Bo’limlar ro’yhati :

1-bo‘lim: OpenCV’ni sozlash

Siz ushbu kitobni qo‘lga olganingizga ko‘ra, ehtimol, OpenCV haqida allaqachon ma'lumotga egasiz. Balki siz ilmiy-fantastika filmlarida ko‘rganingiz kabi yuzni aniqlash funksiyalari haqida eshitgansiz va qiziqib qolgandirsiz. Agar shunday bo‘lsa, siz to‘g‘ri tanlov qildingiz.

OpenCV — **"Open Source Computer Vision"** (Ochiq Kodli Kompyuter Ko‘rish) degan ma’noni anglatadi. Bu bepul kompyuter ko‘rish kutubxonasi bo‘lib, tasvirlar va videolarni qayta ishlash orqali turli vazifalarni bajarish imkonini beradi — oddiygina veb-kamera tasvirini aks ettirishdan tortib, robotga haqiqiy hayotdagi obyektlarni tanib olishni o‘rgatishgacha.

Ushbu kitob orqali siz OpenCV va Python dasturlash tilining ajoyib imkoniyatlaridan foydalanishni o‘rganasiz. Python — juda qulay dasturlash tili bo‘lib, oson o‘rganiladi va kuchli imkoniyatlarga ega. Ushbu bo‘limda siz Python 2.7, OpenCV va boshqa zaruriy kutubxonalarni qanday o‘rnatish va sozlashni o‘rganasiz. Shuningdek, OpenCV’ning Python uchun mavjud bo‘lgan namunaviy skriptlari va hujjatlarini ko‘rib chiqamiz.

Eslatma: Agar siz o‘rnatish jarayonini o‘tkazib yuborib, to‘g‘ridan-to‘g‘ri amaliy ishga o‘tishni istasangiz, muallif tomonidan tayyorlangan virtual mashinani yuklab olishingiz mumkin: http://techfort.github.io/pycv/

Bu virtual mashina **VirtualBox** dasturida ishlaydi va **Ubuntu Linux 14.04** tizimiga asoslangan. Unda OpenCV va barcha zaruriy dasturiy ta’minot allaqachon o‘rnatilgan. Ushbu virtual mashina kamida **2 GB operativ xotira (RAM)** talab qiladi, biroq samarali ishlashi uchun 4 GB yoki undan ko‘prog‘ini ajratish tavsiya etiladi.

OpenCV uchun zaruriy kutubxonalar

Ushbu bo‘limda biz OpenCV’ni ishlatish uchun zaruriy kutubxonalarni ko‘rib chiqamiz:

- **NumPy** – OpenCV’ning Python bog‘lamalari uchun asosiy kutubxona. Bu kutubxona katta o‘lchamdagi massivlarni samarali qayta ishlash imkonini beradi.
- **SciPy** – Ilmiy hisob-kitoblar va OpenCV tasvirlarini qayta ishlash uchun foydali kutubxona.
- **OpenNI** (ixtiyoriy) – Chuqurlik kameralari, masalan, Asus XtionPRO’ni qo‘llab-quvvatlash uchun ishlatiladi.
- **SensorKinect** (ixtiyoriy) – OpenNI uchun plagindir va Microsoft Kinect qurilmasi bilan ishlash imkonini beradi.

Muhim: OpenNI va SensorKinect faqat **4-bo‘lim: Chuqurlikni baholash va segmentatsiya** mavzusida ishlatiladi. Agar siz chuqurlik kameralaridan foydalanmoqchi bo‘lmasangiz, ushbu kutubxonalarni o‘rnatish shart emas.

OpenCV’ni o‘rnatish uchun mos dasturlar

OpenCV’ni turli operatsion tizimlarga o‘rnatishning bir necha usullari mavjud. Quyida asosiy operatsion tizimlar uchun eng mos usullar keltirilgan:

- **Windows**
  - Tayyor o‘rnatish fayllaridan foydalanish
  - CMake va kompilatorlar bilan qo‘lda yig‘ish
- **Mac OS X**
  - MacPorts orqali o‘rnatish
  - Homebrew orqali o‘rnatish
- **Linux (Ubuntu va shunga o‘xshash tizimlar)**
  - Ubuntu omboridan tayyor paketlarni o‘rnatish
  - Manba koddan OpenCV’ni yig‘ish

Agar siz Windows yoki Mac OS X tizimida ishlasangiz, tayyor paketlardan foydalanish eng qulay usul bo‘lishi mumkin. Linux tizimlarida esa OpenCV’ni manba koddan yig‘ish tez-tez ishlatiladigan usullardan biridir.

Windows tizimida OpenCV’ni o‘rnatish

Windows tizimi Python dasturlash tilini o‘z ichiga olmaydi, ammo tayyor o‘rnatish fayllari orqali OpenCV va unga bog‘liq kutubxonalarni o‘rnatish mumkin. Quyidagi usullardan foydalanish mumkin:

1. Tayyor o‘rnatish fayllaridan foydalanish (chuqurlik kameralari qo‘llab-quvvatlanmaydi)

Agar sizga OpenCV’ni tezda o‘rnatish kerak bo‘lsa va chuqurlik kameralari bilan ishlash shart bo‘lmasa, quyidagi bosqichlarni bajaring:

1. **Python 2.7.9** versiyasini quyidagi havoladan yuklab oling va o‘rnating: https://www.python.org/ftp/python/2.7.9/python-2.7.9.amd64.msi
2. **NumPy va SciPy kutubxonalarini o‘rnating:**
   - NumPy: http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy
   - SciPy: http://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy
3. OpenCV 3.0.0 faylini yuklab oling va ZIP faylni oching: https://github.com/Itseez/opencv
4. **cv2.pyd faylini Python kutubxonalar papkasiga ko‘chiring:**
   - `opencv/build/python/2.7/cv2.pyd` faylini `C:\Python2.7\Lib\site-packages\` papkasiga joylashtiring.
5. Agar Python skriptlari OpenCV’ni topishi uchun `PATH` muhit o‘zgaruvchilarini o‘zgartiring:
   - `C:\Python2.7` manzilini `PATH`ga qo‘shing va kompyuterni qayta yuklang.

Windows tizimida OpenCV’ni o‘rnatish va sozlash (davomi)

Agar tayyor o‘rnatish fayllari orqali OpenCV’ni o‘rnatish sizga yetarli bo‘lmasa yoki chuqurlik kameralarini qo‘llab-quvvatlashni xohlasangiz, OpenCV’ni **manba koddan yig‘ish (build qilish)** usulidan foydalanishingiz mumkin. Buning uchun **CMake va kompilatorlar** kerak bo‘ladi.

---

2. CMake va kompilatorlar yordamida OpenCV’ni manba koddan yig‘ish

Windows tizimi sukut bo‘yicha **C++ kompilatorlari yoki CMake** dasturini o‘z ichiga olmaydi. Shu sababli, ularni qo‘shimcha ravishda o‘rnatish lozim. Agar chuqurlik kameralarini (masalan, **Kinect**) qo‘llab-quvvatlashni istasangiz, **OpenNI va SensorKinect** kutubxonalarini ham o‘rnatishingiz kerak bo‘ladi.

Muhim eslatma: Agar siz OpenCV’ni chuqurlik kameralarisiz ishlatmoqchi bo‘lsangiz, **OpenNI va SensorKinect** o‘rnatish shart emas.

2.1. CMake va kompilatorlarni o‘rnatish

1. **CMake 3.1.2 versiyasini yuklab oling va o‘rnating:**
   - Rasmiy sayt: https://cmake.org/download/
   - O‘rnatish vaqtida "Add CMake to the system PATH" (CMake’ni tizim PATH’iga qo‘shish) opsiyasini tanlang.
2. **Microsoft Visual Studio 2013 yoki MinGW kompilatorlarini o‘rnating:**
   - Agar **Visual Studio** ishlatmoqchi bo‘lsangiz, rasmiy saytga kiring: https://visualstudio.microsoft.com/
   - Agar **MinGW** ishlatmoqchi bo‘lsangiz, quyidagi havoladan yuklab oling: https://sourceforge.net/projects/mingw/
   - O‘rnatish vaqtida **C++ compiler (C++ kompilatori)** ni tanlang.

Muhim: MinGW dasturi uchun **PATH muhit o‘zgaruvchilariga** quyidagi manzilni qo‘shing: `C:\MinGW\bin`

3. **OpenNI va SensorKinect’ni o‘rnatish (agar chuqurlik kameralaridan foydalanmoqchi bo‘lsangiz):**
   - OpenNI yuklab olish: https://github.com/OpenNI/OpenNI
   - SensorKinect yuklab olish: https://github.com/avin2/SensorKinect

---

2.2. OpenCV’ni manba koddan yig‘ish

Quyidagi bosqichlarni bajarish orqali OpenCV’ni to‘g‘ridan-to‘g‘ri manba koddan yig‘ishingiz mumkin.

1. **OpenCV 3.0.0 manba kodini yuklab oling va oching:**
   - Yuklab olish: https://github.com/Itseez/opencv
   - ZIP faylni oching va masalan, `C:\opencv` papkasiga saqlang.
2. **Yangi build papka yaratish:**
   - Buyruq qatori (Command Prompt) yoki **PowerShell** ni oching va quyidagi buyruqlarni kiriting:

```
sh
КопироватьРедактировать
mkdir C:\opencv\build
cd C:\opencv\build
```

3. **CMake’ni ishga tushiring va konfiguratsiya qiling:**

- CMake GUI dasturini oching.
- "Where is the source code?" qatorida OpenCV manba kod papkasini (C:\opencv) tanlang.
- "Where to build the binaries?" qatorida **build papkani** (C:\opencv\build) tanlang.
- "Configure" tugmasini bosing va **Visual Studio 12 2013 Win64** (yoki ishlatayotgan kompilatoringizni) tanlang.
- "Generate" tugmasini bosing.

4. **Visual Studio yoki MinGW yordamida OpenCV'ni kompilyatsiya qilish:**
   - Agar **Visual Studio** ishlatayotgan bo'lsangiz:
     - C:\opencv\build\OpenCV.sln faylini oching.
     - "ALL_BUILD" loyihasini tanlang va "Build" tugmasini bosing.
   - Agar **MinGW** ishlatayotgan bo'lsangiz, buyruq qatoriga quyidagilarni kiriting:

```
sh
КопироватьРедактировать
mingw32-make
```

5. **Tizimga OpenCV kutubxonalarini qo'shish:**
   - "cv2.pyd" faylini quyidagi manzilga nusxalash:

```
sh
КопироватьРедактировать
copy C:\opencv\build\lib\Release\cv2.pyd C:\Python2.7\Lib\site-packages
```

   - **PATH muhit o'zgaruvchisiga OpenCV kutubxonalarini qo'shing:**
     - C:\opencv\build\bin\Release manzilini tizimning "Environment Variables" qismiga qo'shing.

Tizimni qayta yuklang va OpenCV'ni test qilish uchun quyidagi buyruqlarni Python'da bajaring:

```
python
КопироватьРедактировать
import cv2
print(cv2.__version__)
```

Agar OpenCV versiyasi ekranga chiqsa, demak, o'rnatish muvaffaqiyatli bajarilgan.

Mac OS X tizimida OpenCV'ni o'rnatish

Agar siz **Mac OS X** foydalanuvchisi bo'lsangiz, OpenCV'ni **MacPorts yoki Homebrew** yordamida o'rnatishingiz mumkin. Mac'ning ayrim versiyalarida Python 2.7 o'rnatilgan bo'ladi, ammo agar siz yangilangan versiyani ishlatmoqchi bo'lsangiz, uni yuklab olishingiz mumkin.

---

1. MacPorts yordamida OpenCV'ni o'rnatish

MacPorts – Mac operatsion tizimi uchun maxsus paket menejeri bo'lib, OpenCV'ni tez va oson o'rnatish imkonini beradi.

1. **MacPorts’ni yuklab oling va o‘rnating:**
   - https://www.macports.org/install.php
   - O‘rnatish yakunlangandan so‘ng, terminalni oching va quyidagilarni bajaring:

```
sh
КопироватьРедактировать
sudo port selfupdate
sudo port install opencv +python27
```

2. **NumPy va SciPy kutubxonalarini o‘rnatish:**

```
sh
КопироватьРедактировать
sudo port install py27-numpy py27-scipy
```

Mac OS X tizimida OpenCV’ni o‘rnatish (davomi)

Agar siz **MacPorts** o‘rniga **Homebrew** paket menejeridan foydalanmoqchi bo‘lsangiz, OpenCV’ni ushbu usul orqali ham o‘rnatishingiz mumkin.

---

2. Homebrew yordamida OpenCV’ni o‘rnatish (chuqurlik kameralarisiz)

Homebrew – Mac OS X uchun mashhur paket menejeri bo‘lib, u turli kutubxonalarni oson o‘rnatish va yangilash imkonini beradi.

Homebrew’ni o‘rnatish

Agar siz Homebrew’ni hali o‘rnatmagan bo‘lsangiz, terminalni oching va quyidagi buyruqni kiriting:

```
sh
КопироватьРедактировать
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Ushbu buyruq Homebrew’ni yuklab olib, tizimga o‘rnatadi. O‘rnatish tugagandan so‘ng, Homebrew’ning to‘g‘ri ishlayotganini tekshirish uchun quyidagilarni bajaring:

```
sh
КопироватьРедактировать
brew doctor
```

Agar buyrug‘ingiz **"Your system is ready to brew"** degan javobni chiqarsa, Homebrew tayyor.

Python va NumPy kutubxonalarini o‘rnatish

Homebrew yordamida Python va NumPy kutubxonalarini quyidagicha o‘rnatish mumkin:

```
sh
КопироватьРедактировать
brew install python
pip3 install numpy
```

Diqqat! Python o‘rnatilganidan so‘ng, u `python3` sifatida ishlaydi. Agar `python` deb yozsangiz, eski tizim Python versiyasi ishga tushishi mumkin.

OpenCV’ni Homebrew orqali o‘rnatish

Endi OpenCV’ni quyidagi buyruq bilan o‘rnatamiz:

```
sh
КопироватьРедактировать
brew install opencv
```

Bu buyruq OpenCV’ni barcha standart komponentlari bilan yuklab oladi va tizimga o‘rnatadi.

O‘rnatilgan OpenCV’ni tekshirish

O‘rnatish muvaffaqiyatli yakunlangandan so‘ng, OpenCV’ni tekshirish uchun quyidagi Python buyruqlarini bajaring:

```
python
КопироватьРедактировать
import cv2
print(cv2.__version__)
```

Agar OpenCV versiyasi ekranda paydo bo‘lsa, demak, o‘rnatish muvaffaqiyatli yakunlandi.

Muhim eslatma: Homebrew orqali o‘rnatilgan OpenCV **chuqurlik kameralarini (masalan, Kinect) qo‘llab-quvvatlamaydi**. Agar chuqurlik kameralaridan foydalanmoqchi bo‘lsangiz, **MacPorts yordamida o‘rnatish** tavsiya etiladi.

Linux (Ubuntu) tizimida OpenCV’ni o‘rnatish

Agar siz **Linux (Ubuntu yoki unga asoslangan tizimlar)** foydalanuvchisi bo‘lsangiz, OpenCV’ni o‘rnatishning ikkita asosiy usuli mavjud:

1. **Ubuntu rasmiy omboridan o‘rnatish** – bu eng oson usul, ammo OpenCV’ning eski versiyasi o‘rnatiladi.
2. **Manba koddan OpenCV’ni yig‘ish (build qilish)** – bu usul OpenCV’ning so‘nggi versiyasini o‘rnatish imkonini beradi.

---

1. Ubuntu rasmiy omboridan OpenCV’ni o‘rnatish (chuqurlik kameralari qo‘llab-quvvatlanmaydi)

Ubuntu tizimida OpenCV va unga bog‘liq kutubxonalarni o‘rnatish juda oson. Terminalda quyidagi buyruqlarni bajaring:

```
sh
КопироватьРедактировать
sudo apt update
sudo apt install python3-opencv
```

Bu buyruq orqali OpenCV tizimga o‘rnatiladi va Python 3 muhitida ishlaydi. O‘rnatish tugagandan so‘ng, quyidagi buyruq yordamida OpenCV’ning to‘g‘ri ishlayotganini tekshirishingiz mumkin:

```
python
КопироватьРедактировать
import cv2
print(cv2.__version__)
```

Diqqat! Ubuntu omborida OpenCV’ning **eski versiyasi (odatda 2.x yoki 3.x)** bo‘lishi mumkin. Agar siz **so‘nggi OpenCV versiyasini** o‘rnatmoqchi bo‘lsangiz, **manba koddan yig‘ish** usulidan foydalaning.

---

2. Manba koddan OpenCV’ni o‘rnatish (chuqurlik kameralarini qo‘llab-quvvatlash uchun tavsiya etiladi)

Agar siz OpenCV’ning eng yangi versiyasini yoki chuqurlik kameralarini ishlatmoqchi bo‘lsangiz, uni manba koddan yig‘ishingiz kerak bo‘ladi.

2.1. Zaruriy kutubxonalarni o‘rnatish

Avval OpenCV’ni yig‘ish uchun kerak bo‘ladigan kutubxonalarni o‘rnating:

```
sh
КопироватьРедактировать
sudo apt update
sudo apt install build-essential cmake git libgtk2.0-dev pkg-config \
libavcodec-dev libavformat-dev libswscale-dev \
python3-dev python3-numpy libtbb2 libtbb-dev \
libjpeg-dev libpng-dev libtiff-dev libdc1394-22-dev
```

Ushbu buyruq **kompilyatsiya vositalari, tasvir va video kodlovchilar** hamda **Python kutubxonalari** ni o‘rnatadi.

2.2. OpenCV manba kodini yuklab olish

Keyingi bosqichda OpenCV’ning rasmiy **GitHub** sahifasidan eng so‘nggi versiyasini yuklab olamiz:

```
sh
КопироватьРедактировать
git clone https://github.com/opencv/opencv.git
git clone https://github.com/opencv/opencv_contrib.git
```

Bu buyruqlar **OpenCV asosiy kodini va qo‘shimcha modullarini** yuklab oladi.

2.3. OpenCV’ni yig‘ish va o‘rnatish

1. **Yangi build papka yarating va unga o‘ting:**

   ```
   sh
   КопироватьРедактировать
   mkdir -p opencv/build
   ```

```
cd opencv/build
```

2. **CMake yordamida OpenCV’ni sozlash:**

```
sh
КопироватьРедактировать
cmake -D CMAKE_BUILD_TYPE=Release \
      -D CMAKE_INSTALL_PREFIX=/usr/local \
      -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules ..
```

3. **Kompilyatsiya jarayonini boshlash:**

```
sh
КопироватьРедактировать
make -j$(nproc)
```

Bu jarayon biroz vaqt talab qilishi mumkin (taxminan 30-40 daqiqa).

4. **O‘rnatishni yakunlash:**

```
sh
КопироватьРедактировать
sudo make install
sudo ldconfig
```

2.4. O‘rnatilgan OpenCV’ni tekshirish

OpenCV o‘rnatilganligini tekshirish uchun quyidagi Python buyruqlarini bajaring:

```
python
КопироватьРедактировать
import cv2
print(cv2.__version__)
```

Agar OpenCV versiyasi ekranda ko‘rsatilsa, demak, o‘rnatish muvaffaqiyatli yakunlandi.

OpenCV’ning qo‘shimcha (contrib) modullarini o‘rnatish

OpenCV 3.x versiyasidan boshlab, ba’zi funksiyalar asosiy OpenCV kutubxonasidan ajratilgan va alohida **opencv_contrib** moduli sifatida saqlangan. Ushbu modullarda **yuzni tanib olish, obyektlarni aniqlash, chuqurlik xaritalari bilan ishlash** va boshqa ilg‘or funksiyalar mavjud.

1. OpenCV contrib modullarini yuklab olish

Agar siz OpenCV’ni manba koddan o‘rnatayotgan bo‘lsangiz, **opencv_contrib** modullarini ham yuklab olishingiz kerak. Terminalda quyidagi buyruqlarni bajaring:

```
sh
КопироватьРедактировать
git clone https://github.com/opencv/opencv_contrib.git
```

Bu buyruq OpenCV’ning rasmiy **GitHub** sahifasidan qo‘shimcha modullarni yuklab oladi.

Muhim! Ushbu modullarni OpenCV asosiy kutubxonasini yig‘ish jarayonida faollashtirish kerak bo‘ladi.

2. OpenCV’ni qo‘shimcha modullar bilan birga yig‘ish

Agar siz OpenCV’ni oldinroq manba koddan o‘rnatgan bo‘lsangiz, uni qayta yig‘ib, **opencv_contrib** modullarini faollashtirishingiz kerak.

1. **OpenCV manba kod papkasiga o‘ting va yangi build katalog yarating:**

```
sh
КопироватьРедактировать
cd opencv
mkdir -p build
cd build
```

2. **CMake yordamida OpenCV’ni sozlash (opencv_contrib modullari bilan):**

```
sh
КопироватьРедактировать
cmake -D CMAKE_BUILD_TYPE=Release \
      -D CMAKE_INSTALL_PREFIX=/usr/local \
      -D OPENCV_EXTRA_MODULES_PATH=../opencv_contrib/modules ..
```

3. **OpenCV’ni qayta kompilyatsiya qilish:**

```
sh
КопироватьРедактировать
make -j$(nproc)
```

4. **OpenCV’ni o‘rnatish:**

```
sh
КопироватьРедактировать
sudo make install
sudo ldconfig
```

Eslatma: `"OPENCV_EXTRA_MODULES_PATH"` parametri orqali **opencv_contrib** modullarini faollashtirish muhim. Agar bu parametr qo‘shilmasa, ushbu modullar OpenCV’ga kiritilmaydi.

---

3. O‘rnatilgan OpenCV’ni tekshirish

OpenCV’ning yangi modullarini ishlayotganini tekshirish uchun quyidagi buyruqlarni Python muhitida bajaring:

```
python
КопироватьРедактировать
import cv2
print(cv2.getBuildInformation())
```

Agar chiqishda `"Contrib Modules"` qatorida `"YES"` yoki `"Enabled"` so‘zi chiqsa, demak, qo‘shimcha modullar to‘g‘ri o‘rnatilgan.

---

4. OpenCV’ning Contrib modullaridan foydalanish

Agar OpenCV’ning qo‘shimcha modullari muvaffaqiyatli o‘rnatilgan bo‘lsa, siz endi quyidagi ilg‘or funksiyalardan foydalanishingiz mumkin:

4.1. Yuzni tanib olish (Face Recognition)

```
python
КопироватьРедактировать
import cv2
recognizer = cv2.face.LBPHFaceRecognizer_create()
print("LBPH Face Recognizer ishlamoqda!")
```

Muhim! `"cv2.face"` moduli **opencv_contrib** tarkibiga kiradi, shuning uchun uni ishlatish uchun OpenCV’ni ushbu modullar bilan birga o‘rnatgan bo‘lishingiz shart.

4.2. Yuzni aniqlash (Haar Cascade)

```
python
КопироватьРедактировать
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
```

Bu kod OpenCV ichida tayyor bo‘lgan yuzni aniqlash modellaridan foydalanadi.

OpenCV’ni turli platformalarda ishlatish

OpenCV hozirda Windows, Mac OS X va Linux tizimlarida ishlashi mumkin. Quyida har bir platforma uchun qo‘shimcha tavsiyalar berilgan.

Windows tizimida OpenCV’ni ishlatish

Agar siz Windows tizimida OpenCV o‘rnatgan bo‘lsangiz, **cv2.pyd** kutubxonasining to‘g‘ri yo‘lda joylashganligini tekshiring:

```
sh
КопироватьРедактировать
C:\Python3\Lib\site-packages\cv2.pyd
```

Agar OpenCV yuklanmasa, tizim muhit o‘zgaruvchilari (`Environment Variables`) bo‘limida `"PATH"` ichiga `"C:\opencv\build\bin"` manzilini qo‘shing.

Mac OS X tizimida OpenCV’ni ishlatish

Agar siz OpenCV’ni **Homebrew** orqali o‘rnatgan bo‘lsangiz, u avtomatik ravishda **/usr/local/lib** katalogiga o‘rnatiladi. Biroq, ba’zan Python’ning OpenCV’ni topa olmasligi mumkin. Buni hal qilish uchun quyidagi buyruqni bajaring:

```
sh
КопироватьРедактировать
echo 'export PYTHONPATH=/usr/local/lib/python3.9/site-packages:$PYTHONPATH'
>> ~/.bash_profile
source ~/.bash_profile
```

Bu OpenCV’ning Python kutubxonalar yo‘lini to‘g‘ri sozlaydi.

Linux (Ubuntu) tizimida OpenCV’ni ishlatish

Ubuntu tizimida OpenCV’ni to‘g‘ri ishlashini ta’minlash uchun `"LD_LIBRARY_PATH"` muhit o‘zgaruvchisini sozlash muhim:

```
sh
КопироватьРедактировать
echo 'export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH' >> ~/.bashrc
source ~/.bashrc
```

Bu tizimga OpenCV kutubxonalarini yuklashga yordam beradi.

Xulosa

Biz OpenCV’ni turli platformalarda o‘rnatish va sozlashni ko‘rib chiqdik. Endi siz OpenCV kutubxonasidan to‘liq foydalanishingiz mumkin:
✅ **OpenCV asosiy kutubxonasi o‘rnatildi**
✅ **Contrib modullari faollashtirildi**
✅ **Platformaga mos sozlamalar bajarildi**

Keyingi bosqich **OpenCV bilan amaliy ishlash**, ya’ni tasvirlar va videolarni qayta ishlash bo‘ladi.

2-BO‘LIM: FAYLLAR, KAMERALAR VA GUI INTERFEYSLARINI BOSHQARISH

OpenCV tasvirlarni, videolarni va kameradan kadrlarni qayta ishlash uchun kuchli vositalarni taqdim etadi. Ushbu bo‘limda biz quyidagi asosiy mavzularni ko‘rib chiqamiz:

✅ Rasm fayllari bilan ishlash
✅ Video fayllarni yuklash va saqlash
✅ Kameradan real vaqtda tasvir olish
✅ OpenCV yordamida GUI (grafik foydalanuvchi interfeysi) oynalarini boshqarish

---

1. Rasm fayllari bilan ishlash

OpenCV rasm fayllarini yuklash, o‘zgartirish va saqlash imkonini beradi.

1.1. Rasmni yuklash

Quyidagi kod yordamida OpenCV orqali rasmni yuklash mumkin:

```
python
КопироватьРедактировать
import cv2

# Rasmni yuklash
image = cv2.imread("rasm.jpg")

# Rasmni ekranga chiqarish
cv2.imshow("Tasvir", image)

# Klaviaturadan tugma bosilishini kutish
cv2.waitKey(0)

# Barcha oynalarni yopish
cv2.destroyAllWindows()
```

Kod tushuntirishi:

- `cv2.imread("rasm.jpg")` – `"rasm.jpg"` faylini yuklaydi.
- `cv2.imshow("Tasvir", image)` – rasmni ekranga chiqaradi.
- `cv2.waitKey(0)` – tugma bosilmaguncha oynani ochiq saqlaydi.
- `cv2.destroyAllWindows()` – barcha oynalarni yopadi.

Muhim! Agar `"rasm.jpg"` mavjud bo‘lmasa yoki noto‘g‘ri yo‘l ko‘rsatilgan bo‘lsa, OpenCV `"None"` qiymatini qaytaradi.

1.2. Rasmni kulrang (grayscale) formatga o‘tkazish

OpenCV rasmni **BGR** (ko‘k, yashil, qizil) formatda yuklaydi. Agar uni kulrang formatga o‘tkazish kerak bo‘lsa:

```
python
КопироватьРедактировать
# Rasmni kulrang formatga o‘tkazish
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Kulrang tasvir", gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Asl rasm kulrang formatda aks etadi.

1.3. Rasmni saqlash

Rasmni OpenCV orqali saqlash uchun `cv2.imwrite()` funksiyasidan foydalanish mumkin:

```
python
КопироватьРедактировать
cv2.imwrite("kulrang_rasm.jpg", gray_image)
```

Bu buyruq natijani `"kulrang_rasm.jpg"` nomli faylga saqlaydi.

---

2. Video fayllar bilan ishlash

2.1. Video faylni ochish va ijro etish

Videolarni yuklab, OpenCV yordamida ekranga chiqarish mumkin:

```
python
КопироватьРедактировать
import cv2

# Video faylni ochish
video = cv2.VideoCapture("video.mp4")

while True:
    ret, frame = video.read()

    # Agar kadr mavjud bo‘lsa, uni ekranga chiqaramiz
    if not ret:
        break

    cv2.imshow("Video", frame)

    # Agar 'q' tugmasi bosilsa, videoni to‘xtatamiz
    if cv2.waitKey(25) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Kod tushuntirishi:

- `cv2.VideoCapture("video.mp4")` – `"video.mp4"` faylini yuklaydi.
- `video.read()` – har bir kadrni o‘qiydi.
- `cv2.imshow("Video", frame)` – ekranga chiqaradi.
- `"q"` tugmasi bosilganda video ijrosi to‘xtaydi.

2.2. Video faylni yozib olish va saqlash

Agar siz kameradan yoki boshqa manbadan kelayotgan tasvirlarni saqlamoqchi bo‘lsangiz, quyidagi koddan foydalaning:

```
python
КопироватьРедактировать
import cv2

# Kameradan tasvir olish
camera = cv2.VideoCapture(0)

# Video format va kodlash
fourcc = cv2.VideoWriter_fourcc(*"XVID")
output = cv2.VideoWriter("natija.avi", fourcc, 20.0, (640, 480))

while True:
    ret, frame = camera.read()

    if not ret:
        break

    output.write(frame)
```

```
    cv2.imshow("Kamera", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

camera.release()
output.release()
cv2.destroyAllWindows()
```

Natija: `"natija.avi"` nomli video fayl yaratilib, kamera tasviri saqlanadi.

---

3. Kameradan real vaqtda tasvir olish

OpenCV real vaqtda **web-kameradan yoki boshqa kameradan tasvir olish** imkonini beradi.

3.1. Kameradan jonli tasvir olish

```
python
КопироватьРедактировать
import cv2

# Kamerani ishga tushirish (0 - asosiy kamera)
camera = cv2.VideoCapture(0)

while True:
    ret, frame = camera.read()

    if not ret:
        break

    cv2.imshow("Jonli Tasvir", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

camera.release()
cv2.destroyAllWindows()
```

Natija: Kamera tasviri real vaqtda ekranga chiqariladi. `"q"` tugmasi bosilganda kamera yopiladi.

---

4. OpenCV yordamida GUI interfeyslarini boshqarish

OpenCV dasturlarida maxsus oynalar va klaviatura hodisalarini boshqarish mumkin.

4.1. Oyna yaratish va o‘lchamini o‘zgartirish

```
python
КопироватьРедактировать
cv2.namedWindow("Interfeys", cv2.WINDOW_NORMAL)
cv2.imshow("Interfeys", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- `"cv2.WINDOW_NORMAL"` – oynani o‘lchamini o‘zgartirish imkonini beradi.

4.2. Klaviatura tugmalarini boshqarish

```
python
КопироватьРедактировать
key = cv2.waitKey(0) & 0xFF
if key == ord("s"):
    cv2.imwrite("saqlangan_rasm.jpg", image)
```

- Agar `"s"` tugmasi bosilsa, rasm `"saqlangan_rasm.jpg"` sifatida saqlanadi.

Xulosa

Biz OpenCV yordamida **rasm va video fayllarni yuklash, tahrirlash va saqlash**, **kameradan tasvir olish** va **GUI interfeyslarini boshqarish** mavzularini ko‘rib chiqdik. Endi siz OpenCV yordamida:

✅ Rasm va videolarni yuklash va o‘zgartirish
✅ Kameradan real vaqtda tasvir olish
✅ GUI oynalari va klaviatura tugmalarini boshqarish

3-BO‘LIM: OPENCV BILAN TASVIRNI QAYTA ISHLASH

OpenCV yordamida tasvirlarni turli formatlarga o‘tkazish, ularga filtrlar qo‘llash va tasvir tarkibini tahlil qilish mumkin. Ushbu bo‘limda quyidagilarni o‘rganamiz:

✅ **Rang o‘tkazish** – Tasvirni **kulrang (grayscale)**, **HSV**, **LAB** va boshqa formatlarga o‘tkazish
✅ **Filtrlar va konvolyutsiya** – Tasvirlarni silliqlash, kontrastni oshirish
✅ **Chegaralarni aniqlash** – Canny algoritmi yordamida ob’ekt chekkalarini topish
✅ **Shakllarni aniqlash** – Tasvirdagi konturlarni va geometrik figuralarni topish

---

1. Rang o‘tkazish (Color Conversion)

OpenCV tasvirlarni turli rang maydonlariga o‘tkazishga imkon beradi. Eng mashhur rang formatlari:

1️⃣ **BGR (ko‘k, yashil, qizil)** – OpenCV sukut bo‘yicha ishlatadigan format
2️⃣ **GRAY (kulrang)** – Yaltiroqlik (brightness) bo‘yicha faqat bitta kanal
3️⃣ **HSV (hue, saturation, value)** – Ranglarni yaxshiroq ajratish uchun ishlatiladi
4️⃣ *LAB (CIE Lab)** – Ranglarni inson ko‘rish tizimiga mos ravishda saqlash

---

1.1. Tasvirni kulrang formatga o‘tkazish

```
python
КопироватьРедактировать
import cv2
```

```
# Tasvirni yuklash
image = cv2.imread("rasm.jpg")

# BGR → GRAY konversiya
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Natijani chiqarish
cv2.imshow("Kulrang Tasvir", gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Rasm faqat **qora-oq** (kulrang) tuslarda ko‘rinadi.

1.2. Tasvirni HSV formatga o‘tkazish

```
python
КопироватьРедактировать
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
cv2.imshow("HSV Tasvir", hsv)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Rasm **HSV** formatida aks etadi. HSV rang maydoni ranglarni ajratishda samaraliroq.

1.3. Tasvirni LAB formatga o‘tkazish

```
python
КопироватьРедактировать
lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
cv2.imshow("LAB Tasvir", lab)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Rasm inson ko‘rish tizimiga moslashgan **LAB** rang formatiga o‘tkaziladi.

---

2. Filtrlar va konvolyutsiya (blurring va sharpening)

Filtrlar tasvirni silliqlash (blur), kontrastni oshirish yoki shovqinni kamaytirish uchun ishlatiladi.

2.1. Tasvirni silliqlash (blurring)

Tasvirni silliqlash uchun **Gauss filtri** yoki **oddiy o‘rtacha filtr** ishlatiladi.

```
python
КопироватьРедактировать
blurred = cv2.GaussianBlur(image, (7, 7), 0)
cv2.imshow("Silliqlangan tasvir", blurred)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

Natija: Rasm **tumanli** (blur) ko‘rinishga ega bo‘ladi.

2.2. Kontrastni oshirish (sharpening)

Tasvirning kontrastini oshirish uchun **keskinlashtirish yadrosi (kernel)** qo‘llaniladi.

```
python
КопироватьРедактировать
import numpy as np

# Keskinlashtirish yadrosi (kernel)
kernel = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]])

# Tasvirga filtrni qo‘llash
sharpened = cv2.filter2D(image, -1, kernel)

cv2.imshow("Keskinlashtirilgan tasvir", sharpened)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Tasvirning kontrasti oshadi va detallari aniqroq bo‘ladi.

---

3. Chegaralarni aniqlash (Canny Edge Detection)

Canny algoritmi tasvirdagi **ob’ektlarning chegaralarini** aniqlash uchun ishlatiladi.

```
python
КопироватьРедактировать
edges = cv2.Canny(image, 100, 200)
cv2.imshow("Chegaralar", edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Rasmda faqat **chegaralar (konturlar)** qoladi.

Muhim! `cv2.Canny(image, 100, 200)` buyrug‘ida **100 va 200** – past va yuqori chegaraviy qiymatlar. Ular tasvirga qarab o‘zgartirilishi mumkin.

---

4. Shakllarni aniqlash (Contours detection)

Tasvirdagi ob’ektlarning tashqi chegaralarini aniqlash uchun konturlar ishlatiladi.

4.1. Konturlarni aniqlash

```
python
КопироватьРедактировать
```

```
# Kulrang formatga o‘tkazish
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Threshold qo‘llash
_, thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)

# Konturlarni topish
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Konturlarni chizish
cv2.drawContours(image, contours, -1, (0, 255, 0), 2)

cv2.imshow("Konturlar", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Tasvirdagi barcha ob’ektlarning tashqi chegaralari yashil rang bilan chiziladi.

---

5. Doira va chiziqlarni aniqlash

5.1. Chiziqlarni aniqlash (Hough Transform)

```
python
КопироватьРедактировать
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150)

# Chiziqlarni topish
lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 50, minLineLength=50,
maxLineGap=10)

# Chiziqlarni chizish
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 2)

cv2.imshow("Chiziqlar", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Rasmda to‘g‘ri chiziqlar aniqlanadi va yashil rang bilan chiziladi.

Xulosa

Biz OpenCV yordamida **tasvirlarni qayta ishlash**, **filtrlar qo‘llash** va **ob’ektlarni aniqlash** mavzularini o‘rganib chiqdik. Endi siz:

✅ **Rang o‘zgartirish** (GRAY, HSV, LAB)
✅ **Tasvirga filtrlar qo‘llash** (blur, sharpening)
✅ **Canny Edge Detection bilan chegaralarni aniqlash**
✅ **Konturlarni va shakllarni topish**

4-BO‘LIM: TASVIRNI SEGMENTATSIYA QILISH VA CHUQURLIKNI BAHOLASH

Tasvirni segmentatsiya qilish — bu tasvirni turli qismlarga ajratish jarayoni bo‘lib, obyektlarni fon yoki boshqa elementlardan ajratish uchun ishlatiladi. Ushbu bo‘limda biz quyidagilarni ko‘rib chiqamiz:

✅ **Chuqurlik kameralaridan tasvir olish**
✅ **Chuqurlik xaritalari yordamida niqob yaratish**
✅ **Watershed va GrabCut algoritmlari bilan obyektlarni segmentatsiya qilish**
✅ **Obyektlarni fonidan ajratish va maskalash**

---

1. Chuqurlik kameralaridan tasvir olish

Chuqurlik kameralaridan foydalanish uchun **OpenNI yoki Microsoft Kinect** kabi texnologiyalar talab etiladi. Agar chuqurlik kamerasi mavjud bo‘lsa, OpenCV yordamida quyidagicha tasvir olish mumkin:

```
python
КопироватьРедактировать
import cv2

# Chuqurlik kamerani ishga tushirish
camera = cv2.VideoCapture(cv2.CAP_OPENNI)

while True:
    ret, depth_map = camera.read()

    if not ret:
        break

    cv2.imshow("Chuqurlik xaritasi", depth_map)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

camera.release()
cv2.destroyAllWindows()
```

Natija: Kameradan real vaqtda **chuqurlik xaritasi** olinadi va ekranga chiqariladi.

Muhim! OpenCV faqat **OpenNI yoki Kinect bilan mos keluvchi kameralar** bilan chuqurlikni o‘lchay oladi.

---

2. Chuqurlik xaritasidan niqob yaratish

Chuqurlik xaritasi yordamida **aniq obyektlarni fonidan ajratish** mumkin. Quyidagi kod tasvirdagi **ma’lum masofadan uzoq obyektlarni o‘chiradi**:

```
python
КопироватьРедактировать
```

```
import numpy as np

# Minimal va maksimal chuqurlik chegaralarini o‘rnatish
min_depth = 500  # 50 sm
max_depth = 1500  # 150 sm

# Niqob yaratish
mask = (depth_map > min_depth) & (depth_map < max_depth)

# Natijani chiqarish
filtered_image = np.where(mask, depth_map, 0)

cv2.imshow("Filtrlangan chuqurlik xaritasi", filtered_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Chuqurligi **500 mm dan 1500 mm gacha** bo‘lgan obyektlar ekranda qoladi, qolganlari esa o‘chirib tashlanadi.

---

3. Watershed algoritmi yordamida obyektlarni segmentatsiya qilish

Watershed algoritmi tasvirdagi obyektlarni fon va oldingi qatlamlarga ajratish uchun ishlatiladi.

```
python
КопироватьРедактировать
import cv2
import numpy as np

# Tasvirni yuklash
image = cv2.imread("obyekt.jpg")

# Kulrang formatga o‘tkazish
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Tasvirni threshold bilan segmentatsiya qilish
_, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

# Tashqi fonni aniqlash
kernel = np.ones((3, 3), np.uint8)
sure_bg = cv2.dilate(thresh, kernel, iterations=3)

# Old fonni aniqlash
dist_transform = cv2.distanceTransform(thresh, cv2.DIST_L2, 5)
_, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255,
0)

# Farqni topish
unknown = cv2.subtract(sure_bg, sure_fg)

# Markerlarni yaratish
_, markers = cv2.connectedComponents(sure_fg.astype(np.uint8))

# Watershed algoritmini qo‘llash
markers = markers + 1
markers[unknown == 255] = 0
markers = cv2.watershed(image, markers)
```

```
# Natijani chiqarish
image[markers == -1] = [0, 0, 255]  # Chegaralarni qizil rangda chizish
cv2.imshow("Watershed segmentatsiya", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Watershed algoritmi yordamida **obyektlarning chegaralari aniqlanadi** va qizil rangda chiziladi.

---

4. GrabCut algoritmi yordamida obyektlarni fonidan ajratish

GrabCut algoritmi aniq segmentatsiya qilish uchun ishlatiladi va **obyektni fonidan ajratish** imkonini beradi.

```
python
КопироватьРедактировать
# Tasvirni yuklash
image = cv2.imread("obyekt.jpg")

# Niqob yaratish
mask = np.zeros(image.shape[:2], np.uint8)

# GrabCut uchun vaqtinchalik arraylar
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)

# Qiziqarli hudud (ROI) - obyekt joylashgan taxminiy joy
rect = (50, 50, 400, 500)

# GrabCut algoritmini qo‘llash
cv2.grabCut(image, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)

# Fon piksellarini qora rangga aylantirish
mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype("uint8")
segmented = image * mask2[:, :, np.newaxis]

cv2.imshow("Segmentatsiya qilingan obyekt", segmented)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Obyekt fonidan ajratiladi va ortiqcha elementlar olib tashlanadi.

Muhim! `rect = (50, 50, 400, 500)` qismi segmentatsiya qilish uchun taxminiy obyekt joylashgan hududni belgilaydi.

---

5. Obyektni maskalash va fonni almashtirish

Agar segmentatsiya qilingan obyektni boshqa fon bilan almashtirmoqchi bo‘lsak, maskadan foydalanamiz.

```
python
КопироватьРедактировать
# Yangi fon tasvirini yuklash
background = cv2.imread("new_background.jpg")
```

```
# Maskani qayta ishlash
mask_inv = cv2.bitwise_not(mask2)

# Yangi fonni joylashtirish
bg_part = cv2.bitwise_and(background, background, mask=mask_inv)
fg_part = cv2.bitwise_and(image, image, mask=mask2)
final_result = cv2.add(bg_part, fg_part)

cv2.imshow("Yangi fon bilan obyekt", final_result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Obyekt foni o‘zgartirilib, yangi fon joylashtiriladi.

Xulosa

Biz OpenCV yordamida **tasvirni segmentatsiya qilish va chuqurlik xaritalari bilan ishlash** bo‘yicha muhim usullarni o‘rganib chiqdik. Endi siz:

✅ **Chuqurlik xaritalari bilan ishlash**
✅ **Watershed va GrabCut algoritmlarini qo‘llash**
✅ **Tasvirdagi obyektlarni maskalash va fonni almashtirish**

5-BO‘LIM: OPENCV YORDAMIDA YUZNI ANIQLASH VA TANIB OLISH

Yuzni aniqlash va tanib olish sun’iy intellekt va kompyuter ko‘rish sohasida keng qo‘llaniladi. Ushbu bo‘limda quyidagilarni ko‘rib chiqamiz:

✅ **Haar Cascade yordamida yuzni aniqlash**
✅ **DNN modeli yordamida yuzni aniqlash**
✅ **LBPH (Local Binary Patterns Histogram) yordamida yuzni tanib olish**
✅ **Yuzni niqoblash va yuz xususiyatlarini chiqarish**

---

1. Haar Cascade yordamida yuzni aniqlash

Haar Cascade – OpenCV kutubxonasida mavjud bo‘lgan yengil va samarali algoritm bo‘lib, tasvirdan yuzlarni tezda aniqlashga yordam beradi.

1.1. Haar Cascade XML modelini yuklash

OpenCV oldindan o‘rgatilgan Haar Cascade modelini taqdim etadi. Uni ishlatish uchun avval yuklab olish kerak:

```
python
КопироватьРедактировать
import cv2

# Yuzni aniqlash modeli
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
```

```
# Tasvirni yuklash
image = cv2.imread("face.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Yuzlarni aniqlash
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))

# Yuz atrofida to‘rtburchak chizish
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.imshow("Aniqlangan yuzlar", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Yuzlar yashil to‘rtburchak bilan ajratib ko‘rsatiladi.

Muhim! Haar Cascade modeli sodda va tez ishlaydi, lekin ba’zan noto‘g‘ri natijalar berishi mumkin. Agar aniqroq model kerak bo‘lsa, **DNN (Deep Neural Networks) usulidan foydalanamiz.**

---

2. DNN modeli yordamida yuzni aniqlash (Deep Learning)

Deep Learning asosidagi modellar Haar Cascade’ga qaraganda ancha aniq ishlaydi. OpenCV **ResNet Caffe modeli** bilan **Face Detection** (yuzni aniqlash) funksiyasini ta’minlaydi.

2.1. DNN modelini yuklab olish

Avval OpenCV bilan taqdim etilgan **ResNet Caffe** modelini yuklab olamiz:

- **Model fayli:** res10_300x300_ssd_iter_140000.caffemodel
- **Arxitektura fayli:** deploy.prototxt

2.2. DNN model yordamida yuzni aniqlash

```
python
КопироватьРедактировать
import cv2

# Model va konfiguratsiyani yuklash
net = cv2.dnn.readNetFromCaffe("deploy.prototxt",
"res10_300x300_ssd_iter_140000.caffemodel")

# Tasvirni yuklash
image = cv2.imread("face.jpg")
(h, w) = image.shape[:2]

# Tasvirni DNN tarmog‘iga tayyorlash
blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size=(300, 300),
mean=(104.0, 177.0, 123.0))

# Yuzni aniqlash
net.setInput(blob)
detections = net.forward()
```

```
# Aniqlangan yuzlarni chizish
for i in range(detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if confidence > 0.5:  # Ishonchlilik darajasi 50% dan yuqori bo‘lsa
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (x, y, x1, y1) = box.astype("int")
        cv2.rectangle(image, (x, y), (x1, y1), (0, 255, 0), 2)

cv2.imshow("DNN yordamida yuzni aniqlash", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: DNN modeli aniqroq yuz aniqlash imkonini beradi.

Afzallik: DNN modeli Haar Cascade’ga qaraganda ancha ishonchli natija beradi.

---

3. LBPH yordamida yuzni tanib olish (Face Recognition)

LBPH (Local Binary Patterns Histogram) – OpenCV’da mavjud bo‘lgan yuzni tanib olish algoritmi bo‘lib, **ma’lum shaxslarning yuzini o‘rgatish va tanib olish** uchun ishlatiladi.

3.1. LBPH yordamida yuzlarni o‘rgatish

Avval ma’lum bir shaxsning yuz tasvirlari bo‘yicha **o‘quv modeli** yaratamiz:

```
python
КопироватьРедактировать
import cv2
import numpy as np
import os

# Yuzni aniqlash uchun Haar Cascade modeli
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")

# LBPH yuzni tanib olish modeli
recognizer = cv2.face.LBPHFaceRecognizer_create()

# Yuz tasvirlarini va ularning yorliqlarini (ID) saqlash
faces = []
labels = []

# Tasvirlar joylashgan katalog
dataset_path = "dataset/"

# Har bir foydalanuvchi uchun
for label, person in enumerate(os.listdir(dataset_path)):
    person_path = os.path.join(dataset_path, person)
    for image_name in os.listdir(person_path):
        image_path = os.path.join(person_path, image_name)
        img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        faces.append(img)
        labels.append(label)

# Modelni o‘rgatish
recognizer.train(faces, np.array(labels))

# O‘rgatilgan modelni saqlash
```

```
recognizer.save("face_model.yml")
```

Natija: Yuz tasvirlari asosida model o‘rgatiladi va `"face_model.yml"` faylida saqlanadi.

3.2. Yuzni tanib olish

Endi **o‘rgatilgan model** yordamida yuzni tanib olamiz:

```
python
КопироватьРедактировать
# O‘rgatilgan modelni yuklash
recognizer.read("face_model.yml")

# Test tasvirni yuklash
test_image = cv2.imread("test_face.jpg", cv2.IMREAD_GRAYSCALE)

# Yuzni aniqlash
faces = face_cascade.detectMultiScale(test_image, scaleFactor=1.2,
minNeighbors=5)

for (x, y, w, h) in faces:
    face_roi = test_image[y:y+h, x:x+w]
    label, confidence = recognizer.predict(face_roi)

    # Natijani chiqarish
    cv2.putText(test_image, f"ID: {label}, Conf: {confidence:.2f}", (x, y-
10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
    cv2.rectangle(test_image, (x, y), (x + w, y + h), (0, 255, 0), 2)

cv2.imshow("Tanib olingan yuz", test_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: Agar yuz **o‘quv ma’lumotlarida mavjud bo‘lsa**, model uni tanib oladi va **ID raqamini chiqaradi**.

Xulosa

Biz OpenCV yordamida **yuzni aniqlash va tanib olish** usullarini o‘rganib chiqdik.

✅ Haar Cascade bilan oddiy yuzni aniqlash
✅ DNN modeli yordamida aniqroq yuz aniqlash
✅ LBPH modeli bilan yuzni tanib olish

6-BO‘LIM: OPENCV YORDAMIDA OBYEKTLARNI KUZATISH VA HARAKATNI ANIQLASH

Obyektlarni kuzatish va harakatni aniqlash kompyuter ko‘rishning muhim sohalaridan biri bo‘lib, kuzatuv tizimlari, robototexnika va xavfsizlik kameralarida keng qo‘llaniladi. Ushbu bo‘limda quyidagilarni ko‘rib chiqamiz:

✅ **Frame Differencing (Kadrlar orasidagi farq)**
✅ **Harakatni aniqlash va kuzatish**
✅ **Background Subtraction (Fonni ajratish)**
✅ **YOLO va Deep Learning yordamida obyektlarni kuzatish**

---

1. Frame Differencing yordamida harakatni aniqlash

Frame Differencing – tasvirdagi harakatni aniqlashning oddiy usullaridan biri bo‘lib, ikkita ketma-ket kadr o‘rtasidagi farqni hisoblash orqali ishlaydi.

1.1. Harakatni aniqlash algoritmi

```
python
КопироватьРедактировать
import cv2
import numpy as np

# Videoni ochish
video = cv2.VideoCapture("video.mp4")

# Birinchi kadrni o‘qish
ret, first_frame = video.read()
gray_first = cv2.cvtColor(first_frame, cv2.COLOR_BGR2GRAY)
gray_first = cv2.GaussianBlur(gray_first, (21, 21), 0)

while True:
    ret, frame = video.read()
    if not ret:
        break

    # Joriy kadrni kulrang formatga o‘tkazish va silliqlash
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (21, 21), 0)

    # Kadrlar orasidagi farqni hisoblash
    frame_diff = cv2.absdiff(gray_first, gray)

    # Threshold qo‘llash
    _, thresh = cv2.threshold(frame_diff, 30, 255, cv2.THRESH_BINARY)

    # Natijani chiqarish
    cv2.imshow("Harakat aniqlash", thresh)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Agar tasvirda harakat bo‘lsa, oq rangda aks etadi.

Afzallik: Oddiy va tez ishlaydi.
Kamchilik: Ko‘p shovqin (noise) hosil qilishi mumkin.

---

2. Background Subtraction yordamida harakatni aniqlash

Background Subtraction usuli tasvir fonini o‘rganib, undan farqli obyektlarni aniqlashga asoslangan.

2.1. MOG2 yordamida fonni ajratish

```
python
КопироватьРедактировать
# Fonni ajratish uchun MOG2 modeli
fgbg = cv2.createBackgroundSubtractorMOG2()

video = cv2.VideoCapture("video.mp4")

while True:
    ret, frame = video.read()
    if not ret:
        break

    # Fonni ajratish
    fgmask = fgbg.apply(frame)

    cv2.imshow("Fonni ajratish", fgmask)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Harakatdagi obyektlar oq rangda, fon esa qora rangda aks etadi.

Afzallik: Kichik o‘zgarishlarni avtomatik filtrlash imkonini beradi.
Kamchilik: Dastlabki fonni o‘rganishi uchun vaqt talab etadi.

---

3. CSRT yordamida obyektlarni kuzatish

CSRT (**Channel and Spatial Reliability Tracker**) algoritmi harakat qilayotgan obyektni real vaqtda kuzatish uchun ishlatiladi.

3.1. Obyektni qo‘lda belgilash va kuzatish

```
python
КопироватьРедактировать
import cv2

# Videoni ochish
video = cv2.VideoCapture("video.mp4")

# Tracker yaratish
tracker = cv2.TrackerCSRT_create()

# Birinchi kadrni o‘qish
ret, frame = video.read()

# Obyektni tanlash
```

```
bbox = cv2.selectROI("Obyektni tanlang", frame, False)

# Trackerga obyektni qo‘shish
tracker.init(frame, bbox)

while True:
    ret, frame = video.read()
    if not ret:
        break

    # Obyektni kuzatish
    success, bbox = tracker.update(frame)

    if success:
        x, y, w, h = map(int, bbox)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    cv2.imshow("Obyektni kuzatish", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Obyekt harakatlanishi bilan yashil to‘rtburchak yordamida kuzatiladi.

Afzallik: Aniqligi yuqori.
Kamchilik: Obyekt ekrandan chiqib ketganida yo‘qoladi.

---

4. YOLO yordamida obyektlarni kuzatish

YOLO (**You Only Look Once**) Deep Learning modeli obyektlarni aniq aniqlash va kuzatish uchun ishlatiladi.

4.1. YOLO modelini yuklash

Avval quyidagi fayllarni yuklab olish kerak:

- **yolov3.weights** – oldindan o‘rgatilgan model
- **yolov3.cfg** – model arxitekturasi
- **coco.names** – obyekt turlari

4.2. YOLO yordamida obyektlarni kuzatish

```
python
КопироватьРедактировать
import cv2
import numpy as np

# YOLO modelini yuklash
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
layer_names = net.getUnconnectedOutLayersNames()

# Obyekt nomlarini yuklash
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]
```

```
video = cv2.VideoCapture("video.mp4")

while True:
    ret, frame = video.read()
    if not ret:
        break

    # Tasvirni YOLO modeliga tayyorlash
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), swapRB=True,
crop=False)
    net.setInput(blob)
    outs = net.forward(layer_names)

    # Obyektlarni aniqlash
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]

            if confidence > 0.5:
                x, y, w, h = detection[:4] * np.array([frame.shape[1],
frame.shape[0], frame.shape[1], frame.shape[0]])
                x, y, w, h = int(x - w / 2), int(y - h / 2), int(w), int(h)
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv2.putText(frame, f"{classes[class_id]}: {confidence:.2f}",
(x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    cv2.imshow("YOLO obyekt kuzatish", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: YOLO real vaqtda obyektlarni taniydi va ularni kuzatadi.

Xulosa

Biz OpenCV yordamida **obyektlarni kuzatish va harakatni aniqlash** usullarini o‘rganib chiqdik.

✅ Frame Differencing va Background Subtraction yordamida harakatni aniqlash
✅ CSRT yordamida obyektlarni real vaqtda kuzatish
✅ YOLO modeli bilan obyektlarni aniq aniqlash

7-BO‘LIM: OPENCV YORDAMIDA QO‘L HARAKATLARINI ANIQLASH (GESTURE RECOGNITION)

Qo‘l harakatlarini aniqlash kompyuter interfeysi, robototexnika va sun’iy intellekt ilovalarida qo‘llaniladi. Ushbu bo‘limda quyidagilarni ko‘rib chiqamiz:

✅ **Qo‘lni aniqlash va konturlarini topish**
✅ **Barmoqlar sonini aniqlash**
✅ **Qo‘l shaklini aniqlash va maskalash**
✅ **Rejalashtirilgan harakatlarni (gesture) real vaqtda tanib olish**

---

1. Qo‘lni aniqlash va konturlarni topish

Qo‘lni tasvirdan ajratish uchun **rang maydonini filtrlash va konturlarni aniqlash** usulidan foydalanamiz.

1.1. Qo‘lni aniqlash va fonni olib tashlash

```
python
КопироватьРедактировать
import cv2
import numpy as np

# Kamera orqali video olish
video = cv2.VideoCapture(0)

while True:
    ret, frame = video.read()
    if not ret:
        break

    # Tasvirni HSV formatga o‘tkazish
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Qo‘l rangini filtrlash (odatiy inson terisi rangi uchun)
    lower_skin = np.array([0, 20, 70], dtype=np.uint8)
    upper_skin = np.array([20, 255, 255], dtype=np.uint8)
    mask = cv2.inRange(hsv, lower_skin, upper_skin)

    # Natijani chiqarish
    cv2.imshow("Qo‘l maskasi", mask)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Kamera orqali tasvir olinadi va inson qo‘li **oq rangda** ajratiladi, fon esa **qora rangga** o‘tkaziladi.

Muhim! Bu usul ishlashi uchun yorug‘lik sharoiti yaxshi bo‘lishi kerak.

---

2. Konturlar yordamida qo‘l shaklini aniqlash

Konturlar tasvirdagi shakllarni aniqlash uchun ishlatiladi.

2.1. Konturlarni topish va qo‘l chegaralarini chizish

```
python
КопироватьРедактировать
# Konturlarni topish
contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

if len(contours) > 0:
    max_contour = max(contours, key=cv2.contourArea)  # Eng katta konturni
tanlash
    cv2.drawContours(frame, [max_contour], -1, (0, 255, 0), 2)

cv2.imshow("Qo‘l konturi", frame)
```

Natija: Qo‘l atrofida yashil chiziq chiziladi.

Afzallik: Konturlar yordamida qo‘l shaklini aniqlash ancha aniq natija beradi.

---

3. Barmoqlar sonini aniqlash

Qo‘l barmoqlarini aniqlash uchun **konveks defektlar (convex defects)** usulidan foydalanamiz.

3.1. Qo‘l konturi va barmoqlar sonini aniqlash

```
python
КопироватьРедактировать
hull = cv2.convexHull(max_contour, returnPoints=False)
defects = cv2.convexityDefects(max_contour, hull)

# Barmoqlarni sanash
finger_count = 0

for i in range(defects.shape[0]):
    s, e, f, d = defects[i, 0]
    start = tuple(max_contour[s][0])
    end = tuple(max_contour[e][0])
    far = tuple(max_contour[f][0])

    # Masofa yetarlicha katta bo‘lsa, barmoq sifatida sanaymiz
    if d > 10000:
        finger_count += 1
        cv2.circle(frame, far, 5, (0, 0, 255), -1)  # Barmoqlar orasidagi
nuqtalarni belgilash

cv2.putText(frame, f"Barmoqlar soni: {finger_count + 1}", (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
cv2.imshow("Barmoqlarni aniqlash", frame)
```

Natija: Barmoqlar orasidagi nuqtalar qizil doira bilan belgilab qo‘yiladi va ekranga barmoqlar soni chiqariladi.

Muhim! Model aniq ishlashi uchun tasvirni yaxshi yoritish kerak.

---

4. Qo‘l harakatlarini real vaqtda aniqlash (Gesture Recognition)

Quyidagi kod **oldindan belgilangan qo‘l harakatlarini tanib olish** uchun ishlatiladi.

4.1. Harakatlarni aniqlash

```
python
КопироватьРедактировать
if finger_count == 1:
    gesture = "Salom!"
elif finger_count == 2:
    gesture = "V ishorasi"
elif finger_count == 5:
    gesture = "Ochilgan kaft"

cv2.putText(frame, f"Gesture: {gesture}", (50, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)
cv2.imshow("Qo‘l harakati", frame)
```

Natija: Agar qo‘lda 5 barmoq bo‘lsa, `"Ochilgan kaft"` deb yoziladi.

Afzallik: Bu usul real vaqtda ishlaydi.
Kamchilik: Modelni yaxshilash uchun qo‘l joylashuvi va yorug‘lik ta’siri hisobga olinishi kerak.

---

5. DNN yordamida qo‘l harakatlarini aniqlash

Deep Learning (DNN) modellaridan foydalangan holda qo‘l harakatlarini yanada aniq aniqlash mumkin. **MediaPipe** modeli bu jarayonni tez va samarali bajaradi.

5.1. MediaPipe kutubxonasini o‘rnatish

```
sh
КопироватьРедактировать
pip install mediapipe
```

5.2. MediaPipe yordamida qo‘l harakatlarini aniqlash

```
python
КопироватьРедактировать
import mediapipe as mp

mp_hands = mp.solutions.hands
hands = mp_hands.Hands()
mp_draw = mp.solutions.drawing_utils

while True:
    ret, frame = video.read()
    if not ret:
        break

    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    result = hands.process(rgb_frame)

    if result.multi_hand_landmarks:
        for hand_landmarks in result.multi_hand_landmarks:
            mp_draw.draw_landmarks(frame, hand_landmarks,
mp_hands.HAND_CONNECTIONS)

    cv2.imshow("MediaPipe bilan qo‘l harakatlarini aniqlash", frame)
```

```
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Qo‘l barmoqlari va harakatlari aniq belgilab chiqiladi.

Afzallik: AI modeli aniq natija beradi.
Kamchilik: Resurs talab qiladi.

Xulosa

Biz OpenCV yordamida **qo‘l harakatlarini aniqlash va gesture recognition** mavzusini o‘rganib chiqdik. Endi siz:

✅ **Qo‘lni aniqlash va fonni olib tashlash**
✅ **Barmoqlar sonini aniqlash**
✅ **Qo‘l harakatlarini real vaqtda tanib olish**
✅ **Deep Learning yordamida qo‘l harakatlarini kuzatish**

8-BO‘LIM: OPENCV YORDAMIDA YUZ IFODALARINI ANIQLASH VA EMOTSIYALARNI TANIB OLISH

Yuz ifodalarini aniqlash va ularni emotsiyalar bilan bog‘lash inson-kompyuter o‘zaro aloqasi, sun’iy intellekt va aqlli kuzatuv tizimlarida keng qo‘llaniladi. Ushbu bo‘limda quyidagilarni ko‘rib chiqamiz:

✅ **Haar Cascade yordamida yuzni aniqlash**
✅ **Dlib yoki MediaPipe yordamida yuz nuqtalarini topish**
✅ **CNN va Deep Learning yordamida emotsiyalarni tanib olish**
✅ **Real vaqt rejimida yuz ifodalarini aniqlash**

---

1. Haar Cascade yordamida yuzni aniqlash

Yuz ifodalarini tahlil qilishdan oldin, dastlab yuzni aniqlash lozim. OpenCV’dagi Haar Cascade modeli bunga yordam beradi.

1.1. Yuzni aniqlash va ramkaga olish

```
python
КопироватьРедактировать
import cv2

# Yuzni aniqlash uchun Haar Cascade modeli
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
```

```
# Kamerani ishga tushirish
video = cv2.VideoCapture(0)

while True:
    ret, frame = video.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5)

    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    cv2.imshow("Aniqlangan yuz", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Yuzlar yashil to‘rtburchak bilan ajratib ko‘rsatiladi.

Kamchilik: Bu model yuz ifodalarini tahlil qilish uchun yetarli emas. Yuz mushaklarini aniq aniqlash uchun **Dlib yoki MediaPipe** kutubxonalaridan foydalanamiz.

---

2. Dlib yordamida yuz nuqtalarini (landmarks) aniqlash

Dlib kutubxonasi yuzning 68 ta asosiy nuqtalarini aniqlash imkonini beradi.

2.1. Dlib modelini yuklab olish

```
sh
КопироватьРедактировать
pip install dlib
```

2.2. Yuz mushaklarini aniqlash

```
python
КопироватьРедактировать
import dlib
import cv2

# Dlib modeli va yuz detektori
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

video = cv2.VideoCapture(0)

while True:
    ret, frame = video.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = detector(gray)
```

```
    for face in faces:
        landmarks = predictor(gray, face)

        for n in range(68):
            x, y = landmarks.part(n).x, landmarks.part(n).y
            cv2.circle(frame, (x, y), 2, (0, 255, 0), -1)

    cv2.imshow("Yuz nuqtalari", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Yuz mushaklari yashil nuqtalar bilan belgilab chiqiladi.

Afzallik: Dlib modeli yuz mushaklarini juda aniq aniqlaydi.
Kamchilik: Bu model emotsiyalarni tanib olish uchun ishlatilmaydi, faqat **yuz harakatlarini tahlil qilish** mumkin.

---

3. CNN yordamida emotsiyalarni tanib olish (Deep Learning)

Convolutional Neural Networks (CNN) yordamida yuz ifodalarini aniqlash ancha ishonchli bo‘ladi. **FER2013** datasetida o‘qitilgan CNN modeli **6 ta asosiy emotsiyani** ajrata oladi:

- **0** – Baxtli
- **1** – G‘azablangan
- **2** – Hayratlangan
- **3** – Qo‘rqgan
- **4** – Xafa
- **5** – Betaraf

3.1. CNN modelini yuklab olish

FER2013 datasetida oldindan o‘qitilgan CNN modelini yuklab olamiz:

```
sh
КопироватьРедактировать
pip install tensorflow keras
```

3.2. Yuz ifodalarini aniqlash CNN modeli bilan

```
python
КопироватьРедактировать
import tensorflow as tf
from tensorflow.keras.models import load_model
import numpy as np

# Oldindan o‘qitilgan modelni yuklash
model = load_model("emotion_model.h5")
```

```
# Emotsiya nomlari
emotions = ["Baxtli", "G‘azablangan", "Hayratlangan", "Qo‘rqgan", "Xafa",
"Betaraf"]

video = cv2.VideoCapture(0)

while True:
    ret, frame = video.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5)

    for (x, y, w, h) in faces:
        face_roi = gray[y:y+h, x:x+w]
        face_roi = cv2.resize(face_roi, (48, 48)) / 255.0
        face_roi = np.expand_dims(face_roi, axis=0)
        face_roi = np.expand_dims(face_roi, axis=-1)

        # Model orqali emotsiyani aniqlash
        prediction = model.predict(face_roi)
        emotion_label = np.argmax(prediction)
        emotion_text = emotions[emotion_label]

        cv2.putText(frame, emotion_text, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 0, 0), 2)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    cv2.imshow("Yuz ifodalari", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: CNN modeli yuz ifodasini tanib oladi va mos emotsiyani ekranga chiqaradi.

Afzallik: Sun’iy intellekt yordamida aniq emotsiyalarni tanib olish mumkin.
Kamchilik: Modelni ishlatish uchun **oldindan o‘qitilgan dataset kerak**.

Xulosa

Biz OpenCV yordamida **yuz ifodalarini aniqlash va emotsiyalarni tanib olish** mavzusini o‘rganib chiqdik. Endi siz:

✅ **Haar Cascade yoki Dlib yordamida yuzni aniqlash**
✅ **Yuz mushaklarini va nuqtalarini aniqlash**
✅ **CNN modeli yordamida emotsiyalarni tanib olish**

9-BO‘LIM: OPENCV YORDAMIDA KO‘Z HARAKATLARINI ANIQLASH VA KUZATISH (EYE TRACKING)

Ko‘z harakatlarini aniqlash **aqlli interfeyslar, kuzatuv tizimlari, nevrologik tadqiqotlar va haydovchini monitoring qilish** kabi sohalarda qo‘llaniladi. Ushbu bo‘limda quyidagilarni ko‘rib chiqamiz:

✅ **Haar Cascade yordamida ko‘zlarni aniqlash**
✅ **Dlib yordamida ko‘z nuqtalarini (landmarks) aniqlash**
✅ **Ko‘z qorachiqlarining joylashuvini kuzatish**
✅ **Ko‘z yumilishini (blink detection) aniqlash**

---

1. Haar Cascade yordamida ko‘zlarni aniqlash

Haar Cascade modeli yordamida oddiy ko‘z detektori yaratish mumkin.

1.1. Yuz va ko‘zlarni aniqlash

```
python
КопироватьРедактировать
import cv2

# Yuz va ko‘zlarni aniqlash uchun oldindan o‘rgatilgan Haar Cascade
modellarini yuklash
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_frontalface_default.xml")
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
"haarcascade_eye.xml")

# Kamerani ishga tushirish
video = cv2.VideoCapture(0)

while True:
    ret, frame = video.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Yuzlarni aniqlash
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5)

    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

        # Yuz hududidagi ko‘zlarni aniqlash
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray)

        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex + ew, ey + eh), (0, 255,
0), 2)
```

```
    cv2.imshow("Ko‘zlarni aniqlash", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Yuzlar **ko‘k**, ko‘zlar esa **yashil** to‘rtburchak bilan belgilab chiqiladi.

Afzallik: Oddiy va tez ishlaydi.
Kamchilik: Ba’zan noto‘g‘ri natija berishi mumkin. **Dlib yoki MediaPipe** ishlatish tavsiya etiladi.

---

2. Dlib yordamida ko‘z nuqtalarini aniqlash

Dlib modeli **ko‘z konturlarini (68 ta landmark)** aniqlash imkonini beradi.

2.1. Dlib kutubxonasini o‘rnatish

```
sh
КопироватьРедактировать
pip install dlib
```

2.2. Ko‘z nuqtalarini (landmarks) aniqlash

```
python
КопироватьРедактировать
import dlib
import cv2

# Dlib modeli va yuz detektori
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

video = cv2.VideoCapture(0)

while True:
    ret, frame = video.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = detector(gray)

    for face in faces:
        landmarks = predictor(gray, face)

        # Ko‘z nuqtalarini chizish (chap: 36-41, o‘ng: 42-47)
        for n in range(36, 48):
            x, y = landmarks.part(n).x, landmarks.part(n).y
            cv2.circle(frame, (x, y), 2, (0, 255, 0), -1)

    cv2.imshow("Ko‘z nuqtalari", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
```

```
video.release()
cv2.destroyAllWindows()
```

Natija: Ko‘z atrofidagi muhim nuqtalar yashil nuqtalar bilan belgilanadi.

Afzallik: Yuz va ko‘z shakllarini aniq aniqlaydi.
Kamchilik: Resurs talab qiladi.

---

3. Ko‘z qorachiq joylashuvini aniqlash

Ko‘z qorachiq (pupil) harakatini aniqlash orqali insonning qarash yo‘nalishini kuzatish mumkin.

3.1. Qorachiqni aniqlash

```
python
КопироватьРедактировать
import numpy as np

for face in faces:
    landmarks = predictor(gray, face)

    # Chap va o‘ng ko‘z uchun ROI (Region of Interest)
    left_eye = np.array([(landmarks.part(n).x, landmarks.part(n).y) for n in
range(36, 42)])
    right_eye = np.array([(landmarks.part(n).x, landmarks.part(n).y) for n in
range(42, 48)])

    # Qorachiqni aniqlash
    min_x = np.min(left_eye[:, 0])
    max_x = np.max(left_eye[:, 0])
    min_y = np.min(left_eye[:, 1])
    max_y = np.max(left_eye[:, 1])

    eye_roi = gray[min_y:max_y, min_x:max_x]
    _, threshold_eye = cv2.threshold(eye_roi, 50, 255, cv2.THRESH_BINARY_INV)

    cv2.imshow("Ko‘z qorachig‘i", threshold_eye)
```

Natija: Qorachiq (ko‘z ichi) qora nuqta sifatida aniqlanadi.

Afzallik: Ko‘z yo‘nalishini kuzatish mumkin.
Kamchilik: Yorug‘lik sharoitiga bog‘liq.

---

4. Blink Detection (Ko‘z yumilishi) aniqlash

Agar **ko‘z nuqtalari** bir-biriga yaqinlashsa, bu **ko‘z yumilganligini** bildiradi.

4.1. Ko‘z yumilganligini aniqlash

```
python
КопироватьРедактировать
def eye_aspect_ratio(eye):
    A = np.linalg.norm(eye[1] - eye[5])
```

```
    B = np.linalg.norm(eye[2] - eye[4])
    C = np.linalg.norm(eye[0] - eye[3])
    return (A + B) / (2.0 * C)

for face in faces:
    landmarks = predictor(gray, face)

    left_eye = np.array([(landmarks.part(n).x, landmarks.part(n).y) for n in
range(36, 42)])
    right_eye = np.array([(landmarks.part(n).x, landmarks.part(n).y) for n in
range(42, 48)])

    left_ear = eye_aspect_ratio(left_eye)
    right_ear = eye_aspect_ratio(right_eye)

    avg_ear = (left_ear + right_ear) / 2.0

    if avg_ear < 0.2:
        cv2.putText(frame, "Ko'z yumuldi!", (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
```

Natija: Agar **ko‘zlar yumulsa**, ekranda `"Ko'z yumuldi!"` degan yozuv chiqadi.

Xulosa

Biz OpenCV yordamida **ko‘z harakatlarini aniqlash va kuzatish** mavzusini o‘rganib chiqdik. Endi siz:

✅ **Haar Cascade yoki Dlib yordamida ko‘zlarni aniqlash**
✅ **Ko‘z qorachiqlarini va qarash yo‘nalishini aniqlash**
✅ **Ko‘z yumilganligini aniqlash (Blink Detection)**

Keyingi bo‘limda **OpenCV yordamida ob’ektlarni 3D ko‘rinishda kuzatish va AR (Augmented Reality) effektlarini qo‘llash** mavzusini tarjima qilamiz.

10-BO‘LIM: OPENCV YORDAMIDA 3D OBYEKTLARNI KUZATISH VA AR (AUGMENTED REALITY) EFFEKTINI QO‘LLASH

Kengaytirilgan reallik (AR) texnologiyalari turli sohalarda qo‘llaniladi, jumladan:

✅ **Virtual obyektlarni real muhitga joylashtirish**
✅ **Markerlar yordamida obyektlarni kuzatish**
✅ **3D koordinatalar tizimini yaratish**
✅ **Kamerani kalibrovka qilish va AR effektlar qo‘shish**

---

1. 3D Koordinatalarni aniqlash

3D obyektlarni kuzatish uchun kameradan olingan tasvirlarni real koordinatalar bilan bog‘lash kerak.

1.1. Kameraning kalibrovkasi

Kameraning optik buzilishlarini tuzatish uchun kalibrovka bajariladi. Avval **shaxmat doskasi (checkerboard) tasviri** ishlatiladi.

1.1.1. Kamerani kalibrovka qilish

```
python
КопироватьРедактировать
import cv2
import numpy as np
import glob

# Shaxmat doskasining hajmi (ichki kvadratlar soni)
checkerboard_size = (7, 7)

# 3D koordinatalar
objp = np.zeros((checkerboard_size[0] * checkerboard_size[1], 3), np.float32)
objp[:, :2] = np.mgrid[0:checkerboard_size[0],
0:checkerboard_size[1]].T.reshape(-1, 2)

# Kalibrovatsiya uchun tasvirlar
obj_points = []  # 3D obyekt nuqtalari
img_points = []  # 2D tasvir nuqtalari

images = glob.glob("calibration_images/*.jpg")

for image_path in images:
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Shaxmat doskasining burchaklarini aniqlash
    ret, corners = cv2.findChessboardCorners(gray, checkerboard_size, None)

    if ret:
        obj_points.append(objp)
        img_points.append(corners)

# Kamerani kalibrovka qilish
ret, camera_matrix, dist_coeffs, rvecs, tvecs =
cv2.calibrateCamera(obj_points, img_points, gray.shape[::-1], None, None)

# Natijalarni saqlash
np.savez("camera_calib_data.npz", camera_matrix=camera_matrix,
dist_coeffs=dist_coeffs)
```

Natija: Kalibrovka natijalari `"camera_calib_data.npz"` faylida saqlanadi va kameraning aniq tasvir olishini ta’minlaydi.

Afzallik: Tasvirni buzilishlarsiz olish mumkin.
Kamchilik: Kalibrovka uchun bir nechta rasmlar kerak bo‘ladi.

---

2. AR Markerlarni aniqlash va kuzatish

AR texnologiyalarida **fiducial markerlar** ishlatiladi, masalan, **ArUco markerlari**.

2.1. ArUco markerlarini yaratish

```
python
КопироватьРедактировать
import cv2
import cv2.aruco as aruco

# ArUco marker yaratish
aruco_dict = aruco.Dictionary_get(aruco.DICT_4X4_50)
marker = np.zeros((200, 200), dtype=np.uint8)
marker = aruco.drawMarker(aruco_dict, 0, 200)

cv2.imwrite("aruco_marker.png", marker)
cv2.imshow("ArUco Marker", marker)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Natija: `"aruco_marker.png"` fayli yaratiladi, u AR tizimlar uchun marker sifatida ishlatiladi.

Afzallik: Markerlardan real obyektlarni kuzatish uchun foydalanish mumkin.
Kamchilik: Faqat oldindan belgilangan markerlar ishlatiladi.

2.2. ArUco markerlarini real vaqtda aniqlash

```
python
КопироватьРедактировать
video = cv2.VideoCapture(0)
aruco_dict = aruco.Dictionary_get(aruco.DICT_4X4_50)
parameters = aruco.DetectorParameters_create()

while True:
    ret, frame = video.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    corners, ids, _ = aruco.detectMarkers(gray, aruco_dict,
parameters=parameters)

    if ids is not None:
        aruco.drawDetectedMarkers(frame, corners, ids)

    cv2.imshow("ArUco Marker Kuzatish", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Kamera orqali **ArUco markerlari** aniqlanadi va ularning raqami ekranga chiqariladi.

---

3. 3D AR modellarni joylashtirish

Agar biz tasvirdagi **aniqlangan marker** ustiga 3D obyekt joylashtirmoqchi bo‘lsak, **pose estimation** ishlatiladi.

3.1. 3D obyektni joylashtirish

```
python
КопироватьРедактировать
# 3D obyekt koordinatalari (z = 0, tekislikda)
object_points = np.array([
    [-0.5, -0.5, 0],
    [0.5, -0.5, 0],
    [0.5, 0.5, 0],
    [-0.5, 0.5, 0]
], dtype=np.float32)

video = cv2.VideoCapture(0)

while True:
    ret, frame = video.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    corners, ids, _ = aruco.detectMarkers(gray, aruco_dict,
parameters=parameters)

    if ids is not None:
        rvecs, tvecs, _ = cv2.aruco.estimatePoseSingleMarkers(corners, 0.05,
camera_matrix, dist_coeffs)

        for i in range(len(ids)):
            cv2.drawFrameAxes(frame, camera_matrix, dist_coeffs, rvecs[i],
tvecs[i], 0.05)

    cv2.imshow("3D AR Model Joylashtirish", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Marker ustida virtual 3D koordinatalar hosil qilinadi.

Afzallik: Haqiqiy muhitga virtual obyektlar joylashtirish mumkin.
Kamchilik: Kamera kalibrovkasi aniq bo‘lishi shart.

---

4. Virtual obyektlarni joylashtirish

Agar 3D modelni marker ustiga joylashtirmoqchi bo‘lsak, **OpenGL yoki Blender** dan foydalanish mumkin. OpenCV’ning `cv2.projectPoints()` funksiyasi orqali virtual obyektlarni markerga bog‘lash mumkin.

Xulosa

Biz OpenCV yordamida **3D obyektlarni kuzatish va AR effektlarini qo‘llash** mavzusini o‘rganib chiqdik. Endi siz:

✅ **Kamerani kalibrovka qilish**
✅ **ArUco markerlarini yaratish va kuzatish**
✅ **3D obyektlarni real tasvirga joylashtirish**

11-BO‘LIM: OPENCV YORDAMIDA VIDEO STABILIZATSIYASI VA TEBRANISHLARNI KAMAYTIRISH

Video stabilizatsiyasi **mobil kameralar, dronlar, xavfsizlik tizimlari va monitoring tizimlari** uchun muhim texnologiyalardan biri hisoblanadi. Ushbu bo‘limda quyidagilarni ko‘rib chiqamiz:

✅ **Optik oqim yordamida kadrlarni tekislash**
✅ **Homografiya va affin transformatsiyalar yordamida video stabilizatsiyasi**
✅ **Kalman filtri bilan silliqlashtirish (Kalman Filtering)**
✅ **Real vaqt rejimida video barqarorlashtirish**

---

1. Optik oqim yordamida kadrlarni tekislash

Optik oqim — bu **tasvirdagi piksel harakatlarini aniqlash** usuli bo‘lib, **Lucas-Kanade Optical Flow** modeli orqali amalga oshiriladi.

1.1. Lucas-Kanade optik oqimi yordamida kadrlarni tekislash

```
python
КопироватьРедактировать
import cv2
import numpy as np

# Videoni ochish
video = cv2.VideoCapture("shaky_video.mp4")

# Birinchi kadrni olish
ret, prev_frame = video.read()
prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)

# Lucas-Kanade optik oqim parametrlari
lk_params = dict(winSize=(15, 15), maxLevel=2,
criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Harakatni kuzatish uchun belgilangan nuqtalar
feature_params = dict(maxCorners=100, qualityLevel=0.3, minDistance=7,
blockSize=7)

prev_pts = cv2.goodFeaturesToTrack(prev_gray, mask=None, **feature_params)

while True:
    ret, frame = video.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Keyingi kadr uchun optik oqimni hisoblash
    next_pts, status, _ = cv2.calcOpticalFlowPyrLK(prev_gray, gray, prev_pts,
None, **lk_params)
```

```
    # Faqat muvaffaqiyatli topilgan nuqtalarni olish
    good_new = next_pts[status == 1]
    good_old = prev_pts[status == 1]

    # Transformatsiya matritsasini topish
    transform_matrix, _ = cv2.estimateAffinePartial2D(good_old, good_new)

    # Kadrni tekislash
    stabilized_frame = cv2.warpAffine(frame, transform_matrix,
(frame.shape[1], frame.shape[0]))

    cv2.imshow("Stabilizatsiya qilingan video", stabilized_frame)

    prev_gray = gray.copy()
    prev_pts = good_new.reshape(-1, 1, 2)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Kadrlar orasidagi siljishlar bartaraf etiladi, video silliq bo‘lib chiqadi.

Afzallik: Tez ishlaydi va real vaqt rejimida foydalanish mumkin.
Kamchilik: Juda katta harakatlarni barqaror qila olmaydi.

---

2. Homografiya va affiin transformatsiyalar yordamida video stabilizatsiyasi

Homografiya transformatsiyasi orqali har bir kadrni **oldingi kadr bilan moslashtirish** mumkin.

2.1. Homografiya yordamida stabilizatsiya

```
python
КопироватьРедактировать
# Videoni ochish
video = cv2.VideoCapture("shaky_video.mp4")

ret, prev_frame = video.read()
prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)
prev_pts = cv2.goodFeaturesToTrack(prev_gray, maxCorners=200,
qualityLevel=0.01, minDistance=30, blockSize=3)

while True:
    ret, frame = video.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    next_pts, status, _ = cv2.calcOpticalFlowPyrLK(prev_gray, gray, prev_pts,
None)

    good_old = prev_pts[status == 1]
    good_new = next_pts[status == 1]

    # Homografiya matritsasini hisoblash
    H, _ = cv2.findHomography(good_old, good_new, cv2.RANSAC)
```

```
    # Kadrni transformatsiya qilish
    stabilized_frame = cv2.warpPerspective(frame, H, (frame.shape[1],
frame.shape[0]))

    cv2.imshow("Homografiya bilan stabilizatsiya", stabilized_frame)

    prev_gray = gray.copy()
    prev_pts = good_new.reshape(-1, 1, 2)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Video silkinishlarini minimallashtirish uchun homografiya transformatsiyasi ishlatiladi.

Afzallik: Juda aniq natijalar beradi.
Kamchilik: Real vaqt rejimida ishlashi biroz sekin.

---

3. Kalman filtri yordamida video stabilizatsiyasi

Kalman filtri yordamida **video tebranishlarini silliqlashtirish** mumkin.

3.1. Kalman filtri orqali kadrlarni barqarorlashtirish

```
python
КопироватьРедактировать
import cv2
import numpy as np

# Kalman filtri
kalman = cv2.KalmanFilter(4, 2)
kalman.measurementMatrix = np.array([[1, 0, 0, 0], [0, 1, 0, 0]], np.float32)
kalman.transitionMatrix = np.array([[1, 0, 1, 0], [0, 1, 0, 1], [0, 0, 1, 0],
[0, 0, 0, 1]], np.float32)

video = cv2.VideoCapture("shaky_video.mp4")
ret, prev_frame = video.read()

while True:
    ret, frame = video.read()
    if not ret:
        break

    # Kalman filtrini ishlatish
    measured = np.array([[np.float32(frame.shape[1] / 2)],
[np.float32(frame.shape[0] / 2)]])
    predicted = kalman.predict()
    kalman.correct(measured)

    # Kadrni tekislash
    dx = int(predicted[0] - measured[0])
    dy = int(predicted[1] - measured[1])
    stabilized_frame = cv2.warpAffine(frame, np.float32([[1, 0, dx], [0, 1,
dy]]), (frame.shape[1], frame.shape[0]))
```

```
    cv2.imshow("Kalman bilan stabilizatsiya", stabilized_frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Kalman filtri video silkinishlarini **real vaqt rejimida silliqlashtiradi**.

Afzallik: Video stabilizatsiyasini real vaqt rejimida amalga oshiradi.
Kamchilik: Juda katta tebranishlarni to‘liq bartaraf eta olmaydi.

Xulosa

Biz OpenCV yordamida **video stabilizatsiyasi va silkinishlarni kamaytirish** bo‘yicha muhim usullarni o‘rganib chiqdik. Endi siz:

✅ **Optik oqim yordamida kadrlarni tekislash**
✅ **Homografiya orqali harakatni barqarorlashtirish**
✅ **Kalman filtri yordamida video silkinishlarini silliqlashtirish**

12-BO‘LIM: OPENCV YORDAMIDA REAL VAQTDA OBYEKTLARNI ANIQLASH VA KUZATISH

Obyektlarni real vaqt rejimida aniqlash va kuzatish **kuzatuv tizimlari, robototexnika, xavfsizlik va transport monitoringi** kabi sohalarda keng qo‘llaniladi. Ushbu bo‘limda quyidagilarni o‘rganamiz:

✅ **Hareketlanuvchi ob’ektlarni aniqlash**
✅ **CSRT, MOSSE va KCF trackerlar yordamida ob’ektlarni kuzatish**
✅ **Deep Learning asosida real vaqt kuzatuv tizimlarini yaratish**
✅ **YOLO modeli yordamida ko‘p obyektli kuzatuv**

---

1. Harakatlanuvchi ob’ektlarni aniqlash

Obyektlarni aniqlash uchun fonni olib tashlash va tasvirlar orasidagi farqni hisoblash usuli ishlatiladi.

1.1. Background Subtraction (Fonni ajratish) yordamida aniqlash

```
python
КопироватьРедактировать
import cv2

# MOG2 modelidan foydalanish
fgbg = cv2.createBackgroundSubtractorMOG2()
```

```
video = cv2.VideoCapture("video.mp4")

while True:
    ret, frame = video.read()
    if not ret:
        break

    fgmask = fgbg.apply(frame)

    cv2.imshow("Harakatni aniqlash", fgmask)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Harakatlanayotgan obyektlar oq rangda, fon esa qora rangda aks etadi.

Afzallik: Oddiy va tez ishlaydi.
Kamchilik: Yorug‘lik o‘zgarishlariga sezgir.

---

2. Obyektlarni kuzatish uchun CSRT, MOSSE va KCF trackerlaridan foydalanish

OpenCV’da ob’ektlarni kuzatish uchun quyidagi algoritmlar mavjud:

- **CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability)** – yuqori aniqlik
- **MOSSE (Minimum Output Sum of Squared Error Filter)** – eng tezkor kuzatuv
- **KCF (Kernelized Correlation Filters)** – muvozanatlangan natija

---

2.1. CSRT tracker yordamida ob’ektni kuzatish

```
python
КопироватьРедактировать
import cv2

# Video ochish
video = cv2.VideoCapture("video.mp4")

# CSRT tracker yaratish
tracker = cv2.TrackerCSRT_create()

# Birinchi kadrni olish
ret, frame = video.read()
bbox = cv2.selectROI("Obyektni tanlang", frame, False)

# Tracker boshlash
tracker.init(frame, bbox)

while True:
    ret, frame = video.read()
    if not ret:
        break

    success, bbox = tracker.update(frame)
```

```
    if success:
        x, y, w, h = map(int, bbox)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    cv2.imshow("Obyektni kuzatish", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: Obyekt harakatlanishi bilan u yashil to‘rtburchak yordamida kuzatiladi.

Afzallik: Aniqligi yuqori.
Kamchilik: Sekin ishlaydi, real vaqt uchun eng samarali emas.

2.2. MOSSE tracker yordamida ob’ektni tezkor kuzatish

```
python
КопироватьРедактировать
tracker = cv2.TrackerMOSSE_create()
```

Afzallik: Juda tez ishlaydi.
Kamchilik: Aniqligi past.

---

3. YOLO modeli yordamida real vaqt kuzatuv tizimi

YOLO (**You Only Look Once**) — Deep Learning asosida real vaqt rejimida bir nechta obyektlarni aniqlash va kuzatish uchun ishlatiladi.

3.1. YOLO modelini yuklash

Avval quyidagi fayllarni yuklab olish kerak:

- **yolov3.weights** – oldindan o‘qitilgan model
- **yolov3.cfg** – model arxitekturasi
- **coco.names** – obyekt turlari

3.2. YOLO yordamida real vaqt kuzatish

```
python
КопироватьРедактировать
import cv2
import numpy as np

# YOLO modelini yuklash
```

```
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
layer_names = net.getUnconnectedOutLayersNames()

# Obyekt nomlarini yuklash
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]

video = cv2.VideoCapture(0)

while True:
    ret, frame = video.read()
    if not ret:
        break

    # YOLO uchun tasvirni tayyorlash
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), swapRB=True,
crop=False)
    net.setInput(blob)
    outs = net.forward(layer_names)

    # Obyektlarni aniqlash
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]

            if confidence > 0.5:
                x, y, w, h = detection[:4] * np.array([frame.shape[1],
frame.shape[0], frame.shape[1], frame.shape[0]])
                x, y, w, h = int(x - w / 2), int(y - h / 2), int(w), int(h)
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv2.putText(frame, f"{classes[class_id]}: {confidence:.2f}",
(x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    cv2.imshow("YOLO obyekt kuzatish", frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

video.release()
cv2.destroyAllWindows()
```

Natija: YOLO **bir nechta obyektlarni aniq aniqlaydi va kuzatadi**.

Afzallik: Juda aniq natijalar beradi, ko‘p obyektni bir vaqtning o‘zida aniqlash mumkin.
Kamchilik: Tez ishlashi uchun GPU talab etiladi.

Xulosa

Biz OpenCV yordamida **real vaqt rejimida ob’ektlarni aniqlash va kuzatish** mavzusini o‘rganib chiqdik. Endi siz:

✅ **Fonni olib tashlash yordamida harakatni aniqlash**
✅ **CSRT, MOSSE va KCF trackerlar yordamida ob’ektlarni kuzatish**
✅ **YOLO modeli orqali bir nechta obyektni real vaqt rejimida kuzatish**