동영상 처리

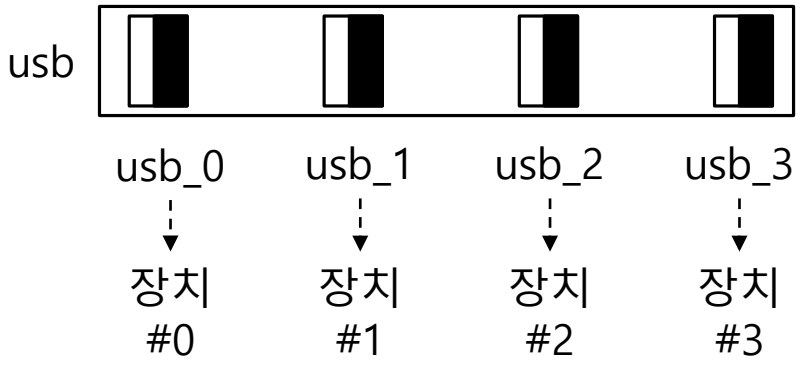
5.1 웹캠 입출력

카메라 입력

- 카메라가 스트리밍 형태로 동작할 때 사용 가능
- 카메라로부터 데이터를 실시간으로 획득

카메라 장치 식별

- 연결된 카메라의 장치 번호 사용 (0, 1, 2, ...)
- 플랫폼에서 카메라에 대한 접근 권한이 허용



일반적인 장치 사용 순서

- 장치 획득(접근 승인)
- 장치 접근(촬영, 획득)
- 장치 해제(반납)

5.1 웹캠 입출력

웹캠 입력 코드

- OpenCV의 기본적인 웹캠 처리 코드를 작성하여 실습

```
import cv2

capture = cv2.VideoCapture(0)

width = capture.get(cv2.CAP_PROP_FRAME_WIDTH)
height = capture.get(cv2.CAP_PROP_FRAME_HEIGHT)

print('Frame width; {}, height {}'.format(width, height))

capture.set(cv2.CAP_PROP_FRAME_WIDTH, 1024)
capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 768)

while cv2.waitKey(32) < 0:
    ret, frame = capture.read()
    if not ret: # ret == False:
        break

    cv2.imshow("Frame", frame)
capture.release()
```

5.1 웹캠 입출력

코드 설명

- 0번 카메라 장치 접근 (VideoCapture)
- 프레임 속성 획득하여 출력
- 프레임 속성 (가로, 세로 해상도) 지정

- 프레임 획득(read)하고 실패하지 않으면 화면 출력(imshow)
- 32 msec 단위로 상기 과정을 반복 (약 1초에 31 프레임)

- 사용자가 임의의 키를 누르면 반복 탈출
- 카메라 장치 반납(release)하고 프로그램 종료

5.1 웹캠 입출력

웹캠 입력 객체 : VideoCapture

- 카메라 장치 객체 생성: VideoCapture() 함수
  - VideoCapture 클래스를 통해서 내장 카메라 또는 외장 카메라에 연결

※ 동영상 접근을 위해서는 VideoCapture 클래스를 활용

클래스명	cv2.VideoCapture(index)
매개변수	- index (int) 카메라 장치 번호 (일반적으로 노트북의 경우 내장 카메라가 0번, 외장카메라가 1번부터 순차적으로 번호가 할당) ※ 내장카메라: 노트북의 디스플레이 중앙 상단에 있는 카메라와 같이 컴퓨팅 장비에 내장되어 있는 카메라 ※ 외장카메라: 웹캠과 같이 외부 장치를 컴퓨팅 장비에 연결하여 사용하는 카메라
리턴값	VideoCapture 객체(cv2.VideoCapture)

- 카메라 장치에 대한 제어 반환: release()함수
  - release()함수를 통하여 장치에 대한 제어 권한을 반환

함수명	VideoCapture.release()
매개변수	없음
리턴값	없음

5.1 웹캠 입출력

카메라 속성 정보 획득 함수: get() 함수

- get() 함수를 통하여 장치에 대한 속성 정보를 획득

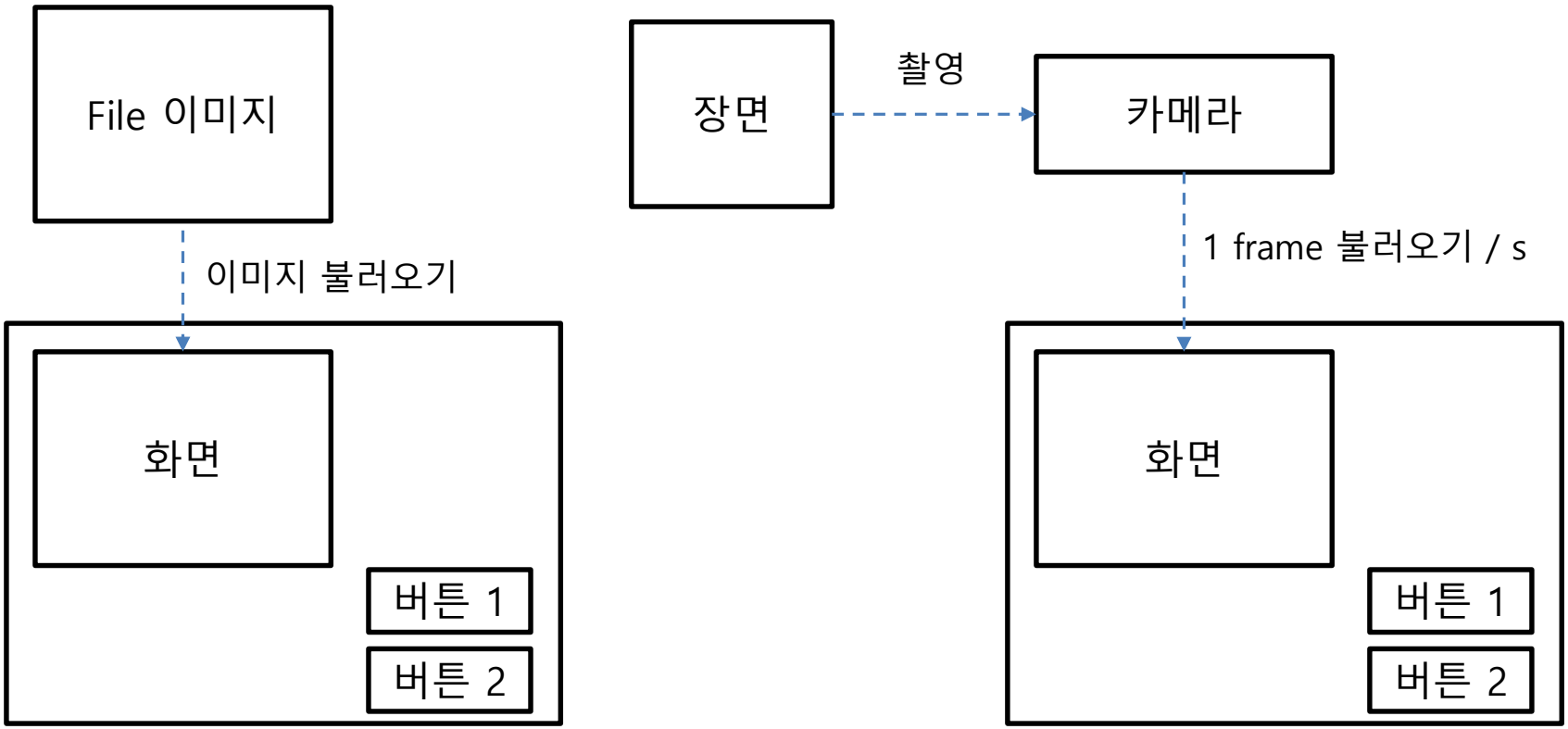
함수명	VideoCapture.get(propId)
매개변수	- propId 획득하고자 하는 속성 정보에 대한 id
리턴값	속성 정보값

propid	- cv2.CAP_PROP_FRAME_WIDTH : 프레임 너비 - cv2.CAP_PROP_FRAME_HEIGHT : 프레임 높이 - cv2.CAP_PROP_FPS : 프레임 속도 (초당 프레임 수) ※다음 prodId 옵션들도 지정 가능 - cv2.CAP_PROP_BRIGHTNESS : 프레임 밝기 (지원되는 경우) - cv2.CAP_PROP_CONTRAST : 프레임 대비값 - cv2.CAP_PROP_SATURATION : 프레임 채도값 - cv2.CAP_PROP_HUE : 프레임 색상값 - cv2.CAP_PROP_GAIN : 프레임 이득값 (지원되는 경우) - cv2.CAP_PROP_EXPOSURE : 프레임 노출값 (지원되는 경우)

5.1 웹캠 입출력

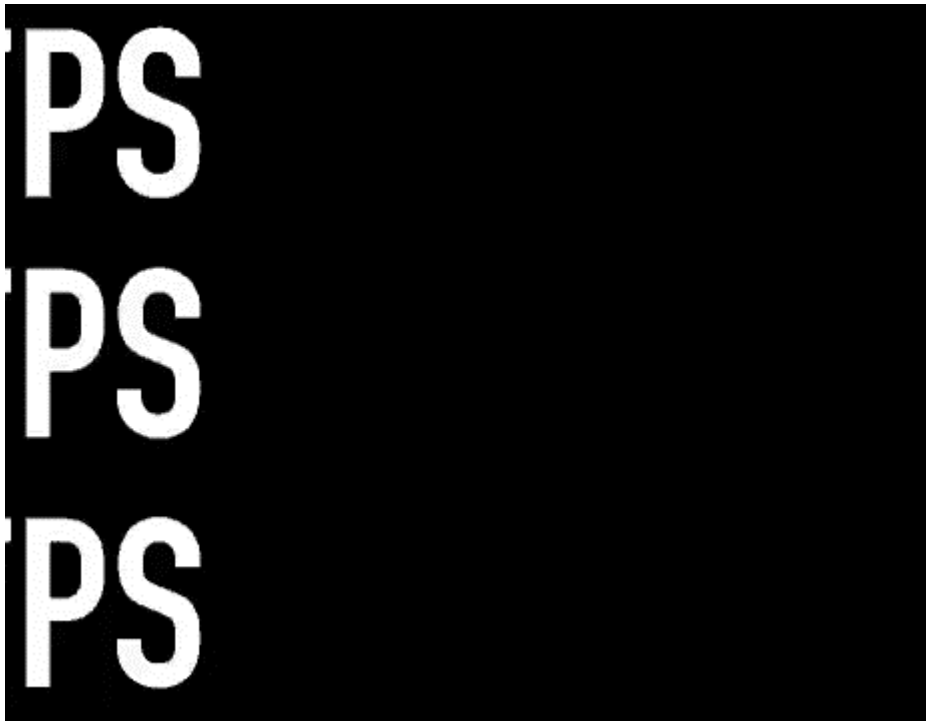
영상 출력 형태

- 단일 이미지는 영상을 불러온 후 화면에 출력
- 카메라를 통한 실시간 화면은 일정 시간 간격으로 카메라에서 입력 받은 영상 frame을 불러온 후 화면에 출력
- 동영상 파일은 카메라와 유사한 방식으로 일정 시간 간격으로 동영상의 frame을 순차적으로 불러온 후 화면에 출력



5.1 웹캠 입출력

- Frame rate
  - 디스플레이 장치가 화면 하나의 데이터를 표시하는 속도를 나타내며, 동영상이 부드럽게 재생되는지를 결정하는 지표
  - FPS(frames per second)는 1초 동안 보여주는 화면의 수
  - 애니메이션 제작에는 15~18fps의 동영상을 만들며, 제작 환경에 따라 그 이상도 제작
  - 일반적으로 눈의 잔상을 이용한 자연스러운 화면에는 1초에 30번 이상의 Frame 출력이 필요



5.1 웹캠 입출력

- 영상의 처리시간과 FPS 관계
  - 한 프레임의 영상처리에 소요되는 연산 시간 T와 FPS는 서로 반비례 관계를 가진다
  - 연산 시간이 길어지면, 화면에 출력하기 위한 프레임 처리가 느려 지기 때문에 FPS가 감소

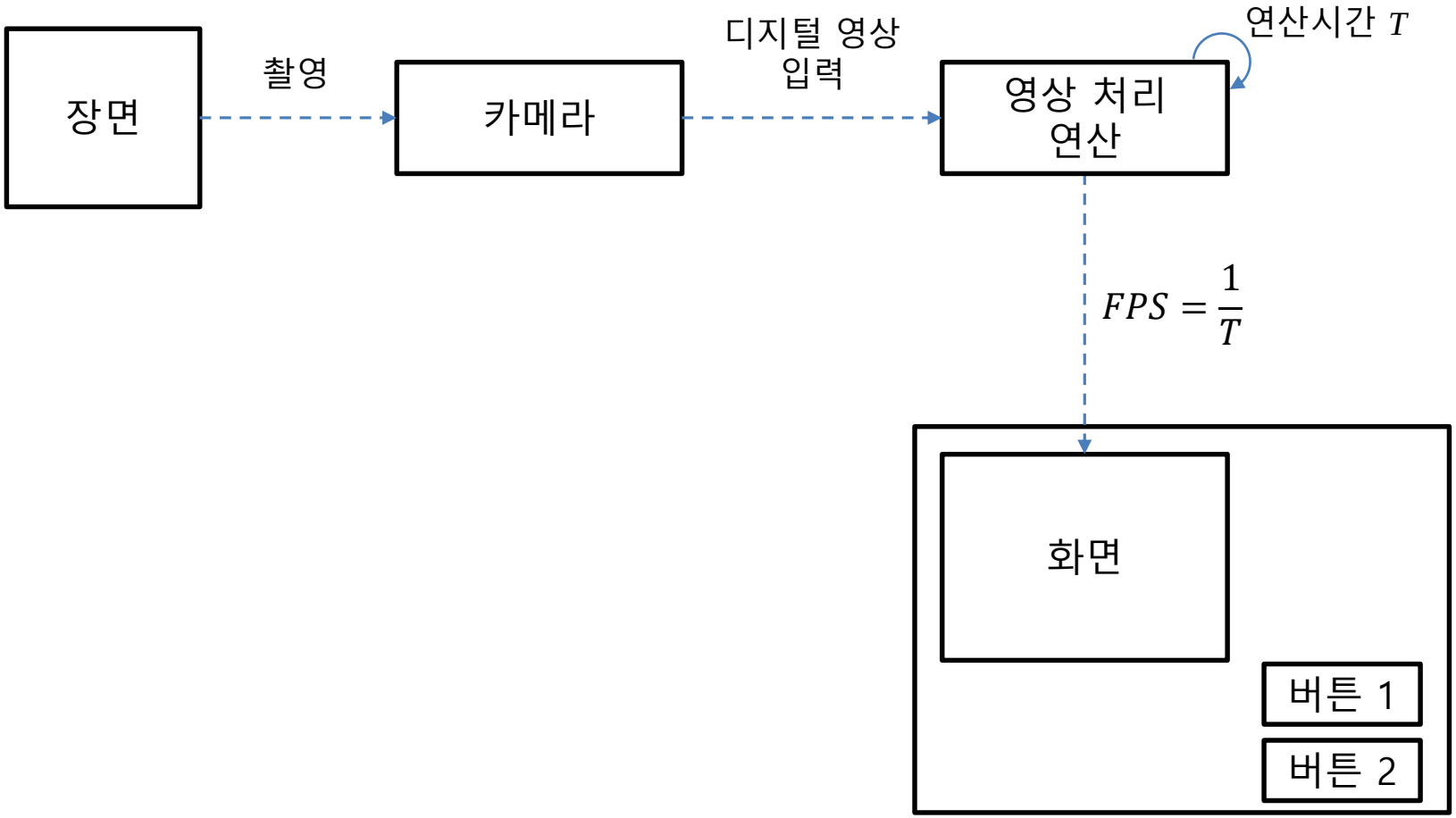
$$FPS = \frac{1}{T}$$

  - 예) 한 프레임의 처리 시간이 0.033초(33millisecond)일 경우, 1초 동안 보여지는 프레임의 수는 약 30 frame 이므로, 30 FPS로 동작하게 된다

$$FPS = \frac{1}{0.033} \approx 30.3$$

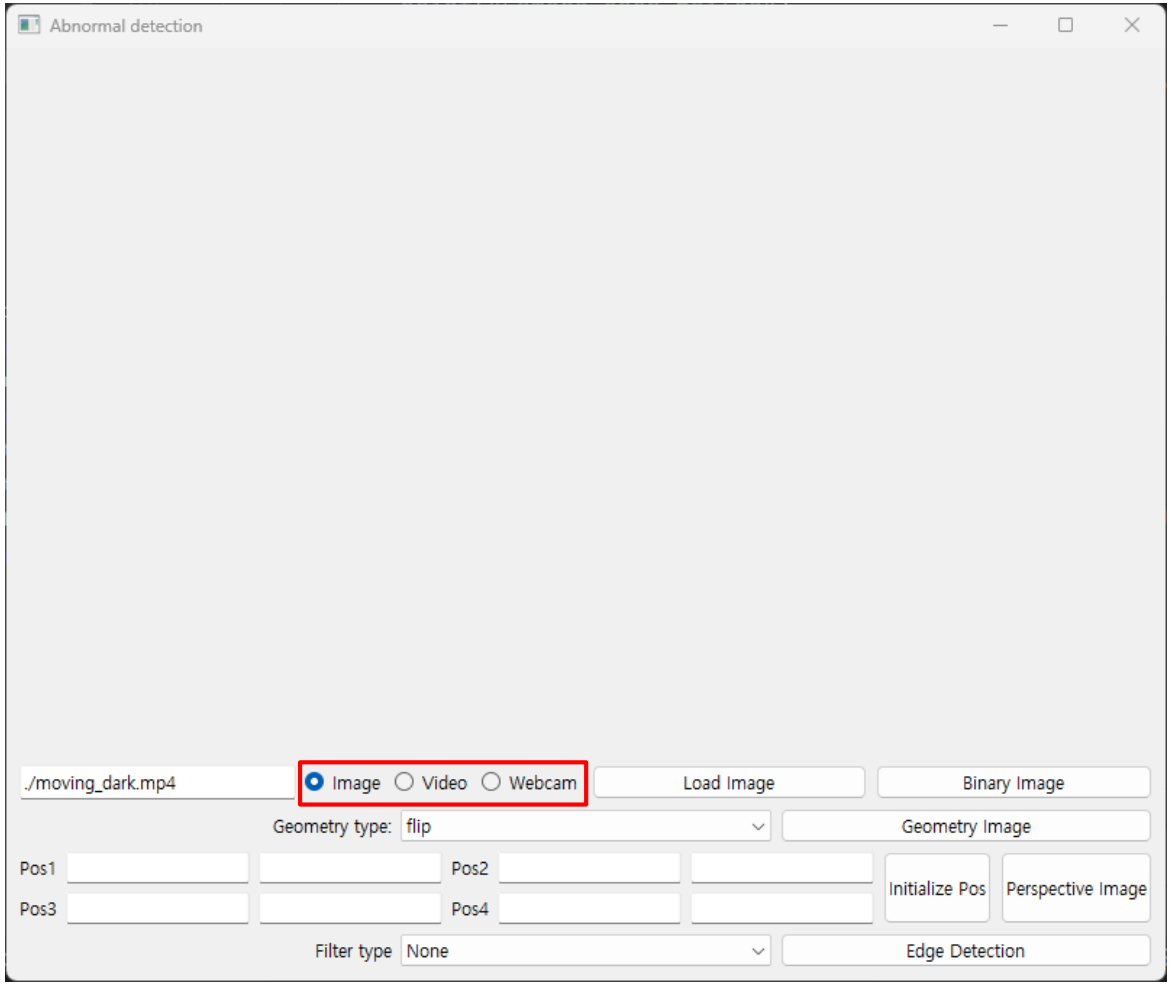
5.1 웹캠 입출력

- 영상의 처리시간과 FPS 관계



5.1 웹캠 입출력

- QRadioButton을 이용하여 Image/Video/Webcam을 선택적으로 불러오기



5.1 웹캠 입출력

QRadioButton 객체 선언

- QRadioButton 클래스의 setChecked(True) 매서드는 초기 radiobutton의 선택 유무를 지정할 수 있음

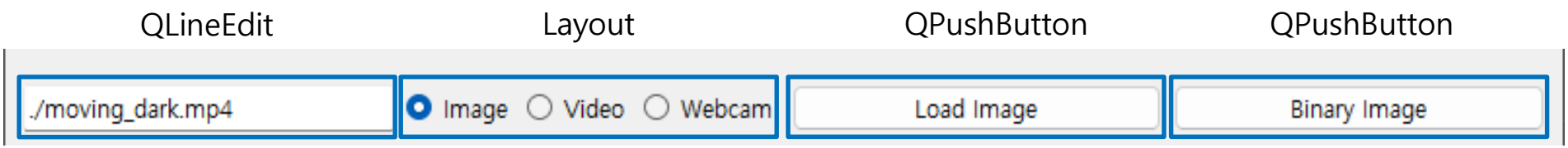
```
from PySide6.QtWidgets import (QMainWindow, QRadioButton)
class Window(QMainWindow):
    def __init__(self):
        super().__init__()
        self.radiobutton_1 = QRadioButton("Image")
        self.radiobutton_2 = QRadioButton("Video")
        self.radiobutton_3 = QRadioButton("Webcam")
        self.radiobutton_1.setChecked(True)
```

Radiobutton 수평배치

```
layout_loading_type = QHBoxLayout()
layout_loading_type.addWidget(self.radiobutton_1)
layout_loading_type.addWidget(self.radiobutton_2)
layout_loading_type.addWidget(self.radiobutton_3)
```

5.1 웹캠 입출력

- 수평배치를 완성해 보세요



5.1 웹캠 입출력

QTimer 객체 선언

- QTimer.timeout은 지정된 시간에 도달하면 timeout 이벤트가 발생되며, connect 매서드를 통하여 슬롯을 timeout 이벤트와 연결 시키면 timeout이 발생될 때마다 슬롯이 실행

```
from PySide6.QtCore import QTimer

class Window(QMainWindow):
    def __init__(self):
        super().__init__()

        self.timer = QTimer(self)
        self.timer.timeout.connect(self.display_video_stream)
```

Timer 사용 방법

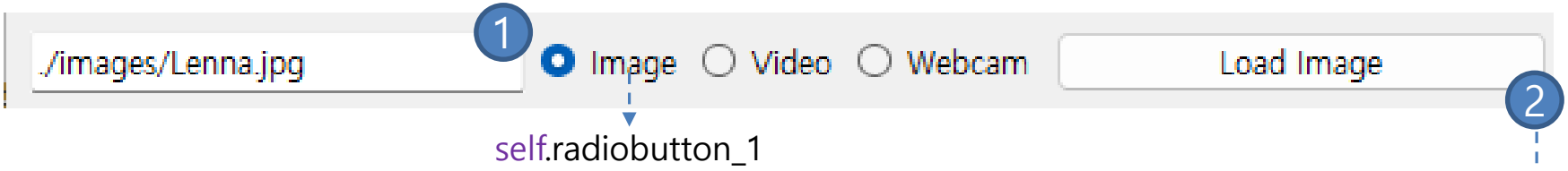
- .start(ms) 함수는 타이머를 시작시키는 함수로, 밀리초 단위의 시간이 경과하면 timeout 이벤트가 발생함

```
self.timer.start(30)
```

----→ start 인자의 값이 30이므로, 30 milliscecond마다 timeout 이벤트가 발생

5.2 Image 불러오기

- Load 버튼의 동작 (※ radiobutton의 `**Image**'가 선택된 경우)

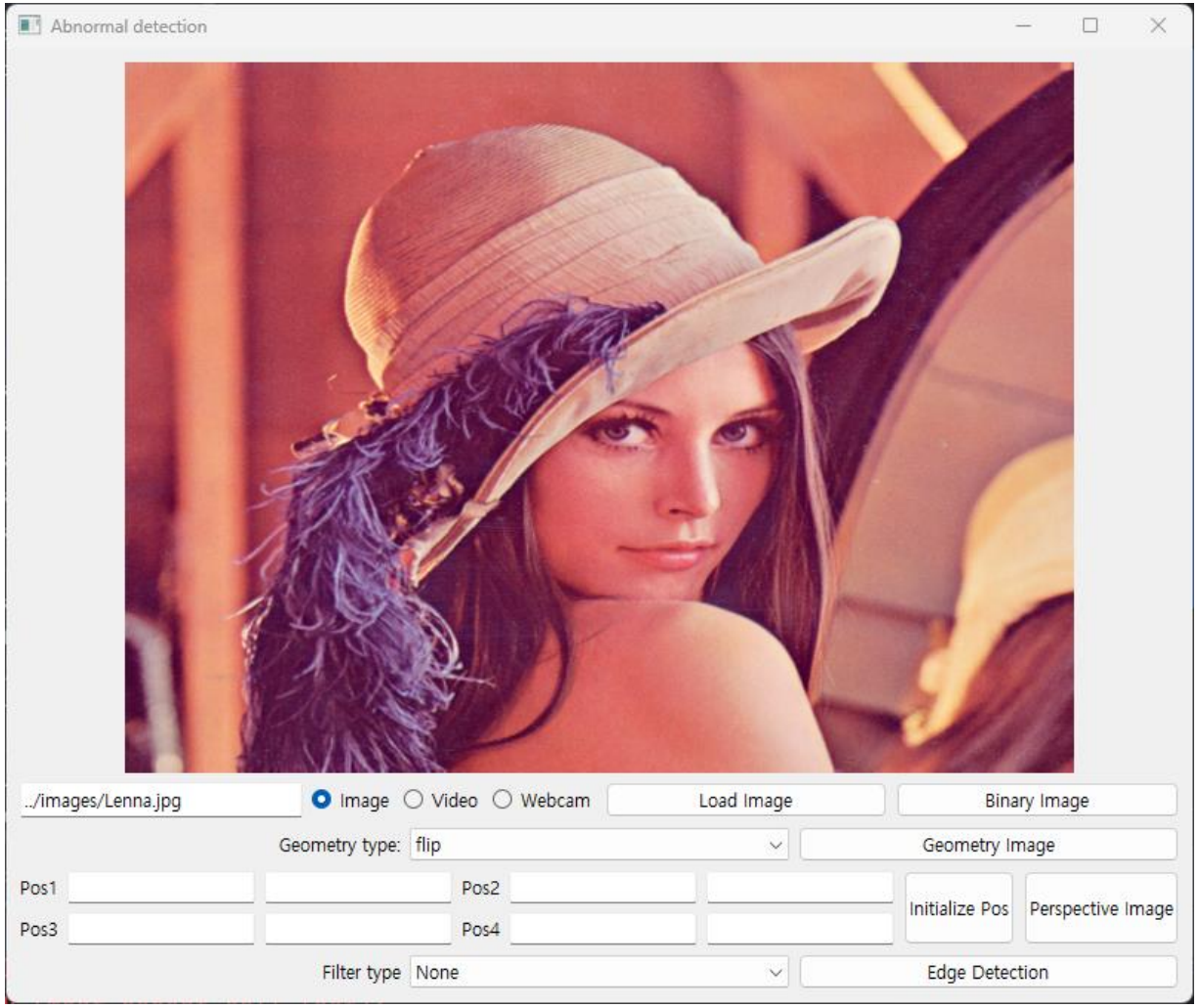


  - 1) 파일경로 입력 및 Image 라디오 버튼 선택
  - 2) Load Image 버튼 클릭
    - 2-1) 버튼 클릭 이벤트로 인하여 load_img_func 함수(슬롯) 자동실행

```
def load_img_func(self):
    if self.radiobutton_1.isChecked() is True:
        self.MODE_VIDEO = False
        self.m_main_img = cv2.imread(f"{self.edit.text()}", cv2.IMREAD_COLOR)
        self.m_main_img = cv2.resize(self.m_main_img, (640, 480), interpolation=cv2.INTER_CUBIC)
        self.m_proc_img = self.m_main_img.copy()
        self.update_image(self.m_proc_img)
        print("update image")
    elif self.radiobutton_2.isChecked() is True:
        self.MODE_VIDEO = True
        self.setup_camera(f"{self.edit.text()}")
    elif self.radiobutton_3.isChecked() is True:
        self.MODE_VIDEO = True
        self.setup_camera(0)
```

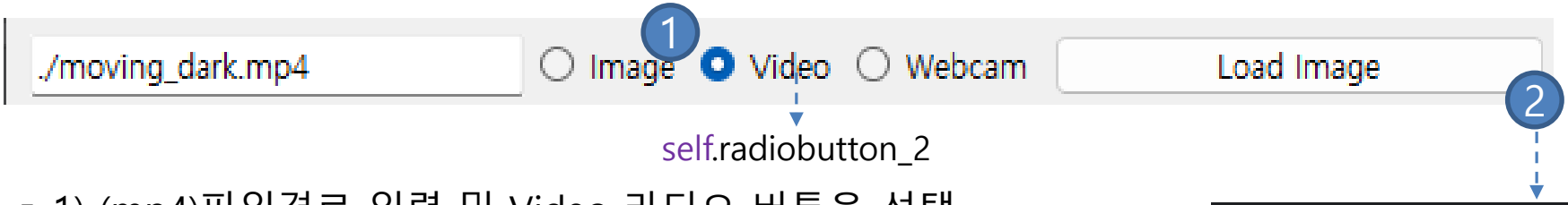
5.2 Image 불러오기

- Load 버튼의 동작 (※ radiobutton의 `**Image**'가 선택된 경우) **결과화면**



5.3 Video 또는 Webcam 불러오기

- Load 버튼의 동작 (※ radiobutton의 `**Video**'가 선택된 경우)

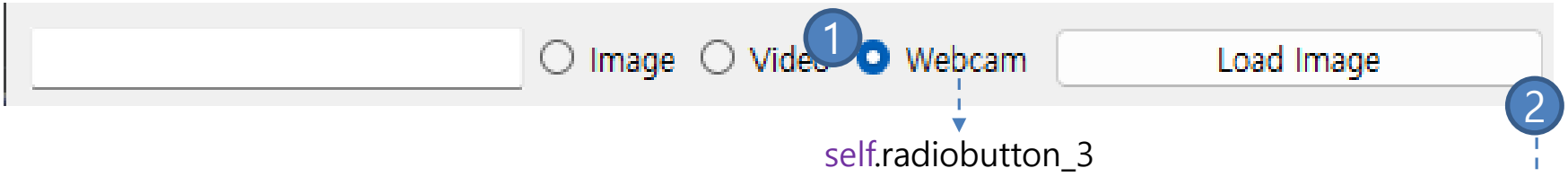


  - 1) (mp4)파일경로 입력 및 Video 라디오 버튼을 선택
  - 2) Load Image 버튼 클릭
    - 2-1) 버튼 클릭 이벤트로 인하여 load_img_func 함수(슬롯) 자동실행

```
def load_img_func(self):
    if self.radiobutton_1.isChecked() is True:
        self.MODE_VIDEO = False
        self.m_main_img = cv2.imread(f"{self.edit.text()}", cv2.IMREAD_COLOR)
        self.m_main_img = cv2.resize(self.m_main_img, (640, 480), interpolation=cv2.INTER_CUBIC)
        self.m_proc_img = self.m_main_img.copy()
        self.update_image(self.m_proc_img)
        print("update image")
    elif self.radiobutton_2.isChecked() is True:
        self.MODE_VIDEO = True
        self.setup_camera(f"{self.edit.text()}")
    elif self.radiobutton_3.isChecked() is True:
        self.MODE_VIDEO = True
        self.setup_camera(0)
```

5.3 Video 또는 Webcam 불러오기

- Load 버튼의 동작 (※ radiobutton의 `**Webcam**'이 선택된 경우)



  - 1) Webcam 라디오 버튼을 선택
  - 2) Load Image 버튼 클릭
    - 2-1) 버튼 클릭 이벤트로 인하여 load_img_func 함수(슬롯) 자동실행

```
def load_img_func(self):
    if self.radiobutton_1.isChecked() is True:
        self.MODE_VIDEO = False
        self.m_main_img = cv2.imread(f"{self.edit.text()}", cv2.IMREAD_COLOR)
        self.m_main_img = cv2.resize(self.m_main_img, (640, 480), interpolation=cv2.INTER_CUBIC)
        self.m_proc_img = self.m_main_img.copy()
        self.update_image(self.m_proc_img)
        print("update image")
    elif self.radiobutton_2.isChecked() is True:
        self.MODE_VIDEO = True
        self.setup_camera(f"{self.edit.text()}")
    elif self.radiobutton_3.isChecked() is True:
        self.MODE_VIDEO = True
        self.setup_camera(0)
```

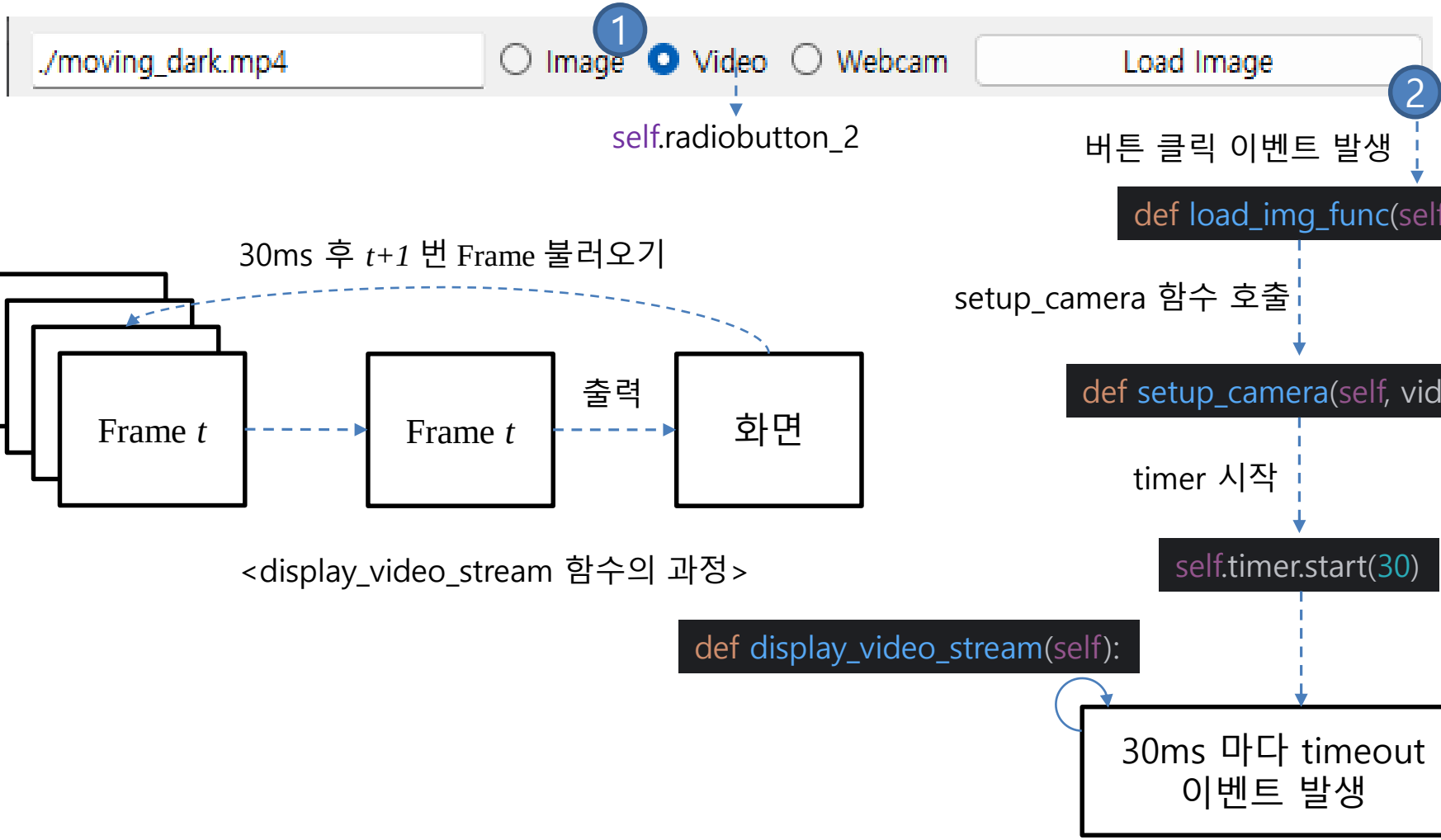
5.3 Video 또는 Webcam 불러오기

- Load 버튼의 동작 (※ radiobutton의 `**Video**'가 선택된 경우)
  - self.setup_camera(vid)는 동영상 또는 웹캠 장치로부터 영상을 불러올 수 있도록 cv2.VideoCapture 클래스의 객체 생성과 QTimer를 실행 시키는 함수

```
def setup_camera(self, vid):
    self.capture = cv2.VideoCapture(vid)
    if not self.capture.isOpened():
        print("Camera open failed")
    self.capture.set(cv2.CAP_PROP_FRAME_WIDTH, self.img_size.width())
    self.capture.set(cv2.CAP_PROP_FRAME_HEIGHT, self.img_size.height())

    self.timer.start(30)
```

5.3 Video 또는 Webcam 불러오기

- Load 버튼의 동작 (※ radiobutton의 `**Video**'가 선택된 경우)



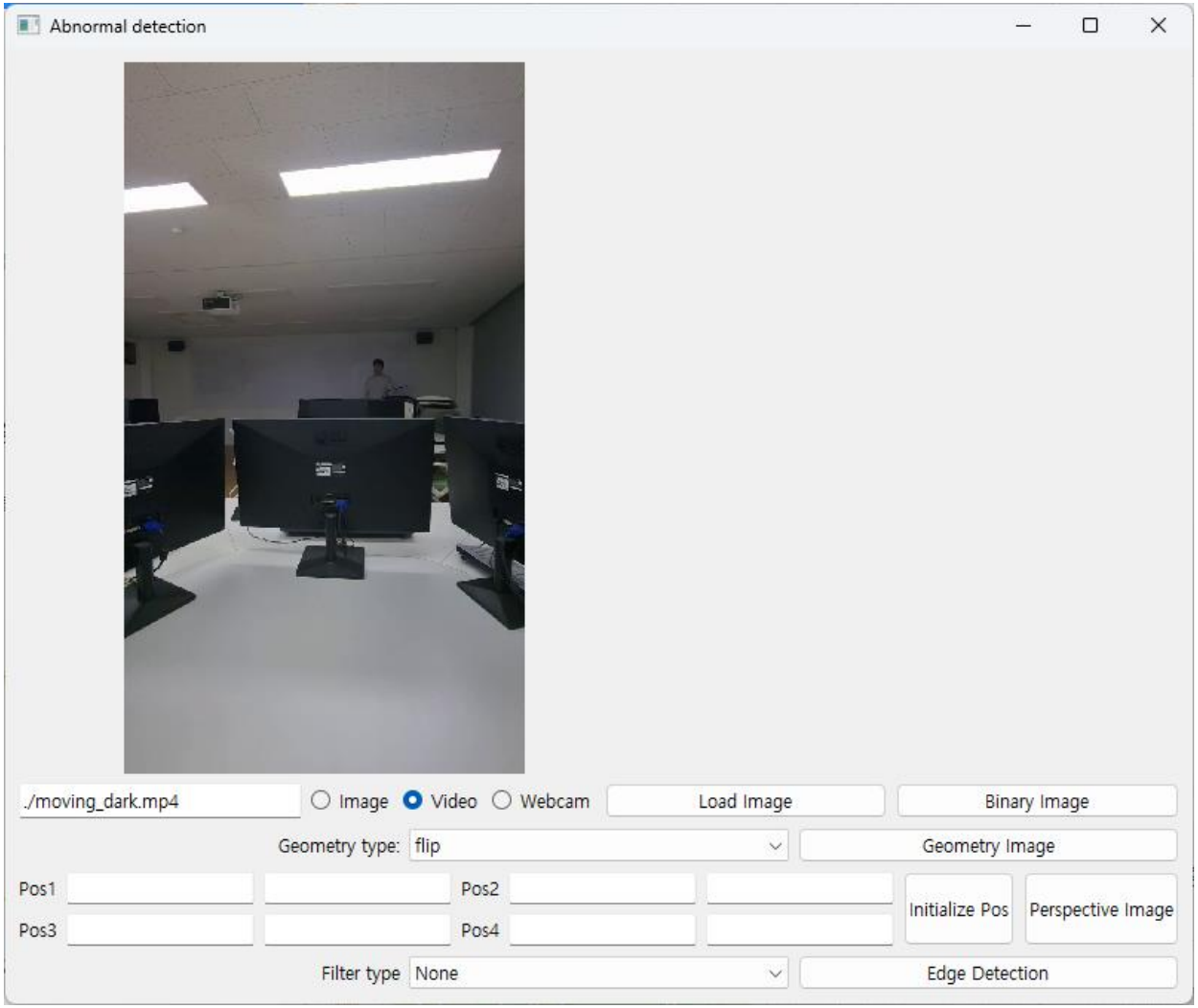
5.3 Video 또는 Webcam 불러오기

- Load 버튼의 동작 (※ radiobutton의 `**Video**'가 선택된 경우)
  - display_video_stream 함수는 읽어온 frame을 그대로 출력하거나, 영상처리 작업을 수행한 후 출력하는 함수

```
def display_video_stream(self):
    retval, self.m_proc_img = self.capture.read()
    if not retval:  # 새로운 프레임을 못받아 왔을 때 braek
        return

    self.update_image(self.m_proc_img)

    if self.MODE_VIDEO is False:
        self.timer.stop()
```

5.3 Video 또는 Webcam 불러오기

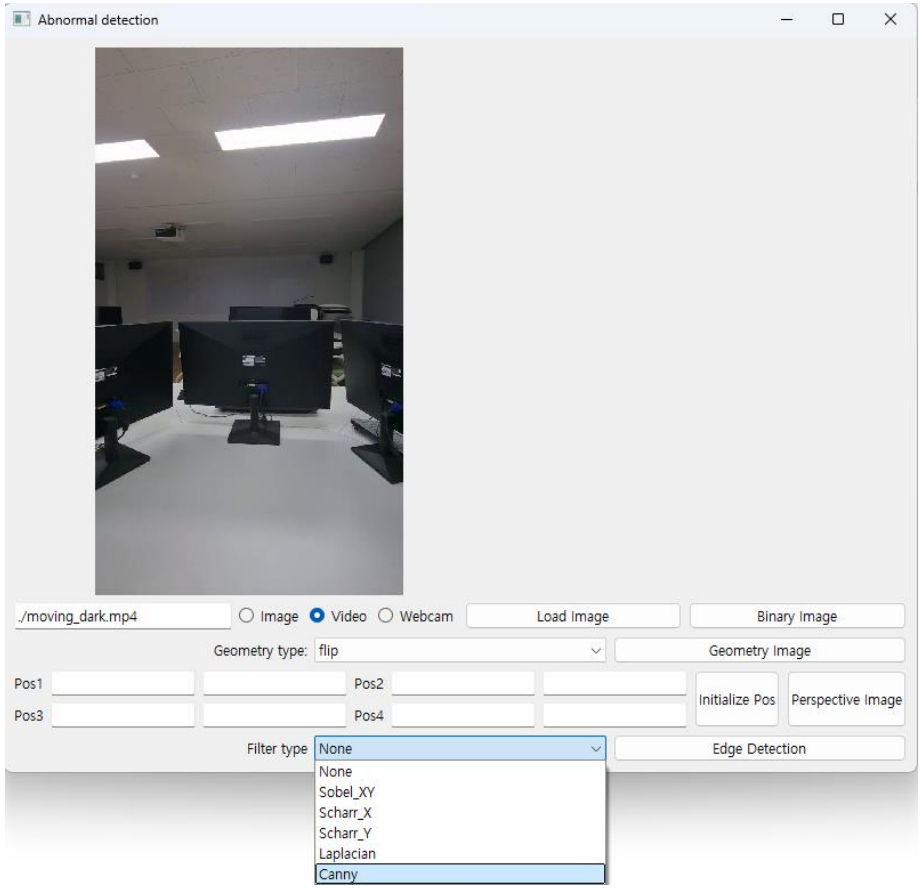
- Load 버튼의 동작 (※ radiobutton의 `**Video**'가 선택된 경우) **결과화면**



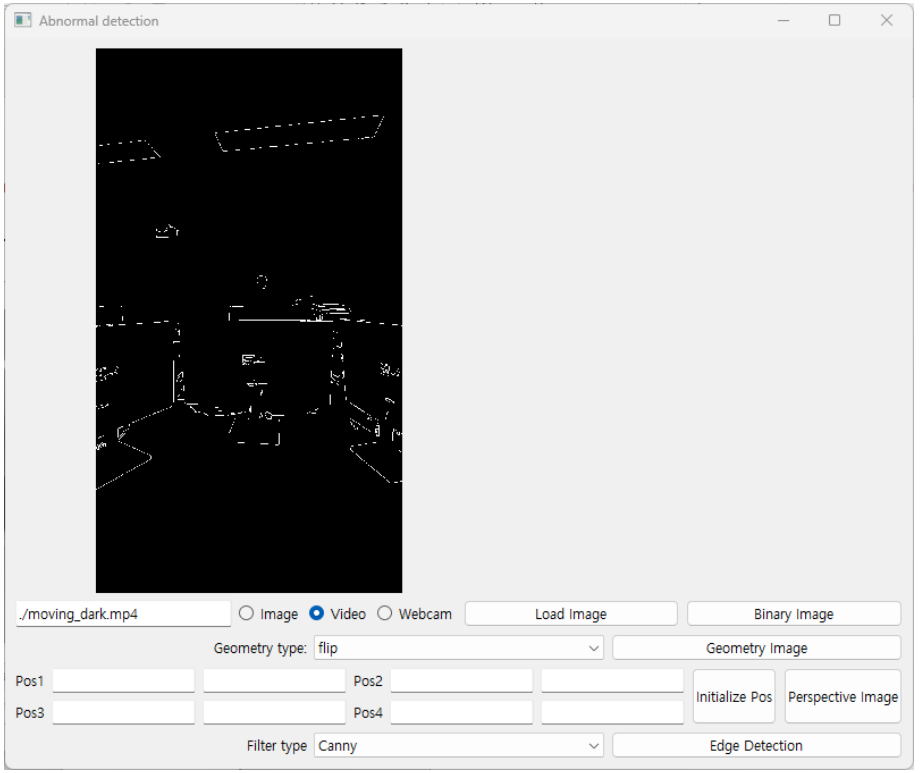
5.4 실시간 동영상 처리

- 동영상을 영상 처리 알고리즘이 적용된 결과가 출력되도록 하기

<원본 동영상 출력>



<edge detection이 적용된 동영상 출력>



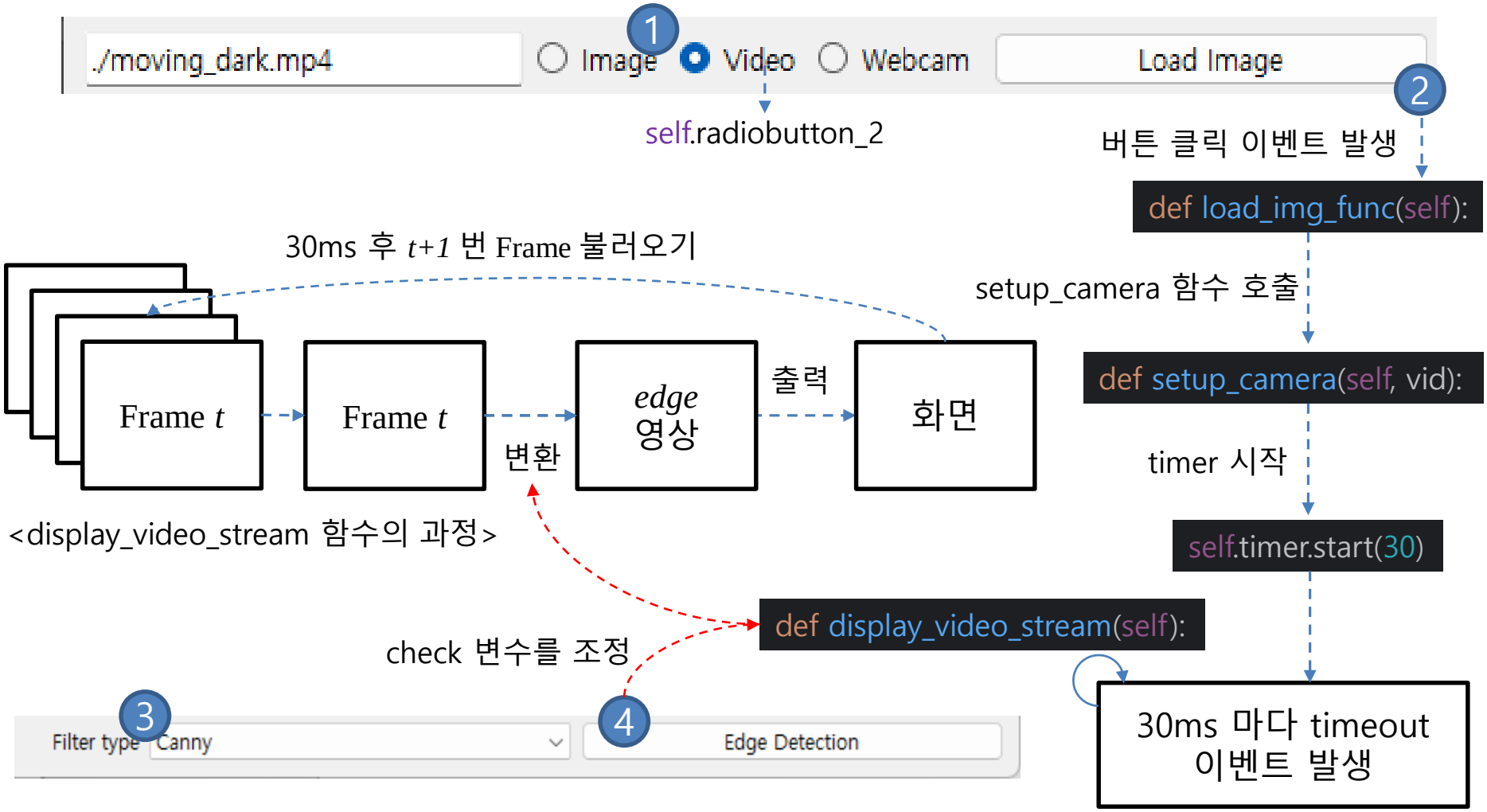
5.4 실시간 동영상 처리

- 기존 이미지 처리 과정

- 1) edge 방법 선택
- 2) Edge Detection 버튼을 클릭
- 3) Edge Detection 버튼의 슬롯 함수 실행
- 4) 영상변환 후 화면 출력

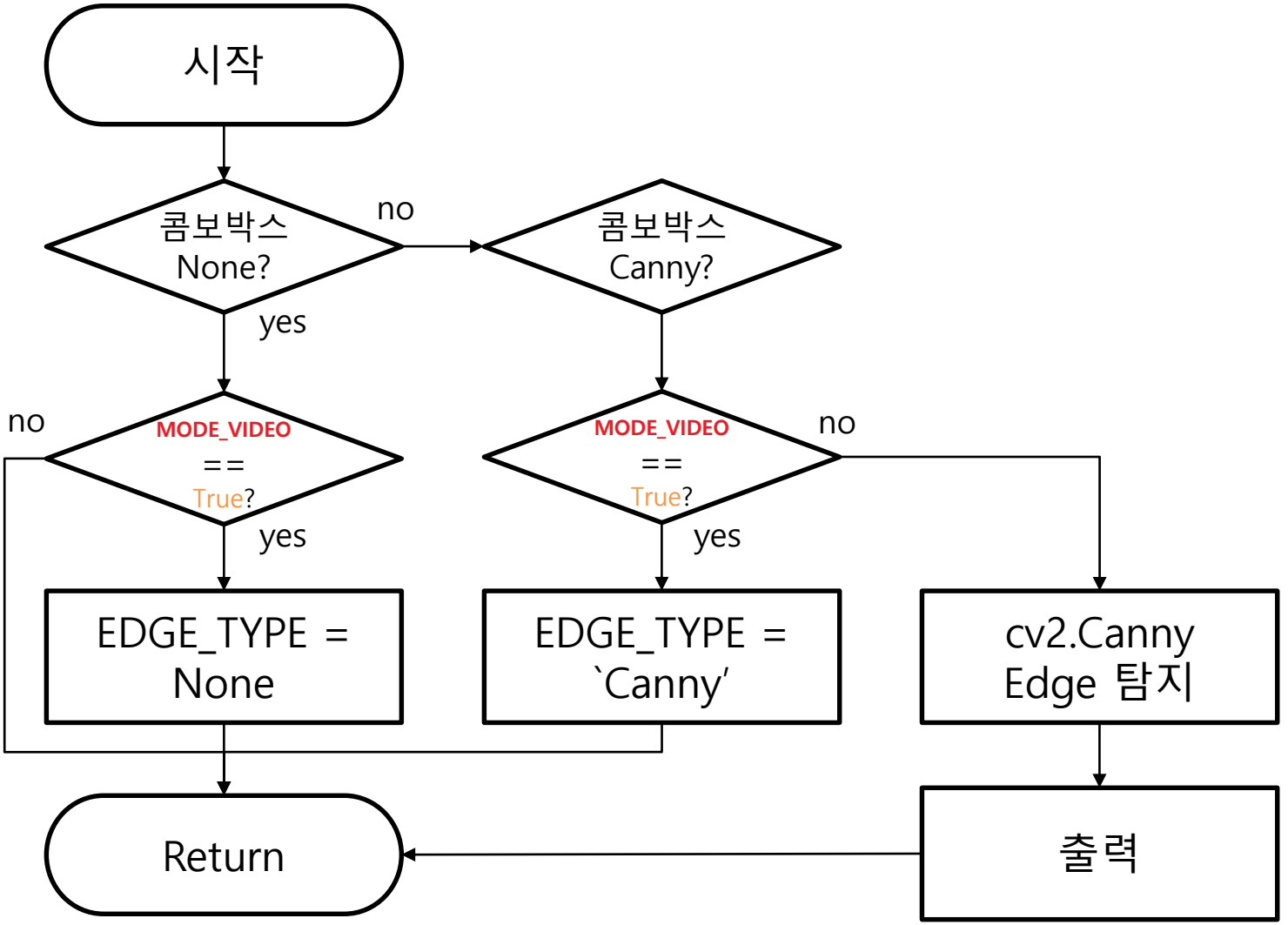
5.4 실시간 동영상 처리

- 동영상 처리 과정



5.4 실시간 동영상 처리

- 동영상 처리를 위한 Edge 탐지버튼의 순서도



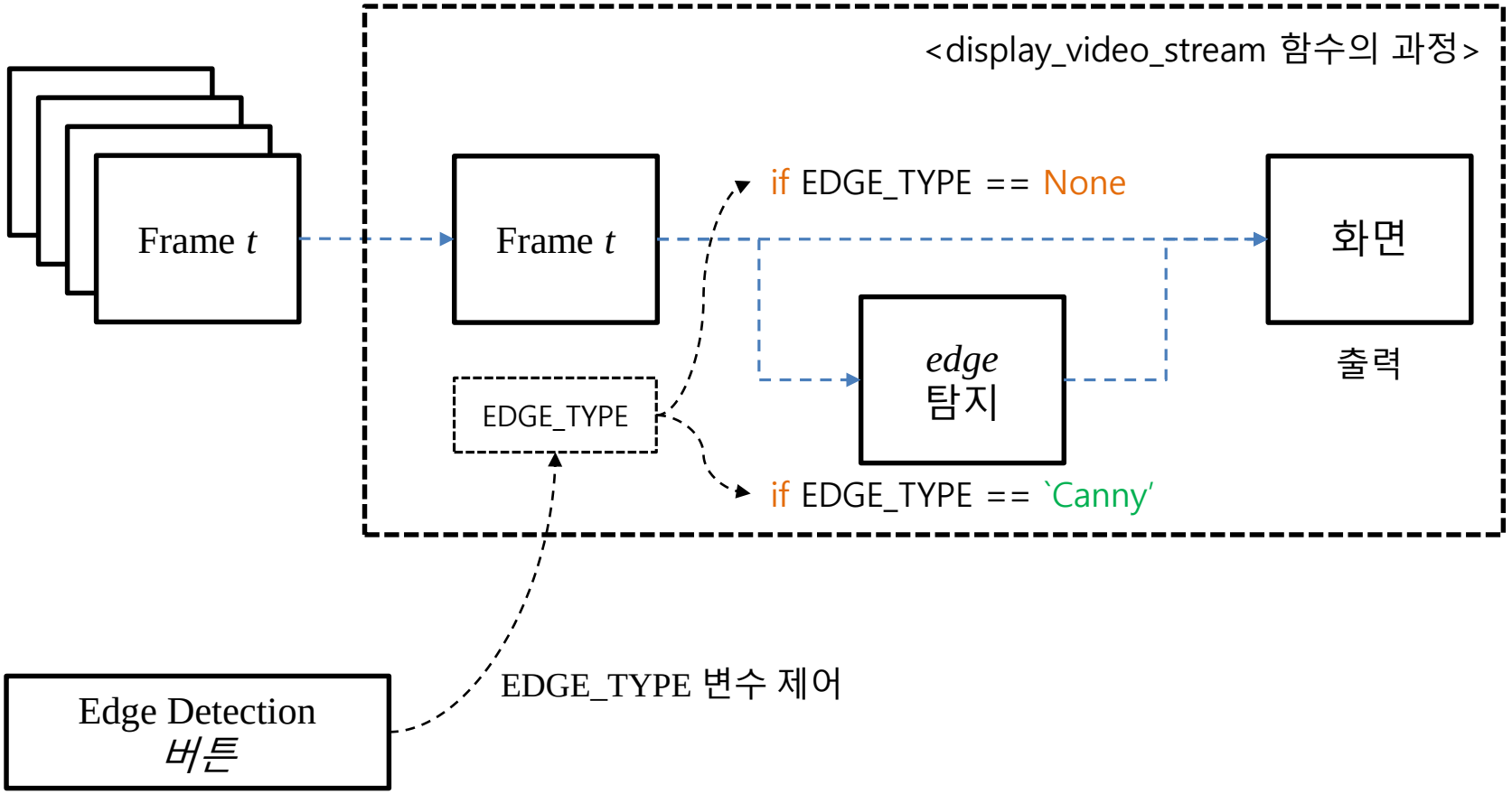
5.4 실시간 동영상 처리

- 동영상 처리를 위한 Edge 탐지버튼의 구현

```
def method_edge_detection(self):
    if self.m_proc_img is not None:
        if len(self.m_proc_img.shape) >= 3:
            self.m_proc_img = cv2.cvtColor(self.m_proc_img,cv2.COLOR_BGR2GRAY)

        if self._edgeType_combo_box.currentText() == 'None':
            if self.MODE_VIDEO is True:
                self.EDGE_TYPE = None
            return
        elif self._edgeType_combo_box.currentText() == 'Sobel_XY':
            edge_img = cv2.Sobel(self.m_proc_img, cv2.CV_8U, 1, 1, ksize=3)
        elif self._edgeType_combo_box.currentText() == 'Laplacian':
            edge_img = cv2.Laplacian(self.m_proc_img, cv2.CV_8U, ksize=3)
        elif self._edgeType_combo_box.currentText() == 'Canny':
            if self.MODE_VIDEO is True:
                self.EDGE_TYPE = 'Canny'
                return
            edge_img = cv2.Canny(self.m_proc_img, 150, 300)
        self.update_image(edge_img)
```

5.4 실시간 동영상 처리

- 동영상 처리와 Edge 탐지버튼의 관계도



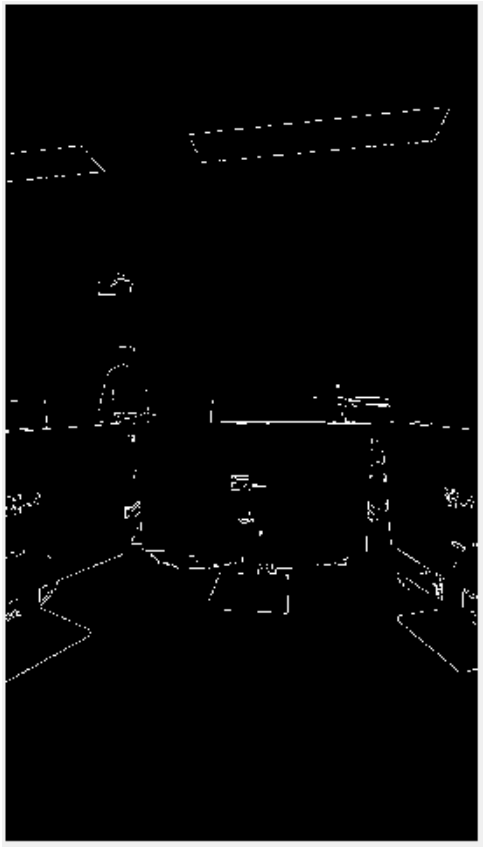
5.4 실시간 동영상 처리

- 동영상 처리와 Edge 탐지버튼의 관계도

```
def display_video_stream(self):
    retval, self.m_proc_img = self.capture.read()
    if not retval:  # 새로운 프레임을 못받아 왔을 때 braek
        return

    if self.EDGE_TYPE == 'Canny':
        self.m_proc_img = cv2.Canny(self.m_proc_img, 150, 300)
    elif self.EDGE_TYPE == 'Laplacian':
        self.m_proc_img = cv2.Laplacian(self.m_proc_img, cv2.CV_8U, ksize=3)
    self.update_image(self.m_proc_img)

    if self.MODE_VIDEO == False:
        self.timer.stop()
```

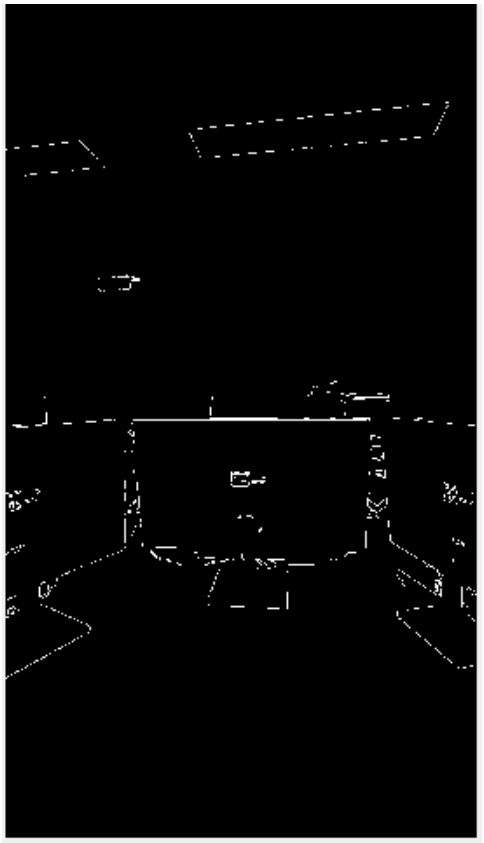
5.5 Edge와 실시간 matplot 그래프

- Edge 정보로 정상/비정상을 구분할 수 없을까?



정상(0)

“강단에 사람 edge가 포함된 경우”

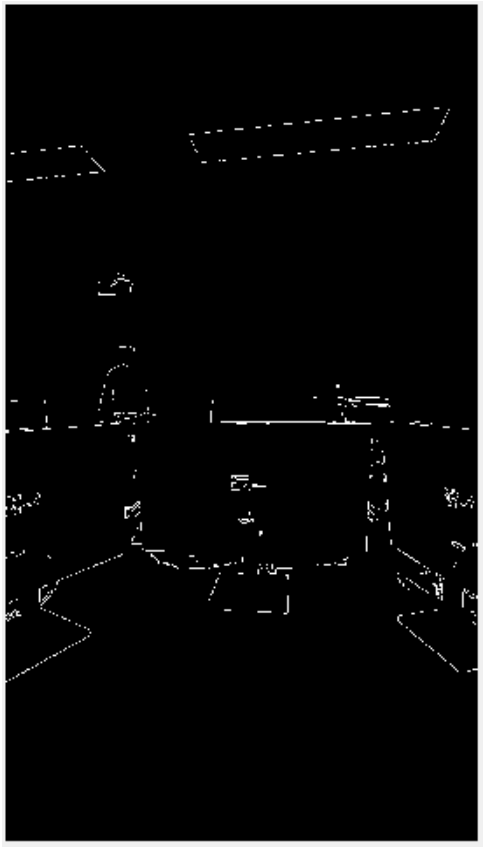


비정상(1)

“강단에 사람 edge가 발견되지 않는 경우”

5.5 Edge와 실시간 matplot 그래프

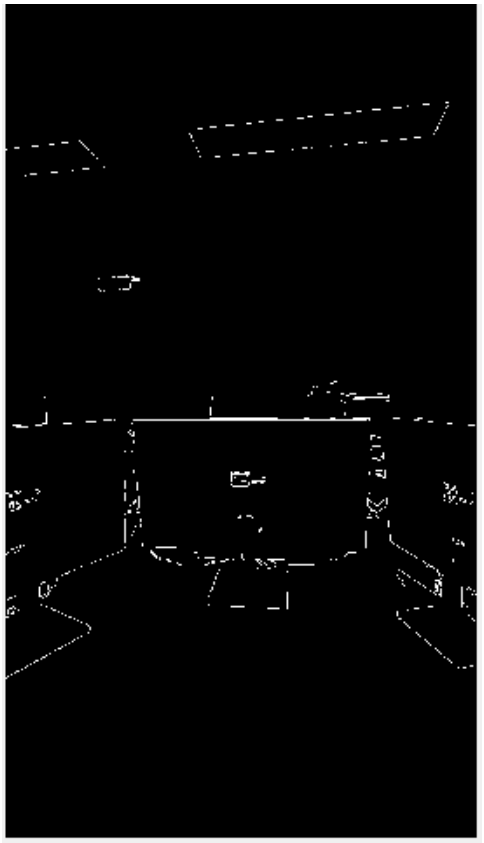
- Edge 정보로 정상(Normal)/비정상(Abnormal)을 구분할 수 없을까?
  - ⇒Edge 화소(pixel)의 수로 정상/비정상을 구분할 수 없을까?



정상(0)

"강단에 사람 edge가 포함된 경우"

Edge가 발견되는 화소의 수 = **5500**



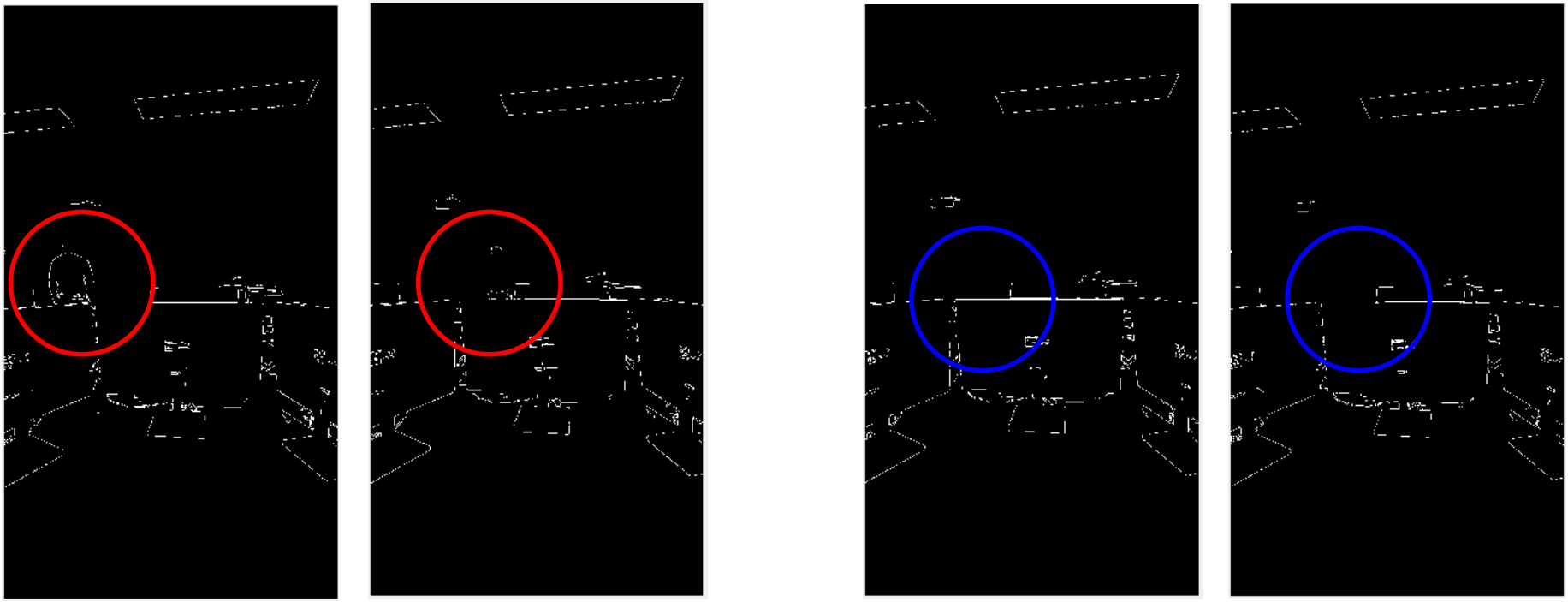
비정상(1)

"강단에 사람 edge가 발견되지 않는 경우"

Edge가 발견되는 화소의 수 = **5000**

5.5 Edge와 실시간 matplot 그래프

- Edge로 탐지되는 화소의 수와 임계값
  - 강단에 **사람이 없는 경우**에서의 edge의 수와 강단에 **사람이 있는 경우**의 edge의 수를 대략적으로 알고 있어야만 적절한 임계값을 설정할 수 있다
  - 같은 상황일지라도 공간 안의 조명 변화에 따라 edge 검출의 결과가 달라질 수 있음



5.5 Edge와 실시간 matplot 그래프

- Edge가 탐지된 동영상의 실시간 그래프 그리기
  - FigureCansvas는 그림이 그려지는 영역을 정의하며, Figure 객체를 생성자의 인자로 사용

```
from matplotlib.backends.backend_qtagg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure
import matplotlib

class Window(QMainWindow):
    def __init__(self):
        super().__init__()
        self.MODE_VIDEO = False
        self.EDGE_TYPE = None

        self.canvas = FigureCanvas(Figure(figsize=(0, 0.5)))
        self.axes = self.canvas.figure.subplots()

        self.axes.set_ylim([6000,10900])

        n_data = 50
        self.xdata = list(range(n_data))
        self.axes.set_xticks(self.xdata, [])
        self.ydata = [0 for i in range(n_data)]
```

5.5 Edge와 실시간 matplot 그래프

- Edge가 탐지된 동영상의 실시간 그래프 그리기

```
def update_plot(self, img):

    temp = img > 250
    sum_value = temp.sum()

    self.ydata = self.ydata[1:] + [sum_value]

    if self.previous_plot is None:
        self.previous_plot = self.axes.plot(self.xdata, self.ydata, 'r')[0]
    else:
        self.previous_plot.set_ydata(self.ydata)

    # Trigger the canvas to update and redraw.
    self.canvas.draw()
```

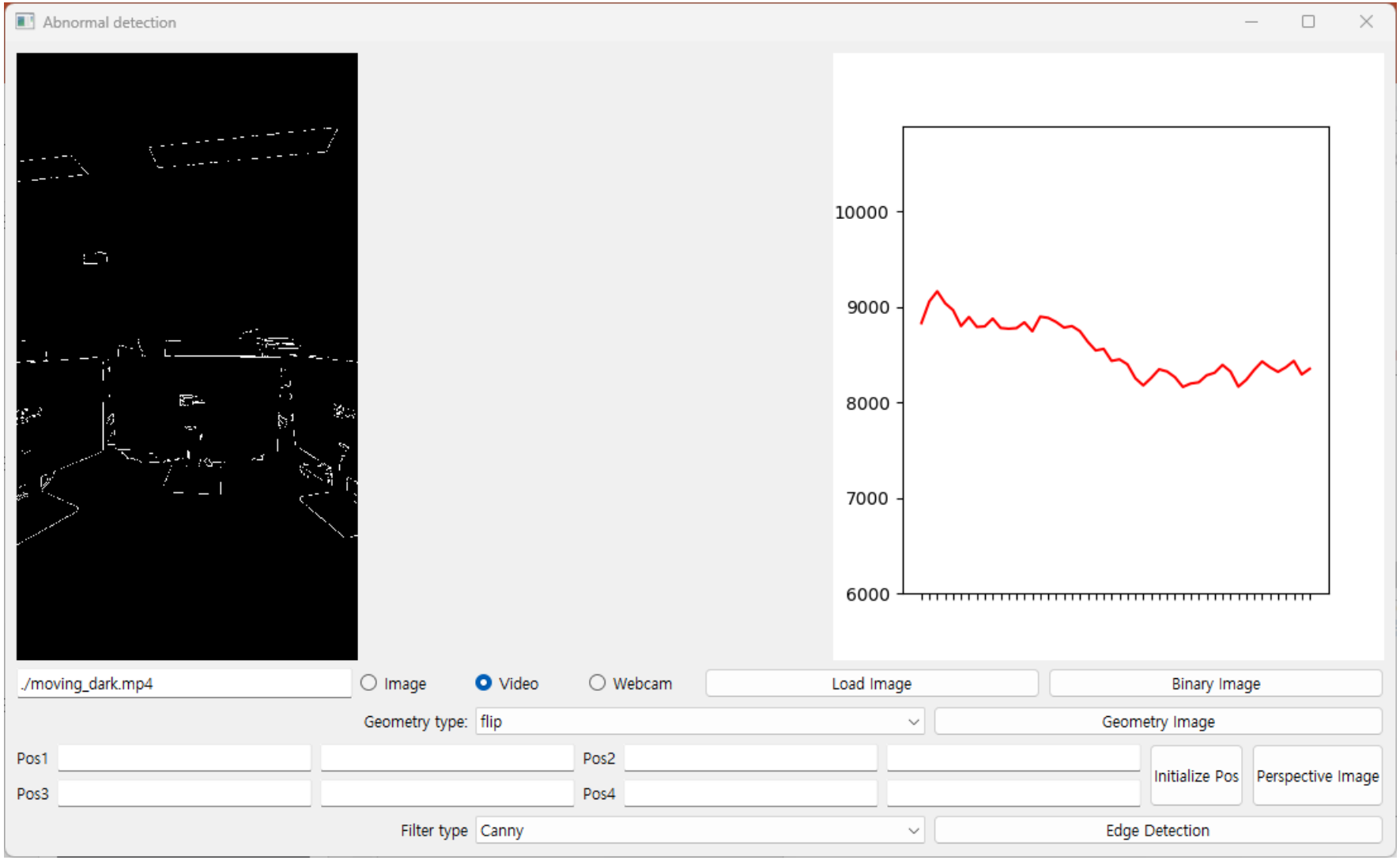
5.5 Edge와 실시간 matplot 그래프

- Edge가 탐지된 동영상의 실시간 그래프 그리기

```
def display_video_stream(self):
    retval, self.m_proc_img = self.capture.read()
    if not retval:  # 새로운 프레임을 못받아 왔을 때 braek
        return

    if self.EDGE_TYPE == 'Laplacian':
        self.m_proc_img = cv2.Laplacian(self.m_proc_img, cv2.CV_8U, ksize=3)
        self.update_plot(self.m_proc_img)
    elif self.EDGE_TYPE == 'Canny':
        self.m_proc_img = self.edge_det_canny(self.m_proc_img)
        self.update_plot(self.m_proc_img)

    self.update_image(self.m_proc_img)

    if self.MODE_VIDEO is False:
        self.timer.stop()
```

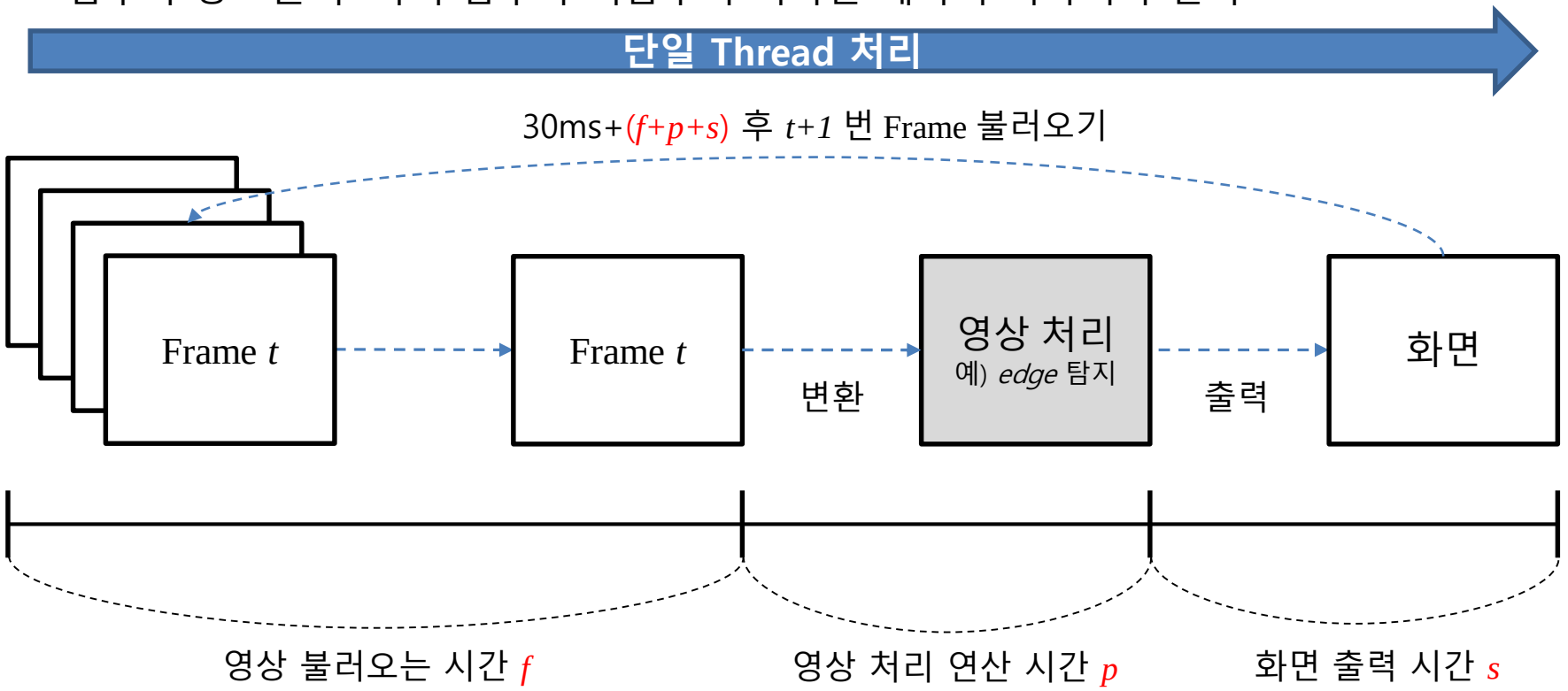
5.5 Edge와 실시간 matplot 그래프

- Edge가 탐지된 동영상의 실시간 그래프 그리기 결과화면



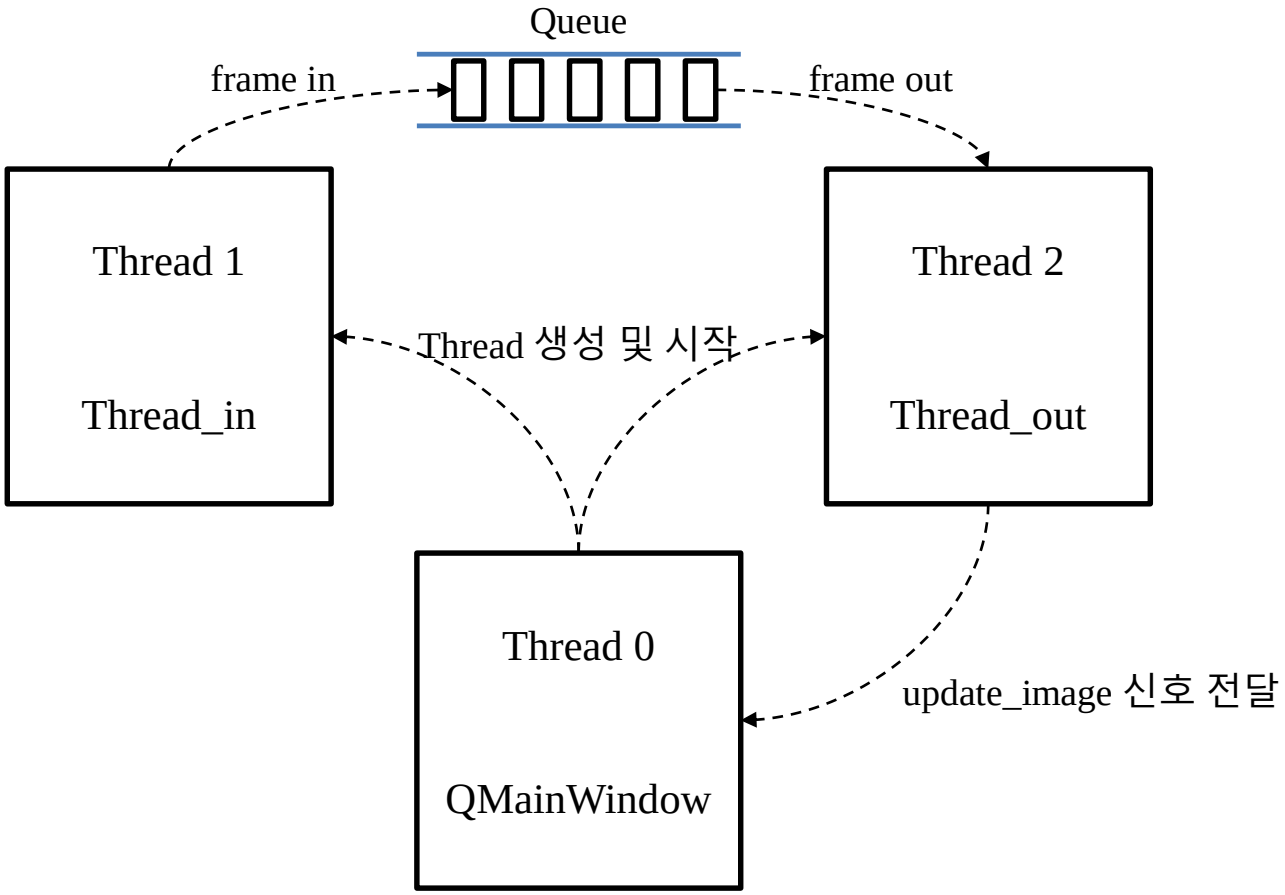
5.6 영상출력의 Thread 처리

- 동영상 처리에서 단일 Thread의 문제점
  - 단일 Thread 상에서 영상 처리 단계는 한 frame의 영상을 화면에 출력하기 위한 시간에 상당한 영향을 미치게 된다 (시간 지연)
  - 프로그래밍 언어는 절차적 언어이기 때문에 다음 프레임을 불러오는 것은 Thread 내에서 함수가 종료된 후 다시 함수가 처음부터 시작할 때까지 기다려야 한다



5.6 영상출력의 Thread 처리

- Thread
  - 어떠한 프로그램 내에서, 프로세스 내에서 실행되는 흐름의 단위
  - 한 프로그램은 하나의 스레드를 가지고 있지만, 프로그램 환경에 따라 둘 이상의 스레드를 동시에 실행할 수 있으며, 이를 멀티스레드(multi-thread)라고 한다



5.6 영상출력의 Thread 처리

- Thread_in 클래스 정의
  - Thread_in 클래스는 비디오 영상 또는 웹캠으로부터 frame을 획득하고 큐에 frame을 삽입하는 Thread

```
from PySide6.QtCore import QThread

class Thread_in(QThread):
    updateFrame = Signal(object)

    def __init__(self, img_queue, parent=None):
        QThread.__init__(self, parent)
        self.status = True
        self.capture = None
        self.qu = img_queue
        self.vid = None

# 현재 오른쪽 def run(self) 함수를 이어서 작성하세요.
```

```
    def run(self):
        self.capture = cv2.VideoCapture(self.vid)
        if not self.capture.isOpened():
            print("Camera open failed")

        prevTime = 0
        fps = 24
        while self.status:
            curTime = time.time()   # 현재 시간
            sec = curTime - prevTime
            if sec > 1/fps:
                prevTime = curTime
                ret, frame = self.capture.read()
                if not ret:
                    continue
                self.qu.put(frame)
        sys.exit(-1)
```

5.6 영상출력의 Thread 처리

- Thread_out 클래스 정의
  - Thread_out 클래스는 큐로부터 frame을 꺼내고 영상처리 및 화면 출력을 수행

```
from PySide6.QtCore import QThread

class Thread_out(QThread):
    updateFrame = Signal(object)
    updatePlot = Signal(object)
    def __init__(self, img_queue, parent=None):
        QThread.__init__(self, parent)
        self.status = True
        self.qu = img_queue
        self.EDGE_TYPE = None
    def run(self):
        while self.status:
            frame = self.qu.get()
            if self.EDGE_TYPE == 'Laplacian':
                frame = cv2.Laplacian(frame, cv2.CV_8U, ksize=3)
                self.updatePlot.emit(frame)
            elif self.EDGE_TYPE == 'Canny':
                frame = cv2.Canny(frame, 150, 300)
                self.updatePlot.emit(frame)

            self.updateFrame.emit(frame)      # Emit signal
        sys.exit(-1)
```

5.6 영상출력의 Thread 처리

- Thread 클래스 객체 생성

Thread 간의 Frame 전달을 위한 큐 생성

Thread_in 과 Thread_out 클래스 생성

Thread의 종료 이벤트 처리

Thread_out의 영상 및 그래프 출력을 위한 Slot 연결

```
class Window(QMainWindow):
    def __init__(self):
        super().__init__()

        self.qu = queue.Queue()

        self.th_in = Thread_in(self.qu)
        self.th_out = Thread_out(self.qu)

        self.th_in.finished.connect(self.close)
        self.th_out.finished.connect(self.close)

        self.th_out.updateFrame.connect(self.update_image)
        self.th_out.updatePlot.connect(self.update_plot)
```