SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

KALIBRÁCIA KAMERY

ZADANIE

2024 **Bc. Maroš Kocúr, Bc. Eduard Zelenay**

Obsah

Zoznam obrázkov a tabuliek

Úvod

Cieľom zadania je naučiť sa pracovať s kamerou, nakalibrovať ju, pracovať s OpenCV a orientovať sa v dokumentáciach. Ďalšou časťou zadania je využiť už existujúcu implementáciu Houghovej transformácie na detegovanie kruhov.

1 Úlohy

1. Pripravte sa na zadanie a dohľadajte potrebné postupy pre realizáciu zadania v zmysle úloh nižšie na stránke dokumentácie k OpenCV. Prezentujte vami nájdené postupy a riešenie

2. Realizujte kalibráciu kamery pomocou šachovnice a zistite jej vnútorné parametre f_x, f_y, c_x, c_y

3. Využite už existujúcu implementáciu na stránke OpenCv pre Houghovu transformáciu na detekciu kružníc a vytvorte program ktorý bude vediet' detegovat' a kruhom označo- vat' kružnice na stene v miestnosti D 618 (Pri odovzdávaní môže byt' zadanie testované aj na iných kruhoch!)

4. Urobte si poriadok na vašom GIT repozitári aby každé zadanie bolo v samostatnom priečinku s názvom "Zadanie_<čislozadania>" a všetky údaje ste mali v branch master

5. Zdokumentujte svoj postup a dosiahnuté výsledky. Dokumentáciu k zadaniu odovzdávajte v PDF

2 Kalibrácia kamery

Kamera sníma obraz, ale prináša skreslenie výsledného obrazu, kvôli vlastnostiam šošovky, ktoré nám kalibrácia pomáha porozumieť a následne vďaka nim obraz upraviť.

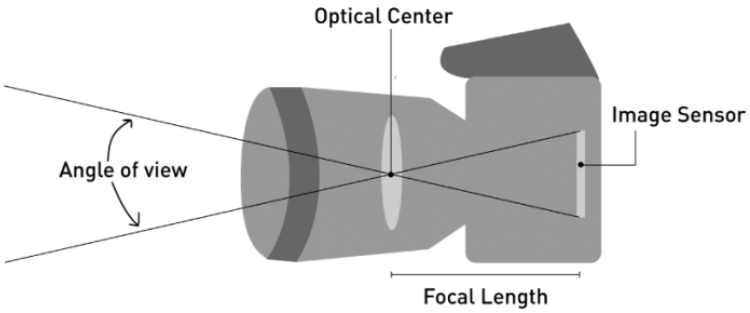
2.1 Proces kalibrácie

Pomocou kamery sme nafotili šachovnicu, ktorú sme analyzovali pomocou kódu v Pythone. Na fotkách šachovnice sa ľahko vníma skreslenie, ktoré šošovka prináša do obrazu.

Obr. 2.1: Fotka z kamery

Na fotke 2.1 môžeme pozorovať, že v realite má šachovnica rovné čiary, no kamera nám vniesla skreslenie a čiary sa javia ako krivky.

Pomocou kalibrácie vieme vypočítať vnútorné parametre kamery f_x, f_y, c_x, c_y, kde f_x, f_y je ohnisková vzdialenosť a c_x, c_y optický stred.



Obr. 2.2: Parametre kamery

Na obrázku 2.2 sú vyznačené parametre kamery, kde ohnisková vzdialenosť je vzdialenosť

od optického stredu ku senzoru a optický stred je kde sa reálny obraz otáča a otočený dopadá na senzor. Na odstránenie skreslenia potrebujeme taktiež parametre skreslenia k_1, k_2, p_1, p_2, k_3.

Na kalibráciu sme použili šachovnicu o veľkosti 8x6 a funkcie z knižnice OpenCV ktorá hľadá vnútorné rohy, takže v kóde sme zadefinovali, 7x5.

2.2 Vysvetlenie kódu procesu kalibrácie

Na kalibráciu kamery potrebujeme aspoň 10 obrázkov, aby sme získali lepšie výsledky.

```
img = cv.imread(fname)
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

Načítame odfotený obrázok a z farebného obrázku spravíme monochromatický.

```
ret, corners = cv.findChessboardCorners(gray, (7,5), None)
```

Pomocou príkazu nájdeme rohy šachovnice, kde vstupom je monochromatický obrázok a počet vnútorných rohov, pomocou posledného vstupu si môžeme urýchliť kód, napríklad ak niesu nájdene rohy tak vráti none. Výstupom príkazu je premenná, ktorá značí to, či boli rohy nájdené a ak áno, sú jej súčasťou aj dané rohy.

```
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER,
            30, 0.001)
corners2 = cv.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
```

Príkaz zlepšuje precíznosť lokácie rohov na subpixeli, je to iteračná funkcia. Vstupmi sú monochromatický obrázok, nájdene rohy, veľkosť prehľadávaného okna, polovica mŕtvej zóny, ktorá sa používa aby sa zabránilo možným singularitám autokorelačnej matice (-1,-1) značí že taká zóna neexistuje, kritéria na zastavenie iterácii. Kritéria sú buď 30 iterácii alebo ϵ je menší než 0.001.

```
_, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints,
                                                gray.shape[::-1], None, None)
h,  w = img.shape[:2]
newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx, dist, (w,h),
                                                 1, (w,h))
dst = cv.undistort(img, mtx, dist, None, newcameramtx)
```

Prvá funkcia slúži na získanie parametrov kamery. Kde vstupmi sú matice 3D body v reálnom svete, 2D body na obrázku(naše rohy), veľkosť obrázka. Výstupom je matica s parametrami kamery a parametre skreslenia. Parametre sa použijú na získanie novej optimálnejšej matice kamery. Zavolaním poslednej funkcie získame už upravený obrázok bez skreslenia. [1]

2.3 Výsledky

Po kalibrácii sme získali maticu parametrov kamery:

$$\text{mtx} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 418.80290929 & 0 & 155.93710829 \\ 0 & 503.26200421 & 127.3783637 \\ 0 & 0 & 1 \end{bmatrix}$$

Obr. 2.3: Výsledok kalibrácie

Obrázok 2.3 je ten istý čo 2.1, no aplikovali sme na neho algoritmus opísaný v tejto sekcii 2.2 a podarilo sa nám vyrovnať hrany šachovnice.

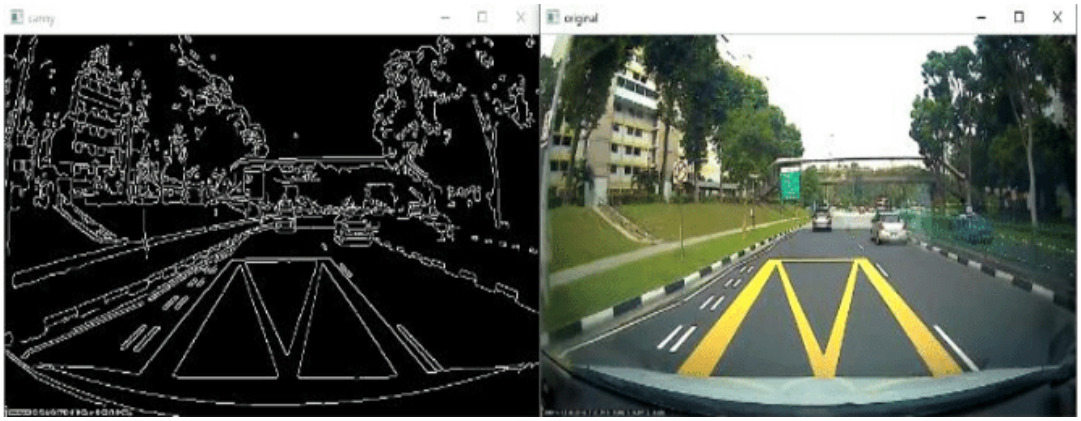
3 Detekcia kruhov

Na detekciu kruhov sme vytvorili program v Pythone, ktorý využíval hotovú implementáciu Houghovej transformácie zo stránky OpenCV.

3.1 Houghova transformácia

Houghova transformácia je technika spracovania obrazu využívaná na detekciu tvarov akými sú priamky, kružnice, elipsy a tak ďalej. Transformuje priestorovú informáciu z obrazu do parametrického priestoru, z ktorého sa získavajú kandidáti objektov ako lokálne maximá v akumuluačnom priestore, ktorý je vytvorený algoritmom pre výpočet Houghovej transformácie.

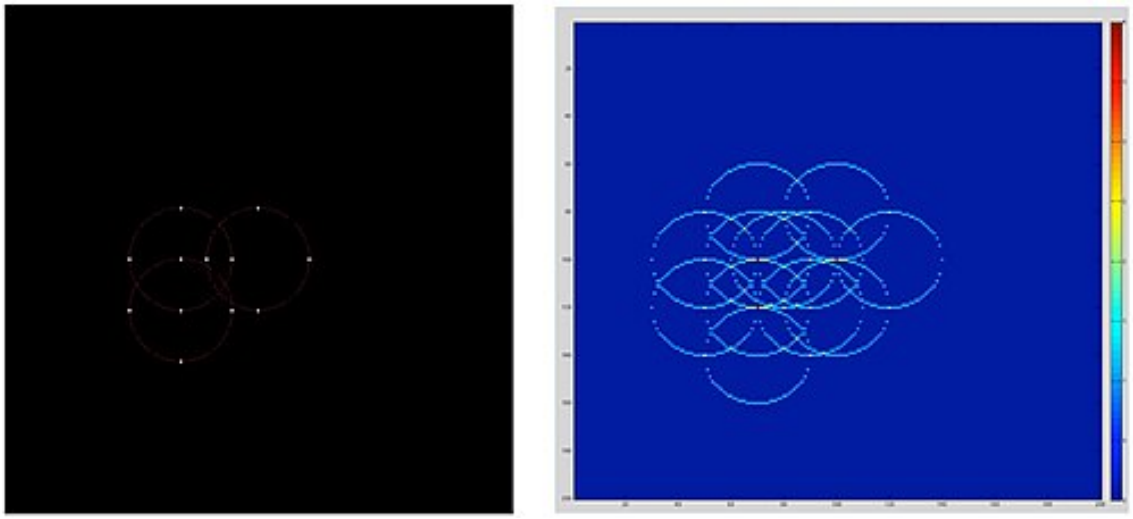
Predtým, ako môžeme začať detegovať kruhy je potrebné si obrázok predspracovať. Na obrázok sa použije Gaussov filter, ktorý vytvorí efekt rozmazania, ďalej sa obrázok prevedie na monochromatický a následne sa naň aplikuje Cannyho hranový detektor, vďaka ktorému získame hrany, ktoré využijeme na detekciu. Príklad detekcie hrán môžeme vidieť na 3.6.



Obr. 3.1: Cannyho detekcia hrán (naľavo) reálny obrázok (napravo)

Akumulačný priestor, alebo akumulátor si môžeme predstaviť ako maticu, ktorej počet rozmerov bunky matice sa rovná počtu hľadaných parametrov. Ak by sme poznali polomer detegovaného kruhu, tak by sme hľadali dva parametre, avšak, v našom prípade chceme byť schopní detegovať kruhy rôznych veľkostí a tak potrebujeme nájsť tri parametre popisujúce hľadaný kruh. Vychádzame teda z rovnice popisujúcej kružnicu: $(x-a)^2 + (y-b)^2 = r^2$, kde a, b popisujú stred kružnice a r je polomerom kružnice.

Pre každý pixel, ktorý je súčasťou detegovanej hrany sa vytvoria kruhy so stredmi a polomermi z prehľadávaného priestoru a každej bunke z akumulačnej matice sa pridá hlas, ak ňou kruh prechádza. Tento proces sa nazýva "voľby". Po voľbách vieme v akumulačnej matici nájsť lokálne maximá. Tie prislúchajú stredom kruhov v reálnom priestore



Obr. 3.2: Reálne kruhy (vľavo) Akumulačná matica (vpravo)

Na 3.2 napravo môžeme vidieť akumulačnú maticu, ktorej lokálne maximá (najhustejšie priesečníky) reprezentujú stredy kruhov v reálnom obrázku, ktoré môžeme vidieť naľavo.

3.2 Vysvetlenie kódu detekcie kruhov

V nekonečnej slučke sme zaznamenávali obrázky z kamery.

```
cam.get_image(image)
gimg = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
gimg = cv.GaussianBlur(gimg,(5,5),0)
```

Následne sme obrázky previedli na monochromatické a pomocou Gaussovho filtra sme ich rozmazali.

```
circles = cv.HoughCircles(gimg,cv.HOUGH_GRADIENT,1,20,
                          param1=cv.getTrackbarPos('P1','image'),
                          param2=cv.getTrackbarPos('P2','image'),
                          minRadius=cv.getTrackbarPos('minR','image'),
                          maxRadius=cv.getTrackbarPos('maxR','image'))
```

Na nájdenie kruhov sme využili funkciu HoughCircles, ktorá je implementovaná v knižnici OpenCV. Do nej sme vložili predspracovaný upravený obrázok, metódu, ktorú chceme použiť, rozlíšenie akumulačnej matice, minimálnu vzdialenosť medzi stredmi kruhov, vstupný parameter pre Cannyho detekciu hrán, parameter prahu citlivosti detekcie kruhov, minimálny a maximálny polomer kruhu.

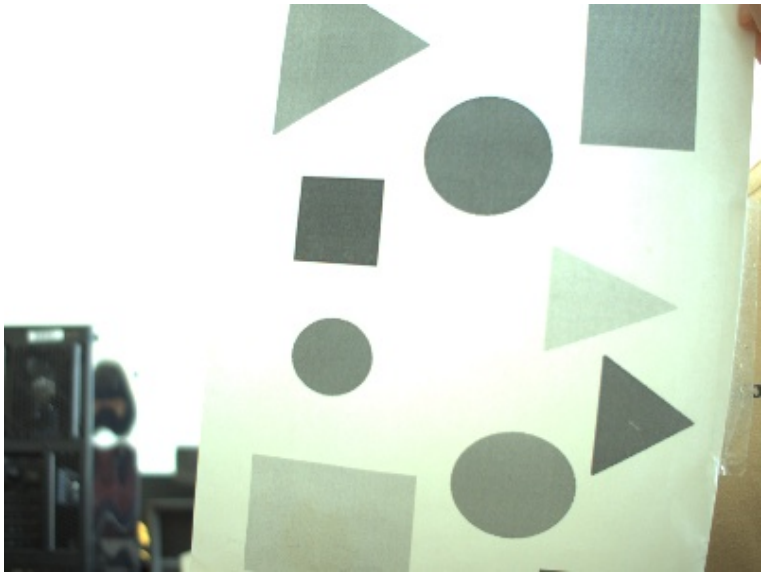
```
if circles is not None:
            circles = np.uint16(np.around(circles))
            for i in circles[0,:]:
        # draw the outer circle
                cv.circle(image,(i[0],i[1]),i[2],(0,255,0),2)
        # draw the center of the circle
                cv.circle(image,(i[0],i[1]),2,(0,0,255),3)
```

Do matice circles sa nám uložili hodnoty parametrov kružníc a teda ich stredy a polomery. Na to, aby sme boli schopní rýchlejšie odladiť, respektíve pochopiť Houghovu transformáciu sme parametre načítavali z trackbarov.

```
canny = cv.Canny(gimg,
threshold1=cv.getTrackbarPos('P1','image'),
threshold2=cv.getTrackbarPos('P1','image')*2)
```

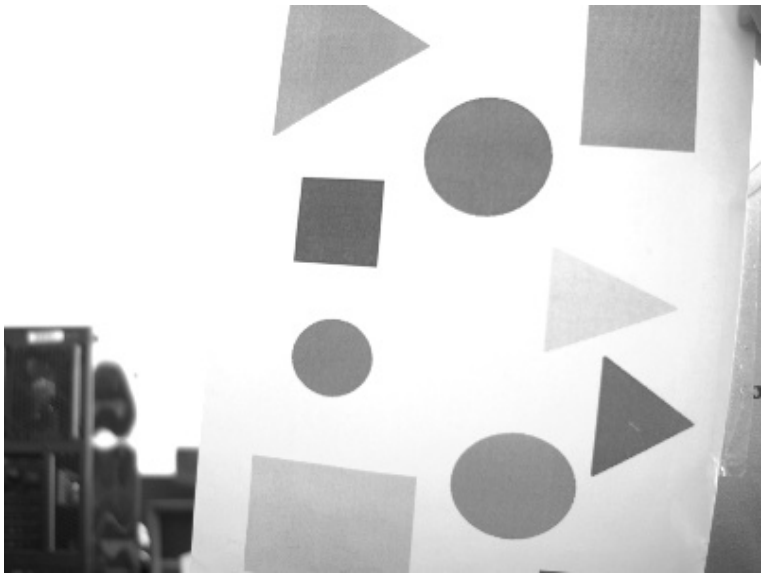
Taktiež sme si pre lepšie pochopenie procesu vykreslovali Cannyho detekciu hrán, aby sme videli, aký dopad na ňu má parameter, ktorý posúvame aj do funkcie Houghovej transformácie.[2]

3.3 Výsledky



Obr. 3.3: Pôvodná fotka

Na 3.3 môžeme vidieť pôvodnú fotku z kamery.

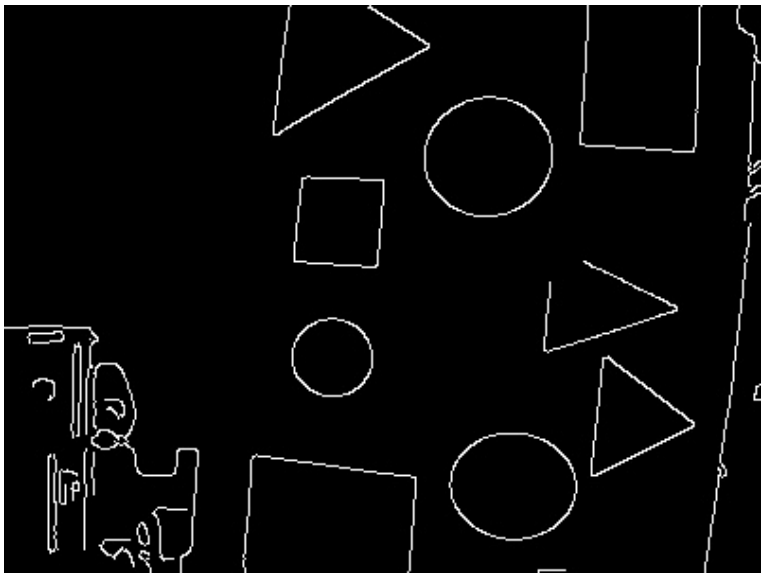


Obr. 3.4: Monochromatický obrázok z originálu

Následne sme z nej vytvorili monochromatický obrázok, ktorý môžeme vidieť na 3.4

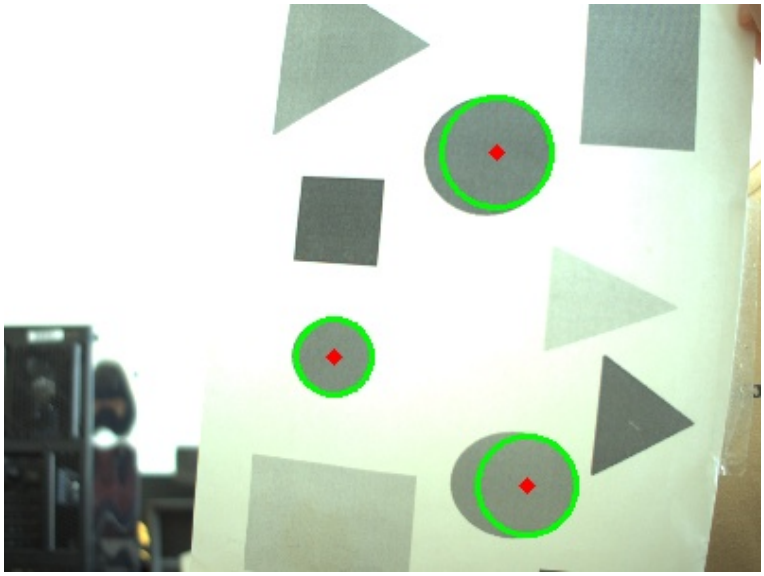
Obr. 3.5: Obrázok rozmazaný pomocou Gaussovho filtra

Potom sme na ňu aplikovali Gaussov filter, ktorý vytvoril rozmazaný obrázok viditeľný na 3.5



Obr. 3.6: Cannyho detektor hrán

Následne sme na obrázok aplikovali Cannyho detektor hrán, ktorý sme si vykreslili z dôvodu, aby sme lepšie pochopili, ako Houghova transformácia funguje. Tú môžeme vidieť na 3.6



Obr. 3.7: Detegované kruhy na originálnom obrázku

Do funkcie z OpenCV pre nájdenie kruhov sme vložili rozmazaný monochromatický obrázok, teda ten istý, ktorý sme vkladali do Cannyho detektora, keďže v implementácii funkcie je Cannyho detektor už využitý. Obrázok s nájdenými kruhmi môžeme vidieť na 3.7

Zoznam použitej literatúry

1. OPENCV. Camera Calibration. In: Open Source Computer Vision, 1-3-2024. Dostupné tiež z: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html.
2. OPENCV. Hough Circle Transform. In: Open Source Computer Vision, 4-3-2024. Dostupné tiež z: https://docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html.