex); // разблокирование мьютекса
Пример: описание критической секции

```
pthread_mutex_lock(&mutex);
Use(data);
pthread_mutex_unlock(&mutex);
```

4. Условные переменные

Условная переменная используются для приостановки процесса, выполнение которого может продолжаться после выполнения некоторого условия. Об этом ему

сигнализирует другой процесс. С каждой условной переменной связана очередь ожидающих ее процессов.
Описание условной переменной:
pthread_cond_t cond;
Инициализация условной переменной (с атрибутами по умолчанию):
pthread_cond_init(&cond,NULL);
Операции над условными переменными:
pthread_cond_wait(&cond,&mutex); // постановка процесса в очередь ожидания
// (разблокирует mutex)
pthread_cond_signal(&cond); // запуск первого процесса в очереди
pthread_cond_broadcast(&cond); // запуск всех процессов в очереди
Пример: реализация барьера

```
pthread_cond_t go;
pthread_mutex_t barrier;
int n=0,nproc;
void barrier()
{ pthread_mutex_lock(&barrier);
  n++;
  if (n<nproc) pthread_cond_wait(&go,&barrier);
        else { n=0; pthread_cond_broadcast(&go); }
  pthread_mutex_unlock(&barrier);
}
```

Группы операций
Управление потоками
- Мьютексы
- Условные переменные

Управление потоками
Создание и удаление потоков
Создание потока:
int pthread_create (pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);
Завершение потока:
void pthread_exit (void *value_ptr);
Инициализация атрибутов потока:
int pthread_attr_init (pthread_attr_t *attr);
Уничтожение атрибутов потока:
int pthread_attr_destroy (pthread_attr_t *attr);
Завершение потока из другого потока: pthread_cancel
Ожидание завершения потока:
int pthread_join (pthread_t thread, void **value_ptr);
Отсоединение потока:
int pthread_detach (pthread_t thread, **value_ptr);

Разное
Получение идентификатора текущего потока:
pthread_t pthread_self (void);
Определение идентичности идентификоторов потока:
int pthread_equal (pthread_t t1, pthread_t t2);
Выполнение функции однократно в процессе:
int pthread_once (pthread_once_t *once_control, void (*init_routine)(void));

pthread_once_t once_control = PTHREAD_ONCE_INIT;
Переключиться на другой поток (отказаться от текущего кванта времени):
void pthread_yield ();

Мьютексы
Mutual exclusion – взаимное исключение. Используются для взаимоисключающего доступа к ресурсу. Поток может заснуть только при ожидании мьютекса.
Мьютекс: pthread_mutex_t.
Атрибуты мьютекса: pthread_mutexattr_t.

Создание и удаление мьютексов
Инициализация мьютекса:
int pthread_mutex_init (pthread_mutex_t *mutex, pthread_mutexattr_t *attr);
Уничтожение мьютекса:
int pthread_mutex_destroy (pthread_mutex_t *mutex);
Инициализация атрибутов мьютекса:
int pthread_mutexattr_init (pthread_mutexattr_t *attr);
Удаление атрибутов мьютекса:
int pthread_mutexattr_destroy (pthread_mutexattr_t *attr);

Захват и освобождение мьютекса
Захват мьютекса:
int pthread_mutex_lock (pthread_mutex_t *mutex);
Попытка захвата мьютекса:
int pthread_mutex_trylock (pthread_mutex_t *mutex);
Освобождение мьютекса:
int pthread_mutex_unlock (pthread_mutex_t *mutex);

Условные переменные
Позволяют синхронизировать потоки в зависимости от значений переменных. Используются только совместно с мьютексами.
Условная переменная: pthread_cond_t.
Атрибуты условной переменной: pthread_condattr_t.

Создание и удаление условных переменных
Инициализация условной переменной:
int pthread_cond_init (pthread_cond_t *condition, pthread_condattr_t *attr);
Удаление условной переменной:
int pthread_cond_destroy (pthread_cond_t *condition);
Инициализация атрибутов условной переменной:
int pthread_condattr_init (pthread_condattr_t *attr);
Удаление атрибутов условной переменной:
int pthread_condattr_destroy (pthread_condattr_t *attr);

Использование условных переменных
Ожидание события:
int pthread_cond_wait (pthread_cond_t *cond);
int pthread_cond_timedwait(
pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime);
Сигнализация о событии:
int pthread_cond_signal (pthread_cond_t *condition);
int pthread_cond_broadcast (pthread_cond_t *condition);

2. Установка библиотек OpenCV, glog, FreeImage, tclap

Сначала обновляем операционную систему:

Код:

```
sudo apt-get update
sudo apt-get upgrade
```

Потом устанавливаем необходимые библиотеки:

Код:

```
sudo apt-get remove x264 libx264-dev

sudo apt-get install build-essential checkinstall cmake pkg-config yasm
sudo apt-get install git gfortran
sudo apt-get install libjpeg8-dev libjasper-dev libpng12-dev

sudo apt-get install libtiff5-dev

sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libdc1394-22-dev
sudo apt-get install libxine2-dev libv4l-dev
sudo apt-get install libgstreamer0.10-dev libgstreamer-plugins-base0.10-dev
sudo apt-get install qt5-default libgtk2.0-dev libtbb-dev
sudo apt-get install libatlas-base-dev
sudo apt-get install libfaac-dev libmp3lame-dev libtheora-dev
sudo apt-get install libvorbis-dev libxvidcore-dev
sudo apt-get install libopencore-amrnb-dev libopencore-amrwb-dev
sudo apt-get install x264 v4l-utils

sudo apt-get install libprotobuf-dev protobuf-compiler
sudo apt-get install libgoogle-glog-dev libgflags-dev
sudo apt-get install libgphoto2-dev libeigen3-dev libhdf5-dev doxygen
```

Устанавливаем необходимые библиотеки:

Код:

```
sudo apt-get install python-dev python-pip python3-dev python3-pip
sudo -H pip2 install -U pip numpy
sudo -H pip3 install -U pip numpy
```

Теперь начнём установку OpenCV в виртуальную среду. Дело в том, что люди часто устанавливают одни библиотеки, потом вторые, и через некоторое время между ними могут возникать конфликты версий: одним нужны библиотеки одних версий, другим -- других. И всё может перестать работать. Поэтому лучше установить OpenCV в изолированную среду, чтобы она всегда запускалась правильно.

Соответственно, устанавливаем виртуальную среду под названием my:

Код:

```
# Install virtual environment
sudo pip2 install virtualenv virtualenvwrapper
sudo pip3 install virtualenv virtualenvwrapper
echo "# Virtual Environment Wrapper"  >> ~/.bashrc
echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.bashrc
source ~/.bashrc
```

```
#create virtual environment
mkvirtualenv my python3
workon my

pip install numpy scipy matplotlib scikit-image scikit-learn ipython

deactivate
```

Теперь входим в виртуальную среду и активируем её, а потом качаем OPenCV

Код:

```
workon my
git clone https://github.com/opencv/opencv.git
cd opencv
git checkout 3.3.1
cd ..

git clone https://github.com/opencv/opencv_contrib.git
cd opencv_contrib
git checkout 3.3.1
cd ..
```

Создаём каталог build и переходим в него

Код:

```
cd opencv
mkdir build
cd build
```

Запускаем CMAKE:

Код:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
      -D CMAKE_INSTALL_PREFIX=/usr/local \
      -D INSTALL_C_EXAMPLES=ON \
      -D INSTALL_PYTHON_EXAMPLES=ON \
      -D WITH_TBB=ON \
      -D WITH_V4L=ON \
      -D WITH_QT=ON \
      -D WITH_OPENGL=ON \
      -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules \
      -D BUILD_EXAMPLES=ON ..
```

Компилируем и устанавливаем: для установки используем четыре процессора. Соответственно, если есть желание использовать шесть процессоров, то можно написать -j6.

Код:

```
make -j4
sudo make install
sudo sh -c 'echo "/usr/local/lib" >> /etc/ld.so.conf.d/opencv.conf'
sudo ldconfig
```

Следующий момент -- нам нужно найти, где у нас лежит файл cv2*.so, а потом создать для него ярлык.

Пишем команду и запоминаем, где лежит этот файл:

Код:

```
find /usr/local/lib/ -type f -name "cv2*.so"
```

Теперь создаём ярлык: но нужно заменить следующий путь "/usr/local/lib/python3.6/dist-packages/cv2.cpython-36m-x86_64-linux-gnu.so" на свой свой, где лежит cv2*.so файл

Код:

```
cd ~/.virtualenvs/my/lib/python3.6/site-packages
ln -s /usr/local/lib/python3.6/dist-packages/cv2.cpython-36m-x86_64-linux-gnu.so cv2.so
```

Проверяем, что всё установлено правильно. В терминале активируем виртуальную среду с помощью команды workon my, а затем запускаем python

Код:

```
workon my
python
```

Затем набираем команду:

PythonВыделить код

```
1 import cv2
```

Инструкция использования библиотеки

1) Открыть терминал

В Ubuntu консоль запускается при загрузке системы. Терминал — это тоже консоль, но уже в графической оболочке. Его можно запустить, набрав слово Терминал в поисковой строке ОС, или через комбинацию клавиш Ctrl+Alt+T.

В общем виде в Ubuntu команды имеют такой вид:

программа -ключ значение

Программа — это сам исполняемый файл. Другими словами, это программа, которая будет выполняться по команде.

Ключ — обычно у каждой программы свой набор ключей. Их можно найти в мануале к программе.

Значение — параметры программы: цифры, буквы, символы, переменные.

Напомним, что для выполнения команды нужно ввести её в командную строку — Ubuntu console или эмулирующий работу консоли терминал.

Рассмотрим основные команды консоли Ubuntu:

- **sudo**

Промежуточная команда **sudo** (SuperUser DO — суперпользователь) позволяет запускать программы от имени администратора или root-пользователя.

Вы можете добавить sudo перед любой командой, чтобы запустить её от имени суперпользователя.

- **apt-get**

Команда **apt-get** используется для работы с программными пакетами для установки программных пакетов (sudo apt-get install имя-пакета), обновления репозитория с пакетами (sudo apt-get update) и обновления пакетов, которые установлены в систему (sudo apt-get upgrade).

- **pwd**

Команда **pwd** (print working directory — вывести рабочую директорию) показывает полное имя рабочей директории, в которой вы находитесь.

- **ls**

Команда **ls** (list — список) выводит все файлы во всех папках рабочей директории.
С помощью ls -a можно вывести и скрытые файлы.

- **cd**

Команда **cd** (change directory — изменить директорию) позволяет перейти в другую директорию.
Можно ввести как полный путь до папки, так и её название. Например, чтобы попасть в папку **Files**, лежащую в директории **/user/home/Files**, введите cd Files или cd /user/home/Files.
Чтобы попасть в корневую директорию, введите cd /.

- **cp**

Команда **cp** (copy — копировать) копирует файл.
Например, cp file1 file2 скопирует содержимого файла file1 в file2.
Команда cp file /home/files скопирует файл с названием **file** в директорию **/home/files**.

- **mv**

Команда **mv** (move — переместить) помогает перемещать файлы.
Также с помощью mv можно переименовывать файлы. Например, у нас есть файл **file.txt**. С помощью команды mv file.txt new_file.txt мы можем перенести его в ту же директорию, но у файла уже будет новое название **new_file.txt**.

- **rm**

Команда **rm** (remove — удалить) удаляет файлы и каталоги.
Так, команда rm file.txt удалит текстовый файл с названием **file**, а команда rm -r Files удалит директорию **Files** со всеми содержащимися в ней файлами.

- **mkdir**

С помощью **mkdir** (make directory — создать директорию) можно создать новую директорию.
Так, команда mkdir directory создаст новую директорию с именем **directory** в текущей рабочей директории.

- **man**

Команда **man** (manual — мануал) открывает справочные страницы с подробной информацией о команде.
Введите man, а затем через пробел название команды, о которой вы хотите узнать подробнее. Например, man cp выведет справочную страницу о команде **cp**.

2) Перейти в удобный каталог: cd testing

3) Склонировать репозиторий: git clone https://github.com/ChervyakovLM/FaceMetric.git

Для проверки **заданной библиотеки биометрической верификации** необходимо реализовать класс наследник от FACEAPITEST: Interface.

```
class Interface {
public:
    virtual ~Interface() {}

    virtual ReturnStatus
    initialize(const std::string &configDir) = 0;

    virtual ReturnStatus
    createTemplate(
        const Multiface &faces,
        TemplateRole role,
        std::vector<uint8_t> &templ,
```

```
        std::vector<EyePair> &eyeCoordinates,
        std::vector<double> &quality) = 0;

    virtual ReturnStatus
    matchTemplates(
        const std::vector<uint8_t> &verifTemplate,
        const std::vector<uint8_t> &initTemplate,
        double &similarity) = 0;

    virtual ReturnStatus
    train(
        const std::string &configDir,
        const std::string &trainedConfigDir) = 0;

    static std::shared_ptr<Interface>
    getImplementation();
};
```

Класс наследник должен содержать реализацию следующих функций:

- initialize - инициализация алгоритма вычисления биометрических шаблонов;
- createTemplate - вычисление шаблона;
- matchTemplates - сравнение шаблонов;
- train - донастройка алгоритма вычисления биометричесекийх шаблонов;
- getImplementation – получение указателя на реализацию.

Пример реализации класса наследника приведён в файлах face_api_example_V.h и face_api_example_V.cpp, содержащиеся в каталогах include и src соответсвенно.

Для проверки **заданной библиотеки биометрической идентификации** необходимо реализовать класс наследник от FACEAPITEST: IdentInterface.

```
class IdentInterface {
public:
    virtual ~IdentInterface() {}

    virtual ReturnStatus
    initializeTemplateCreation(
        const std::string &configDir,
        TemplateRole role) = 0;

    virtual ReturnStatus
    createTemplate(
        const Multiface &faces,
        TemplateRole role,
        std::vector<uint8_t> &templ,
        std::vector<EyePair> &eyeCoordinates) = 0;

    virtual ReturnStatus
    finalizeInit(
          const std::string &configDir,
        const std::string &initDir,
        const std::string &edbName,
        const std::string &edbManifestName) = 0;
```

```
    virtual ReturnStatus
    initializeIdentification(
        const std::string &configDir,
        const std::string &initDir) = 0;

    virtual ReturnStatus
    identifyTemplate(
        const std::vector<uint8_t> &idTemplate,
        const uint32_t candidateListLength,
        std::vector<Candidate> &candidateList,
        bool &decision) = 0;

    virtual ReturnStatus
    galleryInsertID(
        const std::vector<uint8_t> &templ,
        const std::string &id) = 0;

    virtual ReturnStatus
    galleryDeleteID(
        const std::string &id) = 0;

    static std::shared_ptr<IdentInterface>
    getImplementation();
};
```

Класс наследник должен содержать реализацию следующих функций:

initializeTemplateCreation - инициализация алгоритма вычисления биометрических шаблонов;

createTemplate - вычисление шаблона;

finalizeInit - создание индекса из всех шаблонов;

initializeIdentification - инициализация алгоритма поиска по индексу;

identifyTemplate – поиск по индексу;

galleryInsertID - добавление шаблона в индеск;

galleryDeleteID - удаление шаблона из индекса;

getImplementation – получение указателя на реализацию.

Пример реализации класса наследника приведён в файлах face_api_example_I.h и face_api_example_I.cpp, содержащиеся в каталогах include и src соответственно.