```cpp
#include <cstdio>
#include <rclcpp/rclcpp.hpp>

class TestNode : public rclcpp::Node
{
public:
    TestNode() : Node("test_node")
    {
        this->declare_parameter("test_param", 123);
        int test_param = this->get_parameter("test_param").as_int();
        RCLCPP_INFO(this->get_logger(), "Test parameter: %d", test_param);

        parameters_client_ = std::make_shared<rclcpp::AsyncParametersClient>(this);
        while (!parameters_client_->wait_for_service(std::chrono::seconds(1))) {
            RCLCPP_INFO(this->get_logger(), "Waiting for parameters server...");
        }

        auto parameters_callback = [this](const rcl_interfaces::msg::ParameterEvent::SharedPtr event) {
            for(auto & changed_parameter : event->changed_parameters) {
                RCLCPP_INFO(this->get_logger(), "Parameter event received: %s, %d", changed_parameter.name.c_str(), changed_parameter.value.integer_value);
            }
        };

        subscription_ = parameters_client_->on_parameter_event(parameters_callback);
    }
    rclcpp::AsyncParametersClient::SharedPtr parameters_client_;
    rclcpp::Subscription<rcl_interfaces::msg::ParameterEvent>::SharedPtr subscription_;
};

int main(int argc, char ** argv)
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<TestNode>());
  rclcpp::shutdown();
  return 0;
}
```

```cpp
#include <cstdio>
#include <rclcpp/rclcpp.hpp>

class TestNode : public rclcpp::Node
{
public:
    TestNode() : Node("test_node")
    {
        this->declare_parameter("test_param", 123);
        int test_param = this->get_parameter("test_param").as_int();
        RCLCPP_INFO(this->get_logger(), "Test parameter: %d", test_param);

        parameters_client_ = std::make_shared<rclcpp::AsyncParametersClient>(this);
        while (!parameters_client_->wait_for_service(std::chrono::seconds(1)))
        {
            RCLCPP_INFO(this->get_logger(), "Waiting for parameters server...");
        }


        subscription_ = parameters_client_->on_parameter_event(std::bind(&TestNode::parameters_callback, this, std::placeholders::_1));
    }
    rclcpp::AsyncParametersClient::SharedPtr parameters_client_;
    rclcpp::Subscription<rcl_interfaces::msg::ParameterEvent>::SharedPtr subscription_;

    void parameters_callback(const rcl_interfaces::msg::ParameterEvent::SharedPtr event)
    {
        for (auto &changed_parameter : event->changed_parameters)
        {
            RCLCPP_INFO(this->get_logger(), "Parameter event received: %s, %d", changed_parameter.name.c_str(), changed_parameter.value.integer_value);
        }
    };
};

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<TestNode>());
    rclcpp::shutdown();
    return 0;
}
```

첫 번째 코드는 단순히 `parameters_callback`을 직접 전달하여 이벤트에 대한 콜백을 등록하는 반면 **두번째 코드**는 `std::bind`를 사용하여 `parameters_callback` 함수를 바인딩한다.이는 `TestNode` 클래스의 멤버 함수를 호출할 때 현재 객체(`this`)와 함께 인수를 전달할 수 있도록 한다. 이 방식은 멤버 함수가 특정 객체의 컨텍스트에서 호출되도록 보장할 수 있다.

즉 두 코드의 가장 큰 차이는 `on_parameter_event`에 콜백을 전달하는 방법이며, 두 번째 코드가 `std::bind`를 사용함으로써 클래스의 멤버 함수가 올바르게 호출될 수 있도록 보장한다.

<declare_parameter>
: 노드의 파라미터를 정의 및 초기값 설정. 이 파라미터는 노드 내에서 사용되며, 외부에서 접근하거나 수정가능.
declare_parameter("파라미터 이름", 파라미터 초기값);

<get_parameter>
: ROS 2에서 선언된 파라미터의 값을 가져오는 데 사용되는 함수. 특정 파라미터의 현재 값을 노드에서 쉽게 조회할 수 있게 해줌.
get_parameter(“파라미터 이름”)

*as_type(): 가져온 파라미터 값을 원하는 데이터 타입으로 변환하는 메서드
ex) as_int(), as_double()

<get_logger>
: ROS 2에서 노드의 로깅 기능을 사용할 수 있도록 해주는 메서드이다. 이 함수를 사용하면 노드의 로거 인스턴스를 가져와 다양한 로그 레벨의 메시지를 기록할 수 있다.

*로깅 기능 : 시스템의 상태와 동작을 기록하고 모니터링할 수 있는 방법을 제공하는 기능

<RCLCPP_INFO>
: ROS 2에서 정보를 로깅하기 위해 사용하는 매크로다. 이 매크로를 사용하면 특정 로거 인스턴스를 통해 INFO 레벨의 로그 메시지를 쉽게 기록할 수 있다.
RCLCPP_INFO(logger, "로그 메시지 형식", 변수1, 변수2, ...);

여기서 logger은 일반적으로 this->get_logger()로 가져온다.

<make_shared>
: 객체를 동적으로 생성하고 이를 std::shared_ptr로 감싸주는 기능을 제공. 이 메서드는 메모리 관리와 성능 면에서 유리한 점이 많아, 특히 ROS 2와 같은 C++ 프로젝트에서 자주 사용된다.

// std::make_shared를 사용하여 MyClass의 인스턴스를 생성하고 shared_ptr로 관리
Ex) std::shared_ptr<MyClass> my_object =
std::make_shared<MyClass>(10);

*인스턴스 : 특정 클래스로부터 생성된 구체적인 객체를 의미함
*SharedPt : C++에서 객체의 소유권을 공유하는 스마트 포인터의 한 형태로, std::shared_ptr의 별칭이다. shared_ptr는 객체의 수명을 자동으로 관리하며, 여러 개의 포인터가 동일한 객체를 가리킬 수 있도록 허용합니다.

<parameters_callback>
: 콜백 함수는 노드의 파라미터가 변경될 때 호출되며, 주로 동적으로 파라미터 값을 업데이트하거나 로그를 기록하는 데 사용된다.

<changed_parameters>
: rcl_interfaces::msg::ParameterEvent 메시지의 멤버 변수 중 하나로, 파라미터 이벤트가 발생했을 때 변경된 파라미터들의 목록을 포함한다.

<on_parameter_event>
: on_parameter_event 는 ROS 2 에서 파라미터 이벤트를 구독하기 위한 메서드로, 파라미터가 변경될 때 호출될 콜백 함수를 등록하는 데 사용된다.
Subscription<rcl_interfaces::msg::ParameterEvent>::SharedPtr on_parameter_event(std::function< void(const rcl_interfaces::msg::ParameterEvent::SharedPtr)> callback);
여기서 callback 이 파라미터 이벤트가 발생했을때 호출할 콜백 함수이다.

<rclcpp::init>
: rclcpp::init 는 ROS 2 C++ 클라이언트 라이브러리에서 제공하는 초기화 함수로, ROS 2 의 노드와 관련된 모든 기능을 사용하기 위해 반드시 호출해야 하는 함수이다. 이 함수는 ROS 2 환경을 설정하고, 내부적으로 필요한 리소스를 초기화한다.

<rclcpp::spin>
: rclcpp::spin 은 ROS 2 C++ 클라이언트 라이브러리에서 제공하는 함수로, 노드의 이벤트 루프를 실행하고, 노드에서 등록된 콜백을 처리하는 역할을 한다. 이 함수는 ROS 2 노드가 동작하는 동안 계속 실행되며, 비동기적으로 발생하는 이벤트(예: 메시지 수신, 타이머, 서비스 호출 등)를 처리한다.

<rclcpp::shutdown>
: rclcpp::shutdown 는 ROS 2 C++ 클라이언트 라이브러리에서 제공하는 함수로, ROS 2 시스템을 종료하는 데 사용된다. 이 함수는 노드가 더 이상 필요하지 않을 때 호출되며, ROS 2 와 관련된 리소스를 정리하고 메모리를 해제한다.

<rclcppAsyncParametersClient>

: AsyncParametersClient는 ROS 2 C++ 클라이언트 라이브러리(rclcpp)에서 제공하는 클래스로, 비동기 방식으로 파라미터 서버와 상호작용할 수 있도록 해주는 기능을 한다. 이 클래스는 파라미터를 설정하거나 조회하는 작업을 비동기적으로 처리할 수 있게 도와주며, 노드가 다른 작업을 수행하는 동안에도 파라미터 서버와의 통신을 가능하게 한다.

- **set_parameters**
- **get_parameters**
- **wait_for_service**
- **on_parameter_event**

<cv::resize>

: resize는 주로 이미지 처리 또는 데이터 구조의 크기를 조정하는 데 사용되는 함수 또는 메서드입니다

cv::resize(src, dst, dsize)

src 는 원본 이미지, dst 는 크기 조정된 이미지를 저장할 대상 이미지, dsize 는 새로운 크기이며, dsize 는 cv::Size(width, height) 형식으로 지정한다.

<**getStructuringElement>**

:getStructuringElement 는 OpenCV 에서 형태학적 변환(morphological transformation) 에 사용되는 함수로, 주어진 형태와 크기의 구조 요소(structuring element)를 생성합니다. 이 함수는 다양한 형태의 마스크를 생성하여, 이미지의 형태학적 작업(예: 침식, 팽창, 열기, 닫기 등)에 사용됩니다.

cv::Mat getStructuringElement(int shape, cv::Size size, cv::Point anchor = cv::Point(-1, -1));

- shape: 구조 요소의 형태를 정의하는 값 (예: cv::MORPH_RECT, cv::MORPH_ELLIPSE, cv::MORPH_CROSS 등).
- **size**: 구조 요소의 크기를 정의하는 cv::Size 객체.
- **anchor**: 구조 요소의 중심점으로 사용할 cv::Point 객체. 기본값은 cv::Point(-1, -1)로 설정되어 있어 자동으로 중앙으로 설정된다.

<GaussianBlur>

: `GaussianBlur` 는 OpenCV 라이브러리에서 제공하는 함수로, 이미지에 가우시안 블러를 적용하는 데 사용됩니다. 가우시안 블러는 이미지의 노이즈를 줄이고, 세부 정보를 부드럽게 하여 더 자연스러운 시각 효과를 생성하는 데 유용하다. 이 필터는 픽셀 주변의 가중치를 기반으로 하여 적용되며, 중간 픽셀의 값은 주변 픽셀의 값에 따라 조정된다.

```
cv::GaussianBlur(src, dst, ksize, sigmaX, sigmaY, borderType);
```

- **src**: 입력 이미지 (블러를 적용할 이미지).
- **dst**: 블러가 적용된 결과 이미지.
- **ksize**: 커널 크기 (홀수 정수로 지정, 예: `cv::Size(5, 5)`).
- **sigmaX**: X 방향의 가우시안 표준 편차. 기본값은 0 이며, 이 경우 커널 크기에 따라 자동으로 결정됨.
- **sigmaY**: Y 방향의 가우시안 표준 편차. 기본값은 0 이며, 이 경우 `sigmaX` 와 동일한 값이 사용된다.
- **borderType**: 테두리 픽셀을 처리하는 방법을 지정합니다. 일반적으로 `cv::BORDER_DEFAULT` 를 사용한다.
- 마지막 매개변수인 표준편차가 클수록 블러의 강도가 강해지고 이미지가 더 흐려진다.

<cvtColor>

: `cvtColor` 는 OpenCV 라이브러리에서 제공하는 함수로, 이미지의 색상 공간을 변환하는 데 사용됩니다. 이 함수는 다양한 색상 공간 변환을 지원하며, 예를 들어 BGR 에서 RGB 로 변환하거나, 그레이스케일로 변환하는 등의 작업을 수행할 수 있다.

```
cv::cvtColor(src, dst, code);
```

- **src**: 입력 이미지 (변환할 이미지).
- **dst**: 변환된 결과 이미지를 저장할 변수.
- **code**: 변환할 색상 공간을 지정하는 코드. OpenCV 에서 정의된 상수 값을 사용한다. 예를 들어:
- `cv::COLOR_BGR2GRAY`: BGR 이미지를 그레이스케일로 변환
- `cv::COLOR_BGR2RGB`: BGR 이미지를 RGB 로 변환
- `cv::COLOR_RGB2HSV`: RGB 이미지를 HSV 로 변환

<inRange>

: `inRange` 는 OpenCV 에서 제공하는 함수로, 주어진 색상 범위 내에 있는 픽셀을 찾는 데 사용된다. 이 함수는 일반적으로 이미지에서 특정 색상을 필터링하고 마스크 이미지를 생성하는 데 활용된다. 예를 들어, 특정 색상 (예: 빨간색, 초록색 등)의 객체를 찾거나 배경을 제거하는 등의 작업에 유용하다.

```
cv::inRange(src, lowerb, upperb, dst);
```

- **`src`**: 입력 이미지 (색상 필터링을 적용할 이미지).
- **`lowerb`**: 필터링할 색상의 하한 값. 이 값은 OpenCV 에서 사용하는 색상 공간에 따라 다릅니다.
- **`upperb`**: 필터링할 색상의 상한 값.
- **`dst`**: 결과 마스크 이미지를 저장할 변수.

<erode>

`erode` 는 `OpenCV` 에서 제공하는 형태학적 변환함수 중 하나로, 이미지를 침식시키는 데 사용된다. 이 함수는 이미지의 객체 경계를 다듬거나, 잡음을 줄이는 데 효과적이다.

```
cv::erode(src, dst, kernel, anchor, iterations, borderType, borderValue);
```

- **`src`**: 입력 이미지 (침식할 이미지).
- **`dst`**: 침식이 적용된 결과 이미지.
- **`kernel`**: 구조 요소 (침식에 사용될 마스크).
- **`anchor`**: 구조 요소의 기준점. 기본값은 `Point(-1, -1)`으로 설정되어 있어 구조 요소의 중앙을 기준으로 한다.
- **`iterations`**: 침식 반복 횟수. 기본값은 1 입니다.
- **`borderType`**: 테두리 픽셀을 처리하는 방법. 일반적으로 `cv::BORDER_DEFAULT` 를 사용한다.
- **`borderValue`**: 테두리 픽셀의 값입니다. 기본값은 0 입니다.

<dilate>

: dilate는 OpenCV에서 제공하는 형태학적 변환함수 중 하나로, 이미지를 팽창시키는 데 사용된다. 이 함수는 이미지의 객체를 확장하거나 두드러지게 만들어, 형태를 강조하는 데 효과적이다.

```
cv::dilate(src, dst, kernel, anchor, iterations, borderType, borderValue);
```

- **src**: 입력 이미지 (팽창할 이미지).
- **dst**: 팽창이 적용된 결과 이미지.
- **kernel**: 구조 요소 (팽창에 사용될 마스크).
- **anchor**: 구조 요소의 기준점. 기본값은 Point(-1, -1)으로 설정되어 있어 구조 요소의 중앙을 기준으로 한다.
- **iterations**: 팽창 반복 횟수. 기본값은 1 입니다.
- **borderType**: 테두리 픽셀을 처리하는 방법. 일반적으로 cv::BORDER_DEFAULT 를 사용한다.
- **borderValue**: 테두리 픽셀의 값이다. 기본값은 0 이다.

*ROI : "Region of Interest"의 약자로, 이미지 또는 비디오에서 분석하거나 처리하고자 하는 특정 영역을 나타낸다.

<Mat::zeros>

Mat::zeros는 OpenCV에서 제공하는 함수로, 지정된 크기와 타입의 모든 요소가 0으로 초기화된 행렬(이미지)을 생성하는 데 사용된다. 이 함수는 주로 빈 이미지나 마스크를 생성할 때 사용된다.

```
cv::Mat Mat::zeros(int rows, int cols, int type);
```

- **rows**: 행렬의 행 수.
- **cols**: 행렬의 열 수.
- **type**: 행렬의 데이터 타입. OpenCV 에서 정의된 상수로 설정한다.

<fillPoly>

: fillPoly는 OpenCV에서 제공하는 함수로, 주어진 다각형을 채우는 데 사용된다. 이 함수는 다각형의 내부를 지정된 색상으로 채워주며, 이미지에서 다양한 도형을 시각적으로 강조하거나 특정 영역을 마킹하는 데 유용하다.

```
cv::fillPoly(img, pts, color, lineType, shift);
```

매개변수

- **img**: 입력 이미지. 다각형이 채워질 대상 이미지이다.
- **pts**: 다각형의 점들이 저장된 배열. 각 다각형의 점들은 `std::vector<cv::Point>` 형태로 정의되며, 이 점들의 배열은 `std::vector<std::vector<cv::Point>>` 형태로 전달되어야 한다.
- **color**: 다각형 내부를 채울 색상. 일반적으로 `cv::Scalar` 객체로 지정한다.
- **lineType**: 선의 유형을 지정한다. 기본값은 `cv::LINE_8` 이다.
- **shift**: 점의 좌표를 보정하는 데 사용할 비트 수를 지정한다. 기본값은 0 이다.

<bitwise_and>

: bitwise_and는 OpenCV에서 제공하는 함수로, 두 이미지를 픽셀 단위로 비트 논리 AND 연산을 수행하는 데 사용된다. 이 함수는 주로 마스크와 원본 이미지를 결합하여 특정 영역만 남기고 나머지를 제거하는 데 유용하다.

```
cv::bitwise_and(src1, src2, dst, mask);
```

- **src1**: 첫 번째 입력 이미지.
- **src2**: 두 번째 입력 이미지.
- **dst**: 결과 이미지를 저장할 변수.
- **mask** (선택적): 마스크 이미지로, 이 마스크가 적용된 부분만 연산을 수행한다.

<Canny>

: Canny 는 OpenCV 에서 제공하는 엣지 감지 알고리즘 중 하나로, 이미지에서 객체의 경계를 감지하는 데 사용된다. 이 알고리즘은 엣지 검출 기능을 제공하며, 다양한 컴퓨터 비전 작업에 필수적으로 사용됩니다.

```
cv::Canny(image, edges, threshold1, threshold2, apertureSize, L2gradient);
```

- **image**: 입력 이미지 (그레이스케일 이미지).
- **edges**: 엣지 감지 결과가 저장될 이미지.
- **threshold1**: 하한 임계값.
- **threshold2**: 상한 임계값.
- **apertureSize**: Sobel 커널의 크기 (3, 5, 7 중 하나).
- **L2gradient**: 가우시안 블러링 후 경계의 강도를 계산할 때 L2 노름을 사용할지 여부를 지정하는 플래그.

<**Vec4i**>

: OpenCV 에서 4 개의 정수 값을 저장하는 벡터 클래스이다. 보통 선의 시작점과 끝점 또는 다른 형태의 정보를 표현하는 데 사용된다.

<HoughLinesP>

: HoughLinesP 는 OpenCV 에서 제공하는 함수로, 확률적 허프 변환을 사용하여 이미지에서 직선을 감지하는 데 사용됩니다. 이 방법은 일반적인 허프 변환보다 계산 비용이 낮고, 더 적은 메모리를 사용하며, 더 정확하게 직선을 찾아낼 수 있는 장점이 있습니다.

```
cv::HoughLinesP(image, lines, rho, theta, threshold, minLineLength, maxLineGap);
```

- **image**: 입력 이미지 (엣지 감지 후의 이진 이미지).
- **lines**: 감지된 직선을 저장할 벡터.
- **rho**: 거리 해상도 (픽셀 단위).
- **theta**: 각도 해상도 (라디안 단위).
- **threshold**: 선이 감지되기 위해 필요한 최소 투표 수.
- **minLineLength**: 감지할 직선의 최소 길이 (픽셀 단위).
- **maxLineGap**: 직선 간의 최대 간격 (픽셀 단위).

<cvRound>

:cvRound는 OpenCV에서 제공하는 함수로, 실수 값을 가장 가까운 정수로 반올림하는 데 사용된다. 이 함수는 주로 이미지 처리나 컴퓨터 비전 작업에서 픽셀 좌표를 정수로 변환할 때 유용하다.

```
int cvRound(double value);
```

- **value**: 반올림할 실수 값입니다.

<line>

: line 함수는 OpenCV에서 이미지를 처리할 때 특정한 두 점을 연결하는 직선을 그리는 데 사용되는 함수이다. 이 함수는 주로 이미지의 특정 요소를 강조하거나, 그래픽 오버레이를 추가할 때 유용하게 사용된다.

- **img**: 직선을 그릴 대상 이미지.
- **pt1**: 직선의 시작점 (x, y 좌표).
- **pt2**: 직선의 끝점 (x, y 좌표).
- **color**: 직선의 색상. Scalar 객체를 사용하여 RGB 또는 BGR 색상을 지정한다.
- **thickness** (선택적): 선의 두께. 기본값은 1 이다.
- **lineType** (선택적): 선의 유형. 기본값은 LINE_8 이다.
- **shift** (선택적): 점의 좌표를 보정하는 데 사용할 비트 수이다.

<connectedComponentsWithStats>

:connectedComponentsWithStats는 OpenCV에서 제공하는 함수로, 이진 이미지에서 연결된 구성 요소를 찾고, 각 구성 요소에 대한 통계 정보를 반환합니다. 이 함수는 객체 인식, 이미지 분석 및 형태학적 작업에서 매우 유용하게 사용됩니다.

```
int connectedComponentsWithStats(InputArray image, OutputArray labels,
OutputArray stats, OutputArray centroids, int connectivity = 8, int ltype =
CV_32S);
```

매개변수

- **`image`**: 입력 이진 이미지.
- **`labels`**: 각 픽셀의 구성 요소 레이블을 저장할 출력 배열.
- **`stats`**: 각 구성 요소의 통계 정보를 저장할 배열. 행은 구성 요소를 나타내고, 각 열은 다음 정보를 포함한다:
  - `x` 좌표 (최소 x 좌표)
  - `y` 좌표 (최소 y 좌표)
  - `width` (구성 요소의 폭)
  - `height` (구성 요소의 높이)
  - `area` (구성 요소의 픽셀 수)
- **`centroids`**: 각 구성 요소의 중심점을 저장할 배열.
- **`connectivity`**: 연결 방식 (4 또는 8). 기본값은 8 이다.
- **`ltype`**: 레이블 이미지의 데이터 타입. 기본값은 CV_32S 이다.