Látórendszer

Gépi látás alapú mérőrendszer fejlesztése mobil robotokhoz

Gépész Zoltán

HRHDWX

Konzulens: Kiss Domokos, Csorba Kristóf

0. Bevezetés

Az önálló laboratóriumi munkám célja egy gépi látás alapú mérőrendszer fejlesztése volt mobil robotokhoz.

A rendszerrel szemben támasztott követelmények a következőek:

- Egyidejűleg több robot pozíciójának és orientációjának valósidejű meghatározása a robotok tetején elhelyezett azonosító mintázatok alapján
- Egyszerű alakzatokból (pl. dobozokból) felépített környezet felülnézeti modelljének megalkotása
- Kommunikációs protokoll kialakítása a mérőrendszer és a robotok között a fenti mért adatok továbbításához
- A mérőrendszer csatlakoztatása egy robotszimulációs környezethez
    - Virtuális robotok mozgatása a mért pozíciók alapján
    - Virtuális akadályok ábrázolása a felülnézeti környezetmodell alapján

A feladat megoldásához rendelkezésemre állt egy FireWire interfésszel rendelkező ipari kamera (DFK 21AF04, 640*480 felbontással, 30 FPS színes módban), mely elhelyezésre került egy labor plafonján, a padlóra merőlegesen tájolt objektívvel.

A robotszimulációs környezet adott volt a V-REP (Virtual Robot Experimentation Platform) program képében. A V-REP felé történő integrációt előző féléves munkám során már megoldottam.

A képfeldolgozási és kamerakezelési feladatok elvégzéséhez az OpenCV (Open Source Computer Vision) szoftverkönyvtárat hívtam segítségül.

A félév során megismertem az OpenCV struktúráját, alapvető használatát és a szükségesnek vélt komponenseinek használatával elkezdtem a rendszer implementációját.

A továbbiakban röviden ismertetem az általam elvégzett munkát.

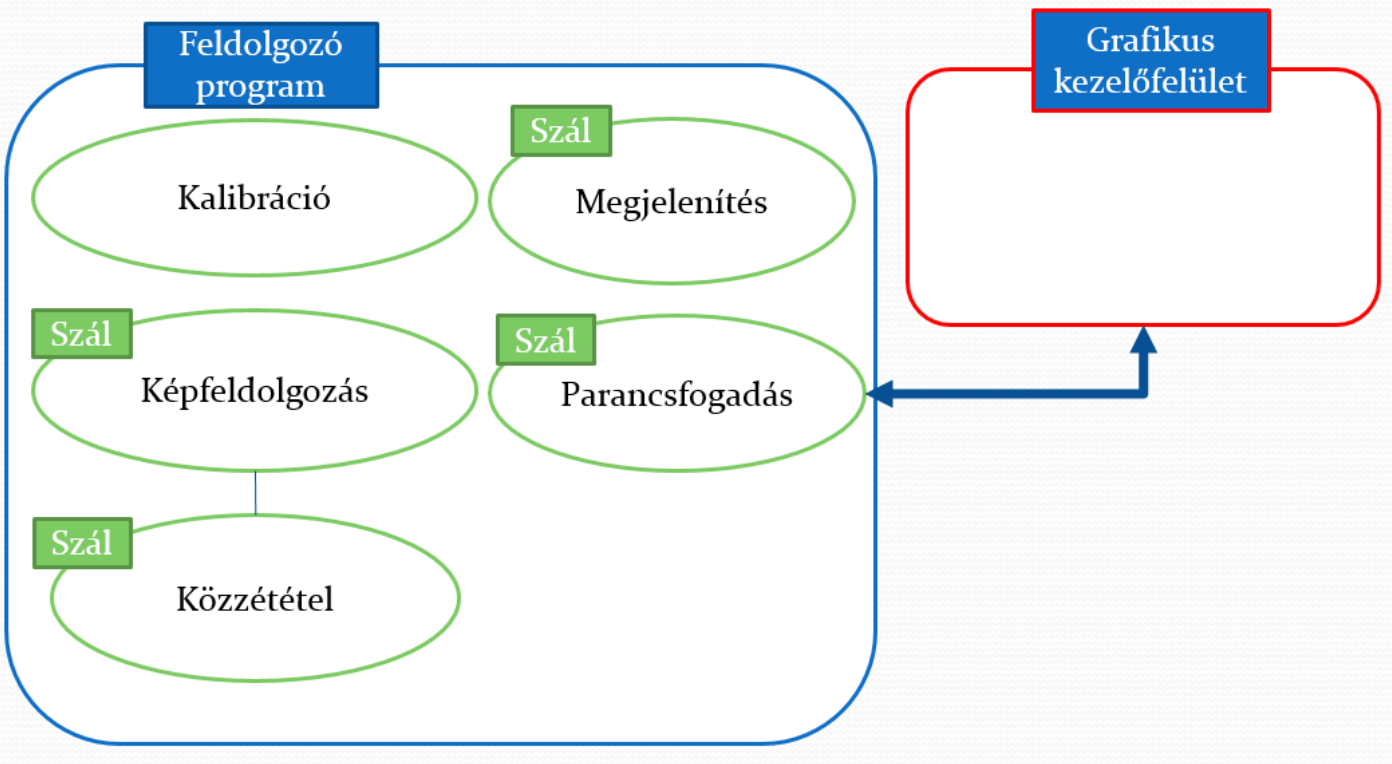
1. Bevezetés

A program az OpenCV 2.4.10 verzióját használja. Ennek installációja és konfigurációja már megtörtént a QBF120-ban található Labor120 nevű ☺ Linuxos gépen.

A források fordításához a Readme.md fájlban található segítség.

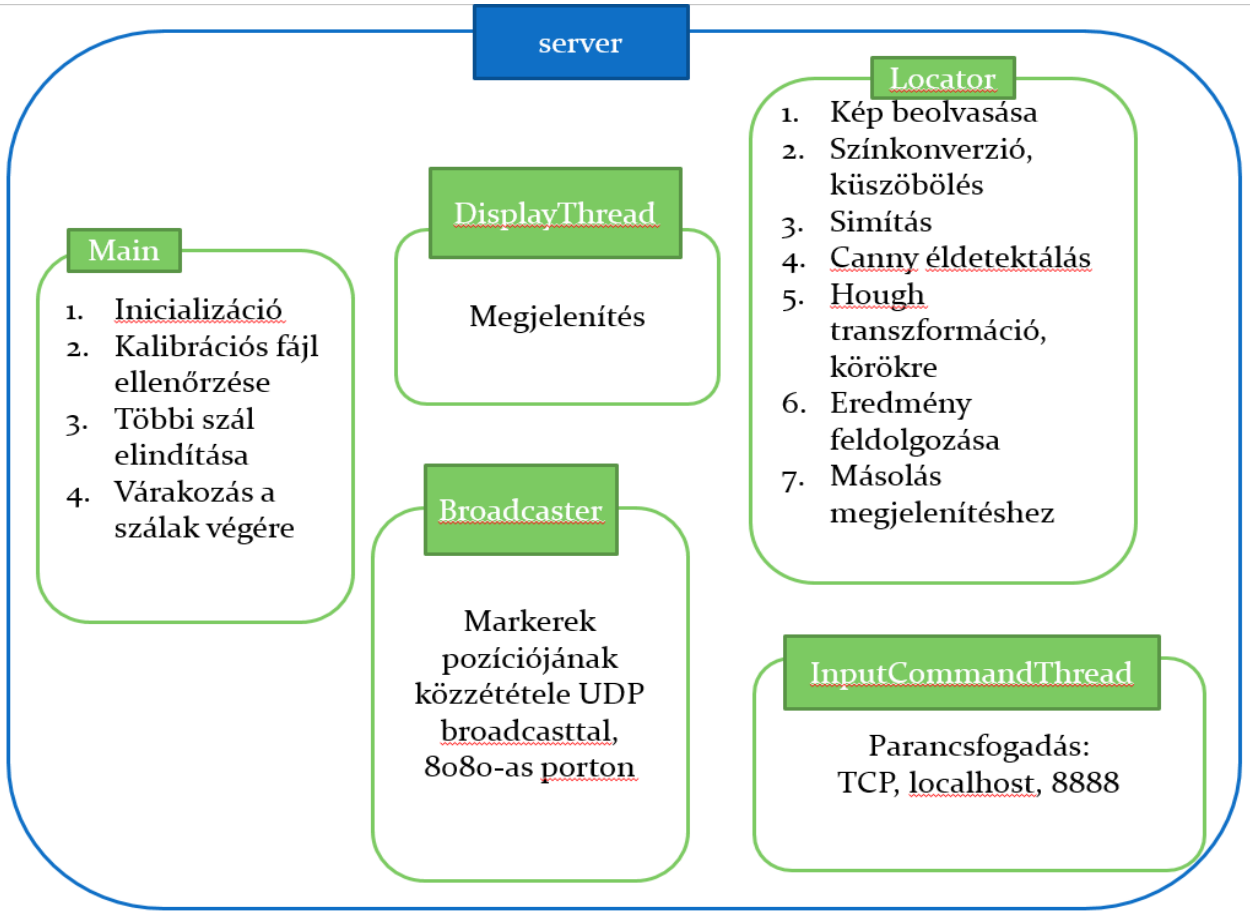
1. A látórendszer felépítése

A látórendszer tervezett felépítése az alábbi ábrán látható:



A rendszer kliens-szerver architektúrára épül. Két külön végrehajtható állományt képező része a feldolgozó program (szerver) és a grafikus kezelőfelület (kliens).

A kliens-szerver architektúra előnye a különböző funkcionalitást biztosító programok szeparáltsága. A kliens oldalon fellépő esetleges hibák, lassulások így nem befolyásolják a szerver képfeldolgozásának a folyamatát, továbbá egy grafikus kezelőfelület kiszolgálása nem bonyolítja és lassítja a szervert. Fontos megemlíteni azt az előnyt is, hogy kliensként tetszőleges alkalmazás implementálható, így a kezelőfelület bármikor könnyedén megváltoztatható.



2. A feldolgozó program (szerver)

A szerver program egy Linux operációs rendszer alatt futó többszálú folyamat. Belépési pontja a main függvény, ami elindítja a különböző funkciókat biztosító szálakat.

3.1 Indulás, kalibráció ellenőrzése (ezen dolgozni kell)

A szerver indulásakor a *main()* függvény ellenőrzi, hogy rendelkezésre állnak-e egy korábbi kamera-kalibráció eredményei az *out_camera_data.xml* nevű állományban. Amennyiben rendelkezésre állnak az adatok, akkor megkísérli betölteni őket és felvenni a kamera mátrixot. [ez nincs megvalósítva a programban, a linkelt példa kód módosításával működik].

Ha a *calibration_data.xml* fájl nem létezik, a program beolvassa a *default.xml* fájlban megadott kezdeti paramétereket a kalibrációhoz és elvégzi a kamera kalibrációját [ez nincs megvalósítva a programban, a linkelt példa kód módosításával működik]. Ehhez szükséges paraméterek a következők [lásd tutorial illetve default.xml, out_camera_data.xml és images.xml] :

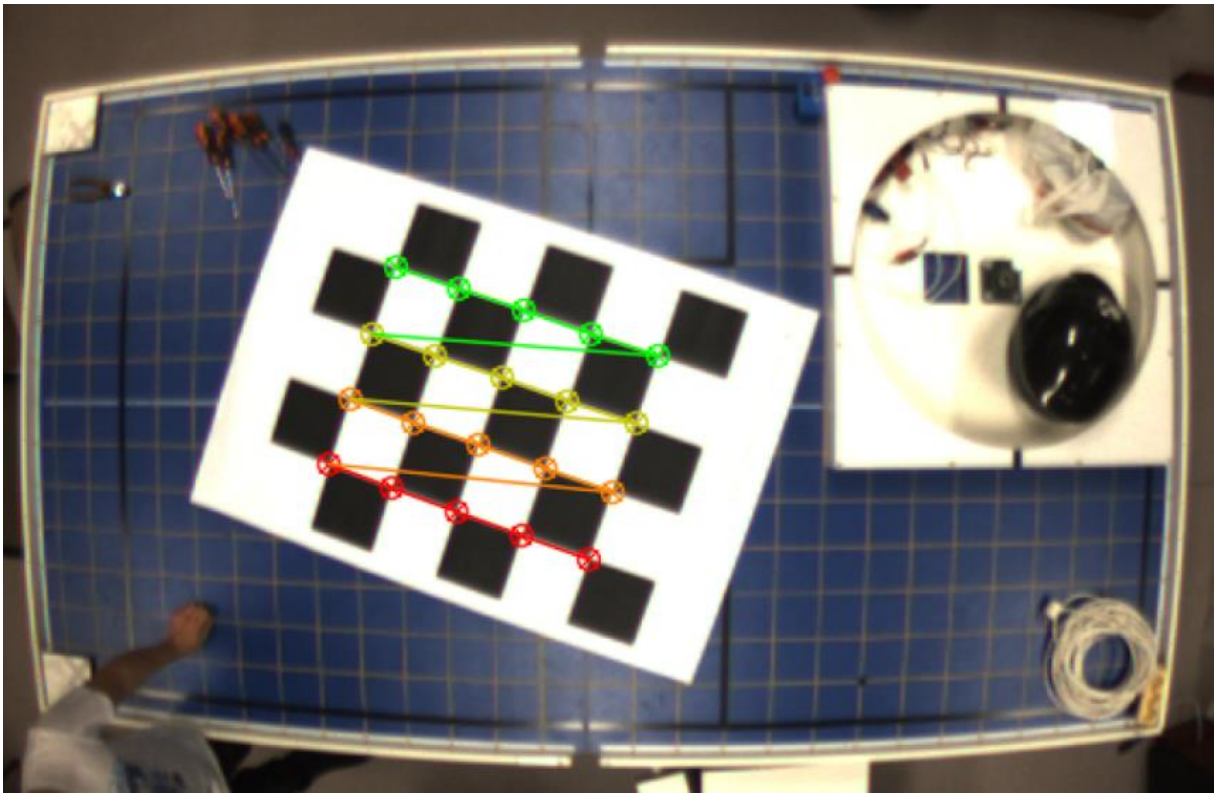
- a használt sakktábla mintázat belső sarokpontjainak száma
- egy mező mérete mm-ben

- a kalibrációhoz használt képek könyvtára
- a felhasznált képek száma

Az eredményes kalibrációhoz legalább 3 darab, egymáshoz képest eltolt helyzetben és orientációban szükséges felvételt készíteni a sakktábláról. Az adatok robusztusságának növelése céljából én 9 darab felvétellel dolgoztam a tesztek során.

A kalibráció kódjáról egy tutorial érhető el itt:
http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

Kisebb módosításokkal ez került felhasználásra.



Az XML fájlok kezelése az OpenCV könyvtár *FileStorage* osztájának segítségével történik, a kalibrációt a *calcBoardCornerPositions()*, *findChessboardCorners()*, *calibrateCamera()* függvények végzik el.

A kalibrációs folyamat megértését segít(het)ik a tutorialon kívül a következő olvasmányok:

- A practical Introduction to Computer vision with OpenCV: 2.fejezet
- OReilly Learning OpenCV: 11. fejezet (részletes, nehéz)

Sikeres kalibráció esetén elindításra kerül a képfeldolgozó, publikáló és parancsfogadó szál.

3.2. A képfeldolgozó szál (Locator)

A képfeldolgozó szál példányosít egy VideoCapture objektumot, melynek segítségével a rendszerben található alapértelmezett kamera eszköz képe olvasható be. Amennyiben ez sikeres, a szál elindítja a megjelenítő szálat majd egy végtelen ciklusba lépve megkezdi a kamera képének a feldolgozását, kör alakú mintázatok keresése céljából.

A képfeldolgozás lépései a következőek:

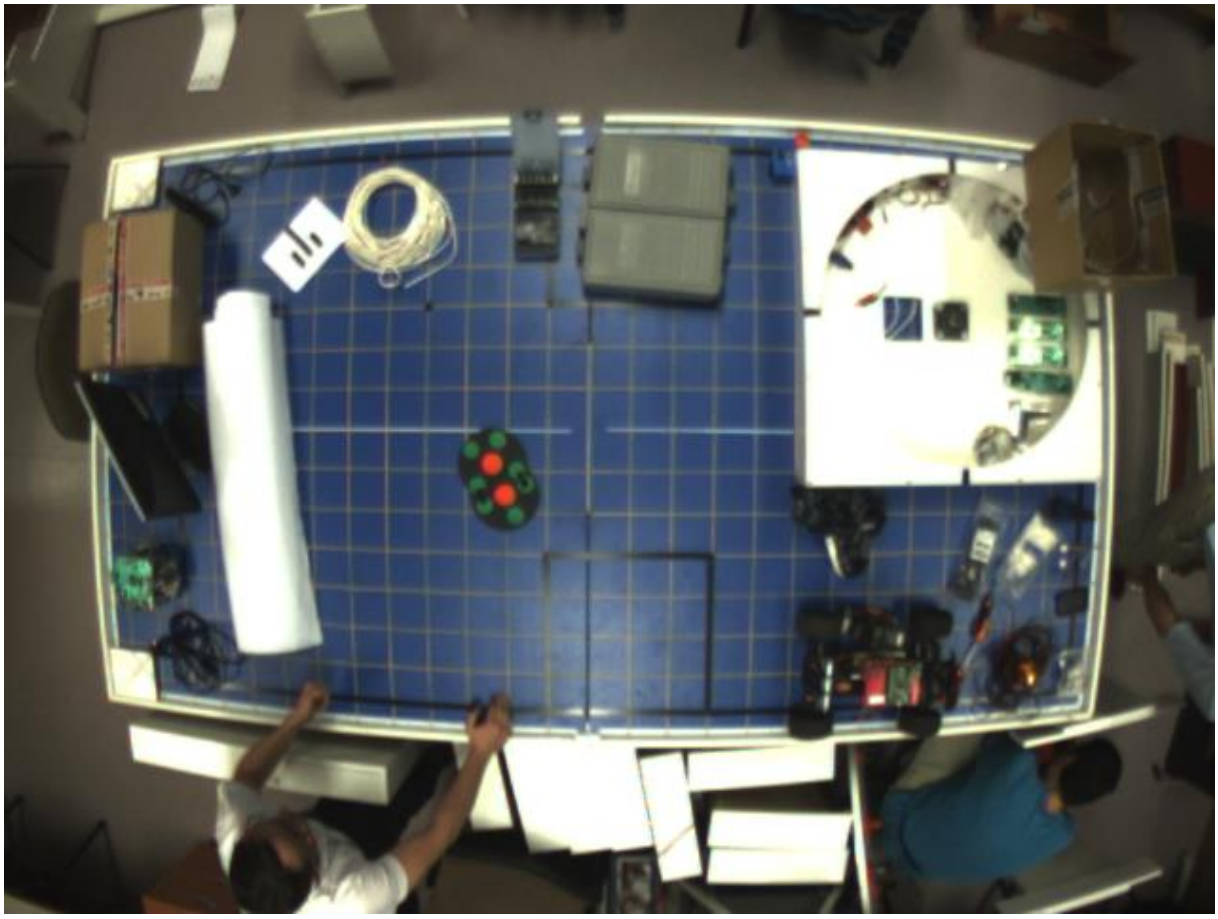
1. szín szerinti küszöbölés
2. élsimítás
3. Hough-transzformáció, körkeresés
4. Koordináták meghatározása

3.2.1 Szín szerinti küszöbölés

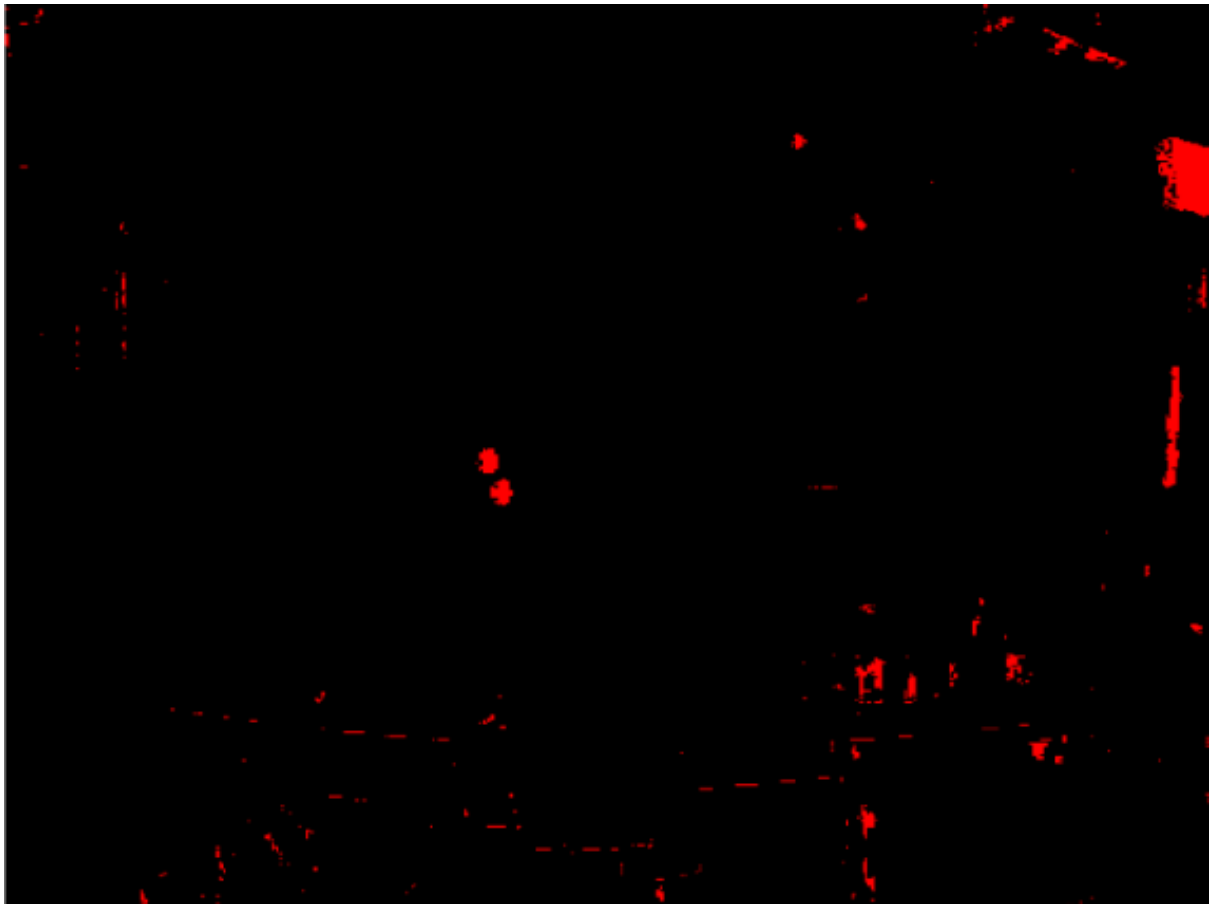
A küszöböléshez a keret a *cvtColor()* függvény segítségével HSV színképbe konvertálódik. A HSV használatának előnye, hogy a hue (árnyalat) érték és telítettség (saturation) küszöbök használatával a küszöbölés (nagy mértékben) függetlenné válik a terem világítási viszonyaitól, nem szükséges a küszöbölés határait egy-egy zavarás (pl. teremvilágítás felkapcsolása) után újra beállítani.

A küszöbölés során a bemenő kép minden olyan pontja, ami kívül esik a küszöbökön feketére kerül beállításra, a többi pont pedig változatlan (illetve a megjelenítéshez vörös színűre festve) marad. Az így nyert kép végül szürkeárnyalatossá alakul.

Jelenleg a küszöbölés határai fixek (vörös szín).



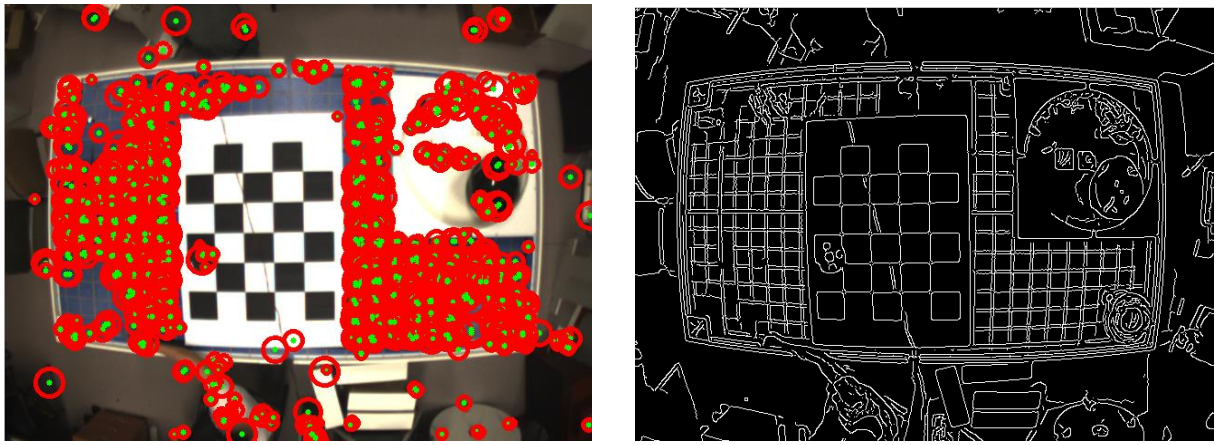
Kép küszöbölés előtt



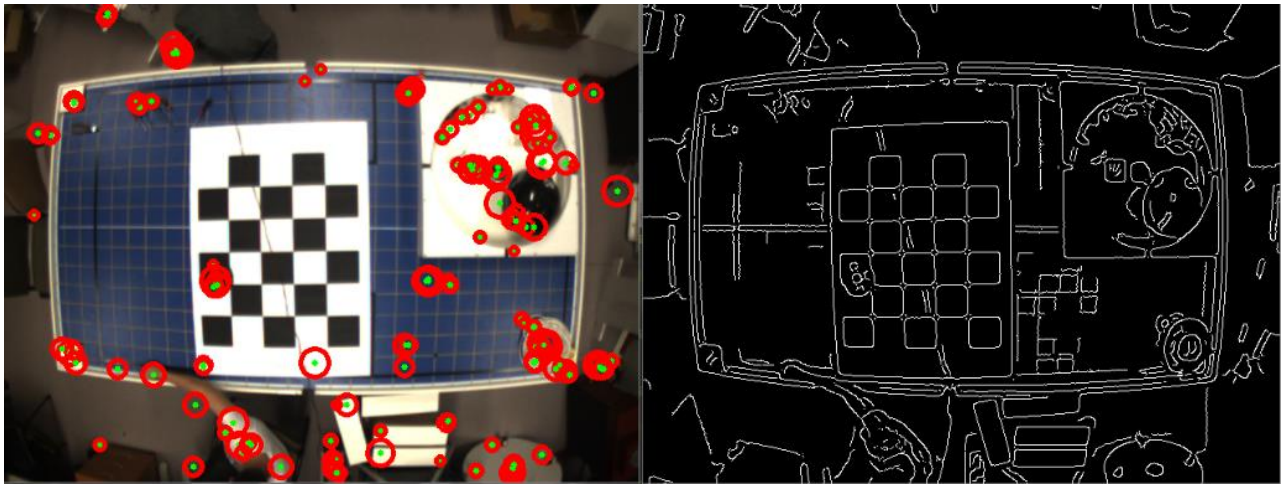
Kép küszöbölés után

3.2.2 Élsimítás

A köröket kereső Hough-transzformációt megvalósító függvény elsőnek a keresés kiindulópontjainak meghatározásához detektálja a forrás kép éleit. Abból a célból, hogy a körkeresés ne találjon túl sok kezdeti élt, a keretek egy Gauss-os zajt tartalmazó kernellel konvolválódnak (élsimítás), így csökken a zajosan detektált élek száma.



Körkeresés és éldetektálás simítás nélküli, küszöböletlen képen

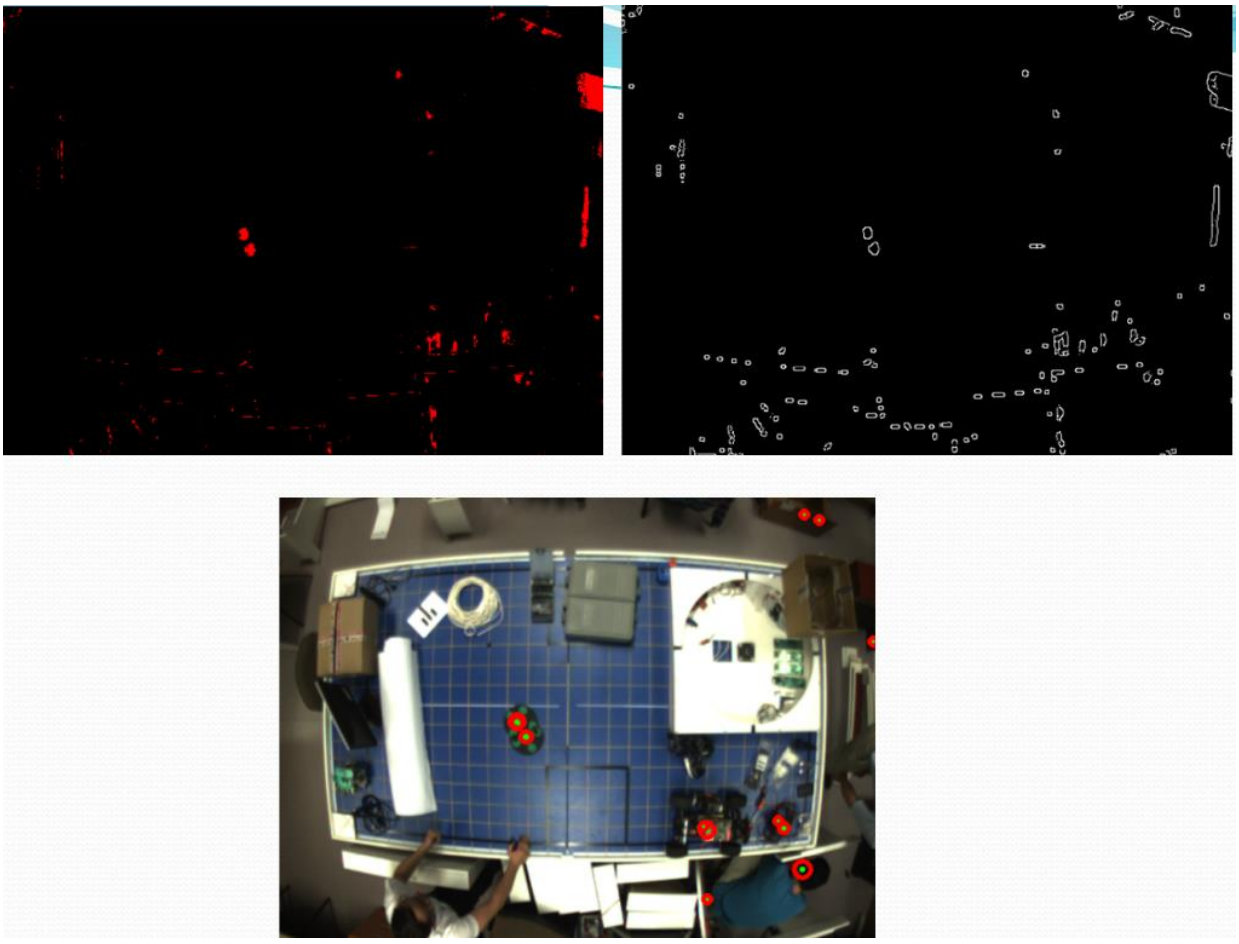


Körkeresés és éldetektálás simított, küszöböletlen képen

3.2.3 Hough transzformáció, körkeresés

A Hough transzformáció feladata egyszerű formák keresése, mint pl. egyenesek és körök, de a transzformáció általánosítható tetszőleges alakzat keresésére is.

Az OpenCV a *HoughCircles()* függvénnyel támogatja körök keresését.



Körkeresés simított, küszöbölt képen

Mivel a transzformáció több kört is megtalál, mint ami fontos a számunkra, ezek közül a ***DropCirclesByRadius()*** nevű függvény eldobja azokat, amelyeknek nem a megadott határok között mozog a sugara. Ezután a szál lockolja a Markerek pozícióját tároló vektort védő mutexet és a ***LocateByDistance()*** feltölti a vektort a megtalált markerek pozícióival. Azok a detektált pozíciók, amik egy megadott távolsághatáron belüli pontokat reprezentálnak átlagolásra kerülnek (pl a példaképen az egyetlen két piros körrel ellátott markerből 3-at látna enélkül a program, így viszont egy pozíciót érzékel, ami a valóságnak megfelelő).

A detektált köröket(piros-zöld) és a megtalált marker középpont pozíciót (kék) a szál rárajzolja az eredeti képre, majd az így átalakított képekről másolatot készít a megjelenítő szál számára. Ennek az az oka, hogy ha a megjelenítő (DisplayThread) egyszerűen zárolná a képeket védő mutexet és megjelenítené a képeket, akkor legtöbbször a feldolgozás nélküli állapot lenne látható (hiszen versenyhelyzet lenne a megjelenítő és a feldolgozó szál között a képekért. Feldolgozni (Hough transzformáció) sokáig tart egy képet, azaz lassan születnek eredmények, amiket rárajzolhatna a Locator az eredeti képre. Ennek eredményeként a megjelenítés sokkal több ideig a módosítatlan képeket látná, és nem látszódna a feldolgozás eredménye.)

Végezetül a detekció eredményét tartalmazó vektort a Locator szál mutex-el védett globális változókon keresztül publikálja a közzétételt végző szál számára és jelzésével jelzi a megjelenítő szál számára, hogy új adatokat jeleníthet meg.

3.2.4 Koordináták meghatározása [Nincs megvalósítva]

A kalibrációs folyamat során nyert kameramátrix, a detektált mintázat képbeli koordinátái és a kamera távolsága alapján a detektált körök valós térbeli koordinátái meghatározhatóak.

$$q = MQ, \quad \text{where} \quad q = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, \quad M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

A képletben M a kalibráció során nyert kameramátrix, ami tartalmazza a kamera pixelekre leképező egységének nem teljesen négyzet alakja miatt bevezetett f_x és f_y fókusztávolságokat, és a chip középpontjának a képsík optikai tengelyétől való távolságát jellemző c_x és c_y paramétereket. A q vektor a detektált pont koordinátáit tartalmazza a képsíkon kiegészítve a kamerától vett távolsággal (w = Z), a Q vektor pedig a pont való világbeli koordinátáit tartalmazza.

A fenti képlettel adott egyenletet Q-ra megoldva kapjuk a detektált körök valós térbeli koordinátáit.

3.3. A közzétevő szál

A közzétevő szál lokális UDP broadcast csomagokba ágyazva teszi közzé JSON formátumban a detektált markerek koordinátáit.

3.4. A megjelenítő szál

A megjelenítő szál az OpenCV *namedWindow()* és *imshow()* függvényeinek használatával külön ablakokban vizualizálja a feldolgozás alatt lévő, illetve a küszöbölt keretet és a detekció eredményével módosított eredeti keretet is.

Mindez azonban csak a megjelenítést kapcsoló globális változó bekapcsolt állapota esetén történik meg.

3.4. A parancsfogadó szál

A parancsfogadó szál TCP socket-en keresztül képes parancsok fogadására, a 8888-as porton. Lefordítva és a Readme.md-nek megfelelően futtatva a példa kódokat látható a kommunikációs szálak működése.

3. A grafikus kezelőfelület (kliens)

A látórendszert támogató kliens GUI még nem készült el, célszerű lenne socket alapú kommunikációt használó program fejlesztése, hogy akár másik gépről is vezérelhető legyen a program.

5. Összefoglalás, fejlesztési lehetőségek

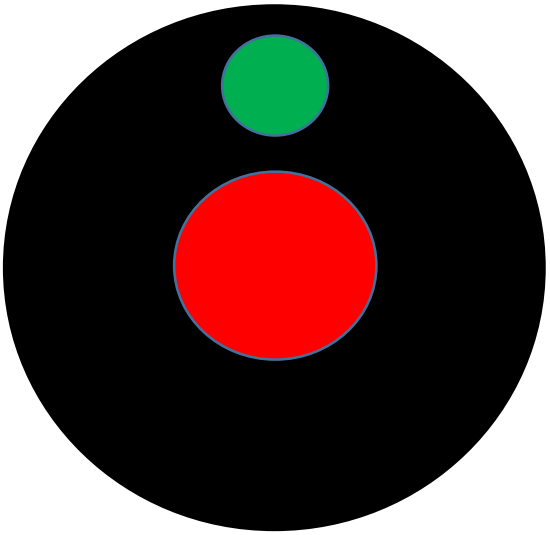
Összefoglalva a félév során megismertem az OpenCV alapvető használatát, a felhasznált algoritmusok elméleti hátterét és kialakítottam egy körök pozíciójának detektálására alkalmas bővíthető szerver programot.

A szerver képes 10 FPS sebességű kördetektálásra, kikapcsolt megjelenítés mellett egy Intel Core 2 Duo, 2GB RAM konfigurációjú Linuxos rendszeren.

A bekapcsolt megjelenítés a rendszer terheltségétől függően akadozik.

5.1 A jövőbeni fejlesztések

A kördetektálás felhasználásával és a küszöbölés határértékeinek módosításával olyan mintázat készíthető, aminek pozíciója mellett az orientációja is pontosan meghatározható.



Az eredeti célkitűzések teljesítéséhez szükséges továbbá az akadályfelismerés algoritmusának kidolgozása és implementálása is.

Ennek megoldására javaslom a detektálandó akadályok széleinek megjelölését egy egyedi színnel, majd a kamera képét ezen szín szerint küszöbölve, az éleket detektálva összefüggő vonalak keresését. Az így kapott összefüggő vonalak képeznék a detektált akadályok széleit.

A parancsfogadó szál FIFO alapú megoldása lecserélhető TCP/IP socket alapú fogadásra is, ami által a grafikus interfész másik rendszeren is futtathatóvá, fejleszthetővé válna. Ez a megoldás csökkentené a szervert futtató gép terheltségét és gyorsítaná a feldolgozást [KÉSZ].