# ROBOT-PROGRAMMING MID-TERM REPORT

**응용물리학과 2017103038 권인회**

## < Mission #1 >

### - Goal

연결된 카메라 화면을 Topic Subscriber로 받아 Mat 객체로 변환하는 패키지를 만들었다.

### - Timeline

*$ catkin_create_pkg ros_sample_image_transport std_msgs roscpp*

이후, 패키지 설정 파일과 빌드 설정 파일은 참고 링크에서 가져와 사용하였다.
(CMakeLists.txt의 find_package 항목에 cv_bridge, image_transport, sensor_msgs 등 추가, package.xml도 build_depend와 run_depend에 항목 추가하였다.)
노드로 사용할 소스 파일 역시 참고 링크에서 가져와 수정하였다.
(https://github.com/tzutalin/ros_sample_image_transport)

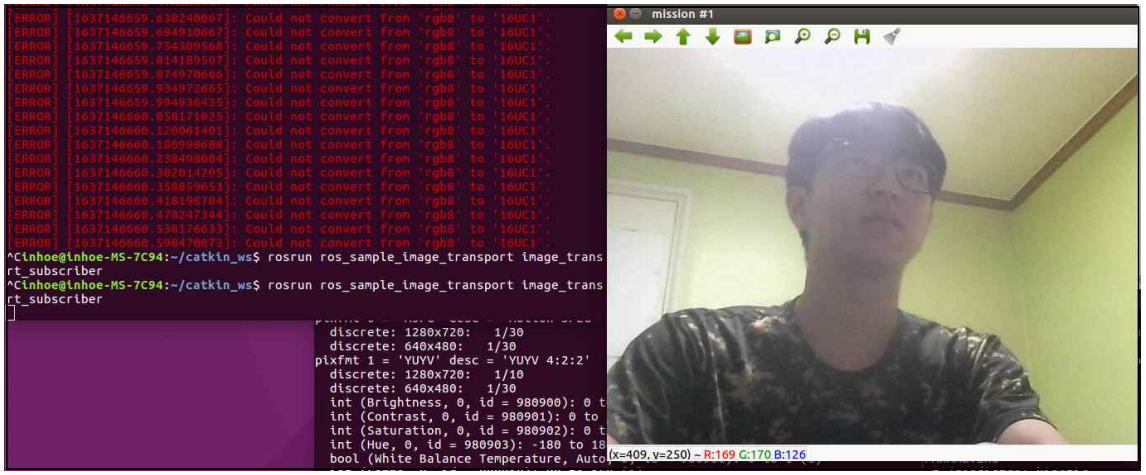
** image_transport_publisher.cpp 수정 사항 **
1. 우리가 사용할 topic은 image_raw기 때문에 TOPIC_NAME을 변경해주었다.
2. filepath 또한 우리는 필요로 하지 않기 때문에 제외해주었다.
3. 화면을 변환 후 전송하는 방식과 변환하지 않고 전송하는 방식이 있는데, 이 중에서 변환하지 않고 전송하는 방식을 사용하였다.

** image_transport_subscriber.cpp 수정 사항 **
1. 위와 마찬가지로 우리가 사용할 topic은 image_raw기 때문에 TOPIC_NAME을 변경해주었다.
2. Depth 관련된 정보는 필요하지 않으므로 관련 변수 및 함수를 주석 처리 하였다.
3. 창의 이름이 원하는대로 출력되도록 몇몇 cv 함수의 parameter를 변경해주었다.

이와 같은 수정 사항을 거친 뒤, 두 노드를 실행하자 정상적으로 mission #1의 영상을 얻을 수 있었다.

### - How to use

*$ rosrun ros_sample_image_transport image_transport_publisher*
*$ rosrun ros_sample_image_transport image_transport_subscriber*




< 그림 1 - mission #1 결과 사진 >

## < Mission #2 >

### - Goal

주어진 Optical Flow 알고리즘을 이용하여 연결된 카메라 화면에 적용하고 그에 따른 motion vector의 방향을 출력한다.

### - Timeline

*$ catkin_create_pkg visual_operator std_msgs roscpp*

이후, 패키지 설정 파일과 빌드 설정 파일은 Mission #1에서 가져와 사용하였다.
(Mission #1과 동일하게 추가된 내용이 발생하였다.)
노드로 사용할 소스 파일은 제공된 파일과 Mission #1에서 가져와 수정하였다.

** optical_flow_sample.cpp -> visual_operator_subscriber.cpp 수정 사항 **

1. publisher와 통신하기 위해 Mission #1의 코드를 남겨두고 제공된 코드를 전처리문을 포함하여 추가하였다.
2. 영상 테스트에 성공하였기 때문에 화면으로 사용할 객체를 영상 파일이 아니라 기기에 연결된 기본 카메라를 받도록 코드를 수정해주었다. (capture)

이 정도만 수정하여도 motion vector의 방향이 출력되는 정보를 얻을 수 있었다. 하지만, 이 방향이 어느 쪽인지 출력하는 단계에서 문제를 많이 겪었다.

** 해결 시도 **

1. 각 점에서 motion vector에 따라 선이 그어지는데 이 선들의 x축 성분의 값과 y축 성분의 값의 합을 저장하여 이 값을 토대로 가장 값이 큰 쪽으로 motion이 진행되었다고 결정하는 알고리즘을 구상하여 사용하였다.

2. 함께 제공되는 hsv 화면을 통해 여기에서 화면 속 motion이 위, 아래, 왼쪽, 오른쪽으로 향하는지에 따라 빨강, 초록, 파랑, 보라로 구분되어 나타난다는 것을 파악하였고 화면 전체적의 hue 값을 평균 내어 이 값을 통해 motion 방향을 결정하는 알고리즘을 구상하여 사용하였다.

** 문제점 **

1. 카메라 노이즈 문제로 인해 가만히 있어도 Optical Flow가 화면 전체에서 계속 나타났다. -> 평균값 필터, 가우시안 필터 등 노이즈를 줄여주는 알고리즘 등을 적용하려 하였으나 관련 알고리즘만 적용하면 카메라 연결이 끊어져버리는 현상이 발생하여 사용하지 못하였다.

2. 주요하지 않은 부분에서의 motion vector 값도 같이 계산이 되어버린다. -> 이 문제는 10픽셀 간격으로 각 점에서 motion 즉, m 값이 0.5가 넘지 않으면 계산을 하지 않도록 자체적으로 임계값을 설정해두었다. 따라서, motion이 심하지 않으면 그 부분에 대한 계산을 하지 않고, 선도 나타나지 않도록 하였다.

** 해결 **

- motion의 정도를 축적하는 변수를 재설정하고 인식되는 임계값을 아주 큰 수로 두었더니 노이즈와 상관없이 가장 큰 변화에 대해 인식하여 motion의 방향을 정상적으로 인지하였다.

```
test1 += m * cos(ang);
test2 += m * sin(ang);
```

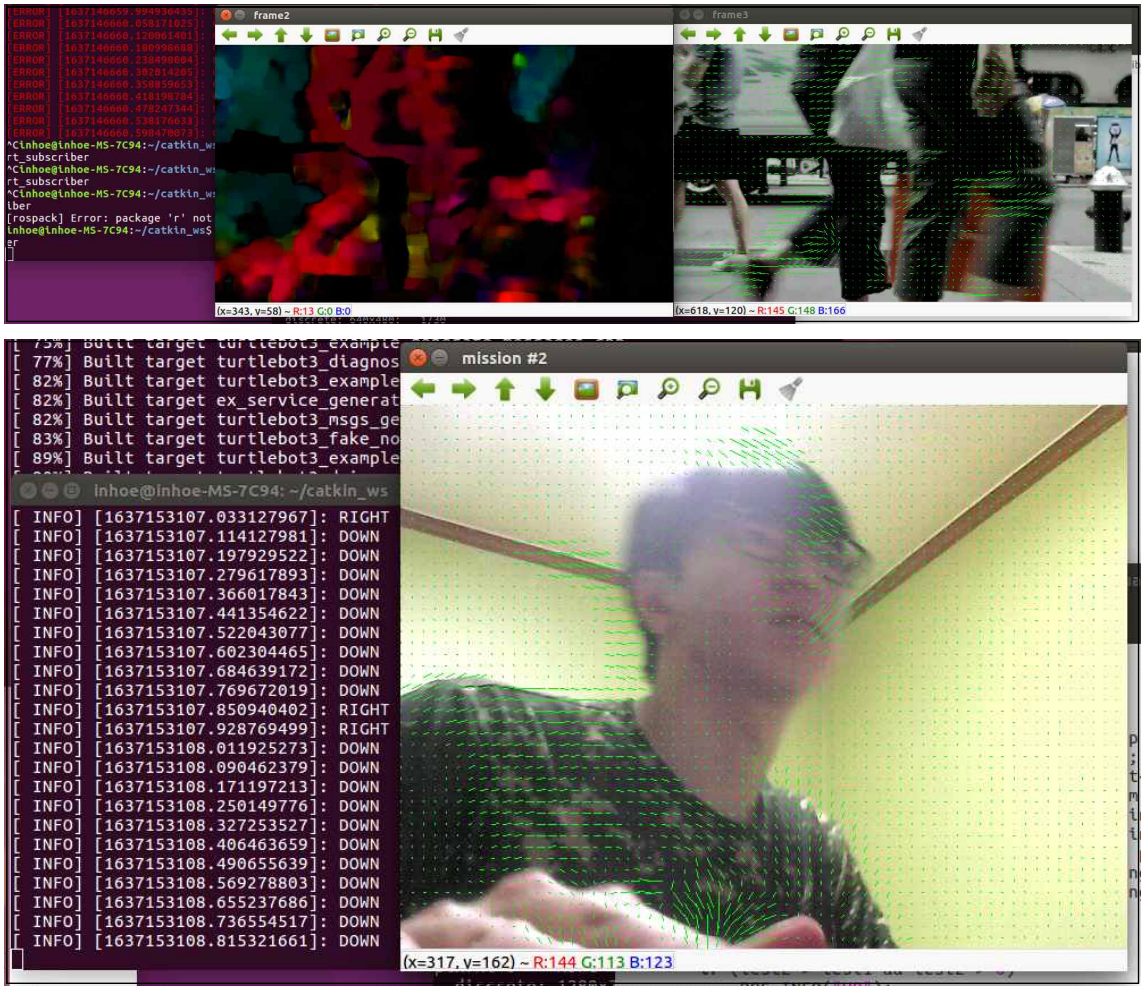
< 그림 2 - motion의 정도를 축적하는 변수 >

```
if (test2 > test1 && test2 > 3000)
    ROS_INFO("DOWN");
else if (test2 < test1 && test2 < -3000)
    ROS_INFO("UP");
else if (test1 > test2 && test1 > 3000)
    ROS_INFO("RIGHT");
else if (test1 < test2 && test1 < -3000)
    ROS_INFO("LEFT");
```

< 그림 3 - 변수를 분석하여 방향 판정 >

- **How to use**

*$ rosrun visual_operator visual_operator_subscriber*




< 그림 4 - Mission #2 결과 사진 >

## < Mission #3 >

- Goal

#2에서 출력되는 motion vector의 방향에 따라 turtlebot이 움직이도록 한다.

- Timeline

우선 CMakeLists.txt에 find_package 항목에 geometry_msgs를 추가하였다. 이후 위에서 사용한 visual_operator_subscriber 노드에 아래와 같은 코드들을 추가하여 cmd_vel topic으로 publish할 수 있도록 하였다. cmd_vel topic에 관한 정보는 rqt를 통해 linear.x, linear.y, linear.z, angular.x, angular.y, angular.z가 float64로 구성되어있는 것을 확인하였고 turtlebot3_teleop 노드를 통해 turtlebot을 움직일 때 사용하는 것은 linear.x와 angular.z인 것을 파악하였다. 이처럼 기존의 turtlebot package를 분석하고 참고문헌을 통해 다음과 같이 publish하는 코드를 만들 수 있었다.

```
ros::init(argc, argv, "visual_operator_publisher");
ros::NodeHandle nh;
ROS_INFO("move start");
ros::Publisher cmd_pub = nh.advertise<geometry_msgs::Twist>("/cmd_vel", 5);
ros::Rate r(5);

geometry_msgs::Twist go_forward;
go_forward.linear.x = 0.0;

geometry_msgs::Twist go_left;
go_left.angular.z = 0.0;
```

```
if (key == 'w'){
        target_linear_vel = target_linear_vel + LIN_VEL_STEP_SIZE;
        go_forward.linear.x += 0.01;
        cmd_pub.publish(go_forward);


else if (key == 'x'){
        target_linear_vel = target_linear_vel - LIN_VEL_STEP_SIZE;
        go_forward.linear.x -= 0.01;
        cmd_pub.publish(go_forward);


else if (key == 'a'){
        target_angular_vel = target_angular_vel + ANG_VEL_STEP_SIZE;
        go_left.angular.z += 0.05;
        cmd_pub.publish(go_left);


else if (key == 'd'){
        target_angular_vel = target_angular_vel - ANG_VEL_STEP_SIZE;
        go_left.angular.z -= 0.05;
        cmd_pub.publish(go_left);

else
    if (key == '\x03')
        break;
```

< 그림 5 - optical flow를 토대로 linear 또는 angular를 변화시킨다. >

- How to use

*$ rosrun visual_operator visual_operator_publisher*

- References

1. https://swimminglab.tistory.com/102
2. https://github.com/tzutalin/ros_sample_image_transport
3. https://docs.opencv.org/3.4.15/d4/dee/tutorial_optical_flow.html

- Postscript

해당 이름을 가진 패키지를 create하시고 압축 파일 안에 있는 ros_sample_image_transport 및 visual_operator 폴더를 *~/catkin_ws/src/* 경로에 넣어주신 뒤 빌드하시고 사용법에 따라 rosrun 해주시면 됩니다.

또한, 노드 파일 중 파일명 뒤에 '(clear)' 가 붙은 파일은 주석이나 필요없는 코드를 제외한 clean code 입니다! 구상한 과정을 보여드리기 위해 주석을 남겨둔 코드도 함께 첨부하였습니다.

모든 패키지 정상적으로 빌드되고 실행되었지만, 문제가 있다면 알려주시면 감사하겠습니다! 해당 과제 영상 파일도 함께 첨부하였습니다.