라즈베리파이 텔레그램 봇 만들기

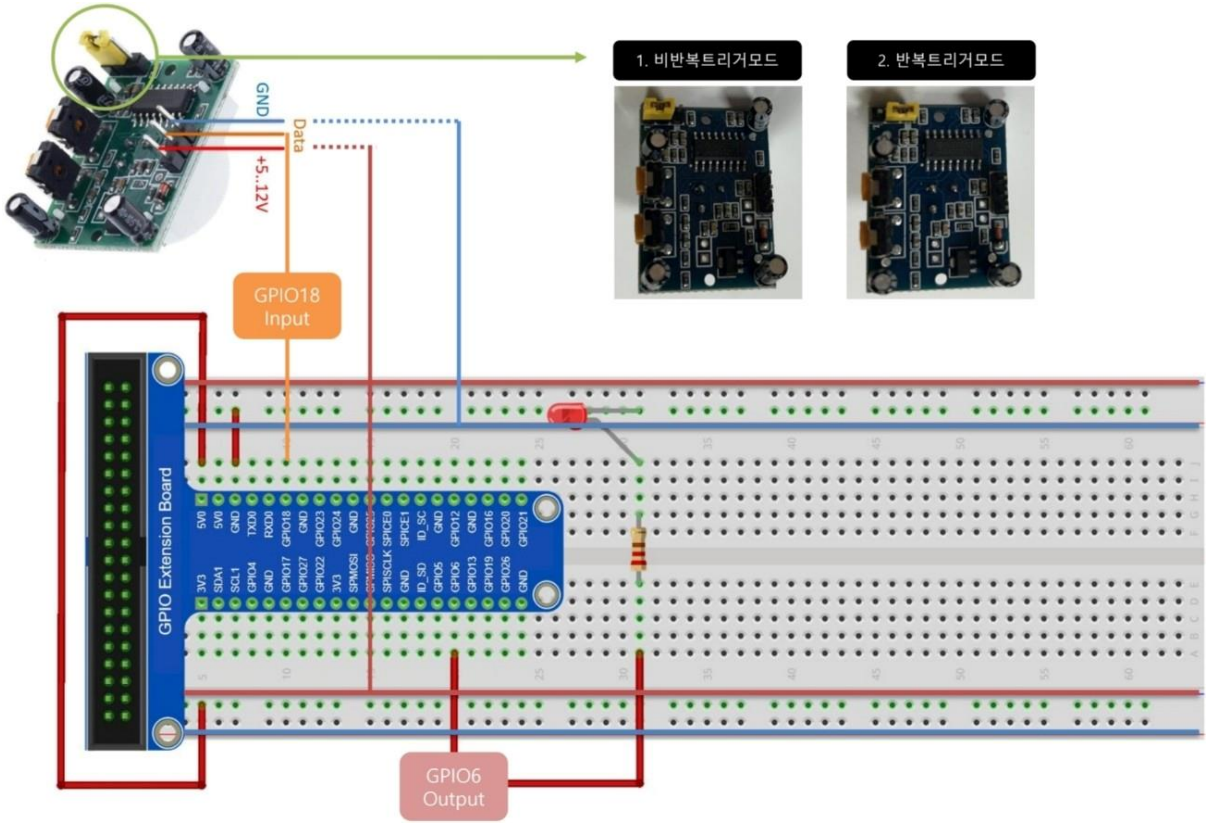
- 카메라와 인체감지센서를 활용한 도둑잡기 봇 -

이 름	박기쁨
학 과	산업보안학과
학 번	32231594

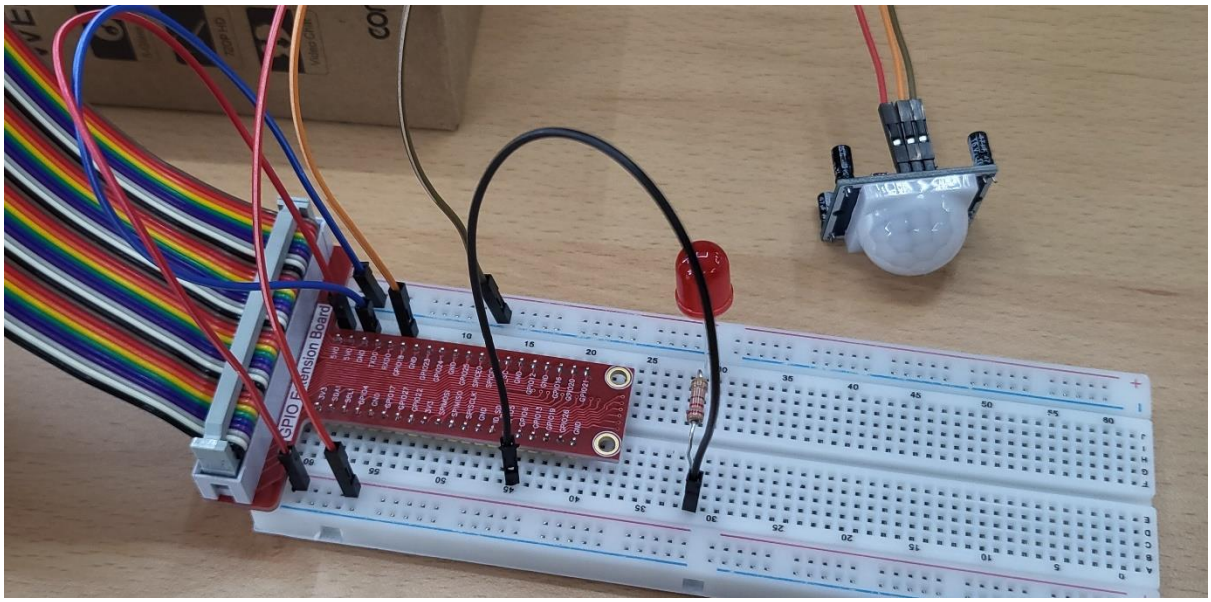
1. 준비물 및 회로 소개

1-1. 준비물: 라즈베리파이 보드, 카메라, 인체감지센서, 저항, LED 전구, 브레드보드, T자형 GPIO 확장 보드, 40핀 플랫 케이블, M/M 점퍼 와이어

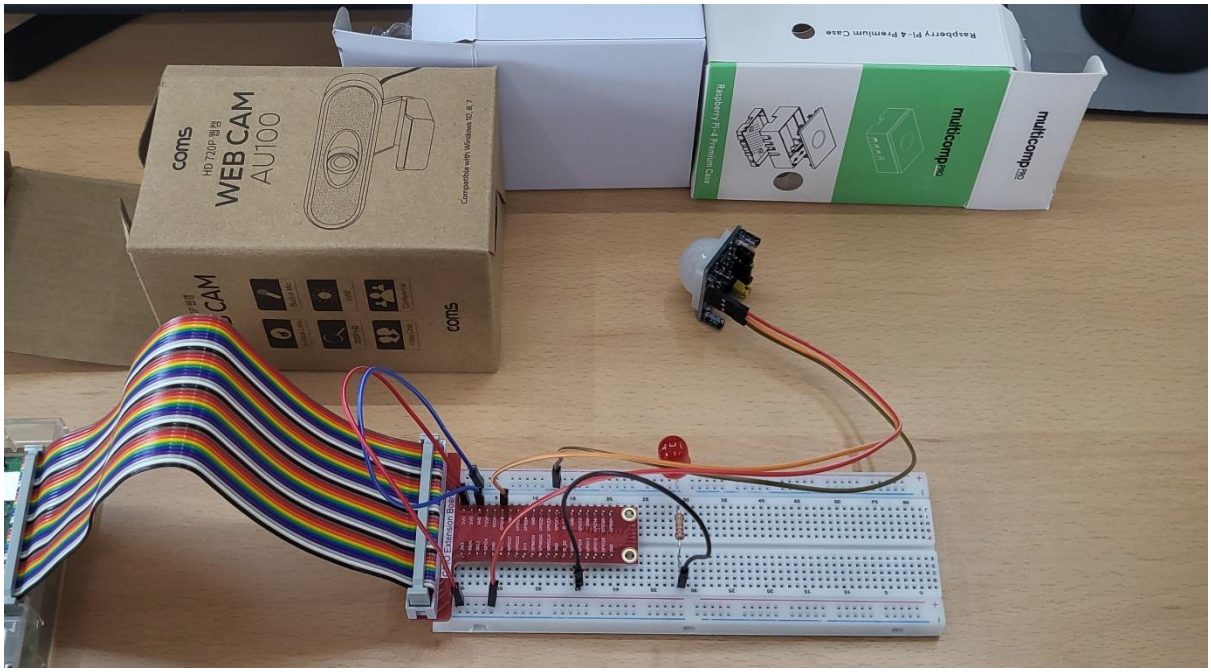
1-2. 회로



(회로도)



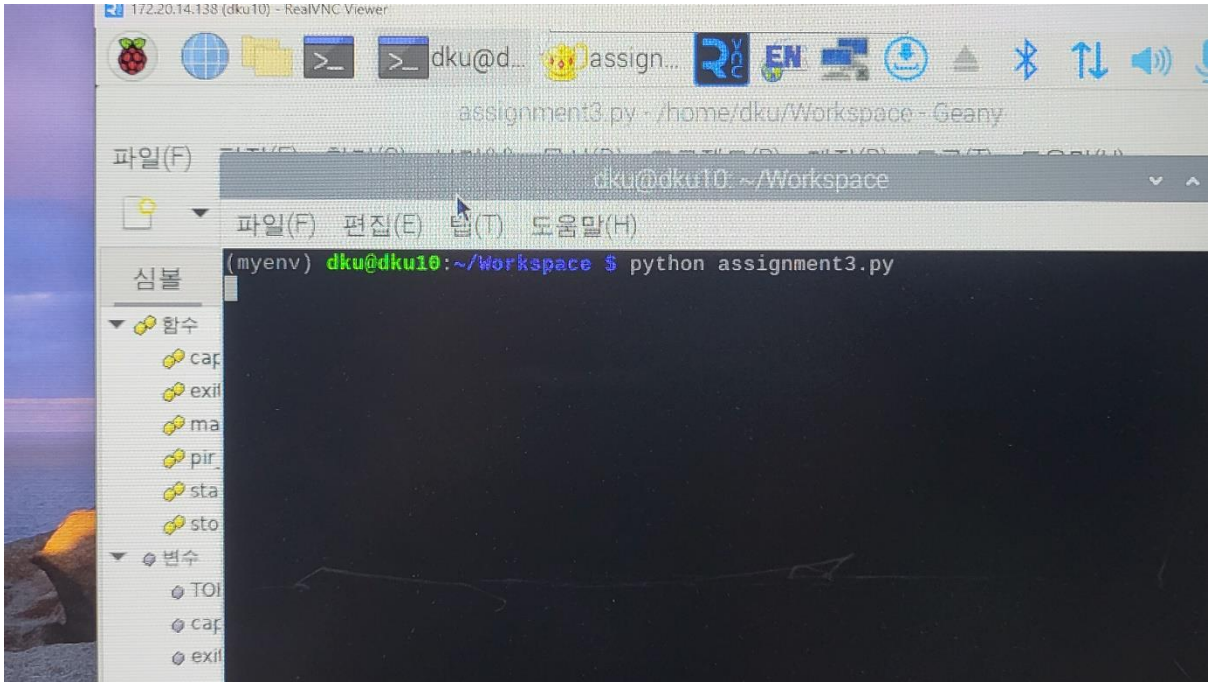
(실제 회로)



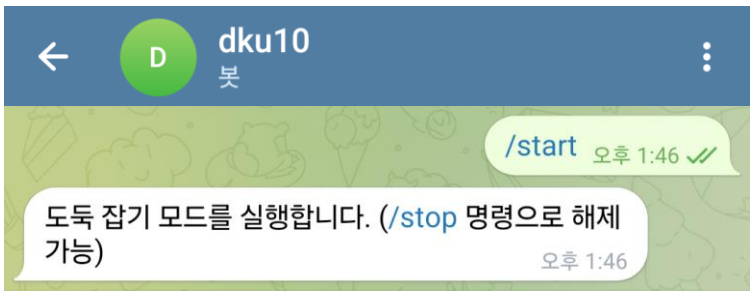
(실습 환경 (카메라는 라즈베리파이 USB 포트에 연결))

2. 동작 사진 및 동영상

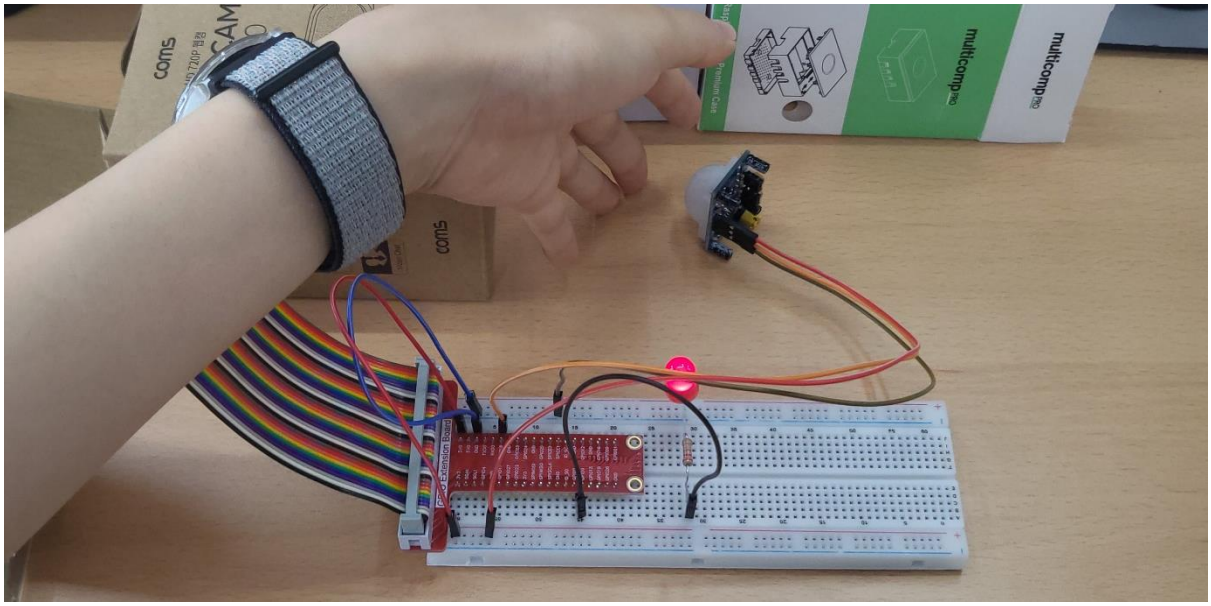
2-1. 가상환경을 활성화한 후 Python 코드를 실행한다.

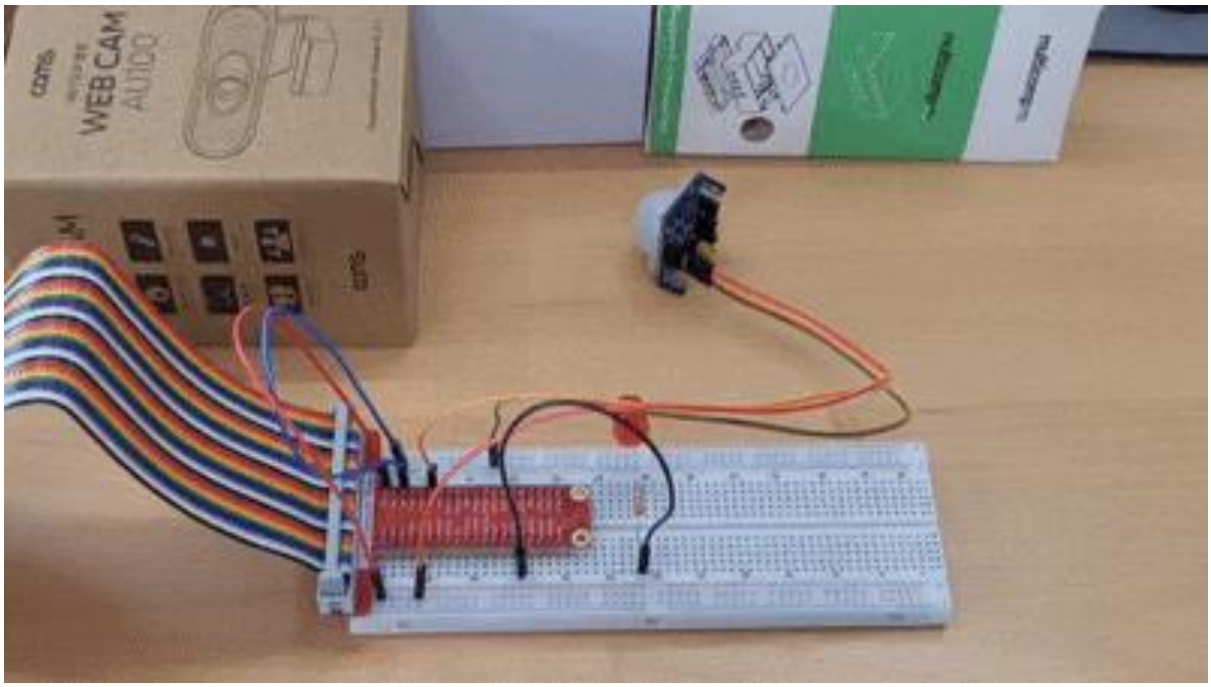


2-2. 텔레그램 봇에게 /start 명령을 전송한다. 도둑 잡기 모드가 실행된다.



2-3. 인체 감지 센서에 신체가 감지되면 LED가 (동영상이 촬영되는)10초간 켜진다.

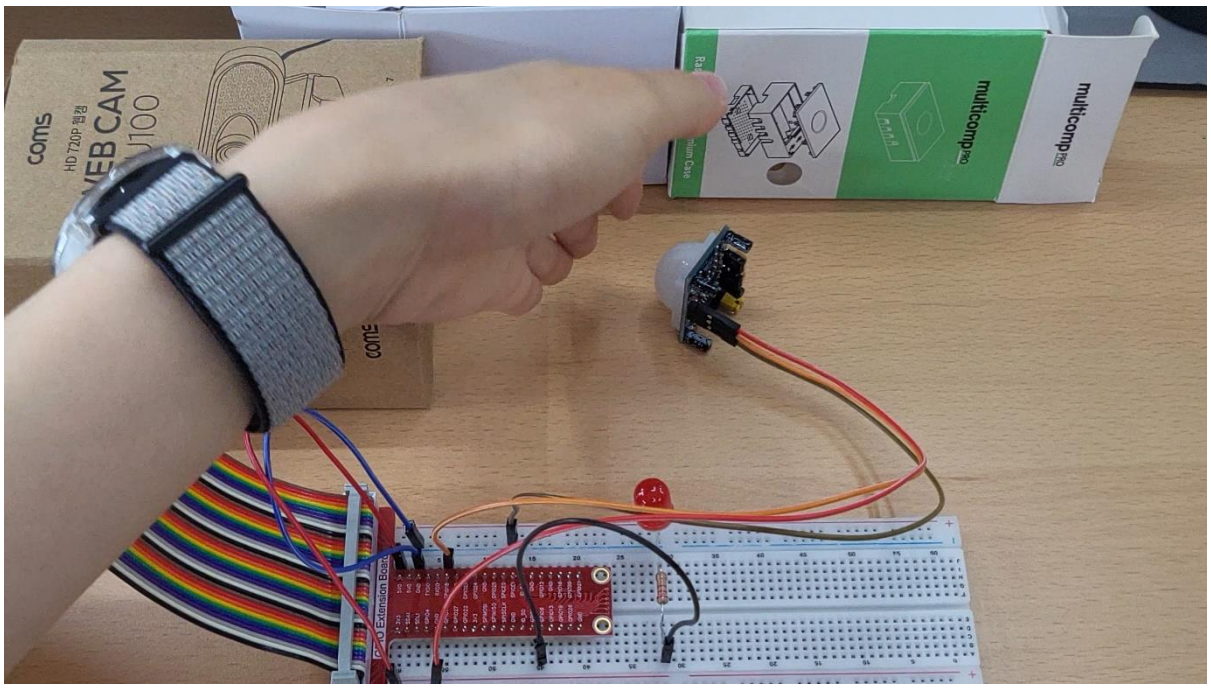


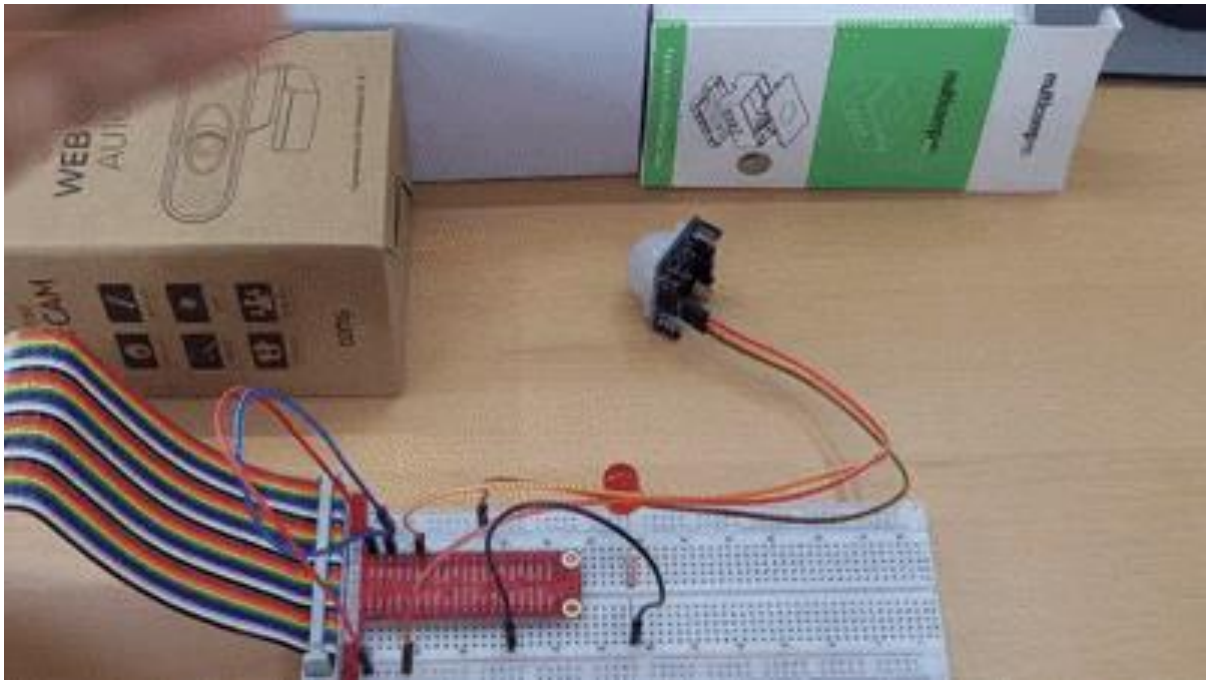


2-4. 텔레그램 봇으로부터 움직임을 감지했다는 메시지를 수신하고, 이후 10초간 촬영한 동영상을 수신한다.

2-5. 텔레그램 봇에게 /stop 명령을 전송하여 도둑 잡기 모드를 (일시) 해제할 수 있다.

2-6. 도둑 잡기 모드가 해제되면 인체 감지 센서의 작동이 일시 중단된다.

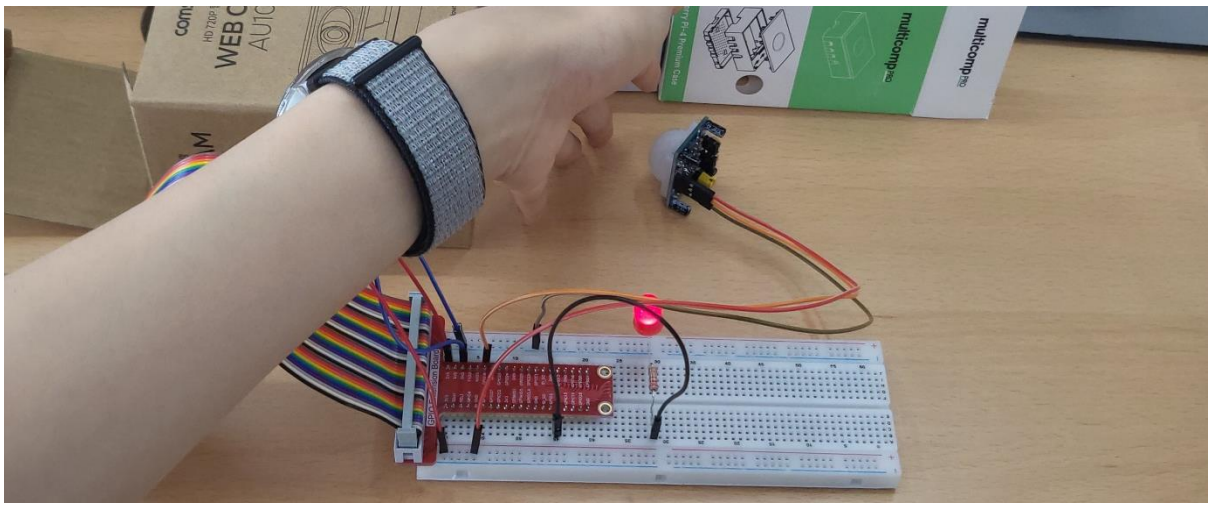


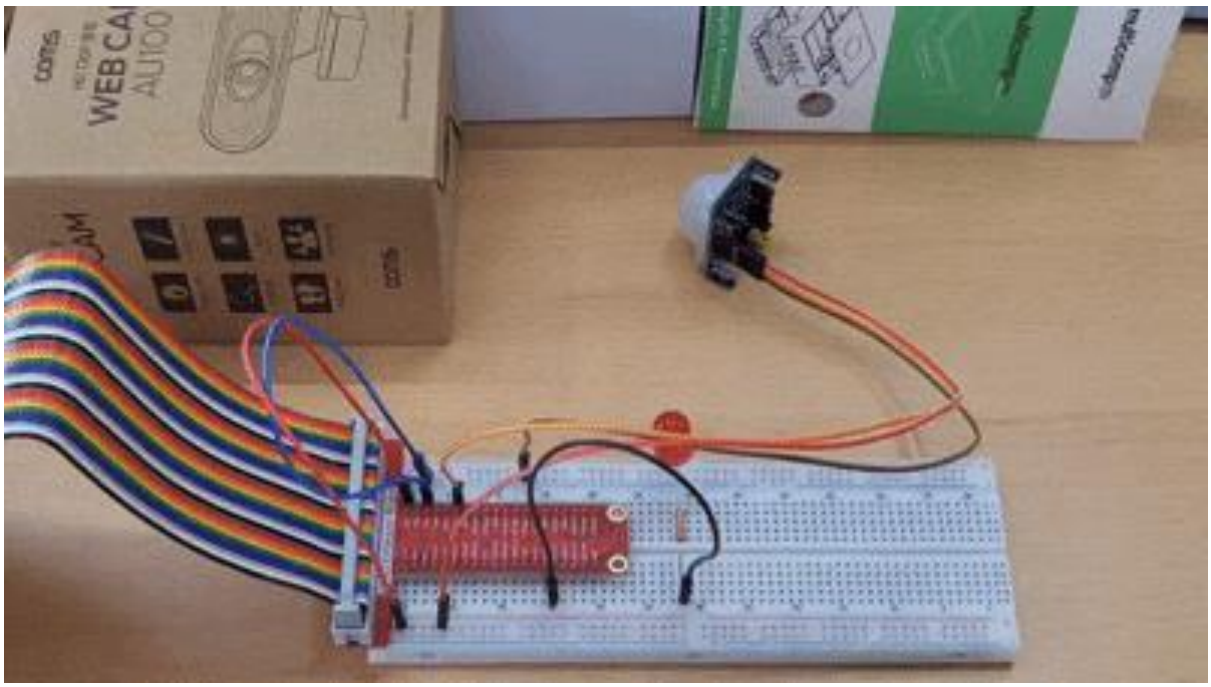


(다음 페이지에서 계속)

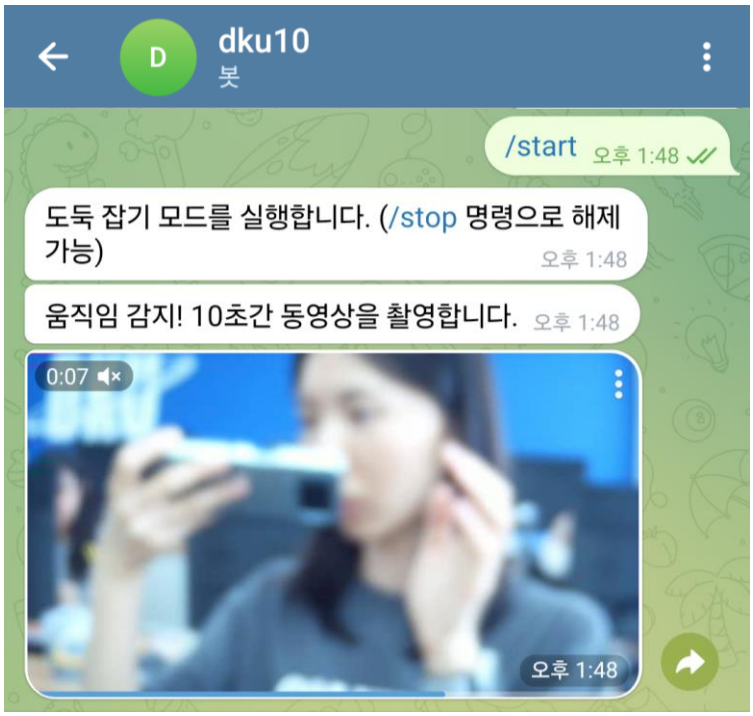
2-7. 텔레그램 봇에게 다시 /start 명령을 전송하여 도둑 잡기 모드를 재실행 할 수 있다.

2-8. 다시 인체 감지 센서가 감지를 시작하고, 신체가 감지되면 10초간 LED가 켜진다.

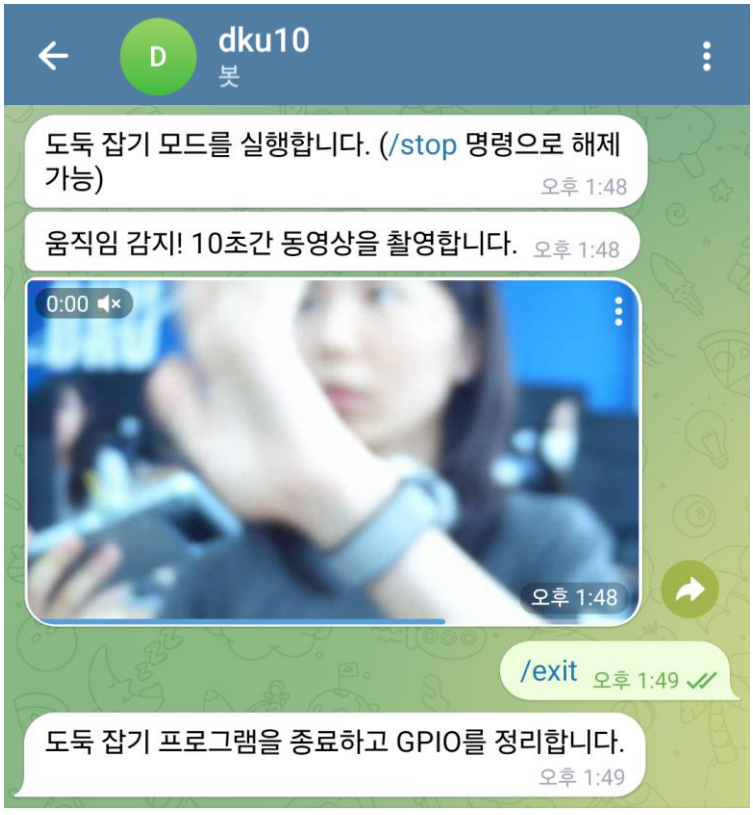




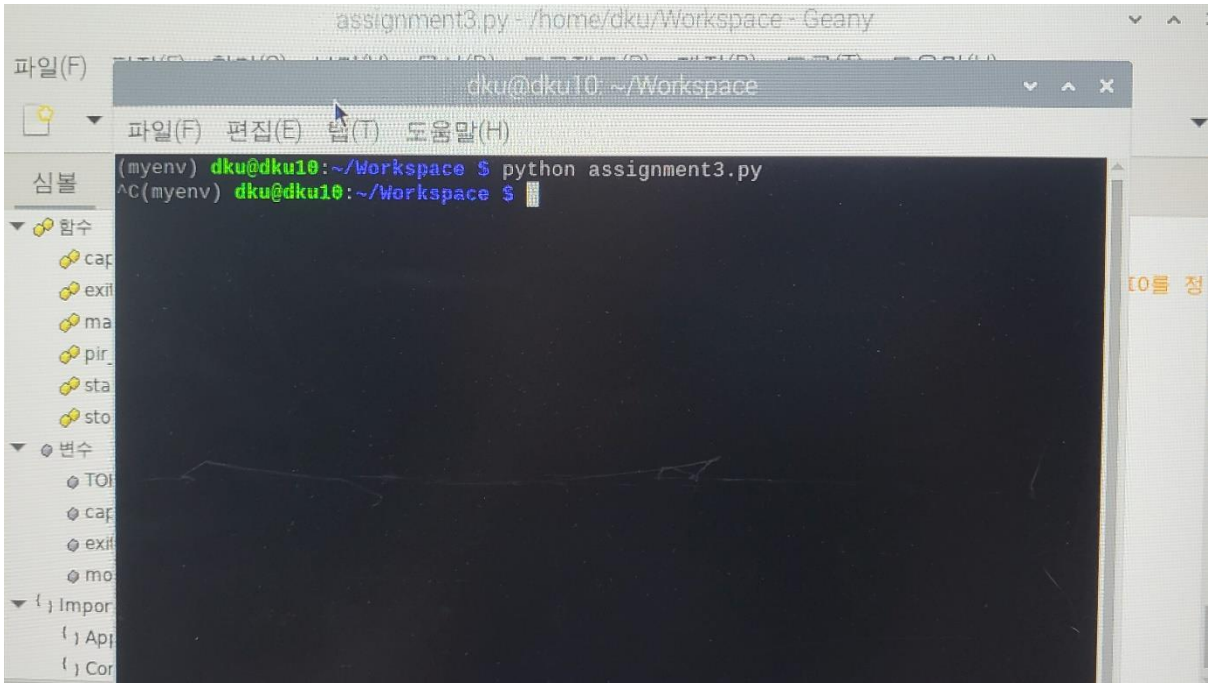
2-9. 텔레그램 봇으로부터 움직임을 감지했다는 메시지를 수신하고, 이후 10초간 촬영한 동영상을 수신한다.



2-10. 텔레그램 봇에게 /exit 명령을 전송하면 도둑 잡기 프로그램을 완전히 종료한다.



2-11. 이후 실행했던 Python 코드를 Ctrl+C로 종료한다.



3. 코드 소개

3-1. 토큰 및 로그

```
1   import asyncio              # 비동기 I/O를 위한 asyncio 모듈 (백그라운드 태스크 실행 등에 사용)
2   import time                 # 시간 측정을 위한 모듈 (녹화 시간 체크 등에 사용)
3   import RPi.GPIO as GPIO     # 라즈베리파이 GPIO 핀 제어 모듈
4   import cv2                  # OpenCV - 비디오 촬영 및 저장을 위한 라이브러리
5   import logging              # 로깅 설정 모듈 (디버깅 시 로그 출력에 사용됨)
6   from telegram import Update  # Telegram Bot API 사용을 위한 모듈
7   from telegram.ext import ApplicationBuilder, CommandHandler, ContextTypes  # Telegram Bot 명령어 처리 프레임워크
8   from typing import Optional  # Optional 타입 힌트 제공을 위한 typing 모듈
9
10  # Telegram Bot API 인증 토큰
11  TOKEN = "8170323752:AAEo049JtFJM0TL0jP_md5bFxSpkvge1FuE"
12
13  # 로그 출력 형식과 레벨을 설정
14  logging.basicConfig(
15      format="%(asctime)s -%(name)s -%(levelname)s -%(message)s",
16      level=logging.WARNING
17  )
18
```

- line 1~8: 코드 실행을 위해 필요한 여러 모듈들을 import 한다.

- line 11: 봇을 작동시키는 데 필요한 API 토큰을 입력한다.

- line 14~17: 로그가 출력될 때 시간, 모듈 이름, 로그 레벨, 메시지 형식을 설정한다. WARNING 레벨 이상만 출력한다.

3-2. 동영상 촬영 함수(1)

```
20      # 동영상 촬영 함수 (움직임 감지 시 호출됨)
        1 usage
21      def capture_video() -> Optional[str]:
22          # 녹화된 영상을 저장할 경로 설정
23          video_path = "/tmp/capture.mp4"
24
25          # 카메라 장치 열기 (디바이스 0번, V4L2 backend 사용)
26          camera = cv2.VideoCapture(0, cv2.CAP_V4L)
27
28          # 카메라 오픈 실패 시 None 반환
29          if not camera.isOpened():
30              return None
31
32          # 해상도와 FPS 설정
33          camera.set(cv2.CAP_PROP_FRAME_WIDTH, 640)    # 가로 해상도 설정
34          camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 360)  # 세로 해상도 설정
35          camera.set(cv2.CAP_PROP_FPS, 10)             # 초당 프레임 수 설정
36
37          # mp4v 코덱 설정 (OpenCV에서 mp4 파일을 저장할 때 사용)
38          fourcc = cv2.VideoWriter_fourcc(*"mp4v")
39
40          # 비디오 라이터 객체 생성 (파일명, 코덱, FPS, 해상도)
41          writer = cv2.VideoWriter(video_path, fourcc, 10, (640, 360))
42
43          # 현재 시간 저장 → 녹화 시간 측정용
44          start_time = time.time()
45
```

- **line 21**: 동영상 촬영함수를 정의한다. 반환값이 string일 수도 (함수가 정상적으로 종료되어 video_path를 반환하는 경우), None일 수도 (동영상 파일 열기에 실패하는 경우) 있기 때문에, 타입 힌트(-> Optional[str]:)를 사용한다.

- **line 23**: 녹화된 영상을 저장할 경로를 설정한다.

- **line 26**: 첫 번째 파라미터 0은 0번 카메라를 의미하고, 리눅스의 경우 두 번째 파라미터 값으로 cv2.CAP_V4L을 사용한다.

- **line 29~30**: cv2.VideoCapture 객체가 생성될 때, 매개변수로 주어진 동영상 파일이 성공적으로 열리지 않았을 경우 None을 반환한다.

- **line 33~35**: 프레임 크기를 640*360으로 하고, camera가 카메라로부터 읽어오는 주기(FPS)를 10으로 설정한다.

- **line 38**: 코덱을 mp4v로 지정한다.

- **line 41**: writer 객체를 생성하며, 파일명(경로), 코덱, FPS, 해상도를 지정한다.

- **line 44**: 녹화 시간을 측정하여 10초로 지정하기 위해 현재 시간을 저장한다.

(다음 페이지에서 계속)

3-3. 동영상 촬영 함수(2) 및 플래그 변수

```
46          # 10초 동안 프레임을 읽어서 파일에 저장
47          while time.time() - start_time < 10:
48              ret, frame = camera.read()    # 카메라에서 프레임 읽기
49              if not ret:                   # 읽기 실패 시 종료
50                  break
51              writer.write(frame)          # 프레임을 비디오 파일에 저장
52
53          # 사용한 리소스 해제
54          writer.release()
55          camera.release()
56
57          # 저장된 파일 경로 반환
58          return video_path
59
60
61      # 상태 플래그 변수 (전역 변수로 상태 관리)
62      monitoring = False      # 도둑 잡기 모드 실행 여부
63      capturing = False       # 현재 비디오 촬영 중 여부
64      exit_requested = False  # 프로그램 종료 요청 여부
65
```

- line 47~51: 녹화 시간이 10초 이내일 때, 카메라에서 프레임을 읽는다. 녹화 시간이 10초가 되거나 읽기에 실패하면 종료하고, 프레임을 동영상 파일에 저장한다.

- line 54~55: 동작 종료 후 사용한 writer, camera 객체를 해제한다.

- line 58: 저장된 동영상 파일 경로를 반환하며 함수를 마무리한다.

- line 62~64: 프로그램의 동작을 전역적으로 관리하기 위해, 도둑 잡기 모드 실행 여부, 동영상 촬영 중 여부, 프로그램 종료 요청 여부 플래그를 생성한다.

3-4. PIR 센서 감시 함수(1)

```
67      # PIR 센서 감시용 비동기 루프 함수
68      # (백그라운드에서 지속적으로 PIR 센서를 감시)
        1 usage
69      async def pir_monitor(chat_id: int, application):
70          global monitoring, capturing, exit_requested
71
72          # GPIO 핀 번호 설정 (BCM 기준)
73          led = 6          # LED를 연결한 GPIO 번호
74          pir_sensor = 18  # PIR 센서를 연결한 GPIO 번호
75
76          # GPIO 초기화 설정
77          GPIO.setwarnings(False)       # 경고 메시지 비활성화
78          GPIO.setmode(GPIO.BCM)        # 핀 번호를 BCM 모드로 설정
79
80          # 핀의 입출력 모드 설정
81          GPIO.setup(led, GPIO.OUT)     # LED 핀을 출력 모드로 설정
82          GPIO.setup(pir_sensor, GPIO.IN)  # PIR 센서 핀을 입력 모드로 설정
83
```

- line 69~70: PIR 센서를 감시할 비동기 핸들러 함수를 정의한다. 앞서 생성한 3개의 플래그 또한 함수 내부에 명시한다.

- line 73~74: LED와 PIR 센서를 연결한 GPIO 번호를 각각 6번과 18번으로 명시한다.

- line 77~78: 경고 메시지를 비활성화하고, GPIO 핀 번호를 BCM 모드로 설정한다.

- line 81~82: LED 핀과 PIR 핀을 각각 출력 모드와 입력 모드로 설정한다.

3-5. PIR 센서 감시 함수(2)

```
84      try:
85          # 메인 감시 루프
86          while not exit_requested:
87              # /exit 명령이 오면 exit_requested = True → 루프 종료
88
89              if monitoring:
90                  # 도둑 감지 모드가 활성화된 경우
91
92                  # PIR 센서에 움직임 감지 && 현재 촬영 중이 아닐 때
93                  if GPIO.input(pir_sensor) == 1 and not capturing:
94                      capturing = True              # 촬영 중 상태로 전환
95                      GPIO.output(led, GPIO.HIGH)   # LED ON
96
97                      # 텔레그램으로 감지 메시지 전송
98                      await application.bot.send_message(
99                          chat_id=chat_id, text="움직임 감지! 10초간 동영상을 촬영합니다."
100                     )
101
```

- line 84~86: try-finally 문을 사용한다. /exit 명령이 오지 않는 동안 이하 명령을 수행하고, /exit 명령이 오면 루프를 종료한다. (플래그 이용)

- line 89: 도둑 감지 모드가 활성화된 경우 이하 명령을 수행한다. (플래그 이용)

- line 93~95: PIR 센서에 움직임이 감지되고, 현재 촬영 중이 아니라면 이하 명령을 수행한다. (플래그 이용)

- line 98~100: 움직임이 감지되어 동영상을 촬영한다는 메시지를 텔레그램으로 전송한다.

(다음 페이지에서 계속)

3-6. PIR 센서 감시 함수(3)

```
102                     # 비디오 촬영 실행
103                     video_path = capture_video()
104                     if video_path:
105                         with open(video_path, "rb") as vid:
106                             # 텔레그램으로 비디오 전송
107                             await application.bot.send_video(chat_id=chat_id, video=vid)
108                     else:
109                         # 에러 메시지 전송
110                         await application.bot.send_message(
111                             chat_id=chat_id, text="카메라를 사용할 수 없습니다."
112                         )
113 
114                     # 촬영 후 LED 끄기
115                     GPIO.output(led, GPIO.LOW)
119 
120                     # 촬영 완료 후 상태 초기화
121                     capturing = False
122 
123                 # 센서 폴링 주기 (0.3초마다 확인)
124                 await asyncio.sleep(0.3)
125         finally:
126             # 프로그램 종료 시 GPIO 핀 정리
127             GPIO.cleanup()
128 
```

- **line 103~107**: 동영상 촬영 함수를 호출하여 동영상 촬영을 시작하고, 만약 함수가 video_path를 반환하였다면 (동영상이 정상적으로 촬영되었다면), 동영상을 읽기 전용으로 열어 텔레그램으로 전송한다.

- **line 108~112**: 함수가 video_path를 반환하지 않았다면 (동영상이 정상적으로 촬영되지 않았다면), 카메라를 사용할 수 없다는 메시지를 텔레그램으로 전송한다.

- **line 115**: 동영상 촬영을 마친 후 LED를 끈다.

- **line 121**: 동영상 촬영을 마친 후 동영상 촬영 중 여부 플래그를 초기화한다.

- **line 124**: PIR 센서 폴링 주기를 0.3초로 설정하여 센서가 너무 자주 감지하지 않도록 한다.

- **line 125~127**: 프로그램이 종료되어 try 문을 탈출하면 GPIO 핀을 초기화한다.

(다음 페이지에서 계속)

3-7. /start 명령 핸들러

```
129
130     # /start 명령 핸들러 : 도둑 잡기 모드 시작
131     async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
132         global monitoring
133         if monitoring:
134             await update.message.reply_text("이미 도둑 잡기 모드가 실행 중입니다.")
135             return
136
137         # 도둑 잡기 모드 ON
138         monitoring = True
139
140         # 사용자에게 알림 전송
141         await update.message.reply_text("도둑 잡기 모드를 실행합니다. (/stop 명령으로 해제 가능)")
142
143         # 감시 루프를 백그라운드 태스크로 실행
144         context.application.create_task(
145             pir_monitor(update.effective_chat.id, context.application)
146         )
147
```

- line 131~132: /start 명령의 핸들러 함수를 정의한다. 도둑 잡기 모드 실행 여부 플래그 또한 함수 내부에 명시한다.

- line 133~135: 현재 도둑 잡기 모드 실행 중이라면, 이미 도둑 잡기 모드가 실행 중이라는 메시지를 텔레그램으로 전송한다.

- line 138~141: 위 경우가 아니라면 도둑 잡기 모드를 실행하고, 도둑 잡기 모드를 실행한다는 메시지를 텔레그램으로 전송한다.

- line 144~146: PIR 센서 감시 함수를 백그라운드 태스크로 실행한다.

3-8. /stop 명령 핸들러

```
149     # /stop 명령 핸들러 : 도둑 잡기 모드 종료
        1 usage
150     async def stop(update: Update, context: ContextTypes.DEFAULT_TYPE):
151         global monitoring
152         if not monitoring:
153             await update.message.reply_text("현재 도둑 잡기 모드 실행 중이 아닙니다.")
154             return
155
156         # 모니터링 종료
157         monitoring = False
158         await update.message.reply_text("도둑 잡기 모드를 해제합니다. (/start 명령으로 실행 가능)")
159
```

- line 150~151: /stop 명령의 핸들러 함수를 정의한다. 도둑 잡기 모드 실행 여부 플래그 또한 함수 내부에 명시한다.

- line 152~154: 도둑 잡기 모드 실행 중이 아니라면, 현재 도둑 잡기 모드가 실행 중이 아니라는 메시지를 텔레그램으로 전송한다.

- line 157~158: 위 경우가 아니라면 도둑 잡기 모드를 해제하고, 도둑 잡기 모드를 해제한다는 메시지를 텔레그램으로 전송한다.

3-9. /exit 명령 핸들러

```
161     # /exit 명령 핸들러 : 프로그램 완전 종료 + GPIO 정리
        1 usage
162     async def exit_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
163         global exit_requested, monitoring
164
165         # 종료 요청 및 모니터링 중단
166         exit_requested = True
167         monitoring = False
168
169         # 사용자에게 알림
170         await update.message.reply_text("도둑 잡기 프로그램을 종료하고 GPIO를 정리합니다.")
171
172         # GPIO 핀 초기화
173         GPIO.cleanup()
174
```

- line 162~163: /stop 명령의 핸들러 함수를 정의한다. 프로그램 종료 요청 여부 플래그, 도둑 잡기 모드 실행 여부 플래그 또한 함수 내부에 명시한다.

- line 166~167: 프로그램 종료 요청 여부 플래그를 활성화시키고, 도둑 잡기 모드 실행 여부 플래그를 해제시킨다.

- line 170~173: 도둑 잡기 프로그램을 종료하고 GPIO를 정리한다는 메시지를 텔레그램으로 전송하고, GPIO 핀을 초기화한다.

3-10. main 함수

```
176     # main 함수 - Telegram 봇 실행
177     def main() -> None:
178         # Telegram 봇 앱 생성
179         app = ApplicationBuilder().token(TOKEN).build()
180
181         # 명령어 핸들러 등록
182         app.add_handler(CommandHandler("start", start))        # /start → 도둑 잡기 모드 (모니터링) 시작
183         app.add_handler(CommandHandler("stop", stop))          # /stop  → 도둑 잡기 모드 (모니터링) 종료
184         app.add_handler(CommandHandler("exit", exit_command))  # /exit  → 프로그램 종료 및 GPIO 정리
185
186         # 봇 폴링 시작 (계속해서 명령을 기다림)
187         app.run_polling()
188
189
190     # Python 프로그램의 진입점
191     if __name__ == "__main__":
192         main()
193
```

- line 177: main 함수를 정의한다. 타입 힌트를 사용하여 반환 값이 없음을 명시한다.

- line 179: 텔레그램 봇을 생성하고, TOKEN을 설정한다.

- line 182~184: /start, /stop, /exit 명령을 핸들러로 등록한다.

- line 187: 봇을 실행한다. 텔레그램 서버에 지속적으로 폴링을 요청한다.

- line 191~192: 프로그램의 진입점이 된다.