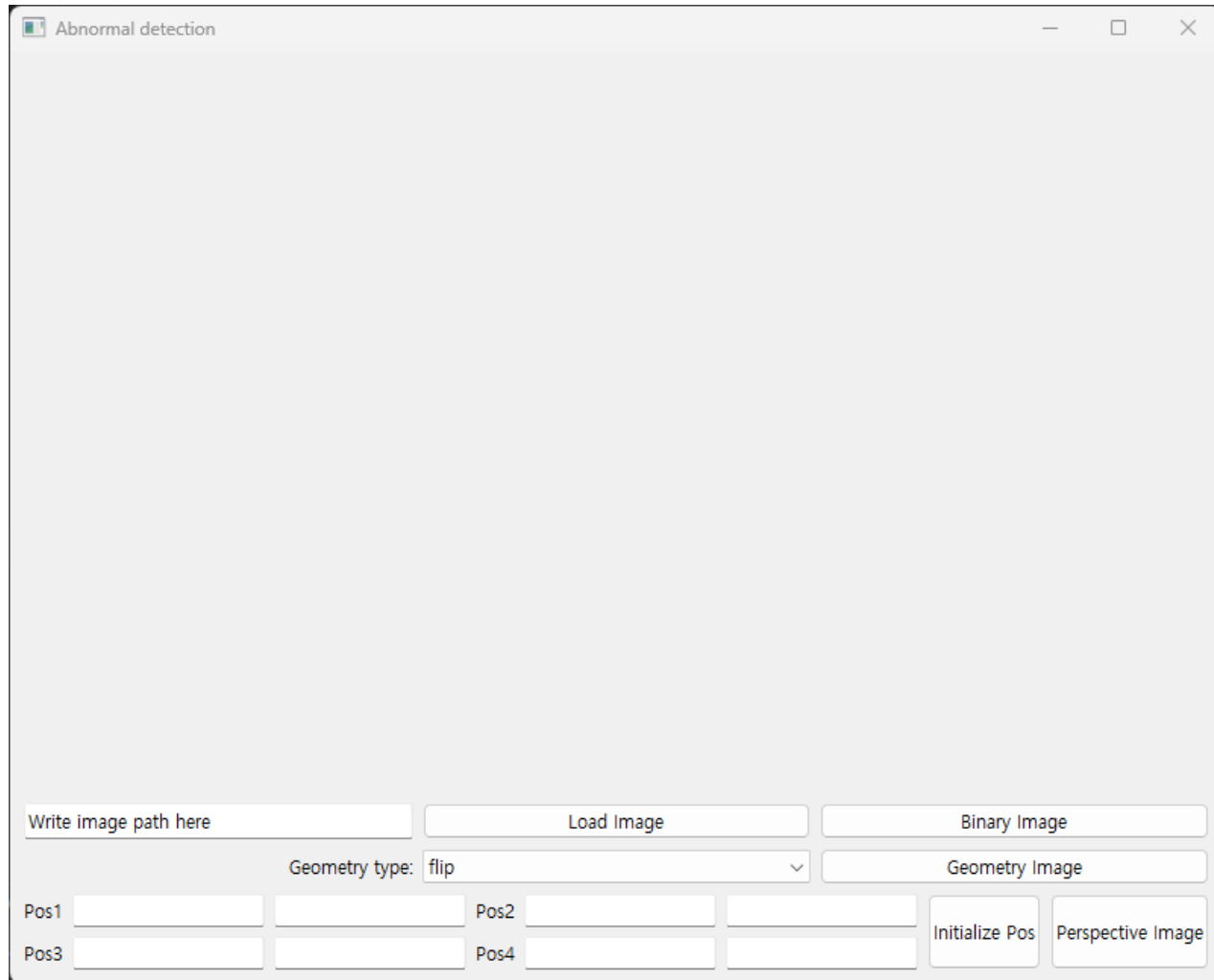


## 3-4. 실습

## ■ 주요 구성

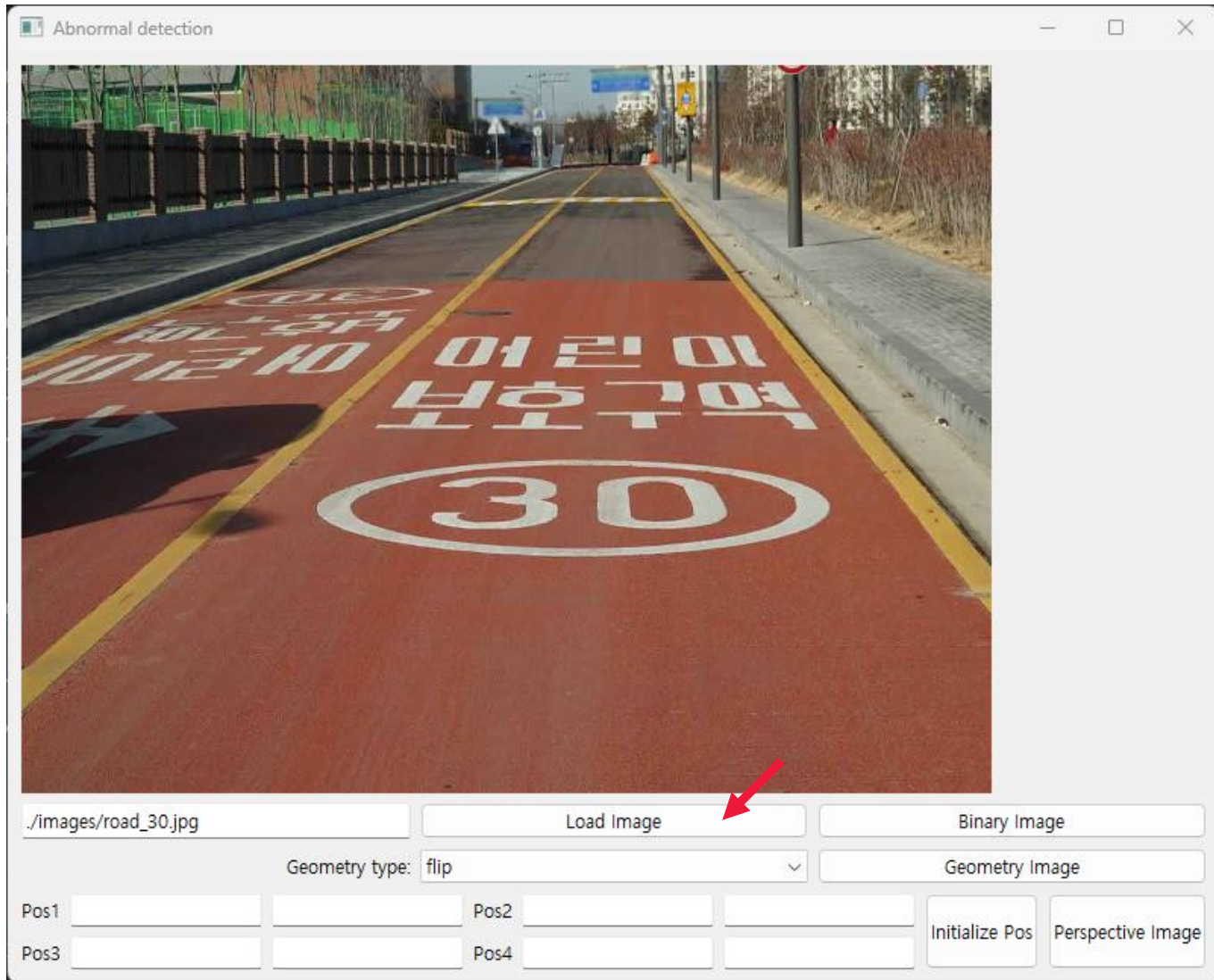


초기화면

# 마우스 클릭 이벤트 처리와 Perspective 기능 추가

3

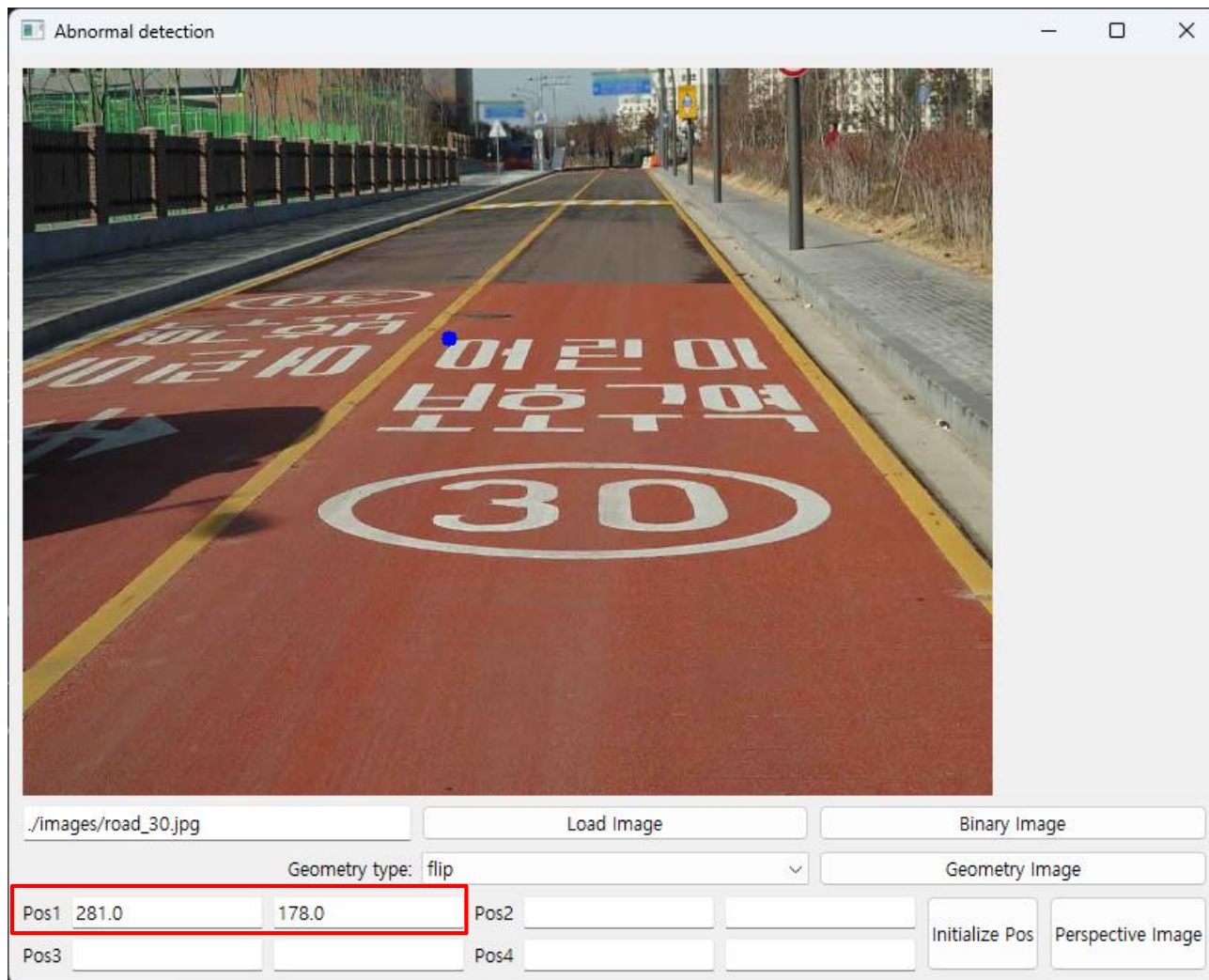
## 1) Load Image 버튼 실행



# 마우스 클릭 이벤트 처리와 Perspective 기능 추가

4

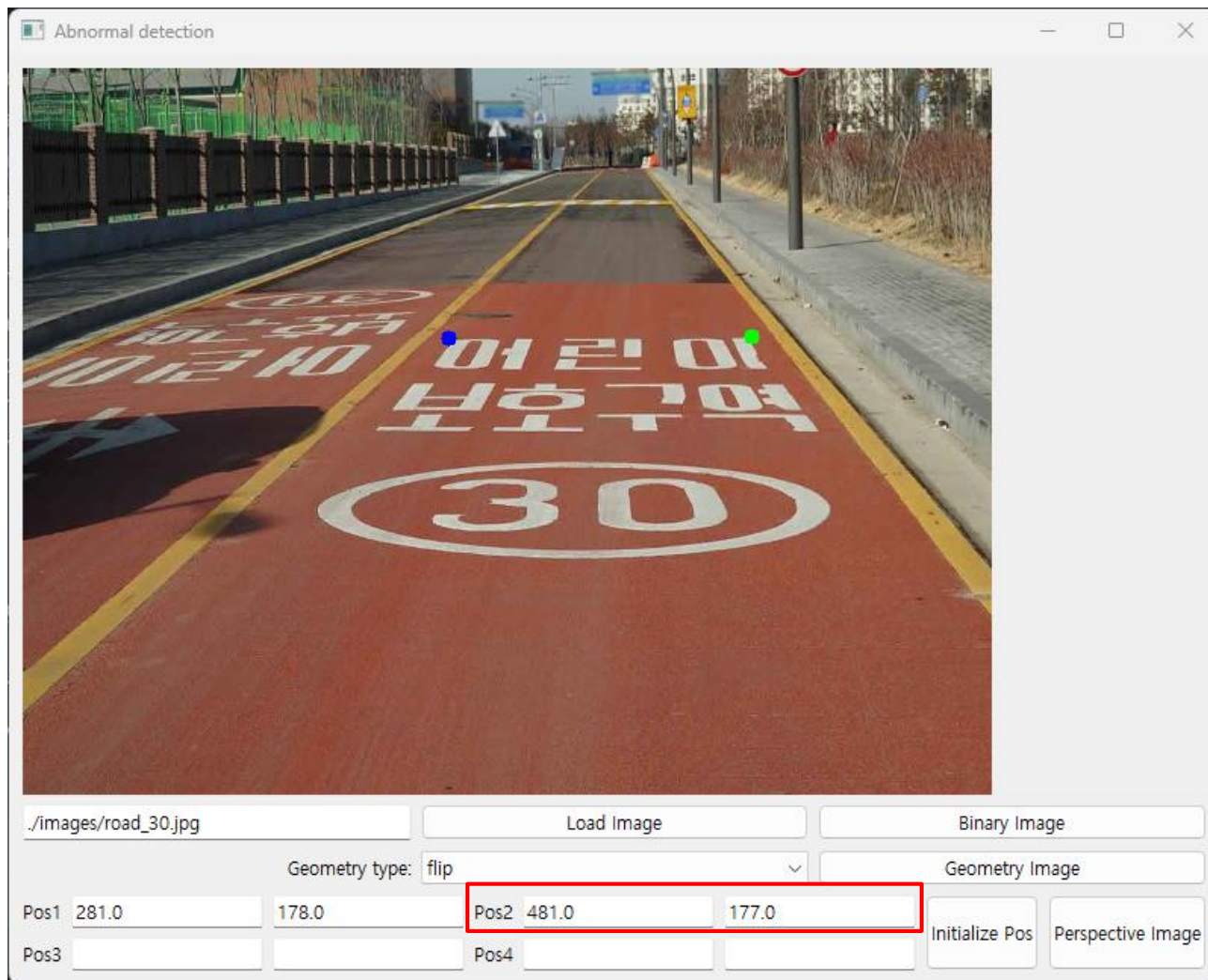
2) 첫 번째 좌표에서 마우스 클릭 시 **파란색 점**이 찍히고 Pos1에 자동으로 좌표 입력



# 마우스 클릭 이벤트 처리와 Perspective 기능 추가

5

3) 두 번째 좌표에서 마우스 클릭 시 초록색 점이 찍히고 Pos2에 자동으로 좌표 입력





# 마우스 클릭 이벤트 처리와 Perspective 기능 추가

6

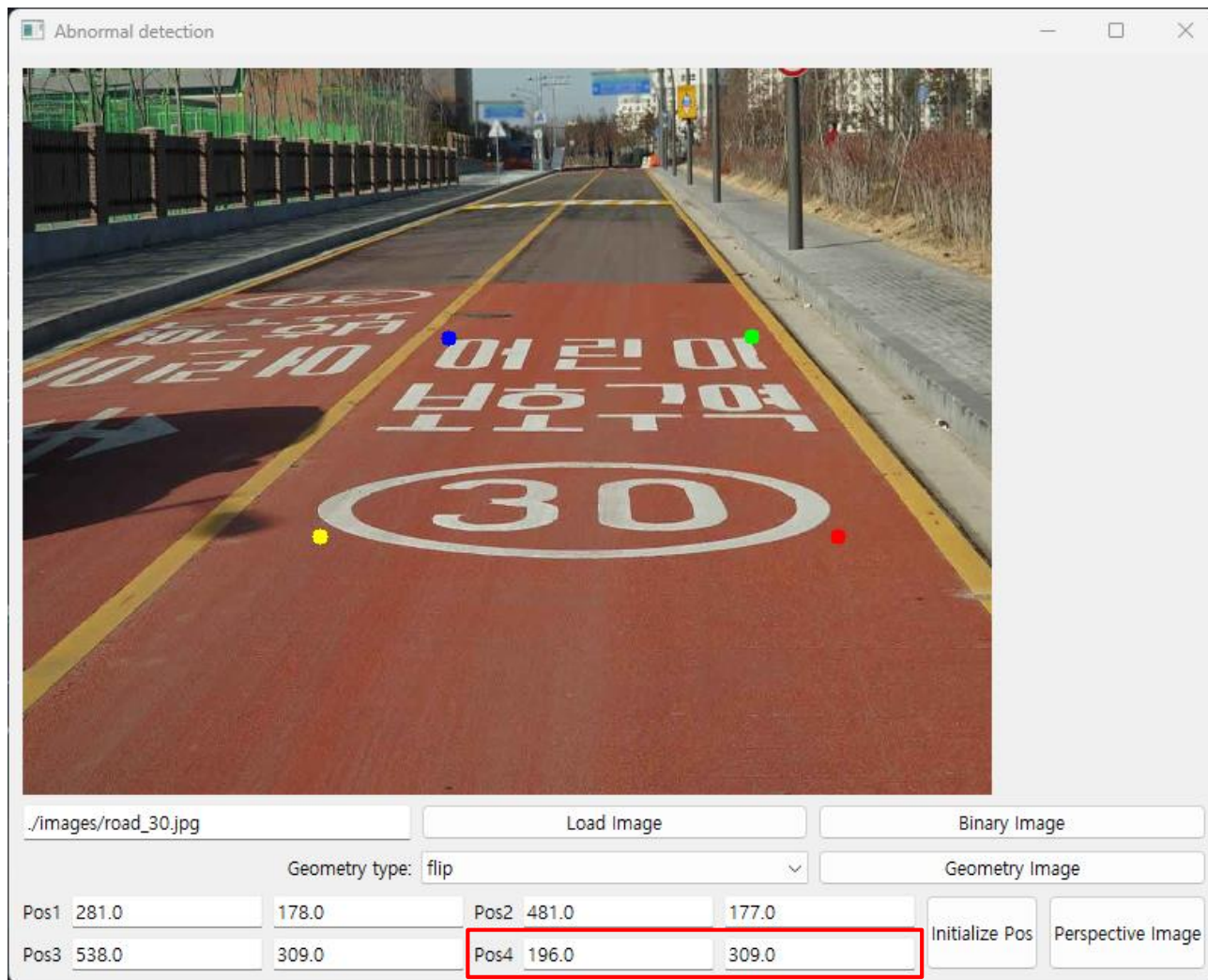
4) 세 번째 좌표에서 마우스 클릭 시 **빨간색 점**이 찍히고 Pos3에 자동으로 좌표 입력



# 마우스 클릭 이벤트 처리와 Perspective 기능 추가

7

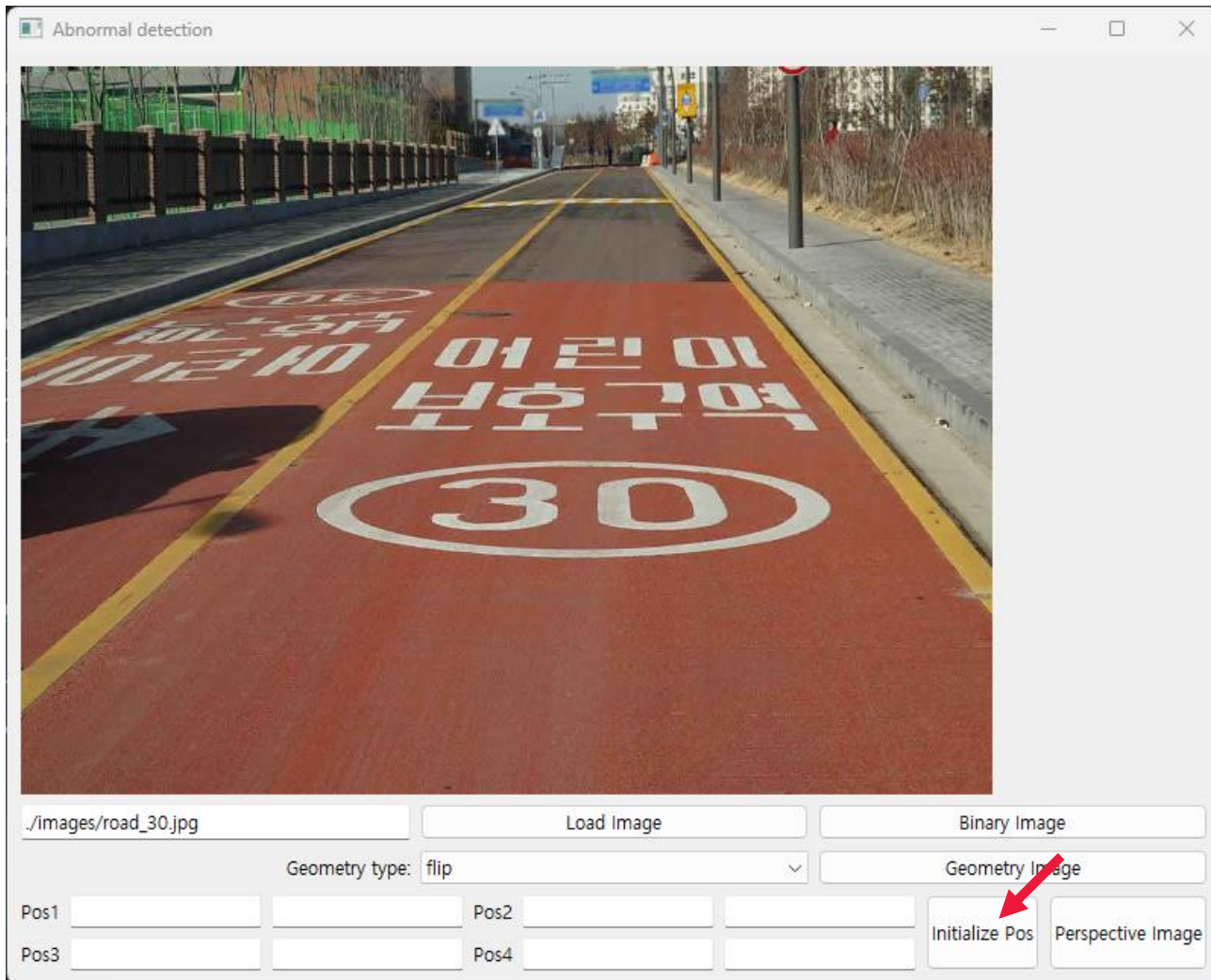
5) 네 번째 좌표에서 마우스 클릭 시 **노랑색 점**이 찍히고 Pos4에 자동으로 좌표 입력



# 마우스 클릭 이벤트 처리와 Perspective 기능 추가

8

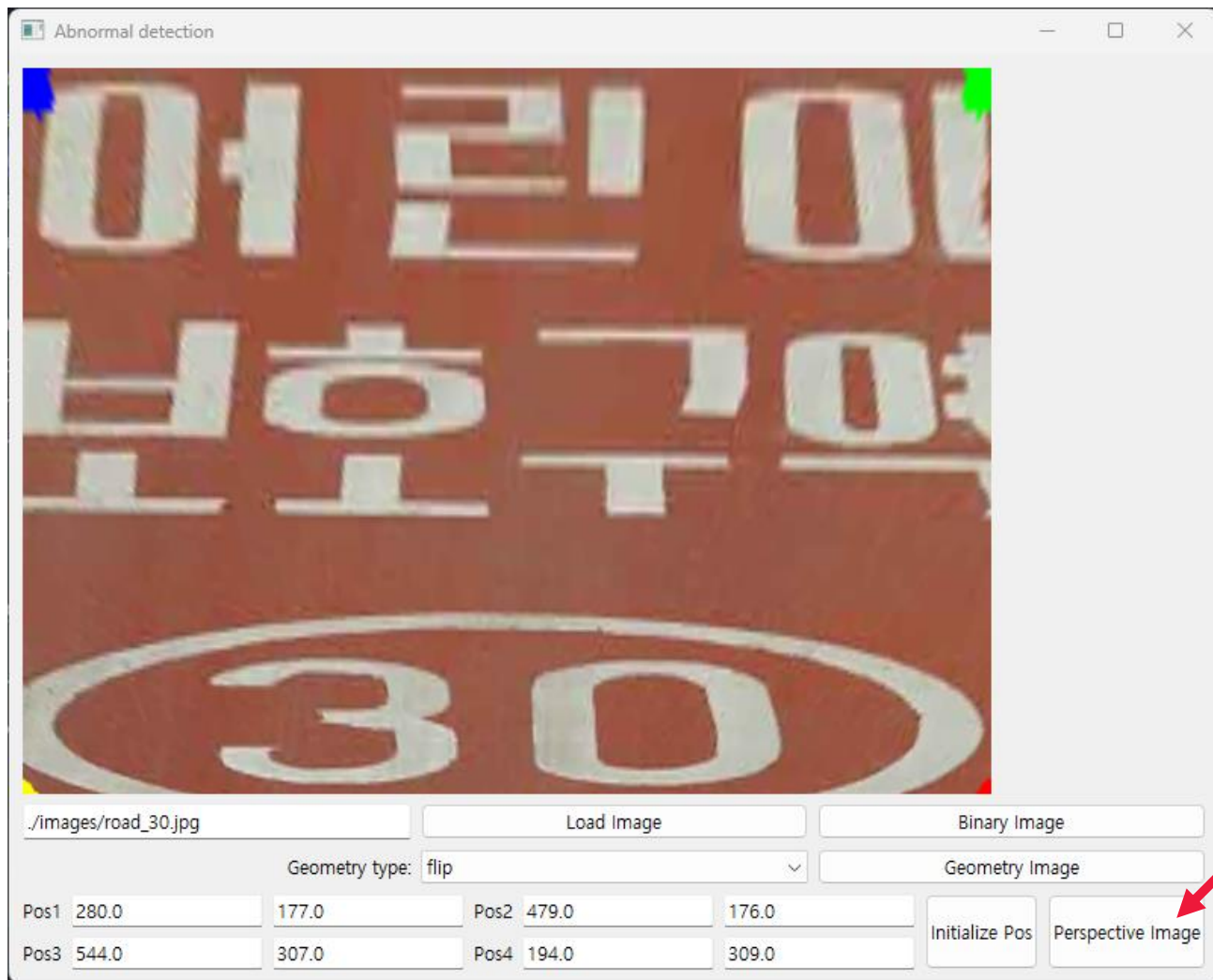
6) Initialize Pos 버튼을 클릭 시 4개의 점들을 초기화하고 원본 영상을 출력





# 마우스 클릭 이벤트 처리와 Perspective 기능 추가

7) 4개의 좌표가 입력된 상태에서 Perspective image 버튼을 실행 시 원근 변환이 실행된 영상을 출력



## ■ 구현 순서

- 실습3-2 기본 구성
- 4개의 QLabel 위젯 생성

```
self.label_pos1 = QLabel("Pos1")  
self.label_pos2 = QLabel("Pos2")  
self.label_pos3 = QLabel("Pos3")  
self.label_pos4 = QLabel("Pos4")
```

- 8개의 QLineEdit 위젯 생성

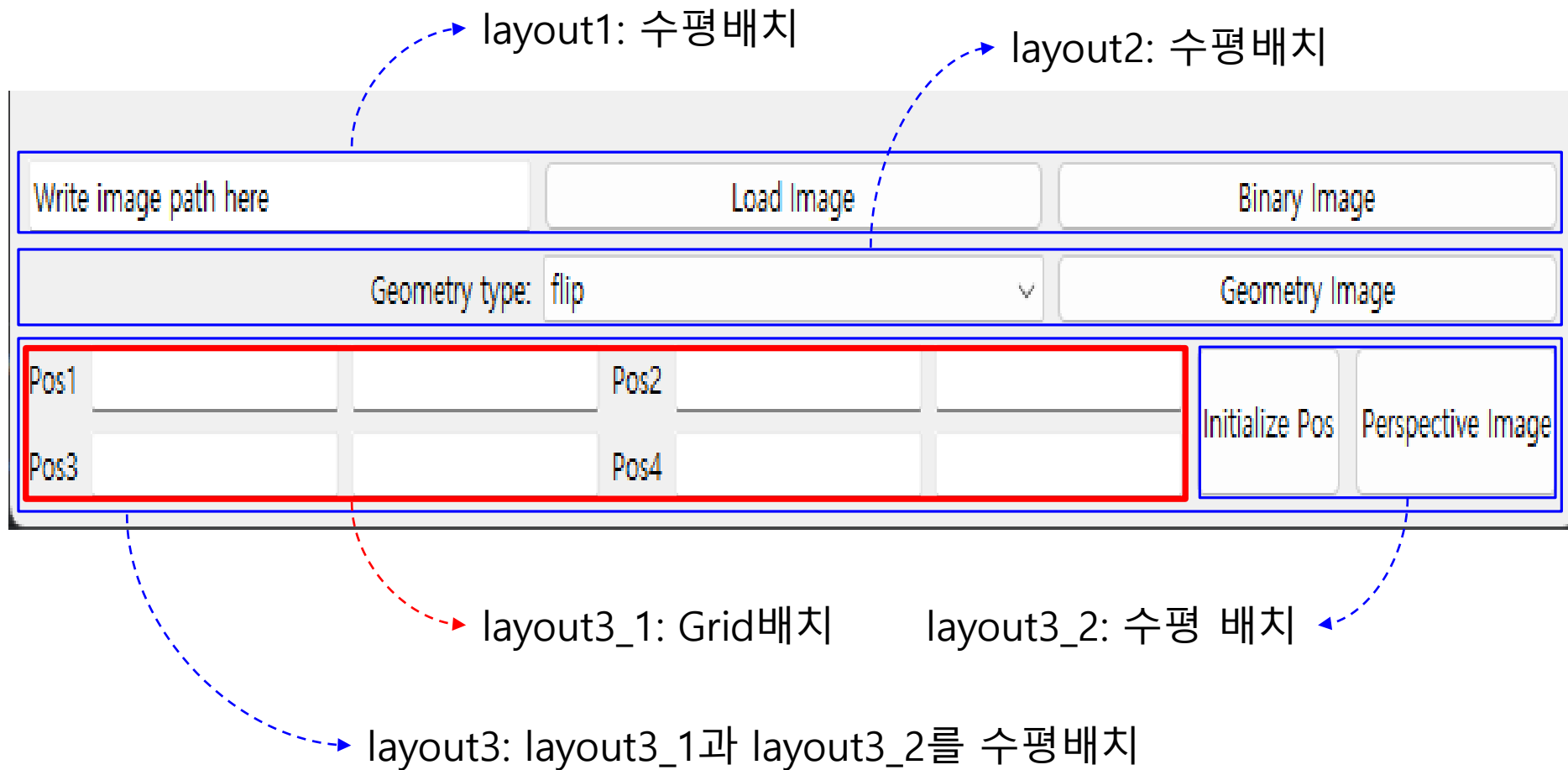
```
self.Ledit_x1 = QLineEdit()  
self.Ledit_y1 = QLineEdit()  
self.Ledit_x2 = QLineEdit()  
self.Ledit_y2 = QLineEdit()  
self.Ledit_x3 = QLineEdit()  
self.Ledit_y3 = QLineEdit()  
self.Ledit_x4 = QLineEdit()  
self.Ledit_y4 = QLineEdit()
```

- QGridLayout 객체 생성 및 위젯 추가

```
pos_grid = QGridLayout()  
pos_grid.addWidget(self.label_pos1,0,0)  
pos_grid.addWidget(self.Ledit_x1,0,1)  
pos_grid.addWidget(self.Ledit_y1,0,2)  
pos_grid.addWidget(self.label_pos2,0,3)  
pos_grid.addWidget(self.Ledit_x2,0,4)  
pos_grid.addWidget(self.Ledit_y2,0,5)
```

## ■ 구현 순서

- Layout 배치



## ■ 구현 순서

- Layout 배치
  - QLabel, layout1, layout2, layout3 순서로 수직 배치





## ■ 구현 순서

- update\_image 함수 구현
- Window 클래스에 영상 정보가 담긴 변수를 화면에 출력시키는 함수를 작성

```
def update_image(self, img):  
    # Creating and scaling QImage  
    if len(img.shape) < 3:  
        h, w = img.shape  
        ch = 1  
        img_format = QImage.Format_Grayscale8  
    else:  
        h, w, ch = img.shape  
        img_format = QImage.Format_RGB888  
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  
  
    img = QImage(img.data, w, h, ch * w, img_format)  
    scaled_img = img.scaled(640, 480, Qt.KeepAspectRatio)  
  
    self.label_image.setPixmap(QPixmap.fromImage(scaled_img))
```

## ■ 구현 순서

- 파일에서 영상 Load 후 원본영상과 임시 처리 파일을 분리하여 관리
  - `self.m_main_img` : 원본 파일을 저장
  - `self.m_proc_img` : 영상 처리 결과들이 저장 될 임시 처리 파일로 사용될 복사본을 저장

```
def load_img_func(self):  
    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))  
    self.m_proc_img = self.m_main_img.copy()  
    self.update_image(self.m_proc_img)
```

## ■ 구현 순서

- 마우스 클릭 이벤트 처리
  - Window 클래스에서 "**mousePressEvent**" method 정의

커서의 x좌표 값 (float형)

커서의 y좌표 값 (float형)

x와 y의 데이터 타입을  
int형으로 변환


빈칸에 마우스 클릭 카  
운팅을 할 수 있는 코드  
를 작성해 보세요

*tip) 인스턴스 변수가 필요합니다.*

```
def mousePressEvent(self, event):  
    x = event.position().x() - self.label_image.x()  
    y = event.position().y() - self.label_image.y()  
    x, y = int(x), int(y)  
    if  == 0:  
        self.Ledit_x1.setText(f"{x}")  
        self.Ledit_y1.setText(f"{y}")  
          
        cv2.circle(self.m_proc_img, (x, y), 5, (255, 0, 0), -1)  
  
    elif  == 1:  
        self.Ledit_x2.setText(f"{x}")  
        self.Ledit_y2.setText(f"{y}")  
          
        cv2.circle(self.m_proc_img, (x, y), 5, (0, 255, 0), -1)  
    self.update_image(self.m_proc_img)
```

## ■ 구현 순서

- perspective image 버튼의 slot 함수 정의
- 빈칸에는 입력된 x, y 좌표들의 정보를 불러오는 코드를 작성해 보세요.

```
def perspective_image(self):
    rows, cols = self.m_proc_img.shape[:2] # channel 여부 무시
    x1 = 
    y1 =
    x2 =
    y2 =
    x3 =
    y3 =
    x4 =
    y4 =
    # pts1 좌표 표시
    pts1 = np.float32([[x1, y1], [x2, y2], [x3, y3], [x4, y4]])
    pts2 = np.float32([[0, 0], [cols-1, 0], [cols-1, rows-1], [0, rows-1]])
    Mat1 = cv2.getPerspectiveTransform(pts1, pts2)
    r_image = cv2.warpPerspective(self.m_proc_img, Mat1, (cols, rows))

    self.update_image(r_image)
```