Project COLOS - Technical Report 2.1

Camera Module for Onboard Relative Localization in SWARM of robots

User Manual

Jan Faigl, Tomáš Krajník, Jan Chudoba, Martin Saska, Libor Přeučil

Version 1.0

The Gerstner Laboratory for Intelligent Decision Making and Control
Department of Cybernetics (K13133)
Faculty of Electrical Engineering, Czech Technical University
166 27 Prague 6, Technická 2, Czech Republic
xfaigl@labe.felk.cvut.cz

Contents

1 Hardware Description **5**

1.1 Overo minicom board 5

1.2 Voltage regulator board 5

1.3 Boards Connections 5

1.4 LED Indicators 6

1.5 Disabling Undervoltage Switch 7

2 Blob Detection **7**

2.1 Principle of Detection 7

2.2 Performance Indicators 9

2.2.1 Real Computational Requirements 9

2.2.2 Precision of the Localization 10

2.2.3 Detectability of the Pattern 14

3 System Configuration **17**

3.1 WiFi Configuration 18

3.2 Image Capturing 20

3.3 Initialization of the Tracker Server 25

4 Camera Settings **28**

4.1 Practical Tips for Using the Overo Caspa Camera 30

4.1.1 Camera Control Settings 30

4.1.2 Auto exposure 30

5 Applications Settings **31**

5.1 Tracker Server 31

5.2 Tracker Client 34

5.3 `tcapture` - Capturing mode 38

5.4 Applications for Remote Access to the Camera 39

5.5 Creating Camera Control Settings File 43

5.6 Taking Images for the Camera Calibration 43

6 Future Work **44**

A Specifications **45**

B Camera Module Hardware **46**

C Applications **48**

List of Figures

1 Hardware boards of the camera module. 6
2 LED indicators of the hardware boards. 7
3 Additional pins for the power source of the regulator without undervoltage switch. 7
4 B/W pattern for the circle detector. 8
5 Measured distances for the image resolutions tested. 12
6 Examples of captured images. 14
7 Detectability and localization precision in $x-$axis for the pattern with outer diameter 3.5 cm and two sizes of the squared bounding box. 15
8 Detectability and localization precision in $x-$axis for the pattern with outer diameter 3.5 cm and two sizes of the squared bounding box. 16
9 Examples of `tcam` usage. 42
10 Connectors of the Overo minicom board. 46
11 Schematic of the voltage regulator and battery undervoltage switch. 46
12 Circuits of the voltage regulator and battery undervoltage switch. 47

List of Tables

1 Maximal FPS achieved 10
2 Minimal FPS achieved 10
3 Measured distances for the image resolutions tested 12
4 Measured distances in the y-axes 13
5 Measured distances in the z-axes 13
6 Hardware parameters 45
7 Access parameters 45
8 Tracker parameters 45

Listings

1 An example of ~/.ssh/config file. 18
2 An example of the /etc/ntp.conf file. 18
3 An example of the /etc/hosts file with a local database of the host names. 18
4 An example of the local startup configuration file /etc/local.start. 19
5 The /opt/wifid script for periodical checking of the WiFi connection. 19
6 The /opt/check_wifi.sh script for testing availability of the WiFi connection. 19
7 The /opt/wlan_colos script for configuring WiFi connection to the *colos* network. 20
8 Camera driver initialization /etc/init.d/camera_init. 20
9 Initialization of user input handling in the /etc/init.d/camera_event script. 21
10 Initialization of capturing service in /etc/init.d/camera_capture script. 21
11 Activation of the capturing service scripts. 22
12 Handling of user inputs coming GPIO pints. 23
13 Starting of the capturing. 24
14 Tracker service initialization. 25
15 Tracker program execution. 26
16 Directory layout of the tracker program. 27
17 Printed version of the tracker program. 27
18 Gumstix Caspa camera control settings accessible by the V4L2 interface provided by the mt9v032 driver. 28
19 Gumstix Caspa camera control settings accessible by the V4L2 interface provided by the mt9v032 driver. 30
20 An example of the tracker server configuration file (tracker-480x360.cfg) 31
21 An example of the camera parameters found by the Matlab toolbox [1]. 33
22 A structure send by the tracker server (tracked_object.h) 34
23 A simple client of the tracker server. 35
24 A configuration for the capturing program. 38
25 A configuration file for the tcapture program in the server mode. 39
26 Selected program options of the tcam application. 40
27 All program options of the tcam application. 48

1 Hardware Description

The camera module consists of four main electronic boards. The core of the module is the Gumstix Overo board with the OMAP 3503 Application processor running at 600 Mhz and accompanied with 128 MB RAM and 802.11b/g wireless communication. The second board is the Caspa camera board with the Aptina MT9V032 CMOS sensor. This board is also an off-the-self product provided by Gumstix [2].

Two additional boards are custom boards providing power and interfaces to Gumstix Overo. The boards are called the Overo minicom and voltage regulator boards. The boards are shown in Fig. 1. A summary of the hardware parameters is shown in Table 6.

1.1 Overo minicom board

The Gumstix Overo processor board is attached to the developed Overo minicom board, which is an interface board for Overo processor boards. Onboard circuits provide 3.3 V power supply for Overo, and 1.8 V for a logic-level conversion between 1.8 V (Overo levels) and 5 V (external levels). The Overo minicom board itself is powered by 5 V supply voltage thru J1 connector. The J19 connector make accessible the first serial port of Overo COM (i.e., the `/dev/ttyS0` device under Linux based operating systems) in 5 V TTL levels. The J15 connector provides the Overo console serial port also in 5 V levels. Locations of the connectors are depicted in Fig. 10.

1.2 Voltage regulator board

The voltage regulator board can be divided into two parts, which are also physically collected at the board. The first part is a battery undervoltage switch, which avoids battery damage due to excessive discharge. The second part is a voltage regulator providing 5 V. A scheme of the board is depicted in Fig. 11 and circuits placement in Fig. 12.

The undervoltage switch disconnects battery in a case of battery voltage below 8.7 V, which is safe value for the 3-cell Li-Pol battery with the nominal voltage 11.4 V. At the voltage 9.3 V a warning LED (D3) is lighted up. The voltage threshold may be adjusted by a resistor divider consisting of resistors R6, R7, and R8. The divider voltage

$$U_{\mathrm{div}} = \frac{(\mathrm{R7} + \mathrm{R8})}{\mathrm{R7} + \mathrm{R8} + \mathrm{R9}}$$

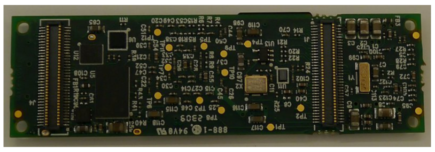
is compared to the reference voltage 1.2 V. The **undervoltage protection** can be **disabled** by removing resistor R8.

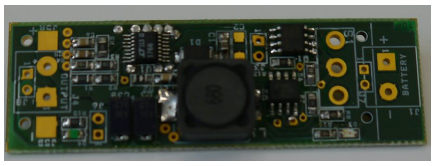
The voltage regulator is based on the LT1766 adjustable switching regulator. The output voltage is set to 5 V by a voltage divider consisting of resistors R1 - R4. The maximal load current is 1.5 A. Regarding the LT1766 datasheet the expected efficiency is about 90%.

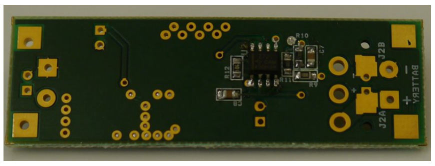
1.3 Boards Connections

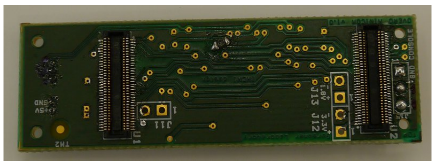
The boards are connected together as follows. The Gumstix Caspa camera board is connected to the Gumstix Overo board using a supplied flex cable. The Gumstix Overo board itself is attached to the Overo minicom board by two 60-pin Hirose connectors. The 5 V voltage provided by the voltage regulator board is connected by two wires. Finally the battery is connected to the voltage regulator board using a dean-T connector. All boards connected together are shown in Fig. 1i.

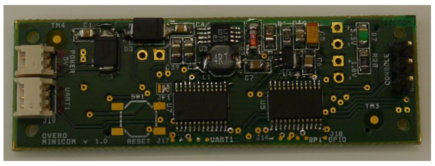
(a) Gumstix Overo - top

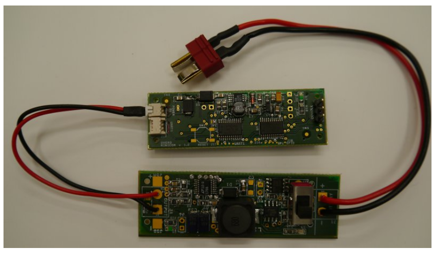

(b) Gumstix Overo - bottom

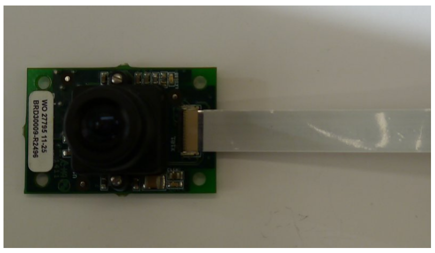

(c) voltage regulator board - top

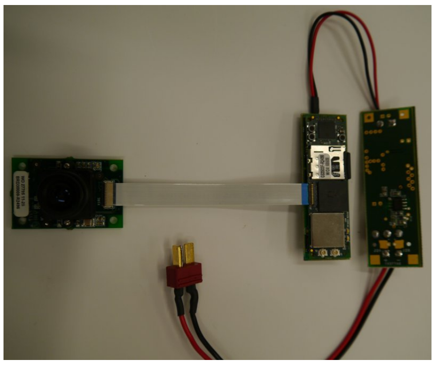

(d) voltage regulator board - bottom


(e) minicom board - top


(f) minicom board - bottom


(g) minicom connected to the power board


(h) Gumstix Caspa camera


(i) overview of the connected boards

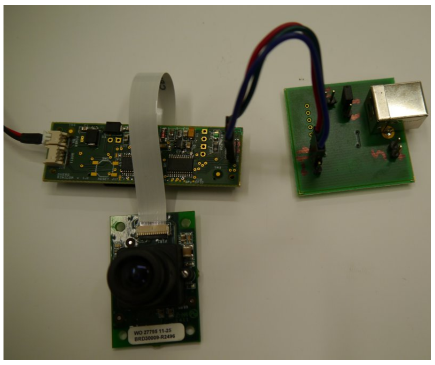
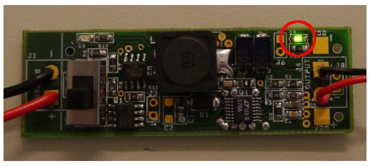

(j) attached console USB converter

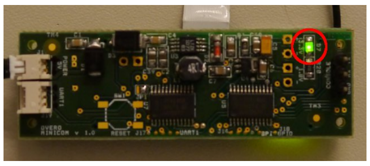
Figure 1: Hardware boards of the camera module.

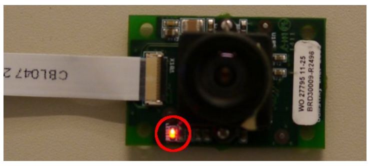
A USB to RS232 converter can be attached to the Overo minicom board by 3-pins. The pins are labeled on the board. However, if the module is used with the supplied converter, it is necessary to cross the TX and RX pins, see Fig. 1j.

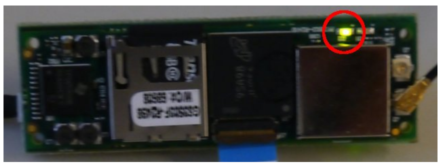
1.4 LED Indicators

The boards are equipped with LEDs indicated its operational modes. The LEDs are shown in Fig. 2.


(a) voltage regulator boards (switched on)


(b) Minicom power indicator (switched on)


(c) Camera usage indicator (capturing / tracking is running


(d) Overo power indicator (switched on)

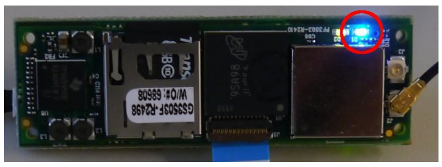

(e) Overo WiFi indicator (connection to a network)

Figure 2: LED indicators of the hardware boards.

1.5 Disabling Undervoltage Switch

If it is necessary to disable the undervoltage switch protecting the battery, it can be realized in several ways. First, it can be done by changing values of the resistor divider, R6, R7, and R8, see Fig. 11 and the resistors locations in Fig. 12.

The second way is a bypassing the mosfet transistor IRF7416 (a part Q1 in Fig. 11 and Fig. 12), which is located next to the main switch S1, see Fig. 3.

Finally, the power voltage regulator can be powered without the undervoltage switch if pins of J3 are used for powering instead of J1. A location of the pins is shown in Fig. 3.

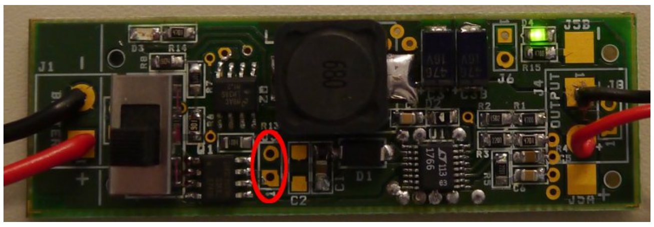


Figure 3: Additional pins for the power source of the regulator without undervoltage switch.

2 Blob Detection

2.1 Principle of Detection

A principle of the blob detection is based on an image segmentation and finding two discs forming a black and white ring. The idea is based on a comparison of the blob size (in the number of pixels) with the expected size of the pattern used, i.e, disk and ring. An example of the pattern is shown in Fig. 4.

In particular the algorithm works as follows. First, an image is thresholded to separate black and white pixels. Then, the image is searched for segments. A segment is labeled as a candidate segment if it has a minimal expected size (in number of pixels), its bounding box fits to the expected dimensions, and ratio of its size with the expected size corresponds to the selected values. As two segment form the B/W ring pattern,

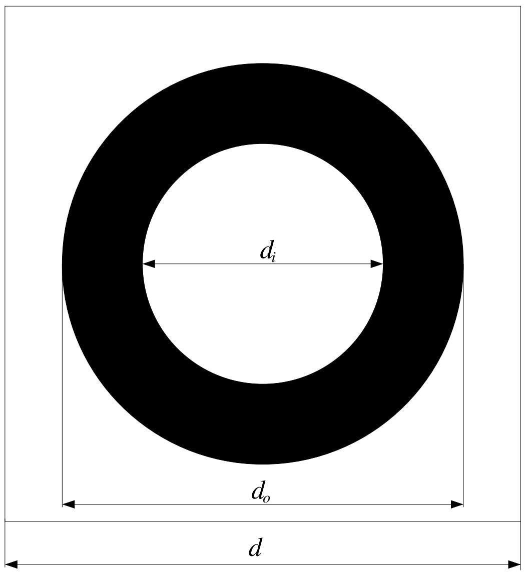


Figure 4: B/W pattern for the circle detector.

two segments with appropriate outer and inner parameters have to be found consecutively. Regarding the pattern, these two segments should have the same center point (considering a small tolerance). Besides, a ratio of the segments' sizes have to meet selected tolerance (according to the width of the ring) in order to denote the segments as a found blob.

The above finding procedure can be repeated until the whole image is processed, and several blobs can be found. In a case of a single blob assumed in the scene, the searching process can be terminated once the first blob is detected. Moreover, the searching procedure can be speeded up using previous position (center) of the blob, which represents some kind of *tracking* mechanism. However, if the pattern's position is changed significantly between two consecutive images, the whole image is processed.

Although B/W pattern, which should be robust to changes of the light conditions, is considered, the appropriate threshold separating the dark and light shades has to be selected according to the current light conditions. The threshold is determined as follows. If the blob is found, the threshold is set as the average value of the mean intensity values of each disk forming the ring. The intensity value is a single value in a case of grayscale image or a sum of particular intensity values of each color channel, e.g. in a case of the RGB image. In a case the current threshold does not lead to detect a blob, its value is increased (if its value is less than the last value for which a blob has been found) or decreased by a selected bias value. The threshold value is bounded by zero and the maximal value (e.g., 768 from 3×256 for RGB images). If the threshold value reaches one of the bounds, the bias is set to zero, the threshold value is restricted to the particular bound, and the last threshold is set into half of the threshold value interval. Notice, the threshold bias is increased by a selected step after each increase of the threshold value, e.g. 60, to set the threshold value progressively.

The threshold changing procedure can be applied multiply times for a single image until a blob is not found. If the changes of the light conditions are not fast and significant, it is usually sufficient to process several consecutive images to determine the threshold.

In addition, it is also necessary to have a blank space around the outer disk of the pattern to separate the ring from the background, e.g., see Fig. 4 where a square bounding box forms a 2 cm width strip around the disk.

The pattern is defined by three parameters: the diameter of the outer disc d_o, the diameter of the inner disc d_i, and the squared bounding box with the side d. The discs are located at the center of the box.

2.2 Performance Indicators

The above described blob detection has been implemented in C++ and its performance has been experimentally verified in a series of tests for analyzing its computational requirements and achievable precision of relative localization.

Regarding the computational requirements, the implemented algorithm has been compiled by the G++ ver. 4.3.3 cross-compiler for the `arm-angstrom-linux` distribution used on the Gumstix Overo board. As the blob detection considers only B/W pattern, the computational burden can be decreased using a grayscale image, which also reduced computational time of the image conversion from the YUV format used for reading the image from the Caspa device.

The precision of the localization depends on the estimated parameters of the camera [3]. In particular, the camera has been calibrated using the Matlab toolbox [1].

Results presented in Section 2.2.1 and Section 2.2.2 have been measured using a pattern with d_o=14 cm, d_i=8.4 cm, and d =18 cm.

2.2.1 Real Computational Requirements

The computational requirements has been measured as the maximal number of the processed images per second (herein denoted as FPS following standard conventions) in the full processing loop, i.e., including the capture time, YUV to grayscale conversion, blob detection, transformation of the coordinates using camera parameters, and transfer of the coordinates from the camera module to a client computer using UDP protocol over WiFi. Thus, the results presented demonstrate real application and really achievable FPS.

The image processing time depends on several factors. Although the image conversion and coordinates transformation depends only on the image resolution, the blob detection can vary. First, using previously know position of the blob in the image can significantly reduce the required time to find the segments. Besides, the processing time also depends on the number of pixels forming the segments; thus, a smaller pattern or a pattern at a longer distance can be detected faster than a larger pattern or a pattern close to the camera. The worst case scenario is a situation when a blob is not detected, because it requires searching of the whole image. In such a situation, the processing time depends only on the image resolution. Regarding these aspects the achievable FPS has been measured for four selected resolutions 320×240, 480×360, 640×480, and 752×480 and for a pattern placed at various distances from the camera. The detailed results are presented in Table 1 and an overview of the FPS in Table 2.

In Table 1, the maximal achieved FPS using the tracking (i.e., the previous position of the found blob) is presented. For the lowest resolution considered, the processing is limited by the Caspa camera, which can provide images at 60 Hz. On the other hand, a lower resolution limits maximal measured distance. In the case of 320×240 resolution, the blob is sporadically detected also for the distance 3.5 m, but the blob is detected approximately in about 4 % cases. For the 480×360 resolution, the maximal usable distance is 3.5 m, and for the distance 3.8 m the blob is detected in about 15 % cases. For longer distances the blob is not detected reliably.

The guaranteed numbers of processed images per second are depicted in Table 2, where FPS$_{worse}$ are for situations where the blob is not detected at all, and therefore, the whole image must be searched for a segment. The column FPS$_{min}$ denotes minimal image processing frequency when the blob is perfectly found using its previous position and its position is not closer than 0.5 m.

Table 1: Maximal FPS achieved

L	**320×240**	**480×360**	**640×480**	**752×480**
[m]	FPS_{max}	FPS_{max}	FPS_{max}	FPS_{max}
0.5	60.0	35.0	23.0	20.0
1.0	60.0	46.0	30.0	27.0
1.5	60.0	50.0	33.0	30.0
2.0	60.0	51.0	34.0	30.0
2.5	60.0	51.0	35.0	30.0
3.0	60.0	51.0	35.0	31.0
3.2	60.0	52.0	35.0	31.0
3.5	-	52.0	34.0	31.0
4.0	-	-	34.0	30.0
4.5	-	-	35.0	31.0
5.0	-	-	35.0	30.0
5.5	-	-	35.0	31.0

Table 2: Minimal FPS achieved

Resolution	FPS$_{worse}$	FPS$_{min}$
320x240	33	60
480x360	18	35
640x480	9	23
752x480	7	20

It is worth to mention that the image is captured from the camera in the YUV format, which needs a conversion to the RGB (or grayscale), alternatively the segmentation can be done directly in YUV. In the second revision of the camera module, a new optimized implementation of the conversion is used. The real required computational times of the conversion (for 480×360 resolution) are about 11 ms and 7 ms for the RGB and grayscale images, respectively. The achieved FPS presented above are high enough for the COLOS project needs, therefore an eventual expected benefit of the direct segmentation in YUV is not evident because of a more complex implementation. Thus, the direct segmentation in YUV is not implemented for the second revision of the camera module. Besides, it is expected that the `mt9v032` driver will allow direct usage of the Bayer10 format in the future revisions.

2.2.2 Precision of the Localization

Similarly to the presented real computational requirements the precision of the localization has been measured using the camera module and a client computer receiving information about the detected blobs. The precision is measured as a difference between the position provided by the camera module and the real distance measured by a tape measure. The provided relative position of the detected blob consists of 3D coordinates x, y, z, estimated rotations $roll$, $pitch$, yaw, and two pixel ratios of the blob (ratio of dark and light pixels and ratio of the current and expected number of pixels). Although rotations are estimated, only the position coordinates are considered in the experimental evaluation of the camera module presented here.

The information about the tracked object provided by the UDP server of the camera module also contains a bit flag denoting if the estimated values are valid or not. The validity is determined according to detected blob; thus the valid flags is true if the blob has been detected, otherwise it is false. The current version of the tracker does not use sophisticated methods of the sensor fusion, and therefore, it may happened that another object is recognized as the ring pattern. However, this situation does not happen during the experimental evaluation as the scene does not contain objects, which can eventually be interpreted as the pattern.

The relative position of the detected blobs can vary due to small changes in the captured images, and therefore, average values of the measured values of the tracked object are considered. The average values are computed only from valid information about the tracked object, i.e., several measurements over a period are considered providing more than 20 measurements. In fact, the pattern has been successfully detected for several such periods without fall outs during the experimental evaluation. The pattern has been sporadically detected only in few cases when the maximal usable distance for object tracking has been measured. These cases are described in Section 2.2.1.

First, measured distances in the x-axis are presented in Table 3. In this test, the distance of the camera module from the pattern has been changed while its orientation has been fixed; however, only roughly as the module has been positioned by hands. Average measured distance is subtracted from the real distance L, the average error is denoted as L_e, and the standard deviation as s_e. The standard deviation in percents of the measured value is denoted as $\%s_e$. The results indicate that a higher resolution provides more precise estimations at longer distances, but principally it provides the same performance, which can be easily seen from Fig. 5, where the average errors in percentage points of the real distances are presented. Regarding the standard deviations, the repeatability of the measurements are in units of millimeters, which means tenths percentage points of the measured distance. The results are in a correlation of the expected precision reported in [4]. Also be aware that the camera has been placed and pointed out to the pattern imprecisely by a hand with an expected precision about one centimeter.

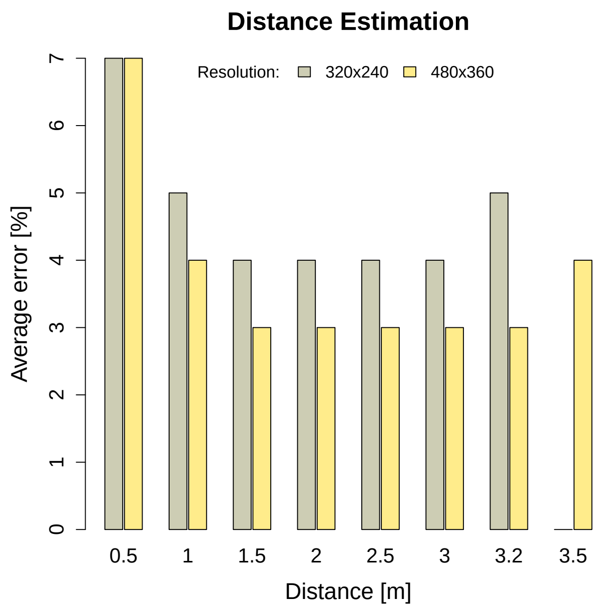
Precision in other axes has been tested only for the resolution 480×360, because it provides a good trade-off between achieved FPS and the maximal measured distance. In this experimental setup, the camera has been fixed at the position 3 m far from a wall and the pattern has been placed into various locations providing measurements in y and z axes. The orientation of the axes depend on the placement of the camera, i.e., on the orientation of the CMOS sensor according to the real world coordinates. The results presented in this report correspond to the placement in which x-axis represents depth, y-axis is aligned with the vertical direction, and z-axis is aligned with the horizontal axis. In this setup the pattern has been placed at the wall by a hand, but the estimated precision of the placement is subjectively a bit better than in the previous setup.

The results for changes in the y-axis are presented in Table 4 and in Table 5 for the changes in z-axis, where L_e is the average value of the error of the measured distances in the particular axis, $\%L_e$ is the same error in the percentage points of the real distance, s_e is the standard deviation, and $\%s_e$ is the standard deviation in the percentage points of the measured distance. All the results presented are computed from more than 20 measurements and in all cases the detection of the blob is valid and the validity is stable over hundreds of images. Notice, the repeatability of the measurements is very high, as the standard deviations are very low.

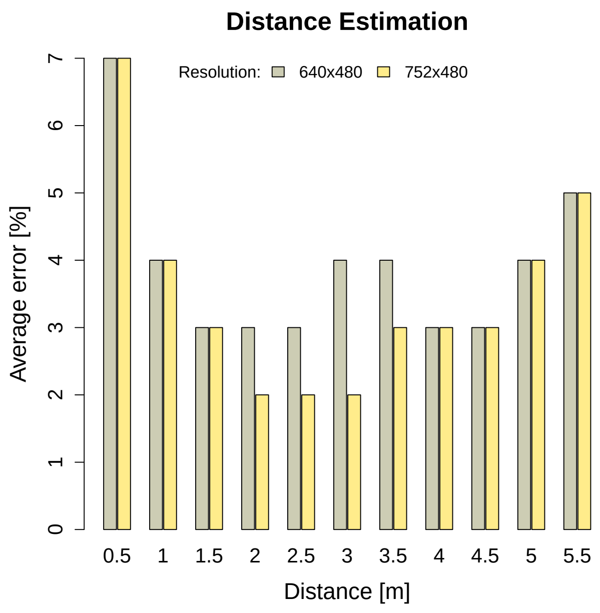
The results presented in Table 5 also indicate how the radial distortion affects the maximal usable field of view. If a blob is placed at the height y=–1 m, it is detectable up to horizontal position z=1.1 m. However,

Table 3: Measured distances for the image resolutions tested

L	320×240			480×360			640×480			752×480		
	L_e	s_e	$\%s_e$	L_e	s_e	$\%s_e$	L_e	s_e	$\%s_e$	L_e	s_e	$\%s_e$
[m]	[cm]	[cm]	[%]	[cm]	[cm]	[%]	[cm]	[cm]	[%]	[cm]	[cm]	[%]
0.5	3.6	0.1	0.2	3.5	0.9	2.0	3.5	0.3	0.6	3.3	0.6	1.3
1.0	4.5	0.1	0.1	4.2	0.1	0.1	3.6	0.3	0.3	3.5	0.6	0.6
1.5	6.1	0.1	0.1	4.4	0.2	0.1	4.4	0.1	0.1	3.8	0.8	0.5
2.0	8.7	1.0	0.5	5.6	0.2	0.1	5.4	0.3	0.1	4.8	0.2	0.1
2.5	9.0	0.6	0.2	8.0	0.4	0.2	8.0	0.4	0.2	6.0	0.4	0.2
3.0	12.0	0.7	0.2	10.0	1.0	0.3	11.0	0.6	0.2	6.0	0.5	0.2
3.2	17.0	0.8	0.3	8.1	2.2	0.7	8.4	0.6	0.2	8.6	0.8	0.3
3.5	-	-	-	12.5	3.2	0.9	12.4	0.9	0.3	10.0	0.6	0.2
4.0	-	-	-	-	-	-	10.0	1.5	0.4	11.1	15.7	4.0
4.5	-	-	-	-	-	-	14.9	1.2	0.3	14.1	1.1	0.2
5.0	-	-	-	-	-	-	22.2	3.0	0.6	22.2	3.0	0.6
5.5	-	-	-	-	-	-	28.7	2.6	0.5	27.5	3.7	0.7



(a)



(b)

Figure 5: Measured distances for the image resolutions tested.

it is not the case of its position at y=–0.5 m that allows horizontal position z=1.0 m, or the height y=0 m providing maximal horizontal position z=0.9 m.

Table 4: Measured distances in the y-axes

Pattern Position			L_e	$\%L_e$	s_e	$\%s_e$
(x, y, z) [m]			[cm]	[%]	[cm]	[%]
3.0	0.0	0.0	0.9	n/a	0.001	0.11
3.0	-0.4	0.0	0.2	0.5	0.070	0.18
3.0	-0.8	0.0	0.1	0.1	0.280	0.35
3.0	-1.2	0.0	2.1	1.7	0.470	0.40
3.0	-1.6	0.0	2.7	1.7	0.700	0.43
3.0	-2.0	0.0	3.7	1.8	1.700	0.87

Table 5: Measured distances in the z-axes

(a) y=0.0 m

Pattern Position			L_e	$\%L_e$	s_e	$\%s_e$
(x, y, z) [m]			[cm]	[%]	[cm]	[%]
3.0	0.0	-0.9	0.8	0.9	0.490	0.55
3.0	0.0	-0.6	0.2	0.3	0.200	0.33
3.0	0.0	-0.3	0.6	2.0	0.070	0.23
3.0	0.0	0.0	0.7	n/a	0.001	0.14
3.0	0.0	0.3	1.1	3.7	0.300	1.04
3.0	0.0	0.6	3.1	5.2	0.090	0.16
3.0	0.0	0.9	2.9	3.2	0.150	0.17

(b) y=-0.5 m

Pattern Position			L_e	$\%L_e$	s_e	$\%s_e$
(x, y, z) [m]			[cm]	[%]	[cm]	[%]
3.0	-0.5	-1.0	7.4	7.4	0.100	0.11
3.0	-0.5	-0.6	2.2	3.7	0.100	0.17
3.0	-0.5	-0.9	2.2	2.4	0.170	0.19
3.0	-0.5	-0.3	2.1	7.0	0.060	0.22
3.0	-0.5	0.0	1.7	n/a	0.004	0.24
3.0	-0.5	0.3	1.2	4.0	0.110	0.35
3.0	-0.5	0.6	0.3	0.5	0.060	0.10
3.0	-0.5	0.9	0.7	0.8	0.190	0.21
3.0	-0.5	1.0	0.3	0.3	0.120	0.12

(c) y=-1.0 m

Pattern Position			L_e	$\%L_e$	s_e	$\%s_e$
(x, y, z) [m]			[cm]	[%]	[cm]	[%]
3.0	-1.0	-1.1	10.8	9.8	0.650	0.66
3.0	-1.0	-0.6	5.5	9.2	0.200	0.37
3.0	-1.0	-0.9	5.9	6.6	0.250	0.30
3.0	-1.0	-0.3	4.8	16.0	0.070	0.28
3.0	-1.0	0.0	4.5	n/a	0.001	0.02
3.0	-1.0	0.3	4.1	13.7	0.800	2.35
3.0	-1.0	0.6	3.6	6.0	0.140	0.22
3.0	-1.0	0.9	3.1	3.4	0.480	0.52
3.0	-1.0	1.1	1.3	1.2	0.020	0.02

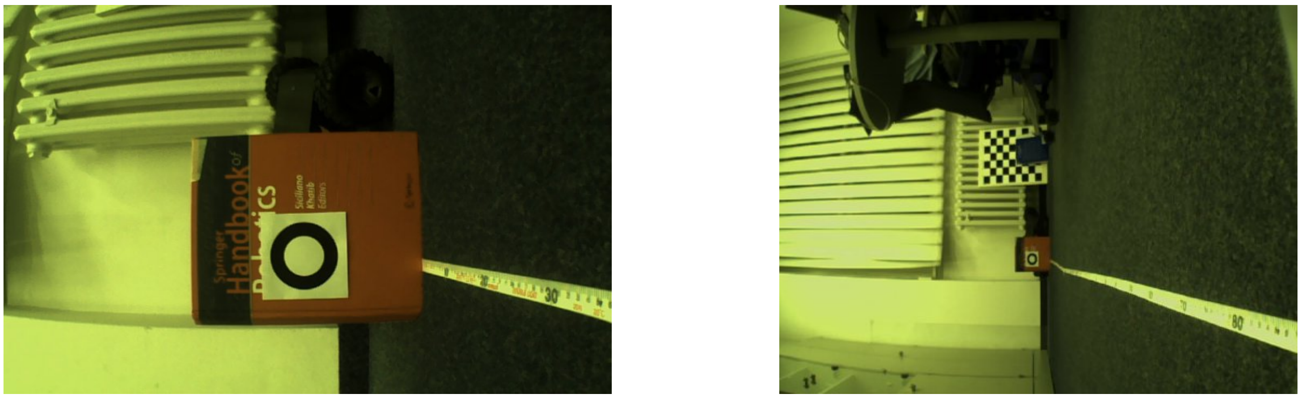


Figure 6: Examples of captured images.

2.2.3 Detectability of the Pattern

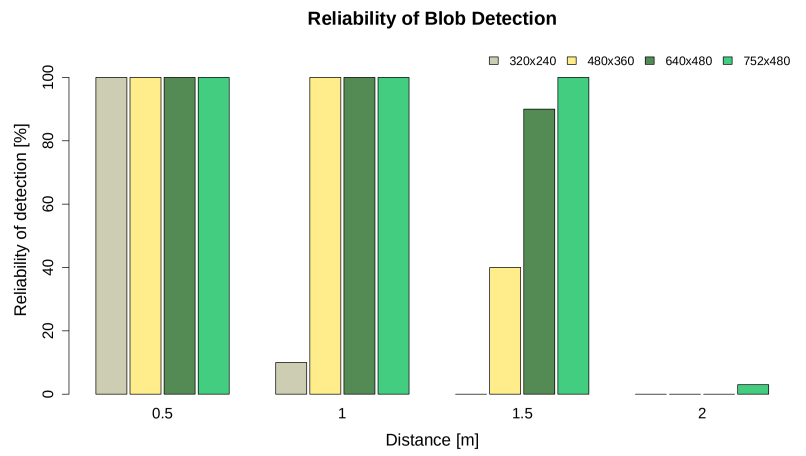
The blob detection is based on separation of two discs forming a B/W ring pattern, see Fig. 4. It is clear that the size of the pattern affects practical deployment of the presented blob detection algorithm. Small patterns can be more practical; however, the size of the pattern affects the maximal measurable distance as too small pattern can be hardly detected from long distances. Regarding the computational requirements small patterns contain less pixels; thus, the processing can be faster. Therefore, a higher resolution can be used for better detection of small patterns, with lower computational requirements than for larger patterns. However, in the worst case (i.e., not detected pattern), the computational requirements are same as presented in Table 2.

The detectability of patterns of various sizes has been studied in a similar experimental setup like in Section 2.2.2, i.e., the position of the pattern has been fixed and the camera module has been pointed to the center of the pattern and positioned at different distances (manually by hand), see examples of captured images by the camera module showed in Fig. 6.

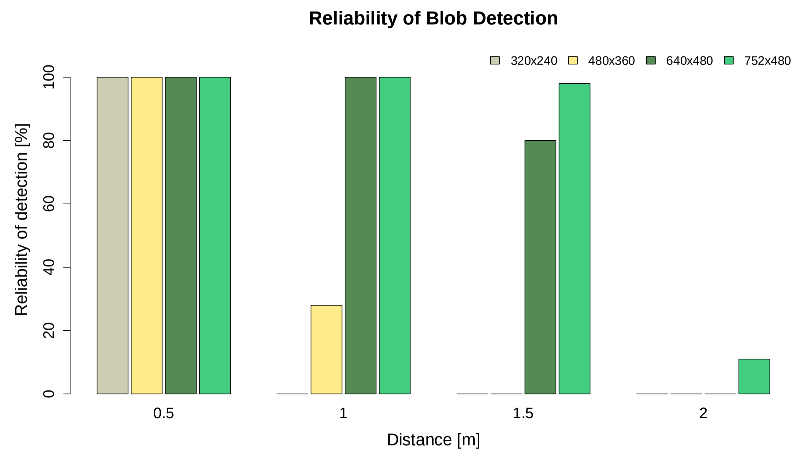
The detectability of the pattern is measured as a number of detected blobs from 100 captured images; thus, the number of successfully detected blobs is an estimation of the detectability of the pattern at the given distance in percentage points. In addition, an estimation of the localization precision in $x-$axis is measured as the average value of the error in percentage points of the real distance as in the previous sections. The camera module reference point has been established for the distance 1 m from the blob. It means, that the module position has been adjusted until the provided distance is 1 m with precision in units of millimeters. Therefore the error at the distance 1 m is typically smaller than for other distance. It is worth to mention that the error can be decreased using an additional corrections addressing systematic error of the distance measurement.

First, a small pattern with the outer diameter d_o=3.5 cm has been considered with two dimensions of the bounding square: d=7.5 cm and d=5.5 cm. Results are presented in Fig 7. The results show that such a small pattern can be used reliability within a range about one meter. Using the highest resolution available 752×480 the blob detection algorithm can be used up to 1.5 m. The results also indicate that a wider blank space around the disc increases detectability a bit (in particular using the 480×360 resolution).

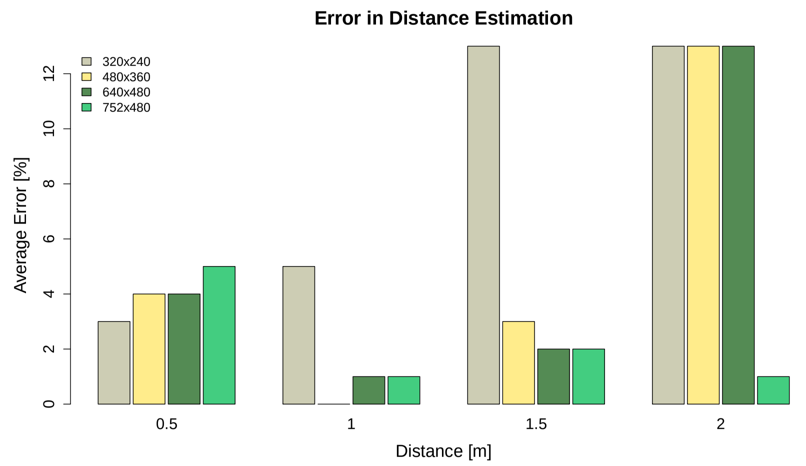
A larger disc can be detected from a longer distances as can be see in Fig. 8. In this experimental evaluation, patterns with the discs' outer diameters 7.0 cm and 6 cm have been considered. For both patterns, the width of the outer blank space is formed from the 1 cm width strip.



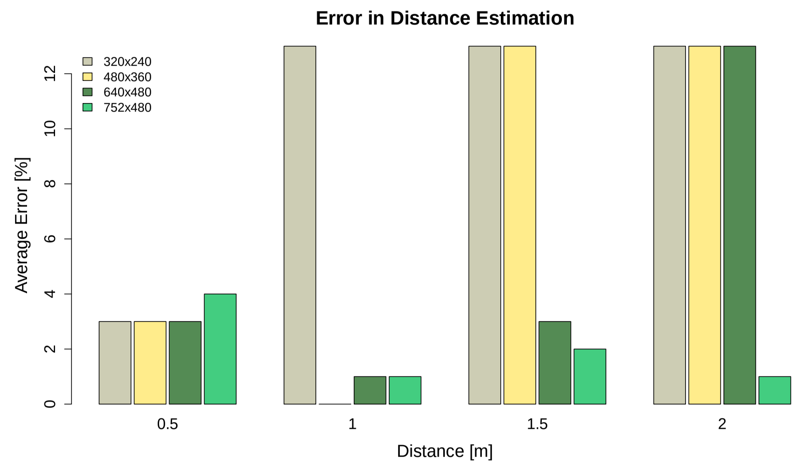
(a) d_o=3.5 cm, d_i=2.1 cm, d=7.5 cm



(b) d_o=3.5 cm, d_i=2.1 cm, d=5.5 cm

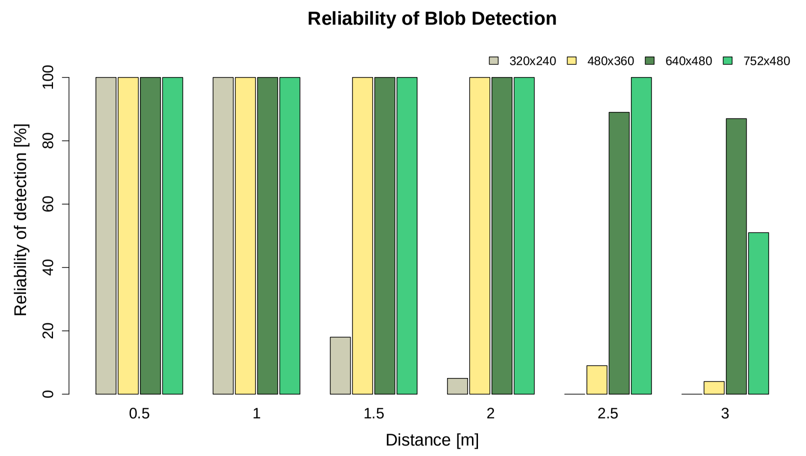


(c) d_o=3.5 cm, d_i=2.1 cm, d=7.5 cm

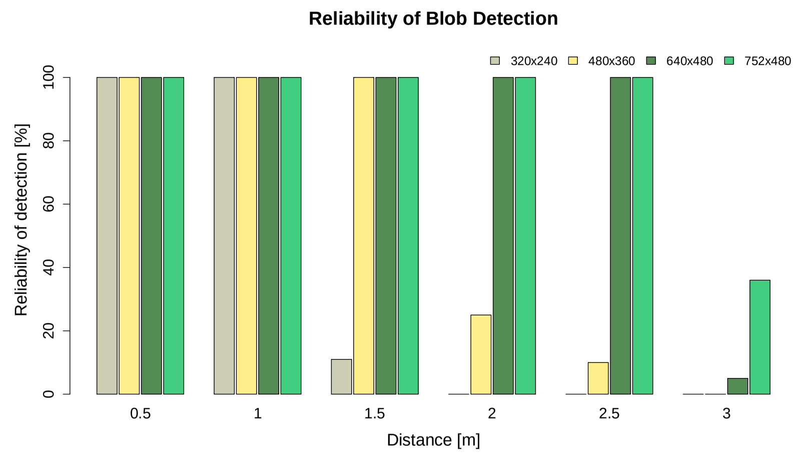


(d) d_o=3.5 cm, d_i=2.1 cm, d=5.5 cm

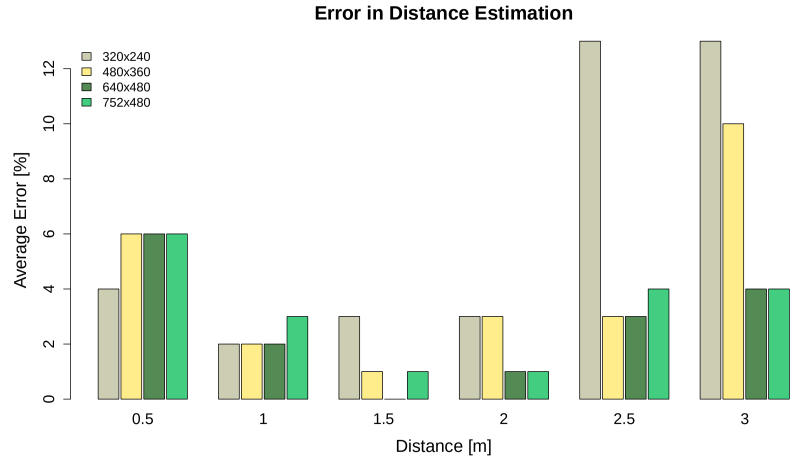
Figure 7: Detectability and localization precision in $x-$axis for the pattern with outer diameter 3.5 cm and two sizes of the squared bounding box.



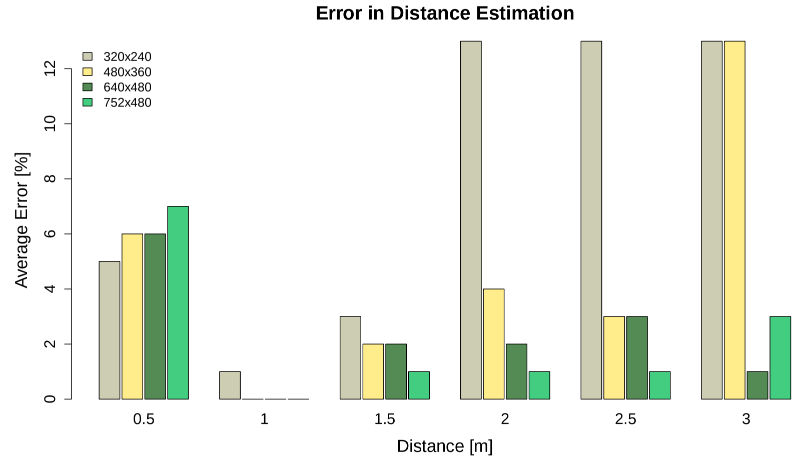
(a) d_o=7.0 cm, d_i=4.2 cm, d=9.0 cm



(b) d_o=6.0 cm, d_i=3.6 cm, d=8.0 cm



(c) d_o=7.0 cm, d_i=4.2 cm, d=9.0 cm



(d) d_o=6.0 cm, d_i=3.6 cm, d=8.0 cm

Figure 8: Detectability and localization precision in $x-$axis for the pattern with outer diameter 3.5 cm and two sizes of the squared bounding box.

In both cases, the usable distance of the camera module from the blob is about 2.5 m using the higher resolutions (640×480 or 752×480). However, using lower resolutions the detection of the smaller pattern (8×8 cm) is more difficult. In the case of the resolution 480×360 the reliable distance is about 2 m for the pattern with d_o=7 cm, and about 1.5 m for d_o=6 cm.

Regarding the results presented a smaller pattern can be used; however, with limited maximal distance between the pattern and the camera module.

Lessons Learned

Based on the experimental evaluation, the detectability of the blobs depends of several factors. The factors can be particularly addressed, which can eventually provide a better detectability of small patterns at longer distances than in the results presented above. Comments considering practical tuning of camera module, selected pattern, and the expected operational distances are presented here.

- The border between the dark and white parts of the pattern should be sharp enough. For example, the black ring on a white background needs white surroundings. The B/W separating threshold is computed from the mean values of the intensity of the detected segment; so, blurred borders can lead to a different number of detected and expected pixels resulting in not detected blob.
- The color of the pattern can be exchanged, e.g., a white ring on a black background.
- It may happened that a reflection of light from the surface of the black ring can lead to a white color in the image. Therefore, in such a situation using a different material can help.
- Although the ratio of the black and white discs of the used pattern seems to be good invariant, it is not guaranteed that such a ratio is not also detected for another objects. So, it may eventually happened that a false positive blob are reported as valid. This issue can be addressed by considering a different diameters of the discs, different ratio, or size of the pattern. In addition, it can also be addressed by more sophisticated sensor fusion techniques, e.g., considering previous position of the blob and expected maximal movement, or constraining shape of the segment as the current version of the algorithm considers only a bounding box of each detected segment.
- Having a strict range of distances where the pattern will be placed, the lens can be adjusted to provide a sharper image. However, once the focus length is changed, it is necessary to find new parameters of the camera.
- Here, it is worth to mention that the lens used does not provide an excellent image. With all respect, a cheap web camera provides (subjectively) better image (sharper) for longer distances.

3 System Configuration

The developed camera module is based on the Gumstix Overo board with the OMAP 3503 application processor. The operating system is a Linux based distribution created using the OpenEmbedded build framework [5]. The used Linux kernel is in version 2.6.34 tweaked for the Gumstix Overo board. The basic system configuration follows common installation of the Linux based operating system including system tools and initialization. The boot loader as well as the operating system itself support the console port of the Overo board; thus, any system administration can be made using a standard serial connection at 115200 baud rate, i.e., 8 bits, without parity, and one stop bit. For example, one can connect to the camera module using the provided USB adapter and the application `jerm` [6] or `minicom` [7]. The default account is `root` with an empty password.

In addition to the console, sshd daemon is started at the boot time, and it listens on all available interfaces. So, if a WiFi connection is available, one can connect to the camera module using the remote shell account. In the default settings, the user root can also connect via the ssh without a password.

A typical user does not work under root account, therefore, the local ssh client can be configured to prefer root for accessing the camera module as follows. Let the camera module IP address is recorded in the /etc/hosts file with the cam1 host name. Then, to simply connect to the module by the 'ssh cam1' command, the local configuration file of the ssh client can be adjust as it is shown in Listing 1.

```
Host cam1
User root
```

Listing 1: An example of ~/.ssh/config file.

Local time at the camera module is not preserved due to missing backup battery. Therefore to synchronize the system clock a ntpd daemon is started at the boot time. An example of the daemon configuration file is depicted in Listing 2, where the synchronization server is timerserver, which IP address is specified in /etc/hosts, see Listing 3 for an example.

```
# The driftfile must remain in a place specific to this
# machine - it records the machine specific clock error
driftfile /etc/ntp.drift

server timeserver

# Using local hardware clock as fallback
# server 127.127.1.0
fudge 127.127.1.0 stratum 14

# Defining a default security setting
restrict default nomodify nopeer
```

Listing 2: An example of the /etc/ntp.conf file.

```
127.0.0.1       localhost.localdomain       localhost
10.10.40.1      timeserver
```

Listing 3: An example of the /etc/hosts file with a local database of the host names.

The additional important configurations are related to the WiFi connection and initialization of the programs for capturing images and tracking. These are described in the following sections.

3.1 WiFi Configuration

Availability of a WiFi connection to the camera module is one of the most important system configuration, because it is the only possible way of the remote access to the module. However, a WiFi connection depends on the signal strength, and once a signal is lost, the configuration can be lost as well. Therefore, to avoid such misconfigured situation a simple script wifid is started at the boot time to periodically check WiFi connection availability. The script is started using the local startup configuration specified in the /etc/local.start file that is depicted in Listing 4. The wifid script itself is located at the /opt directory and its content is shown in Listing 5. The script calls two additional scripts. The first one is the /opt/check_wifi.sh script that performs check of the filled ESSID in the wlan0 interface, which is empty in a case of the lost connection. The second script is /opt/wlan_config, which is responsible for real configuration of the `wlan0` interface. In fact, the script `/opt/wlan_config` is a symbolic link to provide a flexibility of changing wifi configuration without necessity to rewrite the scripts and restart the `wifid` *service*. So, it is sufficient to create a new file and then to create a new symbolic link; however, it is recommended to consider console access or the checking script `check_wifi.sh`. It may happened the `check_wifi.sh` script can reconfigure the network using a wrong configuration file (e.g., due to missing `wlan_config`) and the connection to the camera module is lost, which makes impossible to change the configuration using the WiFi access. An example of the WiFi configuration using the *colos* network without any protections is depicted in Listing 7. This file is located at `/opt/wlan_colos` and it is the target of the `/opt/wlan_config` symbolic link.

```
#!/bin/bash
echo "Starting␣local␣..."

echo "Initializing␣GPIO␣..."
/opt/sb_init

echo "Starting␣wifi␣..."
/opt/wlan_config

ntpdate timeserver

nohup /opt/wifid > /dev/null &
```

Listing 4: An example of the local startup configuration file `/etc/local.start`.

```
PID=`pidof 'wifid'`
if [ -n "$PID" ]; then
   for pid in $PID; do
      if [ "$pid" != "$$" ]; then
         echo "wifid␣already␣running␣(pid␣$pid)" >&2
         exit 0
      fi
   done
fi

D=`date +'%Y-%m-%d %H:%M:%S'`
echo "$D␣Starting␣wifid␣..." >> /var/log/wifi
while true; do
   sleep 10
   /opt/check_wifi.sh 2>&1 >> /var/log/wifi
done
```

Listing 5: The `/opt/wifid` script for periodical checking of the WiFi connection.

```
LOGFILE=/var/log/wifi

ESSID=`iwconfig 2>/dev/null | \
   awk '/wlan0/ { print $4 }' | \
   sed -e 's/.*ESSID:"\([^"]*\)"/\1/'`

D=`date␣+'%Y-%m-%d␣%H:%M:%S'`

if␣[␣-n␣"$ESSID"␣];␣then
␣␣␣exit␣0
fi
```

```
echo␣"$D Restarting wifi ..."␣>>␣$LOGFILE

/opt/wlan_config␣&>/dev/null
```

Listing 6: The `/opt/check_wifi.sh` script for testing availability of the WiFi connection.

```
ESSID=colos
NET=10.10.40.
ADDRESS=174

ifconfig wlan0 up
iwconfig wlan0 essid x enc off mode ad-hoc
sleep 2
iwconfig wlan0 essid $ESSID
ifconfig wlan0 ${NET}${ADDRESS} netmask 255.255.255.0
route add default gw ${NET}1
```

Listing 7: The `/opt/wlan_colos` script for configuring WiFi connection to the *colos* network.

A summary of accessing parameters is depicted in Table 7.

3.2 Image Capturing

Image capturing functionality has been introduced in the first version of the camera module and it is briefly described in [4]. In this section, a more detailed description together with particular scripts are presented.

The image capturing is a service started at the boot time of the system. The initialization of the capturing consists of three system scripts located in the `/etc/init.d` directory. First, the camera driver is initialized by the `camera_init` service. This allows a custom specification of the driver (kernel module). The used initialization is depicted in Listing 8

```
#!/bin/sh

flash() {
   for i in `seq 1 1 5`
   do
      echo 1 > /sys/class/gpio/gpio145/value
      sleep 0.2
      echo 0 > /sys/class/gpio/gpio145/value
      sleep 0.2
   done
}

case $1 in
   start)
      rmmod mt9v032
      insmod /lib/modules/2.6.34/kernel/drivers/media/video/mt9v032.ko auto_exp=0
           auto_gain=0 low_light=0 hdr=0
      echo "mt9v032␣loaded"
      flash
      ;;
   stop)
      ;;
   restart)
```

```
        ;;
    status)
        flash
        ;;
    *)
esac
```

Listing 8: Camera driver initialization `/etc/init.d/camera_init`.

After that the `camera_event` service is started, which provides handling of the users' interaction with the module using selected GPIO pins. In this service, a named pipe `/tmp/caspa` is created and the `/opt/bin/capture_event.sh` script is executed, see Listing 9.

```
#!/bin/sh

pipe=/tmp/caspa

case $1 in
    start)
        rm -rf $pipe
        mkfifo $pipe
        /opt/bin/capture_event.sh 2>&1 1>/var/log/capture_event.log &
        ;;
    stop)
        ;;
    restart)
        ;;
    status)
        ;;
    *)
esac
```

Listing 9: Initialization of user input handling in the `/etc/init.d/camera_event` script.

Finally the capturing service itself is started by the `camera_capture` script, which is shown in Listing 10. The script contains simple user notification using 144 PIN of the GPIO; so, the red led is flashing for a while after starting the service[1].

```
#!/bin/sh

flash() {
    for i in `seq 1 1 5`
    do
        echo 0 > /sys/class/gpio/gpio144/value
        sleep 0.2
        echo 0 > /sys/class/gpio/gpio144/value
        echo 1 > /sys/class/gpio/gpio144/value
        sleep 0.2
    done
}

case $1 in
    start)
```

[1]The second version of the camera module does not include LEDs, therefore, the herein presented description is for the first version of the module.

```
      pid=`ps aux | grep "/bin/bash /opt/bin/capture.sh" | grep -v grep | awk '{
          print $2}'`
      if [ -n "$pid" ]
      then
         echo "camera capture already running"
      else
         /opt/bin/capture.sh 2>&1 1>/var/log/capture.log &
         echo "capture started"
      fi
      flash
      ;;
   stop)
      pid=`ps aux | grep "/bin/bash /opt/bin/capture.sh" | grep -v grep | awk '{
          print $2}'`
      if [ -n "$pid" ]; then
         kill -9 $pid;
         echo "capture.sh killed"
      else
         echo "capture not runnig"
      fi
      ;;
   restart)
      pid=`ps aux | grep "/bin/bash /opt/bin/capture.sh" | grep -v grep | awk '{
          print $2}'`
      if [ -n "$pid" ]; then
         kill -9 $pid;
         echo "capture.sh killed"
      else
         echo "capture not runnig"
      fi
      /opt/bin/capture.sh 2>&1 1>/var/log/capture.log &
      echo "capture started"
      flash
      ;;
   status)
      flash
      ;;
   *)
esac
```

Listing 10: Initialization of capturing service in `/etc/init.d/camera_capture` script.

The system scripts are started at the run level 5 that is configured using the `update-rc.d` tool, see Listing 11 for an example.

```
update-rc.d camera_init start 90 5 .
update-rc.d camera_event start 95 5 .
update-rc.d camera_capture start 99 5 .
```

Listing 11: Activation of the capturing service scripts.

Beside the system service scripts, the image capturing consists of two additional scripts. Although it may looks a bit complex, these scripts provide flexibility how to initiate the capturing process as they provide independent way of handling user events from the GPIO pins, where buttons can be connected. The scripts are placed at /opt/bin/capture_event.sh and /opt/bin/capture.sh files [2], and are depicted in Listings 12 and 13. The first script handles user inputs coming from GPIO signals using gpio-event tool. Even though gpio-event supports a kind of filtering of the input signal, it does not work reliably, and therefore, additional conditions of the input events are considered in the script. A communication between the scripts is realized using a named pipe, which should be created at the system boot time by the camera_event service. The named pipe is located at /tmp/caspa. The capture.sh script is an infinity loop in which a finite machine with two states is implemented. At the first state, the pipe is read for an incoming command. If the command is "start" a program for the capturing is started at the background and the script waits for a termination of the program. A detailed description of the capturing program is presented in Section 5.

```
#!/bin/bash

capture_pin=75
capture_edge=R

pipe=/tmp/caspa
capture_pid=/var/run/capture.pid

cmd_pattern=tcapture

led_on() {
   echo 0 > /sys/class/gpio/gpio145/value
}

led_off() {
   echo 1 > /sys/class/gpio/gpio145/value
}

events=/dev/gpio-event
last=0
gpio-event $capture_pin:b:30

while true; do
   echo "Watch for event"
   if [ ! -p $pipe ]
   then
      echo "Create pipe $pipe"
      rm -rf $pipe; mkfifo $pipe
   fi
   gpio-event $capture_pin:b:30

   led_on
   read < $events b e t

   if [ $e = "R" ]
   then
      dt=`echo "$t - $last" | bc`
      dt=`printf "%.0f" $dt`
      if [[ $b -eq $capture_pin && $dt -ge 1 ]]
      then
         echo "`date` INFO: Toggle capture"
```

[2] In fact, these files are symbolic links to capture_event.caspa.sh and capture.caspa.sh.

```
            start=1
            if [ -f $capture_pid ]
            then
                pid=`cat $capture_pid`
                if [[ -n $pid && -n `ps -p $pid | grep $pid | grep $cmd_pattern` ]]
                then
                    start=0
                fi
            fi

            led_off
            if [ $start -eq 0 ]
            then
                echo "`date` INFO: Stop capturing"
                kill `cat $capture_pid` 2>/dev/null
                killall -9 tcapture 2>/dev/null # for sure
                rm -rf $capture_pid  # for sure
            else
                echo "`date` INFO: Start capturing"
                exec 5<> $pipe
                read -t 0 -u 5 #read eventual content of the pipe
                echo "start" > $pipe
            fi
        fi

        last=$t
        echo "Last $last; read events"
        exec 5<> $events
        read -t 0 -u 5 #read eventual events
        echo "DONE"
    fi
done
```

Listing 12: Handling of user inputs coming GPIO pints.

```
#!/bin/bash

pipe=/tmp/caspa
capture_pid=/var/run/capture.pid
out_prefix=/media/mmcblk0p1
led_capture="/sys/class/gpio/gpio144/value"

on=0
off=1

while true; do
    if [ ! -p $pipe ]
    then
        echo "Create pipe $pipe"
        rm -rf $pipe; mkfifo $pipe
    fi
    echo $off > $led_capture
    echo "`date` INFO: led capture off, wait for cmd"
    read <$pipe cmd
    if [[ -n $cmd && $cmd = "start" ]]
    then
        d=`date +"%Y-%m-%d-%H.%M.%S"`
        echo "$cmd - $d"
        out=$out_prefix/$d
        mkdir $out
        if [ $? -eq 0 ]
        then
            echo "`date` INFO: output directory set to $out"
        else
            echo "`date` ERROR: cann't create output directory $out"
        fi
        /opt/bin/tcapture -c /opt/tcapture.cfg -l /opt/logger.caspa_capture.cfg --
            image-directory $out &
        pid=$!
        echo $pid > $capture_pid
        echo $on > $led_capture
        echo "`date` INFO: led capture on"
        wait $pid
        rm -rf $capture_pid
        echo "INFO: Capturing end with return value $?"
    fi
done
```

Listing 13: Starting of the capturing.

The main advantage of the independent handling of event and capturing is that the capturing can be managed remotely from the terminal by the following commands:

- `echo start > /tmp/caspa` - starts the capturing;
- `killall -9 tcapture` - terminated the capturing program.

This is very handful, if the buttons are not in a reachable distance, e.g., a camera module is placed at an UAV in a high altitude, or the module does not have buttons at all, e.g. like in the second version of the camera module.

3.3 Initialization of the Tracker Server

Initialization of the tracker server is very similar to the capturing program. It is done using the system service `/etc/init.d/tracker`, which is shown in Listing 14. The service is added to the run level 5 by the command `update-rc.d tracker start 99 5 .` The script starts (or eventually stop) the infinity loop in which the `tracker` program is repeatedly started, see Listing 15. The loop allows to restart the tracker with a different configuration, which is provided by the files `tracker.cfg`, `cam.cfg`, and `cam.m`. Besides, the loop also resolves situations in which the tracker program is suddenly terminated, e.g., due to some unknown bug.

```
#!/bin/sh

case $1 in
    start)
        pid=`ps aux | grep "/bin/sh /opt/tracker/tracker.sh" | grep -v grep | awk
            '{print $2}'`
        if [ -n "$pid" ]
        then
            echo "tracker is already running"
        else
            /opt/tracker/tracker.sh 2>&1 1>/var/log/tracker.log &
            echo "tracker started"
        fi
        ;;
    stop)
        pid=`ps aux | grep "/bin/sh /opt/tracker/tracker.sh" | grep -v grep | awk
            '{print $2}'`
        if [ -n "$pid" ]; then
            kill -9 $pid;
            killall -9 tracker.sh;
            kill -9 `pidof /opt/tracker/tracker`
            echo "tracker.sh killed"
        else
            echo "tracker is not runnig"
        fi
        ;;
    restart)
        pid=`ps aux | grep "/bin/sh /opt/tracker/tracker.sh" | grep -v grep | awk
            '{print $2}'`
        if [ -n "$pid" ]; then
            kill -9 $pid;
            echo "tracker.sh killed"
            killall -9 tracker.sh;
            kill -9 `pidof /opt/tracker/tracker`
        else
            echo "tracker is not runnig"
        fi
        /opt/tracker/tracker.sh 2>&1 1>/var/log/tracker.log &
        echo "tracker started"
        ;;
    status)
        ;;
    *)
esac
```

Listing 14: Tracker service initialization.

```
#!/bin/sh

prefix=/opt/tracker

while true; do
    ls -l $prefix/tracker
    ls -l $prefix/tracker.cfg
    ls -l $prefix/cam.cfg
    ls -l $prefix/cam.m
    echo "`date` INFO: Start tracker"
    $prefix/tracker --config $prefix/tracker.cfg --camera-control-settings $prefix
        /cam.cfg --camera-parameters $prefix/cam.m --logger-config $prefix/logger.
        cfg
    echo "`date` INFO: Tracker has been terminated"
done
```

Listing 15: Tracker program execution.

```
/opt/tracker:
cam.cfg
cam.m->etc/cam4-480x360.m
etc
logger.cfg
tcapture
tracker->tcapture
tracker.cfg->etc/tracker-480x360.cfg
tracker.sh

/opt/tracker/etc:
cam4-320x240.m
cam4-480x360.m
cam4-640x480.m
cam4-752x480.m
tracker-320x240.cfg
tracker-480x360.cfg
tracker-640x480.cfg
tracker-752x480.cfg
```

Listing 16: Directory layout of the tracker program.

The tracker program is identical binary file as the capturing program `tcapture`, just a different configuration is used. The behaviour of the program is also changed according to the name of the binary file, i.e., if the name is `tracker` or `ttracker` the behaviour of the program is forced to act as a tracker providing information about the tracked object. In fact, the `tracker` file is a symbolic link with the target to the `tcapture` binary, see a content of the `opt/tracker` directory showed in Listing 16. The origin of the program can be printed by a `-v` command arguments as can be seen in Listing 17.

```
root@cam4:/opt/tracker# ./tracker -v
Caspa capture application 0.1
Id: $Id: tcapture.cc 125 2012-01-10 20:12:27Z honza $
root@cam4:/opt/tracker#
```

Listing 17: Printed version of the `tracker` program.

The configuration of the `tracker` program (and the `tcapture` program as well) consists of the following four configuration files.

- `tracker.cfg` - main configuration file of the program in a format for the Boost.Program_options [8];
- `logger.cfg` - a specific configuration for the log4cxx logger [9];
- `cam.cfg` - values of the camera control settings in a v4l2 "*format*" [10], see Section 5 for further details;
- `cam.m` - identified parameters of the camera found as a result of the calibration [1].

All the program options can be specified in the main configuration file; however, they can also be specified as a program's command line arguments, which have a higher priority. Detailed description of the most important parameters is presented in Section 5. The main tracker directory `/opt/tracker` contains several configuration files and files with the camera parameters for several resolutions. The selected default resolution is 480×360; thus, the particular configurations files are the target for the symbolic links `cam.m` and `tracker.cfg`.

The tracker can be used simultaneously with the capturing program; however, the camera device (i.e., `/dev/video0`) can be opened only by a single application. Therefore, the tracker server monitors "*connected*" clients and closes the device once none of clients is connected[3]. The opened camera device is signalized by a red LED at the Overo Caspa board.

A summary of tracker parameters is depicted in Table 7.

4 Camera Settings

The Gumstix Caspa camera is accessible using the v4l2 programming interface [10] provided by the `mt9v032` driver, and therefore, it can be used like any other v4l device. However, the driver supports only limited control settings that are implemented by the driver. A list of available settings is depicted in Listing 18. Here, it is worth to mention that the version *r100* of the `mt9v032` driver (the current version at the time of developing the camera module) contains a small bug, which prevent to use the control settings. The bug has been fixed during development of the camera module and all the listed settings can be used.

```
=== VIDIOC_QUERYCAP ===

driver: omap3
card: omap3/mt9v032//
version: 0
capabilites:
        V4L2_CAP_VIDEO_CAPTURE
        V4L2_CAP_STREAMING

=== VIDIOC_ENUMINPUT ===

Input 1
        name: camera
        type: V4L2_INPUT_TYPE_CAMERA
        status: (not all checked)

=== VIDIOC_ENUM_FMT (type = V4L2_BUF_TYPE_VIDEO_CAPTURE) ===

Format 1:
        description: UYVY, packed
        pixelformat: 0x59565955 (UYVY)
Format 2:
        description: YUYV (YUV 4:2:2), packed
        pixelformat: 0x56595559 (YUYV)
Format 3:
        description: Bayer10 (GrR/BGb)
        pixelformat: 0x30314142 (BA10)

=== VIDIOC_QUERYCTRL ===

Control: 0
```

[3]The server is UDP; thus, the connection is monitored using an explicit unsubscription of the connected clients.

```
          id: V4L2_CID_BASE + 0
          name: Brightness
          type: integer
          minimum: 0
          maximum: 255
          step: 1
          default: 1
Control: 1
          id: V4L2_CID_BASE + 1
          name: Contrast
          type: integer
          minimum: 0
          maximum: 255
          step: 1
          default: 16
Control: 2
          id: V4L2_CID_BASE + 17
          name: Exposure
          type: integer
          minimum: 2
          maximum: 480
          step: 1
          default: 480
          flags: 0x00000020
Control: 3
          id: V4L2_CID_BASE + 18
          name: Automatic Gain
          type: boolean
          default: 1
Control: 4
          id: V4L2_CID_BASE + 19
          name: Analog Gain
          type: integer
          minimum: 16
          maximum: 64
          step: 1
          default: 16
          flags: 0x00000020
Control: 5
          id: V4L2_CID_BASE + 20
          name: Flip Horizontally
          type: boolean
          default: 0
Control: 6
          id: V4L2_CID_BASE + 21
          name: Flip Vertically
          type: boolean
          default: 0
Control: 7
          id: V4L2_CID_BASE + 31
          name: Color Effects
          type: other: 3
          minimum: 0
          maximum: 2
          step: 1
```

```
        default: 0
Control: 8
        id: V4L2_CID_CAMERA_CLASS_BASE + 1
        name: Automatic Exposure
        type: boolean
        default: 1
```

Listing 18: Gumstix Caspa camera control settings accessible by the V4L2 interface provided by the mt9v032 driver.

In addition to this issue, the current version of the mt9v032 does not provide sufficient stability for capturing an image in the BA10 format, that is why the V4L2_PIX_FMT_YUYV (YUYV) format is used.

Two above mentioned applications (the tcapture and tracker) use a simple text file to store desired control settings of the camera. The format follows a simple interface to control a device using the ioctl system call and definitions in the V4L2. Each control setting is at one line, where three columns are separated by a space (or any whitespace) character. The first column is a string denoting the name of V4L2 request (in a human readable form), the second column is integer representation of the request, and finally the third column is the requested value. An example of the file is shown in Listing 19.

```
V4L2_CID_BRIGHTNESS 9963776 2
V4L2_CID_CONTRAST 9963777 16
V4L2_CID_GAIN 9963795 16
V4L2_CID_EXPOSURE_AUTO 10094849 0
V4L2_CID_EXPOSURE 9963793 375
V4L2_CID_AUTOGAIN 9963794 1
V4L2_CID_COLORFX 9963807 1
```

Listing 19: Gumstix Caspa camera control settings accessible by the V4L2 interface provided by the mt9v032 driver.

4.1 Practical Tips for Using the Overo Caspa Camera

The usage of the camera is pretty much straightforward; however, it may happened that one can be surprised by the device behaviour due to implementation details. Even though, such facts can be found in the documentation, they are presented here to provide a reminder of them.

4.1.1 Camera Control Settings

Although the above defined control camera settings files (see Listing 19) is straightforward, the behaviour of the camera driver needs a special consideration regarding the settings related to the auto exposure and auto gain controls. The manual settings of the exposure or gain take effect only if the auto exposure or auto gain, respectively, is disabled. It means that disabling the automatic settings must be done prior setting the desired values. Regarding the above described applications and the configuration file, this can be done by an order of the control settings in the file. However, the applications developed fix this behaviour.

4.1.2 Auto exposure

The auto exposure might reduce the frame rate as low as 15 frames per second as it is noted at the Wiki page of the Caspa board [11]. In such cases, the real required time for processing image together with time for reporting local position of the detected blob and the time need to capture the image are similar. In a consequence, this significantly reduce the number of processed new images per second. Therefore, it is recommended to use manual settings of the exposure, because it provides higher FPS as it is presented in Section 2.2.1.

5 Applications Settings

This section presents a more detailed description of the main applications for the camera module. The description is not exhaustive because the applications contains many settings and options, which are mostly for developing and debugging purposes. Therefore, the description is focused on the most important parts from a user point of view. Additional functions can be observed from the source codes of the applications.

5.1 Tracker Server

The tracker server acts as a daemon providing information about the tracked object. The application itself is the same binary as the capturing program, and therefore, particular program options have to be set. The most important options defining the role of the program are also set if the program name is `tracker` or `ttracker`. The default configuration of the tracker server startup is described in Section 3, and more detailed description of the all program options is provided by the program itself using `-h` or `--help` options (the Boost.Program_options [8] library is used for parsing the options).

```
camera-control-settings=cam.cfg
logger-config=logger.cfg


use-circle-detector=1
circle-detector-tracking=1
grayscale=1

tracked-object-diameter=0.14

width=480
height=360
camera-parameters=cam4-480x360.m

camera-device=/dev/video0
fps=-1

server=1
tracker-server=1
tracker-server-port=9001

tracker-server-max-msg-size=1024
tracker-server-max-clients=2

tracker-server-close-camera-no-clients=1
```

Listing 20: An example of the tracker server configuration file (`tracker-480x360.cfg`)

An example of the tracker server configuration file is shown in Listing 20 and a description of the particular options is as follows.

- *System options*:

  - `logger-config=logger.cfg` - the configuration file for the log4cxx logger [9], it allows to control what type of messages are printed out and how (e.g., to standard output like a screen, or into a file).

- *Camera options*:
  - `camera-device=/dev/video0` - a file name representing the camera device.
  - `width=480` - requested width of captured images.
  - `height=360` - requested height of captured images.
  - `camera-control-settings=cam.cfg` - a file with particular camera control settings loaded and set before capturing and processing images.
  - `camera-parameters=cam4-480x360.m` - camera parameters found by the toolbox [1], the file is directly the Matlab file produced by the toolbox. An example of the file is shown in Listing 21.
  - `grayscale=1` - consider the grayscale images. The option will cause the capture images are in grayscale, which reduced time for the image conversion and also decrease the image processing time a bit.

- `Tracker / Detector settings`
  - `use-circle-detector=1` - use detector of the blob based on the B/W ring pattern.
  - `circle-detector-tracking=1` - if set, the previous position of the detected blob is used to speed up the segment searching process. If the blob is detected continuously, the tracking can significantly reduced the required processing time.
  - `fps=-1` - desired number of processed images per second, for values less than zero images are processed as fast as possible.
  - `tracked-object-diameter=0.14` - the outer diameter of the B/W ring pattern.

- *Tracker server options*
  - `server=1` - set the application to act as a server.
  - `tracker-server=1` - set the application to act as the tracker server.
  - `tracker-server-port=9001` - a port on which the server listen for clients (using UDP).
  - `tracker-server-close-camera-no-clients=1` - close the camera device if none of the clients is connected (i.e., all clients close the connection to the server), it allows to use tracker server simultaneously with other applications accessing the camera device. If the option is set to *false*, the camera device is opened all the time the tracker server is running. Notice, if the device is opened, the red LED at the camera board is switched on.

  - `tracker-server-max-msg-size=1024` - A maximum message size considered for the communicating with clients. (*The default value should be sufficient in most cases.*)
  - `tracker-server-max-clients=2` - A number of clients, which can connect to the server and to which the information about the tracker object are sent. (*The default value should be sufficient in most cases.*)

```
% Intrinsic and Extrinsic Camera Parameters
%
% This script file can be directly excecuted under Matlab to recover the camera
    intrinsic and extrinsic parameters.
% IMPORTANT: This file contains neither the structure of the calibration objects
    nor the image coordinates of the calibration points.
%            All those complementary variables are saved in the complete matlab
    data file Calib_Results.mat.
% For more information regarding the calibration model visit http://www.vision.
    caltech.edu/bouguetj/calib_doc/


%-- Focal length:
fc = [ 401.944133237755523 ; 471.921886874863844 ];

%-- Principal point:
cc = [ 234.749987607683408 ; 203.779018171760271 ];

%-- Skew coefficient:
alpha_c = 0.000000000000000;

%-- Distortion coefficients:
kc = [ -0.414761516174968 ; 0.191135963054818 ; 0.000917014079415 ;
    0.000757979352098 ; 0.000000000000000 ];

%-- Focal length uncertainty:
fc_error = [ 0.380510813779749 ; 0.471484705407820 ];

%-- Principal point uncertainty:
cc_error = [ 0.780420962614166 ; 0.760419558228052 ];

%-- Skew coefficient uncertainty:
alpha_c_error = 0.000000000000000;

%-- Distortion coefficients uncertainty:
kc_error = [ 0.002961741054041 ; 0.008264122674892 ; 0.000349479648818 ;
    0.000261311125835 ; 0.000000000000000 ];

%-- Image size:
nx = 480;
ny = 360;


%-- Various other variables (may be ignored if you do not use the Matlab
    Calibration Toolbox):
%-- Those variables are used to control which intrinsic parameters should be
    optimized

n_ima = 22;                                         % Number of calibration
    images
est_fc = [ 1 ; 1 ];                                 % Estimation indicator of
     the two focal variables
est_aspect_ratio = 1;                           % Estimation indicator of the
    aspect ratio fc(2)/fc(1)
```

```
center_optim = 1;                                  % Estimation indicator of
    the principal point
est_alpha = 0;                                     % Estimation indicator of
    the skew coefficient
est_dist = [ 1 ; 1 ; 1 ; 1 ; 0 ];      % Estimation indicator of the distortion
   coefficients
```

Listing 21: An example of the camera parameters found by the Matlab toolbox [1].

5.2 Tracker Client

The tracker server uses a simple UDP protocol for transmitting position of the tracked object. The position is encapsulated in the `imr::STrackedObject` structure, see Listing 22 and it is serialized using XDR [12]. A client has to connect to the server, which will then starts sending positions according to configuration, e.g., desired fps. Although the client address can be retrieved from the incoming request at the server side, such an address can be unreachable from the server because of poorly configured network environment, or too complex system setup (e.g., a client can have multiple network interfaces and an address resolving service can be unreachable from the camera module). Therefore, to guarantee the correct address to deliver the messages from the server, the clients have to send particular address and port in its initial message to the server. Then, the server will use the address for transmitting the messages; thus, the client have to read from a socket binded into the provided address and port.

```
/*
 * File name: tracked_object.h
 * Date:      2012/01/07 14:04
 * Author:    Jan Faigl
 */

#ifndef __TRACKED_OBJECT_H__
#define __TRACKED_OBJECT_H__

namespace imr {

   struct STrackedObject {
      bool valid; //true if the blob info is valid
      float x;
      float y;
      float z;
      float pitch;
      float roll;
      float yaw;
      float pixel_ratio; //ratio of the expected and detected pixels of the
      float bw_ratio; //ratio of the detected color (grays)
   };

} //end namespace imr;


#endif

/* end of tracked_object.h */
```

Listing 22: A structure send by the tracker server (`tracked_object.h`)

The address including the port also serves as a client identification. The identification is used in cases, when a connection between the server and client is suddenly lost, e.g., due to limited range of WiFi, and the client is reconnected to the server. In such a case, the server does not know the client is not available as the UDP transmit protocol is used. So, once the client from the previously registered address is reconnected to the server, the subscribing request is accepted. The UDP protocol is also the reason why an explicit *disconnect* message has to be send to the server, which can then eventually close (and release) the camera device.

All the communication primitives and marshaling / unmarshaling functions are part of the provided library. An example of the library usage is depicted in Listing 23, where a source code of a simple application called `tclient` is presented.

```
/*
 * File name: tclient.cc
 * Date:      2012/01/06 07:27
 * Author:    Jan Faigl
 */

#include <iostream>
#include <cstring>
#include <cstdlib>

#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>

#include <unistd.h>
#include <signal.h>

#include "tracked_object.h"
#include "serialization.h"
#include "tracker_client.h"


/// ----------------------------------------------------------------------------
imr::CTrackerClient* client = 0;
bool quit = false;

/// ----------------------------------------------------------------------------
void terminate_handler(int sig) {
   if (client) {
      std::cerr << "Terminate client!" << std::endl;
      quit = true;
   }
}

/// ----------------------------------------------------------------------------
void print_object(unsigned long id, unsigned time, const imr::STrackedObject& obj
   ) {
   std::cout << "Object id: " << id << " time: " << time
      << " valid: " << obj.valid
      << " coords: " << obj.x << " " << obj.y << " " << obj.z
      << " rotations: " << obj.pitch << " " << obj.roll << " " << obj.yaw
      << " pixel_ratio: " << obj.pixel_ratio << " bw_ratio: " << obj.bw_ratio
      << std::endl;
```

```
}

/// - main function ------------------------------------------------------------
int main(int argc, char* argv[]) {
   int ret = -1;

   const std::string host = argc > 1 ? std::string(argv[1]) : "localhost";
   const int server_port  = argc > 2 ? atoi(argv[2])        : 9001;
   const std::string myip = argc > 3 ? std::string(argv[3]) : "127.0.0.1";
   const int client_port  = argc > 4 ? atoi(argv[4]) : 9005;
   const bool BLOCKING_READ = argc > 5 ? atoi(argv[5]) : false;

   client = new imr::CTrackerClient(host, server_port, myip, client_port);
   signal(SIGINT, terminate_handler); /// register signal handler
   std::cerr << "INFO:"
      << " connect to the tracker " << host << ":" << server_port
      << " using address " << myip  << ":" << client_port
      << std::endl;
   // allocate buffer for the message
   const int SIZE = 2 * (sizeof(imr::STrackedObject) + 2 * sizeof(unsigned long))
      ;
   char buf[SIZE];
   unsigned long id;
   unsigned long time;
   imr::STrackedObject obj;

   if (BLOCKING_READ) {
      //An example of blocked receiving
      if (client->connect()) {
         std::cerr << "INFO: Connected to the tracker server" << std::endl;
      } else {
         std::cerr << "WARN: Connection to the tracker server fail!" << std::endl
            ;
      }
      quit = !client->isConnected();
      while(!quit) {
         std::cerr << "quit: " << quit << std::endl;
         int l = client->receive(&buf[0], SIZE, false);
         if (l > 0 and imr::unpack(&buf[0], SIZE, id, time, obj) > 0) {
            print_object(id, time, obj);
         } else if (l < 0 ) {
            std::cerr << "WARN: Error in mesage receive" << std::endl;
            usleep(100*1000);
         }
      } //end receiving loop
   } else {
      // An example of non-blocked receive with reconnection
      const int MSG_WAIT = 10; //in ms
      const int MAX_CONNECT_FAILS = 10;
      const int MAX_RECEIVE_FAILS = 10;
      int state = 0;
      int numConnectRetry = 0;
      int numReceiveRetry = 0;

      while(!quit) {
            const int prev_state = state;
            switch(state) {
               case 0: //connecting to the server
                  if (client->connect()) {
                     numConnectRetry = 0;
                     numReceiveRetry = 0;
                     state = 1; //switch to receive the message
                  } else {
                     numConnectRetry++;
                     std::cerr << "Connection to server fails " << numConnectRetry
                        << " times!" << std::endl;
                  }
                  if (numConnectRetry >= MAX_CONNECT_FAILS) { //decide what to do
                     std::cerr << "Connection to server fails too many times!" <<
                         std::endl;
                     quit = true; //exit the program
                  }
                  break;
               case 1: //try to receive message
                  while(client->receive(&buf[0], SIZE, true) > 0) { //read all
                      messages
                     if (imr::unpack(&buf[0], SIZE, id, time, obj) > 0) {
                        numReceiveRetry = 0;
                        print_object(id, time, obj);
                     }
                  }
                  if (numReceiveRetry < MAX_RECEIVE_FAILS) { //wait for a while
                     numReceiveRetry++;
                     usleep(MSG_WAIT * 1000);
                  } else { //
                     std::cerr << "Receive fails " << numReceiveRetry << " times, 
                         try to reconnect" << std::endl;
                     client->disconnect();
                     state = 0;
                  }
                  break;
            } //end switch
            if (prev_state != state) {
               std::cerr << "Change state " << prev_state << "->" << state << std::
                   endl;
            }
         }
         client->disconnect();
      }
   return ret;
}

/* end of tclient.cc */
```

Listing 23: A simple client of the tracker server.

5.3 `tcapture` - Capturing mode

The capturing program is called `tcapture` and it provides many functions and modes, e.g., the tracker server described above. In this section, particular program options related to the simple capturing of images are described. A basic configuration file is shown in Listing 24 and a detailed description of the options is following.

- *Program options*:
  - `logger-config=logger.cfg` - the configuration file for the log4cxx logger [9].
  - `camera-device=/dev/video0` - a file name representing the camera device.
  - `camera-control-settings=cam.cfg` - a file with particular camera control settings loaded and set before capturing and processing images.
  - `fps=10` - a desired FPS, it should be set to a reasonable value, because the images are stored in jpeg format, which reduces the required space; however, it is time consuming. The recommended values are around 10 or less than 15 fps for resolution 480×360. The achievable number of captured images per second also depends on the quality of jpeg images, resolution, and settings of the auto exposure, see Section 4.1.2.
- *Image options*:
  - `width=480` - requested width of captured images.
  - `height=360` - requested height of captured images.
  - `grayscale=1` - consider the grayscale images. The option will cause the capture images are in grayscale, which reduced time for the image conversion.
  - `jpeg-quality=75` - a compression parameter of the jpeg images, lower values are less computational intensive.
- `Image saving options`:
  - `image-directory=images/` - the output directory into which images are saved.
  - `image-pattern=%09u.jpg` - defines a name of the image files (following `printf (3)`[4]), the input variable is the number of the captured image or time in milliseconds from the start of the capturing (if the `enable-capture-timestamp` is 1).
  - `enable-capture-timestamp=1` - determines if the image names will be derived from the number of captured images or time.

```
logger-config=logger.cfg
camera-device=/dev/video0
camera-control-settings=cam.cfg
server=false

fps=10
width=480
height=360
jpeg-quality=75
grayscale=1

```

[4]see man 3 printf

```
image-pattern=%09u.jpg
image-directory=images/

#using timestamping in milliseconds
enable-capture-timestamp=1
```

Listing 24: A configuration for the capturing program.

In addition to the presented simple capturing mode, the `tcapture` program supports another mode suitable for evaluating performance of the detectors being developed. In this alternative mode, the images can be stored in a raw format and also information about the detected segments and blobs can be stored into additional files. The mode is mainly for developing and it is out of the scope of this report, therefore, it is not described here. A reader interested in this mode is referred to the source codes.

5.4 Applications for Remote Access to the Camera

In this section, a brief description of applications allowing remote access is presented. These applications are mainly for testing and development purposes, and therefore, only their basic configurations are presented. The main motivation of the applications is to provide an easy way for developing new algorithms and their testing with real images captured by the Overo Caspa camera. Moreover, it also allows to tune camera control settings and creates the configuration file described in Section 4.

```
server=true
port=9009
poll-timeout=200
default-buffer-size=8192

camera-control-settings=cam.cfg

use-circle-detector=1

number-blobs=2
rgb-cube-size=64
```

Listing 25: A configuration file for the `tcapture` program in the server mode.

The applications follow a client/server architecture, where the server is the `tcapture` program described above, which is in the *server* mode. An example of the configuration file is presented in Listing 25. The purpose of the already described options is same as above, as it is the same application, therefore only the newly introduced options are described in the following list.

- *System options*:
  - `server=true` - start a server.
  - `port=9009` - port at which the server is listening (the transport protocol is TCP).
  - `poll-timeout=200` - a period used for checking new incoming messages from a client.
  - `default-buffer-size=8192` - an initial buffer size used for transmitted messages. (*The default value should be sufficient in most cases.*)
- *Detectors' options (selected)*:
  - `use-circle-detector=1` - use detector of the blob based on the B/W ring pattern (otherwise color blob finder is used).

- `number-blobs=2` - the number of considered blobs for the color segmentation.
- `rgb-cube-size=64` - a dimension of the RGB cube used in the color segmentation.

The client application is called `tcam` and has many options, see Listing 27. A part of the options is related to the blob finders. In particular the current version of `tcam` contains two blob finders: the color based and B/W ring pattern based detectors. The main features provided by the `tcam` application can be summarized as follows.

- Testing blob finder algorithms using previously save images.
- Testing performance of blob finders.
- On-line capturing images using the camera module.
- On-line adjustment of the camera control settings and creation of settings file (`cam.cfg`).
- On-line testing of the blob finder algorithms running at the camera module.

```
cam-server=cam1
cam-port=9009

process-on-refresh=1

width=640
height=480
jpeg-quality=85

image-directory=images
image-pattern==%04d.jpg

use-circle-detector=1
```

Listing 26: Selected program options of the `tcam` application.

The program options are pretty much similar to the `tcapture` application (as most of them is related to the particular blob finder algorithms), therefore only selected of them are shown in an example of the configuration file in Listing 26.

- `cam-server=cam1` - a host name of the `tcapture` application running in the server mode.
- `cam-port=9009` - a port of the `tcapture` server.
- `process-on-refresh=1` - if enabled, the selected blob finder is called for a new image (read from the directory or received from the camera module).
- `image-directory=images/` - the input/output directory where images are stored. List of images presented in the directory is created after the `tcam` startup. Then, images can be used for testing the image processing algorithms. Newly received images from the `tcapture` server are stored into this directory in an incremental way (using a counter of the captured images).
- `image-pattern=%04d.jpg` - defines a name of the image files (following `printf (3)`[5]), for which the application is looking at the startup. The same pattern is used to store the images.

[5] see man 3 printf

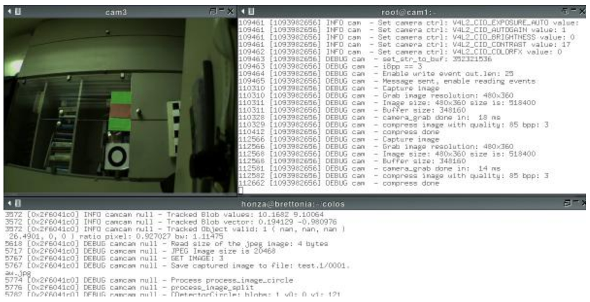
Although the `tcam` application can be used for benchmarking algorithms being developed, probably its main feature is an interactive mode. In this mode, a window containing an image (loaded from the given image directory or captured from the camera module) is shown. From a user point of view, the most important are interactive controls of the application that are based on keyboard commands. A summary of the keys / controls are as follows, a key is represented by its usual character.

- *Main controls*:
  - `'q'` - quit the application.
  - `'b'`, `'n'` - load previous / next image from the image directory.
  - `'p'` - process the current image using the selected blob finder.
  - `'l'` - perform blob finding step on the current image at the camera module, only the resulting found segments are shown in the window (the image is not updated - received from the camera module).
- *Color based blob finder related controls*:
  - `'r'` - reset RGB cube of the color blob finder.
  - `'s'` - save the current RGB cube to a file specified in the `cube-rgb` program option.
  - `'k'` - download the current RGB cube to the camera module.
  - *mouse click* - add the color at the cursor position to the RGB cube.
- *Camera server related controls*:
  - `'c'` - connect to the `tcapture` server.
  - `'o'` - open the camera at the server (connection has to be already establish).
  - `'i'` - request information from the camera server.
  - `'f'` - request an image from the camera in the jpeg format, the received file is stored in the image directory.
  - `'g'` - request a raw image from the camera, the received file is stored in the image directory in the bmp format.
  - `'1'`, `'2'` - increase / decrease the camera **contrast** about 1 (and about 10 if `shift` key is pressed).
  - `'3'`, `'4'` - increase / decrease the camera **brightness** about 1 (and about 10 if `shift` key is pressed).
  - `'5'`, `'6'` - increase / decrease the camera **exposure** about 1 (and about 10 if `shift` key is pressed).
  - `'7'`, `'8'` - increase / decrease the camera **gain** about 1 (and about 10 if `shift` key is pressed).
  - `'9'` - enable / disable the camera **auto exposure**.
  - `'0'` - enable / disable the camera `auto gain`.
  - `'e'` - change the color mode of the camera (three values are supported `V4L2_COLORFX_NONE`, `V4L2_COLORFX_BW`, and `V4L2_COLORFX_SEPIA`).
  - `'d'` - request the current control settings of the camera.
  - `'z'` / `'x'` - set / store the camera control settings from / to the file specified in `camera-control-settings` program options (at the client side).

– `'-'` / `'='` save / load the camera control settings at the server side using the file specified in the `camera-control-settings` program options of the `tcapture` program.

The full list of the control keys can be found in the `cam_app.cc` file in definition of the `CCamApplication::-` `-start(void)` method of the `tcam` application sources.

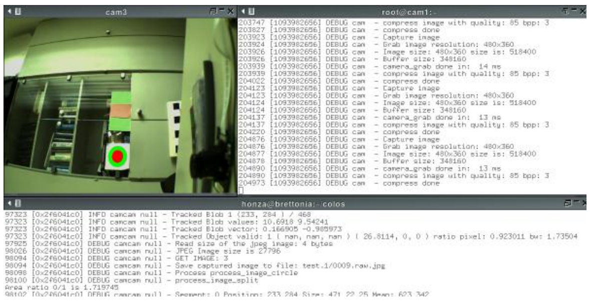
Examples of the demonstrative usage of the `tcam` application is shown in Fig. 9. In each sub-figure, the top right text window is a remote session at the camera module (the output of the `tcapture` program in the server mode). The bottom text window is output of the `tcam` application and the image at the left is an actual image with visualized results of the found segments forming the detected blob.



```
109461 [1093982656] INFO can  - Set camera ctrl: V4L2_CID_EXPOSURE_AUTO value:
109461 [1093982656] INFO can  - Set camera ctrl: V4L2_CID_AUTOGAIN value: 1
109461 [1093982656] INFO can  - Set camera ctrl: V4L2_CID_BRIGHTNESS value: 0
109461 [1093982656] INFO can  - Set camera ctrl: V4L2_CID_CONTRAST value: 17
109462 [1093982656] INFO can  - Set camera ctrl: V4L2_CID_COLORFX value: 0
109463 [1093982656] DEBUG cam  - set_str_to_buf: 352321536
109463 [1093982656] DEBUG cam  - iBpp == 3
109464 [1093982656] DEBUG cam  - Enable write event out.len: 25
109465 [1093982656] DEBUG cam  - Message sent, enable reading events
110310 [1093982656] DEBUG cam  - Capture image
110310 [1093982656] DEBUG cam  - Grab image resolution: 480x360
110311 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
110311 [1093982656] DEBUG cam  - Buffer size: 348160
110328 [1093982656] DEBUG cam  - camera_grab done in:  18 ms
110329 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
110412 [1093982656] DEBUG cam  - compress done
112566 [1093982656] DEBUG cam  - Capture image
112566 [1093982656] DEBUG cam  - Grab image resolution: 480x360
112568 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
112568 [1093982656] DEBUG cam  - Buffer size: 348160
112581 [1093982656] DEBUG cam  - camera_grab done in:  14 ms
112582 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
112662 [1093982656] DEBUG cam  - compress done
```

```
3572 [0x2f6041c0] INFO camcam null - Tracked Blob values: 10.1682 9.10064
3572 [0x2f6041c0] INFO camcam null - Tracked Blob vector: 0.194129 -0.980976
3572 [0x2f6041c0] INFO camcam null - Tracked Object valid: 1 ( nan, nan, nan )
 26.4901, 0, 0 ) ratio pixel: 0.927027 bw: 1.11475
5618 [0x2f6041c0] DEBUG camcam null - Read size of the jpeg image: 4 bytes
5717 [0x2f6041c0] DEBUG camcam null - JPEG Image size is 20468
5767 [0x2f6041c0] DEBUG camcam null - GET IMAGE: 3
5767 [0x2f6041c0] DEBUG camcam null - Save captured image to file: test.1/0001.
aw.jpg
5774 [0x2f6041c0] DEBUG camcam null - Process process_image_circle
5776 [0x2f6041c0] DEBUG camcam null - process_image_split
5782 [0x2f6041c0] DEBUG camcam null - CDetectorCircle: blobs: 1 v0: 0 v1: 121
```

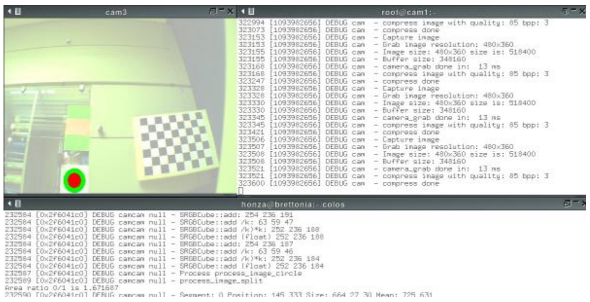
(a) a received image from the camera



```
203747 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
203827 [1093982656] DEBUG cam  - compress done
203923 [1093982656] DEBUG cam  - Capture image
203924 [1093982656] DEBUG cam  - Grab image resolution: 480x360
203926 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
203926 [1093982656] DEBUG cam  - Buffer size: 348160
203939 [1093982656] DEBUG cam  - camera_grab done in:  14 ms
203939 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
204022 [1093982656] DEBUG cam  - compress done
204123 [1093982656] DEBUG cam  - Capture image
204123 [1093982656] DEBUG cam  - Grab image resolution: 480x360
204124 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
204124 [1093982656] DEBUG cam  - Buffer size: 348160
204137 [1093982656] DEBUG cam  - camera_grab done in:  13 ms
204137 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
204220 [1093982656] DEBUG cam  - compress done
204876 [1093982656] DEBUG cam  - Capture image
204876 [1093982656] DEBUG cam  - Grab image resolution: 480x360
204877 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
204878 [1093982656] DEBUG cam  - Buffer size: 348160
204890 [1093982656] DEBUG cam  - camera_grab done in:  13 ms
204890 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
204973 [1093982656] DEBUG cam  - compress done
```

```
97323 [0x2f6041c0] INFO camcam null - Tracked Blob 1 (233, 284 ) / 468
97323 [0x2f6041c0] INFO camcam null - Tracked Blob values: 10.6918 9.54241
97323 [0x2f6041c0] INFO camcam null - Tracked Blob vector: 0.166905 -0.985973
97323 [0x2f6041c0] INFO camcam null - Tracked Object valid: 1 ( nan, nan, nan ) ( 26.8114, 0, 0 ) ratio pixel: 0.923011 bw: 1.73504
97925 [0x2f6041c0] DEBUG camcam null - Read size of the jpeg image: 4 bytes
98026 [0x2f6041c0] DEBUG camcam null - JPEG Image size is 27796
98094 [0x2f6041c0] DEBUG camcam null - GET IMAGE: 3
98094 [0x2f6041c0] DEBUG camcam null - Save captured image to file: test.1/0009.raw.jpg
98098 [0x2f6041c0] DEBUG camcam null - Process process_image_circle
98100 [0x2f6041c0] DEBUG camcam null - process_image_split
Area ratio 0/1 is 1.719745
98102 [0x2f6041c0] DEBUG camcam null - Segment: 0 Position: 233 284 Size: 471 22 25 Mean: 623 342
```

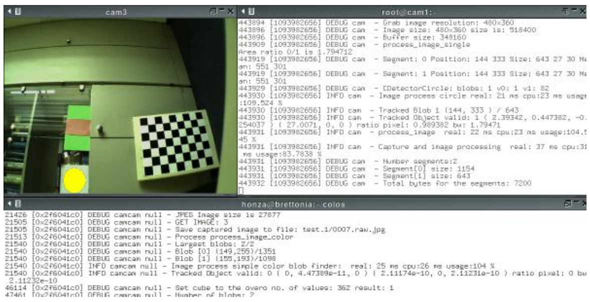
(b) locally detected ring pattern from the newly received image from the camera



```
322994 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
323073 [1093982656] DEBUG cam  - compress done
323153 [1093982656] DEBUG cam  - Capture image
323153 [1093982656] DEBUG cam  - Grab image resolution: 480x360
323155 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
323155 [1093982656] DEBUG cam  - Buffer size: 348160
323168 [1093982656] DEBUG cam  - camera_grab done in:  13 ms
323168 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
323247 [1093982656] DEBUG cam  - compress done
323328 [1093982656] DEBUG cam  - Capture image
323328 [1093982656] DEBUG cam  - Grab image resolution: 480x360
323330 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
323330 [1093982656] DEBUG cam  - Buffer size: 348160
323345 [1093982656] DEBUG cam  - camera_grab done in:  13 ms
323345 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
323421 [1093982656] DEBUG cam  - compress done
323506 [1093982656] DEBUG cam  - Capture image
323507 [1093982656] DEBUG cam  - Grab image resolution: 480x360
323508 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
323508 [1093982656] DEBUG cam  - Buffer size: 348160
323521 [1093982656] DEBUG cam  - camera_grab done in:  13 ms
323521 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
323600 [1093982656] DEBUG cam  - compress done
```

```
232584 [0x2f6041c0] DEBUG camcam null - SRGBCube::add: 254 236 191
232584 [0x2f6041c0] DEBUG camcam null - SRGBCube::add /k: 63 59 47
232584 [0x2f6041c0] DEBUG camcam null - SRGBCube::add /k)*k: 252 236 188
232584 [0x2f6041c0] DEBUG camcam null - SRGBCube::add (float) 252 236 188
232584 [0x2f6041c0] DEBUG camcam null - SRGBCube::add: 254 236 187
232584 [0x2f6041c0] DEBUG camcam null - SRGBCube::add /k: 63 59 46
232584 [0x2f6041c0] DEBUG camcam null - SRGBCube::add /k)*k: 252 236 184
232584 [0x2f6041c0] DEBUG camcam null - SRGBCube::add (float) 252 236 184
232587 [0x2f6041c0] DEBUG camcam null - Process process_image_circle
232589 [0x2f6041c0] DEBUG camcam null - process_image_split
Area ratio 0/1 is 1.671687
232590 [0x2f6041c0] DEBUG camcam null - Segment: 0 Position: 145 333 Size: 664 27 30 Mean: 725 631
```

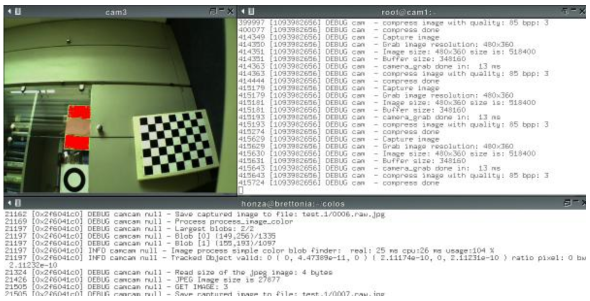
(c) detected blob after changing brightness and contrast of the camera



```
443894 [1093982656] DEBUG cam  - Grab image resolution: 480x360
443896 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
443896 [1093982656] DEBUG cam  - Buffer size: 348160
443909 [1093982656] DEBUG cam  - process_image_single
Area ratio 0/1 is 1.794712
443919 [1093982656] DEBUG cam  - Segment: 0 Position: 144 333 Size: 643 27 30 Me
an: 551 301
443919 [1093982656] DEBUG cam  - Segment: 1 Position: 144 333 Size: 643 27 30 Me
an: 551 301
443929 [1093982656] DEBUG cam  - CDetectorCircle: blobs: 1 v0: 1 v1: 82
443930 [1093982656] INFO can  - Image process circle real: 21 ms cpu:23 ms usage
:109.524 %
443930 [1093982656] INFO can  - Tracked Blob 1 (144, 333 ) / 643
443930 [1093982656] INFO can  - Tracked Object valid: 1 ( 2.39342, 0.447382, -0.
254037 ) ( 27.0071, 0, 0 ) ratio pixel: 0.989382 bw: 1.79471
443931 [1093982656] INFO can  - process_image  real: 22 ms cpu:23 ms usage:104.5
45 %
443931 [1093982656] INFO can  - Capture and image processing  real: 37 ms cpu:31
 ms usage:83.7838 %
443931 [1093982656] DEBUG cam  - Number segments:2
443931 [1093982656] DEBUG cam  - Segment[0] size: 1154
443931 [1093982656] DEBUG cam  - Segment[1] size: 643
443932 [1093982656] DEBUG cam  - Total bytes for the segments: 7200
```

```
21426 [0x2f6041c0] DEBUG camcam null - JPEG Image size is 27877
21505 [0x2f6041c0] DEBUG camcam null - GET IMAGE: 3
21505 [0x2f6041c0] DEBUG camcam null - Save captured image to file: test.1/0007.raw.jpg
21513 [0x2f6041c0] DEBUG camcam null - Process process_image_color
21540 [0x2f6041c0] DEBUG camcam null - Largest blobs: 2/2
21540 [0x2f6041c0] DEBUG camcam null - Blob [0] (149,255)/1351
21540 [0x2f6041c0] DEBUG camcam null - Blob [1] (155,193)/1098
21540 [0x2f6041c0] INFO camcam null - Image process simple color blob finder:  real: 25 ms cpu:26 ms usage:104 %
21540 [0x2f6041c0] INFO camcam null - Tracked Object valid: 0 ( 0, 4.47389e-11, 0 ) ( 2.11174e-10, 0, 2.11231e-10 ) ratio pixel: 0 bw
 2.11232e-10
46114 [0x2f6041c0] DEBUG camcam null - Set cube to the overo no. of values: 362 result: 1
47461 [0x2f6041c0] DEBUG camcam null - Number of blobs: 2
```

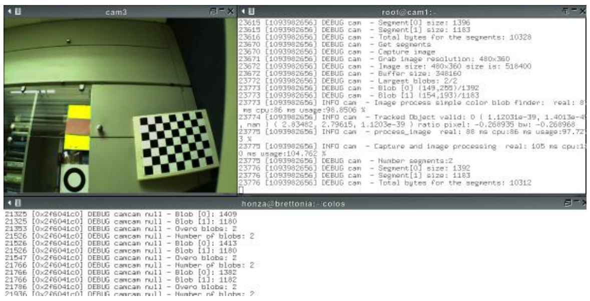
(d) detected blob received from the camera



```
399997 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
400077 [1093982656] DEBUG cam  - compress done
414349 [1093982656] DEBUG cam  - Capture image
414350 [1093982656] DEBUG cam  - Grab image resolution: 480x360
414351 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
414351 [1093982656] DEBUG cam  - Buffer size: 348160
414363 [1093982656] DEBUG cam  - camera_grab done in:  13 ms
414363 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
414444 [1093982656] DEBUG cam  - compress done
415179 [1093982656] DEBUG cam  - Capture image
415179 [1093982656] DEBUG cam  - Grab image resolution: 480x360
415181 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
415181 [1093982656] DEBUG cam  - Buffer size: 348160
415193 [1093982656] DEBUG cam  - camera_grab done in:  13 ms
415193 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
415274 [1093982656] DEBUG cam  - compress done
415629 [1093982656] DEBUG cam  - Capture image
415629 [1093982656] DEBUG cam  - Grab image resolution: 480x360
415630 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
415631 [1093982656] DEBUG cam  - Buffer size: 348160
415643 [1093982656] DEBUG cam  - camera_grab done in:  13 ms
415643 [1093982656] DEBUG cam  - compress image with quality: 85 bpp: 3
415724 [1093982656] DEBUG cam  - compress done
```

```
21162 [0x2f6041c0] DEBUG camcam null - Save captured image to file: test.1/0006.raw.jpg
21169 [0x2f6041c0] DEBUG camcam null - Process process_image_color
21197 [0x2f6041c0] DEBUG camcam null - Largest blobs: 2/2
21197 [0x2f6041c0] DEBUG camcam null - Blob [0] (149,256)/1335
21197 [0x2f6041c0] DEBUG camcam null - Blob [1] (155,193)/1097
21197 [0x2f6041c0] INFO camcam null - Image process simple color blob finder:  real: 25 ms cpu:26 ms usage:104 %
21197 [0x2f6041c0] INFO camcam null - Tracked Object valid: 0 ( 0, 4.47389e-11, 0 ) ( 2.11174e-10, 0, 2.11231e-10 ) ratio pixel: 0 bw
 2.11232e-10
21324 [0x2f6041c0] DEBUG camcam null - Read size of the jpeg image: 4 bytes
21426 [0x2f6041c0] DEBUG camcam null - JPEG Image size is 27877
21505 [0x2f6041c0] DEBUG camcam null - GET IMAGE: 3
21505 [0x2f6041c0] DEBUG camcam null - Save captured image to file: test.1/0007.raw.jpg
```

(e) selected pixels forming the RGB cube for the color based blobfinder



```
23615 [1093982656] DEBUG cam  - Segment[0] size: 1396
23615 [1093982656] DEBUG cam  - Segment[1] size: 1183
23616 [1093982656] DEBUG cam  - Total bytes for the segments: 10328
23670 [1093982656] DEBUG cam  - Get segments
23670 [1093982656] DEBUG cam  - Capture image
23671 [1093982656] DEBUG cam  - Grab image resolution: 480x360
23672 [1093982656] DEBUG cam  - Image size: 480x360 size is: 518400
23672 [1093982656] DEBUG cam  - Buffer size: 348160
23772 [1093982656] DEBUG cam  - Largest blobs: 2/2
23773 [1093982656] DEBUG cam  - Blob [0] (149,255)/1392
23773 [1093982656] DEBUG cam  - Blob [1] (154,193)/1183
23773 [1093982656] INFO can  - Image process simple color blob finder:  real: 8
 ms cpu:86 ms usage:98.8506 %
23774 [1093982656] INFO can  - Tracked Object valid: 0 ( 1.12031e-39, 1.4013e-4
, nan ) ( 2.83482, 2.79615, 1.1203e-39 ) ratio pixel: -0.268935 bw: -0.268968
23775 [1093982656] INFO can  - process_image  real: 88 ms cpu:86 ms usage:97.72
3 %
23775 [1093982656] INFO can  - Capture and image processing  real: 105 ms cpu:1
0 ms usage:104.762 %
23775 [1093982656] DEBUG cam  - Number segments:2
23776 [1093982656] DEBUG cam  - Segment[0] size: 1392
23776 [1093982656] DEBUG cam  - Segment[1] size: 1183
23776 [1093982656] DEBUG cam  - Total bytes for the segments: 10312
```

```
21325 [0x2f6041c0] DEBUG camcam null - Blob [0]: 1409
21325 [0x2f6041c0] DEBUG camcam null - Blob [1]: 1180
21353 [0x2f6041c0] DEBUG camcam null - Overo blobs: 2
21526 [0x2f6041c0] DEBUG camcam null - Number of blobs: 2
21526 [0x2f6041c0] DEBUG camcam null - Blob [0]: 1413
21526 [0x2f6041c0] DEBUG camcam null - Blob [1]: 1180
21547 [0x2f6041c0] DEBUG camcam null - Overo blobs: 2
21766 [0x2f6041c0] DEBUG camcam null - Number of blobs: 2
21766 [0x2f6041c0] DEBUG camcam null - Blob [0]: 1382
21766 [0x2f6041c0] DEBUG camcam null - Blob [1]: 1182
21786 [0x2f6041c0] DEBUG camcam null - Overo blobs: 2
21936 [0x2f6041c0] DEBUG camcam null - Number of blobs: 2
```

(f) detected color blob received from the camera

Figure 9: Examples of `tcam` usage.

5.5 Creating Camera Control Settings File

The interactive mode of the `tcam` application can be used to create a file with camera control settings for the capturing or tracking usage of the camera module. The adjustment of the control settings have to be done according to the current light condition, therefore, on-line visualization of the captured images together with a detected blob is handy.

First, the `tcapture` server has to be started. This can be done by the following sequence of commands, as the needed files are already prepared on the camera module.

1. `ssh cam1`
2. `./tcapture`

Similarly the `tcam` application can be started with the provided configuration file. Then, the following sequence of control commands (keys) can be performed to create a new `cam.cfg` file.

1. `'c'` - connect to the server.
2. `'o'` - open the camera.
3. `'f'` - receive images from the camera to check if the camera is working properly and to see what images are provided by the current settings.
4. adjust the camera control settings using the keys described above.
5. Press `'-'` to save the current camera settings to the file, i.e., into `cam.cfg` file at the server (using the default configuration).
6. Quit the server.
7. Place the created file into appropriate directory at the camera module within the remote shell session, e.g., `cp cam.cfg /opt/cam.cfg`.

5.6 Taking Images for the Camera Calibration

The `tcam` application can also be used for creating a set of images needed to obtain intrinsic and extrinsic camera parameters by the following procedure.

1. Start the `tcapture` server at the camera using a remote `ssh` session.
2. Start the `tcam` application and connect to the camera by pressing `'c'` and `'o'` keys.
3. Adjust the camera control settings if necessary.
4. Place the testing pattern at the appropriate position, check the position using the jpeg image (by pressing `'f'` key. .
5. Save the image in the bmp format by pressing `'g'` key (this is usually slower due to a larger amount of data than for the jpeg images transmitted in the previous step. .
6. If all desired calibration images are retrieved, exit the application and close the remote session. Otherwise repeat the steps 4 and 5 as needed.

6 Future Work

Although the camera module for the COLOS project has been significantly improved in its second revision, there is still potential for additional improvements. They are summarized in the following list.

- Stability of the mt9v032 driver in BA10 mode should be improved, which can allows consideration of 10-bit values of the captured image.
- The conversion from YUV to RGB or grayscale, alternatively conversion from BA10 to RGB or grayscale can be done using the image signal processor, which can reduced the required computational time a bit.
- Automatic exposure settings according to local neighborhood of the detected blob.
- Simultaneous tracking of more B/W patterns.
- Fine tuned model of the radial distortion.
- Distinguishing of more ring patterns using color.
- A client to the tracker server as a ROS node, which will allow a simple distribution of information about a tracked object to other parts of the complex robotic system.
- The tracker server itself can be implemented as a ROS node; however, it requires ROS libraries for the Overo OMAP3503 processor.

A Specifications

Table 6: Hardware parameters

Hardware Specifications	
CPU:	OMAP3503 @ 600 MHz running armv7l GNU/Linux 2.6.34
CMOS sensor:	Aptina MT9V032, 752×480 @ 60 Hz
Lens:	F=2.8, FoV$_{min}$ 42°, IR cut filter
Overo Board:	58 mm × 17 mm × 5 mm, weight 6 g (including μSD card)
Camera Board:	39 mm × 26 mm × 25 mm, weight 22.9 g (with lens)
Voltage Regulator Board:	68 mm × 23 mm × 18 mm, weight 18 g (with cables)
Minicom Board:	60 mm × 19 mm × 7 mm, weight 6 g
Voltage Regulator Effic.:	89 % (cpu idle), 84 % (cpu full load)
Power Consumption:	$\leq$ 2.6 W (2.39 W cpu idle, 2.55 W tracking at full speed)

Table 7: Access parameters

Access Specifications	
Login:	`root` (*password-less*)
Serial Console:	115200 bps, 8 data bits, 1 stop bit, none parity
WiFi:	802.11b/g
WiFi SSID:	COLOS
WiFi mode:	adhoc, open without encryption
WiFi IP address:	10.10.40.174
Remote shell:	ssh

Table 8: Tracker parameters

Tracker Specifications	
B/W Ring Pattern:	Outer diameter 14 cm, inner diameter 8.4 cm (width 2.8 cm)
Image Resolution:	480× 360 (grayscale)
Field of View:	33° × 67° (*considering blob dimensions and radial distortion*)
Images per Second:	Max 52 FPS, Min: 18 FPS
Pattern Distance Range:	0.2 - 3.5 m
Expected Distance Error:	units of cm
Average Distance Error:	$\leq$ 4.1%
Average Repeatability:	$\leq$ 0.5 cm
Tracker Server Address:	10.10.40.174:9001 (UDP/IP)
Tracker FPS restriction:	none (*process as fast as possible*)

B Camera Module Hardware

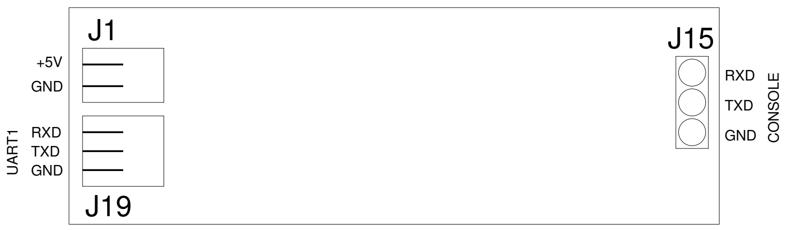


Figure 10: Connectors of the Overo minicom board.

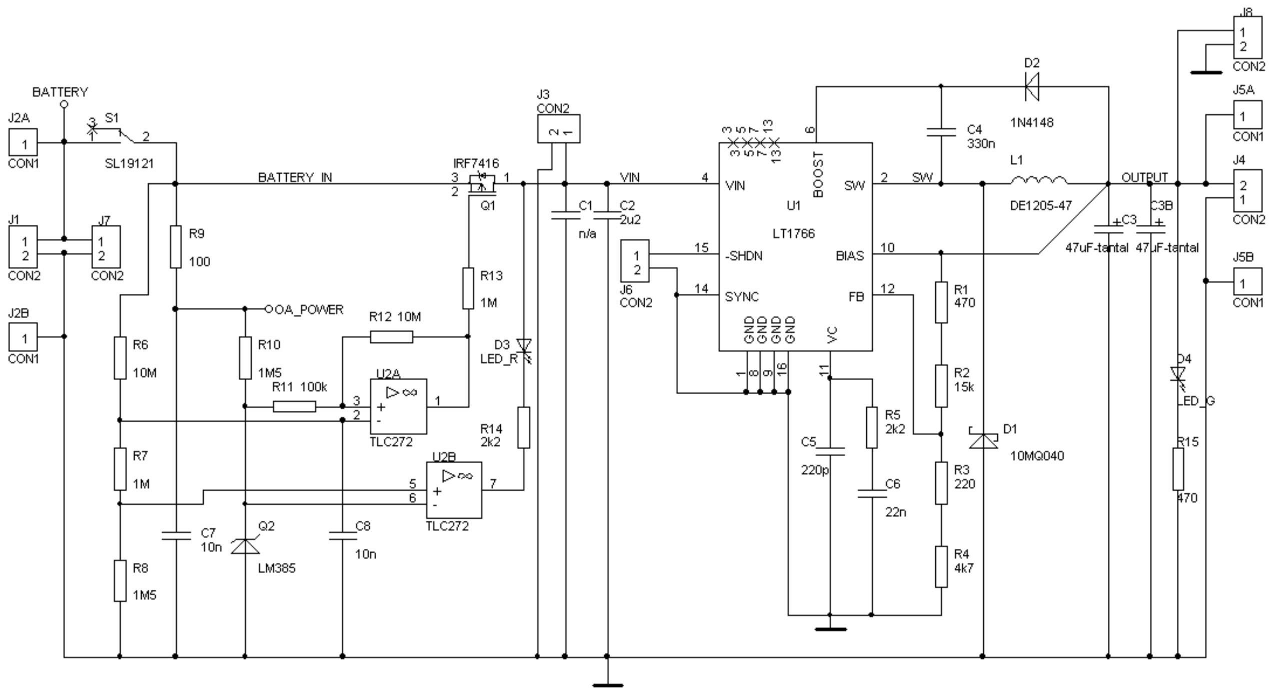
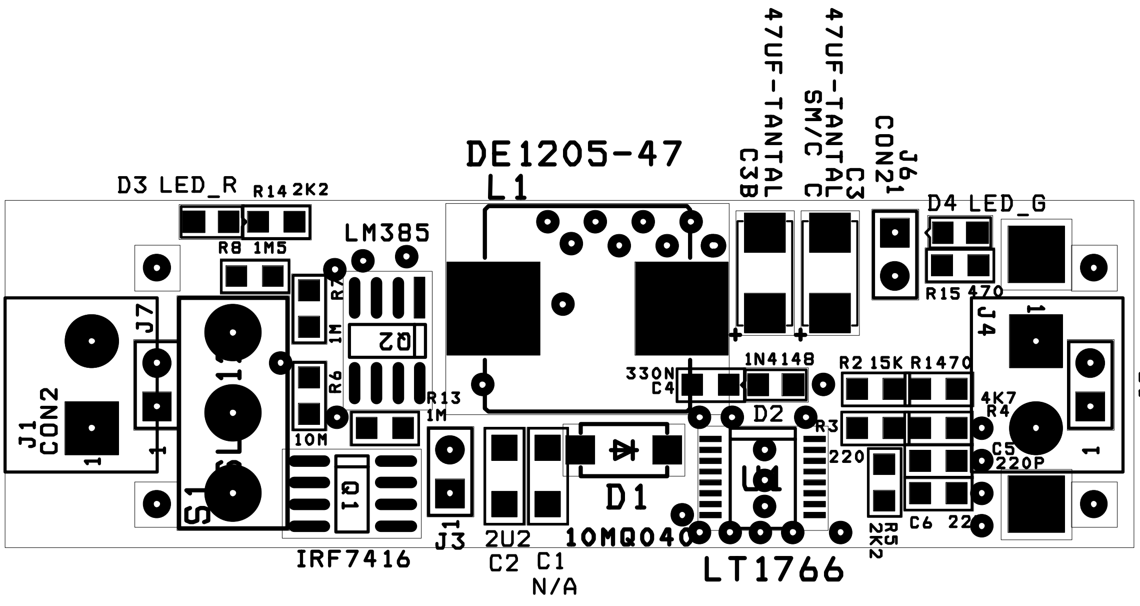
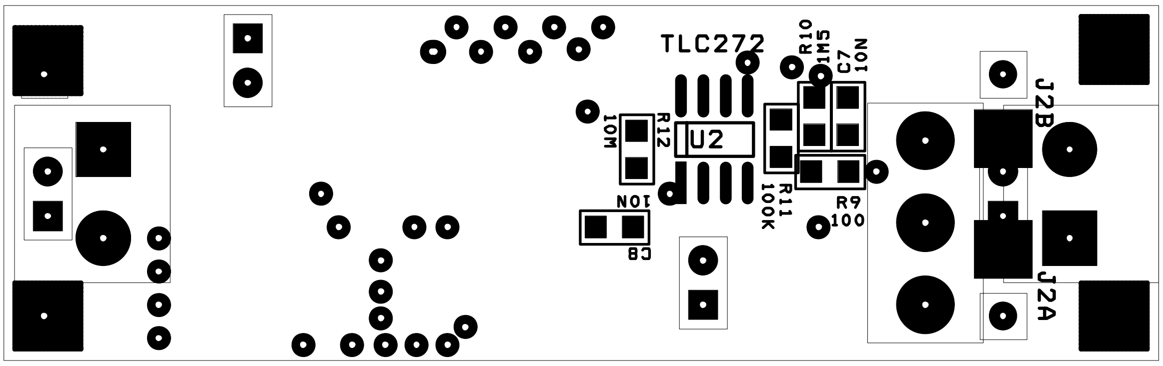


Figure 11: Schematic of the voltage regulator and battery undervoltage switch.



(a) top



(b) bottom

Figure 12: Circuits of the voltage regulator and battery undervoltage switch.

C Applications

```
tcam testing application 0.5

General options:
  -h [ --help ]                     produce help message
  -c [ --config ] arg (=./tcam.cfg) configuration file
  -l [ --logger-config ] arg        logger configuration file

CamApp options:
  --width arg (=480)                      default width
  --height arg (=360)                     default height
  --cam-server arg (=localhost)           hostname of the cam server
  --cam-port arg (=9009)                  port of the cam server
  --image arg                             default image loaded
  --use-image-size arg (=0)               adjust windows size from the loaded
                                          image
  --image-directory arg (=jpg)            image directory to store files capture
                                          from the cam
  --image-pattern arg (=%04d.jpg)         image pattern for captured images
  --out-image-directory arg               output image directory to store
                                          processed files
  --out-jpeg arg (=1)                     Enable/disable jpeg output format (bmp
                                          is used otherwise
  --cube-rgb arg                          file to load/store the rgb cube
  --points-rgb arg                        list of rgb values (r g b - space
                                          separated) used for adding to the cube
                                          using the hsv ranges
  --benchmark arg (=0)                    Enable/disable benchmark mode
  --benchmark-draw-image arg (=0)         Enable/disable drawing result image in
                                          benchmark mode
  --benchmark-result-log arg              Filename to store benchmark results log
  --benchmark-dataset arg                 Name of the dataset used in the
                                          benchmark results log
  --benchmark-name arg                    An additional name used in the
                                          benchmark results log
  --benchmark-max-images arg (=-1)        If >= 0 only given number of images is
                                          processed
  --camera-control-settings arg           File name to store/save camera settings
  --overo-bench-trials arg (=100)         Number of processed images on over for
                                          estimation of the fps
  --overo-process-trials arg (=100)       Number of processed images on over for
                                          estimation of the fps
  --jpeg-quality arg (=85)                JPEG compress quality
  --grayscale arg (=0)                    Enable/disable gray scale
  --use-circle-detector arg (=1)          Enable/disable circle detector, if not
                                          set, simple clolor blobfinder is used
  --process-on-refresh arg (=1)           Initial value of the process on refresh
                                          variable
  --tracked-object-diameter arg (=0.140000001)
                                          Diameter of the tracked object used in
                                          transformation
  --full-unbarrel arg (=0)                Enable/disable full unbarrel in the
                                          transformation of the coords
  --camera-parameters arg               File with the camera parameters in
                                        format required by the
                                        CCameraTransformation
  --max-segments arg (=10000)           Maximal number of the recognized
                                        segments (blobs)
  --number-blobs arg (=1)               No. of detected blobs
  --rgb-cube-size arg (=8)              default size of the rgb cube!
  --hsv-delta-h arg (=5)                range of h value of the hsv model for
                                        adding color ranges
  --hsv-delta-s arg (=3)                range of s value of the hsv model for
                                        adding color ranges
  --hsv-delta-v arg (=3)                range of v value of the hsv model for
                                        adding color ranges
  --hsv-neigh-delta-h arg (=5)          range of h value of the hsv model for
                                        adding color ranges
  --hsv-neigh-delta-s arg (=3)          range of s value of the hsv model for
                                        adding color ranges
  --hsv-neigh-delta-v arg (=3)          range of v value of the hsv model for
                                        adding color ranges
  --circle-detector-tracking arg (=1)   Enable/disable tracking, i.e., usage of
                                        the previous position
  --detector-implementation arg         For debuging purposes
  --circle-detector-max-segments arg (=1000)
                                        Maximal number of segments
  --circle-detector-threshold-init arg (=203)
                                        Initial value of the threshold
  --circle-detector-threshold-max arg (=768)
                                        Max value of the threshold
  --circle-detector-threshold-step arg (=60)
                                        Step value of the threshold
  --circle-detector-max-failed arg (=10)
                                        Max number of detection failes before
                                        threshold update
  --circle-detector-min-size arg (=20)  Min size of the blob
  --circle-detector-center-distance-tolerance arg (=15)
                                        Value of the center distance tolerance
  --circle-detector-circular-tolerance arg (=0.300000012)
                                        Valued of the circular tolerance
  --circle-detector-ratio-tolerance arg (=0.5)
                                        Valued of the ratio tolerance
  --circle-detector-areas-ratio arg (=0.5625)
                                        Discs area ratios
  --circle-detector-outer-area-ratio arg (=0.50265485)
                                        Outer disc area ratio
  --circle-detector-inner-area-ratio arg (=0.785398185)
                                        Inner disc area ratio
```

Listing 27: All program options of the `tcam` application.

References

[1] Camera calibration toolbox for matlab. `http://robots.stanford.edu/cs223b04/JeanYvesCalib/index.html`, [Cit.: 2012-01-17].

[2] Gumstix. `http://www.gumstix.com`, [Cit: 2012-01-16].

[3] J. Heikkila and O. Silven. A four-step camera calibration procedure with implicit image correction. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1106 –1112, jun 1997.

[4] Jan Faigl, Martin Saska, Matt Turpin, Libor Přeučil, and Vijay Kumar. Project colos - technical report 1.1 - camera module for onboard relative localization in uav swarms. Technical report, Czech Technical University in Prague, University of Pennsylvania, October 2011. Tech notes from work performend at UPENN.

[5] Openembedded. `http://www.openembedded.org/wiki/Main_Page`, [Cit.: 2012-01-17].

[6] jerm. `http://www.bsddiary.net/jerm`, [Cit.: 2012-01-17].

[7] Minicom. `http://alioth.debian.org/projects/minicom`, [Cit.: 2012-01-17].

[8] Boost.program_options. `http://www.boost.org/doc/libs/1_48_0/doc/html/program_options.html`, [Cit.: 2012-01-18].

[9] log4cxx. `http://logging.apache.org/log4cxx`, [Cit.: 2012-01-18].

[10] V4l2. `http://linuxtv.org/wiki/index.php/Main_Page`, [Cit.: 2012-01-18].

[11] Caspa camera boards. `http://wiki.gumstix.org/index.php?title=Caspa_camera_boards`, [Cit: 2012-01-19].

[12] M. Eisler. XDR: External Data Representation Standard. RFC 4506 (Standard), May 2006.