6 비전 에이전트

컴퓨터비전이 쓸모 있으려면 환경environment과 상호작용interaction해야 한다. [그림 6-1]이 보여주는 배달 드론, 공장 조립라인에서 불량을 걸러내는 검사 시스템, 상차림을 인식하여 다이어트 조언을 하는 스마트폰 앱은 환경과 활발히 상호작용한다.

[그림 6-1] 환경과 상호작용하는 컴퓨터비전 시스템

컴퓨터비전이 환경과 상호작용하려면 환경에서 영상을 획득하는 기능과 처리한 결과에 따라 적절히 환경에 영향을 미치는 기능이 추가되어야 한다. 2~5장은 배운 이론을 구현하고 제대로 동작하는지 확인할 목적이 강해서 프로그래밍할 때 이런 기능을 고려하지 않았다. 이번 장에서는 비전 프로그램에 사용자 인터페이스를 붙여 환경과 상호작용할 수 있게 확장한다.

이번 장에서 해볼 프로그래밍 실습은 컴퓨터비전에 대한 흥미를 배가시킬 것이며 시장 경쟁력을 갖춘 컴퓨터비전 제품을 개발해보려는 욕구를 불러일으킬 수 있다. 게다가 컴퓨터비전 분야의 프로그램 개발 능력을 크게 향상시켜줄 것이다.

6.1 지능 에이전트로서 비전 에이전트

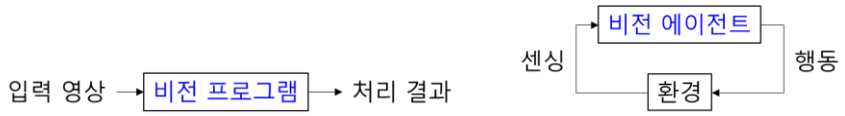
경제학은 사람을 합리적 에이전트rational agent로 바라본다. 이 관점은 사람을 항상 최적의 의사결정을 하려 노력하는 주체로 본다. 합리적 에이전트라는 개념을 컴퓨터에 적용하면 지능 에이전트intelligent agent가 된다. 지능 에이전트도 최적의 의사결정을 하려 노력한다.

세계적으로 유명한 인공지능 교과서 『Artificial Intelligence: A Modern Approach(4판)』에서 러셀은 지능 에이전트를 다음과 같이 정의한다 [Russel2021].

> "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators" 센서를 통해 환경을 지각하고 액츄에이터를 통해 환경에 행동을 가한다고 볼 수 있는 모든 것

지능 에이전트는 비전, 자연언어처리, 지식 표현, 학습, 추론 등의 기능을 종합적으로 발휘해야 한다. 이 책은 비전에 특화된 지능 에이전트를 비전 에이전트vision agent라 부른다. 대략적으로 말하면 앞의 정의에서 sensors를 vision sensors로 바꾸면 비전 에이전트의 정의가 된다.

2~5장에서 수행한 프로그래밍 실습은 환경과 상호작용이 없는 [그림 6-2(a)]의 비전 프로그램이다. [그림 6-2(b)]는 환경과 상호작용하는 과정을 추가한 비전 에이전트다. 비전 에이전트가 환경과 제대로 상호작용하려면 인식한 결과에 따라 로봇을 움직인다거나 자동차의 브레이크와 조향 장치를 조작하는 데까지 확장되어야 한다. 환경에 영향을 미치는 물리적 장치를 액추에이터actuator라 부른다. 물리적 액추에이터의 제어는 이 책의 범위를 넘어선다.



(a) 비전 프로그램 (b) 비전 에이전트

[그림 6-2] 환경과 상호작용하는 비전 에이전트

이 책은 2~5장에서 만든 프로그램에 사용자 인터페이스를 붙여 영상 획득 과정을 보강하고 처리 결과에 따라 적절한 행위를 모방하는 비전 에이전트로 확장한다. 6.3절은 4.5.2절에서 실습해본 GrabCut을 반복적으로 적용하여 원하는 모양을 정교하게 오려낼 수 있게 [프로그램 4-7]을 확장한다. 사용자는 마우스로 페인팅하는 작업을 반복하여 원하는 물체를 정교하게 오려낸다. 6.4절에서는 도로에 설치된 교통약자 보호 표지판을 인식하고 운전자에게 경고 신호를 보내는 과업을 모방하는 프로그래밍 실습을 한다. 이때 5.4절에서 공부한 SIFT라는 지역 특징을 이용하여 표지판을 인식한다. 6.5절에서는 웹캠으로 입력 받은 여러 장의 영상을 봉합하여 파노라마 영상을 제작하는 비전 에이전트를 만들어 본다. 6.6절은 영상에 엠보싱, 카툰, 스케치, 유화의 특수 효과를 발휘하는 프로그래밍 실습을 한다. 웹캠을 통해 들어오는 비디오에도 적용한다.

비전 에이전트를 제작하려면 편리한 사용자 인터페이스가 필수다. 파이썬의 PyQt 모듈을 사용해 그래픽 사용자 인터페이스를 제작하는 일을 시작으로 비전 에이전트를 제작하는 신나는 세계로 들어간다.

6.2 PyQt를 이용한 사용자 인터페이스

비전 프로그램을 비전 에이전트로 확장하려면 적절한 그래픽 사용자 인터페이스GUI(graphical user interface)

를 추가해야 한다. 파이썬에서 GUI 프로그래밍하려면 tkinter 또는 PyQt 모듈을 사용한다. 이 책은 PyQt를 사용한다. 아나콘다를 설치하면 PyQt가 따라 오므로 별도로 설치할 필요가 없다. PyQT를 공부하려면 [Fitzpatrick2021] 또는 [김민휘2020]을 참조한다. [김민휘2020]은 인터넷에 무료 버전이 공개되어 있다.[1] PyQt의 공식 사이트 https://doc.qt.io/qtforpython/가 제공하는 문서를 참조해도 된다.

PyQt 기초 프로그래밍

[프로그램 6-1]은 PyQt를 이용한 간단한 GUI 프로그램이다. 처음 만나는 GUI 프로그램이니 코드를 살펴보기 전에 프로그램을 실행하여 제대로 동작하는지 먼저 확인해보자. 프로그램을 실행하면 [짧게 삑], [길게 삑], [나가기]라는 3개의 버튼을 가진 윈도우가 뜬다. 마우스로 [짧게 삑] 버튼을 클릭하면 0.5초 동안 삑 소리가 나며, [길게 삑]을 클릭하면 3초 동안 삑 소리가 난다. [나가기]를 클릭하면 윈도우가 닫히며 프로그램이 끝난다.

이제 [프로그램 6-1]을 행 별로 꼼꼼히 살펴보자. 1~3행은 PyQt5.QtWidgets, sys, winsound 모듈을 불러온다. winsound는 삑 소리를 내는데 사용한다. 5~34행은 PyQt로 GUI를 제작하는 일을 지원하는 클래스를 선언한다. 클래스 선언이 시작되는 5행에서 클래스 이름을 적절하게 붙이면 되는데 삑 소리를 내주는 클래스라 BeepSound라 하였다. 매개변수에 QMainWindow가 있는데, QMainWindow 클래스를 상속받겠다는 뜻이다. QMainWindow는 윈도우를 생성하고 관리해주는 함수들을 제공하는 핵심적인 클래스다. 6~23행은 BeepSound 클래스로 객체를 생성하면 자동으로 실행되는 생성자 함수 __init__을 정의한다. 8행은 윈도우의 제목 막대에 '삑 소리 내기'라고 쓴다. 실행 결과에 있는 윈도우의 제목 막대에 이 제목이 나타났음을 확인하자. 9행은 윈도우를 화면의 (200,200) 위치에 초기 배치하고 너비와 높이를 각각 500과 100으로 설정하라는 명령어다.

11~14행은 QPushButton 함수로 버튼 3개와 QLabel 함수로 레이블 1개를 만든다. 버튼과 레이블은 위젯$_{widget}$의 일종인데, PyQt는 이들 외에도 QCheckBox, QRadioButton, QComboBox, QLineEdit 등 20여개의 유용한 위젯을 제공한다. 11행은 '짧게 삑'이라고 표시된 버튼을 만들어 shortBeepButton 객체에 저장한다. 12행과 13행도 같은 방식으로 버튼을 만들어 longBeepButton과 quitButton 객체에 저장한다. 14행은 '환영합니다!'라고 쓴 레이블을 만들어 label 객체에 저장한다. 이때 버튼 객체와 달리 앞에 self를 붙였는데, self를 붙이면 멤버 변수가 된다. 멤버 변수는 클래스 어느 곳에서든 접근이 가능하며 클래스로 생성한 객체에서도 접근이 가능하다. 이 프로그램에서는 shortBeepFunction 함수와 longBeepFunction 함수가 label에 접근해야 해서 멤버 변수로 선언한다. 16~19행은 4개 위젯의 위치와 크기를 지정한다. 예를 들어 16행은 shortBeepButton

[1] 무료 책을 공개하는 위키독스 사이트에 있다. URL은 https://wikidocs.net/book/2165이다.

을 윈도우의 (10,10) 위치에 배치하고 너비와 높이를 100과 30으로 설정한다. 21~23행은 사용자가 버튼을 클릭했을 때 수행할 콜백 함수를 지정한다. 예를 들어 21행은 shortBeepButton을 클릭하면 25~27행에 정의되어 있는 shortBeepFunction 함수를 실행한다. shortBeepFunction은 label.setText 명령어로 레이블 위젯에 지정한 텍스트를 쓰고 winsound.Beep(1000,500) 명령어로 주파수 1000인 삑 소리를 500밀리초 동안 들려준다.

이제 메인에 해당하는 36~39행을 살펴보자. 36행은 Qt 실행에 필요한 객체 app을 생성한다. 37행은 BeepSound 클래스의 객체 win을 생성한다. 이때 6~23행에 정의된 BeepSound 클래스의 생성자 함수 __init__이 자동으로 실행된다. 다시 말해 '삑 소리 내기' 제목의 윈도우를 생성하고, 4개 위젯이 만들어지고, 콜백 함수가 등록된다. 38행은 win에 해당하는 윈도우를 실제로 화면에 보인다. 39행은 무한 루프를 돌아 프로그램이 끝나는 것을 방지한다. 39행이 없으면 win 객체에 해당하는 윈도우를 화면에 띄우고 순간 프로그램이 끝나버려 사용자는 프로그램과 상호작용할 기회가 없다. 프로그램 종료는 [나가기] 버튼을 통해 달성된다. [나가기]를 클릭하면 23행에서 등록해 두었던 콜백 함수 quitFunction이 호출되고 34행의 close 함수가 실행되어 프로그램이 종료된다.

[프로그램 6-1] PyQt로 간단한 GUI 만들기(버튼을 클릭하면 삑 소리 들려주기)

```
1   from PyQt5.QtWidgets import *
2   import sys
3   import winsound
4
5   class BeepSound(QMainWindow):
6       def __init__(self) :
7           super().__init__()
8           self.setWindowTitle('삑 소리 내기') # 윈도우 이름과 위치 지정
9           self.setGeometry(200,200,500,100)
10
11          shortBeepButton=QPushButton('짧게 삑',self) # 버튼 생성
12          longBeepButton=QPushButton('길게 삑',self)
13          quitButton=QPushButton('나가기',self)
14          self.label=QLabel('환영합니다!',self)
15
16          shortBeepButton.setGeometry(10,10,100,30) # 버튼 위치와 크기 지정
17          longBeepButton.setGeometry(110,10,100,30)
18          quitButton.setGeometry(210,10,100,30)
19          self.label.setGeometry(10,40,500,70)
20
```

```
21              shortBeepButton.clicked.connect(self.shortBeepFunction) # 콜백 함수 지정
22              longBeepButton.clicked.connect(self.longBeepFunction)
23              quitButton.clicked.connect(self.quitFunction)
24
25          def shortBeepFunction(self):
26              self.label.setText('주파수 1000으로 0.5초 동안 삑 소리를 냅니다.')
27              winsound.Beep(1000,500)
28
29          def longBeepFunction(self):
30              self.label.setText('주파수 1000으로 3초 동안 삑 소리를 냅니다.')
31              winsound.Beep(1000,3000)
32
33          def quitFunction(self):
34              self.close()
35
36  app=QApplication(sys.argv)
37  win=BeepSound()
38  win.show()
39  app.exec_()
```

OpenCV에 PyQt 붙이기

이제 OpenCV에 PyQt의 GUI를 붙여 비전 프로그램을 확장해보자. [프로그램 6-2]는 비디오를 활성화하고 비디오에서 프레임을 획득하고 저장하는 간단한 기능을 제공한다. 먼저 프로그램의 큰 구조를 살펴보자. 5~47행은 GUI 제작을 지원하는 Video 클래스를 선언한다. Video 클래스 내부에 있는 6~24행은 Video 클래스의 생성자 함수 __init__이고, 26~47행은 버튼을 클릭하면 실행될 콜백 함수 4개다. 콜백 함수는 OpenCV 함수를 여러 군데서 사용하고 있다. 프로그램의 메인에 해당하는 49~52행은 [프로그램 6-1]의 36~39행과 같다.

먼저 6~24행의 생성자 함수 __init__을 살펴보자. 8행은 윈도우의 제목 막대에 '비디오에서 프레임

수집'이라고 쓴다. 9행은 화면에 나타날 윈도우의 위치와 크기를 설정한다. 11~14행은 QPushButton 함수로 버튼 4개를 만든다. 16~19행은 4개 버튼의 위치와 크기를 지정한다. 21~24행은 사용자가 버튼을 클릭했을 때 수행할 콜백 함수를 지정한다. [프로그램 6-1]과 논리 흐름이 같다. 프로그램 패턴을 머리 속에 각인시킬 필요가 있다.

4개 버튼을 지원하는 콜백 함수의 동작을 하나씩 살펴보자. 21행은 videoFunction을 videoButton의 콜백 함수로 등록한다. 다시 말해 <비디오 켜기>라는 videoButton이 클릭되는 순간 videoFunction이 호출되도록 등록한다. 26~34행의 vidioFunction은 OpenCV 함수를 이용하여 웹캠으로부터 비디오를 받아 윈도우에 디스플레이하는 일을 한다. videoFunction의 코드는 [프로그램 2-4]를 약간 수정한 것이다. 27행은 웹캠과 연결을 시도하고, 28행은 연결이 안됐을 때 오류 메시지를 출력하고 프로그램을 끝낸다. 성공적으로 연결되면, 30~34행은 루프를 반복하면서 비디오에서 프레임을 획득하여 frame 변수에 저장하고 'video display'라는 윈도우에 표시한다. cap과 frame 변수 앞에 self를 붙여 멤버 변수로 선언했는데, 다른 함수와 공유할 필요가 있어서다. cap 변수는 <나가기> 버튼이 클릭되었 때 quitFunction의 45행에서 비디오 연결을 끊는데 사용된다. frame 변수는 <프레임 잡기> 버튼을 클릭한 순간 captureFunction의 37행이 그 순간의 프레임을 capturedFrame 변수에 저장할 때 사용된다.

22행은 captureFunction을 <프레임 잡기>라는 captureButton의 콜백 함수로 등록한다. captureFunction의 37행은 비디오 프레임을 저장한 frame을 capturedFrame 변수에 저장한다. 38행은 윈도우에 디스플레이한다.

23행은 saveFunction을 <프레임 저장>이라는 saveButton의 콜백 함수로 등록한다. saveFunction은 41행에서 PyQt가 제공하는 QFileDialog.getSaveFileName 함수를 사용하여 사용자가 파일을 저장할 곳을 브라우징하고 파일 이름을 지정할 수 있게 한다. 두번째 인수 '파일 저장'은 브라우징 윈도우의 제목을 지정하고 세번째 인수 './'는 현재 폴더에서 브라우징하라고 지시한다. QFileDialog.getSaveFileName 함수는 사용자가 입력한 파일 이름을 반환하는데 41행은 파일 이름을 fname에 저장한다. 42행은 capturedFrame에 저장해 두었던 프레임을 사용자가 지정한 파일 이름으로 저장한다. 이때 fname이 튜플이기 때문에 fname[0]으로 지정한다.

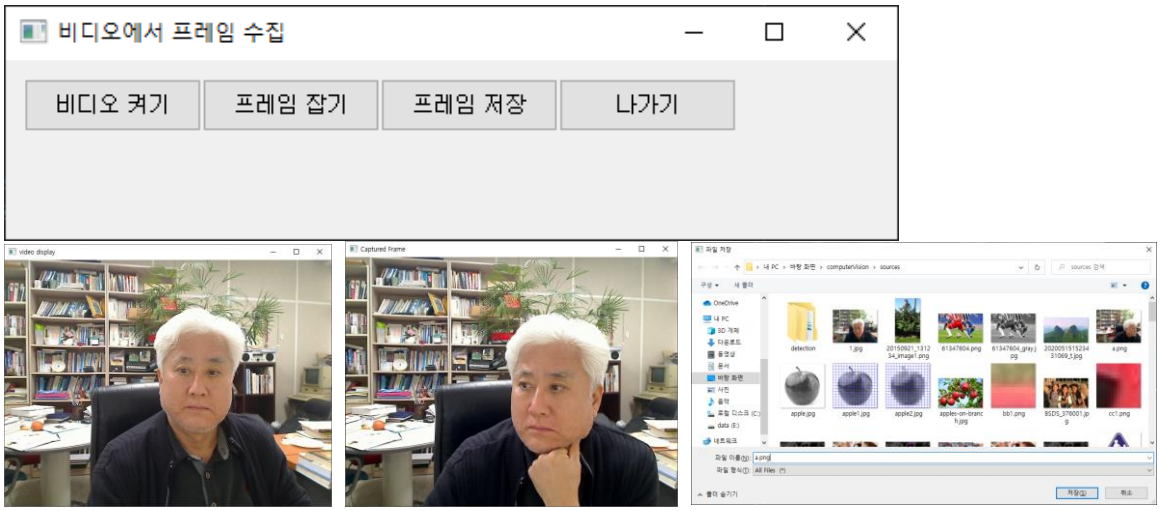
24행은 quitFunction을 <나가기>라는 quitButton의 콜백 함수로 등록한다. quitFunction은 45행에서 비디오 연결을 끊고, 46행에서 OpenCV가 연 모든 윈도우를 닫고, 47행에서 프로그램을 끝낸다.

[프로그램 6-2] OpenCV에 GUI 붙이기(비디오에서 프레임을 잡아 저장하기)

```
1  from PyQt5.QtWidgets import *
2  import sys
3  import cv2 as cv
```

```
4
5   class Video(QMainWindow):
6       def __init__(self) :
7           super().__init__()
8           self.setWindowTitle('비디오에서 프레임 수집') # 윈도우 이름과 위치 지정
9           self.setGeometry(200,200,500,100)
10
11          videoButton=QPushButton('비디오 켜기',self) # 버튼 생성
12          captureButton=QPushButton('프레임 잡기',self)
13          saveButton=QPushButton('프레임 저장',self)
14          quitButton=QPushButton('나가기',self)
15
16          videoButton.setGeometry(10,10,100,30) # 버튼 위치와 크기 지정
17          captureButton.setGeometry(110,10,100,30)
18          saveButton.setGeometry(210,10,100,30)
19          quitButton.setGeometry(310,10,100,30)
20
21          videoButton.clicked.connect(self.videoFunction) # 콜백 함수 지정
22          captureButton.clicked.connect(self.captureFunction)
23          saveButton.clicked.connect(self.saveFunction)
24          quitButton.clicked.connect(self.quitFunction)
25
26      def videoFunction(self):
27          self.cap=cv.VideoCapture(0,cv.CAP_DSHOW) # 카메라와 연결 시도
28          if not self.cap.isOpened(): self.close()
29
30          while True:
31              ret,self.frame=self.cap.read()
32              if not ret: break
33              cv.imshow('video display',self.frame)
34              cv.waitKey(1)
35
36      def captureFunction(self):
37          self.capturedFrame=self.frame
38          cv.imshow('Captured Frame',self.capturedFrame)
39
40      def saveFunction(self): # 파일 저장
```

```
41              fname=QFileDialog.getSaveFileName(self,'파일 저장','./')
42              cv.imwrite(fname[0],self.capturedFrame)
43
44          def quitFunction(self):
45              self.cap.release() # 카메라와 연결을 끊음
46              cv.destroyAllWindows()
47              self.close()
48
49  app=QApplication(sys.argv)
50  win=Video()
51  win.show()
52  app.exec_()
```



지금까지 PyQt를 이용하여 GUI 프로그래밍을 연습했다. 이제 2~5장에서 배운 내용에 GUI를 추가하여 비전 에이전트로 확장한다.

6.6 비전 에이전트4: 특수 효과

3장에서 [그림 3-17(c)]의 필터로 컨볼루션을 수행하면 엠보싱이라는 특수 효과를 거둘 수 있음을 [프로그램 3-7]을 통해 확인했다. 여기서는 엠보싱 뿐 아니라 카툰, 연필 스케치, 유화라는 특수

효과로 확장하고 GUI를 붙인 비전 에이전트를 제작한다.

6.6.1 특수 효과

OpenCV는 특수 효과를 위한 흥미로운 함수를 여럿 제공한다. stylization과 pencilSketch 함수는 OpenCV 매뉴얼의 Non-Photorealistic Rendering 부분에서 설명하고 oilPainting 함수는 Additional photo processing algorithms 부분에서 설명한다.[2] stylization은 카툰 효과, pencilSketch는 연필로 스케치한 효과, oilPainting은 유화 효과를 제공한다. stylization과 pencilSketch는 기본값이 설정되어 있어 생략해도 되지만 oilPainting은 기본값이 없어 꼭 지정해야 한다. 이들 특수 효과에 대한 상세한 내용을 공부하려는 독자는 [Gastal2011]을 참조한다. OpenCV 공식 사이트가 제공하는 함수 선언은 다음과 같다.

cv.stylization(src, sigma_s=60, sigma_r=0.45) → dst

매개변수:

src: 입력 영상 (8-비트 3-채널 입력 영상)

sigma_s: 스무딩을 위한 가우시안의 표준편차 σ (0~200 범위)

sigma_r: 양방향 필터가 사용하는 두번째 가우시안의 표준편차 σ (0~1 범위)

반환값:

dst: 특수 효과 처리된 영상 (8-비트 3-채널 영상)

cv.pencilSketch(src, sigma_s=60, sigma_r=0.07, shade_factor=0.02) → dst1, dst2

매개변수:

src: 입력 영상 (8-비트 3-채널 입력 영상)

sigma_s: 스무딩을 위한 가우시안의 표준편차 σ (0~200 범위)

sigma_r: 양방향 필터가 사용하는 두번째 가우시안의 표준편차 σ (0~1 범위)

shade_factor: 밝은 정도??? (0~0.1 범위)

반환값:

---

[2] stylization과 pencilSketch는 https://docs.opencv.org/3.4/df/dac/group__photo__render.html과 https://learnopencv.com/non-photorealistic-rendering-using-opencv-python-c/을, oilPainting은 https://docs.opencv.org/4.x/de/daa/group__xphoto.html을 참조한다.

dst1: 특수 효과 처리된 명암 영상 (8-비트 1-채널 영상)

dst2: 특수 효과 처리된 컬러 영상 (8-비트 3-채널 영상)

cv.xphoto.oilPainting(src, size, dynRatio,, code) → dst

매개변수:

src: 입력 영상 (8-비트 3-채널 또는 1-채널 입력 영상)

size: 가우시안의 표준편차 ???

dynRatio: 히스토그램의 칸의 수를 정하는데 씀. 클수록 세부 내용 사라짐 ???

code: 사용할 컬러 공간을 지정 ???

반환값:

dst: 특수 효과 처리된 영상 (입력 영상과 같은 모양)

가우시안 필터로 컨볼루션을 수행하면 물체 경계를 포함하여 영상 전체가 흐릿해진다. 때로 물체 경계의 명암 대비를 유지하면서 다른 부분만 흐릿하게 만들 필요가 있다. 이런 조건을 만족하는 필터를 에지 보존$_{\text{edge-preserving}}$ 필터라 부른다. stylization과 pencilSketch 함수는 에지 보존 필터를 활용한다. 식 (6.1)이 정의하는 양방향 필터$_{\text{bilateral filter}}$는 가장 널리 쓰이는 에지 보존 필터다. 간편성을 위해 1차원 컨볼루션으로 설명한다.

$$f'(x) = \sum_{i=-(w-1)/2}^{(w-1)/2} g_s(i) g_r\big(f(x) - f(x+i)\big) f(x+i) \qquad (6.1)$$

식 (6.1)은 식 (3.7)의 필터 u를 가우시안 필터로 대체한 것이다. 가우시안은 [그림 3-18]이 보여주는 바와 같이 중앙이 가장 크고 중앙에서 멀어질수록 작아지는 함수다. 식 (6.1)에 두 개의 가우시안 g_s와 g_r이 있다. $g_s(i)$는 i=0일 때 가장 크고 i가 0에서 멀어질수록 작아진다. $g_r\big(f(x) - f(x+i)\big)$는 필터가 씌워진 현재 화소 x의 값 $f(x)$와 이웃 화소의 값 $f(x+i)$의 차이를 매개변수로 가진다. 두 화소의 값이 같아 차이가 0이라면 $g_r(0)$이 되어 가장 큰 값을 가진다. 반대로 두 화소의 값이 크게 달라 차이가 크다면 g_r은 작아진다. 식 (6.1)은 g_r로 인해 i에 해당하는 화소는 중앙 화소와 값이 많이 다를수록 낮은 가중치를 가진다. 결국 물체 경계에서 서로 다른 물체에 속한 화소는 서로에게 영향력이 낮아져 결국 에지를 잘 보존한다. stylization과 pencilSketch 함수의 sigma_s와 sigma_r 매개변수는 식 (6.1)의 g_s와 g_r의 표준편차다. sigma_s는 클수록 영상을 흐릿하게 만드는 효과가 크다. sigma_r은 작을수록 에지 보존 효과가 크다.

6.6.2 정지 영상의 특수 효과

[프로그램 6-7]의 실행 결과를 보고 사용자 인터페이스를 확인해보자. <사진 읽기> 버튼은 폴더에서 사진 영상을 선택하고 읽어온다. <엠보싱>, <카툰>, <연필 스케치>, <유화> 버튼은 4가지 특수 효과를 지원한다. <저장하기> 버튼은 특수 효과 처리된 영상을 저장해주는데, 사용자는 그 밑에 설치된 콤보박스를 통해 어떤 특수 효과를 저장할지 선택할 수 있다.

프로그램의 전체 구조를 살펴보자. 6~82행은 SpecialEffect 클래스를 선언한다. 7~39행은 생성자 함수 __init__이다. 41~82행은 7개 버튼에 해당하는 콜백 함수다. 84~87행은 프로그램의 메인에 해당한다.

7~39행의 __init__을 살펴본다. 9행은 윈도우의 제목 막대에 '사진 특수 효과'라고 쓰고 10행은 윈도우 위치와 크기를 설정한다. 12~21행은 QPushButton 함수로 버튼 7개, QLabel 함수로 레이블 1개, QComboBox 함수로 콤보박스 1개를 만든다. 처음 사용하는 콤보박스를 살펴보자. 18행은 콤보박스를 만들고 19행은 5개 선택사항을 지정한다. 23~31행은 버튼과 레이블, 콤보박스 위젯의 위치와 크기를 지정한다. 33~39행은 버튼이 클릭됐을 때 수행할 콜백 함수를 등록한다.

7개 버튼에 해당하는 콜백 함수를 하나씩 살펴보자. 41~46행은 <사진 읽기> 버튼이 클릭되면 실행되는 pictureOpenFunction이다. 42~43행은 폴더에서 사진 파일을 브라우징하여 선택하고 읽어온다. 46행은 영상을 윈도우에 디스플레이한다.

48~55행은 <엠보싱> 버튼이 클릭되면 실행되는 embossFunction이다. [프로그램 3-7]의 13~18행과 비슷하다. 49행은 엠보싱 필터를 정의한다. 51~53행은 명암 영상으로 변환하고 컨볼루션을 적용한다. 값의 범위를 다루기 위해 16비트 정수로 변환하는 기법은 [프로그램 3-7]의 설명을 참조한다. 57~59행은 <카툰> 버튼이 클릭되면 실행되는 cartoonFunction이다. 6.6.1절에서 소개한 stylization 함수를 사용하며, 매개변수는 기본값으로 설정했다. 61~64행은 <연필 스케치> 버튼이 클릭되면 실행되는 sketchFunction이다. 6.6.1절에서 소개한 pencilSketch 함수를 사용하며, 매개변수는 기본값으로 설정했다. pencilSketch 함수는 명암 스케치와 컬러 스케치의 2장의 영상을 반환하기 때문에 이들을 두 윈도우에 따로 디스플레이한다. 66~68행은 <유화> 버튼이 클릭되면 실행되는 oilFunction이다. 6.6.1절에서 소개한 oilPainting 함수를 사용하며, size와 dynRatio 매개변수는 각각 10과 1로 설정하고, code 매개변수는 cv.COLOR_BGR2Lab으로 설정한다.

70~78행은 <저장하기> 버튼이 클릭되면 실행되는 saveFunction이다. 사용자는 71행을 통해 영상 이름을 지정한다. 73행은 사용자가 콤보박스에서 선택한 특수 효과의 인덱스를 알아낸다. 74~78행은 사용자의 선택에 따라 특수 효과 영상을 저장한다. 80~82행은 <나가기> 버튼이 클릭되었을 때 실행되는 quitFunction이다.

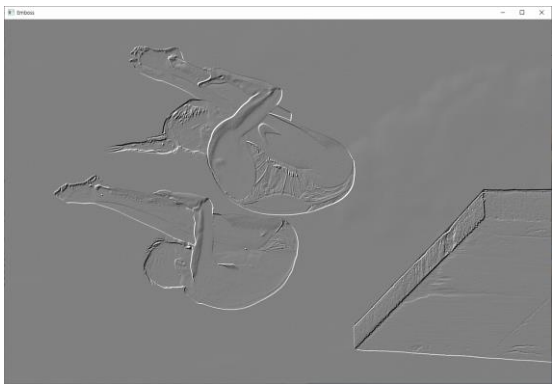
[프로그램 6-7] 사진 특수 효과

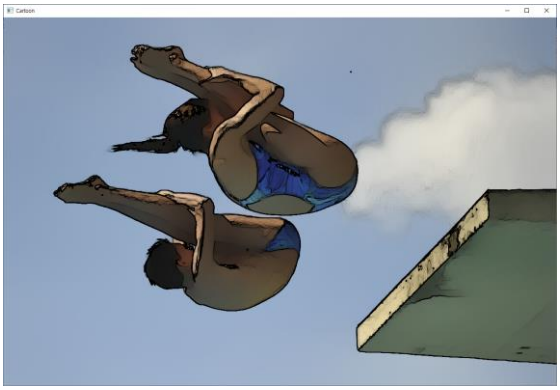
```
1   import cv2 as cv
```

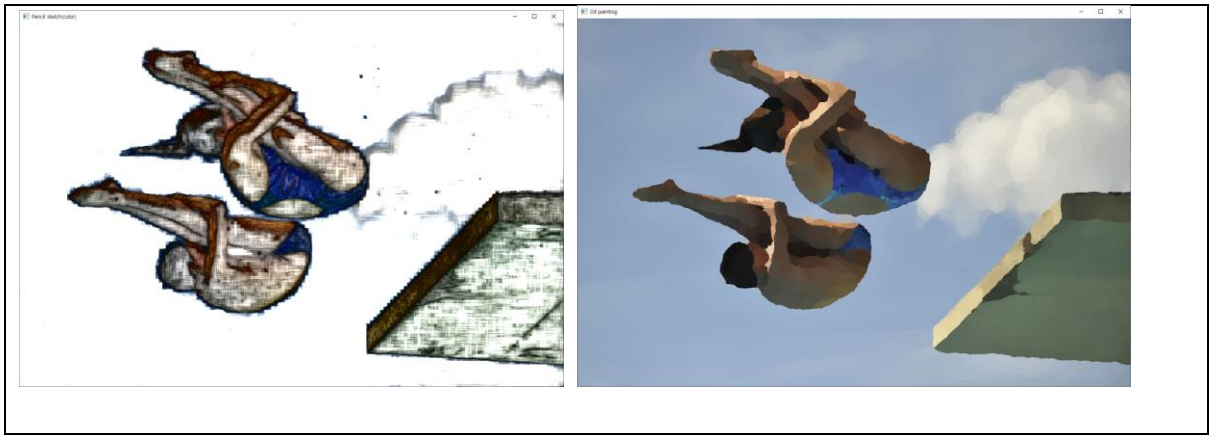
```
2   import numpy as np
3   from PyQt5.QtWidgets import *
4   import sys
5
6   class SpecialEffect(QMainWindow):
7       def __init__(self):
8           super().__init__()
9           self.setWindowTitle('사진 특수 효과')
10          self.setGeometry(200,200,800,200)
11
12          pictureButton=QPushButton('사진 읽기',self)
13          embossButton=QPushButton('엠보싱',self)
14          cartoonButton=QPushButton('카툰',self)
15          sketchButton=QPushButton('연필 스케치',self)
16          oilButton=QPushButton('유화',self)
17          saveButton=QPushButton('저장하기',self)
18          self.pickCombo=QComboBox(self)
19          self.pickCombo.addItems(['엠보싱','카툰','연필  스케치(명암)','연필  스케치(컬러)','유
    화'])
20          quitButton=QPushButton('나가기',self)
21          self.label=QLabel('환영합니다!',self)
22
23          pictureButton.setGeometry(10,10,100,30)
24          embossButton.setGeometry(110,10,100,30)
25          cartoonButton.setGeometry(210,10,100,30)
26          sketchButton.setGeometry(310,10,100,30)
27          oilButton.setGeometry(410,10,100,30)
28          saveButton.setGeometry(510,10,100,30)
29          self.pickCombo.setGeometry(510,40,110,30)
30          quitButton.setGeometry(620,10,100,30)
31          self.label.setGeometry(10,40,500,170)
32
33          pictureButton.clicked.connect(self.pictureOpenFunction)
34          embossButton.clicked.connect(self.embossFunction)
35          cartoonButton.clicked.connect(self.cartoonFunction)
36          sketchButton.clicked.connect(self.sketchFunction)
37          oilButton.clicked.connect(self.oilFunction)
```

```
38              saveButton.clicked.connect(self.saveFunction)
39              quitButton.clicked.connect(self.quitFunction)
40
41          def pictureOpenFunction(self):
42              fname=QFileDialog.getOpenFileName(self,'사진 읽기','./')
43              self.img=cv.imread(fname[0])
44              if self.img is None: sys.exit('파일을 찾을 수 없습니다.')
45
46              cv.imshow('Painting',self.img)
47
48          def embossFunction(self):
49              femboss=np.array([[-1.0, 0.0, 0.0],[ 0.0, 0.0, 0.0],[ 0.0, 0.0, 1.0]])
50
51              gray=cv.cvtColor(self.img,cv.COLOR_BGR2GRAY)
52              gray16=np.int16(gray)
53              self.emboss=np.uint8(np.clip(cv.filter2D(gray16,-1,femboss)+128,0,255))
54
55              cv.imshow('Emboss',self.emboss)
56
57          def cartoonFunction(self):
58              self.cartoon=cv.stylization(self.img,sigma_s=60,sigma_r=0.45)
59              cv.imshow('Cartoon',self.cartoon)
60
61          def sketchFunction(self):
62
    self.sketch_gray,self.sketch_color=cv.pencilSketch(self.img,sigma_s=60,sigma_r=0.07,shade_fac
    tor=0.02)
63              cv.imshow('Pencil sketch(gray)',self.sketch_gray)
64              cv.imshow('Pencil sketch(color)',self.sketch_color)
65
66          def oilFunction(self):
67              self.oil=cv.xphoto.oilPainting(self.img,10,1,cv.COLOR_BGR2Lab)
68              cv.imshow('Oil painting',self.oil)
69
70          def saveFunction(self):
71              fname=QFileDialog.getSaveFileName(self,'파일 저장','./')
72
```

```
73              i=self.pickCombo.currentIndex()
74              if i==0: cv.imwrite(fname[0],self.emboss)
75              elif i==1: cv.imwrite(fname[0],self.cartoon)
76              elif i==2: cv.imwrite(fname[0],self.sketch_gray)
77              elif i==3: cv.imwrite(fname[0],self.sketch_color)
78              elif i==4: cv.imwrite(fname[0],self.oil)
79
80          def quitFunction(self):
81              cv.destroyAllWindows()
82              self.close()
83
84  app=QApplication(sys.argv)
85  win=SpecialEffect()
86  win.show()
87  app.exec_()
```







6.6.3 비디오의 특수 효과

[프로그램 6-7]을 조금 수정하면 비디오 버전이 된다. 비디오 버전에서 신경 써야할 점은 웹캠을 통해 들어오는 초당 30 프레임의 영상을 실시간 처리하는 일에 있다. 한 장을 특수 효과 처리하는데 1/30초 이내 시간이 걸린다면 지연 없이 비디오를 디스플레이할 수 있다. 그렇지 않다면 지연이 발생한다.

[프로그램 6-8]의 실행 결과를 보고 사용자 인터페이스를 확인해보자. <비디오 시작> 버튼은 웹캠과 연결하여 윈도우에 비디오 영상을 디스플레이한다. 두 번째 메뉴는 콤보박스다. 여러 특수 효과 중에서 하나를 선택할 수 있다. <나가기> 버튼은 프로그램을 끝낸다.

프로그램의 전체 구조를 살펴보자. 6~53행은 VideoSpecialEffect 클래스를 선언한다. 7~22행은 생성자 함수 __init__이다. 24~53행은 2개 버튼에 해당하는 콜백 함수다. 55~58행은 프로그램의 메인에 해당한다.

7~22행의 __init__을 살펴본다. 9행은 윈도우의 제목 막대에 '비디오 특수 효과'라고 쓰고 10행은 윈도우 위치와 크기를 설정한다. 12~15행은 QPushButton 함수로 버튼 2개와 QComboBox 함수로 콤보박스 1개를 만든다. 17~19행은 버튼과 콤보박스의 위치와 크기를 지정한다. 21~22행은 버튼이 클릭됐을 때 수행할 콜백 함수를 등록한다.

2개 버튼에 해당하는 콜백 함수를 하나씩 살펴보자. 24~48행은 <비디오 특수 효과 켜기> 버튼이 클릭되면 실행되는 videoSpecialEffectFunction이다. 25행은 웹캠과 연결을 시도한다. 28~48행은 무한 루프를 돌면서, 실시간으로 비디오 프레임을 읽고 지정된 특수 효과를 적용한 결과를 윈도우에 디스플레이한다. 29행은 프레임을 읽는다. 32행은 콤보박스에서 사용자가 선택해 놓은 특수 효과의 번호를 알아낸다. 33~45행은 번호에 따라 0이면 엠보싱, 1이면 카툰, 2면 연필 스케치 명암, 3이면 연필 스케치 컬러, 4면 유화 효과를 적용하여 special_img 객체에 저장한다. 47행은

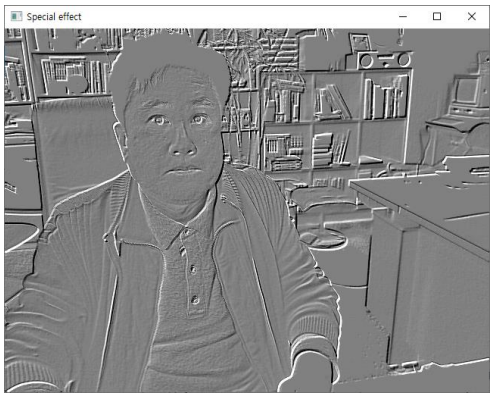
special_img 객체를 윈도우에 디스플레이한다. 50~53행은 <나가기> 버튼이 클릭되었을 때 실행되는 quitFunction이다. 웹캠과 연결을 끊고 윈도우를 모두 닫고 프로그램을 종료한다.

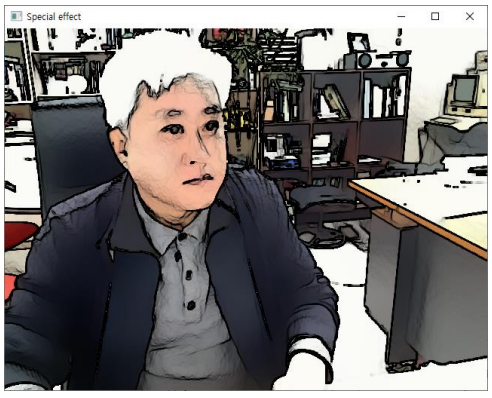
프로그램을 실행해 보면, 약간의 지연이 있지만 큰 무리없이 비디오 처리가 가능함을 알 수 있다.

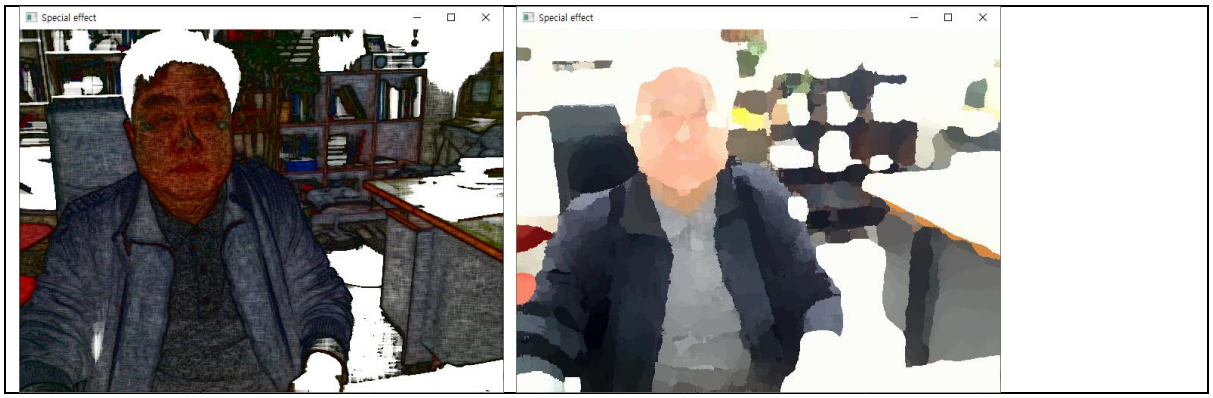
[프로그램 6-8] 비디오 특수 효과

```
1   import cv2 as cv
2   import numpy as np
3   from PyQt5.QtWidgets import *
4   import sys
5
6   class VideoSpecialEffect(QMainWindow):
7       def __init__(self):
8           super().__init__()
9           self.setWindowTitle('비디오 특수 효과')
10          self.setGeometry(200,200,400,100)
11
12          videoButton=QPushButton('비디오 특수 효과 켜기',self)
13          self.pickCombo=QComboBox(self)
14          self.pickCombo.addItems(['엠보싱','카툰','연필 스케치(명암)','연필 스케치(컬러)','유화'])
15          quitButton=QPushButton('나가기',self)
16
17          videoButton.setGeometry(10,10,140,30)
18          self.pickCombo.setGeometry(150,10,110,30)
19          quitButton.setGeometry(280,10,100,30)
20
21          videoButton.clicked.connect(self.videoSpecialEffectFunction)
22          quitButton.clicked.connect(self.quitFunction)
23
24      def videoSpecialEffectFunction(self):
25          self.cap=cv.VideoCapture(0,cv.CAP_DSHOW)
26          if not self.cap.isOpened(): sys.exit('카메라 연결 실패')
27
28          while True:
29              ret,frame=self.cap.read()
30              if not ret: break
31
32              pick_effect=self.pickCombo.currentIndex()
33              if pick_effect==0:
34                  femboss=np.array([[-1.0, 0.0, 0.0],[ 0.0, 0.0, 0.0],[ 0.0, 0.0, 1.0]])
```

```
35                      gray=cv.cvtColor(frame,cv.COLOR_BGR2GRAY)
36                      gray16=np.int16(gray)
37                      special_img=np.uint8(np.clip(cv.filter2D(gray16,-1,femboss)+128,0,255))
38                  elif pick_effect==1:
39                      special_img=cv.stylization(frame,sigma_s=60,sigma_r=0.45)
40                  elif pick_effect==2:
41                      special_img,_=cv.pencilSketch(frame,sigma_s=60,sigma_r=0.07,shade_factor=0.02)
42                  elif pick_effect==3:
43                      _,special_img=cv.pencilSketch(frame,sigma_s=60,sigma_r=0.07,shade_factor=0.02)
44                  elif pick_effect==4:
45                      special_img=cv.xphoto.oilPainting(frame,10,1,cv.COLOR_BGR2Lab)
46
47                  cv.imshow('Special effect',special_img)
48                  cv.waitKey(1)
49
50          def quitFunction(self):
51              self.cap.release()
52              cv.destroyAllWindows()
53              self.close()
54
55  app=QApplication(sys.argv)
56  win=VideoSpecialEffect()
57  win.show()
58  app.exec_()
```






Special effect
Special effect