Kawasaki Robotics

Astorino

Vision System Manual

Preface

This manual describes the handling of the 6-axis robot “astorino" Vision System option.

The ASTORINO is a learning robot specially developed for educational institutions. Pupils and students can use the ASTORINO to learn robot-assisted automation of industrial processes in practice.

1. The "astorino" software included with the ASTORINO is licensed for use with this robot only and may not be used, copied or distributed in any other environment.
2. Kawasaki shall not be liable for any accidents, damages, and/or problems caused by improper use of the ASTORINO robot.
3. Kawasaki reserves the right to change, revise, or update this manual without prior notice.
4. This manual may not be reprinted or copied in whole or in part without prior written permission from Kawasaki.
5. Keep this manual in a safe place and within easy reach so that it can be used at any time. If the manual is lost or seriously damaged, contact Kawasaki.

Symbols

Items that require special attention in this manual are marked with the following symbols.

Ensure proper operation of the robot and prevent injury or property damage by following the safety instructions in the boxes with these symbols.

⚠ WARNING

Failure to observe the specified contents could possibly result in injury or, in the worst case, death.

[**ATTENTION**]

Identifies precautions regarding robot specifications, handling, teaching, operation, and maintenance.

⚠ WARNING

1. **The accuracy and effectiveness of the diagrams, procedures and explanations in this manual cannot be confirmed with absolute certainty. Should any unexplained problems occur, contact Kawasaki Robotics GmbH at the above address.**
2. **To ensure that all work is performed safely, read and understand this manual. In addition, refer to all applicable laws, regulations, and related materials, as well as the safety statements described in each chapter. Prepare appropriate safety measures and procedures for actual work.**

Paraphrases

The following formatting rules are used in this manual:

- For a particular keystroke, the respective key is enclosed in angle brackets, e.g. <F1> or <Enter>.
- For the button of a dialog box or the toolbar, the button name is enclosed in square brackets, e.g. [Ok] or [Reset].
- Selectable fields are marked with a square box ☐.
  If selected a check mark is shown inside the symbol ☑.

List of contents

1 Nomenclature in this manual

The author of the manual tries to use generally valid terminology while achieving the greatest possible logical sense. Unfortunately, it must be noted that the terminology is reversed depending on the point of view when considering one and the same topic. Also it is to be stated that in the course of the computer and software history terminologies developed in different way. One will find therefore in a modern manual no terminologies, which always satisfy 100% each expert opinion.

2 Overview of ASTORINO

The ASTORINO is a 6-axis learning robot developed specifically for educational institutions such as schools and universities. The robot design is based to be 3D printed with PET-G filament. Damaged parts can be reproduced by the user using a compatible 3D printer.

Programming and control of the robot is done by the "astorino" software.

The latest software version and 3D files can be downloaded from the KAWASAKI ROBOTICS FTP server:

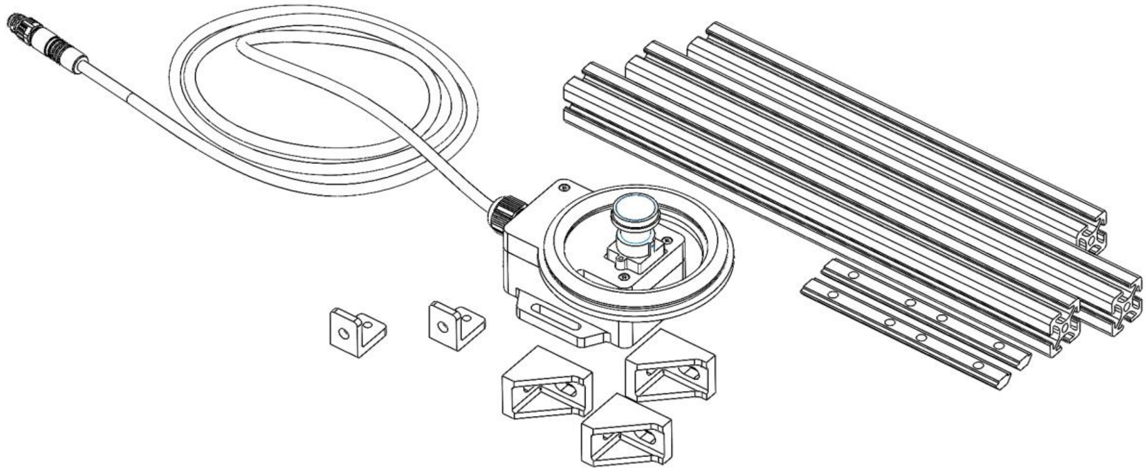
https://ftp.kawasakirobot.de/Software/Astorino/

Just like Kawasaki’s industrial Robots the ASTORINO is programmed using AS language. Providing transferable programming skills from the classroom to real industrial applications.

3 Technical specifications

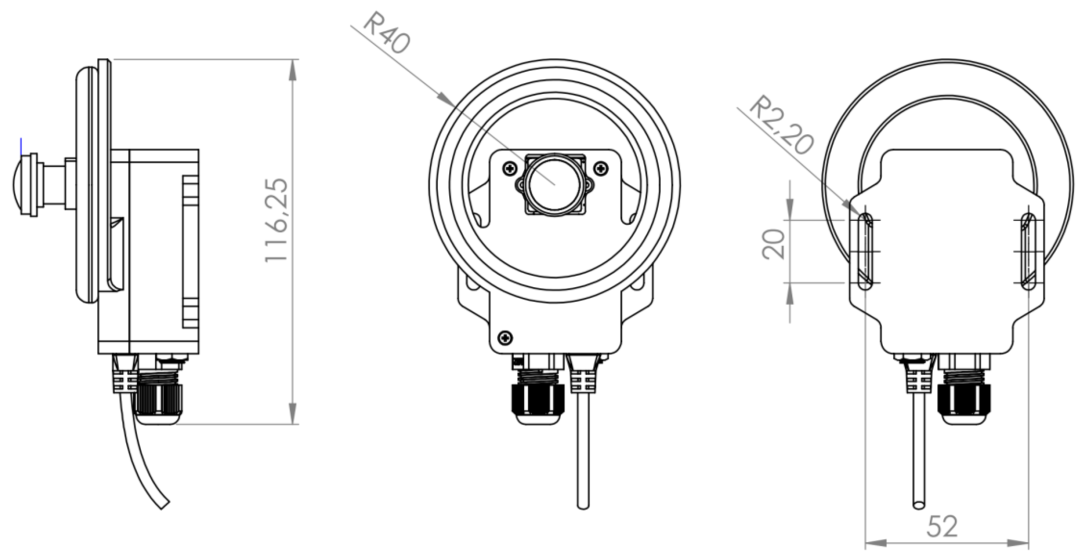
Characteristics		Astorino Conveyor
Working environment	Temperature	0–40°C
	Humidity	35–80%
Processor		ARM® 32-bit Cortex®-M7 CPU
RAM		33MB
Supported Image Formats		Grayscale RGB565 JPEG
Max. Supported Resolutions		2952x1944 25-50 FPS on QVGA (320x240)
Lens Info		Focal Length: 2.8mm Aperture: F2.0 Format: 1/3" HFOV = 70.8°, VFOV = 55.6° Mount: M12*0.5 IR Cut Filter: 650nm (removable)
Power supply		5V – Camera 12V - Light
Power Consumption		Idle: 140mA @ 3.3V Active: 230mA @ 3.3V Light – 300 mA @ 12V
Programming language		MicroPython
Programming interface		MicroUSB
Programming software		OpenMV IDE
Material		PET-G
Colour		Black
Communication		UART (Serial)
Weight		50g
Possible applications		TensorFlow lite for microcontrollers support Frame differencing Colour tracking Marker tracking Face detection Eye-tracking Person detection Optical flow QR code detection/Decoding Data matrix detection/Decoding Linear barcode decoding April Tag tracking Line detection Circle detection Rectangle detection

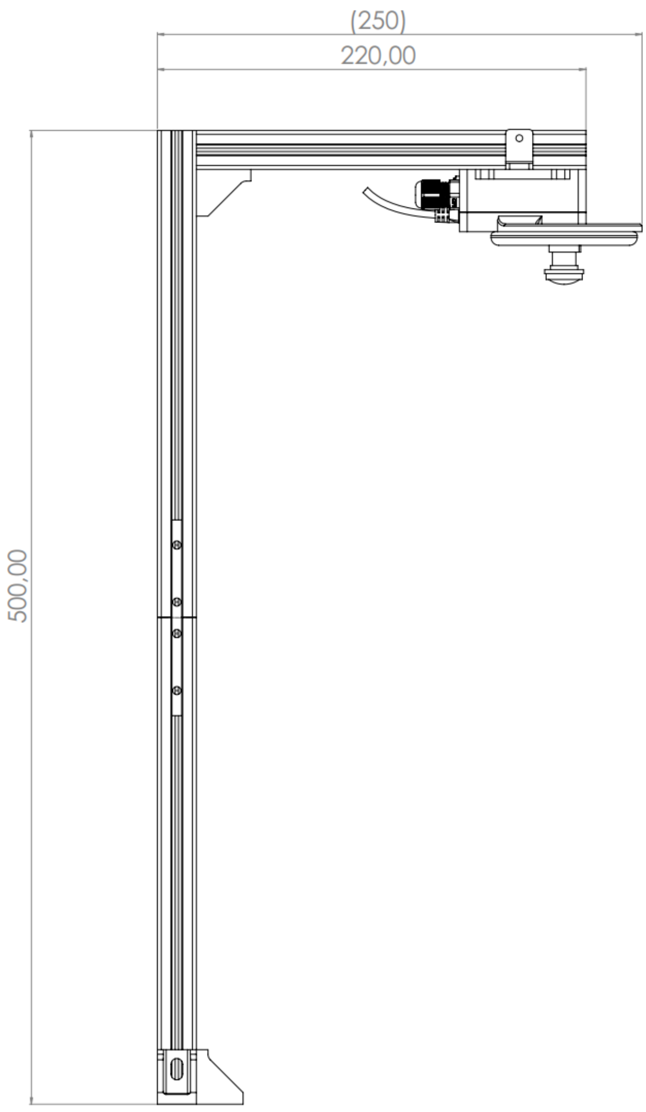
4 Conveyor package contents



Item	quantity	Name
1	3	2020
2	2	Camera mount
3	3	Angle brackets
4	2	12 V Power Supply
5	1	Micro USB cable

5 Dimensions





6 openMV - overview

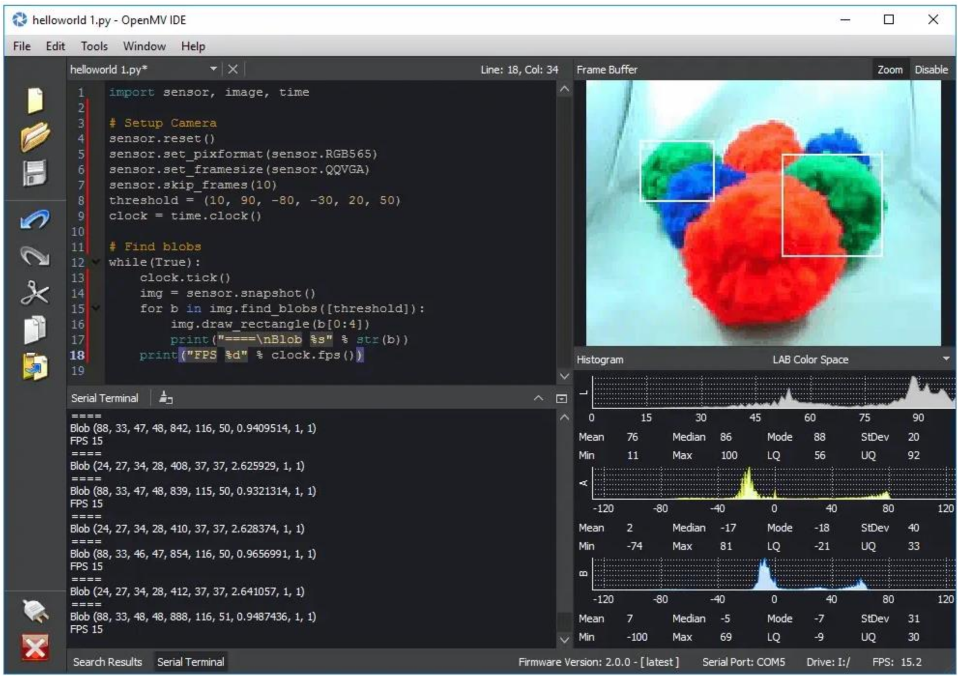
The OpenMV Cam is a small, low power, microcontroller board which allows you to easily implement applications using machine vision in the real-world. You program the OpenMV Cam in high level Python scripts (courtesy of the MicroPython Operating System) instead of C/C++. This makes it easier to deal with the complex outputs of machine vision algorithms and working with high level data structures. But, you still have total control over your OpenMV Cam. For more info, openMV IDE operation manual and MicroPython please see the on-line documentation.

https://docs.openmv.io/openmvcam/tutorial/index.html

7 Installation

To use openMV camera please download and install newest OpenMV IDE from this webside:

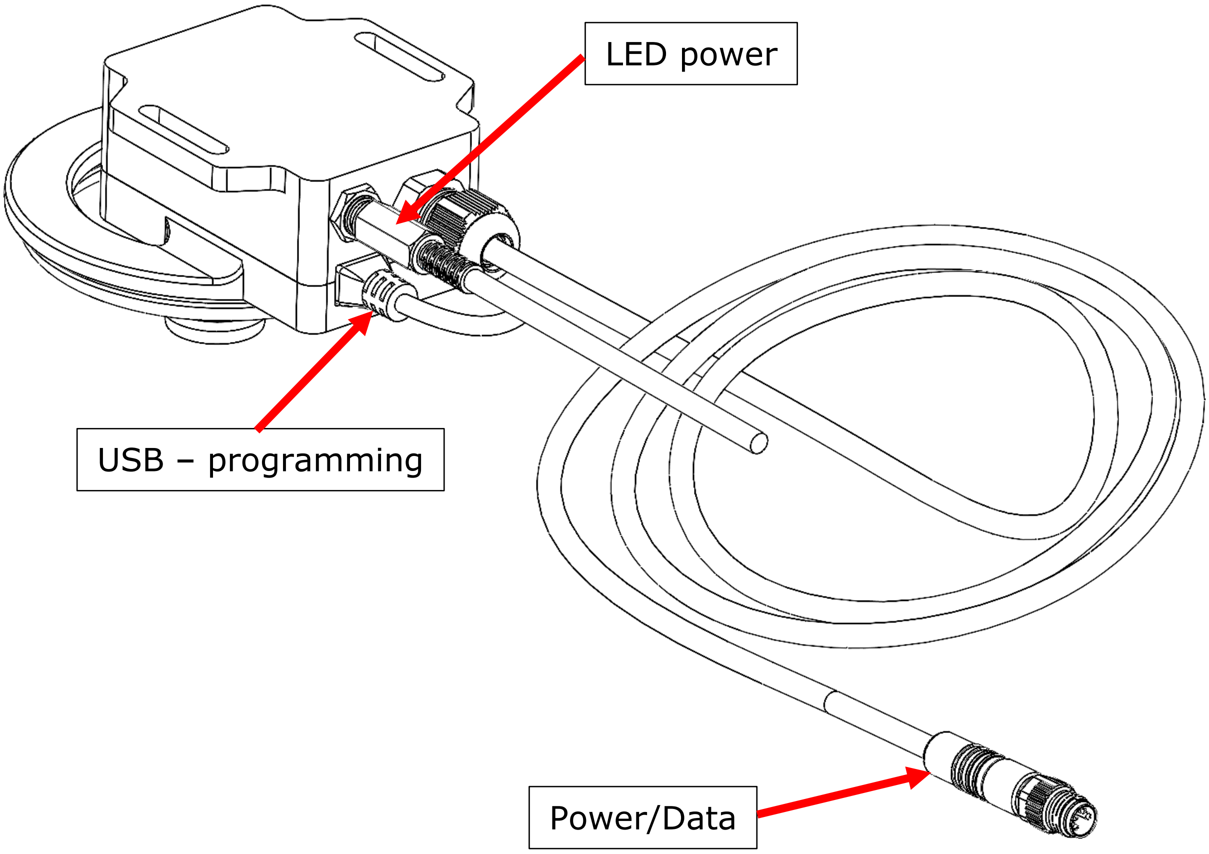
https://openmv.io/pages/download



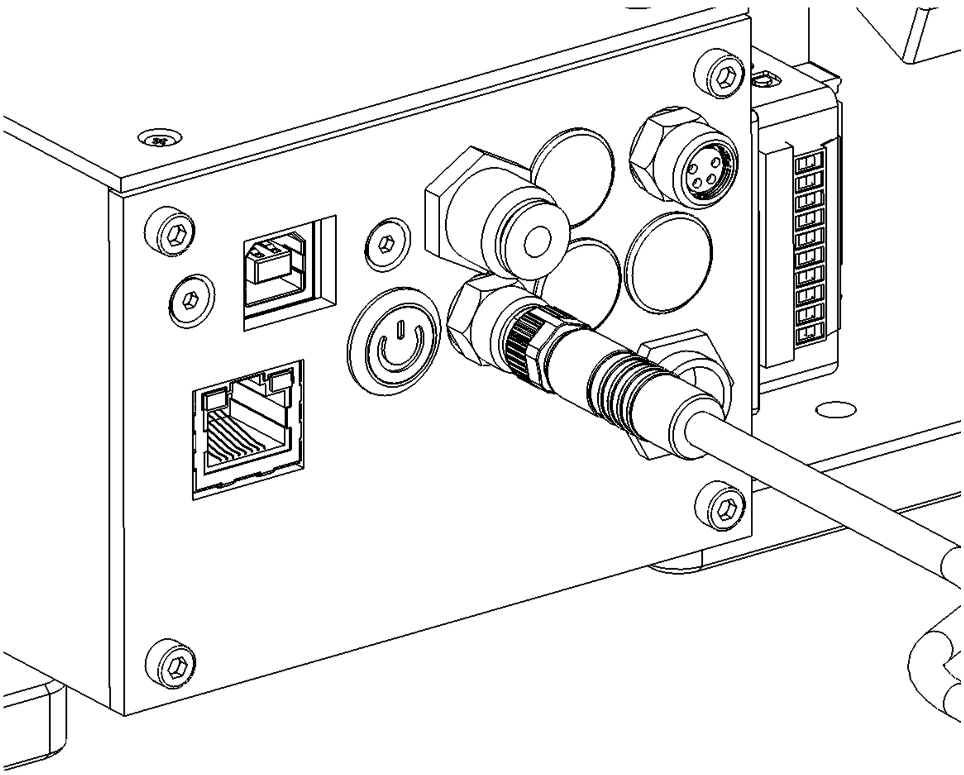
Please also see Quickref to get familiar with the system

https://docs.openmv.io/openmvcam/quickref.html

7.1 Camera connection



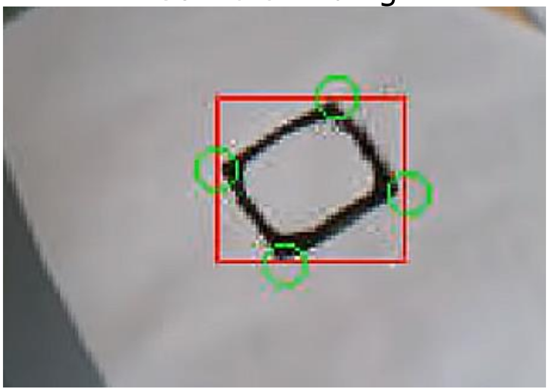
Connect LED power cable, micro USB for programming and Power/Data cable to astorino Serial port.



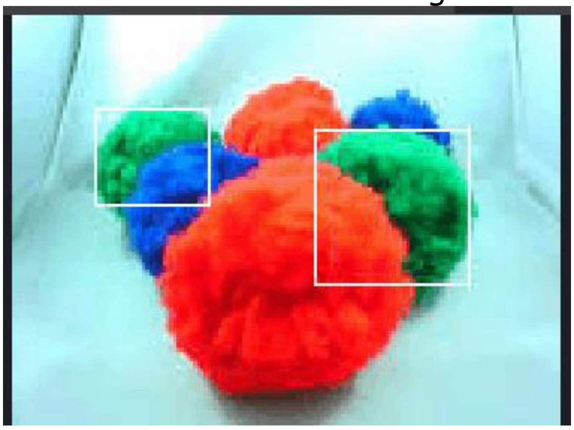
8 Vision system – general information

OpenMV camera can find on the pictures different shapes or colour blobs.

Corners finding

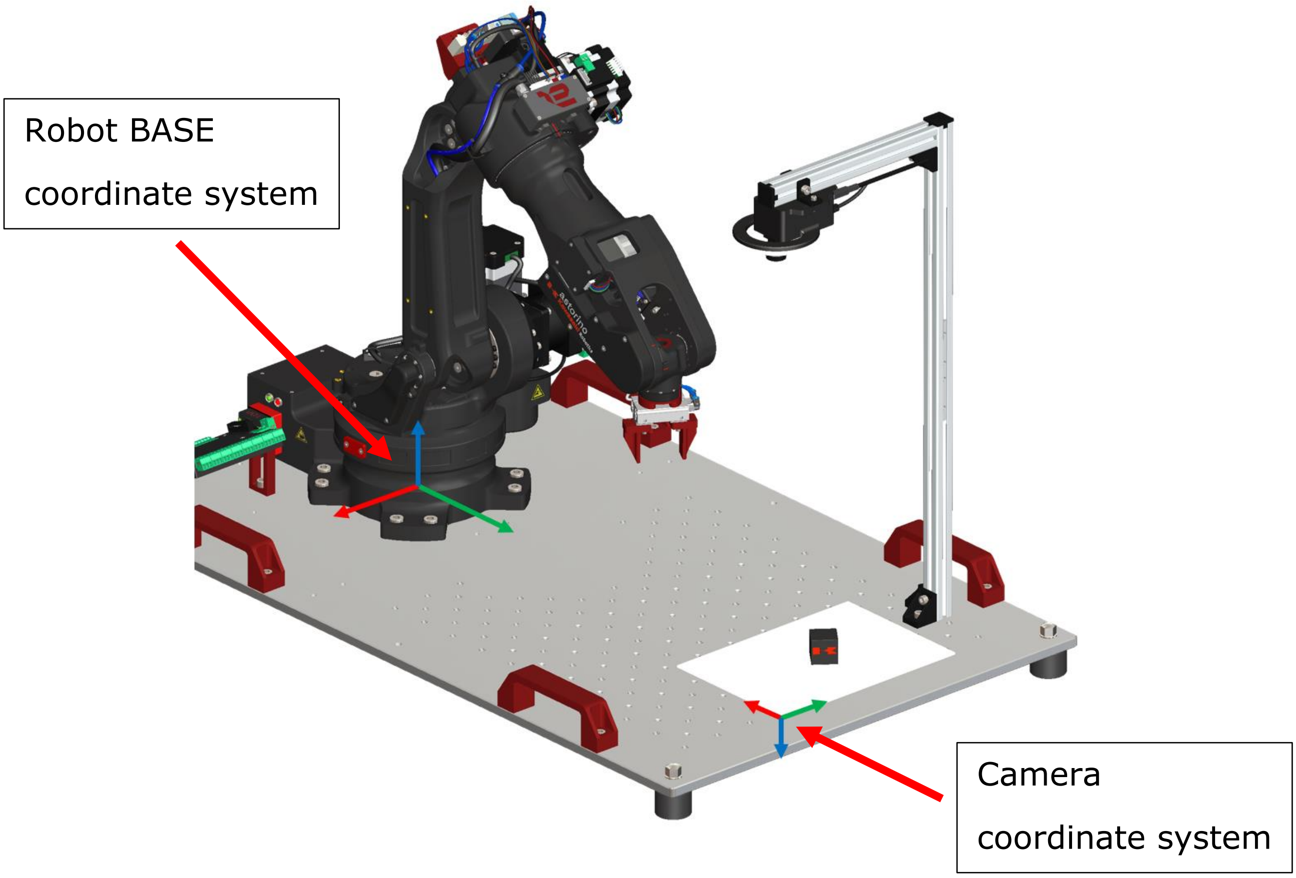


Colour blobs finding

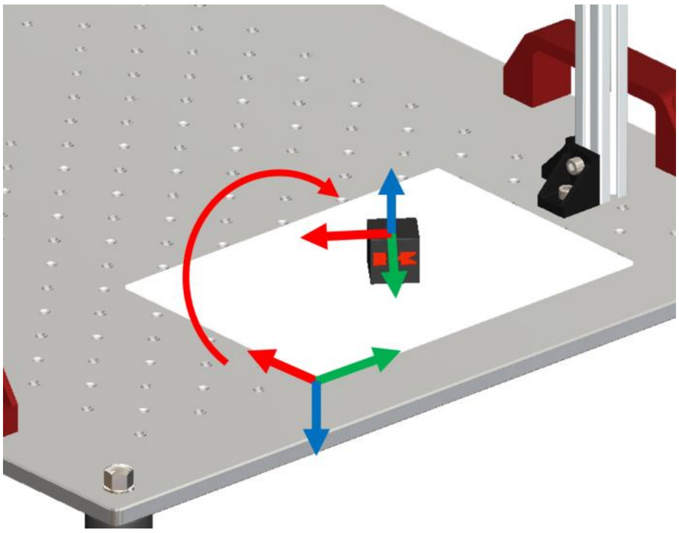


There are many example project to help start with this system with different detection features.

Position of this objects is returned in camera coordinate system and in pixels.



Because the coordinate systems and scales are different robot cannot operate on camera raw data, we need to transform the object position from pixels to mm relative to the robot base.



For that we need to define a rotation matrix [4x4] and scale factor which will transform object position to BASE coordinate system and also scale pixels to millimetres.

$$H = \begin{bmatrix} R1 & R4 & R7 & Tx \\ R2 & R5 & R8 & Ty \\ R3 & R6 & R9 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where R1..R9 are describing the change in rotation and Tx..Tz are describing the change in position.

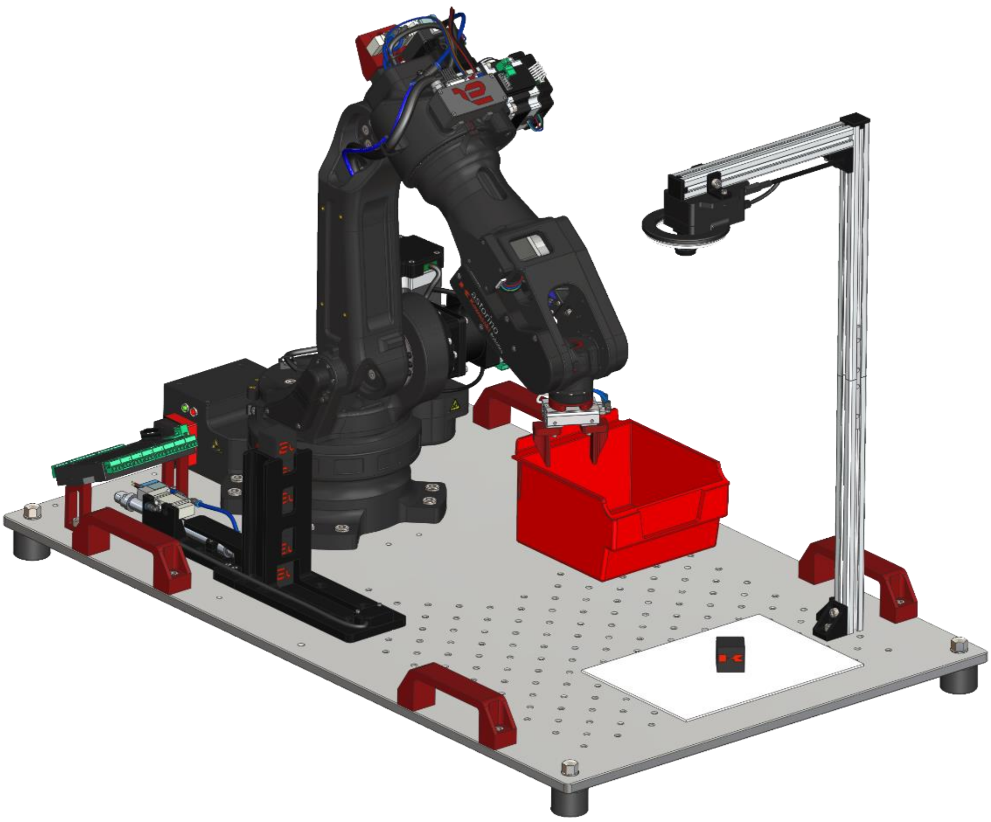
Example app already is using this method.

9 astorino – openMV example app

Astorino example Vision system app is a pick and place demo.

The flow of this demo is like this:

- Robot picks a cube from a cube feeder,
- Places it under the camera,
- Camera takes the picture and finds the cubes,
- Cube position is transmitted to the robot
- Robot picks the cube and places it in the red bucket.



9.1 Camera programs

There are two vision system programs:

- Calibration – used to set a calibration data (H matrix and scale)
- Finding objects – used with robot program to pick cubes)

9.2 Robot program

Ther is one example robot program which reads data from a camera, creates a destination point and goes to taht point to pick the cube.

10 Calibration instructions

The openMV camera detects objects in its field of view and converts the coordinates of their center, in pixels, to coordinates in the robot's working field (in millimeters).

10.1 Program modifications

10.1.1 Code configuration

Depending on the configuration of the camera settings, the exposure of the object and external factors, modify the following part of the code:

```
#kalibracja

w_x = 190
w_y = 124
w_width = 342
w_hight = 208

fisheye_corr = 0.5

c_frm_px = [[74,5],[77,172],[302, 167],[297,25]]
r_frm_p = [[-1.8, 260.0],[132.4, 258.2],[128.7, 441.8],[-4.9, 440.8]]

thresholds = (0, 40)

off_x = 0
off_y = 0
off_ang = -0.0208
```

For easier selection of the following parameters, use the calibration program, which continuously displays the coordinates of detected objects and allows viewing the camera image.

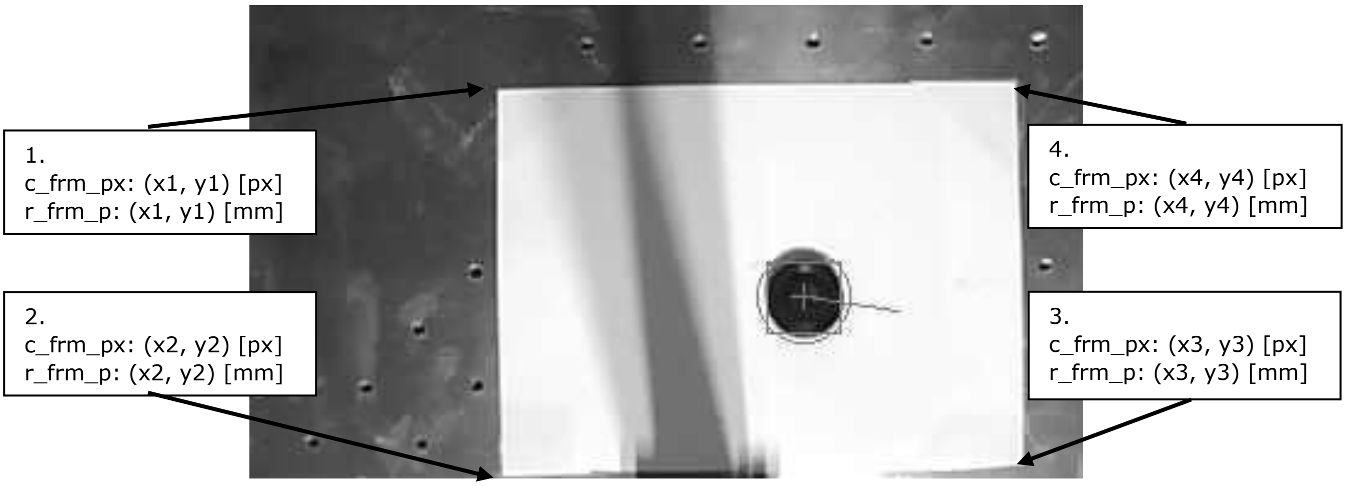
- *w_x* – x-coordinate (in pixels) specifying the upper left corner of the selected image analysis area from the entire camera view.
- *w_y* – y-coordinate (in pixels) specifying the upper left corner of the selected image analysis area from the entire camera view.
- *w_width* – the width of the analyzed area (in pixels) starting from *w_x*.
- *w_hight* – the height of the analyzed area (in pixels) starting from *w_y*.
- *fisheye_corr* – The fisheye correction of the camera should be selected according to the height of its mounting.
- *c_frm_px* – from the camera view, read the coordinates of the boundary points of the work area in pixels (for example, by copying the camera view into a graphics program or by clicking on the corresponding pixels in the view in the OpenMV application). Enter the coordinates in the following order: start with the leftmost point (value

in pixels) in the camera view, enter subsequent points in counter-clockwise order.

Coordinates of the selected pixel displayed under the camera view

- *r_frm_p* – reaching with the robot to the successive boundary points (the same as in the above point) of the working area, detected by the camera, enter the *x* and *y* coordinates read from the astorino application (JOG tab: X, Y). Enter the coordinates in the following order: start from the leftmost point in the camera view, enter the subsequent points in counterclockwise order (the same as in the above point).

It is important that the *c_frm_px* and *r_frm_p* coordinates are entered in the same order!

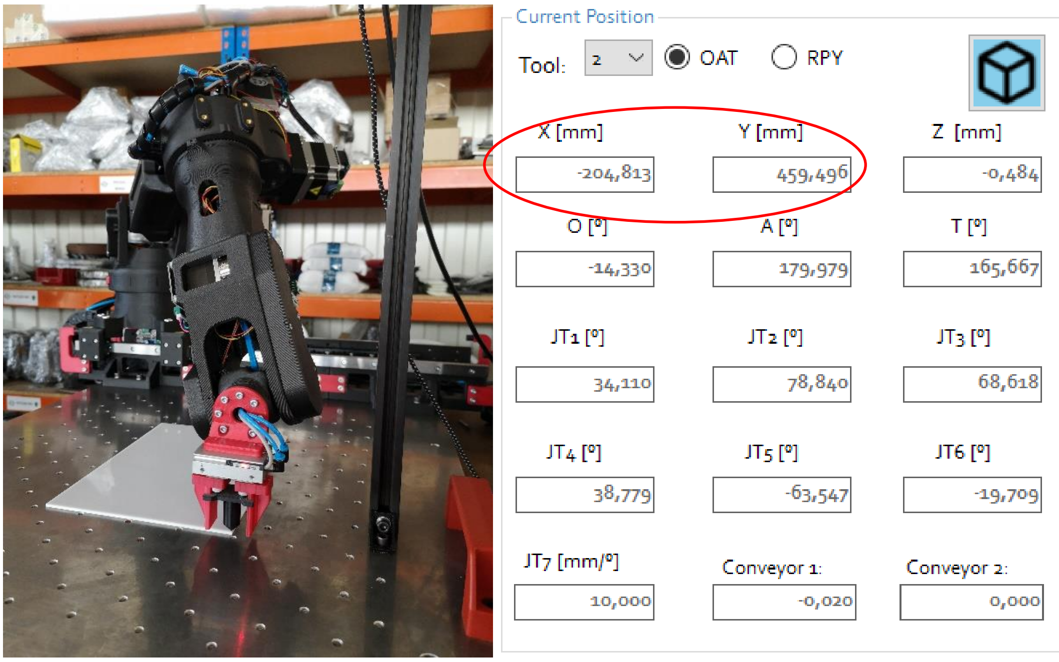


10.2 Calibration

Please run calibration program on OpenMV IDE.

10.2.1 Point 1

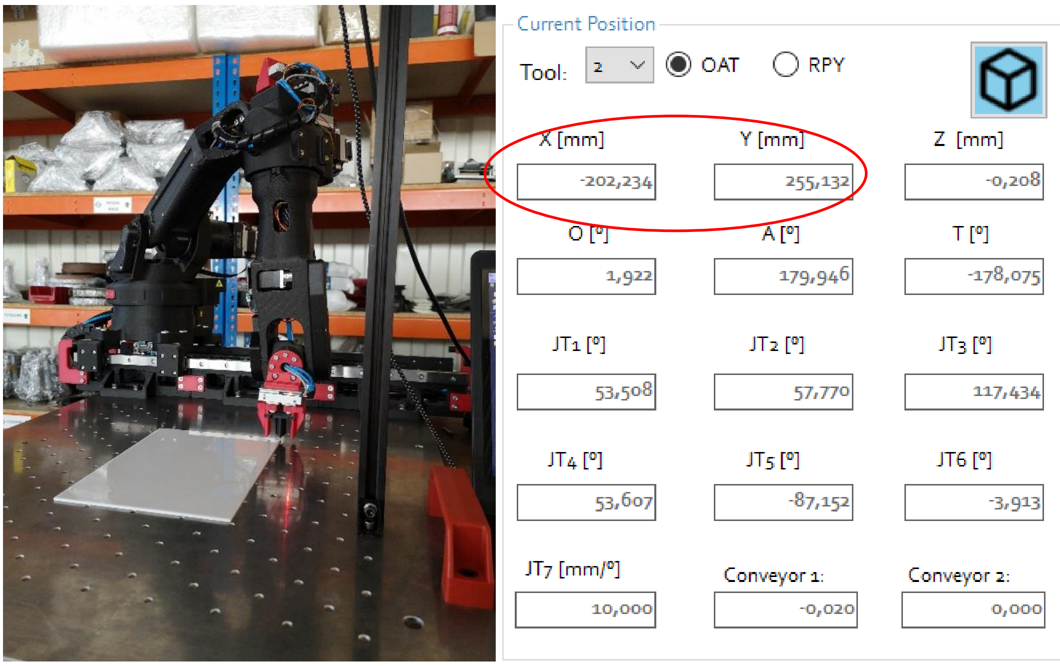
Using mouse cursor read the pixel position of the corner



Move robot to the corner and save the position.

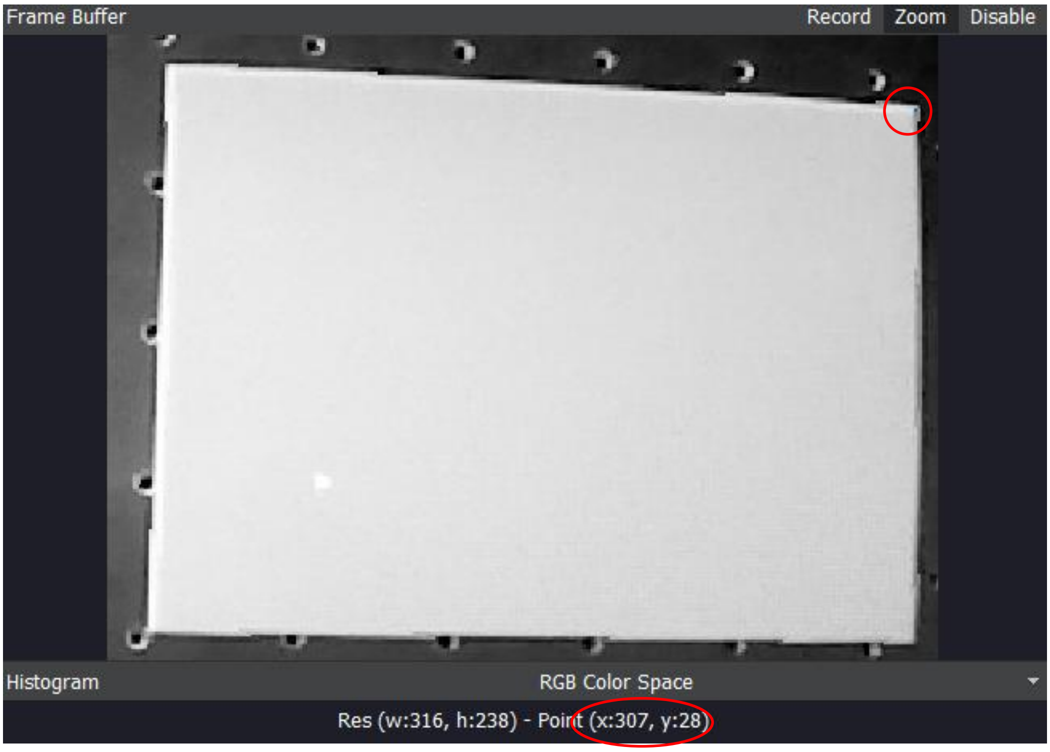
10.2.2 Point 2

Using mouse cursor read the pixel position of the corner

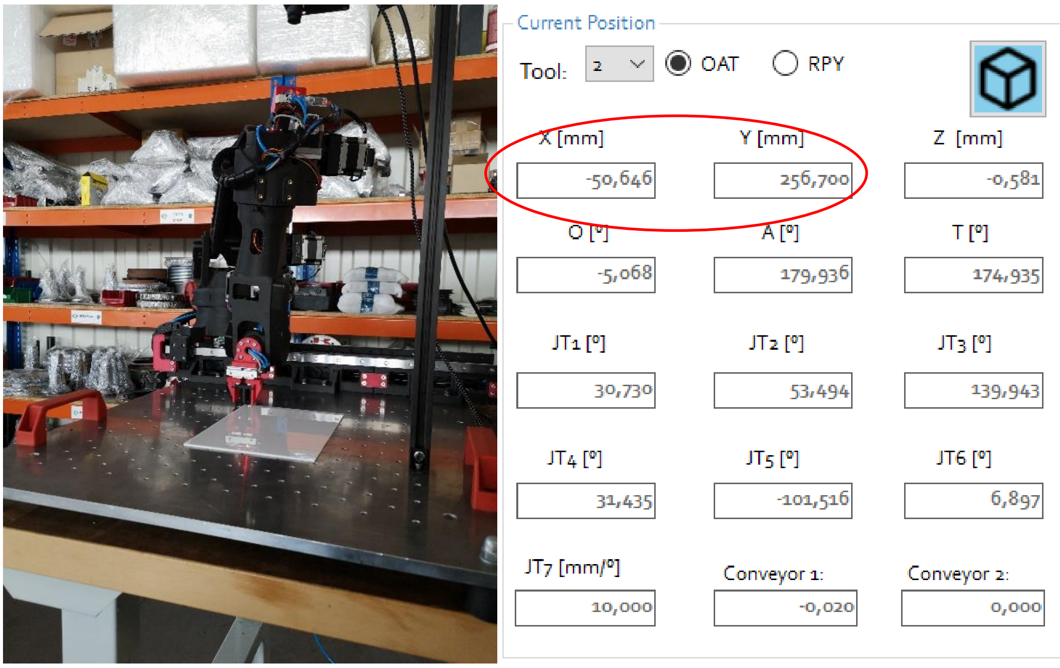


Move robot to the corner and save the position.

10.2.3 Point 3



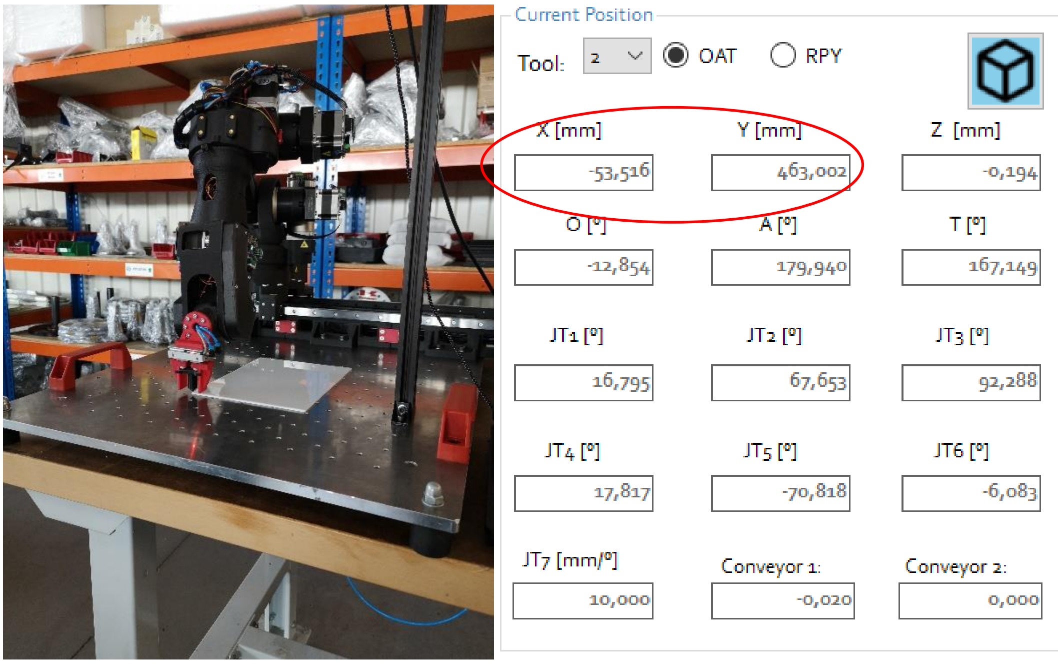
Using mouse cursor read the pixel position of the corner



Move robot to the corner and save the position.

10.2.4 Point 4

Using mouse cursor read the pixel position of the corner



Move robot to the corner and save the position.

10.3 Input the saved positions into the program

```
              1             2             3            4
c_frm_px = [[101, 194],[317, 189],[314, 32],[100, 35]]
r_frm_p = [[132.4, 258.2],[128.7, 441.8],[-4.9, 440.8],[-1.8, 260.0]]
                 1                 2                 3               4
```

- *thresholds* – determines how dark (black) in a grayscale detected object is. It uses the LAB color space (thresholds = [Lmin, Lmax, Amin, Amax, Bmin, Bmax]). When detecting objects in grayscale, you only need to set the first 2 parameters.
- *off_x* – possible offset in the x-direction of the detected points (in millimeters).
- *off_y* – possible offset in the y-direction of the detected points (in millimeters).
- *off_ang* – possible offset of the rotation angle of the camera coordinate system and the robot workspace (in radians).

10.4 Serial communication

The following parameters refer to the configuration of communication between the camera and the robot:

```
#parametry komunikacji
trigger = 'T'
separator1 = '/'
separator2 = '/'
separator3 = '/'
```

- *trigger* – determines what character the camera expects. If the received sign is the same as the trigger variable, the camera will send back the calculated coordinates.
- *separator1* – determines what character will separate the sent *X* coordinate from the *Y* coordinate.
- *separator2* – determines what character will separate the sent *X* coordinate from the angle of rotation - *A*.
- *separator3* – determines what character will appear at the end of the sent batch of data.

For the above example, after receiving the "*T*" sign, the data sent will take the form: *X/Y/A/*

10.5 Quick program setup

1. Mount the camera in the desired location so that the working area can be seen perpendicularly.
2. Turn on the camera and run the calibration program.
3. set the *w_x, w_y, w_width* and *w_hight* parameters so that the preview shows the entire working area dedicated to the camera.
4. Set the *fisheye_corr* parameter to best correct the image distortion caused by the camera's fisheye.
5. enter the appropriate coordinates in the camera view *c_frm_px* [px] and the manipulator workspace *r_frm_p* [mm] (remember the correct order!) as described in the above section.
6. Set the *thresholds* parameter so that your objects are detected properly.
7. Based on the results obtained, make corrections with the parameters *off_x, off_y, off_ang.*
8. Configure the parameters for communication by setting *trigger, separator1, separator2* and *separator3* to the desired characters.

10.6 Algorithm description

Based on the entered parameters (section 1.1), the program calculates successively:

- scale - the ratio of millimeters in the workspace to the pixels visible in the camera image,
- the angle of rotation of the camera in relation to the robot's workspace,
- the corresponding sum angle.

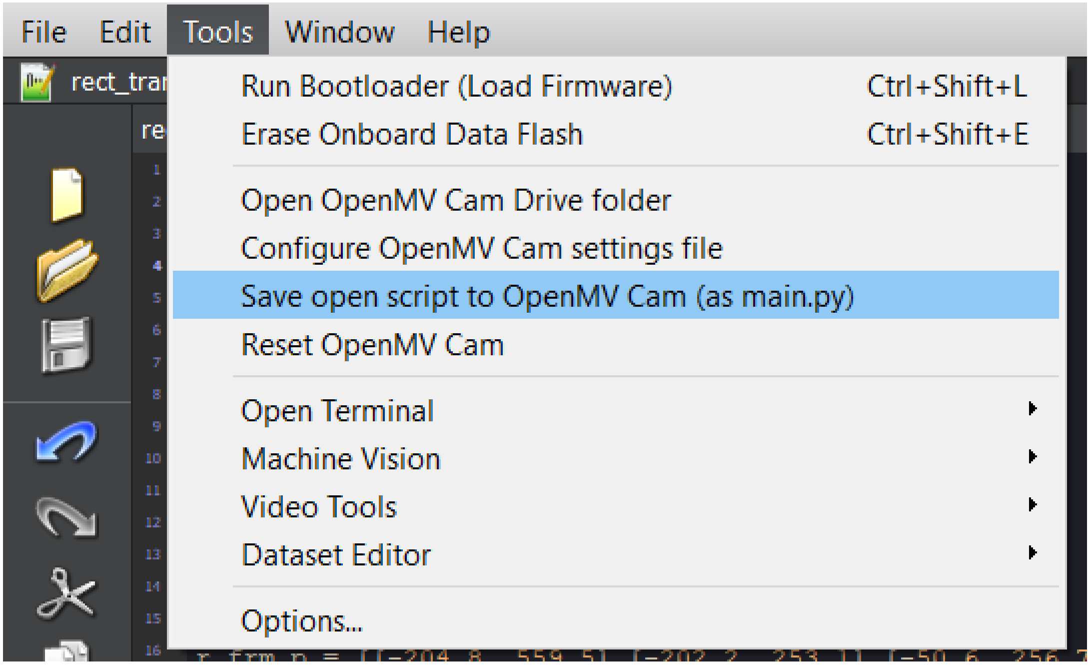
Then the rotation matrix is created and the displacement vector is calculated. The results obtained in this way allow to create a transformation matrix.

10.7 Infinite loop program

1. The program checks the received data via UART. If the received data is as expected, an attempt is made to detect the object.
2. If the object is detected, the coordinates of its occurrence (pixels) are collected.
3. The coordinates are converted from pixels to millimeters (using a transformation matrix).
4. If the object is not detected, the camera will send back the coordinates (X=0, Y=0).
   The converted coordinates are sent via UART.

10.8 Save the program in the camera

After successful calibration, the applied program can be saved to the camera's memory so that it runs when the camera is turned on. To do this, expand the "*tools*" tab from the ribbon of the OpenMV IDE application and select "Save open script to OpenMV Cam (*as main.py*)".



The program saved in this way will start automatically when the camera is connected to the power supply (from a computer via USB port or using an external power supply).

11 Manufacturer information

For further questions, contact Kawasaki Robotics support.

Contact:

Kawasaki Robotics GmbH

tech-support@kawasakirobot.de

+49 (0) 2131 – 3426 – 1310

Kawasaki Robot
Vision System Manual

2024-01: 2nd Edition

Publication: KAWASAKI Robotics GmbH

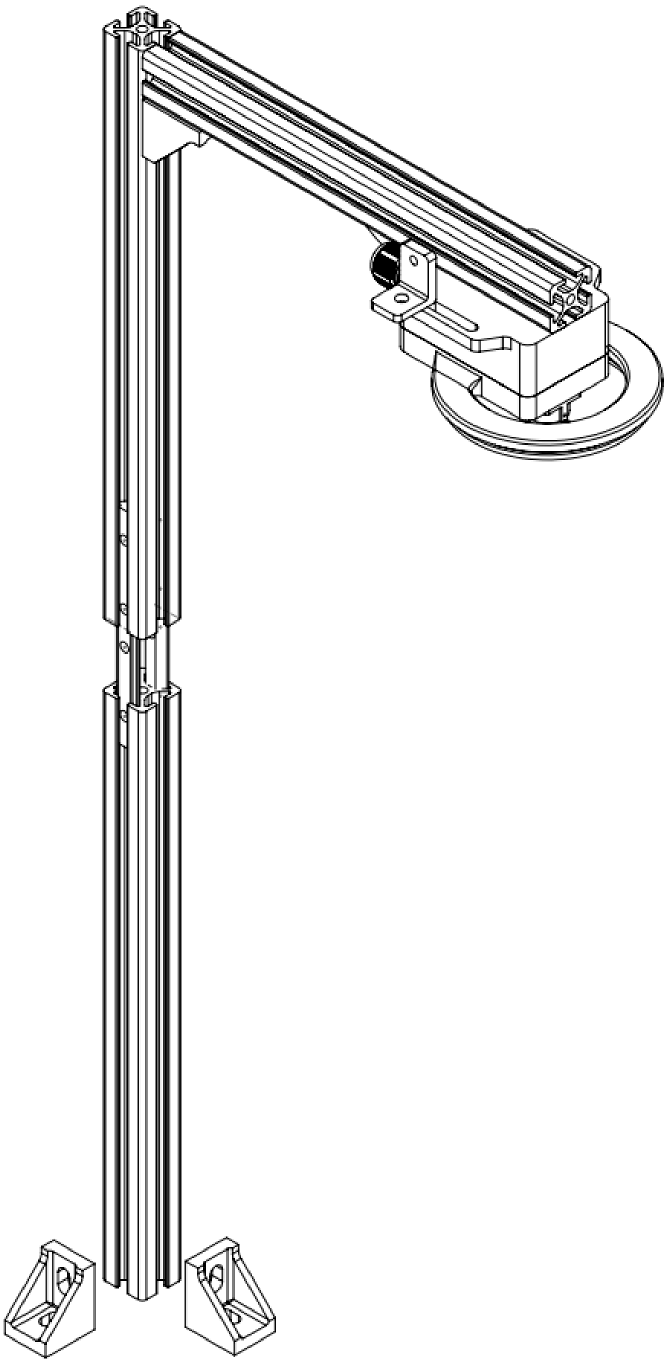
Appendix A – Camera stand assembly

Connect together two 2020x250mm profiles with T-NUT butt joint brackets,

Connect angle brackets to profiles with M5 screws,

Connect 2020x200mm to other profiles with angle bracket, t-nuts and M5 screws,

Install camera to the 2020x200mm using delivered 3D printed angle brackets, T-nuts, M5 screws and M3 screws screws with nuts,



Appendix B – calibration program code

```
import sensor, image, time, lcd, math
from pyb import LED
from pyb import UART
from pyb import Pin

##calibration

w_x = 170 #dimensions and location of the examined area (part of the cam-
era view: x coefficient, y coefficient, width, height)
w_y = 100
w_width = 346
w_hight = 260

fisheye_corr = 0.7 #camera fisheye correction

c_frm_px = [[18, 232],[334, 235],[330, 17],[24, 7]] #pixel coordinates of
physical points
r_frm_p = [[-135.4, 409.6],[-131.0, 203.7],[22.7, 197.6],[16.6,
408.9]]#physical coordinates of points

thresholds = (0, 55) #how gray (black) grayscale objects are to be de-
tected

off_x = 0  #offset (possible position correction in mm)
off_y = 0
off_ang = 0.09004

##########################################################################
##########################

temp = 0;
for i in range(4):
    if r_frm_p[i][1] + r_frm_p[(i + 1)%4][1] < r_frm_p[temp][1] +
r_frm_p[(temp + 1)%4][1]:
        temp = i

pos_case = temp


H0_C = [[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0]]      #trans-
formation matrix
P0 = [[0], [0], [0], [0]]                                          #result
point
R_z = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]                            #rota-
tion matrix

#scale mm/px
r_diff = math.sqrt(math.pow((r_frm_p[2][0] - r_frm_p[0][0]), 2) +
math.pow((r_frm_p[2][1] - r_frm_p[0][1]), 2))
c_diff = math.sqrt(math.pow((c_frm_px[2][0] - c_frm_px[0][0]), 2) +
math.pow((c_frm_px[2][1] - c_frm_px[0][1]), 2))

scale = r_diff/c_diff

#rotation r
d1_r = math.sqrt(math.pow((r_frm_p[1][1] - r_frm_p[2][1]), 2))
d2_r = math.sqrt(math.pow((r_frm_p[1][0] - r_frm_p[2][0]), 2) +
math.pow((r_frm_p[1][1]) - (r_frm_p[2][1]), 2))
```

```
r_angle = (math.asin(d1_r/d2_r))

#rotation c (based on pixel points)
d1_c = math.sqrt(math.pow((c_frm_px[1][1] - c_frm_px[2][1]), 2))
d2_c = math.sqrt(math.pow((c_frm_px[1][0] - c_frm_px[2][0]), 2) +
math.pow((c_frm_px[1][1]) - (c_frm_px[2][1]), 2))
c_angle = math.asin(d1_c/d2_c)

#angles sum
if pos_case == 1 or pos_case == 2:
    pos_case_ang = 0
else:
    pos_case_ang = 1

angle = c_angle + r_angle + (math.pi * pos_case_ang) - off_ang
#rotation 180 (relative to x)
R180 = [[1, 0, 0], [0, math.cos(math.pi), -math.sin(math.pi) ], [0,
math.sin(math.pi), math.cos(math.pi)]]

#rotation matrix of camera arrays by angle
R_a = [[math.cos(angle), -math.sin(angle), 0], [math.sin(angle),
math.cos(angle), 0], [0, 0, 1]]

#rotation matrix
for i in range(len(R180)):
   for j in range(len(R_a[0])):
       for k in range(len(R_a)):
           R_z[i][j] += R180[i][k] * R_a[k][j]

#shift vector - finding the shift vector depending on the position case
x_ = ((c_frm_px[pos_case][1] + c_frm_px[(pos_case+1)%4][1])/2)
y_ = ((c_frm_px[pos_case][0] + c_frm_px[(pos_case+1)%4][0])/2)

d0_x = x_ * math.cos(math.pi/2 - angle) - y_ * math.sin(math.pi/2 - angle)
+ (r_frm_p[pos_case][0] + r_frm_p[(pos_case+1)%4][0])/(2 * scale)
d0_y = x_ * math.sin(math.pi/2 - angle) + y_ * math.cos(math.pi/2 - angle)
+ (r_frm_p[pos_case][1] + r_frm_p[(pos_case+1)%4][1])/(2 * scale)

d0_C = [d0_x, d0_y, 0, 1]

#H0_C matrix (rotation matrix and translation matrix in one)
for i in range(len(R_z)):
    for j in range(len(R_z[i])):
        H0_C[i][j] = R_z[i][j]
for i in range(len(d0_C)):
    H0_C[i][3] = d0_C[i]

###camera settings###

clock = time.clock()

r = (0,0,269,217)
window = (w_x,w_y,w_width,w_hight)
low_threshold = (30, 160)
uart = UART(3, 9600, timeout_char=1000)
angle = 0
red_led   = LED(1)
green_led = LED(2)
blue_led  = LED(3)

a = 0
```

```
red_led.on()
green_led.on()
blue_led.on()

pin9 = Pin('P9', Pin.OUT_PP, Pin.PULL_DOWN)
pin9.high()

sensor.reset()
sensor.set_pixformat(sensor.GRAYSCALE)
sensor.set_framesize(sensor.VGA)
sensor.set_windowing(window)
sensor.skip_frames(time = 2000)
sensor.set_auto_gain(True)                # must be turned off for color
tracking
sensor.set_auto_whitebal(True)            # must be turned off for color
tracking
lcd.init()

clock = time.clock()

# UART 3
uart = UART(3, 500000)

while(True):

    #finding the object
    img = sensor.snapshot()
    img.lens_corr(fisheye_corr)
    isblob = 0;
    roi_set = (roi_val, roi_val, w_width - 2 * roi_val, w_hight - 2 *
roi_val)
    for blob in img.find_blobs([thresholds], roi = roi_set, pixels_thresh-
old=100, area_threshold=100, merge=True):

        # These values depend on the blob not being circular - otherwise
they will be shaky.
        if blob.elongation() > 0.5:
            img.draw_edges(blob.min_corners(), color=0)
            img.draw_line(blob.major_axis_line(), color=0)
            img.draw_line(blob.minor_axis_line(), color=0)

        # These values are stable all the time.
        img.draw_rectangle(blob.rect(), color=127)
        img.draw_cross(blob.cx(), blob.cy(), color=127)

        # Note - the blob rotation is unique to 0-180 only.
        img.draw_keypoints([(blob.cx(), blob.cy(), int(math.de-
grees(blob.rotation())))], size=40, color=127)
        isblob += 1

    #found object coordinates
    if isblob == 1:
        X_Location = blob.cx()
        Y_Location = blob.cy()
    else:
        X_Location = 0
        Y_Location = 0
    PC = [[X_Location], [Y_Location], [0], [1]]

    #calcualting the angle
    if isblob == 1:
```

```
        blob_corn = blob.min_corners()
        blob_corn = sorted(blob_corn)
        d1_blb = math.sqrt(math.pow((blob_corn[2][1] - blob_corn[3][1]),
2))
        d2_blb = math.sqrt(math.pow((blob_corn[2][0] - blob_corn[3][0]),
2) + math.pow((blob_corn[2][1]) - (blob_corn[3][1]), 2))
        if blob_corn[2][1] > blob_corn[3][1]:
            blob_ang = (math.pi - math.asin(d1_blb/d2_blb) + c_angle) *
180/math.pi
        else:
            blob_ang = (math.asin(d1_blb/d2_blb) + c_angle) * 180/math.pi
    else:
        blob_ang = 0

    while blob_ang > 90:
        blob_ang -= 90

    #transform camera coordinate to robot coordinate
    for i in range(len(H0_C)):
        for j in range(len(PC[i])):
            for k in range(len(PC)):
                P0[i][j] += H0_C[i][k] * PC[k][j] * scale

    #offset
    P0[0][0] -= off_x
    P0[1][0] -= off_y

    #clear data
    print(P0[0], P0[1])
    print(blob_ang)
    P0[0][0] = 0
    P0[1][0] = 0
    P0[2][0] = 0
    P0[3][0] = 0
```

Appendix C – cube find program code

```
import sensor, image, time, lcd, math
from pyb import LED
from pyb import UART
from pyb import Pin

##calibration

w_x = 170 #dimensions and location of the examined area (part of the cam-
era view: x coefficient, y coefficient, width, height)
w_y = 100
w_width = 346
w_hight = 260

fisheye_corr = 0.7 #camera fisheye correction

c_frm_px = [[18, 232],[334, 235],[330, 17],[24, 7]] #pixel coordinates of
physical points
r_frm_p = [[-135.4, 409.6],[-131.0, 203.7],[22.7, 197.6],[16.6,
408.9]]#physical coordinates of points

thresholds = (0, 55) #how gray (black) grayscale objects are to be de-
tected

off_x = 0  #offset (possible position correction in mm)
off_y = 0
off_ang = 0.09004

#########################################################################
##########################

temp = 0;
for i in range(4):
    if r_frm_p[i][1] + r_frm_p[(i + 1)%4][1] < r_frm_p[temp][1] +
r_frm_p[(temp + 1)%4][1]:
        temp = i

pos_case = temp


H0_C = [[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0],[0, 0, 0, 0]]      #trans-
formation matrix
P0 = [[0], [0], [0], [0]]                                         #result
point
R_z = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]                           #rota-
tion matrix

#scale mm/px
r_diff = math.sqrt(math.pow((r_frm_p[2][0] - r_frm_p[0][0]), 2) +
math.pow((r_frm_p[2][1] - r_frm_p[0][1]), 2))
c_diff = math.sqrt(math.pow((c_frm_px[2][0] - c_frm_px[0][0]), 2) +
math.pow((c_frm_px[2][1] - c_frm_px[0][1]), 2))

scale = r_diff/c_diff

#rotation r
d1_r = math.sqrt(math.pow((r_frm_p[1][1] - r_frm_p[2][1]), 2))
d2_r = math.sqrt(math.pow((r_frm_p[1][0] - r_frm_p[2][0]), 2) +
math.pow((r_frm_p[1][1]) - (r_frm_p[2][1]), 2))
```

```
r_angle = (math.asin(d1_r/d2_r))

#rotation c (based on pixel points)
d1_c = math.sqrt(math.pow((c_frm_px[1][1] - c_frm_px[2][1]), 2))
d2_c = math.sqrt(math.pow((c_frm_px[1][0] - c_frm_px[2][0]), 2) +
math.pow((c_frm_px[1][1]) - (c_frm_px[2][1]), 2))
c_angle = math.asin(d1_c/d2_c)

#angles sum
if pos_case == 1 or pos_case == 2:
    pos_case_ang = 0
else:
    pos_case_ang = 1

angle = c_angle + r_angle + (math.pi * pos_case_ang) - off_ang
#rotation 180 (relative to x)
R180 = [[1, 0, 0], [0, math.cos(math.pi), -math.sin(math.pi) ], [0,
math.sin(math.pi), math.cos(math.pi)]]

#rotation matrix of camera arrays by angle
R_a = [[math.cos(angle), -math.sin(angle), 0], [math.sin(angle),
math.cos(angle), 0], [0, 0, 1]]

#rotation matrix
for i in range(len(R180)):
   for j in range(len(R_a[0])):
       for k in range(len(R_a)):
           R_z[i][j] += R180[i][k] * R_a[k][j]

#shift vector - finding the shift vector depending on the position case
x_ = ((c_frm_px[pos_case][1] + c_frm_px[(pos_case+1)%4][1])/2)
y_ = ((c_frm_px[pos_case][0] + c_frm_px[(pos_case+1)%4][0])/2)

d0_x = x_ * math.cos(math.pi/2 - angle) - y_ * math.sin(math.pi/2 - angle)
+ (r_frm_p[pos_case][0] + r_frm_p[(pos_case+1)%4][0])/(2 * scale)
d0_y = x_ * math.sin(math.pi/2 - angle) + y_ * math.cos(math.pi/2 - angle)
+ (r_frm_p[pos_case][1] + r_frm_p[(pos_case+1)%4][1])/(2 * scale)

d0_C = [d0_x, d0_y, 0, 1]

#H0_C matrix (rotation matrix and translation matrix in one)
for i in range(len(R_z)):
    for j in range(len(R_z[i])):
        H0_C[i][j] = R_z[i][j]
for i in range(len(d0_C)):
    H0_C[i][3] = d0_C[i]

###camera settings###

clock = time.clock()

r = (0,0,269,217)
window = (w_x,w_y,w_width,w_hight)
low_threshold = (30, 160)
uart = UART(3, 9600, timeout_char=1000)
angle = 0
red_led   = LED(1)
green_led = LED(2)
blue_led  = LED(3)

a = 0
```

```
red_led.on()
green_led.on()
blue_led.on()

pin9 = Pin('P9', Pin.OUT_PP, Pin.PULL_DOWN)
pin9.high()

sensor.reset()
sensor.set_pixformat(sensor.GRAYSCALE)
sensor.set_framesize(sensor.VGA)
sensor.set_windowing(window)
sensor.skip_frames(time = 2000)
sensor.set_auto_gain(True)              # must be turned off for color
tracking
sensor.set_auto_whitebal(True)          # must be turned off for color
tracking
lcd.init()

clock = time.clock()

# UART 3
uart = UART(3, 115200)

while(True):

    clock.tick()
    if UART.any(uart) > 0:
       Data = uart.read(1)
       if ord(Data) == ord(trigger):

            #finding the object
            img = sensor.snapshot()
            img.lens_corr(fisheye_corr)
            isblob = 0;
            roi_set = (roi_val, roi_val, w_width - 2 * roi_val, w_hight -
2 * roi_val)
            for blob in img.find_blobs([thresholds], roi = roi_set, pix-
els_threshold=100, area_threshold=100, merge=True):

                # These values depend on the blob not being circular -
otherwise they will be shaky.
                if blob.elongation() > 0.5:
                    img.draw_edges(blob.min_corners(), color=0)
                    img.draw_line(blob.major_axis_line(), color=0)
                    img.draw_line(blob.minor_axis_line(), color=0)

                # These values are stable all the time.
                img.draw_rectangle(blob.rect(), color=127)
                img.draw_cross(blob.cx(), blob.cy(), color=127)

                # Note - the blob rotation is unique to 0-180 only.
                img.draw_keypoints([(blob.cx(), blob.cy(), int(math.de-
grees(blob.rotation())))], size=40, color=127)
                isblob += 1

            #assign the coordinates
            if isblob == 1:
                X_Location = blob.cx()
                Y_Location = blob.cy()
            else:
                X_Location = 0
```

```
                Y_Location = 0
            PC = [[X_Location], [Y_Location], [0], [1]]

            #calculate the angle
            if isblob == 1:
                blob_corn = blob.min_corners()
                blob_corn = sorted(blob_corn)
                d1_blb = math.sqrt(math.pow((blob_corn[2][1] -
blob_corn[3][1]), 2))
                d2_blb = math.sqrt(math.pow((blob_corn[2][0] -
blob_corn[3][0]), 2) + math.pow((blob_corn[2][1]) - (blob_corn[3][1]), 2))
                if blob_corn[2][1] > blob_corn[3][1]:
                    blob_ang = (math.pi - math.asin(d1_blb/d2_blb) + c_an-
gle) * 180/math.pi
                else:
                    blob_ang = (math.asin(d1_blb/d2_blb) + c_angle) *
180/math.pi #kat w przliczeniu na stopnie
            else:
                blob_ang = 0

            while blob_ang > 90:
                blob_ang -= 90

            #transform camera to robot coordinates
            for i in range(len(H0_C)):
                for j in range(len(PC[i])):
                    for k in range(len(PC)):
                        P0[i][j] += H0_C[i][k] * PC[k][j] * scale

            #ewentualny offset
            P0[0][0] -= off_x
            P0[1][0] -= off_y

            #calc cordinates
            if isblob == 1:
                Pxstr = str(P0[0])[1:len(str(P0[0])) - 1]
                Pystr = str(P0[1])[1:len(str(P0[1])) - 1]
                Astr = str(blob_ang)
            else:
                Pxstr = "0"
                Pystr = "0"
                Astr = "0"

            uart.write(Pxstr)
            uart.write(separator1)
            uart.write(Pystr)
            uart.write(separator2)
            uart.write(Astr)
            uart.write(separator3)

            #clear data
            print(Pxstr, Pystr)
            print(blob_ang)
            P0[0][0] = 0
            P0[1][0] = 0
            P0[2][0] = 0
            P0[3][0] = 0
```

Appendix D – robot program code

```
.PROGRAM CUBE
  TOOL 1
  SPEED 100 MM/S ALWAYS
  HOME
  pick_height = 5
  ;P0 - reference orientation point over pick area
  ;P1 - cube pick position from cubes feeder
  ;P2 - position over cubes bin
  ;#P0 - cube drop position under camera
  SIGNAL 1
  SWAIT 1001 ;wait for user input
  PULSE 4,1
  LAPPRO P1, 50
  SPEED 40 MM/S
  LMOVE P1
  TWAIT 0.5
  CLOSEI
  TWAIT 0.5
  LDEPART 50
  JMOVE #P0
  OPENI
  HOME
  SEND "T" ;trigger the camera
  WHILE EXISTCOM == false DO
    twait 0.1
  END
  $temp = RECEIVE ;receive and decode the frame
  ;input frame X/Y/ANGLE/
  $temp2 = $decode($temp,"/")
  $temp3 = $decode($temp, "/")
  $temp4 = $decode($temp, "/")
  dataX = VAL($temp2)
  dataY = VAL($temp3)
  dataA = VAL($temp4)
  IF ((dataX <> 0) AND (dataY <> 0)) THEN
    POINT test = TRANS(dataX,dataY,pick_height,0,0,0)
    POINT\OAT pick = P0 ;reference orientation point
    POINT pick = pick + RZ(angle) ;adding angle of the cube
    LAPPRO pick, 40
    SPEED 40 MM/S
    LMOVE pick
    TWAIT 0.5
    CLOSEI
    TWAIT 0.5
    LDEPART 50
    JMOVE P2
    OPENI
    TWAIT 0.5
  ELSE
    PRINT "No object found"
  END
.END
```