ROBIT_ROS2_OpenCV

- 로빛 19기 예비단원 김근형

Grayscale

그레이스케일(grayscale)은 컬러 이미지를 흑백 톤으로 변환한 이미지 형식이다.

예시 코드

```
cv::Mat color_image;          // 컬러 이미지
cv::Mat grayscale_image;      // 그레이스케일 이미지

// 그레이스케일 변환
cv::cvtColor(color_image, grayscale_image, cv::COLOR_BGR2GRAY);
```

Binary

Binary 이미지는 픽셀이 **두 가지 값만 가지는 이미지**이다.

Binary 이미지를 만드는 과정은 **이진화(Binarization)**라고 한다. 주로 **그레이스케일 이미지**에서 특정 임계값을 기준으로 픽셀 값을 흑백으로 나누게 되는데, 이때 기준이 되는 값을 **임계값(threshold)**이라고 한다.

예시 코드

```
cv::Mat gray_image;  // 입력 그레이스케일 이미지
cv::Mat binary_image;

// 이진화 적용
cv::threshold(gray_image, binary_image, 127, 255, cv::THRESH_BINARY);
```

- `127` 은 임계값으로, 그레이스케일 이미지에서 이 값보다 큰 값은 **255**(흰색)으로 설정되고, 작은 값은 **0**(검정색)으로 설정된다.
- `cv::THRESH_BINARY`는 기본적인 이진화 모드이다.

HSV

HSV는 색을 나타내는 또 다른 색 공간으로, 색상(Hue), 채도(Saturation), 명도(Value 또는 Brightness)를 기준으로 색을 표현하는 방법이다.

HSV의 구성 요소

1. Hue (색상): 색의 종류를 나타낸다. 0도부터 360도까지의 원형 값으로 나타내며, 주로 **0~179** 범위로 스케일링하여 사용한다.
    - 예를 들어:
        - 빨간색: 0도
        - 노란색: 약 30도
        - 초록색: 약 60도

    - 파란색: 약 120도
2. Saturation (채도): 색의 선명함이나 색의 강도를 나타냅니다. **0**은 회색(색 없음)을 의미하고, **높을수록 더 진한 색**을 나타낸다.
    - 0%는 무채색(회색)
    - 100%는 가장 선명한 색상
3. Value (명도): 색의 밝기를 나타낸다. 값이 낮을수록 어두운 색이며, **0**은 완전히 검정, **높을수록 더 밝은 색**을 나타낸다.

HSV 변환 예시 코드

```
cv::Mat bgr_image = cv::imread("input.jpg");  // BGR 이미지 로드
cv::Mat hsv_image;

// BGR에서 HSV로 변환
cv::cvtColor(bgr_image, hsv_image, cv::COLOR_BGR2HSV);
```

HSV를 사용한 색상 필터링 예시 코드

```
// HSV 이미지 생성
cv::Mat mask;
cv::inRange(hsv_image, cv::Scalar(35, 100, 100), cv::Scalar(85, 255, 255), ma
```

`cv::inRange` 함수는 지정한 HSV 범위(`35-85`의 색상 범위)를 사용하여 마스크를 생성한다. 이렇게 하면 특정 색상만 남긴 이진(Binary) 이미지가 만들어진다.

가우시안 블러

⇒ OpenCV Gaussian Blur는 이미지의 노이즈를 줄이고 부드럽게 만드는 필터링 기술이다.

- **Gaussian Blur의 원리**
    1. **가우시안 함수** : Gaussian Blur는 **가우시안 함수**를 사용하여 이미지의 각 픽셀을 주변 픽셀의 가중 평균으로 대체한다.
    2. **커널**: Gaussian Blur는 주로 **커널(kernel)**이라고 불리는 행렬을 사용하여 적용된다. 커널은 필터의 크기와 모양을 결정하며, 이미지의 각 픽셀에 이 커널을 적용하여 새로운 값을 계산한다.

    ⇒ 커널의 크기를 홀수여야한다.

    3. **블러링 과정**:
        - 각 픽셀에 대해 커널을 중앙에 두고, 커널의 값을 이미지의 해당 픽셀에 곱한 후, 그 결과를 합산하여 새로운 픽셀 값을 만든다.
        - 이 과정은 이미지의 모든 픽셀에 대해 반복된다.

예시 코드

```
// 가우시안 블러 적용
cv::Mat blurred_image;
cv::GaussianBlur(cv_ptr->image, blurred_image, cv::Size(5, 5), 0); // 커널 크기

// 블러링된 이미지를 다시 ROS 메시지로 변환
```

```
sensor_msgs::msg::Image output_msg;
cv_bridge::CvImage out_msg;
out_msg.header = msg->header; // 헤더 복사
out_msg.encoding = sensor_msgs::image_encodings::BGR8;
out_msg.image = blurred_image;
out_msg.toImageMsg(output_msg);

// 블러링된 이미지 발행
image_publisher_->publish(output_msg);
```

가우시안 블러 적용

- `cv_ptr->image`: ROS에서 OpenCV 형식으로 변환된 입력 이미지이다. `cv_ptr`은 일반적으로 `cv_bridge`를 통해 변환된 `sensor_msgs::Image` 형식의 포인터이다.
- `blurred_image`: 가우시안 블러가 적용된 결과 이미지를 저장할 변수로, `cv::Mat` 형식이다.
- `cv::GaussianBlur`: OpenCV에서 가우시안 블러를 적용하는 함수이다.

ROS 메시지로 변환

- `sensor_msgs::msg::Image output_msg`: ROS2에서 사용하는 `Image` 형식의 메시지 객체로, 최종적으로 이 객체가 발행된다.
- `cv_bridge::CvImage`: OpenCV의 `cv::Mat` 이미지와 ROS의 `sensor_msgs::msg::Image` 메시지 간의 변환을 돕는 `cv_bridge` 클래스이다.
- `out_msg.header = msg->header`: 원본 메시지의 헤더를 복사하여 타임스탬프와 프레임 ID 같은 메타데이터를 그대로 유지한다.
- `out_msg.encoding`: 이미지를 BGR8(8비트 3채널) 형식으로 인코딩한다. 이는 RGB 형식의 컬러 이미지를 의미한다.
- `out_msg.image = blurred_image`: `blurred_image`를 `out_msg`의 `image` 필드에 저장하여 OpenCV 이미지 데이터를 ROS 이미지 메시지에 포함한다.
- `out_msg.toImageMsg(output_msg)`: `CvImage` 객체를 ROS의 `sensor_msgs::msg::Image` 형식으로 변환하여 `output_msg`에 저장한다.

침식 erode

커널(kernel)이라는 작은 구조 요소(structuring element)를 사용하여 이미지를 변형한다. 커널은 일반적으로 작은 정사각형 형태로, 각 픽셀을 기준으로 주변 픽셀과의 연산을 수행한다.

1. **커널 설정**: 커널은 3x3, 5×5 등의 크기로 설정되며, 커널의 중심이 현재 처리 중인 픽셀 위치에 놓인다.
2. **연산 적용**: 커널이 덮고 있는 픽셀 영역에서 가장 작은 값을 가져와 중심 픽셀에 할당한다.
3. **결과**: 픽셀 값이 줄어들어 밝은 영역이 줄어들며, 윤곽선이 깎이는 효과가 생긴다.

예시 코드

```
cv::Mat input_image;          // 입력 이미지
cv::Mat eroded_image;         // 침식 결과 이미지
cv::Mat kernel = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(3, 3)); /
```

```
// 침식 연산 적용
cv::erode(input_image, eroded_image, kernel);
```

팽창 dilate

이미지의 각 픽셀을 중심으로 커널(kernel)을 놓고, 커널이 겹치는 영역 중에서 가장 높은 값을 해당 픽셀 위치에 할당한다. 이 과정은 흰색 영역이 더 넓어지게 만들며, 객체의 가장자리가 바깥쪽으로 팽창되는 효과를 준다.

- **커널(kernel)**: 팽창 연산의 모양과 크기를 결정하는 구조 요소이다. 주로 정사각형, 원형 또는 십자형 모양이 사용된다. 커널의 크기는 홀수로 지정하며, 예를 들어 `3x3`, `5x5` 크기 등이 자주 사용된다.
- **픽셀 값**: 커널이 덮는 영역 중 가장 큰 값이 현재 위치에 반영되므로, 밝은 값이 어두운 값 위로 확장된다.

예시 코드

```
cv::Mat input_image;         // 입력 이미지
cv::Mat dilated_image;       // 팽창 결과 이미지
cv::Mat kernel = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(3, 3)); /

// 팽창 연산 적용
cv::dilate(input_image, dilated_image, kernel);
```

Roi 관심영역 추출

ROI는 이미지에서 특정한 관심 영역을 추출하거나 집중해서 처리하고자 할 때 사용된다.

예시 코드

```
cv::Mat image = cv::imread("input.jpg");  // 이미지 로드
cv::Rect roi(50, 50, 200, 200);  // 관심영역 설정 (x, y, width, height)
cv::Mat roi_image = image(roi);  // ROI 추출
```

이미지에서 `(50, 50)` 좌표에서 시작하여 가로 200, 세로 200 크기의 영역을 `roi_image`에 추출한다.

Trapeziod 관심영역 마스크

사다리꼴 모양의 관심영역 마스크는 주로 **차선 인식**과 같은 도로 객체 검출에 사용된다. 사다리꼴은 도로를 나타내는 데 적합한 형태로, 도로가 멀어질수록 좁아지는 시야를 반영하기 때문에 실감 있고 유용한 차선 인식 마스크 형태이다.

예시 코드

```
cv::Mat mask = cv::Mat::zeros(image.size(), image.type());  // 빈 마스크 생성

std::vector<cv::Point> points;  // 사다리꼴 형태의 좌표 설정
points.push_back(cv::Point(100, 300));
points.push_back(cv::Point(500, 300));
points.push_back(cv::Point(600, 500));
points.push_back(cv::Point(0, 500));

cv::fillConvexPoly(mask, points, cv::Scalar(255, 255, 255));  // 사다리꼴 형태로
```

```
cv::Mat roi_image;
cv::bitwise_and(image, mask, roi_image);  // 마스크와 원본 이미지를 결합해 관심영역 
```

사다리꼴 영역만 남겨 `roi_image`에 저장하게 된다.

이 코드는 이미지에서 특정 형태(사다리꼴)의 관심 영역을 추출하기 위해 사용된다. 먼저 빈 마스크를 만들고, 사다리꼴 형태를 정의한 후, 그 형태로 마스크를 채운 다음, 원본 이미지와 마스크를 결합하여 관심 영역만을 추출하는 과정을 보여준다. 이를 통해 이미지 프로세싱 작업에서 필요한 부분만 집중적으로 처리할 수 있다.

canny

캐니 에지 검출은 이미지의 경계를 추출하는 데 사용하는 알고리즘이다. 경계는 이미지에서 밝기 변화가 큰 부분으로, 객체의 모양을 나타내는 중요한 정보이다.

- 노이즈 제거: 가우시안 블러를 사용해 이미지의 노이즈를 줄입니다.
- 경계 기울기 계산: 밝기 변화가 큰 부분을 탐지합니다.
- 비최대 억제: 경계가 아닌 점들을 억제하여 얇고 선명한 경계만 남깁니다.
- 히스테리시스 임곗값: 두 개의 임곗값을 사용하여, 강한 에지와 약한 에지를 결정합니다.

예시 코드

```
cv::Mat edges;
cv::Canny(image, edges, 100, 200);  // 낮은 임곗값 100, 높은 임곗값 200
```

Houghline

허프 선 변환은 직선의 방정식을 이용하여 이미지에서 직선을 검출하는 알고리즘이다. 주로 경계가 뚜렷한 직선을 찾아내는 데 사용되며, 차선 인식 같은 작업에 유용하다.

예시 코드

```
std::vector<cv::Vec4i> lines;
cv::HoughLinesP(edges, lines, 1, CV_PI/180, 50, 50, 10);  // 이미지 내 직선 검출

for (size_t i = 0; i < lines.size(); i++) {
    cv::line(image, cv::Point(lines[i][0], lines[i][1]), cv::Point(lines[i][2
}
```

- 이미지에서 직선을 검출한다(`cv::HoughLinesP` 함수 사용).
- 검출된 직선들의 시작점과 끝점을 얻어 원본 이미지에 빨간색 직선을 그린다.

ladeling

레이블링은 이진 이미지에서 서로 연결된 객체들을 식별하고 각 객체에 고유의 레이블을 부여하는 과정이다. 레이블링을 통해 이미지에서 여러 객체들을 구분하고 각각의 특징(면적, 중심 좌표, 외곽선 등)을 분석할 수 있다. 객체 검출과 객체 추적 작업에 유용하게 사용된다.

```
cv::Mat labels, stats, centroids;
int n_labels = cv::connectedComponentsWithStats(binary_image, labels, stats,

for (int i = 1; i < n_labels; i++) {  // 레이블 0은 배경이므로 1부터 시작
```

```
    int x = stats.at<int>(i, cv::CC_STAT_LEFT);
    int y = stats.at<int>(i, cv::CC_STAT_TOP);
    int w = stats.at<int>(i, cv::CC_STAT_WIDTH);
    int h = stats.at<int>(i, cv::CC_STAT_HEIGHT);
    cv::rectangle(image, cv::Rect(x, y, w, h), cv::Scalar(255, 0, 0), 2);
}
```

이진화된 `binary_image`에서 연결된 객체들을 각각 레이블링하고, 각 객체에 대해 사각형을 그려준다.