Beschreibung des Algorithmus für Kamerakalibrierung

Ref.: https://docs.opencv.org/4.5.2/d9/dab/tutorial_homography.html Demo3

Input:

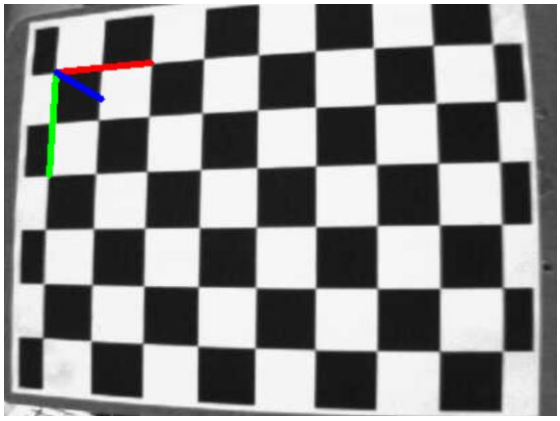
- Originalbild (640*480) als Eingangsbild und dessen Auflösung;
- Zielauflösung des Zielbildes;
- Mustergröße des Schachbretts, bzw. die in zwei Dimensionen (Breite, Höhe) dargestellten inneren Ecken;
- Breite / Höhe des Quadrats vom Schachbrett;
- Intrinsische Parameter von Kamera (Linke Kameramatrix K und Verzerrungskoeffiziente)
- Kamerahöhe, bzw. die Höhe der virtuellen Kamera, h_c;

Output:

- Projektive homographische Matrix ${}^{2}_{1}H$;
- Verhältnis zwischen Pixelanzahl und Einheit Meter und umgekehrt;
- Zielbild (zur Validation).

Prozess:

1. Finde die inneren Ecken (mithilfe von *cv::findChessboardCorners()*, basierend auf Originalbild und die vorgegebene Mustergröße des Schachbretts), die in inhomogenen Bildmatrixkoordinate mit Pixel dargestellt werden;
2. Berechne die entsprechenden inneren Ecken (basierend auf die vorgegebene Mustergröße des Schachbretts und Breite / Höhe des Quadrats vom Schachbrett), die in homogenen Objektkoordinate, bzw. Schachbrettkoordinate mit Millimeter dargestellt werden (Der Ursprung der Objektkoordinatensystem fällt folgendermaßen mit der inneren Ecke oben links zusammen.);



3. Schätze die Kameraposition in Bezug auf dem Objekt (Schachbrett), bzw. die extrinsische Parameter zwischen Kamera und Schachbrett (mithilfe von *cv::solvePnP()*, basierend auf die oben beiden entsprechenden Gruppen der inneren Ecken sowie die intrinsischen Parameter der Kamera), die folgendermaßen in ${}^{c1}_{o}R$ (Rodrigues Rotationsmatrix) und ${}^{c1}_{o}t$(Translationsvektor) dargestellt werden.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = {}^{c}\mathrm{M}_o \begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} {}^{c}\mathrm{R}_o & {}^{c}\mathrm{t}_o \\ 0_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix},$$

wobei

$${}^{c1}_{o}R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$${}^{c1}_{o}t = [t_x \quad t_y \quad t_z]^T.$$

4. Definiere der zweiten virtuellen Position der Kamera in Bezug auf Schachbrett durch ${}^{c2}_{o}R$ und ${}^{c2}_{o}t$, die folgendermaßen sind:

$${}^{c2}_{o}R = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^{c2}_{o}t = h_c * {}^{c1}_{o}t$$

, sodass die virtuelle Kamera sich gerade nach unten orientiert und mit h_c-fache Höhe in Vergleich zur originalen Kamera (auf dem Modellauto montiert) liegt. Der Grund, warum die ersten zwei Hauptelemente -1 betragen, liegt darin, dass die x- und y-Richtung des Schachbrettkoordinatensystems jeweils nach -x- und -y- Richtung des Egokoordinatensystems vom Auto wenn die inneren Ecken wie in Schritt 2 berechnet werden, soll das erwünschte Bildmatrixkoordinatensystem die x- und y-Richtung jedoch

gleich wie die von Egokoordinatensystem vom Auto besitzen.

5. Basierend auf ${}^{c1}_{o}R$, ${}^{c1}_{o}t$, ${}^{c2}_{o}R$ und ${}^{c2}_{o}t$, können wir die Rotationsmatrix und Translationsmatrix zwischen dem originalen Kamerakoordinatensystem und dem virtuellen berechnen, wie folgende:

$$ {}^{2}_{1}R = {}^{c2}_{o}R \cdot {}^{c1}_{o}R^{T} $$
$$ {}^{2}_{1}t = {}^{c2}_{o}R \cdot (-{}^{c1}_{o}R^{T} \cdot {}^{c1}_{o}t) + {}^{c2}_{o}t $$

6. Daraus können wir die Euklidische Homografie H_e wie folgende berechnen:

$$ {}^{2}_{1}H_e = {}^{c2}_{o}R + \frac{{}^{c1}_{o}t \cdot n^{T}}{d}, $$

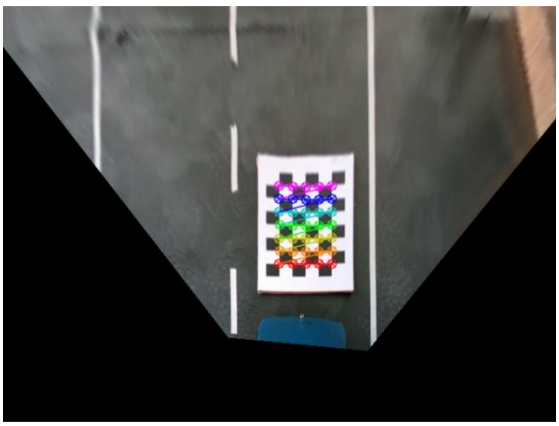
wobei n^T der transponierte Normalvektor des Schachbretts relativ zum originalen Kamerakoordinatensystem ist und d die Entfernung zwischen Ursprung des originalen Kamerakoordinatensystems und Schachbrett ist.

7. Daraus können wir die projektive homographische Matrix mithilfe der Linke Kameramatrix K berechnen:

$$ {}^{2}_{1}H = \gamma \, K \, {}^{2}_{1}H_e \, K^{-1}, $$

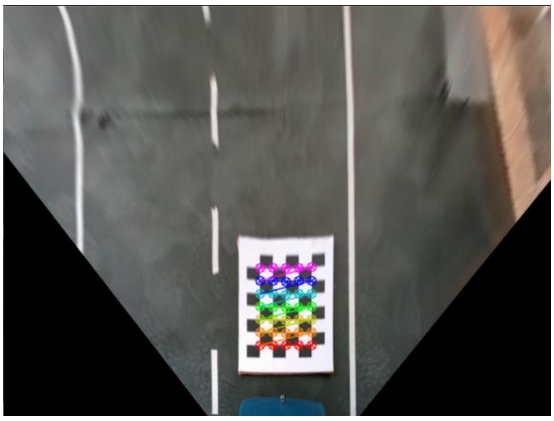
wobei die Parameter γ als 1 genommen werden kann. Danach ist deren Normalisierung notwendig.

Bisher ist die Berechnung die projektive homographische Matrix ohne Änderung der Auflösung abgeschlossen. Das entsprechende transformierte Bild durch ${}^{2}_{1}H$ wird folgendermaßen dargestellt.



8. Aber um mehr Sichtweite zu gewinnen sowie Ausrichtung des Mittelpunktes von den unteren beiden transformierten Bildecken nach der Mittellinie des transformierten Bildes in der horizontalen Richtung (notwendig da der Algorithmus des „Probabilistic Tracking“ setzt die Entfernung der Fahrbahnmarkierungspunkte von der Mittelline des transformierten Bildes als ein Kriterium für Zuordnung des Typs der Fahrbahnmarkierung), wird eine Translationsmatrix benötigt, die wie folgende aussieht.

$$ Translation = \begin{bmatrix} 1 & 0 & x_{off} \\ 0 & 1 & y_{off} \\ 0 & 0 & 1 \end{bmatrix} $$



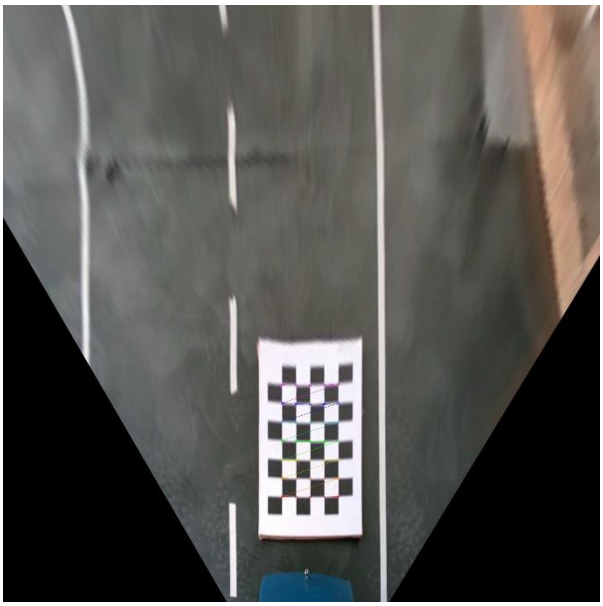
Des Weiteren wird eine Skalierungsmatrix benötigt, um das schon transformierten Bild so zu verändern, dass dessen Auflösung von 640*480 zu 3000*3000 wird, wie angefordert von dem Algorithmus des „Probabilistic Tracking“. Die Skalierungsmatrix sieht wie folgende aus:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Endlich, die resultierende projektive homographische Matrix mit Berücksichtigung der oben beiden Matrizen sieht so aus:

$$^{2}_{1}H = S * Translation * {}^{2}_{1}H$$

Dementsprechend sieht das transformierte Bild mit der Auflösung 3000*3000 so aus:



9. Zur Gewinnung des Verhältnisses zwischen Pixelanzahl und Einheit Meter, kann die durch $^{2}_{1}H$ transformiert inneren Ecken (z.B. den Pixelabstand zwischen den Ecken unten links und unten rechts) mithilfe von *cv::perspectiveTransform()* und die bekannte Breite / Höhe des Quadrats vom Schachbrett (z.B. den entsprechenden Abstand) ausgenutzt.