# 情報計測学基礎Ⅰ課題

T11707M 篠原伸之

レポート課題：オプティカルフローによる物体検出プログラムの開発とインストールマニュアルの制作

【インストールマニュアル】

1. cv_bridge パッケージの準備

① パッケージ保存先のフォルダの作成

mdir –p **report_dir1**/src

※太字部分の名称は自由に設定してよい.

② report_dir1/src まで移動し，新規作成するパッケージ名と継承するパッケージを指定する.

cd ~report_dir1/src

catkin_create_pkg **image_processing** sensor_msgs cv_bridge roscpp std_msgs image_transport

※太字部分の名称は自由に設定してよい.

③ image_processing の src まで移動し，ソースコードを作成する.

cd ~image_processing/src

vi optical_flow.cpp

p4 の【ソースコード】を参照すること.

④ image_processing の中にある CMakeList.txt に以下の 2 行を追加する.

add_executable( **optical_flow** src/optical_flow.cpp )

target_ling_libraries( **optical_flow** ${catkin_LIBRARIES} )

※太字部分の名称は自由に設定してよい.

⑤ report_dir1/src に移動し，catkin_init_workspace を実行する.

cd ~report_dir1/src

catkin_init_workspace

⑥ report_dir1 に移動し，catkin_make を実行する.

cd ~report_dir1

catkin_make

参考 URL :

http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages

2. USB カメラを使用する準備

① usb_cam パッケージのダウンロード

cd ~report_dir1/src

git clone https://github.com/bosch-ros-pkg/usb_cam.git

② usb_cam パッケージの make

cd ~report_dir1

catkin_make

参考 URL :

http://zumashi.blogspot.jp/2016/12/ros-kinetic-usb-cam.html

3. プログラムの実行

① USB カメラを PC に接続する.

② ターミナルを 3 つ用意する.

③ 各ターミナルで report_dir1 まで移動し，setup.bash を実行する.

cd ~report_dir1

source devel/setup.bash

④ それぞれのターミナルで以下のコマンドを実行する.

ターミナル 1：roscore

ターミナル 2：rosrun usb_cam usb_cam_node

ターミナル 3：rosrun image_processing optical_flow

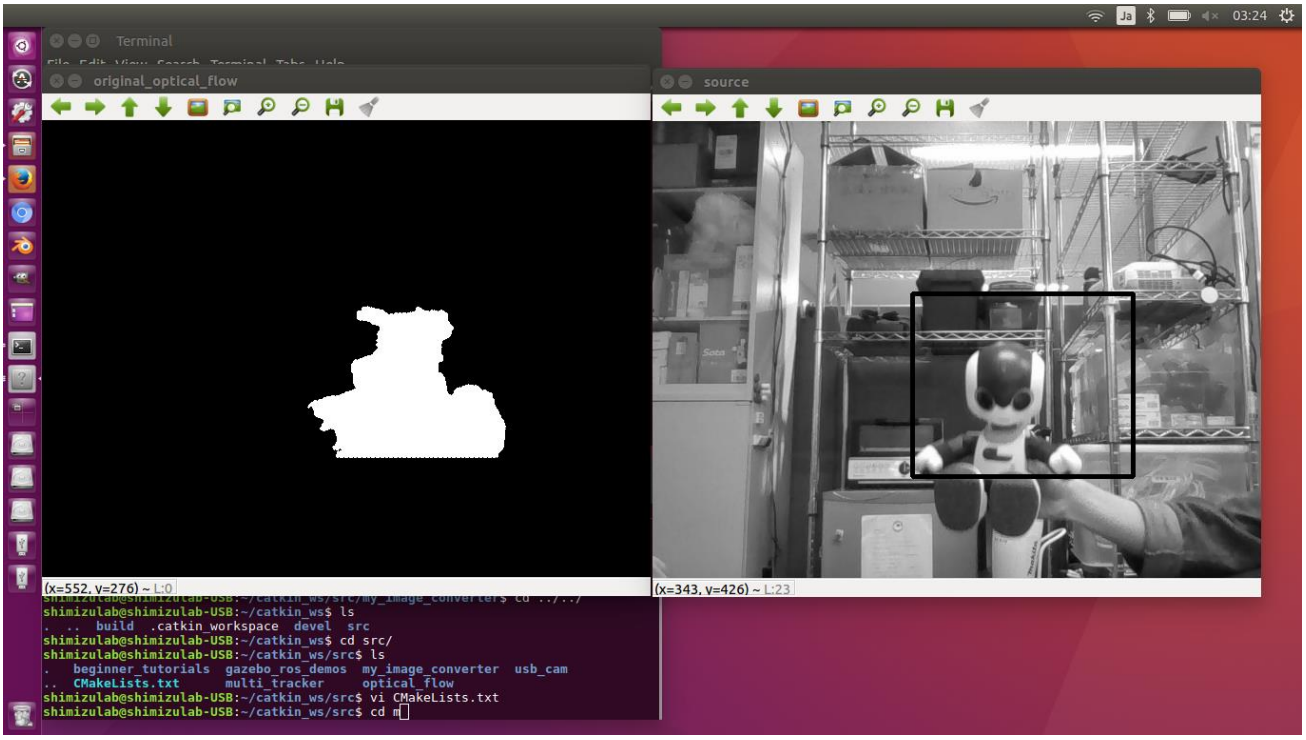


図 1．実行結果の例

source ウインドウ上の移動物体に対してバウンディングボックスが囲われる．
また，original_optical_flow ウインドウでは，強度 0 以上のオプティカルフローを持つ画素（動きのある画素）が白画素で示される．

【ソースコード】

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>

/* ---------------------------------------------------------------------- NS-code ---------------------------------------------------------------------- */
#include "string.h"
#include "stdio.h"
#include "stdlib.h"
#include "iostream"
#include "math.h"
#include "opencv/cv.h"
#include "opencv2/core/core.hpp"
#include "opencv2/video/tracking.hpp"
#include "opencv2/Labeling.h"

#define MAX_VAL 0
#define MIN_VAL 255
#define FG      255
#define BG      0

using namespace std;
using namespace cv;
int frame_count = 0;
/* ---------------------------------------------------------------------- NS-code ---------------------------------------------------------------------- */

static const std::string OPENCV_WINDOW = "Image window";

cv::Mat source_image;
cv::Mat prev_image;

class ImageConverter
{
  ros::NodeHandle nh_;
  image_transport::ImageTransport it_;
  image_transport::Subscriber image_sub_;
  image_transport::Publisher image_pub_;

public:
  ImageConverter()
    : it_(nh_)
  {
    image_sub_ = it_.subscribe("/usb_cam/image_raw", 1,
      &ImageConverter::imageCb, this);
    image_pub_ = it_.advertise("/image_converter/output_video", 1);

    cv::namedWindow(OPENCV_WINDOW);
  }

  ~ImageConverter()
  {
    cv::destroyWindow(OPENCV_WINDOW);
  }
```

```
void imageCb(const sensor_msgs::ImageConstPtr& msg)
{
  cv_bridge::CvImagePtr cv_ptr;
  try
  {
    cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
  }
  catch (cv_bridge::Exception& e)
  {
    ROS_ERROR("cv_bridge exception: %s", e.what());
    return;
  }

  /* ---------------------------------------------------------------------- NS-code ---------------------------------------------------------------------- */

  cv::cvtColor( cv_ptr->image, source_image, CV_BGR2GRAY );
  image_pub_.publish( cv_bridge::CvImage( std_msgs::Header(), "bgr8", source_image).toImageMsg());

  // Setting the image for display
  Mat org_flow = cv::Mat::ones(cv::Size(source_image.cols,source_image.rows),CV_8UC1)*BG;

  if( frame_count == 0 ) prev_image = source_image.clone();

  if( frame_count > 0 )
  {
    vector<cv::Point2f> prev_pts;
    vector<cv::Point2f> next_pts;

    // Define of the number of optical flow (To detect a object:image size, To display a flow:one-tens image size)
    Size flowSize(source_image.rows,source_image.cols);

    Point2f center = cv::Point(source_image.cols/2., source_image.rows/2.);
    for(int i=0; i<flowSize.width; ++i)
    {
      for(int j=0; j<flowSize.width; ++j)
      {
        Point2f p(i*float(source_image.cols)/(flowSize.width-1), j*float(source_image.rows)/(flowSize.height-1));
        prev_pts.push_back((p-center)*0.95f+center);
      }
    }

    // Calculation of optical flow
    Mat flow;
    calcOpticalFlowFarneback(prev_image, source_image, flow, 0.8, 10, 15, 3, 5, 1.1, 0);

    // Drawing the optical flow
    std::vector<cv::Point2f>::const_iterator p = prev_pts.begin();
    for(; p!=prev_pts.end(); ++p)
    {
      const cv::Point2f& fxy = flow.at<cv::Point2f>(p->y, p->x);
      double val_flow = flow.at<double>(p->y,p->x);

      // Check the value of threshold and radius
      if( val_flow > 10.0) cv::circle(org_flow, cv::Point(p->x,p->y), 1,cv::Scalar(FG), -1, 8, 0);
      // Check the value of gain ( *p+fxy*" " )
      //if( val_flow > 10.0) cv::line(disp, *p, *p+fxy*8, cv::Scalar(FG), 1);
      }
```

```
        // Saving the source_image into prev_image
        prev_image = source_image.clone();

        // Morphology calculation
        cv::Mat element(3, 3, CV_8U,cv::Scalar::all(255));
        cv::Mat dilate_flow = org_flow.clone();
        cv::erode(dilate_flow, dilate_flow, cv::Mat(), cv::Point(-1,-1), 10); // Check the number of erodion
        cv::dilate(dilate_flow, dilate_flow, cv::Mat(), cv::Point(-1,-1), 30); // Check the number of dilation

        // Labeling
        LabelingBS labeling;
        unsigned char *src = new unsigned char[source_image.cols*source_image.rows];
        short *result = new short[source_image.cols*source_image.rows];
        int l=0;
        for(int j=0 ; j<source_image.rows ; j++)
        {
           for(int i=0 ; i<source_image.cols ; i++)
           {
               src[l] = dilate_flow.at<unsigned char>(j,i);
               l++;
           }
        }
        labeling.Exec(src, result, source_image.cols, source_image.rows, true, 10);
        int nlabel = labeling.GetNumOfResultRegions();

        // Calculation of Labeling result
        RegionInfoBS *ri;
        for(int k=0 ; k<nlabel ; k++)
        {
           ri = labeling.GetResultRegionInfo(k);
           int minx,miny;
           int maxx,maxy;
           ri->GetMin(minx, miny);
           ri->GetMax(maxx, maxy);
           cv::rectangle(source_image, cv::Point(minx,miny), cv::Point(maxx,maxy), cv::Scalar(0), 3, 4);
        }

        // Display of images
        imshow("original_optical_flow", org_flow);
        imshow("dilate_optical_flow", dilate_flow);
        imshow("source", source_image);
        int c = waitKey(1);
      }

     // Output modified video stream
     image_pub_.publish(cv_ptr->toImageMsg());

     frame_count++;
     /* ---------------------------------------------------------------------- NS-code ---------------------------------------------------------------------- */
   }
}

// Main
int main(int argc, char** argv)
{
   ros::init(argc, argv, "image_converter");
   ImageConverter ic;
   ros::spin();
```

```
    return 0;
}
```