Robotik WS19/20
Übung 1

Marcel Schmidt, Niklas Pauli

17. Oktober 2019

1 Aufgabe 4-2

```
height: 480
width: 640
distortion_model: "plumb_bob"
D: [0.0, 0.0, 0.0, 0.0, 0.0]
K: [383.7944641113281, 0.0, 322.3056945800781, 0.0,
383.7944641113281, 241.67051696777344, 0.0, 0.0, 1.0]
R: [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
P: [383.7944641113281, 0.0, 322.3056945800781, 0.0, 0.0,
383.7944641113281, 241.67051696777344, 0.0, 0.0, 0.0, 1.0,
0.0]
binning_x: 0
binning_y: 0
roi:
  x_offset: 0
  y_offset: 0
  height: 0
  width: 0
  do_rectify: False
---
```

$f_x = 383.7944641113281$
$f_x = 383.7944641113281$
$c_x = 322.3056945800781$
$c_y = 241.67051696777344$

$k_1 = k_2 = t_1 = t_2 = k_3 = 0$

2 Aufgabe 4-3

Wir haben eine Node geschrieben die sich an das Imagetopic subscribed und in der Callback funktion das Thresholdbild erstellt und in einer globalen Variable ablegt. In der selben Node läuft ein Publisher loop, der das Bild aus der globalen Variable liest und das Thresholdbild in einer neuen Topic published.

`https://github.com/MarcelSchmidt89/RobotikWS19/blob/master/catkin_ws_paulischmidt/src/camera_calibration/threshold_node.py`

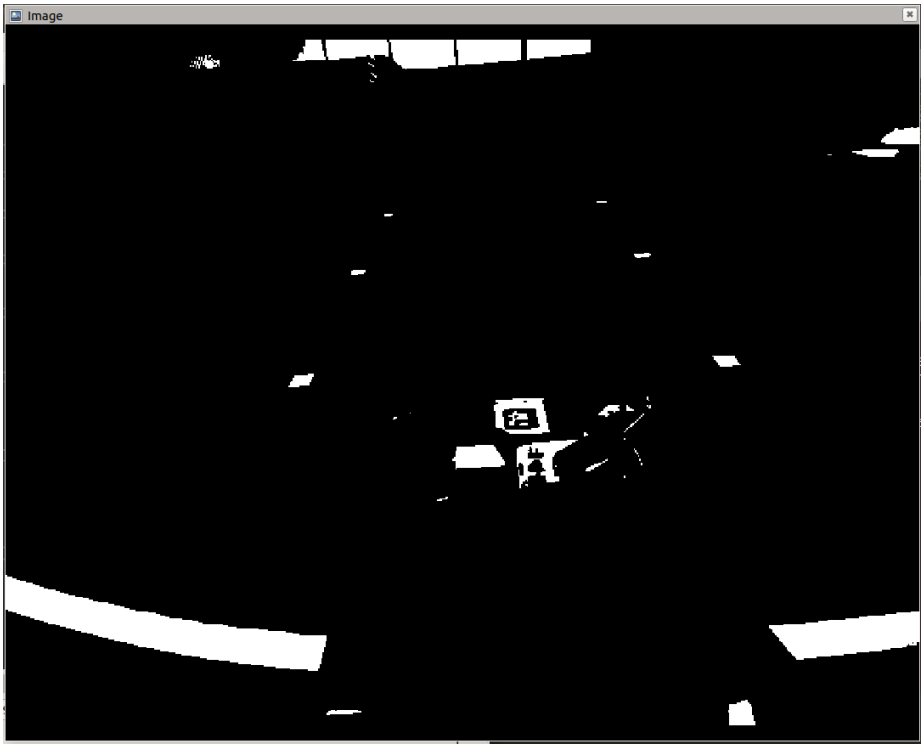

Figure 1: Threshold Bild von Topic in RVIZ

3 Aufgabe 4-4

In einer zweiten Node subscriben wir uns auf unserer Threshold Topic und lesen das Bild aus. Im callback fären wir unsere festgelegten Regionen Rot ein. Wir lesen alle Weißen Bildpunkte in den Regionen aus, berechnen den Durchschnitt der Pixelpositionen und färben die Grün ein. Das markiete Bild wird zur Überprüfung an eine neue Topic gepublished.

https://github.com/MarcelSchmidt89/RobotikWS19/blob/master/catkin_ws_paulischmidt/src/camera_calibration/marker_detection.py

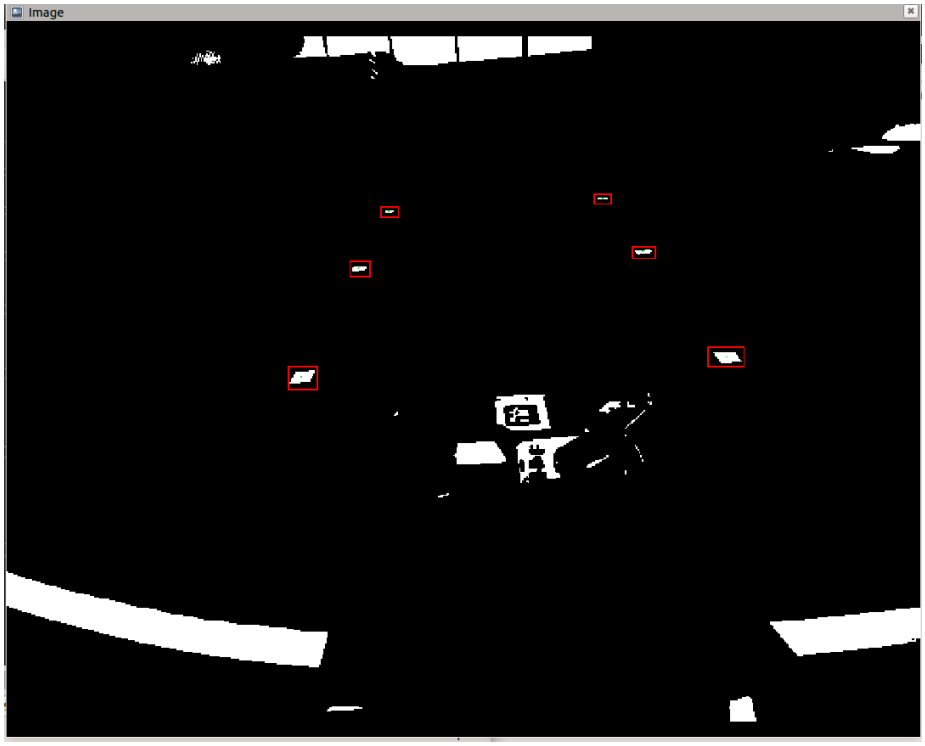

Figure 2: Threshold Bild mit Regionen und Mittelpunkten

4 Aufgabe 4-5

Wir haben unsere marker detection Node erweitert, so dass die ermittelten Punkte in solvePnP übergeben werden und die ermittelten Vektoren in der Console ausgegeben werden.

```
('Translationsvector: ', array([[ 0.03167831],
       [ 0.25024891],
       [ 0.09192414]]))
('Rotatiosnvector: ', array([[ 1.57214964],
       [-1.60328764],
       [ 0.82309567]]))
```

Figure 3: Konsolenausgabe von solvePnP

5 Aufgabe 4-6

Wir übergeben den Rotationsvector von solvePnP an die Rodriguesfunktion und ermitteln so die Rotationsmatrix.

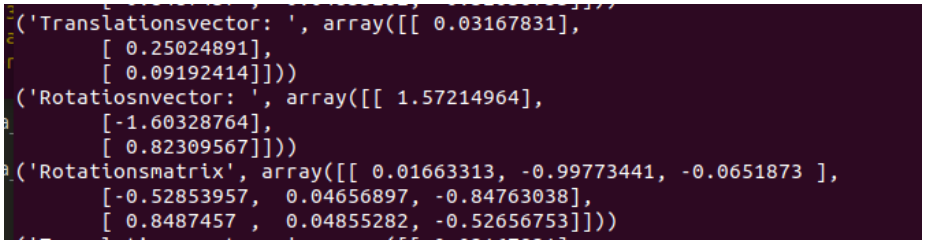

Figure 4: Konsolenausgabe von solvePnP mit Rotationsmatrix

Mit Rotationsmatrix und Translationsvektor ergibt sich folgende homogene Transformationsmatrix:

$$\begin{pmatrix} 0.01663313 & -0.99773441 & -0.0651873 & 0.03167831 \\ -0.52853957 & 0.04656897 & -0.084763038 & 0.25024891 \\ 0.8487457 & 0.04855282 & -0.52656753 & 0.09192414 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die resultierende Transformation befindet sich im lokalen Frame des baselinks (Mittelpunkt der Hinterachse), die die Ursprünglichen Koordinaten der Marker in referenz zu diesem Punkt angegeben waren.

Die Inverse Matirx ist :

$$\begin{pmatrix} -0,057747 & -1,495709 & 0,247917 & 0,353339 \\ -0,991181 & 0,131785 & 0,101491 & -0,01091 \\ -0,184473 & -2,398702 & -1,490129 & 0,743095 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Sie gibt die Transformation vom Koordinatensystem des baselinks in das Koordinatensystem der Kamera an.