ROS2_day4

1. ROS2 파라미터 서버

파라미터 서버는 ros2에서 각 노드가 전역 or 로컬의 파라미터 값을 설정하고 실시간으로 변경할 수 있게 해주는 메커니즘. 노드들이 특정 파라미터에 접근하고 변경 사항을 수신하도록 하는 것이 목적.

- parameter_node
  (기본적인 파라미터 변경 값을 logger로 확인하고 싶은 경우, 다른 노드에 안 보내는 경우)

```
this->declare_parameter<int>("hue_upper", 100);

void parameters_callback(const rcl_interfaces::msg::Parameter
{
    for(auto &changed_parameter : event -> changed_parameters
    {
        RCLCPP_INFO(this->get_logger(), "파라미터 이벤트 수신 : %
    }
}
```

declare_parameter : 파라미터 이름, 기본값 설정

파라미터 콜백 함수 : 실시간으로 파라미터 변경 사항 반영을 위한 콜백함수, logger 출력

하지만 과제2
(parameter 설정 값을 processing 이미지 처리에 반영 되어야 하는 과제)
에서는 파라미터 콜백함수를 processing 노드에 넣어주어 processing 노드에서 실시간으로 파라미터 값을 반영함

⇒set_parameter : 파라미터 값을 동적으로 변경

⇒parameter_events : 직접 Sub하지 않으면 따로 노드와 노드 사이에 토픽 같은 추가적인 통신이 필요하게 되고, ros2의 파라미터 서버, 파라미터 이벤트 메커니즘을 활용하지 못함, 과제2에서 이미지 처리에 적합한것 같아 사용

- ros2 파라미터 접근 방식

  →parameter_events : 파라미터 변경 사항을 실시간으로 감지하여 자동으로 반영
  →parameter_client : 특정 시점에 다른 노드의 파라미터를 명시적으로 요청하거나, 설정 가능,
  클라이언트 - 서버 통신 방식으로 동작하며, 파라미터 서버가 다른 노드로 존재할 때도 접근 가능

  →get_parameter : parameter_client를 사용하지 않고도 파라미터 값을 같은 한 노드 내에서 가져오거나 설정할 수 있으나, 다른 노드의 파라미터에는 접근할 수 없다.

2.OPENCV 함수

1. HSV
   색 공간은 색을 표현하는 효과적인 방법이다. Hue는 색상, Saturation은 색의 채도, Value는 밝기를 나타냄. OpenCV에서는 cv::inRange() 함수를 사용하여 특정 HSV 범위 내의 색상을 필터링 가능

```
cv::Mat hsv_image;
cv::cvtColor(input_image, hsv_image, cv::COLOR_BGR2HSV);
cv::Mat mask;
cv::inRange(hsv_image, cv::Scalar(hue_lower, saturation_lower
            cv::Scalar(hue_upper, saturation_upper, value_upp
```

- cvtColor : 이미지의 색 공간을 변환하는 함수
- cv::COLOR_BGR2HSV : bgr 에서 hsv 로 변환
- inrange : 이미지 내에서 특정 범위의 색상 값을 선택하고 필터링할 때 사용
- mask : cv::Mat 타입 변수라고 생각하면 편함(이미지 변수), 결과 저장할 변수
- scalar : 여러 채널을 가진 색상 or 값을 저장하는데 사용 되는 opencv 클래스

2. 이미지 사이즈(resize)
   이미지의 크기를 변경할 때는 cv::resize() 함수를 사용함
   이 함수는 이미지의 너비와 높이를 새로운 값으로 설정

```
cv::Mat resized_image;
cv::resize(input_image, resized_image, cv::Size(new_width, ne
```

- resize : 이미지의 크기를 조절하는 함수(가로, 세로)

3. erode, dilate
   침식과 팽창은 이미지 처리에서 형태학적 변환으로 사용
   cv::erode() 함수는 객체의 경계를 줄이고, cv::dilate() 함수는 객체를 확장하는 데 사용
   주로 노이즈 제거 및 구조적 요소를 강화하는 데 사용

```
cv::Mat eroded_image, dilated_image;
cv::erode(input_image, eroded_image, cv::Mat());
cv::dilate(input_image, dilated_image, cv::Mat());
```

4. Canny 엣지 감지
   캐니 엣지 감지는 이미지에서 엣지를 찾는 데 사용
   cv::Canny() 함수는 두 개의 임계값을 사용하여 강한 엣지와 약한 엣지를 구별

```
cv::Mat edges;
cv::Canny(input_image, edges, canny_threshold1, canny_thresho
```

threshold1 : 하단 임계값

threshold2 : 상단 임계값

하단 임계값은 edge 주변도 edge일 확률이 높다는 점을 이용하여 threshold2에 의해 판단된 edge와 인접한 부분을 edge로 판단할 때 사용하는 임계값

하단 임계값을 너무 낮게 설정하면 원치 않은 edge까지 검출할 것이기에 적절한 값 요구
상단 임계값은 실질적으로 edge를 판단하는 임계값

즉 상단 임계값 이상의 픽셀은 강한 엣지로 간주하고, 낮음 임계값 사이의 픽셀은 강한 엣지에 연결된 경우에만 엣지로 판단

1번 이미지 : 원본 2번 이미지 : 상단값 : value1 3번 이미지: 상단값 value2(> value1)

일반적으로 threshold1 : threshold2 = 1 : 2 or 1: 3

5. ROI
   ROI는 이미지의 특정 부분을 선택하여 작업할 때 사용
   주로 관심 있는 객체만 분석하기 위해 사용
   이미지에서 ROI를 설정하고 이를 통해 다른 처리를 수행 가능

```
cv::Rect roi(x, y, width, height);
cv::Mat roi_image = input_image(roi);
```

```
std::vector<cv::Point> points = {cv::Point(x1, y1), cv::Point
cv::Mat mask = cv::Mat::zeros(input_image.size(), CV_8UC1);
cv::fillPoly(mask, points, cv::Scalar(255));
cv::bitwise_and(input_image, input_image, output_image, mask)
```

- fillpoly : roi 지정해준 마스크에 마스킹 할때 유용함(선을 그려줌)

```
cv::fillPoly(image, points, color);
예시
std::vector<std::vector<cv::Point>> points = { { cv::Point(10
cv::fillPoly(image, points, cv::Scalar(255, 255, 255));
```

- cv::bitwise
  - cv::bitwise_and : 두 이미지의 대응되는 픽셀들에 대해 AND 연산을 수행
  - cv::bitwise_or : OR 연산, 두 이미지 합성힐 때 사용
  - cv::bitwise_xor : 두 이지미가 다른 부분만 남기고 같으면 제거
  - cv::bitwise_not : 모든 비트 반전, 검은색 흰색 반전시킬 때 사용

6. 블러 및 필터링(가우시안 블러)
   가우시안 블러는 이미지의 노이즈를 줄이고, 부드럽게 만드는 데 사용
   cv::GaussianBlur() 함수는 이미지에 가우시안 필터를 적용

```
cv::Mat blurred_image;
cv::GaussianBlur(input_image, blurred_image,
                 cv::Size(kernel_size, kernel_size),
                 sigmaX);
```

작동 원리 : 가우시안 블러는 중심 픽셀의 값에 가중치를 더하여 주변 픽셀들과 평균을 내는 방식
가우시안 함수에 따라 가까운 픽셀은 큰 가중치를, 먼 픽셀은 작은 가중치를 부여하여 계산
ksize : 커널 크기, 홀수로 지정해줘야함

sigmax : x방향의 표준편차 값, 값이 클수록 블러링이 감해짐

7. 허브 변환(Hough Line)
   허프 변환은 이미지에서 선형 구조를 감지하는 데 사용
   cv::HoughLines() 함수는 직선 형태를 감지

```
std::vector<cv::Vec2f> lines;
cv::HoughLines(edges, lines, 1, CV_PI/180, threshold);
for (size_t i = 0; i < lines.size(); i++) {
    float rho = lines[i][0], theta = lines[i][1];
}
```

작동 원리 :

허프 변환은 이미지의 (x, y) 좌표 공간에서 점을 찾고, 이를 극좌표(ρ, θ)로 변환하여 직선을 검출

- 직선 방정식

$\rho = x \cdot cos(\theta) + y \cdot sin(\theta) \rho = x \cdot \cos(\theta) + y \cdot \sin(\theta)$

$\rho = x \cdot cos(\theta) + y \cdot sin(\theta)$
모든 에지 포인트에 대해 θ 값을 변화시키며 ρ를 계산하고, 누적 배열에서 최대값이 있는 위치가 직선으로 판단

```
cv::HoughLines(input_image, lines, rho, theta, threshold);
예시
std::vector<cv::Vec2f> lines;
cv::HoughLines(edge_image, lines, 1, CV_PI / 180, 150);
```

lines : 검출된 직선 정보가 저장될 벡터
rho : 직선의 거리 해상도
theta : 직선의 각도 해상도
threshold : 직선을 구성하는 최소 포인트 수

8. 레이블링
   이미지 내 객체를 식별하고 레이블을 부여하는 과정입니다.
   cv::connectedComponents() 함수를 사용해 연결된 구성 요소를 찾고, 각 구성 요소에 레이블을 부여

```
cv::Mat labels;
int n_components = cv::connectedComponents(binary_image, labe
```

목적 :
객체의 갯수를 세기 위해, 객체의 위치나 크기를 계산하기 위해, 객체마다 서로 다른 색상이나 레이블을 부여해 구분하기 위해

```
cv::connectedComponents(input_image, labels, connectivity, lt
예시
cv::Mat labels;
int n_labels = cv::connectedComponents(binary_image, labels,
```

labels: 결과 이미지 저장

connectivity : 4-connected 도는 8-connected 연결성을 지정

ltype : 레이블의 데이터 타입