Praktikumsbericht

Christian Waldner

August 30, 2017

1 Anbindung der 'Kinect for Windows' an die ROS-Distribution 'Kinetic'

1.1 Aufgabenstellung

Bei der Microsoft Kinect-Kamera handelt es sich um eine Stereokamera, die neben RGB-Informationen ebenfalls ein IR-Bild und sogenannte PointCloud-Daten liefert. Mit diesen Daten besteht die Möglichkeit die Umgebung in 3D abzubilden und Entfernungen zu messen.
In diesem Projekt war vorgesehen die 'Kinect for Windows' an das ROS-Betriebssystem so anzubinden, dass die Darstellung von PointClouds sowie das Tracking von Bewegungen möglich ist.

1.2 Umsetzung

Um die gewünschte Aufgabenstellung zu erreichen stellt das Robot-Operating-System (ROS) einige Packages zur Verfügung. Als problematisch stellte sich jedoch heraus, dass einige dieser Pakete noch nicht auf die Kinetic-Distribution übertragen waren oder deren Entwicklung eingestellt wurde.
Zudem sind einige dieser Packages auf die Kinect mit der Versionsnummer 1414 ausgelegt. Die 'Kinect for Windows' hat die Versionsnummer 1517 und weißt dementsprechend eine andere Firmware auf.

1.2.1 Implementierung der PointCloud

Um rein eine PointCloud darstellen zu können kann das ROS-Package **freenect_launch** genutzt werden. Dieses Paket greift auf die Bibliothek **libfreenect** zurück und ermöglicht den Zugriff auf die Kinect-Kameras.

Das Paket kann mit folgenden Befehlen installiert werden:

```
$ sudo apt-get install ros-kinetic-freenect-launch
$ sudo apt-get install ros-kinetic-freenect-stack
```

Nach der Installation wird das Paket folgendermaßen gestartet:

```
$ roslaunch freenect_launch freenect.launch
```

Im Anschluss kann RVIZ gestartet werden:

```
$ rosrun rviz rviz
```

In RVIZ sind folgende Einstellungen vorzunehmen:

In Global Options muss der Fixed Frame auf **/camera_link** oder **/camera_depth_frame** geändert werden.
Über **ADD** kann anschließend das Display **PointCloud2** hinzugefügt werden. Hierbei ist der Topic **/camera/depth_registered/points** zu wählen.
Als Color Transformer stehen verschiedene Möglichkeiten zur Verfügung. Während **RGB8** die reale Farbdarstellung wiedergibt, kann über **Intensity** eine Regenbogenfarbdarstellung gewählt werden, die den Abstand farblich kodiert.

Alternativ hierzu kann statt der **PointCloud2** eine **DepthCloud** hinzugefügt werden. Diese bietet keine RGB8-Darstellung, weißt aber eine geringere Latenz, bessere Auflösung und Leistung auf.

1.2.2 Implementierung von PointCloud to Laserscan

Durch die zwei versetzt montierten IR-Kameras der Kinect ist, neben der Darstellung als Pointcloud auch, das Berechnen von Entfernungen möglich. Hierzu dient das Package **depthimage_to_laserscan**.
Dieses Package stellt die notwendige Implementierung zur Verfügung, die es ermöglicht, in RVIZ den Abstand zu beispielsweise Hindernissen farblich darzustellen, sowie auch die notwendigen ROS-Nodes, um die gemessenen Werte weiter verwenden zu können.

Installiert wird das Package mit folgendem Befehl:

```
$ sudo apt-get install ros-kinetic-depthimage-to-laserscan
```

Da das Package unter Anderem das Topic **sensor_msgs/Image** nutzt, das nicht ohne Weiteres erkannt wird, muss der Image-Topic beim Start remapped werden. Dies geschieht mit folgendem Aufruf:

```
$ rosrun depthimage_to_laserscan depthimage_to_laserscan image:=/camera/depth/image_raw
```

Anschließend kann in RVIZ über **ADD** ein Laserscan hinzugefügt werden. Dort ist als Color Transformer **AxisColor** und als Axis **x** auszuwählen, um die Entfernungen in der Horizontalen korrekt darstellen zu können.

1.2.3 Implementierung des Motiontracking

Weitaus umfangreicher stellte sich die Implementierung für das Motiontracking dar. Aufgrund der in Punkt 2 beschriebenen Problematiken, war es nicht möglich ohne Weiteres auf ROS-Packages zurückzugreifen.

Als erstes sind einige Libraries notwendig die mit folgendem Befehl installiert werden können:

```
$ sudo apt-get install freeglut3-dev pkg-config build-essential libxmu-dev libxi-dev
  libusb-1.0-0-dev doxygen graphviz mono-complete
```

Nach erfolgter Installation der Libraries, gilt es OpenNI zu installieren. Hierzu sind folgende Befehle auszuführen:

```
$ mkdir -p ~/kinect
$ cd ~/kinect
$ git clone https://github.com/OpenNI/OpenNI.git
$ git checkout unstable
$ cd OpenNI/Platform/Linux/CreateRedist
$ ./RedistMaker
$ cd ~/kinect/OpenNI/Platform/Linux/Redist/OpenNI-Bin-Dev-Linux-x64-v1.5.8.5/
$ sudo ./install.sh
```

Im Anschluss sind die PrimeSense-Treiber zu installieren:

```
$ cd ~/kinect
$ git clone https://github.com/ph4m/SensorKinect.git
$ git checkout unstable
$ cd SensorKinect/Platform/Linux/CreateRedist
$ ./RedistMaker
$ cd ~/kinect/SensorKinect/Platform/Linux/Redist/Sensor-Bin-Linux-x64-v5.1.2.1/
$ sudo ./install.sh
```

Weiter müssen noch zwei Pakete, in den vorher erstellten kinect-Ordner, heruntergeladen werden.
Zum einen handelt es sich dabei um die **OpenNI SDK v.2.1** und zum anderen um **NiTE v1.5.2.23**

Link: http://openni.ru/openni-sdk/openni-sdk-history-2/index.html

Nach dem Download, müssen beide Pakete entpackt und installiert werden.

```
$ cd ~/kinect
$ unzip <PAKETNAME>
$ tar -jxvf <PAKETNAME>
$ cd <PAKETORDNER>
$ sudo ./install.sh
```

Nachdem die Pakete installiert sind, wird das Package **openni_tracker** geklont und kompiliert.

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/ros-drivers/openni_tracker.git
$ cd ~/catkin_ws
$ catkin_make
```

Bevor der Motiontracker zusammen mit der PointCloud genutzt werden kann, muss noch eine Node remapped werden, da sonst Übersetzungsfehler aufgrund unterschiedlicher Berechnungsperspektiven entstehen.

Dies geschieht mit dem Befehl:

```
$ rosparam set /openni_tracker/camera_frame_id camera_link
```

Jetzt sind sämtliche notwendigen Pakte installiert und der Motiontracker kann folgendermaßen gestartet werden:

```
$ rosrun openni_tracker openni_tracker
```

In RVIZ muss zu der Pointcloud noch via **ADD** das Display **TF** hinzugefügt werden.

1.3 Zusammenfassung

Die Anbindung der 'Kinect for Windows' (K4W) an das 'Robot Operating System' (ROS) ist, nachdem die notwendigen Packages und Treiber installiert worden sind, grundsätzlich ohne Weiteres möglich. Es stellte sich jedoch im Verlauf der Untersuchungen heraus, dass die 'Kinect' mit verschiedenen Versionsnummern auf den Markt gekommen ist. So ist z.B. die 'Kinect for XBox360' mit der Versionsnummer 1414 und später mit 1473 ausgeliefert worden. Die im Institut verwendete K4W hat die Versionsnummer 1517.

Nach meinen Recherchen sind mit den unterschiedlichen Versionsnummern auch unterschiedliche Firmwares implementiert, die nicht alle von den oben genannten ROS-Paketen unterstüzt werden. So ist unter anderem die Funktionalität des Motors, der die Neigefunktion der Kinect ermöglicht, mit der vom Institut genutzten Hardware nicht gegeben.
Das Package 'kinect_aux' stellt in Verbindung mit dem 'OpenNI'-Treibern, diese Funktionalität für die Version 1414 zur Verfügung, jedoch ausschließlich für diese. Für die Versionsnummern 1473 und 1517 gibt es die Möglichkeit, die Firmware zur Laufzeit zu aktualisieren.

Hierzu müssen Teile des Programms 'OpenFrameworks' eingebunden und nach dem Befehl `freenect_init()` einer der beiden folgenden Aufrufe erfolgen.

```
freenect_set_fw_address_nui(kinectContext, ofxKinectExtras::getFWData1473(), ofxKinectExtras::getFWSize1473());
freenect_set_fw_address_k4w(kinectContext, ofxKinectExtras::getFWDatak4w(), ofxKinectExtras::getFWSizek4w());
```

Genauere Informationen hierzu sind unter folgenden Links einsehbar.

https://github.com/openframeworks/openFrameworks/tree/master/addons/ofxKinect/src/extra
https://github.com/OpenKinect/libfreenect/issues/487

Eine weitere Möglichkeit wäre, das Programm 'kinect_aux_node.cpp', vom ROS-Package 'kinect_aux', so umzuprogrammieren, dass gegebenenfalls diese Funktionalität gegeben ist.
Wird das 'kinect_aux'-Package unverändert genutzt, kommt es direkt beim Start zu einer Fehlermeldung, da 'kinect_aux' auf die Versionsnummer 1414, mit der ProduktID 0x02b0 prüft und somit die K4W nicht erkannt wird.

Die Identifikation der K4W kann dadurch sichergestellt werden, dass das `#define MS_MAGIC_MOTOR_PRODUCT` von `0x02b0` (Version 1414) auf `0x02c2` (Version 1517) geändert wird. Dieses dient zum Festlegen der Produkt-ID, nach der die angeschlossenen USB-Geräte durchsucht werden. Für die Abfrage und die weitere Kommunikation wird die 'libusb-1.0-0-dev'-Bibliothek genutzt.
Um die gewünschte Neigung einer 'K4W' zu erhalten besteht die Möglichkeit die Kinect SDK V1.8 herunter zu laden und die Einstellung über ein Windows-Betriebssystem

vorzunehmen.

Im Zuge der Untersuchungen fiel im Weiteren auf, dass die Kamera **nicht** an virtuelle Maschinen angeschlossen und betrieben werden kann.

Mit Ausnahme der Neigemöglichkeit der Kinect, stehen die Funktionalitäten der Point-/DepthCloud und des Motiontrackings zur Verfügung. Neben der dreidimensionalen Darstellung, der PointCloud in RVIZ, ist es ebenfalls möglich die Entfernung zu erkannten Hindernissen, mittels dem Package 'depthimage-to-laserscan", zu ermitteln.

1.4 Ausblick

Nachdem die grundlegenden Funktionalitäten, mit Ausnahme der Tilt-Funktion, gegeben sind, gilt es diese auf Genauigkeit, Zuverlässigkeit und mögliche Nutzen zu untersuchen. Das sind neben dem Erfassen von Verarbeitungsbereichen der Kameras, das Verifizieren der Entfernungsmessungen und das Messen der Zeitverzögerungen, die sich aus der Laufzeit der Berechnung ergeben.
Weiter wäre die Möglichkeit der Bildverarbeitung zu Untersuchen, um beispielsweise eine Mustererkennung zu implementieren.
Neben der Überprüfung der gegebenen Funktionalitäten, ist die Möglichkeit den Tilt-Motor, unter Nutzung von 'OpenFrameworks', in Betrieb zu nehmen zu prüfen.

2 Möglichkeiten der Anwendung von OpenCV in ROS

2.1 Aufgabenstellung

Im zweiten Projekt galt es zu Untersuchen welche Möglichkeiten OpenCV in Verbindung mit dem ROS bietet und wie diese implementiert und eingesetzt werden können.

2.2 Umsetzung

Das Robot Operating System stellt auch für diese Aufgabenstellung verschiedene Pakete zur Verfügung. Zum Einen das Paket **opencv3**, das die Funktionen von OpenCV in das ROS überführt. Des Weiteren ist, für den Betrieb einer Pinholekamera (z.B. einer USB-Webcam), das Paket **usb_cam** notwendig. Dieses Package wandelt die Bilder der Kamera in das von ROS genutzte Format **image_raw** um und stellt diese als Topic zur Verfügung. Da das Paket **opencv3** wiederum ein eigenes Bildformat verwendet ist noch das Paket **cv_bridge**, dass die Bildformate ineinander überführt, notwendig.

2.2.1 Installation der Pakete

Da die für den Betrieb von OpenCV benötigten Pakete nicht Bestandteil des Core Betriebssystems sind, müssen diese noch installiert werden. Hierzu muss folgender Befehl ausgeführt werden.

```
$ sudo apt-get install ros-kinetic-usb-cam ros-kinetic-opencv3 ros-kinetic-cv-bridge
```

Nach erfolgter Installation ist es notwendig einige Einstellungen zu überprüfen und ggf. anzupassen.

2.2.2 Einstellungen

Das Paket **usb_cam** stellt die Verbindung, der angeschlossenen oder schon hardwareseitig installierten Kameras dem ROS, zur Verfügung. Hierzu muss in der Launchdatei die gewünschte Kamera ausgewählt werden. Über das Kommando

```
$ ls -l /dev/
```

können die angeschlossenen Kameras identifiziert werden. Sie werden als `videoX` aufgelistet.

Neben der Auswahl der Kamera, kann in der Launchdatei ebenfalls die gewünschte Auflösung eingegeben werden.

Nachdem diese Einstellungen vorgenommen wurden, kann die Entwicklung von OpenCV - Programmen begonnen werden.

2.3 Programmentwicklung

2.3.1 Erstellen eines ROS-Package

Bevor mit der Entwicklung eigener Programme begonnen werden kann, muss ein ROS-Package erstellt werden.

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg <PACKAGENAME> sensor_msgs cv_bridge roscpp std_msgs image_transport
$ cd ~/catkin_ws/src/<PACKAGENAME>
$ mkdir src
```

Die Argumente nach `catkin_create_pkg <PACKAGENAME>` beschreiben die notwendigen Abhängigkeiten, die mit diesem Befehl automatisch in die **CMakeLists.txt** übernommen werden. Es ist jedoch nicht zu vergessen, dass für jedes Programm die Zeileneinträge `add_executable()` und `target_link_libraries()` erstellt bzw. angepasst werden müssen. Des Weiteren können weitere Abhängigkeiten entstehen, die dann ebenfalls ergänzt werden müssen.

Nachdem das Paket erstellt wurde, kann im Unterordner `~/.../<PACKAGENAME>/src` ein neues Sourcefile erstellt werden.

2.3.2 Aufbau eines Programms

Auf die grundlegenden Prinzipien, ein ROS-Programm zu schreiben, möchte ich an dieser Stelle nicht eingehen und verweise auf die vorhandenen Tutorials. In diesem Abschnitt möchte ich die Besonderheiten, die OpenCV fordert, behandeln.

Da OpenCV ein eigenes Dateiformat für die Bildverarbeitung nutzt muss zuerst der originale Bildtopic subscribed und umgewandelt werden.

Header

ROS-spezifische Headerfiles:

```
1 #include <ros/ros.h>
2 #include <image_transport/image_transport.h>
3 #include <sensor_msgs/image_encodings.h>
```

OpenCV-spezifische Headerfiles:

```
5 #include <cv_bridge/cv_bridge.h>
6 #include <opencv2/imgproc/imgproc.hpp>
7 #include <opencv2/highgui/highgui.hpp>
```

Klassenbeschreibung und Callbackfunktion

Im Konstruktor der Klasse `ImageConverter` werden die Subscriber(Zeile: 22) und Publisher(Zeile: 23) definiert.
Der Subscriber empfängt die Daten des Package **usb_cam** und ruft bei jeder empfangenen Message die Callbackfunktion `imageCb` auf. Der Publisher stellt anschließend die verarbeiteten Daten als neuen Topic zur Verfügung.

```
11 class ImageConverter{
12   ros::NodeHandle nh_;
13   image_transport::ImageTransport it_;
14   image_transport::Subscriber image_sub_;
15   image_transport::Publisher image_pub_;
16
17 public:
18   ImageConverter()
19     : it_(nh_)
20   {
21
22     image_sub_ = it_.subscribe("/camera/image_raw", 1, &ImageConverter::
         imageCb, this);
23     image_pub_ = it_.advertise("/image_converter/output_video", 1);
24
25     cv::namedWindow(OPENCV_WINDOW);
26   }
27
28   ~ImageConverter(){
29     cv::destroyWindow(OPENCV_WINDOW);
30   }
```

Die Callbackfunktion ist der Programmabschnitt, in dem die eigentliche Bildverarbeitung stattfindet. Im ersten Schritt müssen die Bilder vom ROS-Image-Format in das OpenCV-Image-Format umgewandelt werden. Dies geschieht in den Zeilen 32 bis 39. Im Anschluss findet die Verarbeitung der empfangenen Daten statt.

```
32   void imageCb(const sensor_msgs::ImageConstPtr& msg){
33     cv_bridge::CvImagePtr cv_ptr;
34     try{
35       cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::
           BGR8);
36     }catch (cv_bridge::Exception& e){
37       ROS_ERROR("cv_bridge exception: %s", e.what());
38       return;
39     }
40
41 /*
42  * In diesem Bereich wird die eigentliche Funktionalitaet beschrieben
43  *
44  *
45  *
46  * In diesem Bereich wird die eigentliche Funktionalitaet beschrieben
47  */
```

Ausgabe- und Mainfunktion

In den restlichen Programmzeilen wird das Ausgabefenster (Zeile:50-51) sowie der Ausgabetopic (Zeile: 54) generiert. In der Main-Funktion wird zum Einen mit `ros::init()` die neu erzeugte ROS-Node beim Master angemeldet und zum Anderen die Klasse `ImageConverter` deklariert.

```
58     // Generiere GUI Window
59     cv::imshow(OPENCV_WINDOW, cv_ptr->image);
60     cv::waitKey(3);
61
62     // Ausgabe des verarbeiteten Streams
63     image_pub_.publish(cv_ptr->toImageMsg());
64   }
65 };
66
67 int main(int argc, char** argv){
68   ros::init(argc, argv, "image_converter");
69   ImageConverter ic;
70   ros::spin();
71   return 0;
72 }
```

2.3.3 Bisherige Implementierungen

Nach aktuellem Stand sind folgende Möglichkeiten gegeben:

- Farbdetektion / -trennung

- Objekttracking mit und ohne Folgelinie

- Schachbrettabbildung in RVIZ

- Mustererkennung - allgemein oder farblich getrennt

- Erzeugung von Steuersignalen (Twist_msgs/cmd_vel) via Gestensteuerung

- Erfassen von ArUco-Marken

Zu finden sind diese Implementierungen im Package `own_cv`. Die Dokumentation kann via Doxygen generiert werden.

2.4 Zusammenfassung

Die OpenCV-Library bietet eine nahezu unendliche Fülle an Möglichkeiten, die insbesondere in Verbindung mit dem ROS, neue Möglichkeiten im Bereich der Sensorik sowie der Lokalisation und Steuerung bietet.
Grundsätzlich ging es hierbei jedoch um die Feststellung der grundlegenden Möglichkeiten sowie den Aufbau der Programme, die eine Portierbarkeit in das ROS ermöglichen und somit die Verwendung der OpenCV-Library gewährleistet. Häufig traten Schwierigkeiten in der Portierung von Beispielen, die meist für die Version 2.4 geschrieben waren, in die Version 3.0 / 3.2 auf. In diesem Versionsschritt wurde die Syntax und dementsprechend die Erzeugung einiger Elemente stark verändert.

2.5 Ausblick

Mit dem Programm **twist_msg_generator_by_tracking** ist eine erste Möglichkeit dargestellt, wie die verarbeiteten Bilddaten genutzt werden können, um einen Roboter zu steuern. Diese äußerst einfache Implementierung soll die Vielfältigkeit der Bildverarbeitung zeigen. Aus diesem Grund fällt es schwer einen konkreten Ausblick zu formulieren, da die Bandbreite der Möglichkeiten sehr umfangreich und breit aufgestellt sind.
Da die aktuellen Untersuchungen mit einer USB-Webcam (einer Pinholekamera) durchgeführt wurden, ist es ratsam bei folgenden Versuchen die Möglichkeiten einer Stereokamera, wie beispielsweise der Kinect, zu überprüfen. So wäre die Erweiterung des SLAM-Algorithmus, als 3D-Darstellung einer Pointcloud, in Verbindung mit Landmarken eine interessante Möglichkeit die Lokalisation zu verbessern.

Listing 2.1: Grundkonstrukt eines OpenCV-ROS-Programmes

```
/**
 ******************************************************************************

 * @file                template.cpp
 * @author              Christian Waldner
 * @date                19/07/2017
 * @copyright   2011-2017 UniBwM - ETTI - Institut 4
 * @brief               This is a template for writing new OpenCV-ROS-
     Nodes
 * @details     Insert the processingsteps between the described section
     and
 *                              adjust the program as you wish
 ******************************************************************************

 * @todo                imppement functionality
 ******************************************************************************

 *
 */
#include <ros/ros.h>
#include <cv_bridge/cv_bridge.h>
#include <image_transport/image_transport.h>
#include <sensor_msgs/image_encodings.h>

#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

using namespace cv;
using namespace std;

static const char* OPENCV_WINDOW = "Original Image";
static const char* PROCESSED_WINDOW = "Processed Image";

/*                                                          //DEFAULT
    //RED
int iLowH =  0;         //0                     0
int iHighH = 11;        //179           19

int iLowS =  78;        //0                     128
int iHighS = 255;       //255           255

int iLowV =  110;       //0                     37
int iHighV = 255;       //255           255

void redCb(int state, void *ptr){
        cvSetTrackbarPos("LowH",  NULL, 000);
        cvSetTrackbarPos("HighH", NULL, 16);
        cvSetTrackbarPos("LowS",  NULL, 88);
        cvSetTrackbarPos("HighS", NULL, 255);
        cvSetTrackbarPos("LowV",  NULL, 000);
        cvSetTrackbarPos("HighV", NULL, 173);
}

void greenCb(int state, void *ptr){
        cvSetTrackbarPos("LowH",  NULL, 47);
```

```
50          cvSetTrackbarPos("HighH", NULL, 74);
51          cvSetTrackbarPos("LowS",  NULL, 75);
52          cvSetTrackbarPos("HighS", NULL, 255);
53          cvSetTrackbarPos("LowV",  NULL, 71);
54          cvSetTrackbarPos("HighV", NULL, 104);
55  }
56  void blueCb(int state, void *ptr){
57          cvSetTrackbarPos("LowH",  NULL, 89);
58          cvSetTrackbarPos("HighH", NULL, 125);
59          cvSetTrackbarPos("LowS",  NULL, 159);
60          cvSetTrackbarPos("HighS", NULL, 255);
61          cvSetTrackbarPos("LowV",  NULL, 110);
62          cvSetTrackbarPos("HighV", NULL, 255);
63  }
64  void yellowCb(int state, void *ptr){
65          cvSetTrackbarPos("LowH",  NULL, 26);
66          cvSetTrackbarPos("HighH", NULL, 38);
67          cvSetTrackbarPos("LowS",  NULL, 78);
68          cvSetTrackbarPos("HighS", NULL, 255);
69          cvSetTrackbarPos("LowV",  NULL, 110);
70          cvSetTrackbarPos("HighV", NULL, 255);
71  }
72
73  */
74
75
76  /**
77    *     @class          ImageConverter
78    * @brief              ImageConverter
79    * @details    This class converts ROS-images into OpenCV-image format
80    */
81  class ImageConverter{
82          ros::NodeHandle nh_;
83          image_transport::ImageTransport it_;
84          image_transport::Subscriber image_sub_;
85          image_transport::Publisher image_pub_;
86
87
88  public:
89          ImageConverter():it_(nh_){
90                  image_sub_ = it_.subscribe("usb_cam/image_raw", 1, &
                        ImageConverter::imageCb, this);
91
92                  image_pub_ = it_.advertise("Image_converter/
                        thresholded_video", 1); //Topicname anpassen
93
94                  namedWindow(OPENCV_WINDOW);
95                  namedWindow(PROCESSED_WINDOW);
96
97
98                  cvMoveWindow(OPENCV_WINDOW, 100, 50);
99                  cvMoveWindow(PROCESSED_WINDOW, 750, 50);
100
101 /*
102                 //Hue
103                 cvCreateTrackbar("LowH", NULL, &iLowH, 179);
```

```
104                 cvCreateTrackbar("HighH", NULL, &iHighH, 179);
105
106                 //Saturation
107                 cvCreateTrackbar("LowS", NULL, &iLowS, 255);
108                 cvCreateTrackbar("HighS", NULL, &iHighS, 255);
109
110                 //Value
111                 cvCreateTrackbar("LowV", NULL, &iLowV, 255);
112                 cvCreateTrackbar("HighV", NULL, &iHighV, 255);
113
114
115                 cvCreateButton("Red", redCb, NULL, CV_PUSH_BUTTON, 0);
116                 cvCreateButton("Green", greenCb, NULL, CV_PUSH_BUTTON, 0)
                        ;
117                 cvCreateButton("Blue", blueCb, NULL, CV_PUSH_BUTTON, 0);
118                 cvCreateButton("Yellow", yellowCb, NULL, CV_PUSH_BUTTON,
                        0);
119                 //cvCreateButton("Red", redCb, NULL, CV_PUSH_BUTTON, 0);
120 */
121
122         }
123
124         ~ImageConverter(){
125                 destroyWindow(OPENCV_WINDOW);
126         }
127
128         void imageCb(const sensor_msgs::ImageConstPtr& msg){
129                 cv_bridge::CvImagePtr cv_ptr;
130                 try{
131                         cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                                image_encodings::BGR8);
132                 }catch(cv_bridge::Exception& e){
133                         ROS_ERROR("cv_bridge exception: %s", e.what());
134                         return;
135                 }
136
137                 //input functionality here
138                 //
139                 //
140                 //
141                 //input functionality here
142
143
144                 //Show processed Image
145                 imshow(PROCESSED_WINDOW, imgThresholded);
146
147                 //Show original Image
148                 imshow(OPENCV_WINDOW, cv_ptr->image);
149
150                 waitKey(3);
151
152                 //convert and publish thresholded image
153                 cvtColor(imgThresholded, cv_ptr->image, COLOR_GRAY2RGB);
154
155                 image_pub_.publish(cv_ptr->toImageMsg());
156         }
```

```
157 };
158
159 /**
160  * @brief       main function
161  * @details Initializes ROS
162  * @param       [in] argc
163  * @param       [in] argv
164  * @return      escape sequence
165  */
166 int main (int argc, char** argv){
167         ros::init(argc, argv, "image_converter");
168         ImageConverter ic;
169         ros::spin();
170         return 0;
171 }
```

Listing 2.2: Grundaufbau einer OpenCV - ROS-Node

```
1 #include <ros/ros.h>
2 #include <image_transport/image_transport.h>
3 #include <sensor_msgs/image_encodings.h>
4
5 #include <cv_bridge/cv_bridge.h>
6 #include <opencv2/imgproc/imgproc.hpp>
7 #include <opencv2/highgui/highgui.hpp>
8
9 static const std::string OPENCV_WINDOW = "Image window";
10
11 class ImageConverter{
12   ros::NodeHandle nh_;
13   image_transport::ImageTransport it_;
14   image_transport::Subscriber image_sub_;
15   image_transport::Publisher image_pub_;
16
17 public:
18   ImageConverter()
19     : it_(nh_)
20   {
21
22     image_sub_ = it_.subscribe("/camera/image_raw", 1, &ImageConverter::
         imageCb, this);
23     image_pub_ = it_.advertise("/image_converter/output_video", 1);
24
25     cv::namedWindow(OPENCV_WINDOW);
26   }
27
28   ~ImageConverter(){
29     cv::destroyWindow(OPENCV_WINDOW);
30   }
31
32   void imageCb(const sensor_msgs::ImageConstPtr& msg){
33     cv_bridge::CvImagePtr cv_ptr;
34     try{
35       cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::
           BGR8);
36     }catch (cv_bridge::Exception& e){
37       ROS_ERROR("cv_bridge exception: %s", e.what());
38       return;
39     }
40
41 /*
42  * In diesem Bereich wird die eigentliche Funktionalitaet beschrieben
43  *
44  *
45  *
46  * In diesem Bereich wird die eigentliche Funktionalitaet beschrieben
47  */
48
49     // Generiere GUI Window
50     cv::imshow(OPENCV_WINDOW, cv_ptr->image);
51     cv::waitKey(3);
52
53     // Ausgabe des verarbeiteten Streams
```

```
54      image_pub_.publish(cv_ptr->toImageMsg());
55    }
56 };
57
58 int main(int argc, char** argv){
59    ros::init(argc, argv, "image_converter");
60    ImageConverter ic;
61    ros::spin();
62    return 0;
63 }
```

Listing 2.3: ArUco-Marker Erkennung

```
1  /**
2   ********************************************************************************
3   * @file                aruco_detection.cpp
4   * @author              Christian Waldner
5   * @date                19/07/2017
6   * @copyright   2011-2017 UniBwM - ETTI - Institut 4
7   * @brief               Node which finds arUco-markers
8   * @details     This node returns a graphical output in which ArUco-
        markers are
9   *                              contoured and the axis of their
        geometrical systems is drawn
10  ********************************************************************************
11  * @todo                rename: topicname, init-name,
12  * @todo                implement ROS publisher
13  ********************************************************************************
14  */
15 #include <ros/ros.h>
16 #include <cv_bridge/cv_bridge.h>
17 #include <image_transport/image_transport.h>
18 #include <sensor_msgs/image_encodings.h>
19
20 #include <opencv2/highgui/highgui.hpp>
21 #include <opencv2/imgproc/imgproc.hpp>
22 #include <opencv2/aruco.hpp>
23
24 #include <opencv2/calib3d.hpp>
25
26
27 using namespace cv;
28 using namespace std;
29
30 static const char* OPENCV_WINDOW = "Original␣Image";
31 static const char* PROCESSED_WINDOW = "Processed␣Image";
32
33 Ptr<aruco::Dictionary> dictionary = aruco::getPredefinedDictionary(aruco
        ::DICT_4X4_100);
34 Mat cameraMatrix = (Mat1d(3,3) << 989.13747, 0, 660.60807, 0, 989.13747,
        519.38931, 0, 0, 1);
35 Mat distCoeffs =(Mat1d(1,5) << 0.0087, 0.0, 0.00039, 0.01630, -0.14183);
36
37 /**
38   *     @class          ImageConverter
39   * @brief               ImageConverter
40   * @details     This class converts ROS-images into OpenCV-image format
41   */
42 class ImageConverter{
43         ros::NodeHandle nh_;
44         image_transport::ImageTransport it_;
45         image_transport::Subscriber image_sub_;
46         image_transport::Publisher image_pub_;
47
48
```

```
49 public:
50         ImageConverter():it_(nh_){
51                 image_sub_ = it_.subscribe("usb_cam/image_raw", 1, &
                        ImageConverter::imageCb, this);
52                 image_pub_ = it_.advertise("Image_converter/
                        thresholded_video", 1); //Topicname anpassen
53
54                 namedWindow(OPENCV_WINDOW);
55
56                 cvMoveWindow(PROCESSED_WINDOW, 750, 50);
57         }
58
59         ~ImageConverter(){
60                 destroyWindow(OPENCV_WINDOW);
61         }
62
63         void imageCb(const sensor_msgs::ImageConstPtr& msg){
64                 cv_bridge::CvImagePtr cv_ptr;
65                 try{
66                         cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                                image_encodings::BGR8);
67                 }catch(cv_bridge::Exception& e){
68                         ROS_ERROR("cv_bridge exception: %s", e.what());
69                         return;
70                 }
71                 Mat img = cv_ptr->image;
72                 Mat imgCpy;
73
74                 img.copyTo(imgCpy);
75
76                 vector<int> ids;
77                 vector<vector<Point2f> > corners;
78
79                 aruco::detectMarkers(img, dictionary, corners, ids);
80
81                 if(ids.size() > 0){
82                         aruco::drawDetectedMarkers(imgCpy, corners, ids);
83
84                         vector<Vec3d> rvecs, tvecs;
85                                 aruco::estimatePoseSingleMarkers(
                                        corners, 0.05, cameraMatrix,
                                        distCoeffs, rvecs, tvecs);
86
87                                 for(int i = 0; i < ids.size(); i
                                        ++){
88                                         aruco::drawAxis(imgCpy,
                                                cameraMatrix,
                                                distCoeffs, rvecs[i],
                                                 tvecs[i], 0.1);
89
90 //                                      cout << "Eckpunkte: " <<
       corners.back() << endl;
91
92                                         vector<Point3f>
                                                axisPoints;
93                                         axisPoints.push_back(
```

```
                                                    Point3f(0,0,0));
94                                                  axisPoints.push_back(
                                                    Point3f(0.1,0,0));
95                                                  axisPoints.push_back(
                                                    Point3f(0,0.1,0));
96                                                  axisPoints.push_back(
                                                    Point3f(0,0,0.1));
97
98                                                  vector<Point2f>
                                                    imagePoints;
99
100                                                 projectPoints(axisPoints,
                                                     rvecs[i], tvecs[i],
                                                    cameraMatrix,
                                                    distCoeffs,
                                                    imagePoints);
101
102 //                                              cout << "Achsenpunkte:
    "<< imagePoints <<endl;
103
104 //                                              cout << "Ursprung x-
    Koordinate [" << imagePoints[0].x << "] / Ursprung y-Koordinate [" <<
     imagePoints[0].y << "]"<< endl;
105                                                 }
106                 }
107
108                 imshow(OPENCV_WINDOW, imgCpy);
109
110                 waitKey(3);
111
112                 //convert and publish thresholded image
113 //              cvtColor(imgThresholded, cv_ptr->image, COLOR_GRAY2RGB);
114
115                 image_pub_.publish(cv_ptr->toImageMsg());
116         }
117 };
118 /**
119  * @brief       main function
120  * @details Initializes ROS
121  * @param       [in] argc
122  * @param       [in] argv
123  * @return      escape sequence
124  */
125 int main (int argc, char** argv){
126         ros::init(argc, argv, "image_converter");
127         ImageConverter ic;
128         ros::spin();
129         return 0;
130 }
```

Listing 2.4: Schachbretterkennung

```
/**
 ********************************************************************************
 * @file                chessboard_corner_detection.cpp
 * @author              Christian Waldner
 * @date                19/07/2017
 * @copyright   2011-2017 UniBwM - ETTI - Institut 4
 * @brief               Node which finds corners of a chessboard
 * @details     This node returns a graphical output which shows and connects the
 *                              corners of a chessboard (adjust the size of the board in
 *                              chessboardSize)
 *                              After detection the node generates a tf-topic for visualizing the
 *                              board in RVIZ
 ********************************************************************************
 * @todo                rename: topicname, init-name,
 * @todo                implement ROS publisher
 ********************************************************************************
 */
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>


#include <visualization_msgs/Marker.h>


#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include "opencv2/opencv.hpp"

using namespace cv;
using namespace std;

static const string CAM_TOPIC = "/usb_cam/image_raw";
//static const string CAM_TOPIC = "/camera/rgb/image_raw";

double squareSize = 0.02; //[m]
cv::Size chessboardSize = cv::Size(8, 6);

vector<cv::Point3f> objPoints;

static const string OPENCV_WIN1 = "USB-Cam-Window";
//static const string OPENCV_WIN2 = "Kinect-Cam-Window";

static const string CAM_TOPIC_OUT = "/image_converter/output_usbcam";
//static const string KIN_TOPIC_OUT = "/image_converter/output_kinect";

```

```
50 /**
51   *     @class          ImageConverter
52   * @brief              ImageConverter
53   * @details    This class converts ROS-images into OpenCV-image format
54   */
55 class ImageConverter{
56         ros::NodeHandle nh_;
57
58         ros::Publisher marker_pub;
59
60         image_transport::ImageTransport it_;
61         image_transport::Subscriber image_sub_;
62         image_transport::Publisher image_pub_;
63
64 public:
65         ImageConverter():
66                 it_(nh_)
67 {
68                 image_sub_ = it_.subscribe(CAM_TOPIC, 1, &ImageConverter
                        ::imageCb, this);
69                 image_pub_ = it_.advertise(CAM_TOPIC_OUT, 1);
70
71                 marker_pub = nh_.advertise<visualization_msgs::Marker>("
                        visualisation_marker", 1);
72
73                 cv::namedWindow(OPENCV_WIN1);
74
75                 for (int y = 0; y < chessboardSize.height; y++){
76                         for(int x = 0; x < chessboardSize.width; x++){
77                                 objPoints.push_back(cv::Point3f(x *
                                        squareSize, y * squareSize, 0));
78                         }
79                 }
80 }
81         ~ImageConverter(){
82                 cv::destroyAllWindows();
83         }
84
85         void markerBroadcaster(cv::Mat rmat, cv::Mat tvec){
86
87                 visualization_msgs::Marker points;
88
89                 points.header.frame_id = "camera_link"; //camera_link,
                        camera_depth_optical_frame
90                 points.header.stamp = ros::Time::now();
91                 points.ns = "chessboard_marker";
92                 points.action = visualization_msgs::Marker::ADD;
93                 points.type = visualization_msgs::Marker::POINTS;
94                 points.pose.orientation.w = 1.0;
95                 points.id = 0;
96                 points.scale.x = 0.01;
97                 points.scale.y = 0.01;
98                 points.color.r = 1.0;
99                 points.color.g = 0.0;
100                points.color.b = 1.0;
101                points.color.a = 1.0;
```

```
102
103
104                 geometry_msgs::Point point_cur;
105                 vector<cv::Mat> objPointsInCamFrame;
106                 cv::Mat cam_point;
107                 cv::Mat c_point;
108                 cv::Point3f pointInChessFrame;
109
110
111
112                 for(int i = 0; i < int(objPoints.size()); i++){
113                         pointInChessFrame = objPoints.at(i);
114
115                         cv::Mat m = (cv::Mat_<double>(3,1) <<
                                pointInChessFrame.x, pointInChessFrame.y,
                                pointInChessFrame.z);
116
117                         cam_point = rmat * m + tvec;
118
119                         cv::Point3f p(cam_point);
120
121                         point_cur.x = p.z -0.6;
122                         point_cur.y = -p.x -0.45;
123                         point_cur.z = -p.y -0.35;
124
125
126                         points.points.push_back(point_cur);
127                 }
128
129                 marker_pub.publish(points);
130
131         }
132
133         void imageCb(const sensor_msgs::ImageConstPtr& msg){
134                 cv_bridge::CvImagePtr cv_ptr;
135                 try{
136                         cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                                image_encodings::BGR8);
137                 }catch (cv_bridge::Exception& e){
138                         ROS_ERROR("cv_bridge exception: %s", e.what());
139                         return;
140                 }
141
142         vector<cv::Point2f> corner_current;
143         bool found_current;
144     cv::Mat rvec_current;
145     cv::Mat tvec_current;
146     cv::Mat rmat_current;
147
148     cv::Mat camMtx = (cv::Mat1d(3,3) << 989.13747, 0, 660.60807, 0,
        989.13747, 519.38931, 0, 0, 1);
149     cv::Mat distCoe =(cv::Mat1d(1,5) << 0.0087, 0.0, 0.00039, 0.01630,
        -0.14183);
150
151         cv::cvtColor(cv_ptr->image, cv_ptr->image, cv::COLOR_BGR2GRAY);
152
```

```
153         found_current = findChessboardCorners(cv_ptr->image,
                chessboardSize, corner_current, cv::CALIB_CB_ADAPTIVE_THRESH
                + cv::CALIB_CB_NORMALIZE_IMAGE);
154
155         if(found_current){
156                 cv::Mat src1 = cv_ptr->image;
157                 cv::cornerSubPix(cv_ptr->image, corner_current,
                        chessboardSize, cv::Size(-1, -1), cv::TermCriteria(cv
                        ::TermCriteria::EPS + cv::TermCriteria::COUNT, 50,
                        0.1));
158
159                 cv::cvtColor(cv_ptr->image, cv_ptr->image, cv::
                        COLOR_GRAY2RGB);
160                 cv::drawChessboardCorners(cv_ptr->image, chessboardSize,
                        corner_current, found_current);
161
162
163                 cv::solvePnP(objPoints, corner_current, camMtx, distCoe,
                        rvec_current, tvec_current, false, CV_EPNP);
164
165                 cv::Rodrigues(rvec_current, rmat_current);
166
167                 markerBroadcaster(rmat_current, tvec_current);
168         }
169
170         cv::imshow(OPENCV_WIN1, cv_ptr->image);
171         cv::waitKey(3);
172
173         image_pub_.publish(cv_ptr->toImageMsg());
174         }
175
176 };
177
178 /**
179  * @brief       main function
180  * @details Initializes ROS
181  * @param       [in] argc
182  * @param       [in] argv
183  * @return      escape sequence
184  */
185 int main(int argc, char** argv){
186         ros::init(argc, argv, "chess_test");
187         ImageConverter ic;
188         ros::spin();
189         return 0;
190 }
```

Listing 2.5: Farbtrennung

```
1  /**
2   ********************************************************************************
3   * @file                color_seperation.cpp
4   * @author              Christian Waldner
5   * @date                19/07/2017
6   * @copyright   2011-2017 UniBwM - ETTI - Institut 4
7   * @brief               Node to seperate a specific color via
        thresholding
8   * @details     This node is a simple Implementation for seperating
        colors to
9   *                              receive a thresholded image
10  ********************************************************************************
11  * @todo                rename: topicname, init-name,
12  * @todo                implement ROS publisher
13  ********************************************************************************
14  *
15  */
16
17 #include <ros/ros.h>
18 #include <cv_bridge/cv_bridge.h>
19 #include <image_transport/image_transport.h>
20 #include <sensor_msgs/image_encodings.h>
21
22 #include <opencv2/highgui/highgui.hpp>
23 #include <opencv2/imgproc/imgproc.hpp>
24
25  /**define for adjusting the erode /dilate filters*/
26 #define FILTER_SIZE_W 2
27 /**define for adjusting the erode /dilate filters*/
28 #define FILTER_SIZE_H 2
29
30
31
32 using namespace cv;
33 using namespace std;
34
35 static const char* OPENCV_WINDOW = "Original Image";
36 static const char* PROCESSED_WINDOW = "Thresholded Image";
37
38
39 int iLowH =  0;
40 int iHighH = 11;
41
42 int iLowS =  78;
43 int iHighS = 255;
44
45 int iLowV =  110;
46 int iHighV = 255;
47
48
49 /** Callbackfunction for the controlpanel button red */
50 void redCb(int state, void *ptr){
```

```
51          cvSetTrackbarPos("LowH", NULL, 000);
52          cvSetTrackbarPos("HighH", NULL, 16);
53          cvSetTrackbarPos("LowS", NULL, 88);
54          cvSetTrackbarPos("HighS", NULL, 255);
55          cvSetTrackbarPos("LowV", NULL, 000);
56          cvSetTrackbarPos("HighV", NULL, 173);
57  }
58
59  /** Callbackfunction for the controlpanel button green */
60  void greenCb(int state, void *ptr){
61          cvSetTrackbarPos("LowH", NULL, 47);
62          cvSetTrackbarPos("HighH", NULL, 74);
63          cvSetTrackbarPos("LowS", NULL, 75);
64          cvSetTrackbarPos("HighS", NULL, 255);
65          cvSetTrackbarPos("LowV", NULL, 71);
66          cvSetTrackbarPos("HighV", NULL, 104);
67  }
68
69  /** Callbackfunction for the controlpanel button blue */
70  void blueCb(int state, void *ptr){
71          cvSetTrackbarPos("LowH", NULL, 89);
72          cvSetTrackbarPos("HighH", NULL, 125);
73          cvSetTrackbarPos("LowS", NULL, 159);
74          cvSetTrackbarPos("HighS", NULL, 255);
75          cvSetTrackbarPos("LowV", NULL, 110);
76          cvSetTrackbarPos("HighV", NULL, 255);
77  }
78
79  /** Callbackfunction for the controlpanel button yellow */
80  void yellowCb(int state, void *ptr){
81          cvSetTrackbarPos("LowH", NULL, 26);
82          cvSetTrackbarPos("HighH", NULL, 38);
83          cvSetTrackbarPos("LowS", NULL, 78);
84          cvSetTrackbarPos("HighS", NULL, 255);
85          cvSetTrackbarPos("LowV", NULL, 110);
86          cvSetTrackbarPos("HighV", NULL, 255);
87  }
88
89
90  /**
91    *     @class          ImageConverter
92    * @brief              ImageConverter
93    * @details    This class converts ROS-images into OpenCV-image format
94    */
95  class ImageConverter{
96          ros::NodeHandle nh_;
97          image_transport::ImageTransport it_;
98          image_transport::Subscriber image_sub_;
99          image_transport::Publisher image_pub_;
100
101
102 public:
103         ImageConverter():it_(nh_){
104                 image_sub_ = it_.subscribe("usb_cam/image_raw", 1, &
                        ImageConverter::imageCb, this);
105
```

```
106                 image_pub_ = it_.advertise("Image_converter/
                        thresholded_video", 1);
107
108                 namedWindow(OPENCV_WINDOW);
109                 namedWindow(PROCESSED_WINDOW);
110
111                 cvMoveWindow(OPENCV_WINDOW, 100, 50);
112                 cvMoveWindow(PROCESSED_WINDOW, 750, 50);
113
114                 //Hue
115                 cvCreateTrackbar("LowH", NULL, &iLowH, 179);
116                 cvCreateTrackbar("HighH", NULL, &iHighH, 179);
117
118                 //Saturation
119                 cvCreateTrackbar("LowS", NULL, &iLowS, 255);
120                 cvCreateTrackbar("HighS", NULL, &iHighS, 255);
121
122                 //Value
123                 cvCreateTrackbar("LowV", NULL, &iLowV, 255);
124                 cvCreateTrackbar("HighV", NULL, &iHighV, 255);
125
126
127                 cvCreateButton("Red", redCb, NULL, CV_PUSH_BUTTON, 0);
128                 cvCreateButton("Green", greenCb, NULL, CV_PUSH_BUTTON, 0)
                        ;
129                 cvCreateButton("Blue", blueCb, NULL, CV_PUSH_BUTTON, 0);
130                 cvCreateButton("Yellow", yellowCb, NULL, CV_PUSH_BUTTON,
                        0);
131         }
132
133         ~ImageConverter(){
134                 destroyWindow(OPENCV_WINDOW);
135         }
136
137 /**
138  * @brief       The image callback function converts the webcam-stream
                        into an OpenCV format. The stream can be modified
139  *                      to get a thresholded image by adjusting the hue,
                        saturation or value.
140  *                      Open the control window by pressing Ctrl + P
141  * @param       [in] msg ROS/sensor_msgs/image_raw
142  * @return      void
143  */
144         void imageCb(const sensor_msgs::ImageConstPtr& msg){
145                 cv_bridge::CvImagePtr cv_ptr;
146                 try{
147                         cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                                image_encodings::BGR8);
148                 }catch(cv_bridge::Exception& e){
149                         ROS_ERROR("cv_bridge exception: %s", e.what());
150                         return;
151                 }
152
153                 Mat imgHSV;
154
155                 cvtColor(cv_ptr->image, imgHSV, COLOR_BGR2HSV);
```

```
156
157                 Mat imgThresholded;
158
159                 inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(
                        iHighH, iHighS, iHighV), imgThresholded);
160
161
162 //              erode(imgThresholded, imgThresholded,
        getStructuringElement(MORPH_ELLIPSE, Size(FILTER_SIZE_W,
        FILTER_SIZE_H)));
163 //              dilate(imgThresholded, imgThresholded,
        getStructuringElement(MORPH_ELLIPSE, Size(FILTER_SIZE_W,
        FILTER_SIZE_H)));
164
165                 dilate(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(2*
                        FILTER_SIZE_W, 2*FILTER_SIZE_H)));
166                 erode(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(2*
                        FILTER_SIZE_W, 2*FILTER_SIZE_H)));
167
168                 imshow(PROCESSED_WINDOW, imgThresholded);
169                 imshow(OPENCV_WINDOW, cv_ptr->image);
170
171                 waitKey(3);
172
173                 //convert and publish thresholded image
174                 cvtColor(imgThresholded, cv_ptr->image, COLOR_GRAY2RGB);
175
176                 image_pub_.publish(cv_ptr->toImageMsg());
177         }
178 };
179
180
181 /**
182  * @brief       main function
183  * @details Initializes ROS and opens the image to be adjusted
184  * @param       [in] argc
185  * @param       [in] argv
186  * @return      escape sequence
187  */
188 int main (int argc, char** argv){
189         ros::init(argc, argv, "image_converter");
190         ImageConverter ic;
191         ros::spin();
192         return 0;
193 }
```

Listing 2.6: Erkennung von Ecken

```
1  /**
2   ********************************************************************************
3   * @file                corner_detection.cpp
4   * @author              Christian Waldner
5   * @date                19/07/2017
6   * @copyright   2011-2017 UniBwM - ETTI - Institut 4
7   * @brief               Node to find sharp corners and features in a
        stream / image
8   * @details     This node returns a graphical output which shows
        goodFeatures to
9   *                              track, like corners and sharp edges
10  ********************************************************************************
11  * @todo                rename: topicname, init-name,
12  * @todo                implement ROS publisher
13  ********************************************************************************
14  *
15  */
16
17 #include <ros/ros.h>
18 #include <cv_bridge/cv_bridge.h>
19 #include <image_transport/image_transport.h>
20 #include <sensor_msgs/image_encodings.h>
21
22 #include <opencv2/opencv.hpp>
23 #include <opencv2/highgui/highgui.hpp>
24 #include <opencv2/imgproc.hpp>
25
26 #include <stdlib.h>
27
28 using namespace cv;
29 using namespace std;
30
31 static const char* OPENCV_WINDOW = "Original Image";
32 static const char* PROCESSED_WINDOW = "Processed Image";
33
34 int max_corners = 20;
35
36 /**
37   *     @class          ImageConverter
38   * @brief               ImageConverter
39   * @details     This class converts ROS-images into OpenCV-image format
40   */
41 class ImageConverter{
42         ros::NodeHandle nh_;
43         image_transport::ImageTransport it_;
44         image_transport::Subscriber image_sub_;
45         image_transport::Publisher image_pub_;
46
47
48 public:
49         ImageConverter():it_(nh_){
50                 image_sub_ = it_.subscribe("usb_cam/image_raw", 1, &
```

```
                ImageConverter::imageCb, this);
51
52              image_pub_ = it_.advertise("Image_converter/
                    thresholded_video", 1); //Topicname anpassen
53
54              namedWindow(OPENCV_WINDOW);
55              namedWindow(PROCESSED_WINDOW);
56
57              cvMoveWindow(OPENCV_WINDOW, 100, 50);
58              cvMoveWindow(PROCESSED_WINDOW, 750, 50);
59
60              //Corners-Trackbar
61              cvCreateTrackbar("Corners", PROCESSED_WINDOW, &
                    max_corners, 250);
62      }
63
64      ~ImageConverter(){
65              destroyWindow(OPENCV_WINDOW);
66      }
67
68 /**
69  * @brief       The image callback function converts the webcam-stream
        into an OpenCV format. Afterwards the function
70  *                      tries to find sharp edges and corners
71  * @param       [in] msg ROS/sensor_msgs/image_raw
72  * @return      void
73  */
74      void imageCb(const sensor_msgs::ImageConstPtr& msg){
75              cv_bridge::CvImagePtr cv_ptr;
76              try{
77                      cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                            image_encodings::BGR8);
78              }catch(cv_bridge::Exception& e){
79                      ROS_ERROR("cv_bridge exception: %s", e.what());
80                      return;
81              }
82
83              Mat image_gray;
84
85              cvtColor(cv_ptr->image, image_gray, CV_RGB2GRAY);
86
87              float quality = 0.01;
88              int min_distance = 10;
89
90              vector<Point2d> corners;
91
92              goodFeaturesToTrack(image_gray, corners, max_corners,
                    quality, min_distance);
93
94
95              Mat image_corners = cv_ptr->image.clone();
96              for(int i = 0; i < corners.size(); i++){
97                      circle(image_corners, corners[i], 4, CV_RGB(255,
                            0, 0), -1);
98              }
99
```

```
100
101                 //Show processed Image
102
103                 imshow(PROCESSED_WINDOW, image_corners);
104
105                 //Show original Image
106
107                 imshow(OPENCV_WINDOW, cv_ptr->image);
108
109                 waitKey(3);
110
111                 //convert and publish thresholded image
112         //      cvtColor(imgThresholded, cv_ptr->image, COLOR_GRAY2RGB);
113
114                 image_pub_.publish(cv_ptr->toImageMsg());
115         }
116 };
117
118 /**
119  * @brief       main function
120  * @details Initializes ROS
121  * @param       [in] argc
122  * @param       [in] argv
123  * @return      escape sequence
124  */
125 int main (int argc, char** argv){
126         ros::init(argc, argv, "image_converter");
127         ImageConverter ic;
128         ros::spin();
129         return 0;
130 }
```

Listing 2.7: Matching via BruteForce

```
1  /**
2   ********************************************************************************
3   * @file                matching_BF.cpp
4   * @author              Christian Waldner
5   * @date                19/07/2017
6   * @copyright   2011-2017 UniBwM - ETTI - Institut 4
7   * @brief               Node to compare a preloaded picture with an USB-
        camera-stream
8   * @details     This node returns a graphical output in which a static
        picture
9   *                              and a live videostream is compared. It'll
         show the common
10  *                              characteristics while the perspective won
        't affect the detection.
11  ********************************************************************************
12  * @todo                rename: topicname, init-name,
13  * @todo                implement ROS publisher
14  ********************************************************************************
15  *
16  */
17
18 //ROS include files
19 #include <ros/ros.h>
20 #include <cv_bridge/cv_bridge.h>
21 #include <image_transport/image_transport.h>
22 #include <sensor_msgs/image_encodings.h>
23
24 //OpenCV include files
25 #include <opencv2/features2d.hpp>
26 #include <opencv2/xfeatures2d.hpp>
27 #include <opencv2/highgui/highgui.hpp>
28 #include <opencv2/imgproc/imgproc.hpp>
29
30
31 #include <cmath>
32
33 using namespace cv;
34 using namespace std;
35
36 static const char* PROCESSED_WINDOW = "Processed Image";
37
38 Mat train; /**<Global matrix used in ImageConverter class and main method
       */
39
40
41 /**
42   *     @class          ImageConverter
43   * @brief              ImageConverter
44   * @details    This class converts ROS-images into OpenCV-image format
45   */
46 class ImageConverter{
47         ros::NodeHandle nh_;
```

```
48          image_transport::ImageTransport it_;
49          image_transport::Subscriber image_sub_;
50          image_transport::Publisher image_pub_;
51
52
53  public:
54          ImageConverter():it_(nh_){
55                  image_sub_ = it_.subscribe("usb_cam/image_raw", 1, &
                        ImageConverter::imageCb, this);
56
57                  image_pub_ = it_.advertise("Image_converter/
                        thresholded_video", 1); //Topicname anpassen
58
59                  namedWindow(PROCESSED_WINDOW, WINDOW_AUTOSIZE);
60          }
61
62          ~ImageConverter(){
63                  destroyWindow(PROCESSED_WINDOW);
64          }
65
66
67  /**
68   * @brief       The image callback function converts the webcam-stream
        into an OpenCV format. Afterwards the converted
69   *                      frame is compared to the previously opened
        picture.
70   *                      Via 'kkn-matching' common characteristics should
        be found, drawn and shown in a new window
71   * @param       [in] msg ROS/sensor_msgs/image_raw
72   * @return      void
73   */
74          void imageCb(const sensor_msgs::ImageConstPtr& msg){
75                  cv_bridge::CvImagePtr cv_ptr;
76
77                  try{
78                          cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                                image_encodings::BGR8);
79                  }catch(cv_bridge::Exception& e){
80                          ROS_ERROR("cv_bridge exception: %s", e.what());
81                          return;
82                  }
83
84
85                  Mat train_g;
86
87                  cvtColor(train, train_g, CV_BGR2GRAY);
88
89                  vector<KeyPoint> train_kp;
90                  Mat train_desc;
91
92                  Ptr<xfeatures2d::SiftFeatureDetector> featureDetector =
                        xfeatures2d::SiftFeatureDetector::create();
93                  featureDetector->detect(train_g, train_kp);
94
95                  Ptr<xfeatures2d::SiftDescriptorExtractor>
                        featureExtractor = xfeatures2d::
```

```
                SiftDescriptorExtractor::create();
96              featureExtractor->compute(train_g, train_kp, train_desc);
97
98              BFMatcher matcher;
99              vector<Mat> train_desc_collection(1, train_desc);
100             matcher.add(train_desc_collection);
101             matcher.train();
102
103             unsigned int frame_count = 0;
104             double t0 = getTickCount();
105             Mat test_g;
106
107             cvtColor(cv_ptr->image, test_g, CV_BGR2GRAY);
108
109             vector<KeyPoint> test_kp;
110             Mat test_desc;
111
112             featureDetector->detect(test_g, test_kp);
113             featureExtractor->compute(test_g, test_kp, test_desc);
114
115             vector<vector<DMatch> > matches;
116             matcher.knnMatch(test_desc, matches, 2);
117
118             vector<DMatch> good_matches;
119
120             for(int i = 0; i < matches.size(); i++){
121
122                     if(matches[i][0].distance < 0.6 * matches[i][1].
                            distance){
123
124                             good_matches.push_back(matches[i][0]);
125                     }
126             }
127
128             double frames = (getTickFrequency() / (getTickCount() -
                    t0));
129
130             //ROS output
131 //          ROS_INFO("Found: %d matcehs \n", good_matches.size());
132 //          ROS_INFO("Framerate = %f \n" , frames) ;
133
134
135             //Console Output
136             cout << "Found " << good_matches.size() << " matches " <<
                     endl;
137             cout << "Framerate = " << getTickFrequency() /(
                    getTickCount() - t0) << endl;
138
139
140             vector<char> matchesMask;
141
142             Mat img_show;
143             drawMatches(cv_ptr->image, test_kp, train, train_kp,
                    good_matches, img_show); //Draws unmatched points as
                    well
144 //          drawMatches(cv_ptr->image, test_kp, train, train_kp,
```

```
        good_matches, img_show, Scalar::all(-1), Scalar::all(-1), matchesMask
        , DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );
145
146                 good_matches.clear();
147
148
149                 //Show processed Image
150                 imshow(PROCESSED_WINDOW, img_show);
151                 waitKey(3);
152
153 //              convert and publish thresholded image
154 //              cvtColor(img_show, cv_ptr->image, COLOR_GRAY2RGB);
155
156                 image_pub_.publish(cv_ptr->toImageMsg());
157         }
158 };
159
160 /**
161  * @brief       main function
162  * @details Initializes ROS and opens the image to be compared
163  * @param       [in] argc
164  * @param       [in] argv
165  * @return      escape sequence
166  */
167 int main (int argc, char** argv){
168         ros::init(argc, argv, "image_converter");
169
170         train = imread("/home/chris/Schreibtisch/test2.jpg");
171
172         ImageConverter ic;
173         ros::spin();
174         return 0;
175 }
```

Listing 2.8: Matching via ORB-Algorithmus

```
1  /**
2   ********************************************************************************
3   * @file                matching_ORB.cpp
4   * @author              Christian Waldner
5   * @date                19/07/2017
6   * @copyright   2011-2017 UniBwM - ETTI - Institut 4
7   * @brief               Node to compare a preloaded picture with an USB-
        camera-stream
8   *                              using the 'Orientet FAST and Rotated
        BRIEF' (ORB)-algortithm
9   * @details     This node returns a graphical output in which a static
        picture
10  *                              and a live videostream is compared. It'll
         show the common
11  *                              characteristics while the perspective won
        't affect the detection.
12  ********************************************************************************
13  * @todo                rename: topicname, init-name,
14  * @todo                implement ROS publisher
15  ********************************************************************************
16  *
17  */
18
19 //ROS include files
20 #include <ros/ros.h>
21 #include <cv_bridge/cv_bridge.h>
22 #include <image_transport/image_transport.h>
23 #include <sensor_msgs/image_encodings.h>
24
25 //OpenCV include files
26 #include <opencv2/features2d.hpp>
27 #include <opencv2/xfeatures2d.hpp>
28 #include <opencv2/highgui/highgui.hpp>
29 #include <opencv2/imgproc/imgproc.hpp>
30
31
32 using namespace cv;
33 using namespace std;
34
35 static const char* PROCESSED_WINDOW = "Processed Image";
36
37 Mat train; /**<Global matrix used in ImageConverter class and main method
       */
38
39
40 /**
41   *     @class          ImageConverter
42   * @brief              ImageConverter
43   * @details    This class converts ROS-images into OpenCV-image format
44   */
45 class ImageConverter{
46         ros::NodeHandle nh_;
```

```
47          image_transport::ImageTransport it_;
48          image_transport::Subscriber image_sub_;
49          image_transport::Publisher image_pub_;
50
51
52  public:
53          ImageConverter():it_(nh_){
54                  image_sub_ = it_.subscribe("usb_cam/image_raw", 1, &
                        ImageConverter::imageCb, this);
55
56                  image_pub_ = it_.advertise("Image_converter/
                        thresholded_video", 1); //Topicname anpassen
57
58                  namedWindow(PROCESSED_WINDOW, WINDOW_AUTOSIZE);
59          }
60
61          ~ImageConverter(){
62                  destroyWindow(PROCESSED_WINDOW);
63          }
64
65
66  /**
67   * @brief       The image callback function converts the webcam-stream
        into an OpenCV format. Afterwards the converted
68   *                      frame is compared to the previously opened
        picture.
69   *                      Via 'kkn-matching' common characteristics should
        be found, drawn and shown in a new window
70   * @param       [in] msg ROS/sensor_msgs/image_raw
71   * @return      void
72   */
73          void imageCb(const sensor_msgs::ImageConstPtr& msg){
74                  cv_bridge::CvImagePtr cv_ptr;
75
76                  try{
77                          cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                                image_encodings::BGR8);
78                  }catch(cv_bridge::Exception& e){
79                          ROS_ERROR("cv_bridge exception: %s", e.what());
80                          return;
81                  }
82
83
84                  Mat train_g;
85
86                  cvtColor(train, train_g, CV_BGR2GRAY);
87
88                  vector<KeyPoint> train_kp;
89                  Mat train_desc;
90
91                  Ptr<FeatureDetector> featureDetector = ORB::create();
92                  featureDetector->detect(train_g, train_kp);
93
94
95
96                  Ptr<DescriptorExtractor> featureExtractor = ORB::create()
```

```
                ;
97              featureExtractor->compute(train_g, train_kp, train_desc);
98
99              flann::Index flannIndex(train_desc, flann::LshIndexParams
                    (12, 20, 2), cvflann::FLANN_DIST_HAMMING);
100
101             unsigned int frame_count = 0;
102             double t0 = getTickCount();
103             Mat test_g;
104
105             cvtColor(cv_ptr->image, test_g, CV_BGR2GRAY);
106
107
108
109             vector<KeyPoint> test_kp;
110             Mat test_desc;
111
112
113             featureDetector->detect(test_g, test_kp);
114             featureExtractor->compute(test_g, test_kp, test_desc);
115
116
117             Mat match_idx(test_desc.rows, 2, CV_32SC1), match_dist(
                    test_desc.rows, 2, CV_32FC1);
118
119
120             if(test_desc.cols > 0){
121                     flannIndex.knnSearch(test_desc, match_idx,
                            match_dist, 2, flann::SearchParams());
122
123             }else{
124                     ROS_ERROR("No points to match, image frozen");
125             }
126
127
128
129             vector<DMatch> good_matches;
130
131             for(int i = 0; i < match_dist.rows; i++){
132                     if(match_dist.at<float>(i, 0) < 0.6 * match_dist.
                            at<float>(i, 1)){
133                             DMatch dm(i, match_idx.at<int>(i, 0),
                                    match_dist.at<float>(i, 0));
134
135                             good_matches.push_back(dm);
136                     }
137             }
138
139             double frames = (getTickFrequency() / (getTickCount() -
                    t0));
140
141             //ROS output
142 //          ROS_INFO("Found: %d matches \n", good_matches.size());
143 //          ROS_INFO("Framerate = %f \n" , frames) ;
144
145
```

```
146                 //Console Output
147                 cout << "Found " << good_matches.size() << " matches " <<
                         endl;
148                 cout << "Framerate = " << getTickFrequency() /(
                        getTickCount() - t0) << endl <<endl;
149
150
151
152                 vector<char> matchesMask;
153
154                 Mat img_show;
155
156
157
158                 if(good_matches.size() > 0){
159
160 //              drawMatches(cv_ptr->image, test_kp, train, train_kp,
        good_matches, img_show);
161                 drawMatches(cv_ptr->image, test_kp, train, train_kp,
                        good_matches, img_show, Scalar::all(-1), Scalar::all
                        (-1), matchesMask, DrawMatchesFlags::
                        NOT_DRAW_SINGLE_POINTS );
162 //
                                                Scalar(123,213,15)
163
164                 good_matches.clear();
165
166                 //Show processed Image
167                 imshow(PROCESSED_WINDOW, img_show);
168                 waitKey(3);
169                 }
170
171
172                 cv_ptr->image = img_show;
173
174                 image_pub_.publish(cv_ptr->toImageMsg());
175         }
176 };
177
178 /**
179  * @brief       main function
180  * @details Initializes ROS and opens the image to be compared
181  * @param       [in] argc
182  * @param       [in] argv
183  * @return      escape sequence
184  */
185 int main (int argc, char** argv){
186         ros::init(argc, argv, "image_converter");
187
188         train = imread("/home/chris/Schreibtisch/test2.jpg");
189
190         ImageConverter ic;
191         ros::spin();
192         return 0;
193 }
```

Listing 2.9: Formerkennung

```
/**
 ******************************************************************************

 * @file                shape_detection.cpp
 * @author              Christian Waldner
 * @date                19/07/2017
 * @copyright   2011-2017 UniBwM - ETTI - Institut 4
 * @brief               Node which finds corners and overlays contours
     with dirfferent colors
 *                              depending on the number of corners
 * @details     This node returns a graphical output in which different
     shapes will
 *                              be contoured with different colors
 ******************************************************************************

 * @todo                rename: topicname, init-name,
 * @todo                implement ROS publisher
 ******************************************************************************

 *
 */
#include <ros/ros.h>
#include <cv_bridge/cv_bridge.h>
#include <image_transport/image_transport.h>
#include <sensor_msgs/image_encodings.h>

#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>


#include "stdlib.h"

/**define for adjusting the erode /dilate filters*/
#define FILTER_SIZE_W 5
/**define for adjusting the erode /dilate filters*/
#define FILTER_SIZE_H 5

using namespace cv;
using namespace std;

static const string OPENCV_WINDOW = "Image window";

int iThresh = 128;
int iMax = 0;


Mat imgLines;/**<Global matrix used in ImageConverter*/

/**
  *     @class          ImageConverter
  * @brief              ImageConverter
  * @details    This class converts ROS-images into OpenCV-image format
  */
class ImageConverter{
        ros::NodeHandle nh_;
```

```
51          image_transport::ImageTransport it_;
52          image_transport::Subscriber image_sub_;
53          image_transport::Publisher image_pub_;
54
55
56  public:
57          ImageConverter():it_(nh_){
58                  image_sub_ = it_.subscribe("usb_cam/image_raw", 1, &
                        ImageConverter::imageCb, this);
59                  image_pub_ = it_.advertise("Image_converter/output_video"
                        , 1);
60
61                  namedWindow(OPENCV_WINDOW);
62                  cvCreateTrackbar("Threshold", "Image window", &iThresh,
                        255);
63                  cvCreateTrackbar("Threshmode", "Image window", &iMax, 4);
64          }
65
66          ~ImageConverter(){
67                  destroyWindow(OPENCV_WINDOW);
68          }
69
70  /**
71   * @brief       The image callback function converts the webcam-stream
         into an OpenCV format. Afterwards the converted
72   *                      frame is thresholded and the corners of the
         shapes will be counted and contoured
73   * @param       [in] msg ROS/sensor_msgs/image_raw
74   * @return      void
75  */
76          void imageCb(const sensor_msgs::ImageConstPtr& msg){
77                  cv_bridge::CvImagePtr cv_ptr;
78                  try{
79                          cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                                image_encodings::BGR8);
80                  }catch(cv_bridge::Exception& e){
81                          ROS_ERROR("cv_bridge exception: %s", e.what());
82                          return;
83                  }
84                  imgLines = Mat::zeros(cv_ptr->image.rows, cv_ptr->image.
                        cols, CV_8UC3);
85
86                  Mat tmp = cv_ptr->image;
87                  IplImage copy = tmp;
88                  IplImage* img = &copy;
89                  IplImage* imgGrayScale = cvCreateImage(cvGetSize(img), 8,
                         1);
90
91                  cvCvtColor(img, imgGrayScale, CV_BGR2GRAY);
92                  cvThreshold(imgGrayScale, imgGrayScale, iThresh, 255,
                        iMax); //128, 255, CV_THRESH_BINARY
93
94                  CvSeq* contours;
95                  CvSeq* result;
96                  CvMemStorage *storage = cvCreateMemStorage(0);
97
```

```
98              cvFindContours(imgGrayScale, storage, &contours, sizeof(
                    CvContour), CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE,
                    cvPoint(0,0));
99
100             while(contours){
101
102                     result = cvApproxPoly(contours, sizeof(CvContour)
                            , storage, CV_POLY_APPROX_DP,
                            cvContourPerimeter(contours) * 0.01, 0); //
                            0.02
103
104                     if(result->total == 3){
105                             CvPoint *pt[3];
106
107                             for(int i = 0; i < 3; i++){
108                                     pt[i] = (CvPoint*)cvGetSeqElem(
                                            result, i);
109                             }
110                             line(imgLines, *pt[0], *pt[1], cvScalar
                                    (255, 0, 0), 4);
111                             line(imgLines, *pt[1], *pt[2], cvScalar
                                    (255, 0, 0), 4);
112                             line(imgLines, *pt[2], *pt[0], cvScalar
                                    (255, 0, 0), 4);
113                     }
114
115                     else if(result->total == 4){
116                             CvPoint *pt[4];
117
118                             for(int i = 0; i < 4; i++){
119                                     pt[i] = (CvPoint*)cvGetSeqElem(
                                            result, i);
120                             }
121                             line(imgLines, *pt[0], *pt[1], cvScalar
                                    (0, 255, 0), 4);
122                             line(imgLines, *pt[1], *pt[2], cvScalar
                                    (0, 255, 0), 4);
123                             line(imgLines, *pt[2], *pt[3], cvScalar
                                    (0, 255, 0), 4);
124                             line(imgLines, *pt[3], *pt[0], cvScalar
                                    (0, 255, 0), 4);
125                     }
126
127                     else if(result->total == 6){
128                             CvPoint *pt[6];
129
130                             for(int i = 0; i < 6; i++){
131                                     pt[i] = (CvPoint*)cvGetSeqElem(
                                            result, i);
132                             }
133                             line(imgLines, *pt[0], *pt[1], cvScalar
                                    (255, 255, 0), 4);
134                             line(imgLines, *pt[1], *pt[2], cvScalar
                                    (255, 255, 0), 4);
135                             line(imgLines, *pt[2], *pt[3], cvScalar
                                    (255, 255, 0), 4);
```

```
136                                     line(imgLines, *pt[3], *pt[4], cvScalar
                                            (255, 255, 0), 4);
137                                     line(imgLines, *pt[4], *pt[5], cvScalar
                                            (255, 255, 0), 4);
138                                     line(imgLines, *pt[5], *pt[0], cvScalar
                                            (255, 255, 0), 4);
139                             }
140
141                             else if(result->total == 7){
142                                     CvPoint *pt[7];
143
144                                     for(int i = 0; i < 7; i++){
145                                             pt[i] = (CvPoint*)cvGetSeqElem(
                                                    result, i);
146                                     }
147                                     line(imgLines, *pt[0], *pt[1], cvScalar
                                            (0, 0, 255), 4);
148                                     line(imgLines, *pt[1], *pt[2], cvScalar
                                            (0, 0, 255), 4);
149                                     line(imgLines, *pt[2], *pt[3], cvScalar
                                            (0, 0, 255), 4);
150                                     line(imgLines, *pt[3], *pt[4], cvScalar
                                            (0, 0, 255), 4);
151                                     line(imgLines, *pt[4], *pt[5], cvScalar
                                            (0, 0, 255), 4);
152                                     line(imgLines, *pt[5], *pt[6], cvScalar
                                            (0, 0, 255), 4);
153                                     line(imgLines, *pt[6], *pt[0], cvScalar
                                            (0, 0, 255), 4);
154                             }
155                             contours = contours->h_next;
156                     }
157
158                     cv_ptr->image = cv_ptr->image + imgLines;
159
160                     imshow(OPENCV_WINDOW, cv_ptr->image);
161
162                     if(waitKey(30) == 27){
163                             ROS_INFO_STREAM("ESC pressed by user");
164                             cvDestroyAllWindows();
165                             cvReleaseMemStorage(&storage);
166                             cvReleaseImage(&img);
167                             cvReleaseImage(&imgGrayScale);
168                             exit(1);
169                     }
170
171                     image_pub_.publish(cv_ptr->toImageMsg());
172             }
173 };
174
175 /**
176  * @brief       main function
177  * @details Initializes ROS
178  * @param       [in] argc
179  * @param       [in] argv
180  * @return      escape sequence
```

```
181  */
182 int main (int argc, char** argv){
183         ros::init(argc, argv, "image_converter");
184         ImageConverter ic;
185         ros::spin();
186         return 0;
187 }
```

Listing 2.10: Formerkennung mit Farbwahl

```
1  /**
2   ********************************************************************************
3   * @file                shape_detection2.cpp
4   * @author              Christian Waldner
5   * @date                19/07/2017
6   * @copyright   2011-2017 UniBwM - ETTI - Institut 4
7   * @brief               Node which finds corners and overlays contours
        with dirfferent colors
8   *                              depending on the number of corners
9   * @details     This node returns a graphical output in which different
        shapes will
10  *                              be contoured with different colors
11  *                              Difference to 1st Version: Thresholding
        can be adjustet to select
12  *                              colors
13  ********************************************************************************
14  * @todo                rename: topicname, init-name,
15  * @todo                implement ROS publisher
16  ********************************************************************************
17  */
18 #include <ros/ros.h>
19 #include <cv_bridge/cv_bridge.h>
20 #include <image_transport/image_transport.h>
21 #include <sensor_msgs/image_encodings.h>
22
23 #include <opencv2/highgui/highgui.hpp>
24 #include <opencv2/imgproc/imgproc.hpp>
25
26
27 #include "stdlib.h"
28
29 /**define for adjusting the erode /dilate filters*/
30 #define FILTER_SIZE_W 6
31 /**define for adjusting the erode /dilate filters*/
32 #define FILTER_SIZE_H 6
33
34
35 using namespace cv;
36 using namespace std;
37
38 static const char* OPENCV_WINDOW = "Original Image";
39 static const char* PROCESSED_WINDOW = "Thresholded Image";
40
41 int iThresh = 128;
42 int iMax = 0;
43
44 int iLowH =  0;          //0
45 int iHighH = 11;         //179
46
47 int iLowS =  78;         //0
48 int iHighS = 255;        //255
49
```

```
50 int iLowV =  110;        //0
51 int iHighV = 255;        //255
52
53 /** Callbackfunction for the controlpanel button red */
54 void redCb(int state, void *ptr){
55         cvSetTrackbarPos("LowH",  NULL, 000);
56         cvSetTrackbarPos("HighH", NULL, 16);
57         cvSetTrackbarPos("LowS",  NULL, 88);
58         cvSetTrackbarPos("HighS", NULL, 255);
59         cvSetTrackbarPos("LowV",  NULL, 000);
60         cvSetTrackbarPos("HighV", NULL, 173);
61 }
62
63 /** Callbackfunction for the controlpanel button green */
64 void greenCb(int state, void *ptr){
65         cvSetTrackbarPos("LowH",  NULL, 47);
66         cvSetTrackbarPos("HighH", NULL, 74);
67         cvSetTrackbarPos("LowS",  NULL, 75);
68         cvSetTrackbarPos("HighS", NULL, 255);
69         cvSetTrackbarPos("LowV",  NULL, 71);
70         cvSetTrackbarPos("HighV", NULL, 104);
71 }
72
73 /** Callbackfunction for the controlpanel button blue */
74 void blueCb(int state, void *ptr){
75         cvSetTrackbarPos("LowH",  NULL, 89);
76         cvSetTrackbarPos("HighH", NULL, 125);
77         cvSetTrackbarPos("LowS",  NULL, 159);
78         cvSetTrackbarPos("HighS", NULL, 255);
79         cvSetTrackbarPos("LowV",  NULL, 110);
80         cvSetTrackbarPos("HighV", NULL, 255);
81 }
82
83 /** Callbackfunction for the controlpanel button yellow */
84 void yellowCb(int state, void *ptr){
85         cvSetTrackbarPos("LowH",  NULL, 26);
86         cvSetTrackbarPos("HighH", NULL, 38);
87         cvSetTrackbarPos("LowS",  NULL, 78);
88         cvSetTrackbarPos("HighS", NULL, 255);
89         cvSetTrackbarPos("LowV",  NULL, 110);
90         cvSetTrackbarPos("HighV", NULL, 255);
91 }
92
93
94
95 Mat imgLines;/**<Global matrix used in ImageConverter*/
96
97 /**
98   *     @class          ImageConverter
99   * @brief              ImageConverter
100  * @details    This class converts ROS-images into OpenCV-image format
101  */
102 class ImageConverter{
103         ros::NodeHandle nh_;
104         image_transport::ImageTransport it_;
105         image_transport::Subscriber image_sub_;
```

```
106
107         image_transport::Subscriber image_sub2_;
108
109         image_transport::Publisher image_pub_;
110
111
112 public:
113         ImageConverter():it_(nh_){
114
115                 image_sub2_ = it_.subscribe("usb_cam/image_raw", 1, &
                        ImageConverter::imageThreshCb, this);
116
117                 image_pub_ = it_.advertise("Image_converter/
                        shape_detection_video", 1);
118
119                 namedWindow(OPENCV_WINDOW);
120                 cvCreateTrackbar("Threshold", "Image window", &iThresh,
                        255);
121                 cvCreateTrackbar("Threshmode", "Image window", &iMax, 4);
122
123
124                 namedWindow(OPENCV_WINDOW);
125                 namedWindow(PROCESSED_WINDOW);
126
127                 cvMoveWindow(OPENCV_WINDOW, 100, 50);
128                 cvMoveWindow(PROCESSED_WINDOW, 750, 50);
129
130                 //Hue
131                 cvCreateTrackbar("LowH", NULL, &iLowH, 179);
132                 cvCreateTrackbar("HighH", NULL, &iHighH, 179);
133
134                 //Saturation
135                 cvCreateTrackbar("LowS", NULL, &iLowS, 255);
136                 cvCreateTrackbar("HighS", NULL, &iHighS, 255);
137
138                 //Value
139                 cvCreateTrackbar("LowV", NULL, &iLowV, 255);
140                 cvCreateTrackbar("HighV", NULL, &iHighV, 255);
141
142
143                 cvCreateButton("Red", redCb, NULL, CV_PUSH_BUTTON, 0);
144                 cvCreateButton("Green", greenCb, NULL, CV_PUSH_BUTTON, 0)
                        ;
145                 cvCreateButton("Blue", blueCb, NULL, CV_PUSH_BUTTON, 0);
146                 cvCreateButton("Yellow", yellowCb, NULL, CV_PUSH_BUTTON,
                        0);
147         }
148
149         ~ImageConverter(){
150                 destroyWindow(OPENCV_WINDOW);
151         }
152
153         void imageThreshCb(const sensor_msgs::ImageConstPtr& msg){
154                         cv_bridge::CvImagePtr cv_ptr;
155
156                         try{
```

```
157                         cv_ptr = cv_bridge::toCvCopy(msg,
                                sensor_msgs::image_encodings::BGR8);
158                     }catch(cv_bridge::Exception& e){
159                         ROS_ERROR("cv_bridge exception: %s", e.
                                what());
160                         return;
161                     }
162
163                     Mat imgHSV;
164
165                     cvtColor(cv_ptr->image, imgHSV, COLOR_BGR2HSV);
166
167                     Mat imgThresholded;
168
169                     inRange(imgHSV, Scalar(iLowH, iLowS, iLowV),
                            Scalar(iHighH, iHighS, iHighV),
                            imgThresholded);
170
171
172                 //      erode(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(
                        FILTER_SIZE_W, FILTER_SIZE_H)));
173                 //      dilate(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(
                        FILTER_SIZE_W, FILTER_SIZE_H)));
174
175                     dilate(imgThresholded, imgThresholded,
                            getStructuringElement(MORPH_ELLIPSE, Size(
                            FILTER_SIZE_W, FILTER_SIZE_H)));
176                     erode(imgThresholded, imgThresholded,
                            getStructuringElement(MORPH_ELLIPSE, Size(
                            FILTER_SIZE_W, FILTER_SIZE_H)));
177
178
179                     imshow(PROCESSED_WINDOW, imgThresholded);
180
181                     imgLines = Mat::zeros(cv_ptr->image.rows, cv_ptr
                            ->image.cols, CV_8UC3);
182
183                     Mat tmp;
184                     cvtColor(imgThresholded, tmp, COLOR_GRAY2RGB);
185
186                     IplImage copy = imgThresholded; //tmp;
187                     IplImage* img = &copy;
188
189                     CvSeq* contours;
190                     CvSeq* result;
191                     CvMemStorage *storage = cvCreateMemStorage(0);
192
193                     cvFindContours(img, storage, &contours, sizeof(
                            CvContour), CV_RETR_LIST,
                            CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0));
194
195                     while(contours){
196
197                         result = cvApproxPoly(contours, sizeof(
```

```
                    CvContour), storage,
                    CV_POLY_APPROX_DP, cvContourPerimeter
                    (contours) * 0.02, 0); //0.02
198
199                 if(result->total == 3){
200                         CvPoint *pt[3];
201
202                         for(int i = 0; i < 3; i++){
203                                 pt[i] = (CvPoint*)
                                        cvGetSeqElem(result,
                                        i);
204                         }
205
206                         line(imgLines, *pt[0], *pt[1],
                                cvScalar(255, 0, 0), 4);
207                         line(imgLines, *pt[1], *pt[2],
                                cvScalar(255, 0, 0), 4);
208                         line(imgLines, *pt[2], *pt[0],
                                cvScalar(255, 0, 0), 4);
209                 }
210
211                 else if(result->total == 4){
212                         CvPoint *pt[4];
213
214                         for(int i = 0; i < 4; i++){
215                                 pt[i] = (CvPoint*)
                                        cvGetSeqElem(result,
                                        i);
216                         }
217
218                         line(imgLines, *pt[0], *pt[1],
                                cvScalar(0, 255, 0), 4);
219                         line(imgLines, *pt[1], *pt[2],
                                cvScalar(0, 255, 0), 4);
220                         line(imgLines, *pt[2], *pt[3],
                                cvScalar(0, 255, 0), 4);
221                         line(imgLines, *pt[3], *pt[0],
                                cvScalar(0, 255, 0), 4);
222                 }
223
224                 else if(result->total == 6){
225                         CvPoint *pt[6];
226
227                         for(int i = 0; i < 6; i++){
228                                 pt[i] = (CvPoint*)
                                        cvGetSeqElem(result,
                                        i);
229                         }
230
231                         line(imgLines, *pt[0], *pt[1],
                                cvScalar(255, 255, 0), 4);
232                         line(imgLines, *pt[1], *pt[2],
                                cvScalar(255, 255, 0), 4);
233                         line(imgLines, *pt[2], *pt[3],
                                cvScalar(255, 255, 0), 4);
234                         line(imgLines, *pt[3], *pt[4],
```

```
                                        cvScalar(255, 255, 0), 4);
235                                 line(imgLines, *pt[4], *pt[5],
                                        cvScalar(255, 255, 0), 4);
236                                 line(imgLines, *pt[5], *pt[0],
                                        cvScalar(255, 255, 0), 4);
237                         }
238
239                         else if(result->total == 7){
240                                 CvPoint *pt[7];
241
242                                 for(int i = 0; i < 7; i++){
243                                         pt[i] = (CvPoint*)
                                                cvGetSeqElem(result,
                                                i);
244                                 }
245
246                                 line(imgLines, *pt[0], *pt[1],
                                        cvScalar(0, 0, 255), 4);
247                                 line(imgLines, *pt[1], *pt[2],
                                        cvScalar(0, 0, 255), 4);
248                                 line(imgLines, *pt[2], *pt[3],
                                        cvScalar(0, 0, 255), 4);
249                                 line(imgLines, *pt[3], *pt[4],
                                        cvScalar(0, 0, 255), 4);
250                                 line(imgLines, *pt[4], *pt[5],
                                        cvScalar(0, 0, 255), 4);
251                                 line(imgLines, *pt[5], *pt[6],
                                        cvScalar(0, 0, 255), 4);
252                                 line(imgLines, *pt[6], *pt[0],
                                        cvScalar(0, 0, 255), 4);
253                         }
254
255                 contours = contours->h_next;
256                 }
257
258                 //Originalbild
259                 cv_ptr->image = cv_ptr->image + imgLines;
260                 imshow(OPENCV_WINDOW, cv_ptr->image);
261
262                 waitKey(3);
263
264                 //convert and publish thresholded image
265                 cvtColor(imgThresholded, cv_ptr->image,
                        COLOR_GRAY2RGB);
266
267                 image_pub_.publish(cv_ptr->toImageMsg());
268         }
269 };
270
271 /**
272  * @brief       main function
273  * @details Initializes ROS
274  * @param       [in] argc
275  * @param       [in] argv
276  * @return      escape sequence
277  */
```

```
278 int main (int argc, char** argv){
279         ros::init(argc, argv, "image_converter_shape");
280         ImageConverter ic;
281         ros::spin();
282         return 0;
283 }
```

Listing 2.11: Tracking mit Zielkreis

```
1  /**
2   ********************************************************************************
3   * @file                tracking_circle.cpp
4   * @author              Christian Waldner
5   * @date                19/07/2017
6   * @copyright   2011-2017 UniBwM - ETTI - Institut 4
7   * @brief               Node which draws a red Circle around the
        geometrical centre of a
8   *                              selected color
9   * @details     This node returns a graphical output in which a circle is
        drawn
10  *                              around the centre of a selected color.
        Adjust the parameters to
11  *                              get the best results: Ctrl + P
12  ********************************************************************************
13  * @todo                rename: topicname, init-name,
14  * @todo                implement ROS publisher
15  ********************************************************************************
16  *
17  */
18 #include <ros/ros.h>
19 #include <cv_bridge/cv_bridge.h>
20 #include <image_transport/image_transport.h>
21 #include <sensor_msgs/image_encodings.h>
22
23 #include <opencv2/highgui/highgui.hpp>
24 #include <opencv2/imgproc/imgproc.hpp>
25
26
27 #include "stdlib.h"
28
29 /**define for adjusting the erode /dilate filters*/
30 #define FILTER_SIZE_W 5
31 /**define for adjusting the erode /dilate filters*/
32 #define FILTER_SIZE_H 5
33
34
35 using namespace cv;
36 using namespace std;
37
38 static const string OPENCV_WINDOW = "Image window";
39                                             //DEFAULT       //RED
40 int iLowH =  0;         //0                       0
41 int iHighH = 15;        //179             19
42
43 int iLowS =  89;        //0                       128
44 int iHighS = 255;       //255             255
45
46 int iLowV = 61;         //0                       37
47 int iHighV = 255;       //255             255
48
49 int iLastX = -1;
```

```
50 int iLastY = -1;
51
52 int setup = 0;
53
54 Mat imgLines; /**<Global matrix used in ImageConverter*/
55
56
57 /**
58   *     @class          ImageConverter
59   * @brief              ImageConverter
60   * @details    This class converts ROS-images into OpenCV-image format
61   */
62 class ImageConverter{
63         ros::NodeHandle nh_;
64         image_transport::ImageTransport it_;
65         image_transport::Subscriber image_sub_;
66         image_transport::Publisher image_pub_;
67
68
69 public:
70         ImageConverter():it_(nh_){
71                 image_sub_ = it_.subscribe("usb_cam/image_raw", 1, &
                        ImageConverter::imageCb, this);
72
73                 image_pub_ = it_.advertise("Image_converter/output_video"
                        , 1);
74
75                 namedWindow(OPENCV_WINDOW);
76
77                 namedWindow("Control", CV_WINDOW_AUTOSIZE);
78
79
80                 cvCreateTrackbar("LowH", "Control", &iLowH, 179);
81                 cvCreateTrackbar("HighH", "Control", &iHighH, 179);
82
83                 cvCreateTrackbar("LowS", "Control", &iLowS, 255);
84                 cvCreateTrackbar("HighS", "Control", &iHighS, 255);
85
86                 cvCreateTrackbar("LowV", "Control", &iLowV, 255);
87                 cvCreateTrackbar("HighV", "Control", &iHighV, 255); //
                        Erzeugt Speicherzugriffsfehler ?!
88         }
89
90         ~ImageConverter(){
91                 destroyWindow(OPENCV_WINDOW);
92         }
93
94         void imageCb(const sensor_msgs::ImageConstPtr& msg){
95                 cv_bridge::CvImagePtr cv_ptr;
96                 try{
97                         cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                                image_encodings::BGR8);
98                 }catch(cv_bridge::Exception& e){
99                         ROS_ERROR("cv_bridge exception: %s", e.what());
100                        return;
101                }
```

```
102
103                 imgLines = Mat::zeros(cv_ptr->image.rows, cv_ptr->image.
                        cols, CV_8UC3);
104
105                 Mat imgHSV;
106
107                 cvtColor(cv_ptr->image, imgHSV, COLOR_BGR2HSV);
108
109                 Mat imgThresholded;
110
111                 inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(
                        iHighH, iHighS, iHighV), imgThresholded);
112
113                 erode(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(
                        FILTER_SIZE_W, FILTER_SIZE_H)));
114                 dilate(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(
                        FILTER_SIZE_W, FILTER_SIZE_H)));
115
116                 dilate(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(2*
                        FILTER_SIZE_W, 2*FILTER_SIZE_H)));
117                 erode(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(2*
                        FILTER_SIZE_W, 2*FILTER_SIZE_H)));
118
119
120                 Moments oMoments = moments(imgThresholded);
121
122                 double dM01 = oMoments.m01;
123                 double dM10 = oMoments.m10;
124                 double dArea = oMoments.m00;
125
126                 if(dArea > 10000){
127                         int posX = dM10 /dArea;
128                         int posY = dM01 /dArea;
129
130                         if(iLastX >= 0 && iLastY >= 0 && posX >= 0 &&
                                posY >= 0){
131                                 circle(imgLines, Point(posX, posY), 50,
                                        Scalar(0,0,255), 2);
132                                 ROS_INFO("X-Pos: %d, Y-Pos %d", posX,
                                        posY);
133                         }
134                         iLastX = posX;
135                         iLastY = posY;
136                 }
137                 cv_ptr->image = cv_ptr->image + imgLines;
138
139                 imshow("Thresholded Image", imgThresholded);
140                 imshow(OPENCV_WINDOW, cv_ptr->image);
141
142                 if(waitKey(30) == 27){
143                         ROS_INFO_STREAM("ESC pressed by user");
144                         exit(1);
```

```
145                 }
146
147                 image_pub_.publish(cv_ptr->toImageMsg());
148         }
149 };
150
151 /**
152  * @brief       main function
153  * @details Initializes ROS
154  * @param       [in] argc
155  * @param       [in] argv
156  * @return      escape sequence
157  */
158 int main (int argc, char** argv){
159         ros::init(argc, argv, "image_converter");
160         ImageConverter ic;
161         ros::spin();
162         return 0;
163 }
```

Listing 2.12: Tracking mit Linie

```
1  /**
2   ********************************************************************************
3   * @file                tracking_line.cpp
4   * @author              Christian Waldner
5   * @date                19/07/2017
6   * @copyright   2011-2017 UniBwM - ETTI - Institut 4
7   * @brief               Node which draws a red line along the geometrical
        centre of a
8   *                              selected color, while it's moved
9   * @details     This node returns a graphical output in which a line is
        drawn
10  *                              along the centre of a selected color when
         it's moved. Adjust the
11  *                              parameters to get the best results: Ctrl
        + P.
12  *                              By uncommenting the commented lines a
        autoclean is activated
13  ********************************************************************************
14  * @todo                rename: topicname, init-name,
15  * @todo                implement ROS publisher
16  ********************************************************************************
17  */
18 #include <ros/ros.h>
19 #include <cv_bridge/cv_bridge.h>
20 #include <image_transport/image_transport.h>
21 #include <sensor_msgs/image_encodings.h>
22
23 #include <opencv2/highgui/highgui.hpp>
24 #include <opencv2/imgproc/imgproc.hpp>
25
26
27 #include "stdlib.h"
28
29 /**define for adjusting the erode /dilate filters*/
30 #define FILTER_SIZE_W 5
31 /**define for adjusting the erode /dilate filters*/
32 #define FILTER_SIZE_H 5
33
34
35 using namespace cv;
36 using namespace std;
37
38
39 static const string OPENCV_WINDOW = "Image␣window";
40
41 int iLowH =  0;
42 int iHighH = 23;
43
44 int iLowS =  29;
45 int iHighS = 249;
46
47 int iLowV =  0;
```

```
48 int iHighV = 255;
49
50 int iLastX = -1;
51 int iLastY = -1;
52
53 int setup = 0;
54
55 Mat imgLines; /**<Global matrix used in ImageConverter*/
56
57
58 /**
59   *     @class          ImageConverter
60   * @brief              ImageConverter
61   * @details    This class converts ROS-images into OpenCV-image format
62   */
63 class ImageConverter{
64         ros::NodeHandle nh_;
65         image_transport::ImageTransport it_;
66         image_transport::Subscriber image_sub_;
67         image_transport::Publisher image_pub_;
68
69
70 public:
71         ImageConverter():it_(nh_){
72                 image_sub_ = it_.subscribe("usb_cam/image_raw", 1, &
                        ImageConverter::imageCb, this);
73
74                 image_pub_ = it_.advertise("Image_converter/output_video"
                        , 1);
75
76                 namedWindow(OPENCV_WINDOW);
77
78                 namedWindow("Control", CV_WINDOW_AUTOSIZE);
79
80
81                 cvCreateTrackbar("LowH", "Control", &iLowH, 179);
82                 cvCreateTrackbar("HighH", "Control", &iHighH, 179);
83
84                 cvCreateTrackbar("LowS", "Control", &iLowS, 255);
85                 cvCreateTrackbar("HighS", "Control", &iHighS, 255);
86
87                 cvCreateTrackbar("LowV", "Control", &iLowV, 255);
88                 cvCreateTrackbar("HighV", "Control", &iHighV, 255); //
                        Erzeugt Speicherzugriffsfehler ?!
89
90         }
91
92         ~ImageConverter(){
93                 destroyWindow(OPENCV_WINDOW);
94         }
95
96         void imageCb(const sensor_msgs::ImageConstPtr& msg){
97                 cv_bridge::CvImagePtr cv_ptr;
98                 try{
99                         cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                                image_encodings::BGR8);
```

```
100                 }catch(cv_bridge::Exception& e){
101                         ROS_ERROR("cv_bridge exception: %s", e.what());
102                         return;
103                 }
104
105                 if(setup%250 == 0){ //for autoclean (setup%100 == 0)
106                         imgLines = Mat::zeros(cv_ptr->image.rows, cv_ptr
                                ->image.cols, CV_8UC3);
107                         //setup = 1; //comment for autoclean
108                 }
109
110                 Mat imgHSV;
111
112                 cvtColor(cv_ptr->image, imgHSV, COLOR_BGR2HSV);
113
114                 Mat imgThresholded;
115
116                 inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(
                        iHighH, iHighS, iHighV), imgThresholded);
117
118                 erode(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(
                        FILTER_SIZE_W, FILTER_SIZE_H)));
119                 dilate(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(
                        FILTER_SIZE_W, FILTER_SIZE_H)));
120
121                 dilate(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(2*
                        FILTER_SIZE_W, 2*FILTER_SIZE_H)));
122                 erode(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(2*
                        FILTER_SIZE_W, 2*FILTER_SIZE_H)));
123
124                 //Movement detection
125
126                 Moments oMoments = moments(imgThresholded);
127
128                 double dM01 = oMoments.m01;
129                 double dM10 = oMoments.m10;
130                 double dArea = oMoments.m00;
131
132                 if(dArea > 10000){
133                         int posX = dM10 /dArea;
134                         int posY = dM01 /dArea;
135
136                         if(iLastX >= 0 && iLastY >= 0 && posX >= 0 &&
                                posY >= 0){
137                                 line(imgLines, Point(posX, posY), Point(
                                        iLastX, iLastY), Scalar(0, 0, 255),
                                        2);
138                         }
139
140                         iLastX = posX;
141                         iLastY = posY;
142                 }
```

```
143                     cv_ptr->image = cv_ptr->image + imgLines;
144
145
146                     imshow("Thresholded Image", imgThresholded);
147                     imshow(OPENCV_WINDOW, cv_ptr->image);
148
149
150                     if(waitKey(30) == 27){
151                             ROS_INFO_STREAM("ESC pressed by user");
152                             exit(1);
153                     }
154
155                     image_pub_.publish(cv_ptr->toImageMsg());
156
157                     //setup++;      //uncomment for autoclean
158             }
159 };
160
161 /**
162  * @brief       main function
163  * @details Initializes ROS
164  * @param       [in] argc
165  * @param       [in] argv
166  * @return      escape sequence
167  */
168 int main (int argc, char** argv){
169         ros::init(argc, argv, "image_converter");
170         ImageConverter ic;
171         ros::spin();
172         return 0;
173 }
```

Listing 2.13: Steuersignale durch Gestensteuerung

```
1  /**
2   ********************************************************************************
3   * @file                twist_msg_generator_by_tracking.cpp
4   * @author              Christian Waldner
5   * @date                19/07/2017
6   * @copyright   2011-2017 UniBwM - ETTI - Institut 4
7   * @brief               Node which generates cmd_vel-messages by tracking
8   * @details     This node generates cmd_vel-messages by tracking a color.
        To
9   *                              adjust the selection open the contorl
        menu by pressing Ctrl+P
10  ********************************************************************************
11  * @todo                rename: topicname, init-name,
12  * @todo                implement ROS publisher
13  ********************************************************************************
14  */
15 #include <ros/ros.h>
16 #include <cv_bridge/cv_bridge.h>
17 #include <image_transport/image_transport.h>
18 #include <sensor_msgs/image_encodings.h>
19 #include <geometry_msgs/Twist.h>
20
21 #include <opencv2/highgui/highgui.hpp>
22 #include <opencv2/imgproc/imgproc.hpp>
23 #include "stdlib.h"
24
25 /**define for adjusting the erode /dilate filters*/
26 #define FILTER_SIZE_W 5
27 /**define for adjusting the erode /dilate filters*/
28 #define FILTER_SIZE_H 5
29
30
31 using namespace cv;
32 using namespace std;
33
34 static const char* OPENCV_WINDOW = "Original Image";
35 static const char*       PROCESSED_WINDOW = "Processed Image";
36
37 int iLowH =  0;
38 int iHighH = 11;
39
40 int iLowS =  78;
41 int iHighS = 255;
42
43 int iLowV =  110;
44 int iHighV = 255;
45
46 /** Callbackfunction for the controlpanel button red */
47 void redCb(int state, void *ptr){
48         cvSetTrackbarPos("LowH",  NULL, 000);
49         cvSetTrackbarPos("HighH", NULL, 16);
50         cvSetTrackbarPos("LowS",  NULL, 88);
```

```
51          cvSetTrackbarPos("HighS", NULL, 255);
52          cvSetTrackbarPos("LowV",  NULL, 000);
53          cvSetTrackbarPos("HighV", NULL, 173);
54  }
55
56  /** Callbackfunction for the controlpanel button green */
57  void greenCb(int state, void *ptr){
58          cvSetTrackbarPos("LowH",  NULL, 47);
59          cvSetTrackbarPos("HighH", NULL, 74);
60          cvSetTrackbarPos("LowS",  NULL, 75);
61          cvSetTrackbarPos("HighS", NULL, 255);
62          cvSetTrackbarPos("LowV",  NULL, 71);
63          cvSetTrackbarPos("HighV", NULL, 104);
64  }
65
66  /** Callbackfunction for the controlpanel button blue */
67  void blueCb(int state, void *ptr){
68          cvSetTrackbarPos("LowH",  NULL, 89);
69          cvSetTrackbarPos("HighH", NULL, 125);
70          cvSetTrackbarPos("LowS",  NULL, 159);
71          cvSetTrackbarPos("HighS", NULL, 255);
72          cvSetTrackbarPos("LowV",  NULL, 110);
73          cvSetTrackbarPos("HighV", NULL, 255);
74  }
75
76  /** Callbackfunction for the controlpanel button yellow */
77  void yellowCb(int state, void *ptr){
78          cvSetTrackbarPos("LowH",  NULL, 26);
79          cvSetTrackbarPos("HighH", NULL, 38);
80          cvSetTrackbarPos("LowS",  NULL, 78);
81          cvSetTrackbarPos("HighS", NULL, 255);
82          cvSetTrackbarPos("LowV",  NULL, 110);
83          cvSetTrackbarPos("HighV", NULL, 255);
84  }
85
86  int iLastX = -1;
87  int iLastY = -1;
88
89  Mat imgLines;/**<Global matrix used in ImageConverter*/
90
91  /**
92    *     @class          ImageConverter
93    * @brief              ImageConverter
94    * @details    This class converts ROS-images into OpenCV-image format
95    */
96  class ImageConverter{
97          ros::NodeHandle nh_;
98          image_transport::ImageTransport it_;
99          image_transport::Subscriber image_sub_;
100         image_transport::Publisher image_pub_;
101
102         ros::Publisher pub_;
103
104 public:
105
106         ImageConverter():it_(nh_){
```

```
107                 image_sub_ = it_.subscribe("usb_cam/image_raw", 1, &
                        ImageConverter::imageCb, this);
108
109                 image_pub_ = it_.advertise("Image_converter/output_video"
                        , 1);
110
111                 pub_ = nh_.advertise<geometry_msgs::Twist>("turtle1/
                        cmd_vel", 1000);
112
113                 namedWindow(OPENCV_WINDOW);
114                 namedWindow(PROCESSED_WINDOW);
115
116
117                 cvMoveWindow(OPENCV_WINDOW, 100, 50);
118                 cvMoveWindow(PROCESSED_WINDOW, 750, 50);
119
120
121                 //Hue
122                 cvCreateTrackbar("LowH", NULL, &iLowH, 179);
123                 cvCreateTrackbar("HighH", NULL, &iHighH, 179);
124
125                 //Saturation
126                 cvCreateTrackbar("LowS", NULL, &iLowS, 255);
127                 cvCreateTrackbar("HighS", NULL, &iHighS, 255);
128
129                 //Value
130                 cvCreateTrackbar("LowV", NULL, &iLowV, 255);
131                 cvCreateTrackbar("HighV", NULL, &iHighV, 255);
132
133
134                 cvCreateButton("Red", redCb, NULL, CV_PUSH_BUTTON, 0);
135                 cvCreateButton("Green", greenCb, NULL, CV_PUSH_BUTTON, 0)
                        ;
136                 cvCreateButton("Blue", blueCb, NULL, CV_PUSH_BUTTON, 0);
137                 cvCreateButton("Yellow", yellowCb, NULL, CV_PUSH_BUTTON,
                        0);
138         }
139
140         ~ImageConverter(){
141                 destroyWindow(OPENCV_WINDOW);
142         }
143
144         void imageCb(const sensor_msgs::ImageConstPtr& msg){
145                 cv_bridge::CvImagePtr cv_ptr;
146
147                 try{
148                         cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::
                                image_encodings::BGR8);
149                 }catch(cv_bridge::Exception& e){
150                         ROS_ERROR("cv_bridge␣exception:␣%s", e.what());
151                         return;
152                 }
153
154                 imgLines = Mat::zeros(cv_ptr->image.rows, cv_ptr->image.
                        cols, CV_8UC3);
155
```

```
156                 Mat imgHSV;
157
158                 cvtColor(cv_ptr->image, imgHSV, COLOR_BGR2HSV);
159
160                 Mat imgThresholded;
161
162                 inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(
                        iHighH, iHighS, iHighV), imgThresholded);
163
164                 erode(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(
                        FILTER_SIZE_W, FILTER_SIZE_H)));
165                 dilate(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(
                        FILTER_SIZE_W, FILTER_SIZE_H)));
166
167                 dilate(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(2*
                        FILTER_SIZE_W, 2*FILTER_SIZE_H)));
168                 erode(imgThresholded, imgThresholded,
                        getStructuringElement(MORPH_ELLIPSE, Size(2*
                        FILTER_SIZE_W, 2*FILTER_SIZE_H)));
169
170                 //Position detection
171
172                 Moments oMoments = moments(imgThresholded);
173
174                 double dM01 = oMoments.m01;
175                 double dM10 = oMoments.m10;
176                 double dArea = oMoments.m00;
177
178                 double alpha;
179                 double ypsilon;
180
181                 if(dArea > 10000){
182                         int posX = dM10 /dArea;
183                         int posY = dM01 /dArea;
184
185                         if(iLastX >= 0 && iLastY >= 0 && posX >= 0 &&
                                posY >= 0){
186                                 circle(imgLines, Point(posX, posY), 50,
                                        Scalar(0,0,255), 2);
187                                 ROS_INFO("X-Pos: %d, Y-Pos %d", posX,
                                        posY);
188
189                                 //Calculating horizontal angle
190                                 alpha = ((double)posX * 65 / (double)
                                        cv_ptr->image.cols) - 32.5;
191                                 ypsilon = posY;
192                                 //ROS_INFO("%f", alpha); Output angle
193                         }
194                         iLastX = posX;
195                         iLastY = posY;
196                 }
197
198
```

```
199                     cv_ptr->image = cv_ptr->image + imgLines;
200
201                     imshow(PROCESSED_WINDOW, imgThresholded);
202                     imshow(OPENCV_WINDOW, cv_ptr->image);
203
204
205                     if(waitKey(30) == 27){
206                             ROS_INFO_STREAM("ESC pressed by user");
207                             exit(1);
208                     }
209
210                     //Calculating cmd_vel values
211                     geometry_msgs::Twist tw_msg;
212                     tw_msg.angular.z = alpha/10;
213                     tw_msg.linear.x = -(ypsilon-240)/50;
214
215
216                     //Publish cmd_vel & image
217                     pub_.publish(tw_msg);
218                     image_pub_.publish(cv_ptr->toImageMsg());
219             }
220 };
221
222 /**
223  * @brief       main function
224  * @details Initializes ROS
225  * @param       [in] argc
226  * @param       [in] argv
227  * @return      escape sequence
228  */
229 int main (int argc, char** argv){
230         ros::init(argc, argv, "image_converter");
231         ImageConverter ic;
232         ros::spin();
233         return 0;
234 }
```

3 Steuerung des Robot Operating Systems mit MATLAB

3.1 Aufgabenstellung

MATLAB ist in der Forschung und Wissenschaft eine weit verbreitete Software. Die Entwickler von Mathworks stellen für die Software eine Vielzahl an Paketen und Toolboxen zur Verfügung, so auch die 'Robotics System Toolbox'. Diese ermöglicht die Kommunikation mit dem ROS. Die Möglichkeiten der Anbindung und Verwendungszwecke sollen im Anschluss untersucht und genauer beschrieben werden.

3.2 Umsetzung

Neben der Installation von MATLAB ist noch ein lauffähiges ROS-System notwendig. Hierbei spielt es keine Rolle, ob es sich um eine virtuelle Maschine oder um ein natives System handelt. Sie müssen sich nur im selben Netzwerk befinden.

Verbindungsaufbau

Es gibt zwei Möglichkeiten dem ROS-Netzwerk beizutreten. Einerseits kann in MATLAB ein ROSCORE gestartet werden und andererseits kann eine MATLAB-Node an einen bereits laufenden ROSCORE registriert werden. Dies geschieht mit dem Befehl

```
>> rosinit(<hostname>)
>> rosinit(<URI>)
```

Wird keine IP / Adresse oder Hostname angegeben, versucht MATLAB einen laufenden ROSCORE zu finden und sich zu verbinden. Findet MATLAB keinen ROSCORE wird dieser in MATLAB erstellt. Zum Beenden der Verbindung dient der Befehl

```
>> rosshutdown
```

Organisationsbefehle

MATLAB hält einige Kommandos bereit, die es dem Nutzer ermöglichen einen Überblick über das aktuelle System zu erhalten.

`>> rosnode list`	⇒ liefert eine Übersicht der aktuell aktiven ROS-Nodes
`>> rostopic list`	⇒ zeigt die akutell aktiven ROS-Topics
`>> rosservice list`	⇒ zeigt die aktiven ROS-Services
`>> rosnode info </node>`	⇒ zeigt Details zur </node>
`>> rostopic info </topic>`	⇒ zeigt Details zum </topic>
`>> rosservice info </service>`	⇒ zeigt Details zum </service>
`>> rostopic type </topic>`	⇒ zeigt den genutzten Nachrichtentyp
`>> rosmsg show <messagetype>`	⇒ gibt Auskunft über den <messagetype>

Publisher und Subscriber

Neben den Organisationsbefehlen stellt MATLAB auch die Möglichkeit Publisher und Subscriber zu erstellen zur Verfügung.

```
>> <publishername> = rospublisher('</topic>', '<messagetype>')
>> <messagename> = rosmessage(<publishername>)
>> <messagename>.<parameter> = <VALUE>
>> send(<publishername>,<messagename>
```

Die möglichen Parameter der Nachrichten können mit `'rosmsg show <messagetype>'` oder `showdetails(<varname>)` angezeigt werden

```
>> <subscribername> = rossubscriber('</topic>')
>> <varname> = receive(<subscribername>, <timeout>)
```

Die empfangenen Daten können je nach Datentyp weiterverarbeitet und/oder dargestellt werden. Hier ein Beispiel mit Daten eines Laserscanners

```
>> laser = rossubscriber('/scan')
>> scandata = revceive(laser, 10)
>> figure
>> plot(scandata, 'MaximumRange', 5)
```

MATLAB bietet auch die Möglichkeit Daten zu speichern und wieder zu laden. Dies erfolgt mit folgenden Befehlen

```
>> save('<dateiname.mat>', '<varname>'
>> <varname> = load('<dateiname.mat>')
```

Eine kleine Anmerkung zum Kopieren von Daten in MATLAB

```
>> <var1> = <var2>          ⇒ erzeugt eine Referenz auf <var2>
>> <var1> = copy(<var2>)    ⇒ erzeugt eine 'Deepcopy' von <var2>
```

Services

Ein weiterer Weg der Kommunikation innerhalb des ROS-Systems sind sogenannte Services. Auch diese können in MATLAB erzeugt und verwaltet werden.

```
>> <srvname> = rossvcserver('<srvtopic>','<srvmessagetype>', <callbackFunction>)
>> <cltname> = rossvclient('<srvtopic>')
>> <req> = rosmessage(<cltname>)
>> <resp> = call(<cltname>, <req>, 'Timeout', <timeoutValue>)
```

Bagfiles

Bagfiles enthalten ROS-Messages, die aufgezeichnet wurden. Auch hierfür hält MATLAB einige Funktionen bereit.

```
>> <bagname> = rosbag(</path/to/bagfile.bag>)          ⇒ Öffnet das Bagfile
>> <bagname>.AvailableTopics                           ⇒ Zeigt die vorhandenen Topics
>> <bagname>.MessageList(<from>:<to>, <col>:<col>)     ⇒ Zeigt alle Nachrichten (von, bis)
```

Filtern kann man die Bagfiles mit folgendem Befehl

```
>> <bagsel> = select(<bagname>, '<msg-/topicname>', '<msg/topic>')
```

Die genaue Syntax des `select()` - Befehls entnimmt man aus der MATLAB-Dokumentation. Als weiteres Feature können Daten aus den Bagfiles als `timeseries` - Element extrahiert werden.

```
>> <ts> = timeseries(<bagsel>, '<parameter>', '<parameter2>')
```

Auslesen lassen sich die Daten mit:

```
>> <ts>.Data
```

Die `timeseries` - Daten bieten eine Vielzahl an Berechnungsmöglichkeiten. Hier lohnt sich ebenfalls ein Blick in die MATLAB-Dokumentation. Beispielsweise lässt sich der Mittelwert mit `mean(<ts>)` berechnen.
Eine graphische Darstellung der extrahierten Werte kann folgendermaßen erzeugt werden.

```
>> figure
>> plot(<ts>, '<plotOptions>', <plotOptionParameter>)
```

Simulink

Anstelle des normalen 'Command Window' in MATLAB besteht ebenfalls die Möglichkeit SIMULINK zu nutzen, um Topics zu subscriben oder zu publishen. Ebenso können empfangende Messages verarbeitet, aufbereitet und angepasst werden. Auf diese Art und Weise können Algorithmen implementiert und überprüft werden sowie besteht die Möglichkeit einen 'ROS-enabled' Roboter zu steuern. An dieser Stelle möchte ich auf die im Internet und die bei MATLAB mitgelieferten Tutorials hinweisen.
Neben der Simulation und Kommunikation mit dem ROS-System besteht die Möglichkeit direkt aus SIMULINK eine 'standalone' ROS-Node zu generieren. Hierbei generiert SIMULINK den C++ Code aus den Schalt- und Verarbeitungselementen, wie sie auch in der Simulation genutzt werden. Dieser Code wird dann via SSH an das ROS-System übertragen und dort kompiliert und nach Wunsch direkt ausgeführt.
Auch hier möchte ich nochmals auf die ausführlichen Tutorials hinweisen.
Zum Tutorial der standalone ROS-Node Generierung habe ich jedoch zwei Anmerkungen.
Im Abschnitt 3, Punkt 5 heißt es, dass zum kompilieren folgendes Ausgewählt werden soll `'Code > C/C++ Code > Build Model'`. Bei meinem Versuch gab es diesen Eintrag **nicht**. Stattdessen wählte ich `'Code > C/C++ Code > Deploy to Hardware'`.
Ein weiterer Punkt ist, wenn die erstellten ROS-Nodes via MATLAB gestartet und gestoppt werden sollen, muss das Passwort in den Netzwerkeinstellungen gespeichert werden, da sonst keine Verbindung hergestellt werden kann.

3.3 Zusammenfassung

MATLAB stellt ein mächtiges Werkzeug dar, dessen Potential in dieser kurzen Testphase bei weitem nicht ausgeschöpft werden konnte. Allein durch die automatische Codeerzeugung ergibt sich die Möglichkeit Algorithmen relativ einfach zu entwerfen, zu testen und zu optimieren. Auf diese Art und Weise werden unmittelbare Eingriffe in die Software unnötig. Problematisch wird es dann, wenn mehrere in der Entwicklung befindlichen Nodes parallel gestartet werden sollen, da MATLAB -nach aktuellem Kenntnisstand- nur eine ROS-Node beim ROSCORE anmelden kann.
Nach meinen Recherchen ist es nicht möglich bestehende ROS-Packages in MATLAB einzubinden und dort direkt zu nutzen. Es besteht jedoch die Möglichkeit, die gepublishten Daten dieser Pakete zu subscriben und anschließend in MATLAB weiter zu verarbeiten.

3.4 Ausblick

Wie in der Zusammenfassung beschrieben war es nicht möglich MATLAB voll umfänglich zu testen. Daher liegt das Hauptaugenmerk darauf, Algorithmen zu entwickeln, die ggf. die Aufgaben derzeit aktiver ROS-Nodes übernehmen und diese zu optimieren. Beispielsweise kann getestet werden, ob der Wegplanungsalgorithmus von MATLAB besser oder schlechter, im Vergleich zu dem von ROS, arbeitet.

4 Test und Evaluation des ST-SensorFusion-Algorithmus

4.1 Aufgabenstellung

Die Navigation und Positionsbestimmung stellt die Königsdisziplin in der Robotik dar. Nur wenn die Position im Raum exakt bekannt ist, kann ein Wegplanungsalgorithmus verlässlich eine Route berechnen oder ein Mappingalgorithmus eine korrekte Karte anlegen.
Neben GPS stehen auch andere Sensoren zur Verfügung. So können ein linearer Beschleunigungssensor und eine Kreiselplattform algorithmisch so zusammengefasst werden, dass man mittels einer Trägheitsnavigationseinheit (Intertialmeasurement unit - IMU) seine Position im Raum bestimmen kann.
Dieser Algorithmus basiert auf der Berechnung von Quaternionen und wird Sensor-Fusion genannt. Das Prinzip von SensorFusion ist folgendes. Bestimmte Probleme können nicht mit einem einzelnen Sensor gelöst werden. Kombiniert man jedoch zwei vom Problem unabhängige Sensoren miteinander, kann die Fusion beider Sensordaten eine Lösung des Problems, hier Positionsbestimmung, sein.
Die Firma ST bietet ein MEMS (Microelectromechanical systems) Board an, womit die oben beschriebene SensorFusion durchgeführt werden kann. Bei dieser Aufgabenstellung ging es vorrangig darüber herauszufinden, inwieweit die auf dem 'X-NUCLEO-IKS01A1 Board' verwendeten Sensoren für die Anwendung, auf den vom Institut genutzten Robotern, geeignet sind.

4.2 Umsetzung

ST bietet für das MEMS-Board ein Softwarepaket, womit die Funktionen und Fähigkeiten der Platine getestet werden können. In diesem Fall war das Problem, dass diese Software nicht für die μC-Boards, die im Institut vorhanden sind, kompatibel war.
Dementsprechend war eine Portierung der bereitgestellten Software, zu einem der vorhandenen Boards, notwendig. Dies bedeutet, dass die Initialisierungen überprüft und auf den genutzten μC angepasst werden musste, sowie die Kontrolle der Pinbelegungen der I^2C-Schnittstelle, damit die Kommunikation mit den Sensoren gewährleistet ist.

Nachdem die grundlegenden Anpassungen durchgeführt wurden, konnte die Unicleo-GUI gestartet und die Evaluation begonnen werden. Hierzu baute ich eine Testplattform auf, die es ermöglicht die Platine in allen drei Achsen zu bewegen, um

reproduzierbare Ergebnisse zu erhalten. Auffällig hierbei war, dass eine Drehung um 90°durch den Sensor nicht korrekt wahrgenommen wurde und dieser nur ca. 84°gemessen hatte.
Das von ST bereitgestellte Programm ist für mehrere Sensoren und μC ausgelegt, sodass dieses sehr verschachtelt geschrieben wurde, um eine höchstmögliche Kompatibilität zu erreichen. Auf Grund dieser Tatsache gestaltete es sich gerade am Anfang recht schwierig den Funktionsaufrufen zu folgen und die Funktion des Programms vollständig zu verstehen.
Nach einiger Recherche fand ich einen Sensitivity-Faktor, der mit der Anzahl der gezählten Bits multipliziert wird. Durch experimentelles Testen war es möglich den Wert so anzupassen, dass eine 360°Drehung auch als solche erkannt wird. Selbiges gilt für die Drehung um andere Winkel.
Wie oben schon beschrieben war neben den falschen Winkelwerten auch der BIAS-Wert, der durchgehend angezeigt wird. Leider sind die wirklich interessanten Funktionen der SensorFusion in einer Library gepackt, weswegen kein direkter Zugriff möglich ist. So ist auch nicht ersichtlich inwieweit die BIAS-Werte kompensiert werden, ebenso wie die tatsächliche Implementierung der SensorFusion aussieht.

4.3 Zusammenfassung

Auf Grund der Komplexität des Programms und der Problematik, dass die eigentliche Implementierung in einer Library versteckt sind, kann nur recht wenig über die Anwendbarkeit der SensorFusion innerhalb des Institutes gesagt werden. Fakt ist, dass nach Anpassung einiger Werte die Messergebnisse um ein vielfaches genauer wurden und somit zu vermuten ist, dass mithilfe einer eigenen Implementierung zufriedenstellende Ergebnisse zu erwarten sind.

4.4 Ausblick

Im nächsten Schnitt ist geplant den SensorFusion-Algorithmus selbst zu implementieren. Hierzu muss das Programm von Grund auf neu programmiert werden. Dies beginnt bei der Initialisierung des μC, den Start und die Initialisierung der I^2C-Schnittstelle, sowie die Kommunikation mit den Sensoren zu implementieren. Im Anschluss können dann der Kalman-Filter sowie der SensorFusion-Algorithmus geschrieben und getestet werden.