

# 情報計測学基礎 1 レポート

中京大学大学院 工学研究科 機械システム工学専攻  
T11708M 柴田浩史

## 目次

|     |                |             |
|-----|----------------|-------------|
| I   | パッケージの導入       | ．．．．． P2-P3 |
| II  | USB カメラの導入     | ．．．．． P4    |
| III | オブティカルフロー概要    | ．．．．． P4-P7 |
| IV  | オブティカルフロー 動作結果 | ．．．．． P7-P9 |

### 【はじめに】

本授業では物体追跡プログラムの基盤となるオプティカルフローのプログラムを作製した。このレポートではパッケージ導入、USB カメラへの接続、オプティカルフロー概要といった流れで行ったことを説明する。

### 【I パッケージ導入】

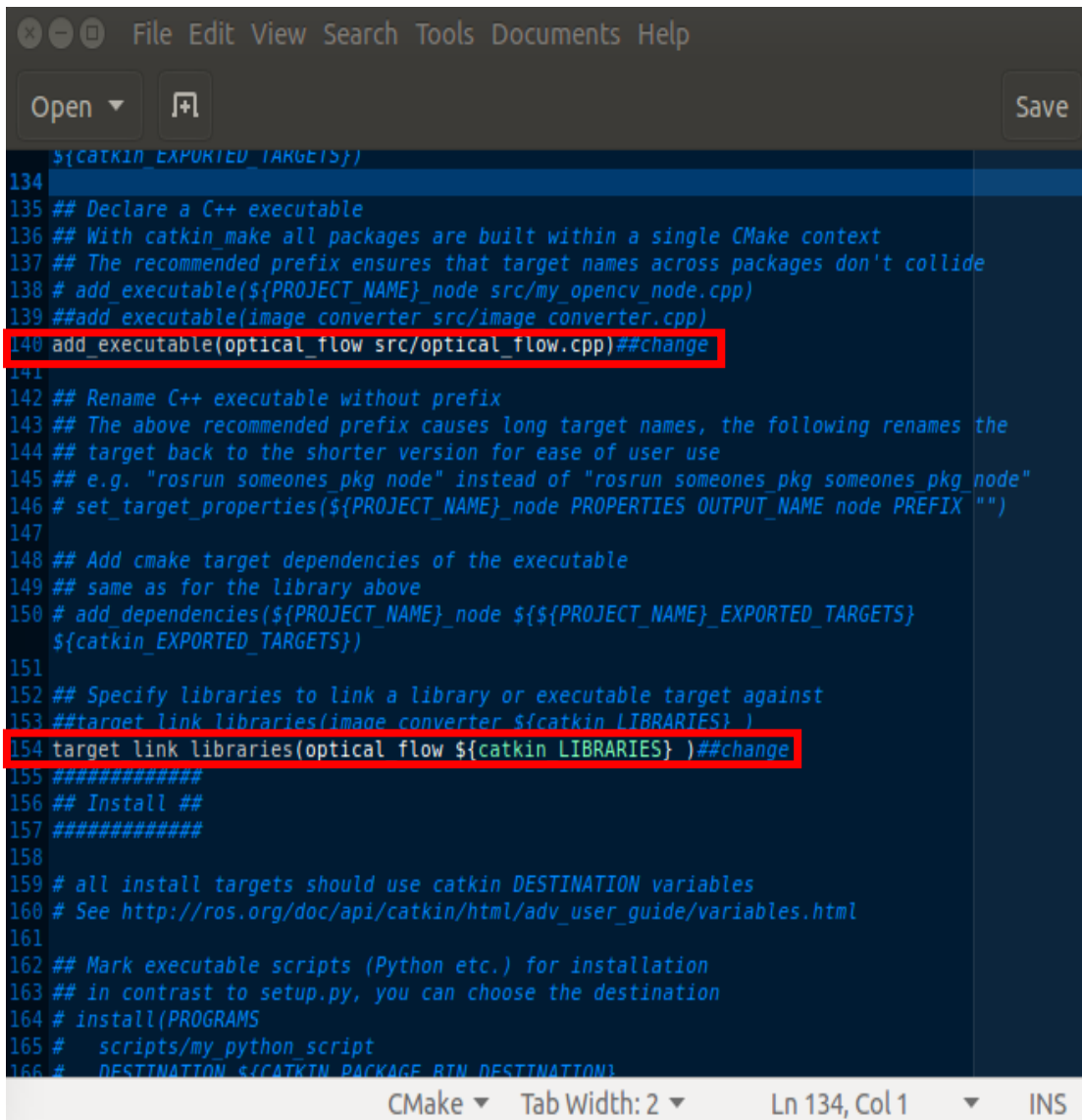
まず、下記のコマンドをターミナルに順に入力することで、ワークスペースを作製する。

- ① `$ mkdir -p my_robot/src` //ディレクトリの作製
- ② `$ cd my_robo/src` //移動
- ③ `$ catkin_init_workspace` // ワークスペース (src フォルダにパッケージがなく、ただ CMakeList.txt のリンクがあるだけ) の作製

次に作製したワークスペースでパッケージを作製し、プログラムを修正する。

- ④ `$ catkin_create_pkg my_opencv sensor_msgs cv_bridge roscpp std_msgs image_transport` // パッケージの作製
- ⑤ `$ cd my_opencv/src` //移動
- ⑥ `$ gedit optical_flow.cpp` //プログラム作製
- ⑦ `$ cd ..` //移動
- ⑧ `$ gedit CMakeList.txt` //プログラムに合わせて修正
- ⑨ `$ cd ..` //移動
- ⑩ `$ catkin_make` //ビルドする

修正した CMakeList を図 1 に表す。赤枠で囲った 140 行目と 154 行目を変更した。オプティカルフローのプログラムは「Ⅲ オプティカルフロー概要」を行う。



```
{catkin_EXPORTED_TARGETS})
134
135 ## Declare a C++ executable
136 ## With catkin_make all packages are built within a single CMake context
137 ## The recommended prefix ensures that target names across packages don't collide
138 # add_executable(${PROJECT_NAME}_node src/my_opencv_node.cpp)
139 ##add_executable(image_converter src/image_converter.cpp)
140 add_executable(optical_flow src/optical_flow.cpp)##change
141
142 ## Rename C++ executable without prefix
143 ## The above recommended prefix causes long target names, the following renames the
144 ## target back to the shorter version for ease of user use
145 ## e.g. "roslaunch someones_pkg node" instead of "roslaunch someones_pkg someones_pkg_node"
146 # set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")
147
148 ## Add cmake target dependencies of the executable
149 ## same as for the library above
150 # add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS}
151 # ${catkin_EXPORTED_TARGETS})
152
153 ## Specify libraries to link a library or executable target against
154 target_link_libraries(optical_flow ${catkin_LIBRARIES} )##change
155 #####
156 ## Install ##
157 #####
158
159 # all install targets should use catkin DESTINATION variables
160 # See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html
161
162 ## Mark executable scripts (Python etc.) for installation
163 ## in contrast to setup.py, you can choose the destination
164 # install(PROGRAMS
165 #   scripts/my_python_script
166 #   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

Figure 1 CMakeList.txt の変更箇所

## 【Ⅱ USB カメラへの接続】

まず、USB カメラのソフトをインストールするため下記のコマンドを入力する。

- ① `$ cd my_robot/src //移動`
- ② `$git clone https://github.com/ros-drivers/usb\_cam //github から ROS で USB カメラを使えるソフトをインストールする.`
- ③ `$cd .. //移動`
- ④ `$ catkin make //ビルドする`

次に USB カメラをポートに接続した後、ターミナルを 3 つ開く。それぞれのターミナルで `my_robot` のディレクトリに移動し、`source devel/setup.bash` を入力し、自分の作成したワークスペースをインストール環境に **Overlay** する。**Overlay** とは主にプログラムが大きすぎてメインメモリに格納できないような場合に、プログラムを機能ごとに分割し、それぞれのセグメントをプログラム自身が管理、制御することである。

上記を行った後以下のコマンドを入力することで取得した画像がウィンドウに表示する。

- ⑤ `$ roscore //ロスを動かす（ノードとの間を通信する）…ターミナル 1`
- ⑥ `$ rosrn usb_cam usb_cam node…ターミナル 2`
- ⑦ `$ rostopic // USBカメラに接続しているか確認する…ターミナル 3`
- ⑧ `$ $rosrn my_opencv optical_flow//ノードを立ち上げる…ターミナル 3`

## 【Ⅲ オプティカルフロー概要】

まず、オプティカルフローとは、物体の動きをベクトルで表すものである。今回作製したプログラムのサンプルは <http://www.cellstat.net/opticalflow/> から取得した。検索ワードはオプティカルフロー `opencv c+` で見つけることができる。また、作製したプログラムは Github の <https://github.com/t11708m-chukyo/My1-strepository> 内の `src/my_opencv` にオプティカルフローを動かすための `CMakeLists.txt` があり、そのさらに下の `src` に今回のプログラムである `optical_flow.cpp` がある。

今回作製したプログラムは図 1,2 のようである。青枠で囲んである部分が新しく加えたコードである。Github の `optical_flow.cpp` と同じ場所にある `image_converter.cpp` が元のコードである。参考コードでは取得画像が `source` であるが、今回は `gray` に変更している。また参考コードの `main` 文を削除している。

```

1 #include <ros/ros.h>
2 #include <image_transport/image_transport.h>
3 #include <cv_bridge/cv_bridge.h>
4 #include <sensor_msgs/image_encodings.h>
5 #include <opencv2/imgproc/imgproc.hpp>
6 #include <opencv2/highgui/highgui.hpp>
7
8 // NS-code
9 #include "string.h"
10 #include "stdio.h"
11 #include "stdlib.h"
12 #include "iostream"
13 #include "math.h"
14 #include "opencv/cv.h"
15 #include <opencv2/core/core.hpp>
16 #include "opencv2/video/tracking.hpp"
17 using namespace std;
18 using namespace cv;
19 int frame_count=0;
20
21 static const std::string OPENCV_WINDOW = "Image window";
22
23 cv::Mat gray;//current fream
24 cv::Mat gray2;//previous frame
25
26 class ImageConverter
27 {
28     ros::NodeHandle nh ;
29     image_transport::ImageTransport it_;
30     image_transport::Subscriber image_sub_;
31     image_transport::Publisher image_pub_;
32
33 public:
34     ImageConverter()
35         : it_(nh_)
36     {
37         // Subscribe to input video feed and publish output video feed
38         // image_sub_ = it_.subscribe("/camera/image_raw", 1,
39         image_sub_ = it_.subscribe("/usb_cam/image_raw", 1,
40             &ImageConverter::imageCb, this);
41         image_pub_ = it_.advertise("/image_converter/output_video", 1);
42
43         cv::namedWindow(OPENCV_WINDOW);
44     }
45
46     ~ImageConverter()
47     {
48         cv::destroyWindow(OPENCV_WINDOW);
49     }
50

```

C++ ▾ Tab Width: 2 ▾ Ln 82

Figure 2 作製した optical\_flow.cpp

```

48 // cv::randu(randu(0, 255, randu));
49 }
50
51 void imageCb(const sensor_msgs::ImageConstPtr& msg)
52 {
53     cv_bridge::CvImagePtr cv_ptr;
54     try
55     {
56         cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
57     }
58     catch (cv_bridge::Exception& e)
59     {
60         ROS_ERROR("cv_bridge exception: %s", e.what());
61         return;
62     }
63
64     // Draw an example circle on the video stream
65     // if (cv_ptr->image.rows > 60 && cv_ptr->image.cols > 60)
66     //     cv::circle(cv_ptr->image, cv::Point(50, 50), 10, CV_RGB(255,0,0));
67
68     // NS-code
69     cv::cvtColor( cv_ptr->image, gray, CV_BGR2GRAY );//make an image monochrome
70
71     image_pub_.publish( cv_bridge::CvImage( std_msgs::Header(), "bgr8", gray).toImageMsg());
72
73     // NS-code OpticalFlow
74     Mat disp = gray.clone();//copy
75     if( frame_count == 0 ) gray2 = gray.clone();//exception handling
76     if( frame_count > 0 )
77     {
78         vector<cv::Point2f> prev_pts;//point
79
80         Size flowSize(50,50);//interval between the flows
81
82         Point2f center = cv::Point(gray.cols/2., gray.rows/2.);
83         for(int i=0; i<flowSize.width; ++i)
84         {
85             for(int j=0; j<flowSize.height; ++j)
86             {
87                 Point2f p(i*float(gray.cols)/(flowSize.width-1), j*float(gray.rows)/(flowSize.height-1));
88                 prev_pts.push_back((p-center)*0.95f+center);//store a calculation result
89             }
90         }
91
92         Mat flow;
93
94         calcOpticalFlowFarneback(gray2, gray, flow, 0.8, 10, 15, 3, 5, 1.1, 0);
95
96
97

```

Figure 3 作製した optical\_flow.cpp

```

96     calcOpticalFlowFarneback(gray2, gray, flow, 0.8, 10, 15, 3, 5, 1.1, 0);
97
98     // オプティカルフローの表示
99     std::vector<cv::Point2f>::const_iterator p = prev_pts.begin();
100    for(; p!=prev_pts.end(); ++p)
101    {
102        const cv::Point2f& fxy = flow.at<cv::Point2f>(p->y, p->x);
103        double val_flow = flow.at<double>(p->y, p->x);
104        if( val_flow > 10.0) cv::line(dis, *p, *p+fxy*8, cv::Scalar(0,1)); //designation of the si
105    }
106
107    gray2 = gray.clone();
108
109    imshow("vector", dis); //result image
110    imshow("source", gray); //former image
111
112    int c = waitKey(1);
113    }
114    // Output modified video stream
115    image_pub_.publish(cv_ptr->toImageMsg());
116
117    // NS-code
118    frame_count++;
119    }
120 }
121 };
122
123 int main(int argc, char** argv)
124 {
125     ros::init(argc, argv, "image_converter");
126     ImageConverter ic;
127     ros::spin();
128     return 0;
129 }

```

C++ Tab Width: 2 Ln 48, Col 38 IN

Figure 4 作製した optical\_flow.cpp

#### 【IV オプティカルフロー 動作結果】

図 5、6 は flowsize(100,100)の結果であり、図 7、8 は flowsize(50,50)に示す。下図のように flowsize を変化させることでオプティカルフローの間隔を調整することができる。flowsize(100,100)では物体自体がオプティカルフローの線で隠れて、flowsize(50,50)では物体自体は見やすくなった。今回の撮影ではカメラから画像に反映されるのに時間がかかってしまうため今後は処理速度が早い PC で行う必要がある。

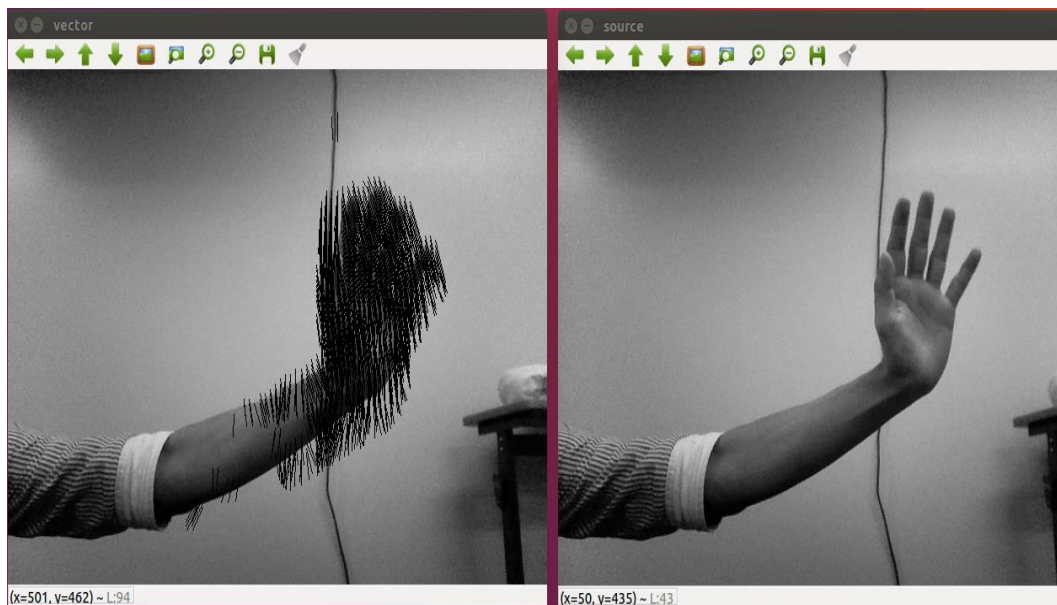


Figure 4 オプティカルフロー結果①

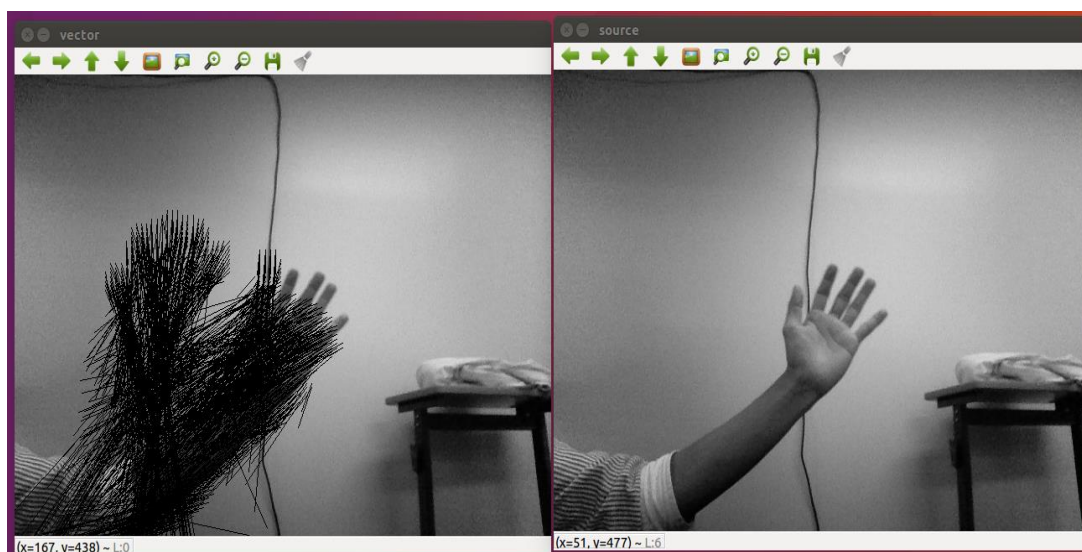


Figure6 オプティカルフロー結果②



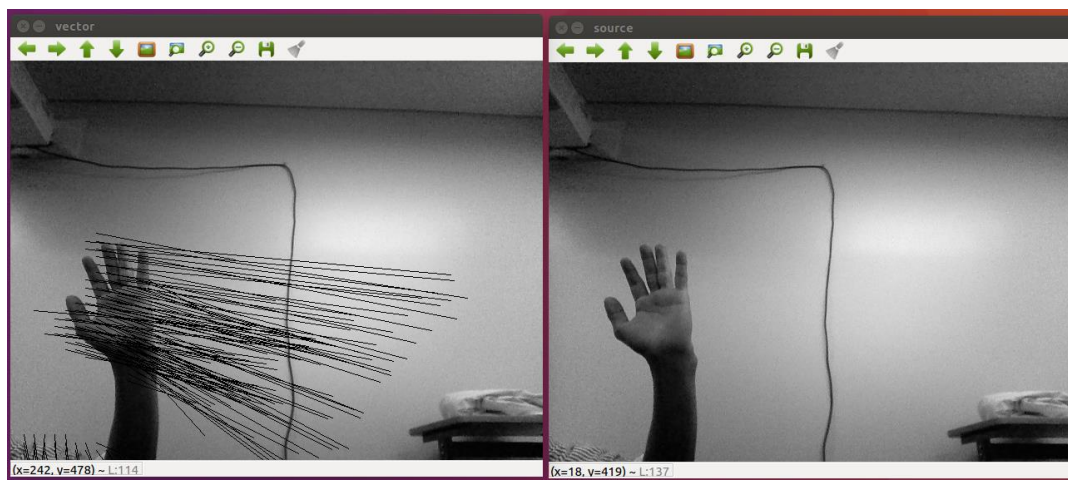


Figure 7 オプティカルフロー結果③

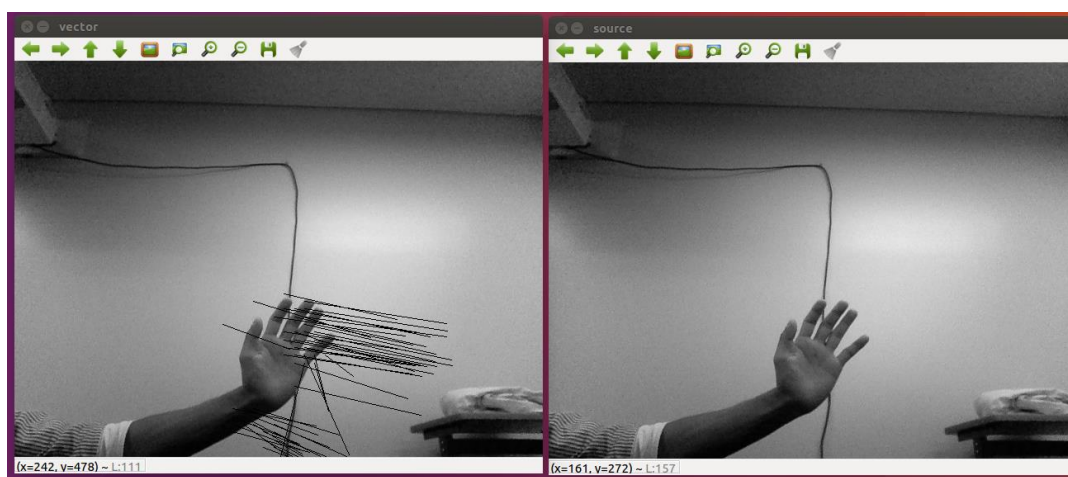


Figure 8 オプティカルフロー結果④