FH Vorarlberg
University of Applied Sciences

Jeff Junior

3 Axis Robot following a Laserpointer by using Stereovision and Templatematching

Applied Robotics

Vorarlberg University of Applied Sciences
Master's in Mechatronics

Submitted to

Prof. (FH) Dipl.-Ing. Robert Amann

Handed in by

Johannes Wuestner, Nicolai Schwartze

Dornbirn, 15.02.2020

Contents

List of Figures **II**

List of Tables **III**

1 Task Description **1**

2 Design **2**

3 Calculation **4**

3.1 Backwards Kinematic 4

3.1.1 Implementation 5

3.2 Goal Robot Configuration 6

4 Stereo Vision **8**

4.1 Camera Calibration 8

4.2 Laser Point Detection 9

4.2.1 Method 1: HSV Threshold 9

4.2.2 Method 2: Template Matching 10

5 Main App **12**

6 Conclusion **14**

Bibliography **15**

Appendices **16**

A Drawings **17**

B Motor Parameters **22**

List of Figures

2.1 final assembly of the robot 3

3.1 robot axis coordinate systems used to calculate the link parameters 4
3.2 camera calibration app flowchart 7

4.1 hand-eye coordinate system transformation 9
4.2 camera calibration app flowchart 11

5.1 camera calibration app flowchart 13

6.1 resting offset between laser points 14

List of Tables

2.1 comparison of the cameras before and after modification 2
2.2 3D render of the assembled robot in SolidWorks 3

3.1 table of link parameters 5

4.1 laser point template as created by the configuration app 10

1 Task Description

This project is about constructing and building a 3-axis robot. The robot is equipped with two cameras and a red laser-pointer. The user can take a green laser-pointer to direct the robot into a different position. In order to avoid permanent sight-damage, the robot should avoid shining the laser into faces of surrounding humans. The following list contains the main working steps:

- design and construct the 3-axis robot
- manufacture and assemble the robot parts
- develop the backwards kinematic
- implement the backwards kinematic in C and compile to DLL
- find green laser point in 3D stereo image with OpenCV in Python
- drive the red laser point of the robot to the green laser point of the human
- planned but optional:
  - plan path from current position to green laser position
  - implement controller to correct deviance between red and green laser point
  - avoid human faces along the way

2 Design

The robot consists of 3 Dynamixel MX-64AT motors, 2 Microsoft HD 3000 lifecam and a laser module DB635-1-3-FA(14x45)-ADJ from Picotronic. Aluminium frames are attached to the motors to build the framework of the robot. The cameras and laser-pointer are connected with aluminium brackets to the robot. The design of the robot and its components is established with SolidWords. The whole robot is mounted on aluminium profiles to improve the handling and to store of the electrical hardware. Beside the cables, a power supply for the motors and laser pointer, an emergency stop button and communication components for the motors are required. The cameras are directly connected to the laptop or computer via the USB ports. This means that the laptop requires 3 USB sockets. A connection via a USB-Hub failed as the OpenCV library could not find the cameras.
The table 2.1 shows how the cameras were modified in order to properly attach them to the robot tool.
The table 2.2 displays a 3D render of the completely assembled robot.
Finally, the finished robot can be seen in figure 2.1

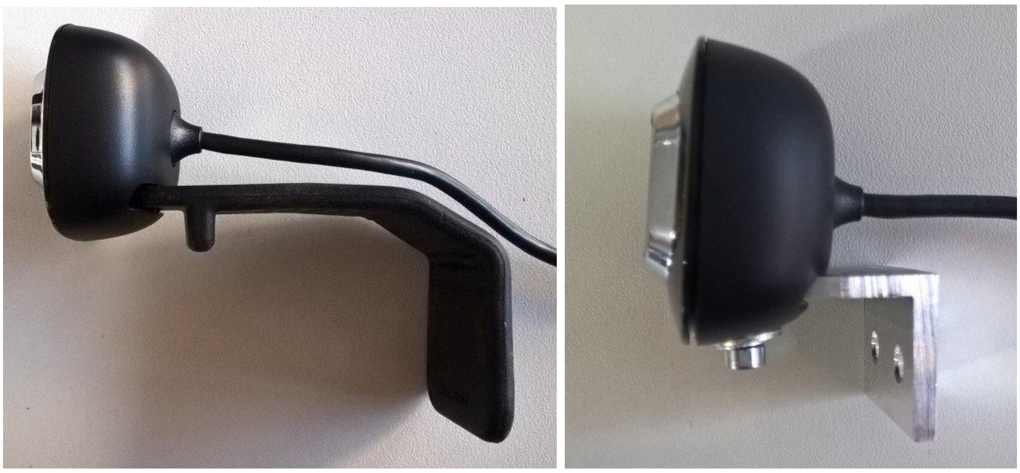


Table 2.1: comparison of the cameras before and after modification

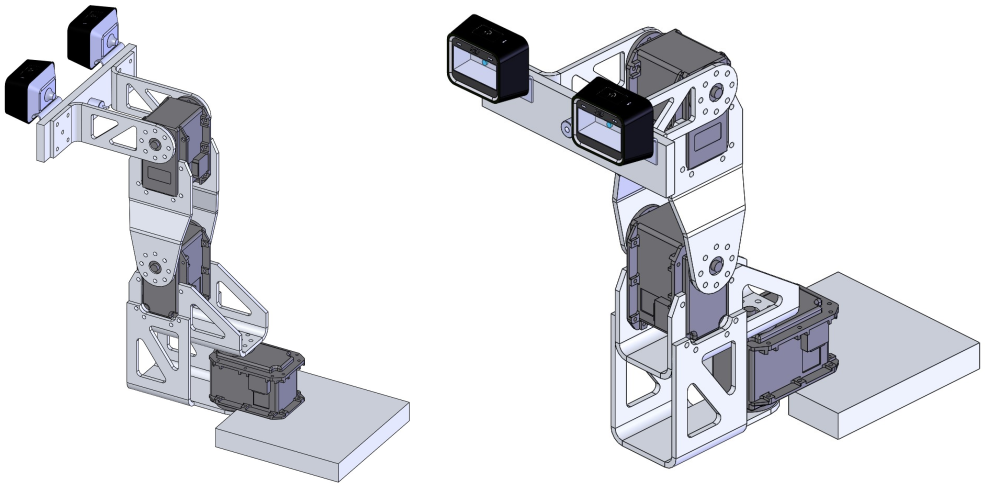


Table 2.2: 3D render of the assembled robot in SolidWorks

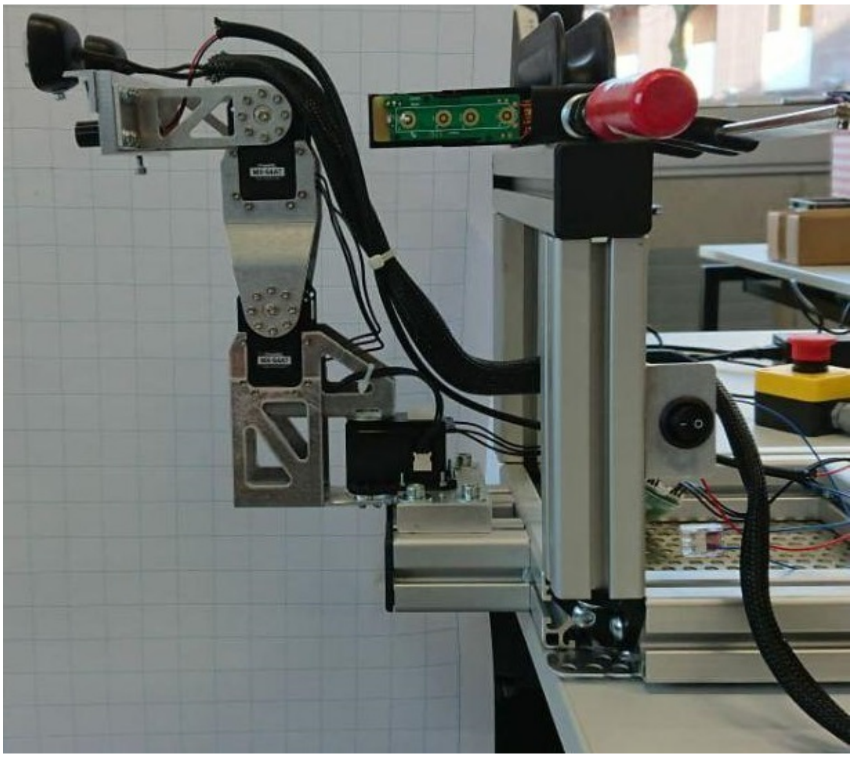


Figure 2.1: final assembly of the robot

3 Calculation

This chapter describes the mathematical formulations that are needed to control the robot. This includes the backwards kinematics as well as the formulas for setting $\theta_{0,1,2}$.

3.1 Backwards Kinematic

The following figure 3.1 shows the coordinate systems that are used to get the link parameters shown in table 3.1 below. The link parameters are established using the Deviant Hartenberg convention mentioned in Craig 2018, p. 70- 79.

With the knowledge of the link parameters, the transformation matrices are generated in the next step. The full matrix can be seen in equation 3.1

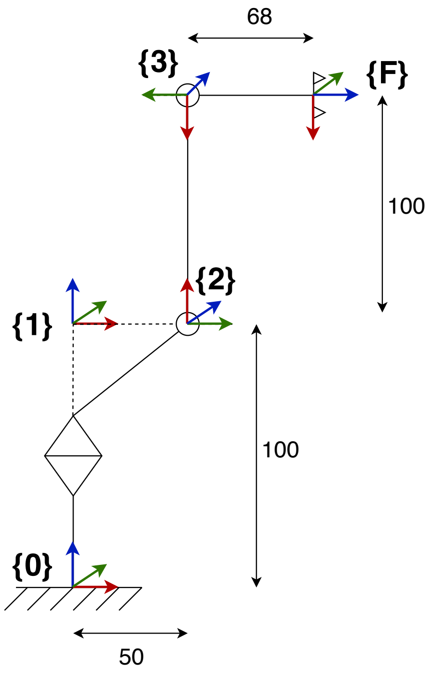


Figure 3.1: robot axis coordinate systems used to calculate the link parameters

i	α_{i-1}	a_{i-1}	d_i	θ_{i-1}
1	0	0	100	θ_0
2	-90	50	0	$\theta_1 - 90$
3	0	100	0	$\theta_2 + 180$
F	90	0	68	0

Table 3.1: table of link parameters

$$T = \begin{bmatrix} -c0c1s2 - c0c2s1 & -s0 & c0c1c2 - c0s1s2 & a2c0 + dF(c0c1c2 - c0s1s2) + a3c0s1 \\ -c1s0s2 - c2s0s1 & c0 & c1c2s0 - s0s1s2 & a2s0 + dF(c1c2s0 - s0s1s2) + a3s0s1 \\ s1s2 - c1c2 & 0 & -c1s2 - c2s1 & d1 + a3c1 + dF(-c1s2 - c2s1) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

3.1.1 Implementation

For controlling the robot, several functions are written in C and compiled to a DLL. The functions are built on the Dynamixel library ROBOTIS-GIT 2019. The DLL provides the following methods:

- `int robot_start();`
  Sets up the communication with the robot by returning the port number. This number must be provided in every other function. Further this function sets up the main parameters of every motor like the velocity limits.

- `void robot_stop(int port_num);`
  Stops the robot by driving to the home position and freeing the device with the port number.

- `void robot_home(int port_num);`
  Separate function for driving the robot to the home position.

- `int robot_drive(float z, float alpha, float beta, int port_num);`
  Tells the robot to drive to a specific position specified by α, β and z. The motion towards this position is not planned.

- `int robot_reached_target(float z, float alpha, float beta, int port_num);`
  This function checks if the robot is at the specified α, β and z position and returns 1 if it reached the target (otherwise 0).

- `void robot_set_theta(float theta0, float theta1, float theta2, int port_num);`
  Instead of specifying a global position, this function allows to set the axis angle θ by its own.
- `float robot_get_theta0(int port_num);`
  Returns the angle θ_0 that the robot is at that time.
- `float robot_get_theta1(int port_num);`
  Returns the angle θ_1 that the robot is at that time.
- `float robot_get_theta2(int port_num);`
  Returns the angle θ_2 that the robot is at that time.

The functions *robot_ drive* as well as the function *robot_ set_ theta* are non-blocking. This means that the function returns, before the robot reached the target. Blocking functions can be easily created outside the DLL by simply polling the functions *robot_ reached_ target* or *robot_ get_ theta*.

3.2 Goal Robot Configuration

The laser point on the wall can be described by only to 2 DOF, vertical and horizontal coordinates. To track the laser point, the robot has to change 2 DOF likewise. Even though the robot has 3 rotation axis, it only satisfy 2 DOF, because $theta_1$ and $theta_2$ directly depend to each other. Therefore, one of these angles can be set to any arbitrary angle. In this case $theta_1$ is constantly set to 0°. With the 2 cameras and the knowledge of the transformation matrices of the robot, the vertical and horizontal coordinates of the laser point are received and transformed into 3D coordinates of the robot base coordinate system.

As a result, the rotation angle of the motors can be calculated. θ_0 is a simple geometric problem and can be easily calculated with trigonometric functions (see figure 3.2). The equation 3.3 for θ_2 is slightly more advanced. Additionally, the length and height of the robot components have to be taken in consideration.

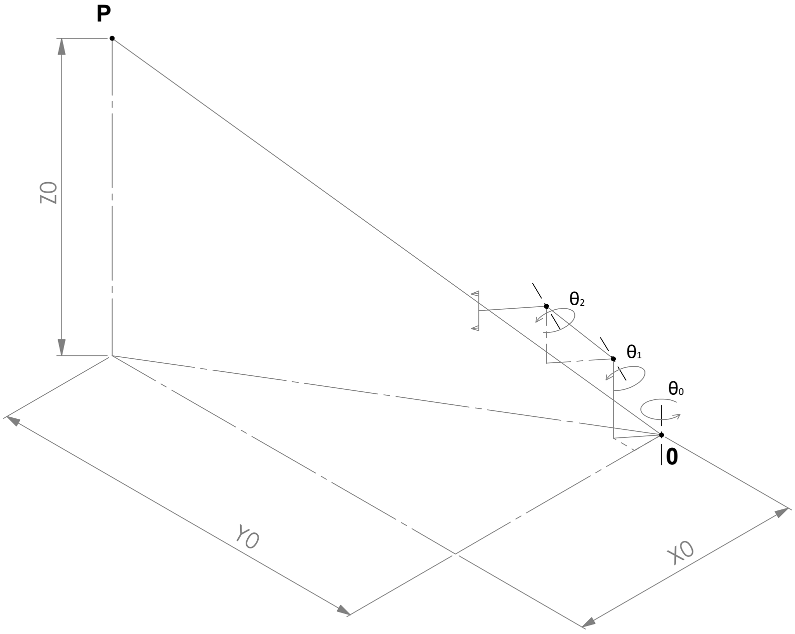


Figure 3.2: camera calibration app flowchart

$$\theta_0 = arctan\left(\frac{Y_0}{X_0}\right) \tag{3.2}$$

$$\theta_2 = -1 \cdot arctan\left(\frac{Z_0 - 100 - 100cos(\theta_1)}{\sqrt{X_0^2 + Y_0^2} - 50 - 100sin(\theta_1)}\right) \tag{3.3}$$

4 Stereo Vision

The idea of this project is to drive directly to the correct point, without using a closed loop controller of any sort. This is done by measuring the 3D point with a stereo camera and calculating back to the desired robot configuration (as shown in 3.2). In order to measure the correct point, the cameras need to be calibrated. Further, a reliable method for finding the laser-point is needed. All vision related tasks are done with *The OpenCV Reference Manual* 2019

4.1 Camera Calibration

At first a camera calibration application is written, that allows to take images, calculate the two single camera matrices and estimates the translation and rotation between the two cameras. The flowchart in figure 4.2 shows the main outline of the camera calibration app.

Three quality measures are deployed to verify that the matrices are correct:

- measuring a known 3D structure with the calibrated stereo camera tool
- using the same pictures of the chessboard in MATLAB and comparing the calculated matrices
- plausibility check for the translation vector and the rotation matrix between the cameras

Besides the camera calibration towards each other, the main camera (left) must also be calibrated to the TCP. This is called the hand-eye calibration. The following matrix 4.1 shows the hand-eye calibration. Although there are more complex methods to generate this transformation matrix, it can also be measured from the real tool. The figure 4.1 shows the rotation between the TCP coordinate system and the camera coordinate system.

$$HE = \begin{bmatrix} 0.0 & 1.0 & 0.0 & -28.5 \\ -1.0 & 0.0 & 0.0 & 45.0 \\ 0.0 & 0.0 & 1.0 & 28.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{4.1}$$

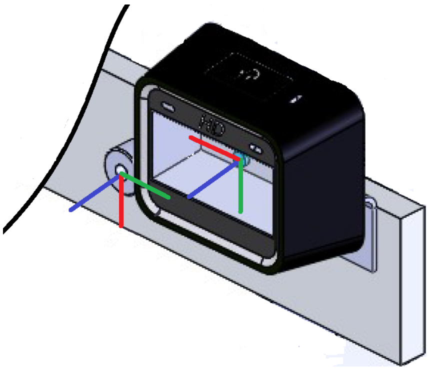


Figure 4.1: hand-eye coordinate system transformation

4.2 Laser Point Detection

The second part is about detecting the laser point in an image. To make the task simpler, the laser point may only be projected onto a white flip board chart.
Two methods for detecting the laser point are tested.

4.2.1 Method 1: HSV Threshold

As lasers are monochromatic one could think about looking for a certain colour. The idea is to define a threshold in HSV-colour space and only look at the pixels that are within this band. By applying a dilation function to the binary image, the weakly connected pixels are morphed together and form blobs. Then the function *findCountours* is deployed to detect the connected areas of white pixels. Finally, the first (biggest) area contour is taken and the centre of mass is calculated. This is the 2D point in one image.
This method did not work very well. Some problems have been identified:

- Taking the biggest area is not very sensible. More constraints could be helpful like: size threshold, convexity or previous point.
- As the automatic mode of the camera is not deactivated the colour of the laser point might change with different lighting scenarios.
- The definition of the colour threshold is not trivial: Either too many pixels or too few are taken. Especially with a colourful surrounding this trade-off is hard to control.

4.2.2 Method 2: Template Matching

The second method uses template matching to find the green laser point. The process outline as follows: a template slides over the image and at every pixel they are compared. The comparison results a scalar value which are then stacked together to create something like a ‘heat-map’. The larger the scalar is at an index, the better is the correspondence between the image and the template.
There are many different methods for comparing two images defined in OpenCV. The utilize method was found to work best for this sort of object detection. The formula is shown in equation 4.2.
The templates are created before every run. In order to make that process as user-friendly as possible, a GUI based configuration app is created. The templates can be seen in the table 4.1.
The main characteristics of this method are:

- This method works much better with different lighting scenarios than the HSV-threshold.
- Motion blurring results in an unstable detection of the laser point. Thus, fast movements with the robot and by hand should be avoided.
- Whenever there is no green laser point in the image, the red laser is detected as green, since this is now the best match.
- This method only works with the background (e.g. white flip board) specified in the template.

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}} \tag{4.2}$$

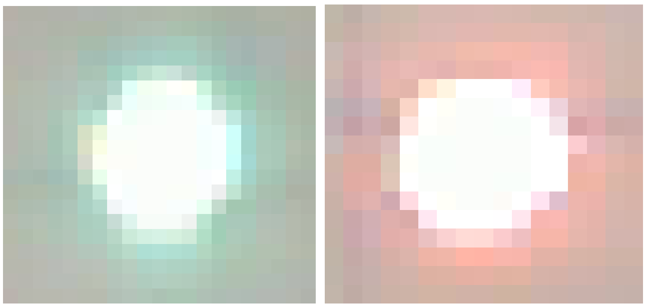


Table 4.1: laser point template as created by the configuration app

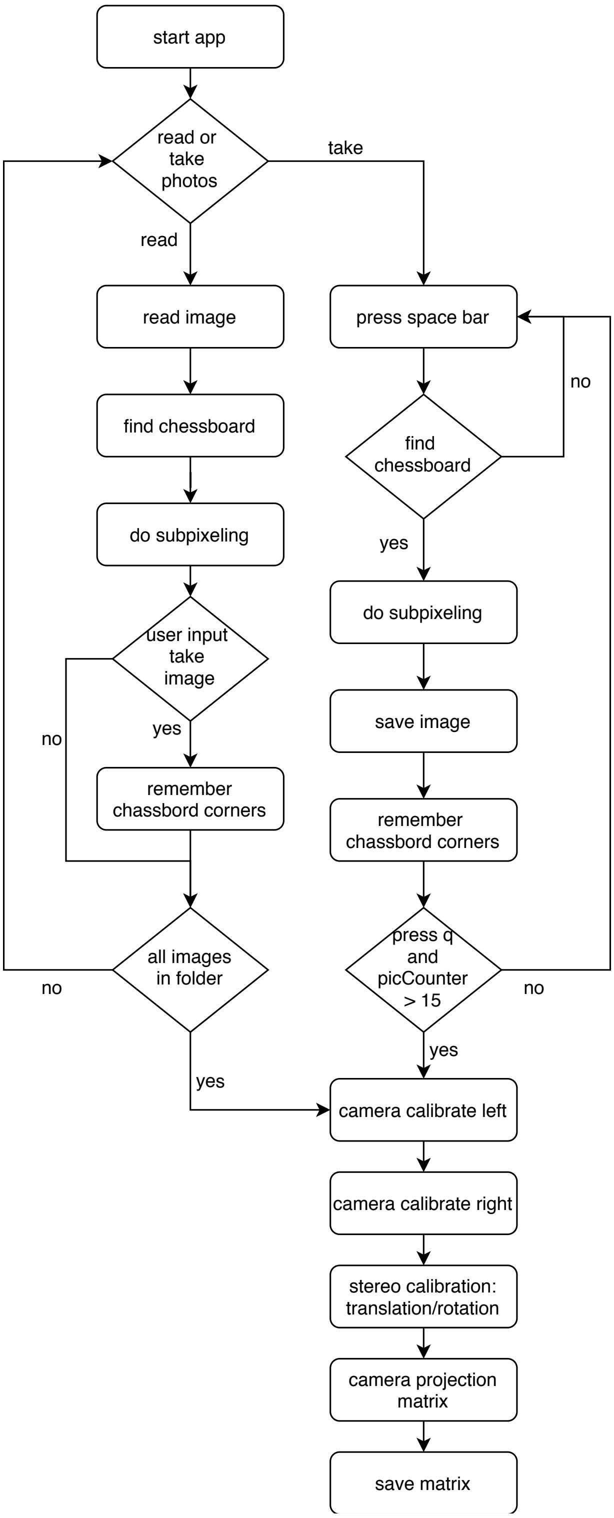


Figure 4.2: camera calibration app flowchart

5 Main App

This chapter describes the algorithmic steps taken in the main application as seen in the flowchart 5.1. The action *transform 3D point to robot base* consists of multiple steps itself:

- Multiply the 3D point with the hand-eye calibration to transform the point into the TCP frame, as seen in 4.1
- Transform the point to the robot base coordinate system by multiplying with the backwards kinematic in 3.1.

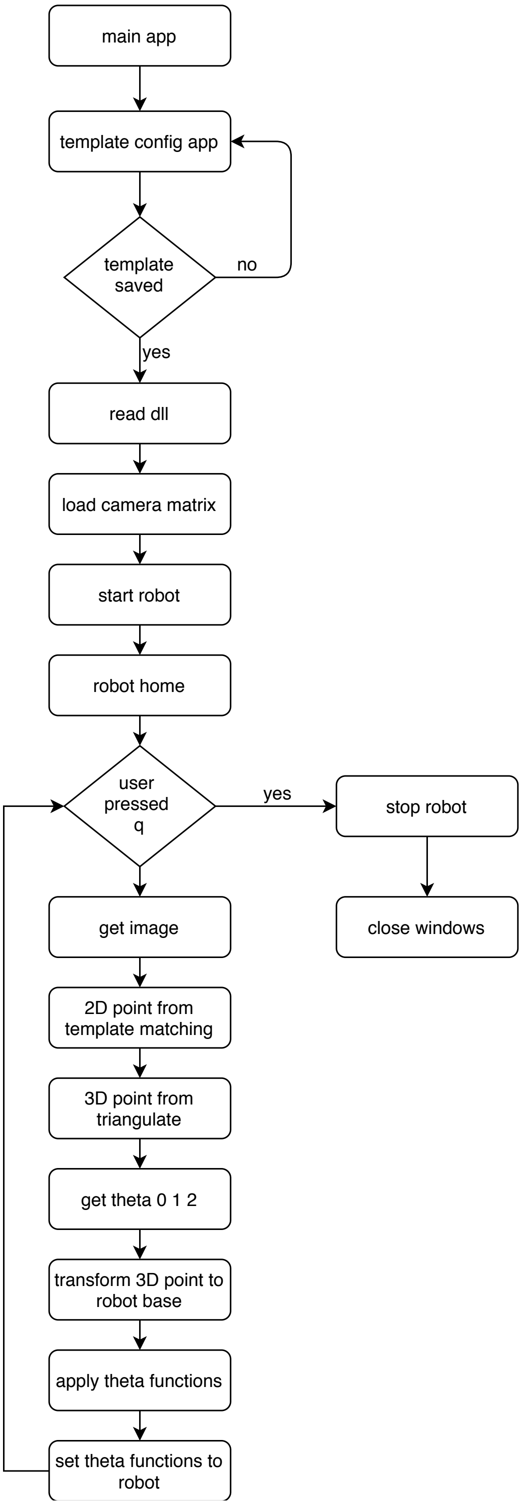


Figure 5.1: camera calibration app flowchart

6 Conclusion

We find, that template matching method is better suited to detect small objects in a constantly changing image rather than searching the images for a specific colour.

The resting distance between green and red laser point, as seen in figure 6.1 results from the offset of the laser pointer to the robot tool centre. That can be easily adjusted with a new bracket. To avoid any systematic errors, the laser pointer has to be mounted colinear with the z-axis of the robot tool.

Another current problem is the high energy consumption of the laser pointer. Within a half an hour, the intensity of the laser pointer dropped significantly. In the next step, a more appropriate power supply should be installed.

An additional option could be to specify $theta_1$ as a variable instead of a constant and find a more advanced system of equations.

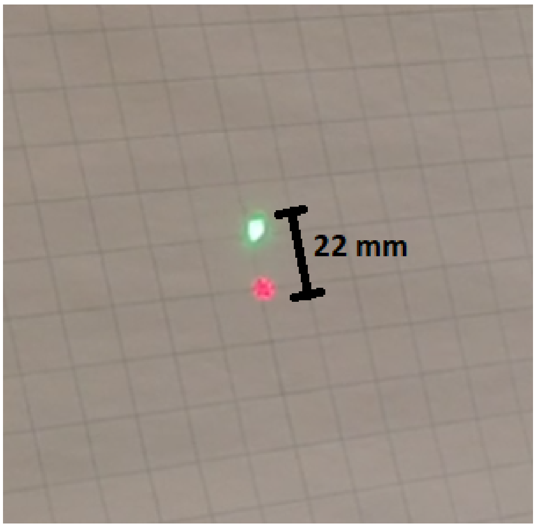


Figure 6.1: resting offset between laser points

Bibliography

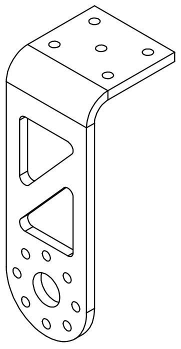
Craig, John (2018): *Introduction to Robotics Mechanics and Control* (cit. on p. 4).

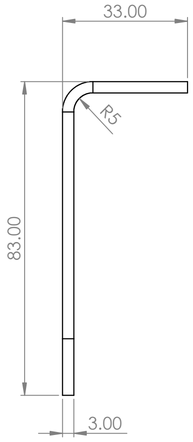
ROBOTIS-GIT (2019): *DynamixelSDK*. Robotis Ltd. GitHub. URL: `github.com/ROBOTIS-GIT/DynamixelSDK/tree/master/c` (cit. on p. 5).

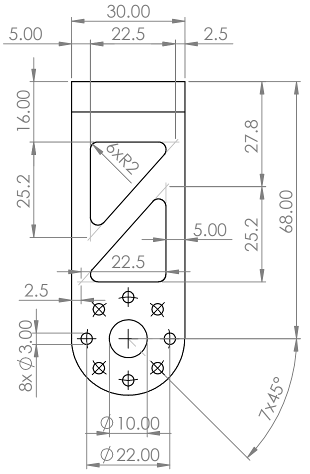
The OpenCV Reference Manual(July 2019). 4.1.1. OpenCV (cit. on p. 8).

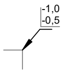
Appendices

A Drawings

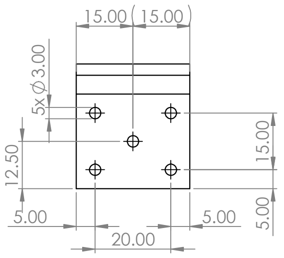


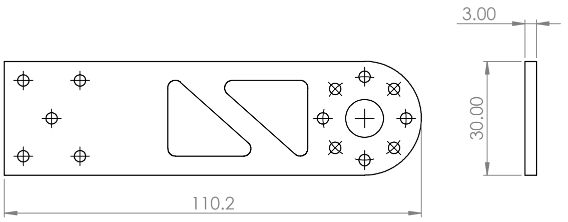






BRENNKONTUR
auf Blechmitte gerechnet





Masstoleranzen
Laserstrahlschneiden EN ISO 9013 2 2 1

Werkstoff: ALMG3 F22 DIN1783

gestreckte Länge 110,2 mm
(auf Blechmitte gerechnet)
Die gestreckte Länge ist durch
den Verarbeiter zu prüfen!

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND
BREAK SHARP
EDGES

DO NOT SCALE DRAWING

REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

TITLE:

MATERIAL:

DWG NO.

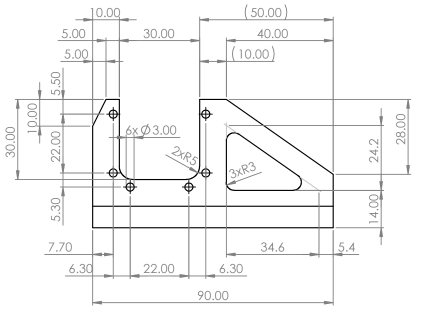
AKT L 3x30x80x30

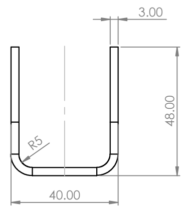
A3

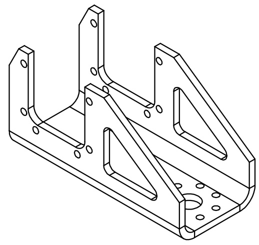
WEIGHT:

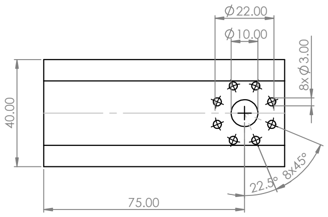
SCALE:1:1

SHEET 1 OF 1

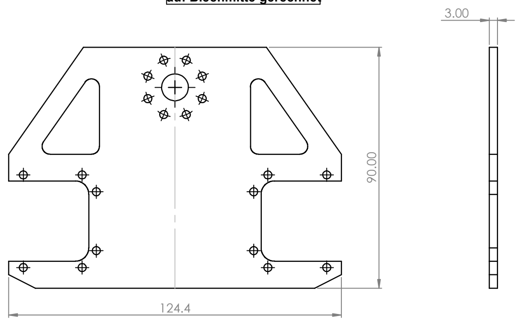








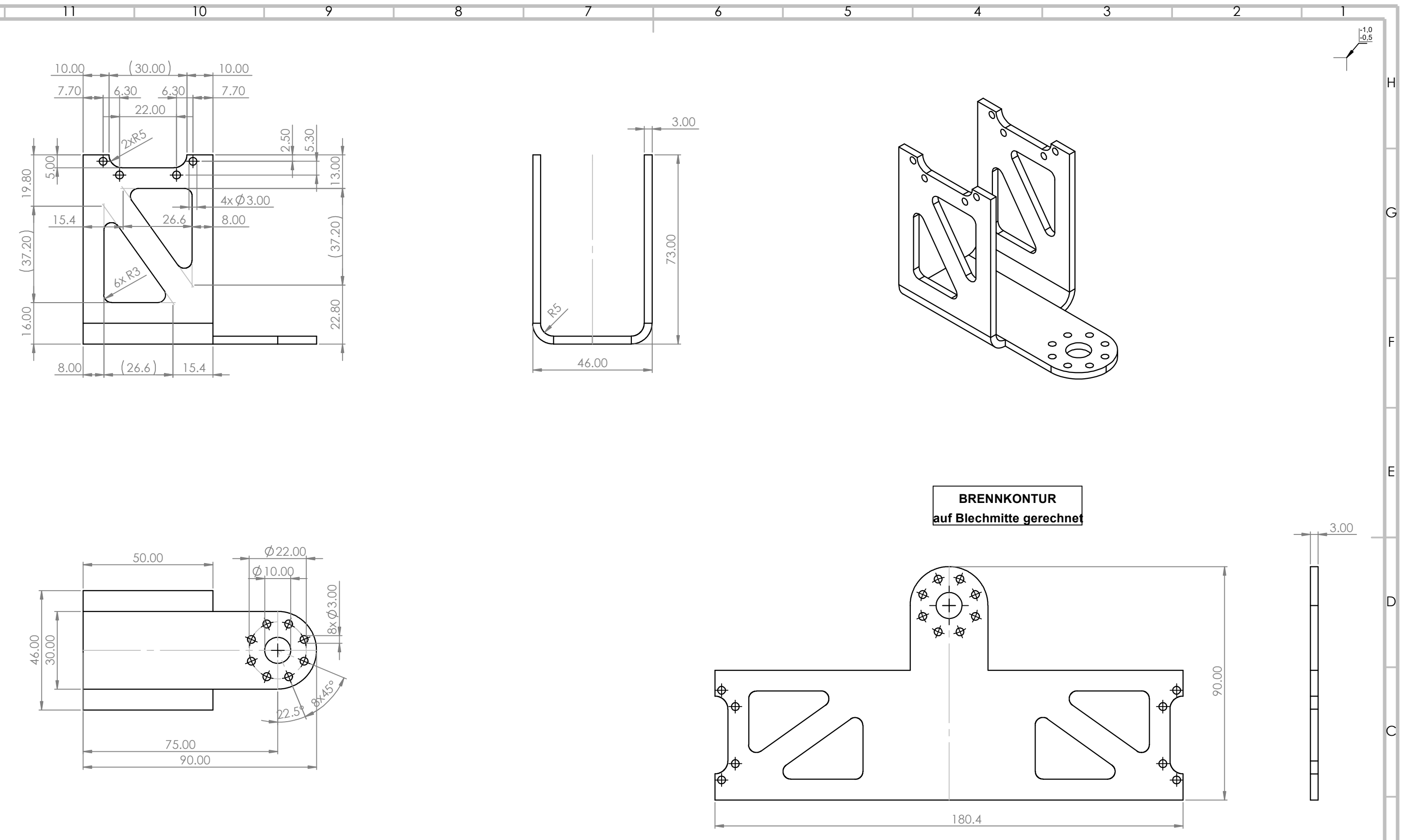
BRENNKONTUR
auf Blechmitte gerechnet



Masstoleranzen
Laserstrahlschneiden EN ISO 9013 2 2 1 Werkstoff: ALMG3 F22 DIN1783

gestreckte Länge 124,4 mm
(auf Blechmitte gerechnet)
Die gestreckte Länge ist durch
den Verarbeiter zu prüfen!

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	TITLE:	
DRAWN					
CHK'D					
APPV'D					
MFG					
Q.A			MATERIAL:	DWG NO. AKT U 3x48x40x48x90	A2
			WEIGHT:	SCALE:1:1	SHEET 1 OF 1



Masstoleranzen
Laserstrahlschneiden EN ISO 9013 2 2 1

Werkstoff: ALMG3 F22 DIN1783

gestreckte Länge 180,4 mm
(auf Blechmitte gerechnet)
Die gestreckte Länge ist durch
den Verarbeiter zu prüfen!

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION

	NAME	SIGNATURE	DATE	
DRAWN				TITLE:
CHK'D				
APPV'D				
MFG				
Q.A				

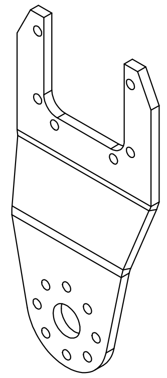
MATERIAL:

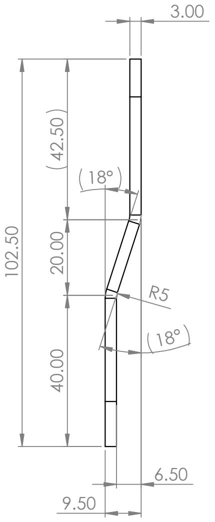
DWG NO. AKT U 3x73x46x73x90 A2

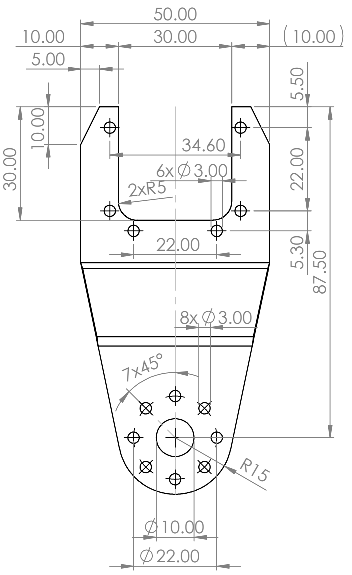
WEIGHT:

SCALE:1:1

SHEET 1 OF 1

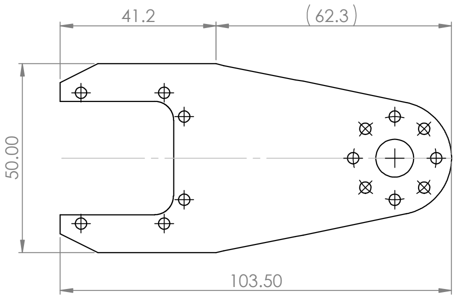


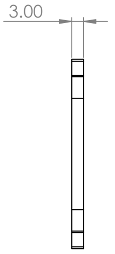




-1,0
-0,5

BRENNKONTUR
auf Blechmitte gerechnet





gestreckte Länge 103,5 mm
(auf Blechmitte gerechnet)
Die gestreckte Länge ist durch
den Verarbeiter zu prüfen!

Masstoleranzen
Laserstrahlschneiden EN ISO 9013 2 2 1

Werkstoff: ALMG3 F22 DIN1783

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE	TITLE:	
DRAWN					
CHK'D					
APPV'D					
MFG					
Q.A			MATERIAL:	DWG NO. AKT Z 3x40x20x42,5x50	A3
			WEIGHT:	SCALE:1:1	SHEET 1 OF 1

B Motor Parameters

parameter	axis 0	axis 1	axis 2
home offset	2288	2060	3074
min angle	1580	2060	1650
max angle	3000	3200	3100
max velocity	20	20	20
home	2290	2060	3072

Table B.1: table of motor parameters