SZAKDOLGOZAT-FELADAT

Sági Tamás (BK4AIA)
szigorló villamosmérnök hallgató részére

Kamera alapú intelligens gyártósori ellenőrző rendszer képfeldolgozó algoritmusának fejlesztése

Az Industry 4.0 trend térnyerésével egyre nagyobb szerepet kapnak az intelligens rendszerek a gyárak irányításában és minőségbiztosításában.

A gyártósorok mérete és bonyolultsága termékenként változhat, az emberi beavatkozás azonban a legtöbb helyen még mindig jelen van. Jelenleg a magyarországi Bosch gyárakban is vannak olyan gyártósorok, ahol betanított munkások végeznek bizonyos folyamatokat. Például a gyártandó termék alkatrészeit kézzel helyezik az összeszerelő tálcákba, amelynek teljességét egy munkás ellenőrzi a folyamat végén. Pozitív visszajelzés esetén a tálca a következő állomásra kerül, ahol szintén egy munkás végzi el az összeszerelést. Abban az esetben, ha a tálca hiányosan kerül az összeszerelő állomásra, vissza kell küldeni pótlására, ami további költséget és várakozási időt jelent. Az emberi pontatlanság kiküszöbölése érdekében javasolt egy kamera alapú, intelligens megfigyelő rendszer alkalmazása.

A hallgató feladata egy sztereó kamerából álló konfiguráció mellett egy keretrendszer megvalósítása, amely képes felismerni az összeszerelő tálcában az alkatrészek jelenlétét/hiányát.

A hallgató feladatának a következőkre kell kiterjednie:

- Mutassa be a sztereó kamerás rendszerek felépítését, matematikai összefüggéseit;
- Mutassa be a mélységi térkép létrehozásának folyamatát;
- Mutassa be a gépi tanulással kapcsolatos alapösszefüggéseket;
- Mutassa be a rendszert: kamerák száma, típusa, pozíciója;
- Határozza meg a használandó keretrendszert (Python, OpenCV, Matlab);
- Készítse el a képfeldolgozó keretrendszer prototípusának implementációját;
- Elemezze és hasonlítsa össze a következőket:
  - Sztereó képalkotással történő képfeldolgozás
  - Gépi tanulással történő képfeldolgozás
- Válassza ki a jobbnak bizonyult algoritmust, döntését igazolja.

Tanszéki konzulens: Dr. Sujbert László, docens
Külső konzulens: Anka László (Robert Bosch Kft.)

Budapest, 2017. március 10.

.....
Dr. Dabóczi Tamás
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.E.444.
Telefon: 463-2057, Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Sági Tamás Viktor

KAMERA ALAPÚ INTELLIGENS GYÁRTÓSORI ELLENŐRZŐ RENDSZER KÉPFELDOLGOZÓ ALGORITMUSÁNAK FEJLESZTÉSE

KONZULENS

Sujbert László

BUDAPEST, 2017

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott Sági Tamás Viktor, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2017. 05. 16.

...…….
Sági Tamás Viktor

Összefoglaló

Az Industry 4.0 trend térnyerésével egyre nagyobb szerepet kapnak az intelligens rendszerek a gyárak irányításában és minőségbiztosításában. A gyártósorok mérete és bonyolultsága termékenként változhat, azonban az emberi beavatkozás a legtöbb helyen még mindig jelen van. Jelenleg a magyarországi Robert Bosch Kft gyáraiban is vannak olyan gyártósorok, ahol betanított munkások végeznek bizonyos folyamatokat. Például a gyártandó termék alkatrészeit kézzel helyezik az összeszerelő tálcákba, aminek a teljességét egy munkás ellenőrzi a folyamat végén. Pozitív visszajelzés esetén a tálca a következő állomásra kerül, ahol szintén egy munkás végzi el az összeszerelést. Abban az esetben, ha a tálca hiányosan kerül az összeszerelő állomásra, vissza kell küldeni annak pótlására, ami további költséget és várakozási időt jelent. Az emberi pontatlanság kiküszöbölése érdekében javasolt egy kamera alapú, intelligens megfigyelő rendszer alkalmazása.

Feladatom egy keretrendszer szoftverprototípusának megtervezése és megvalósítása, amely képes felismerni az összeszerelő tálca jelenlévő alkatrészeit és vizuálisan jelezni a hiányzóakat. Az alkatrészek megtalálására két különböző megoldást próbálok ki, ezek közül kiválasztom a jobbnak bizonyulót. Az egyik során sztereó kamerák képeinek diszparitását vizsgálom és ebből határozom meg az alkatrészek jelenlétét. A másik megoldás során betanított algoritmusokkal keresem az alkatrészeket. Itt morfológiai műveleteket és template matching-et használok. A felvétel készítése és az algoritmus futása nem vehet igénybe többet három másodpercnél, így a jobbnak bizonyuló algoritmus választási szempontjai között első helyen a gyorsaság és a pontosság áll.

Implementáltam az alkatrészek szegmentálására és megtalálására szolgáló algoritmusokat, ehhez létrehoztam egy egyszerű grafikus felhasználói felületet, amivel lehetőség nyílik valós időben tesztelni az alkalmazást és megváltoztatni a megoldások során használt paraméterek értékeit. A sztereó kamerás megoldás sokkal erőforrásigényesebbnek és pontatlanabbnak bizonyult, mint a betanított algoritmus, amivel jó eredményeket sikerült elérnem. Utóbbit valós időben (élő kamera képén) is sikerült tesztelnem, amely során ugyan olyan pontosnak bizonyult.

Abstract

With the increasing significance of Industry 4.0, intelligent systems and services are gaining more and more importance in the life of an assembly factory. Whole product lines are being controlled and supervised by such systems in order to ensure the quality requirements of the product. However the size and the complexity of an assembly line differs from product to product and the individual steps can vary based on the manufacturer, the human factor is playing an important role in the process and hence human error shall be mitigated. Currently there are product lines in the factories of Robert Bosch, where the assembly itself is being performed by employees. The parts of the product will be gathered manually into an assembly tray from the so called product super markets. A completeness check will be initiated by the employee after the gathering process finished. If the result is positive, the assembly tray will be forwarded to the assembly station, where the trained employees are performing the final steps. In case of a false negative completeness check (part is not there, but is not recognized), the tray shall be sent back to the gathering station that produces unnecessary cost and delay. Therefore a system shall be developed that performs a camera sensor based completeness check.

My task is to create an experimental framework and evaluate the performance of different image processing algorithms. A proposal for possible solution with the highest accuracy and availability is to be given. The framework shall recognize the present parts and visually mark the location of the empty slots. For the recognition I will try two different algorithms and choose the better one. The first one is the examination of the disparity map created from stereo images. The other solution which I try is based on simple machine learning techniques. Those are morphology image processing and template matching. The whole process (capturing the images and running the algorithm) shall not take longer than three seconds, which means the decision of the better solution will based on runtime and accuracy.

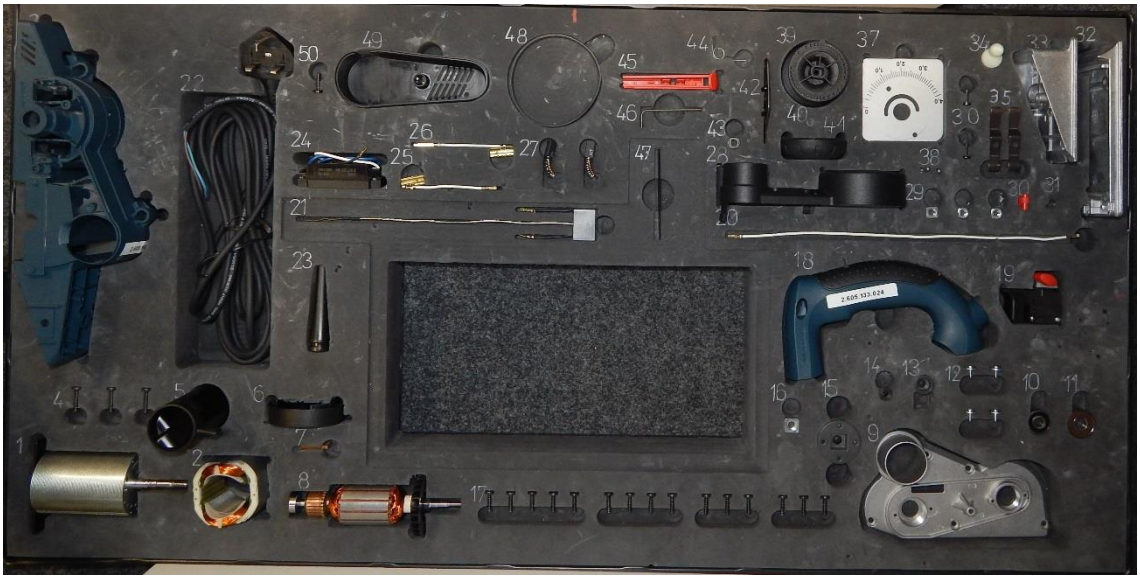
First I have implemented the algorithms for segmentation and recognition, then I have created a graphical user interface for real time configuration and visualization. The stereo vision algorithm was slower and more resource intensive, than template matching and morphology image processing. With those I have reached fairly good results even on real time camera image.

1 Bevezetés

Az első pontban adok egy átfogó képet a feladatomról és betekintést nyújtok a későbbi fejezetekbe.

1.1 A feladat részletes ismertetése

Szakdolgozatom során a következő összeszerelő tálcával foglalkoztam, ami esetemben egy gyalugép alkatrészeit tárolja (1.1. ábra).



1.1. ábra Összeszerelő tálca alkatrészekkel

A tálca $1m$ széles, $0.5m$ mély és $6cm$ magas. Gyári körülmények között az alkatrészeket munkások helyezik a megfelelő helyükre. Miután a munkások behelyezték a szükséges alkatrészeket, egy felülvizsgáló rendszer ellenőrzi, hogy minden alkatrész a helyén van-e. A rendszernek három másodperc áll rendelkezésére az ellenőrzés végigfutására. Pozitív kiértékelés esetén a tálca továbbhalad az összeszerelő állomáshoz. Ellenkező esetben az elkészített szoftvernek vizuálisan jeleznie kell, hogy melyik alkatrész hiányzik. Ehhez a rendszerhez hozzá tartoznak a megvilágítások, a tálca rögzítésére és pozícionálására szolgáló eszközök, és a számításokat végző számítógép. A megrendelők teljesen szabad kezet adtak az algoritmussal és a felhasznált eszközökkel kapcsolatban. Természetesen egy ilyen alkalmazásnál a futásidő, pontosság és a felhasznált eszközök ára a mérvadó. A Robert Bosch Kft.-nél én egy kamerás képfeldolgozással foglalkozó csapatban dolgozom, ezért esett a választás a kamerás

rendszerre. Feladatom, hogy eldöntsem, alkalmas-e ennek a problémának a megoldására egy egyszerű kamerás rendszer vagy aktív alkalmazás (például lézerszkennelés) szükséges hozzá. Az alkalmasság azt jelenti, hogy megfelelő pontossággal felismeri az elkészített alkalmazás az alkatrészek hiányát/meglétét, a feldolgozási folyamat befejeződik az elvárt három másodperc alatt.

Az alapgondolat az, hogy sztereó kamerás rendszer mélységi képének vizsgálatával vagy egy előre betanított algoritmussal lenne a leghatékonyabb felismerni az alkatrészek hiányát/meglétét. Konkrét feladatom, hogy a felülvizsgáló rendszer algoritmusának és keretrendszerének a prototípusát megtervezzem és megvalósítsam. Ezen kívül teszteltem a megvilágítás hatását, a mozgatható alkatrészek elmozdulásának velejáróit. Erre több különböző megoldást próbáltam ki, szakdolgozatomban ezen megoldások folyamatát, matematikai hátterét fogom ismertetni, és összehasonlítom őket futásidő és pontosság alapján. Dolgozatom során nem taglalom, hogy a tálca hogyan jut el a felülvizsgáló rendszer alá, ott hogyan kerül fix pozícióba. Ezt már késznek veszem és a felvételeket/teszteket úgy készítettem el, hogy a tálca már fix pozícióba került a kamera alatt. A kamera választásának indoklására is kitérek.

A második fejezet során ismertetem a sztereó látás alapvető összefüggéseit, matematikai leírásait, ezek nehézségeit. Megmutatom, hogy hogyan lesz egy képpárból sztereó kép.

A harmadik fejezetben ismertetem az általam használt betanított algoritmusokat és kitérek a morfológiai képfeldolgozásban használt műveletek matematikai hátterére. Ezután megmutatom ezek gyengeségeit, előnyeit.

A negyedik fejezet a felhasznált szoftverek és a kamera ismertetésével foglalkozik. Bemutatom a létrehozott keretrendszert, annak blokkdiagramját és az ehhez használt függvénykönyvtárakat.

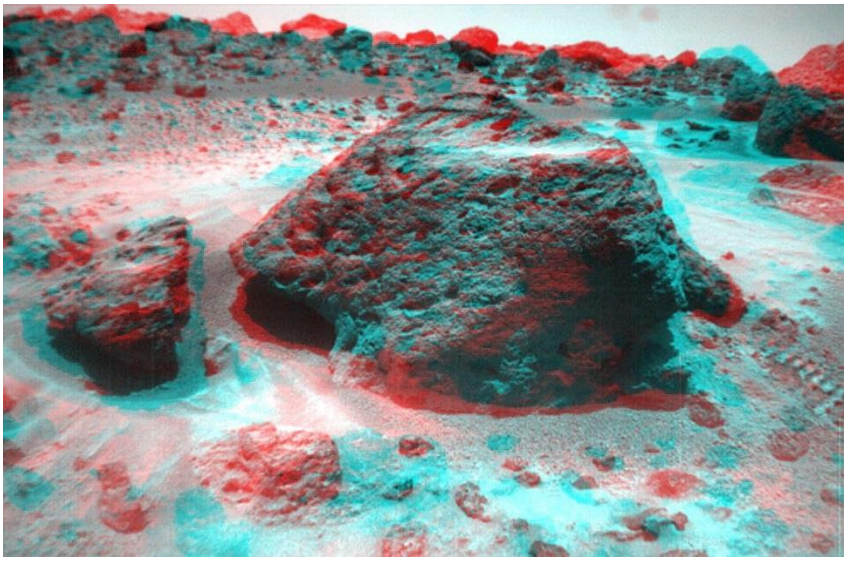
Az ötödik fejezet során megmutatom, hogy milyen eredményeket sikerült elérnem a fenti módszerekkel és összehasonlítom őket futásidő és pontosság alapján.

A hatodik és egyben utolsó fejezet során ismertetem, hogy milyen továbbfejlesztési lehetőségekre lettem figyelmes a fejlesztés során.

2 Sztereó gépi látás alapfogalmai

2.1 Sztereó látás

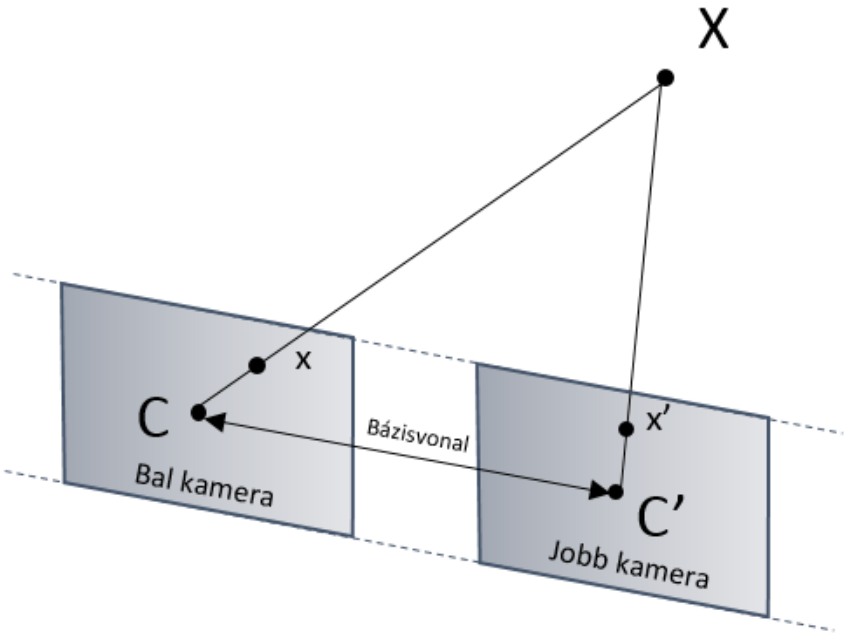
Sztereó látásról akkor beszélünk, ha valamit két szemszögből figyelünk meg egy időben. Alkalmazási területe igen széles. Leggyakrabban előforduló példa a sztereó látás bemutatására az anagliph képek, amilyet már a NASA is készített az 1990-es években a Mars Rover felvételeinél (2.1. ábra).



2.1. ábra Anagliph kép a Marsról[12]

Az anagliph képek úgy készülnek, hogy mindkét kamera képét átszínezik. Az egyiket piros, a másikat ciánkék színűre, majd 50%-os áttetszőséggel egymásra illesztik. A kapott képet megfelelő lencsével nézve sztereoszkópikus 3D látványt kapunk. A kétféle színű, de azonos tárgyhoz tartozó pontok távolsága a diszparitás. Az összetartozó képpontok távolsága csökken a térbeli pontok mélységével. A mélység és a diszparitás egymással fordítottan arányos mennyiségek. Az anagliph képek mozgó változata található meg a háromdimenziós mozikban, amelyek 2009-ben arattak sikert. Ezen kívül az ipar számos területén alkalmazzák, mint például a gyártástechnológiában, robotikában vagy akár az orvosi technológiában. A sztereó látás gyakori alkalmazása annak köszönhető, hogy alkalmas térbeli rekonstrukcióra, azaz feltérképezhető a tér geometriája és meghatározható a térbeli pontok távolsága. A sztereó képeken a térbeli információt a diszparitás hordozza.

Sztereó felvétel készíthető előre gyártott, szoftveres vezérléssel rendelkező sztereó kamerával, több különálló kamerás rendszerrel vagy akár egy kamerával is. Természetesen az előre gyártott sztereó kamerák a legpontosabbak, és garantáltan egy időben készítenek képet az eseményről, így akár sztereó videó is készíthető velük. A sztereó kamerák két optikai középpontját összekötő egyenest bázisvonalnak nevezzük. A kamerákat többféleképpen is el lehet helyezni egy sztereó felvételnél. Speciális eset, ha a kamerák képsíkjai egybeesnek és a bázisvonallal párhuzamosak, továbbá az azonos térbeli pontok vetületei azonos sorszámú rasztersorokban helyezkednek el a két képen. Ezt standard elrendezésnek nevezzük[10]. A következő ábrán látható egy standard sztereó elrendezés (2.2. ábra).

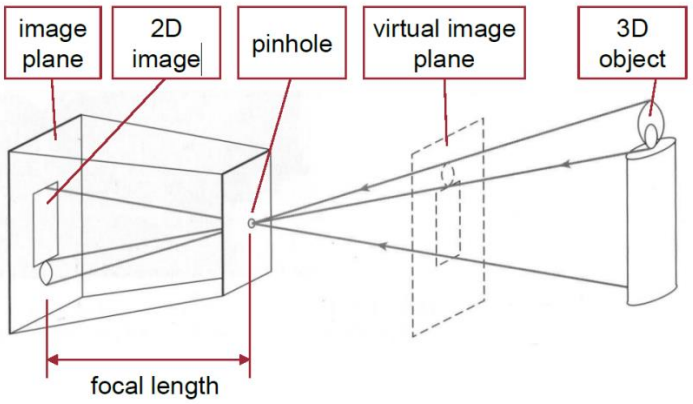


2.2. ábra Standard sztereó elrendezés

Mint említettem, egy egyszerű fényképezőgéppel is készíthető sztereó képpár, azonban a kamera másik pozícióba helyezéséig a fényképezendő tárgynak mozdulatlannak kell maradnia, így ezzel a módszerrel csak reprodukálható eseményekről készíthető sztereó videó. Az általam használt alkalmazások közelítésekkel élnek a számítások megkönnyítése érdekében. Az első ilyen közelítést az adja, hogy nem valódi kameramodellt, hanem egy egyszerűsített modellt alkalmaznak a számítások során.

2.2 Kameramodell

A lyukkamera (*pinhole camera*), úgy modellezhető, mint ha nem rendelkezne lencsével és az apertúrája pedig egy pont lenne. Azaz a tárgyakról visszaverődő fénysugarak (vetítősugarak) egy ponton keresztül érkeznek a szenzorra. A vetítősugarak a kamera optikai középpontját és egy térbeli pontot kötnek össze. Ahol ez a sugár metszi a képsíkot, ott jelenik meg a két dimenzióra leképzett képe.



2.3. ábra Pinhole kamera modell[9]

Azt a pontot, ahol a vetítősugarak a kamerába érkeznek, gyújtópontnak (*pinhole*) nevezzük. A kamera belsejében a képsíkon fordított állású, kicsinyített kép keletkezik. Azt a pontot, ahol a beérkező fénysugár merőleges a képsíkra, optikai középpontnak nevezzük. Az optikai középpont távolsága a gyújtóponttól a fókusztávolság. Nem összetévesztendő a geometriai optikából ismert fókusztávolsággal! A lyukkamerák képe perspektív projekcióval képződik a képsíkon, melynek matematikai leírása a következő:

[8]
$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.1}$$

Ez két nagyobb részre bontható. A kamerák belső (az egyenlőség jobb oldalán álló első mátrix) és külső (az egyenlőség jobb oldalán álló második mátrix) paramétereire. Elöljáróban annyit említenék meg az 1-es egyenletről, hogy ennek segítségével vetíthető egy háromdimenziós pont (X, Y, Z) kétdimenziós síkra (u, v). A következő fejezetben ezekre a paraméterekre még visszatérek. A perspektív leképzés következménye, hogy a

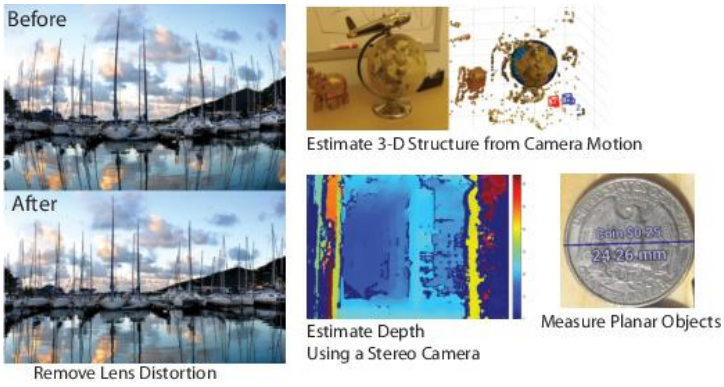
távolabbi objektumok kisebbnek látszanak, illetve a párhuzamos egyenesek a végtelenben vagy a képen találkoznak. A perspektív projekciónál egy pontot eggyel több koordinátával szokás jellemezni, mint ahány dimenziós. A plusz koordinátát homogén koordinátának nevezik. Neve onnan ered, hogy egy nem nulla számmal megszorozva a pont koordinátáit, ugyanazt a pontot kapjuk. A nulla értékű homogén koordináta pedig a végtelen távoli pontot írja le. Azért gyakori a használata, mert képes végtelen távoli elemeket leírni. A Descartes koordinátákra való áttérés nagyon egyszerű, mindössze annyit kell tennünk, hogy a koordinátákat végigosztjuk a homogén koordinátával, majd az egyest elhagyjuk a végéről. Erre egy példa:

$$m = (x, y, z)^T$$
$$m' = (\frac{x}{z}, \frac{y}{z})^T \tag{2.2}$$

Látható, hogy a vesszős változó harmadik koordinátája eltűnik.

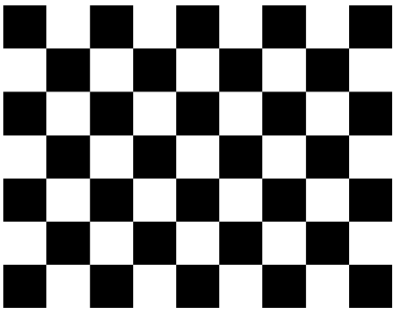
2.3 Kamera kalibráció

A kamerák képalkotása nem tökéletes, azonban a hibáik számokban kifejezhetők. Kamera kalibráció az a folyamat, amely során megbecsülhetőek egy hagyományos vagy akár egy videokamera lencséjének és szenzorjának paraméterei. A paraméterek segítségével: csökkenthető a kamerával készített kép torzítása, megmérhető egy tárgy mérete pixelekben és ebből következtethető a valódi hossza vagy meghatározható a kamera helyzete a képen. Esetemben a pontos rektifikációhoz elkerülhetetlen a paraméterek ismerete. A kamera kalibráció pontosságát gyakran használják még háromdimenziós képrekonstruáláshoz, a robottechnológiában és navigációs rendszerekben[3]. Említésre méltó, hogy egy kamera a mátrixával egyértelműen reprezentálható.



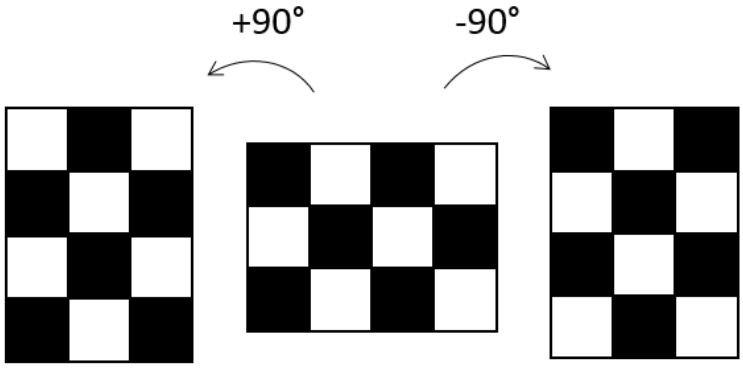
2.4. ábra A kamera kalibráció felhasználási lehetőségei[3]

A kamerák paramétereihez három érték sorolható: a kamera belső, külső paraméterei és a torzítási együtthatók. Ahhoz, hogy meg lehessen határozni ezeket az értékeket, szükség van egy háromdimenziós világbeli koordinátára és a hozzá tartozó kétdimenziós képi koordinátára. Ez az összefüggő pontpár kalibrációs minta használatával kapható meg. A legelterjedtebb kalibrációs minta a sakktábla, amin előre meghatározott oldalhosszúságú és darabszámú négyzetek találhatóak. A következő ábrán (2.5. ábra) látható egy 9x7 –es sakktábla.



2.5. ábraSakktábla alakú kalibrációs minta[8]

Célszerű páros-páratlan számú négyzetekből álló sakktáblát választani, hiszen ekkor egyértelmű, a sakktábla forgatásának iránya, így pontosabb kalibrációhoz jutunk. A 2.6-os ábrán a vízszintesen tartott (középső) 3x4-es kalibrációs mintát az óramutató járásával megegyező és ellentétes irányban is elforgattam. Látható, hogy egyértelműen eldönthető a függőleges minta legfelső és legalsó sorának vizsgálatából, hogy melyik irányba lett forgatva.



2.6. ábra Kalibrációs minta forgatása

Én a feladatom során egy 9x8-as, 2cm oldalhosszúságú négyzetekből álló mintát használtam. Emellett gyakran használnak mintaként két egymásra merőleges, eltérő hosszúságú vonalat, amik mérete ismert. Ennek az „L”betűnek a szárai a kamera képén

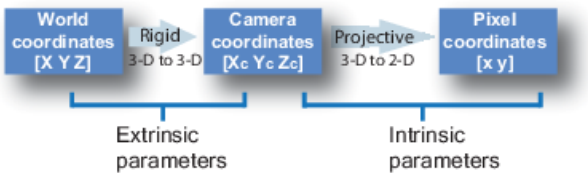
valahányszoros kicsinyítésben láthatók és egymással valamekkora szöget zárnak be, ezen adatok segítségével határozzák meg a külső paramétereket.

A kalibráció többféle inputtal is elvégezhető. Ez lehet korábban készített fénykép, videó vagy akár élő kamerakép is. A számolt paraméterek pontossága az inputként adott minták számával nő. Ahhoz, hogy megfelelő pontosságú kalibrációt kapjunk, a mintát bizonyos szabályszerűségek szerint kell tartanunk a kamera előtt. Az OpenCV-ben található kalibrációs függvények megkövetelik, hogy vízszintesen tartsuk a kalibrációs mintát, ami a sakktábla esetén azt jelenti, hogy vízszintesen több négyzetnek kell elhelyezkednie, mint függőlegesen. Az *X* tengely kalibrációjához a sakktáblát a bal és a jobb oldal felé, míg az *Y* tengely kalibrációjához a felső és alsó oldalak felé kell közelítenünk. A tábla különböző dőléseiből a ferdeségi együttható számolható, ami megadja az *x* és az *y* tengelyek között mérhető pixel-dőlésszöget.

A kalibráció végeztével annak sikerességét ellenőrizhetjük, ha kirajzoljuk a torzított és torzítatlan képet. Tipikusan az epipoláris hiba 0.25 pixel alatt elfogadható, 0.1 pixel alatt tökéletesnek mondható[5].

2.3.1 Belső paraméterek

Sikeres kamera kalibráció elvégzését követően megkapjuk a kamera paramétereit, azaz a belső, külső paramétereket és a torzítási együtthatókat. A belső paraméterek segítségével végezhető el az úgynevezett projektív transzformáció, ami a kamera koordinátarendszerből képi koordinátarendszerbe transzformálja a pontokat.



2.7. ábra Kamera paraméterek transzformációi[3]

A kamera belső paramétereit egy 3x3-as mátrixszal reprezentáljuk. Ez öt különböző adatot tartalmaz, amelyek a fókusztávolság (f_x,f_y), főpont (c_x,c_y) és az *x* és *y* tengely között mérhető pixel-dőlésszög (*s*). A pixel-dőlésszög az esetek túlnyomó részében nulla. Emellett a torzítási együtthatók is a kamera belső paramétereihez sorolhatók. Mivel a belső paraméterek nem függnek a felvételtől, ezért többször

felhasználhatóak, ha a fókusztávolság nem változott meg a két felvétel között. Ezért nem kalibrálhatóak az automatikus fókusszal rendelkező kamerák, mert azok önállóan változtatják a fókusztávolságot. Az 1-es egyenlet jobb oldalán álló első mátrixot nevezzük kamera kalibrációs mátrixnak.

$$\boldsymbol{K} = \begin{bmatrix} fx & s & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

A kalibrációs mátrix egy felső trianguláris mátrix, ami azt jelenti, hogy a főátló alatti összes elem nulla.

2.3.2 Külső paraméterek

A külső paraméterek (***R*** és *t*) meghatároznak egy transzformációt a háromdimenziós világ koordinátarendszer és a háromdimenziós kamera koordinátarendszer között. Ez az alábbi elemi transzformációk kompozíciójaként áll össze: egy *t* eltolás (transzláció), amely a világ koordinátarendszer origóját áttranszformálja a kamera középpontba - az eltolás nagysága pontosan a kamera középpont világ koordinátarendszerbeli inhomogén koordinátája lesz - ezután következik egy *R* forgatás (rotáció), amely a világ koordináta tengelyeket illeszti a kamera koordináta tengelyekre.

$$\boldsymbol{R} = \begin{bmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{bmatrix} \tag{2.4}$$

Ezt a két transzformációt írja le a 3x4-es úgynevezett kamera mátrixot, amely a 3x3-as rotáció mátrixból és a 3x1-es transzláció vektorból áll össze.

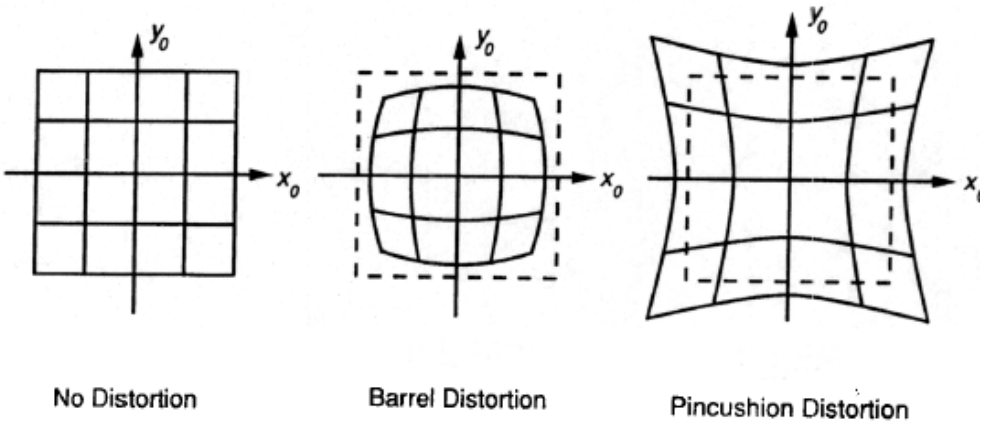
$$\boldsymbol{P} = \begin{bmatrix} r11 & r12 & r13 & t1 \\ r21 & r22 & r23 & t2 \\ r31 & r32 & r33 & t3 \end{bmatrix} \tag{2.5}$$

A ***P*** homogén kamera mátrix egyértelműen meghatározza a kamera pozícióját, mivel transzlációját és rotációját is tartalmazza. A kamera belső és külső paramétereinek segítségével pedig már két mátrixszorzással megkapható a háromdimenziós pontok kétdimenzióssá alakítása.

2.3.3 Torzítások

Minden optikának van geometriai torzítása. A torzulás mértéke általában arányos a kamera minőségével. Míg egy rosszabb minőségű kameránál nagyobb, úgy a jobb minőségűeknél kisebb mértékben jelentkeznek torzítások. Pontos mérésekhez, háromdimenziós rekonstrukcióhoz szükséges ezen torzítások korrekciója.

Leggyakoribbak a radiális torzítások, amelyek a kép széleihez közeledve felerősödnek. Ebből következik, hogy a nagy látószögű optikák mindig erősebben torzítanak. A radiális torzítás típusai: pozitív radiális torzítás (hordó), negatív radiális torzítás (párna). Ezen kívül jelentős lehet még a tangenciális torzítás hatása is, ami a gyártási hibák miatt keletkezik. Mivel a kamerák képéből kinyert információk alapján szeretnénk diszparitásokat számítani, rendkívül fontos, hogy a kamerák pontos képet adjanak. A kamerák torzított képéből torzításmentes képek készíthetőek kamera-kalibráció segítségével.



2.8. ábra A geometriai torzítások típusai[3]

Radiális torzítást akkor tapasztalunk a való életben, ha például egy magas épületről készítünk képet és az épület egyenes vonalai görbének tűnnek (nem párhuzamosak). Ezek a kamera kalibráció során meghatározott konstansokkal egyszerűen javíthatóak.

A radiális torzítás kiküszöbölésének általános alakja:

$$x = x_d(1 + k_1 r^2 + k_2 r^4)$$
$$y = y_d(1 + k_1 r^2 + k_2 r^4) \quad (2.6)$$

Ahol: (x_d,y_d) a torzított pontok koordinátái, r a sugár, k_1 és k_2 torzítási együtthatók, amelyeket a kamera kalibrációja során kapunk, (x,y) a torzításmentes pontok.

2.4 Rekonstrukció

A standard elrendezésben történő rekonstrukcióhoz szükséges a diszparitás és mélység közti összefüggés ismerete. A következőkben belátom, hogy a két mennyiség miért fordítottan arányos egymással. Vegyük a 2.2. ábra elrendezését, és legyen X egy térbeli pont, amelyre rálátnak a kamerák, x és x' pedig ennek a térbeli pontnak a vetületei. Jelölje b a bázistávolságot, f a fókusztávolságot. Tehát:

$$X = (X, Y, Z, 1)^T, \qquad x = (u, v, f)^T, x' = (u', v', f')^T. \tag{2.7}$$

A háromszögek hasonlósága miatt kimondható, hogy:

$$\frac{u}{f} = \frac{X}{Z}, illetve \frac{u'}{f} = \frac{X'}{z}. \tag{2.8}$$

Ebből a diszparitásra az adódik, hogy:

$$\Delta u = -\frac{fb}{Z}, valamint\ Z = -\frac{fb}{\Delta u}. \tag{2.9}$$

Ez az eredmény már mutatja, hogy a Δu diszparitás és a Z mélység fordítottan arányosak. A kifejezés akkor is érvényes lesz, ha a térbeli pont nem pont a képsíkokra merőleges, az optikai tengelyen áthaladó síkok közé esik. A képeken valójában nem a Δu diszparitást tudjuk megfigyelni, mivel x és x' nem a pixel koordináta-rendszerben adottak. Legyen

$$\boldsymbol{K} = \begin{bmatrix} -fx & s & cx \\ 0 & -fy & cy \\ 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

a kamerák azonos kamerakalibrációs mátrixa. A digitális képeken megfigyelhető pontok $Kx = (x, y, 1)^T$ és $Kx' = (x', y', 1)^T$. Felírható, hogy:

$$\lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{bmatrix} -fx & s & cx \\ 0 & -fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} u \\ v \\ f \end{pmatrix} \tag{2.11}$$

ahol λ egy skalár tényező. Látható, hogy $\lambda = f$. Ebből adódik:

$$x = -\frac{u}{s_x} + \frac{s}{f}v + c_x \text{ és } y = -\frac{v}{s_y} + c_y \tag{2.12}$$

Ugyanezzel az analógiával felírható ez az összefüggés *x'* és *y'*-re. Standard elrendezésben az egymásnak megfelelő pontok egy rasztersorban vannak. Ebből következik, hogy *y* = *y'*, amiből pedig, hogy *v* = *v'* is igaz kell legyen. Ekkor a digitális képeken mérhető horizontális diszparitás:

$$d = x' - x = \frac{1}{s_x}(u - u') + \frac{s}{f}(y' - y) \tag{2.13}$$

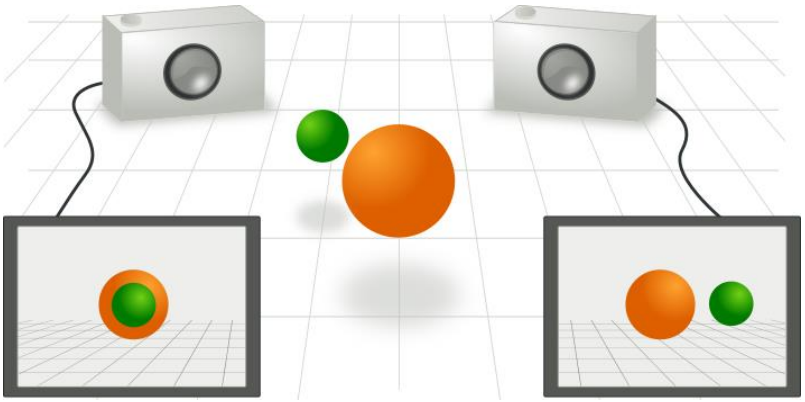
y = *y'* miatt a d diszparitás nem függ s-től. Vegyük még figyelembe a $\Delta u = -\frac{fb}{Z}$ egyenletet, így megállapítható, hogy:

$$d = \frac{f_x b}{Z}, illetve\ Z = \frac{f_x b}{d} \tag{2.14}$$

Z = *Z*(*x*,*y*) az (*x*,*y*) képpontban látszó mélység, míg d = d(x,y) a térbeli pont képei közt mérhető horizontális diszparitás pixelekben.

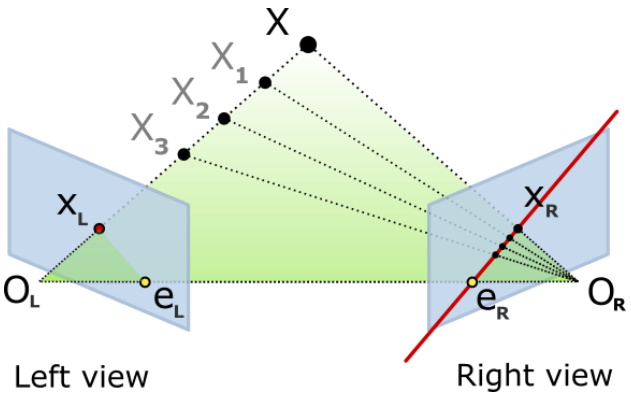
2.5 Epipoláris geometria

Ahogy korábban említettem, az epipoláris geometriát akkor használjuk, ha a két kamera nem standard elrendezést alkot. Az epipoláris geometriát gyakran emlegetik a sztereo látás geometriájának[6]. Amikor két kamera egy háromdimenziós jelenetet rögzít két különböző pozícióból, akkor számos geometriai kapcsolat képezhető a háromdimenziós pontok és azok kétdimenziós vetületei között. Ennek egy tipikus felhasználása látható a következő ábrán (2.9. ábra), ahol két kamera különböző nézőpontból készít képet egy eseményről, itt két gömbről.



2.9. ábra Az epipoláris geometria felhasználása[6]

A 2.10-es ábra két pinhole kamerát ábrázol, amik az *X* pontra néznek. Valódi kameráknál a képsík a fókuszpont mögött helyezkedik el, olyan képet állít elő, amely középpontosan tükrös a fókuszpontra. Itt azonban a probléma egyszerűsödik, egy virtuális képsík van a kamerák lencséinek fókuszpontja előtt, így olyan kép készül, amelyet nem befolyásol a lencse szimmetriája. Ezért használjuk a pinhole kamera egyszerűsítéseit. O_L és O_R a kamerák szimmetria-középpontjait, *X* a kamerák által vizsgált pontot reprezentálják, x_L és x_R az *X* pont vetületei a képsíkra.



2.10. ábra Két nem standard elrendezésű kamera az X pontra néz[10]

Mindkét kamera kétdimenziós információt rögzít a háromdimenziós jelenetről. Ez a konverzió 3D-ből 2D-be a perspektív leképzéshez (2.1) kapcsolódik és a pinhole kamerával írható le. A vetületet gyakran a kamerából kisugárzó, annak fókuszpontján áthaladó fénysugarakkal modellezik[6]. Megjegyzendő, hogy minden fénysugár pontosan egy ponthoz tartozik a képen.

Mivel a kamerák lencséinek optikai középpontjai nem esnek egybe, ezért az egyik kamerából nézve a másik kamera optikai középpontját, ez eltérő pontként jelenik meg a képsíkon. Ez a két pont e_L és e_R, amit epipólusnak vagy más néven epipoláris pontnak nevezünk. A két epipólus úgy kapható meg, hogy a kamerák optikai középpontjait, O_L-t és O_R-t egy vonallal összekötjük, és ennek a vonalnak a döféspontjai a képsíkkal adják az epipólusokat. Ezt az egyenest, azaz a két optikai középpontot összekötő vonalat bázis egyenesnek nevezzük[10].

Az O_L-*X* vonal a bal kamerából nézve egy pontnak látszik, mivel egy vonalba esik a kamera lencséjének optikai középpontjával. A jobb kamerából nézve ennek az egyenesnek a vetülete a képsíkon az e_R-X_R egyenes. Ezt nevezzük epipoláris egyenesnek. Szimmetrikusan, az O_R-*X*a jobb kamerából nézve egy pont, a balból nézve pedig az e_L-X_L epipoláris egyenes.

Megemlítendő még az X, O_L és O_R által kifeszített sík, amelyet epipoláris síknak nevezünk. A kamerák képsíkjainak az epipoláris síkkal vett metszésvonalai az epipoláris egyenesek. Belátható, hogy az epipoláris síkok és epipoláris egyenesek helyzete csak X-től függ. Akárhogyan is választjuk meg az X-et, az epipoláris síkok közös metszete a bázis egyenes és az epipoláris egyenesek képsíkbeli közös metszete az epipólus.

A kamerák relatív pozíciójának ismeretében két fontos szabályszerűséget figyelhetünk meg:

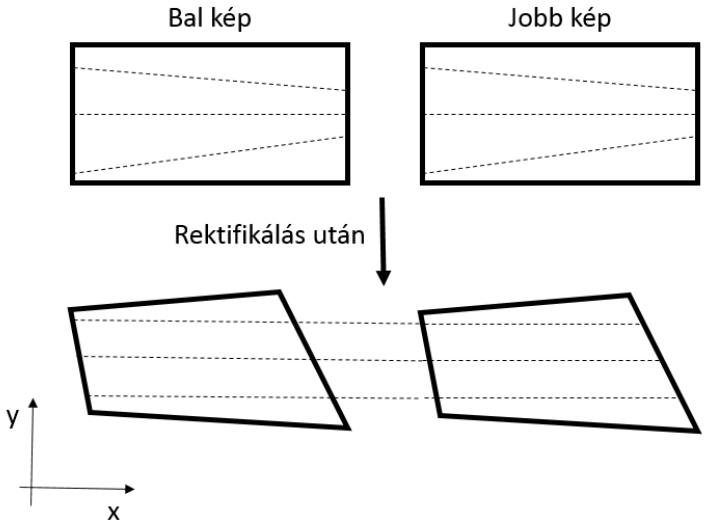
- Ha X_L ismert, akkor az e_R-X_R epipoláris egyenes is ismert (hiszen ez az O_L-X_L egyenes meghosszabbításának a vetülete a jobb kamera képsíkján), az Xpont vetülete a jobb képsíkra az X_R, ami mindig az e_R-X_R epipoláris egyenesre esik. Ez azt jelenti, hogy bármely pontot vizsgáljuk az egyik képsíkon, annak ismerni fogjuk a másik képsíkon létrehozott epipoláris egyenesét. Ez egy úgynevezett epipoláris függőséget ad: X vetületének a jobb kamera képsíkján (X_R) mindig az e_R-X_R epipoláris egyenesen kell lennie. Ez teljesül bármely pontra az O_L-X_L egyenesen (X_1, X_2, X_3). Ezzel a módszerrel tesztelhető, hogy két pont ugyanahhoz a 3D ponthoz tartozik-e. Epipoláris függőségek az esszenciális mátrix vagy fundamentális mátrix segítségével is leírhatóak a két kamera között.
- Ha az X_L és X_R pontok ismertek, akkor az O_L-X_L és O_R-X_R egyenesek számolhatóak. Ha ez a két pont összetartozó, akkor az egyeneseket meghosszabbítva pontosan X-ben metszik egymást. Ez azt jelenti, hogy X számolható két összetartozó képpont koordinátáiból. Ezt a folyamatot háromszögelésnek (*triangulation*) nevezzük.

Az epipoláris geometria egyszerűsödik, ha a két kamera egy síkba esik, azaz standard sztereó elrendezést alkotnak. Ebben az esetben az epipoláris egyenesek szintén egybevágnak, azaz E_L-P_L = E_R-P_R. Továbbá az epipoláris egyenesek párhuzamosak az O_L-O_R (a kamerák optikai középpontjait összekötő egyenesével) egyenessel.

Ha a kamerákat nem lehet úgy elhelyezni, hogy képsíkjaik egy síkba essenek, a kamera képi koordinátái áttranszformálhatóak úgy, mint ha ez mégis teljesülne. Ezt a folyamatot rektifikációnak nevezzük.

2.6 Rektifikáció

Ha a képeink nem standard elrendezésben készültek, akkor is lehetőségünk van diszparitások számolására. Valós körülmények között általában nincsenek standard elrendezésben a kamerák. Ilyenkor képeinket standard elrendezésre lehet hozni, ezt a folyamatot nevezzük sztereó rektifikációnak. Számomra azért fontos, hogy rektifikáljam a képeimet, mert a 2.6-os alfejezetben említett algoritmusok csak standard elrendezést megvalósító kamerák esetén képesek diszparitást számolni. A probléma megértésére a következő ábrát (2.11. ábra) készítettem.



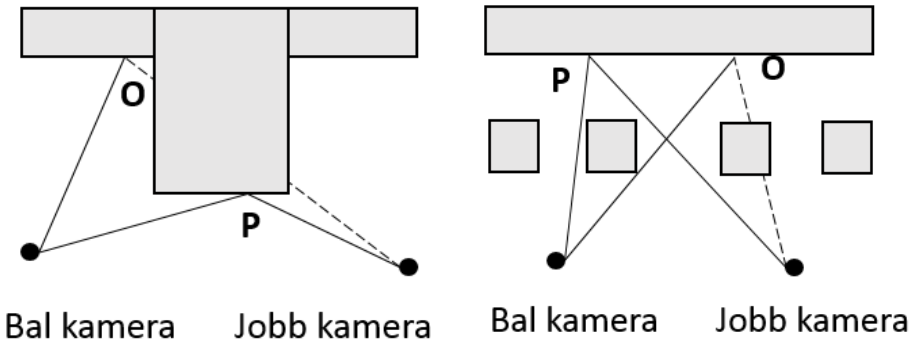
2.11. ábra Rektifikáció

Látható, hogy az epipoláris egyenesek nem horizontálisak, ami esetünkben azt jelenti, hogy nem párhuzamosak egymással és az *x* tengellyel. Sztereó rektifikáció során célunk, hogy a két kép epipoláris egyeneseit horizontálissá tegyük úgy, hogy azok egymásnak megfelelő rasztersorokban helyezkedjenek el. Mindkét képhez olyan homográfiát kell találnunk (sík-sík leképzés), amely az epipoláris egyeneseket horizontálissá teszi. Egy képhez végtelen sok homográfia található, ezek közül keressük azt, amely a legkevésbé torzítja a képeket.

Erre számos, korábban már megírt és bevált algoritmus létezik, én ezeket használtam és mutatom be a következő alfejezetekben, azonban először ismertetnék néhány előforduló problémát a diszparitás előállítása során.

2.7 Alapproblémák

A diszparitás számolás során több kényszerbe és akadályba ütközhetünk. A legelterjedtebb problémát a takarás (az angol irodalomban: *occlusion*) okozza. Ennek két változata létezik, amelyekben közös, hogy az egyik képen egy olyan pont látszik, amely a másik képen takarásban van, vagyis egy térbeli pontnak csak az egyik képen van leképzettje. Ez előfordulhat egy olyan objektum miatt, ami kitakarja a pontot a kamera elől, vagy egyszerűen az egyik pont a kamera látómezején kívülre esik. A következő ábra demonstrálja a takarások legelterjedtebb fajtáit (2.12. ábra).



2.12. ábra Takarások

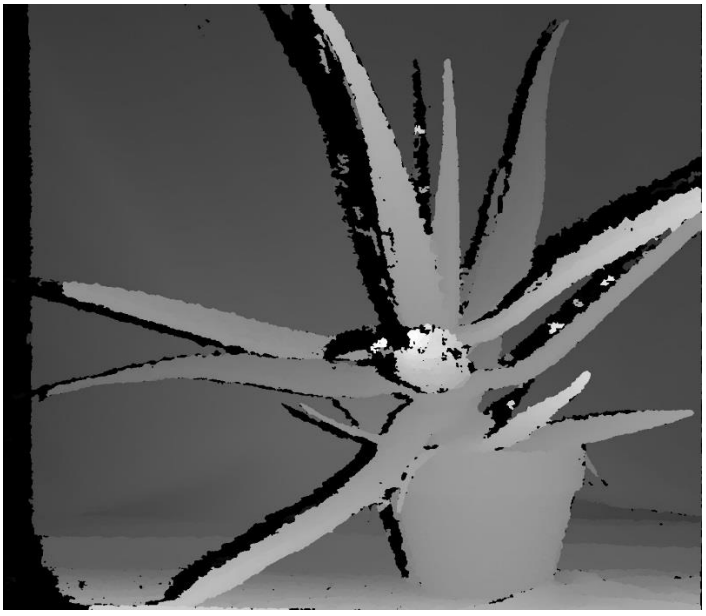
Látható, hogy *P* mind a két kamerán látszódik, így ez a pont megfelelő a diszparitás számolására, azonban *O* csak a bal kamera képén látható mindkét esetben. A jobb oldali képen található a rosszabbik eset, amikor több kisebb részlet takarja a közös pontokat. A sztereó megfeleltetés ezeknél a képeknél általában zavaros, sok olyan pont található, aminek egyáltalán nem határozható meg a diszparitása, ilyenkor hibás értékeket kapnak.

Mivel az algoritmusok a takart pontokhoz is rendelnek diszparitás értékeket, fontos hogy ezeket a hibás értékeket kiszűrjük. Ennek egyik módja a bal-jobb konzisztencia teszt. A teszt során, mint ahogy annak neve is mutatja: először a bal kép diszparitásait határozzuk meg, majd a jobb képét, és az összetartozó pontoknak ugyan azt a diszparitásértéket kell kapniuk abszolút értékben.

2.8 Sztereó megfeleltetés

Egy diszparitásképen a térgeometria úgy van ábrázolva, hogy minél közelebb van egy objektum a kamerához, annál világosabb és minél távolabb van, annál sötétebb. Az ismeretlen diszparitású pontok fekete értéket kapnak. A kiszámolt diszparitás értékeket

átskálázom intenzitásértékekké és azokat jelenítem meg. Mivel a nagy diszparitás kis mélységet jelez (fordított arányosság), ezért a fehérebb foltok közelebbi objektumokat jelölnek. A diszparitásképek elkészítésére több módszer létezik. Ezek két fő részre vannak osztva: lokális és globális diszparitás számítási módszerek. A lokális módszer neve onnan ered, hogy egyszerre csak kisebb ablakokra koncentrál a képeken belül az algoritmus és így keresi a minimum eltérést a pixelkörnyezetekben. A globális módszerek pedig az egész képen keresik az összetartozó pixeleket. Esetemben a globális módszerek szóba sem jöhettek, mert hatalmas erőforrásigényeik vannak és rendkívül lassúak. A tesztjeim során kipróbáltam egy globális módszert, ami percekig futott egy teljes HD felbontású képen és több mint 8GB ram-ot vett igénybe a számításokhoz. Eredménye azonban nem lett jobb, mint a lokális módszereké. Megjegyezném, hogy ezek a módszerek csak standard elrendezés esetén alkalmazhatóak. A következő (2.13-as) ábrán látható egy általam létrehozott diszparitás kép.



2.13. ábra Blokk-illesztéssel létrehozott diszparitás kép

A blokk-illesztés (*block matching*) a legelterjedtebb mód a diszparitás számítás területén, amit annak köszönhet, hogy egyszerű és a többi algoritmushoz képest gyors. Az epipoláris geometriánál említett egyenesekre érvényes függőségeket használja fel a számítás során. Egy rasztersorban egyenként végigmegy az egyik kép pixelein, azokra egy-egy meghatározott méretű ablakot téve. Ennek az ablaknak az egyetlen megkötése, hogy páratlan darabszámú pixelből álló oldalhosszúságúnak kell lennie (például 3x3 pixeles). Minden pixelhez párkeresést végez úgy, hogy a másik kép ugyanazon

rasztersorában végigmegy egy ugyanakkora méretű ablakkal, amellyel végigvizsgálja a sor pixeleit és környezetüket, hogy egyeznek-e a bal képen lévő ablakéval. A legjobb találatot jelöli ki párnak, és annak rasztersorbeli helyzetéből kivonja a bal képen lévő pixel sorbeli helyzetét, ezzel megkapva a diszparitást. A legjobb találat meghatározásához több számítási módszer közül választhatunk. Ezek közül a legegyszerűbb a SAD módszer (*Sum of Absolute Difference*). Az algoritmus veszi az ablakon belülre eső pixelek intenzitásértékeit, és összeveti őket a bal kép ablakában a megfelelő pixelek intenzitásértékeivel, majd a különbségek abszolút értékeit összegzi. Ahol ez a legkisebb, az a megfelelő találat. A következő ábrán (2.14. ábra) láthatóak az alkalmazott mérőszámok, melyekkel a legkisebb eltérést határozzák meg a pontok környezete közt.

Gyakran használt még az NCC (*Normalized Cross Correlation*) módszer is, mert érzéketlen a képek radiometrikus (sugár irányú) torzításaira és eltolódásaira.

MATCH METRIC	DEFINITION
Normalized Cross-Correlation (NCC)	$\frac{\sum_{u,v}\left(I_1(u,v)-\bar{I}_1\right)\cdot\left(I_2(u+d,v)-\bar{I}_2\right)}{\sqrt{\sum_{u,v}\left(I_1(u,v)-\bar{I}_1\right)^2\cdot\left(I_2(u+d,v)-\bar{I}_2\right)^2}}$
Sum of Squared Differences (SSD)	$\sum_{u,v}\left(I_1(u,v)-I_2(u+d,v)\right)^2$
Normalized SSD	$\sum_{u,v}\left(\frac{\left(I_1(u,v)-\bar{I}_1\right)}{\sqrt{\sum_{u,v}\left(I_1(u,v)-\bar{I}_1\right)^2}}-\frac{\left(I_2(u+d,v)-\bar{I}_2\right)}{\sqrt{\sum_{u,v}\left(I_2(u+d,v)-\bar{I}_2\right)^2}}\right)^2$
Sum of Absolute Differences (SAD)	$\sum_{u,v}\left\lvert I_1(u,v)-I_2(u+d,v)\right\rvert$
Rank	$\sum_{u,v}\left(I_1'(u,v)-I_2'(u+d,v)\right)$ $I_k'(u,v)=\sum_{m,n}I_k(m,n)<I_k(u,v)$
Census	$\sum_{u,v}HAMMING\left(I_1'(u,v),I_2'(u+d,v)\right)$ $I_k'(u,v)=BITSTRING_{m,n}\left(I_k(m,n)<I_k(u,v)\right)$

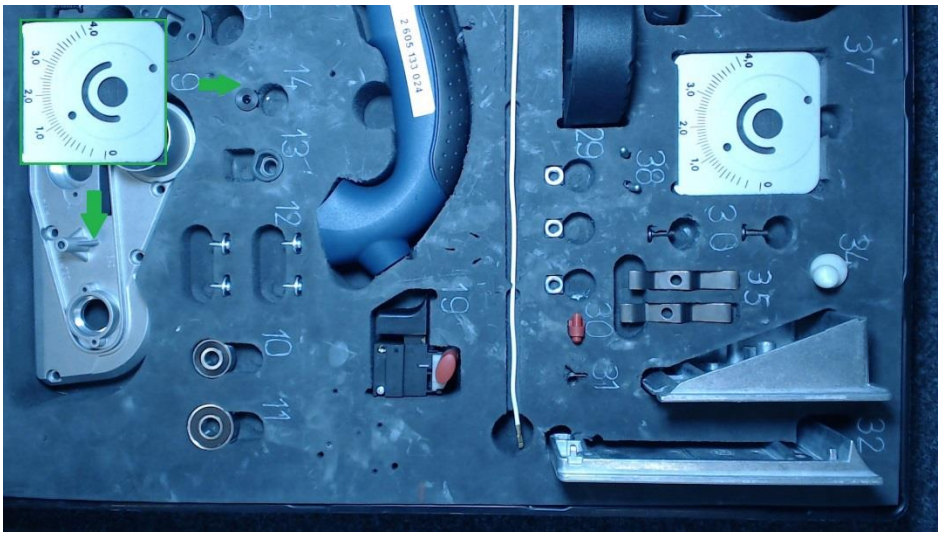
2.14. ábra Blokk-illesztés algoritmusai[14]

3 Betanított algoritmus

A gépi tanulás talán legegyszerűbb formája megmutatni egy algoritmus számára, hogy mit és azt milyen pontossággal keressen a képen, vagy megtanítani, hogy a kép bizonyos régióin milyen feltételeknek kell teljesülnie az alkatrészek jelenlétéhez/hiányához. Szakdolgozatom során ezzel a két módszerrel kísérleteztem. Mindkét algoritmushoz szükséges a kamera és a tálca statikus pozíciója. Ez azt jelenti, hogy bármely két felvétel között a tálcának a képen ugyanazon a pozíción kell lennie. Természetesen ez nem megoldható, de minél kisebb az eltérés, annál kisebb lesz a képen a zaj és annál nagyobb valószínűséggel találják meg a használt algoritmusok az alkatrészeket. A következő alfejezetekben a template matching-et és néhány morfológiai képfeldolgozásban használt eljárás elméletét fogom ismertetni.

3.1 Template matching

A template matching egy gyakran használt technika a digitális képfeldolgozás területén, amely arra alkalmas, hogy egy képen megkeressünk egy kisebb részletet. Használható ipari körülmények között minőségellenőrzésre, mobil robotok irányítására vagy sarokpont detektálásra. Esetemben az alkatrészek megtalálására a tálcán. Az algoritmus inputja egy forráskép, és egy minta (*template*). A mintához legjobban hasonlító részletet keresi az algoritmus a forrásképen.



3.1. ábra Template matching

Ennek folyamata, hogy a mintát a bal felső saroktól kezdve pixelenként jobbra eltolva mozgatja a forrásképen. Amint elért az első sor végére, egy pixellel lejjebb az új sor elején folytatja. Minden pixelen összehasonlítja a mintát a forrásképpel. Erre az összehasonlításra számos lehetőség közül választhatunk, én a következőt használtam:

$$\boldsymbol{R}(x,y) = \frac{\sum_{x',y'}(\boldsymbol{T}(x',y') * \boldsymbol{I}(x+x',y+y'))}{\sqrt{\sum_{x',y'}\boldsymbol{T}(x',y')^2 * \sum_{x',y'}\boldsymbol{I}(x+x',y+y')^2}} \quad (3.1)$$

Ahol **R** az eredmény képe, **T** a template, és **I** a forráskép. Könnyen észrevehető, hogy az algoritmus egy nagyobb felbontású képnél, egy ahhoz képest kis mintával viszonylag nagy lépésszámú. Vegyünk egy (w, h) felbontású forrásképet és egy (x, y) felbontású mintát. Ekkor az algoritmus lépésszáma n:

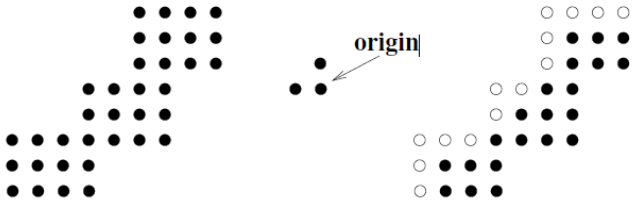
$$n = (w - x + 1) * (h - y + 1) \quad (3.215)$$

Tehát a példa kedvéért: az általam használt 1920x1080-as felbontású képeken egy 50x50-es felbontású alkatrész (ami pont egy anyacsavar mérete) keresése során az algoritmus a 15-ös egyenletet 1929001-szor végzi el.

3.2 Morfológiai műveletek

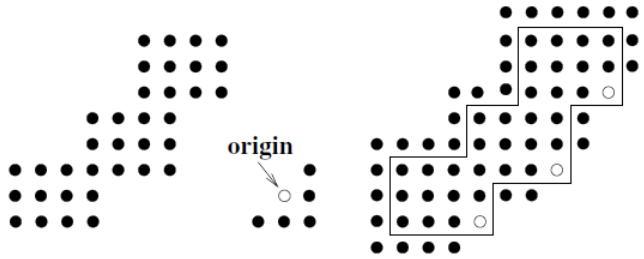
Két morfológiai alapművelet létezik, ezek lineáris kombinációjából képezhető bármely további összetett morfológiai művelet. A két alapművelet az erózió és a dilatáció. Megjegyzendő, hogy a transzformálandó képnek binárisnak kell lennie, ezért ezt a műveletet általában egy bináris thresholdolás előzi meg.

Az erózió során egy úgynevezett strukturáló elem mozog a transzformálandó kép felett. A strukturáló elem általában egy n*n-es négyzet, de akármilyen alakú definiálható. A művelet eredménye 1, ha a strukturáló elem minden pontja egybeesik a transzformálandó kép valamely objektum-pixelével. Egyébként a művelet eredménye 0. A következő ábra (3.2. ábra) közepén látható a strukturáló elem, bal oldalon a transzformálandó kép, jobb oldalon pedig az erodált kép. Az üres körök a törölt pontokat jelölik. Ahogyan az ábrán is látható, erózióval csökkenthető egy alakzat mérete.



3.2. ábra Erózió[13]

A dilatáció nagyon hasonlít az erózióhoz. Eredménye 1, ha a strukturáló elem legalább egy pontja egybeesik a transzformálandó kép objektum-pixelével, egyébként 0. A kontúr az alakzat eredeti helyét mutatja. Látható, hogy törlés ellenére az alakzat nagyobb lett, de elmozdult a bal felső irányba az origó eltolása miatt.



3.3. ábra Dilatáció[13]

Tesztjeim során a képeken megjelenő zajok eltűntetésére a nyitás és zárás műveletét használtam. A nyitás egy eróziót követő dilatáció, a képen lévő kis objektumok eltűntetésére alkalmas. A zárás a nyitás fordítottja, azaz egy dilatációt követő erózió. Lyukak eltűntetésére alkalmas.

Ezeken kívül gyakran használt művelet a morfológiai gradiens, ami a dilatált képből kivont erodált kép. Segítségével az objektumok körvonalai határozhatóak meg[13].

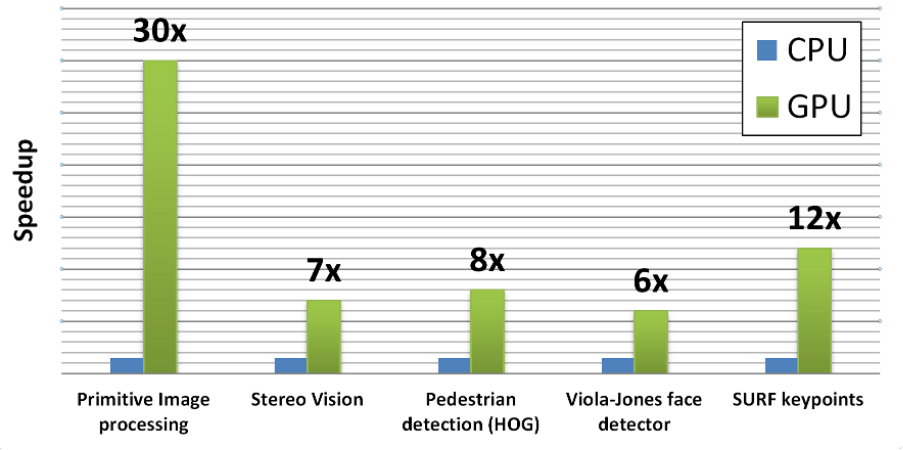
4 Komponensek ismertetése

4.1 Felhasznált szoftverkomponensek és könyvtárak

A felhasznált szoftver és a programnyelv kiválasztása döntő lehet a teljes munka hatékonyságára nézve és fontos a legelején meghatározni, hogy milyen irányba kezdünk fejleszteni. Manapság egyre elterjedtebbek a gépi látással, számítógépes grafikával kapcsolatos alkalmazások, emiatt a képfeldolgozás az elmúlt néhány évben nagy fejlődésen ment keresztül, így megjelentek az erre specializálódott függvénykönyvtárak. A két legelterjedtebb ezek közül az OpenCV és a CUDA, amelyek C illetve C++ programnyelvhez készültek. A két függvénykönyvtár között sok a hasonlóság. Az egyik jelentős eltérés, hogy míg az előbbi tetszőleges processzoron (akár CPU vagy GPU), az utóbbi csak Nvidia grafikus processzoron futtatható. Ebből következik, hogy az OpenCV széles körben használható, bár ebből kifolyólag kicsit lassúbb, a CUDA jobban optimalizált, gyorsabb működést tesz lehetővé. Az OpenCV mellett szólt még az a tény, hogy szakmai gyakorlatom során már foglalkoztam néhány egyszerűbb felhasználási területével. Ezen szempontok figyelembevételével az OpenCV mellett döntöttem.

4.1.1 OpenCV

Az OpenCV (*Open Source Computer Vision Library*), ahogy a neve is mutatja, egy nyílt forráskódú, szabadon felhasználható függvénykönyvtár, amit az Intel fejlesztett ki 1999-ben valós idejű gépi látás elősegítésére. Van C, C++, Python, Java és Matlab interfésze, emellett támogatja a legtöbb Windows verziót, Linux, Android és Mac operációs rendszert. Elsősorban gépi látással kapcsolatos és képfeldolgozási feladatokat ellátó rendszerek fejlesztéséhez nyújt segítséget, amelyek nagy erőforrást igényelnek. Ezekhez szükséges volt valamilyen megoldást találni a gyorsabb adatfeldolgozásra. A CPU (*Central Processing Unit*) alapú programozást kívánták kibővíteni a GPU (*Graphics Processing Unit*) felhasználásával. A GPU-k segítségével gyorsabb számítás lehetséges legfőképp azokon a területeken, ahol a számítások nagy részét mátrixműveletek adják, mivel sokprocesszoros rendszer és párhuzamos feldolgozást is lehetővé tesz. A következő ábrán (4.1. ábra) látszik egy 2011-ben megjelent felsőkategóriás CPU és GPU futásidő alapján történő összehasonlítása, amely egyszerűbb és bonyolultabb képfeldolgozási algoritmusok elvégzését foglalja magában.

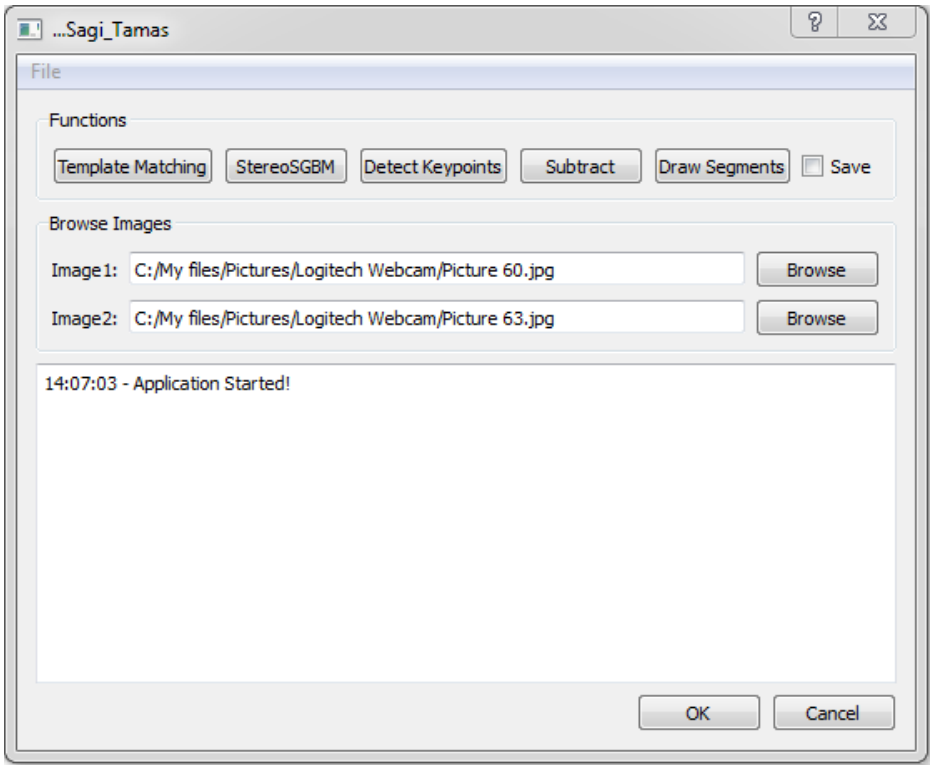


4.1. ábraTesla C2050 GPU vs Core i5-760 2.8Ghz CPU[7]

Számomra külön hasznos, hogy az alapvető szűrőkön és algoritmusokon túl a háromdimenziós rekonstrukcióhoz és kamera kalibrációhoz is implementál néhány széles körben használt eljárást és emellett számos előre megírt mintakód található az OpenCV hivatalos GitHub oldalán[1].

4.1.2 Qt

Az általam megírt algoritmusokhoz készítettem egy egyszerű grafikus felhasználói felületet (*Graphical User Iterface-t*), így valós időben és sokkal egyszerűbben lehet kiválasztani a tesztképeket, megváltoztatni az algoritmusok paramétereit, amik így már nincsenek előre definiálva (*hardcode-olva*). Elsősorban az egyszerűségre törekedtem, hogy minél kevesebb időt vegyen el a GUI fejlesztése az algoritmus fejlesztésétől, ezért először Windows Forms-al szerettem volna megoldani, ami a Visual Studio-ban alapból megtalálható. Sajnos az OpenCV-vel együtt nem lehet Windows Forms-ot használni, ezért a Qt-t választottam, amely széles körben használt, felhasználó felület fejlesztésére alkalmas függvénykönyvtár[11]. Elterjedtsége annak köszönhető, hogy multi platform fejlesztésre alkalmas, azaz a fejlesztést végezhetjük Windows operációs rendszeren, majd futtathatjuk Linuxon vagy a többi támogatott operációs rendszerek egyikén. Az 5.2-es verziótól már Android és IOS támogatással is rendelkezik. A fejlesztés nyelve is széleskörű, lehet C++, Python, Java, Perl, Ruby és még sok más nyelven is lehet fejleszteni. Nem utolsó sorban az elmúlt egy éves munkatapasztalatom során gyakran kellett használnom a Qt-t, ezért is döntöttem emellett. Az elkészült felhasználói felület a következő ábrán látható (4.2. ábra):



4.2. ábra Az általam készített applikáció kinézete

A felhasználói felület segítségével egy gombnyomásra elindíthatóak a különböző algoritmusok a kiválasztott képekre. Bármelyikre rákattintva megjelenik egy újabb ablak, ahol megadhatóak az adott algoritmus paraméterei. A bepipálható „Save” gomb segítségével elmenthetőek az eredmények képei. Az alsó „Log” sávban kiírja, hogy meddig futottak az egyes alkalmazások, éppen hol tart a futásban.

4.1.3 CMake

Fejlesztéshez a Microsoft Visual Studio 2010 64 bites, a Qt 4.8.6-os és az OpenCV 2.4.13-as verziója állt rendelkezésemre. Fontosnak tartom megemlíteni a CMake nevű programot. Ha forráskódunk több fájlból áll, esetleg külső programkönyvtárakat is felhasználunk, a fordítási/linkelési parancsok és a futtatható program előállításának lépései bonyolulttá és hosszadalmassá válnak. Aki használt már OpenCV-t tudja, hogy egy függvénykönyvtár linkelése milyen bonyodalmakkal jár és mennyi időt el tud venni a fejlesztéstől. Ráadásul, ha valamelyik programkönyvtár vagy a fejlesztőkörnyezet verziója vagy elérési útvonala megváltozik, az egészet kezdhetjük az elejéről. Ezen problémák megkönnyítésére való a CMake, ahol egy CMakeLists.txt nevű file-ban megadhatjuk a használni kívánt függvénykönyvtárak elérési útvonalát (ez csak akkor szükséges, ha azok nem lettek korábban hozzáadva a Windows környezeti

változóihoz), az általunk használt header és source file-okat. Fejlesztésemet a saját CMakeLists.txt file-om létrehozásával kezdtem.

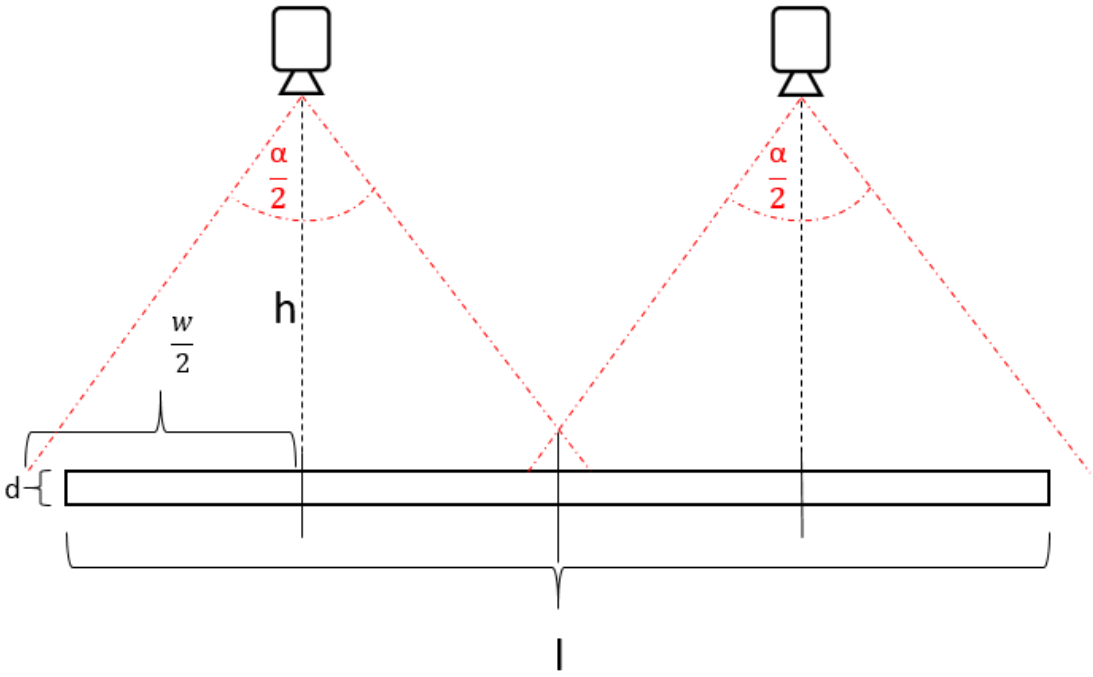
Fejlesztésem során laptop cserére került sor, ezért tesztelhettem is a CMake előnyeit. Így a meglévő CMakeLists.txt file használatával az új projekt létrehozása, a függvénykönyvtárak linkelése nem órákat, hanem néhány másodpercet vett igénybe.

4.2 Felhasznált kamera

Szempontomból egy kamerának három fontos paramétere van: felbontás, látószög és az automatikus funkciók állíthatósága. Manapság már a legegyszerűbb webkamerák is rendelkeznek autófókusz, autófényerő kiegyenlítés funkcióval, ami számomra felesleges, sőt teljesen elrontja a kamera kalibrációt, mivel a fókusztávolság változásával megváltozik a kamera belső paramétere. Így ha két fénykép készítése között megváltozik a kamera fókusztávolsága, újra kéne kalibrálni. Ezt a problémát a 2.3-as fejezet során már körüljártam. A felbontás és a látószög pedig azért fontos paraméter, mert ezek határozzák meg, hogy az elkészült fényképen egy pixel milyen távolságnak felel meg a valóságban. Két ugyanolyan felbontású kamera közül, amik ugyanolyan távol vannak a fényképezendő objektumtól, azok közül a kisebb látószögű több pixelen tárolja az objektumot. Esetemben pedig ez azért fontos, mert egy elég nagy (fél méterszer egy méteres) felületen kell megtalálnom viszonylag kis alkatrészeket. A legkisebb és egyben legtöbb problémát okozó alkatrész egy apró (fél centiméterszer fél centiméteres) rugó, így mindent ehhez mérve választottam.

Kezdetben rendelkezésemre állt egy Nikon Coolpix L820-as típusú fényképezőgép, amivel tesztelhettem, hogy milyen kamerára lesz szükségem a feladat megoldásához. Ez a kézi kamerákhoz képest nagyon jó tulajdonsággal rendelkezett, 4608x3456–os felbontású (16 megapixel) fényképek készítésére is alkalmas. Sajnos nem rendelkezett MF (*Manual Focus*) funkcióval és nem lehetett szoftveresen vezérelni, csak manuálisan egy gomb megnyomásával. A következő fejezetben tárgyalom, hogy ez miért szükséges. A tesztelések során arra a döntésre jutottam, hogy az összeszerelő tálcát két részre osztom, ehhez viszont szükséges a teljes HD felbontású (1920x1080) fényképezőgép, mert ha kisebb felbontásút választanék, nem látszódnának a kis alkatrészek. Így a választás egy Logitech C920 HD Pro webkamerára esett, amely minden fent említett elvárásnak megfelel: szoftveresen ki/be kapcsolható automatikus funkciók, 78°-os látószög, teljes HD videó és fénykép készítésére is alkalmas. Megjegyzendő, hogy

a kamera specifikációjában található látószög szélességi látószög. A kamerák elrendezése a következő ábrán látható (4.3. ábra).

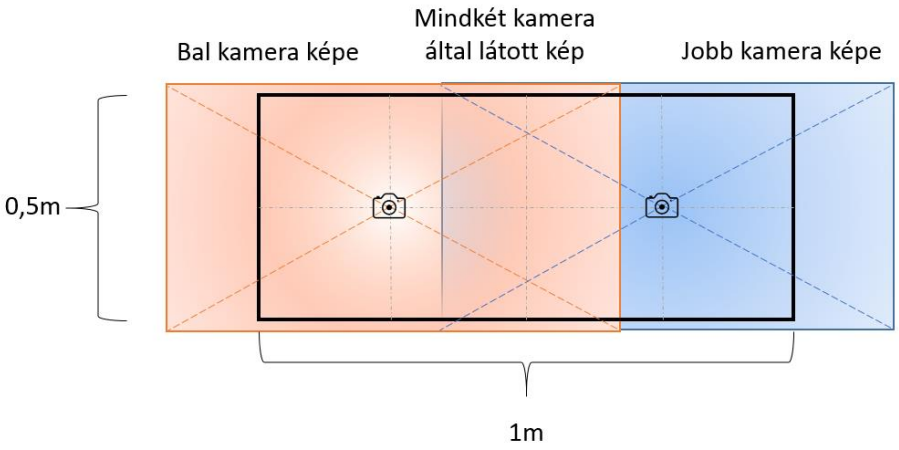


4.3. ábra A kamerák elrendezése oldalnézetből

Az ábrán *w*a kamera által készített kép szélessége, *h* a kamera magassága a tálcától, *α* a látószög, *l* a tálca hossza, *d* pedig a tálca magassága. Figyelembe véve a 78°-os látószöget, a kamera 16:9-es felbontását és hogy két kamerával szeretném lefedni a fél méterszer egy méteres tálcát, felírhatóak a következőkre egyenletek:

$$\tan\left(\frac{\alpha}{2}\right) = \frac{\frac{w}{2}}{h}$$
$$w * \frac{9}{16} \geq 50cm \tag{16}$$

Az első egyenlet egy egyszerű tangensfüggvény felírásából adódik a kamera által belátott téglalap alapú gúlából képzett derékszögű háromszögre. A második egyenlet azt írja le, hogy a kamera által készített kép téglalapjának hosszabb oldala hogyan viszonyul a rövidebb oldalához, azaz a tálca mélységéhez, ami fél méter. A kamerákat a tálca negyedelő vonalaihoz téve, a fenti egyenleteteket megoldva *h*-ra, az adódik, hogy a kamerákat legalább 65*cm* magasra kell tenni, hogy két kamerával belátható legyen az egész tálca szélességében és hosszúságában is. Így azonban lesz egy olyan tartomány, amit mindkét kamera belát, ezt szemléltetem a következő ábrán (1.1. ábra).



4.4. ábra A tálca felülnézetből a két kamerával.

4.3 Teljesítményjelzők

A teljesítményjelzők (az angol irodalomban: *Key Performance Indicator*) egy projekt eredményeinek mérésére alkalmasak és ezeket célszerű a fejlesztések elkezdése előtt tisztázni, hogy mindenki számára egyértelműek legyenek az eredmények. Mindkét bemutatott módszer során egy adott alkatrész meglétét veszem pozitív visszajelzésnek. Így négy különböző kimenet lehetséges, amik az alábbiak:

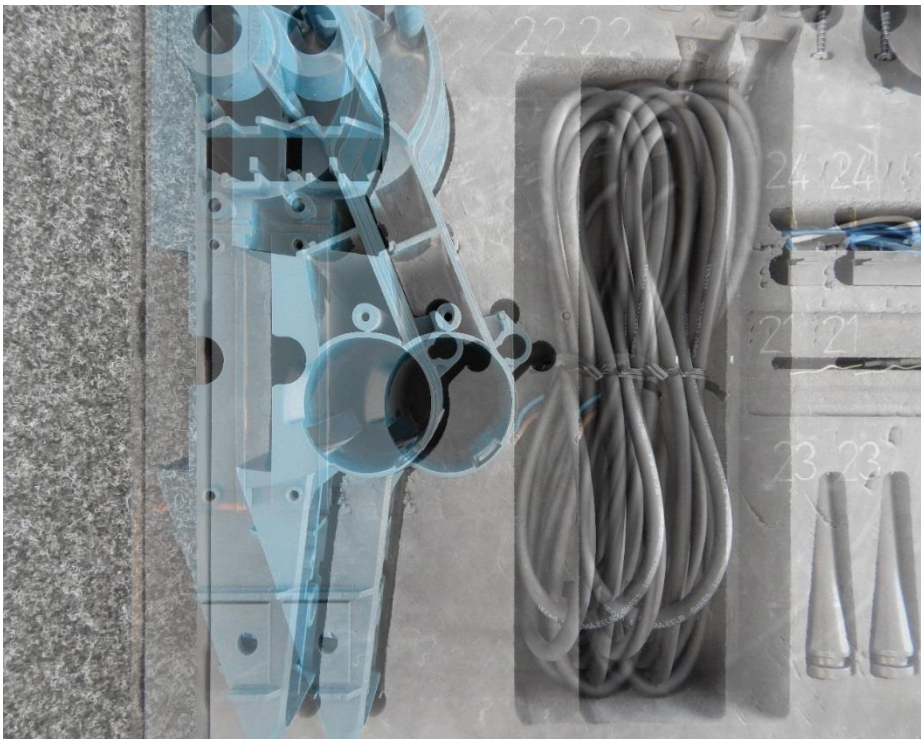
	Az algoritmus megtalálta	Nem találta meg
Ténylegesen ott van	True Positive	False Negative
Ténylegesen hiányzik	False Positive	True Negative

4.5. ábra Teljesítményjelzők

5 Eredmények

5.1 Sztereó

A második fejezet során említett elméleti lépéseket a következő sztereó képpáron (5.1. ábra) fogom bemutatni, amiket egymásra rajzoltam a jobb láthatóság kedvéért. A kamera 30*cm* magasan volt és 2*cm*-es bázisvonalat használtam.



5.1. ábra Sztereó képpár

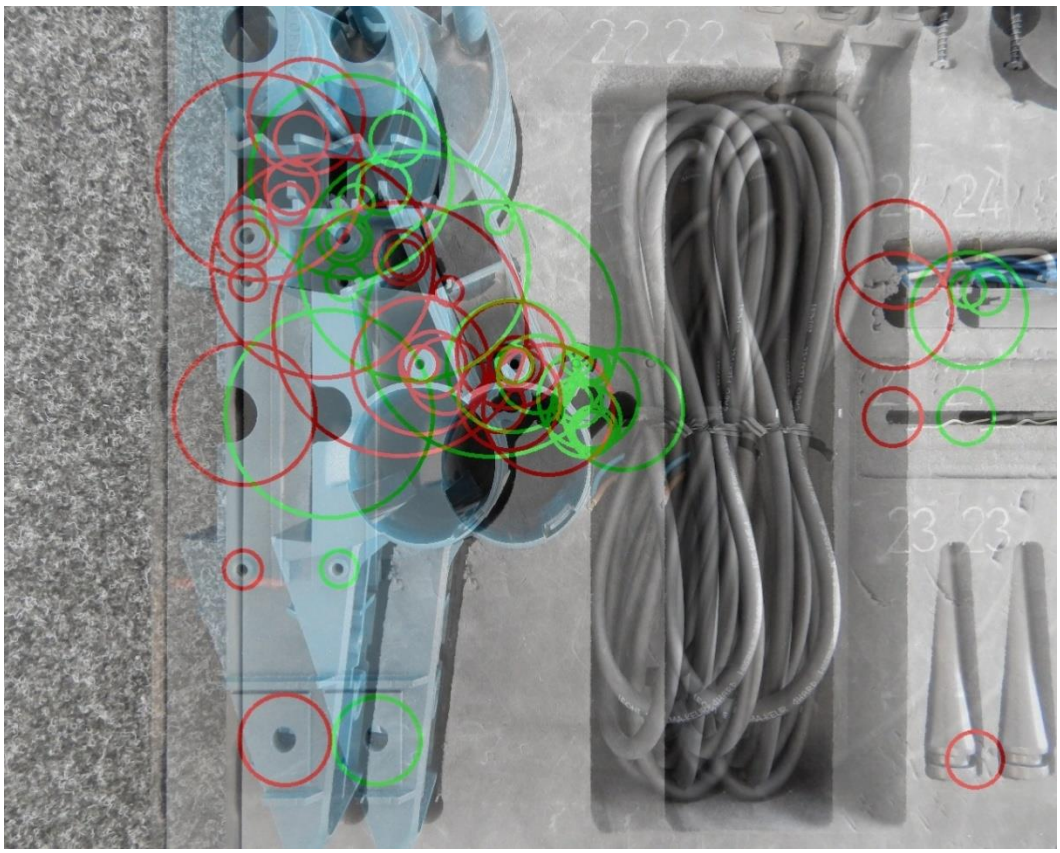
A képen nagyon jól látszik, hogy mekkora bázisvonallal dolgoztam és a bevezetőben említett anagliph képek hatása is megjelenik (a ciánkék – vörös szín nélkül). Első lépésként rektifikálom a képeket, majd blokk-illesztéssel meghatározom a diszparitását. Ahhoz, hogy rektifikáljam a képet, ismernem kell az összetartozó pontokat (*feature* pont) a két képen.

5.1.1 Feature pontok

Feature pontoknak nevezzük a két képen ugyanahhoz a világbeli ponthoz tartozó pontokat. Minden sztereó megfeleltetésen alapuló algoritmus ezeket keresi a képen, azonban nem vizualizálja azokat. Készítettem egy egyszerű kódrészletet, amivel

kirajzoltathatom az összetartozó pontpárokat, így sokkal szemléletesebbé válik, hogy mi történik diszparitás számolás közben a háttérben.

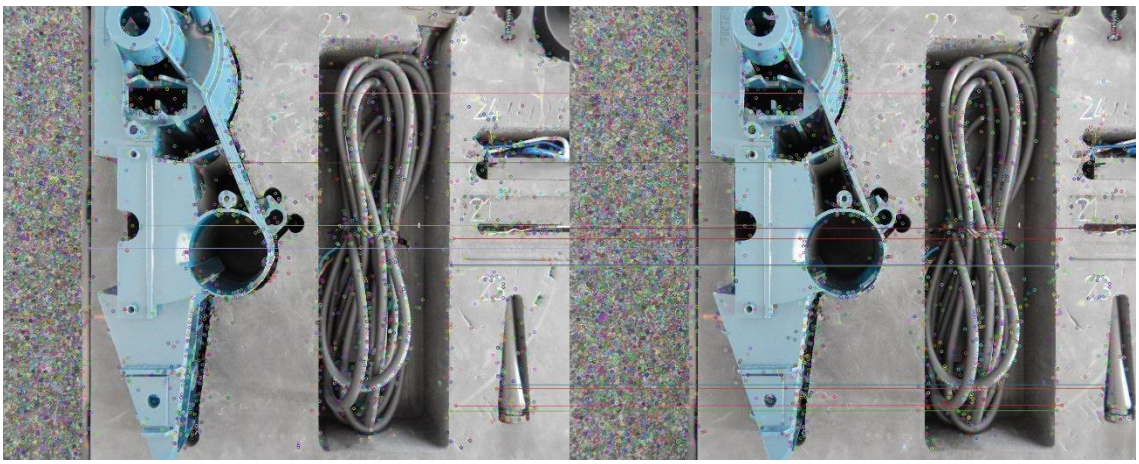
Ennek első lépése, hogy megkeresem az összes lehetséges összetartozó pontot mindkét képen. Az OpenCV-ben erre egy külön osztály található: a *SurfFeatureDetector*. Az algoritmus minden találathoz rendel egy erősséget, ezt jelképezi a körök sugarainak nagysága. Minél nagyobbak a körök sugarai, annál jobbnak számít az adott pont. Az algoritmus a két képen külön-külön rengeteg (több mint ezer) feature pontot talál, de ezek közül csak a 30 legjobbat jelenítettem meg. A két kamera képét ismét egymásra rajzolva a következőt kaptam (5.2. ábra):



5.2. ábra Feature pontok, a bal és jobb oldali képen egymásra rajzolva

A bal oldali képen pirossal, a jobb oldalin zölddel jelöltem a feature pontokat. Látható, hogy már a 30 legjobbnak számító pont között sincs mindnek párja. Ez abból adódik, hogy a szivacs nagyon matt, és a háttérben lévő szőnyeg nagyon zajos. Azonban az algoritmus a jobb alsó sarokban lévő kúp alakú alkatrészhez is csak a bal képen talált feature pontot. Valószínűleg azért mert a jobb oldali képen már nem látszik az érdekes részlet.

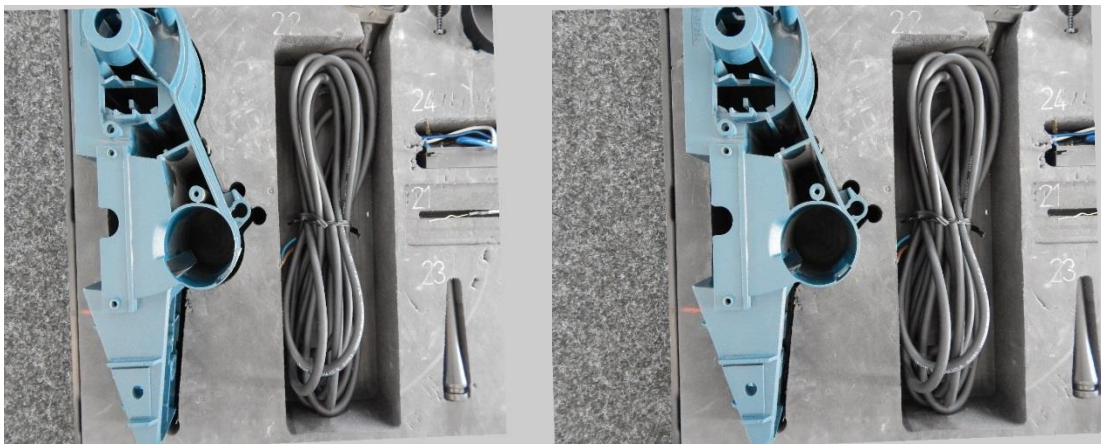
Ezután egy szűrési eljárás következik, amely megkeresi a ténylegesen összetartozó pontokat. Erre két különböző módszert is kipróbáltam, amik implementálva vannak az OpenCV-ben. Az egyik a Bruteforce nevezetű, a másik pedig a Flann-algoritmus. A Flann-módszer (*Fast Approximate Nearest Neighbor*), mint ahogyan a neve is mutatja, a legközelebbi szomszédokat keresi a képeken. A Flann-módszer sokkal jobbnak bizonyult, mint a Brutefirce, így csak ennek az eredményeit mutatom, ahol már nem csak a 30 legjobbnak számító pontot jelenítem meg, hanem mindet.



5.3. ábra Flann algoritmus találatai

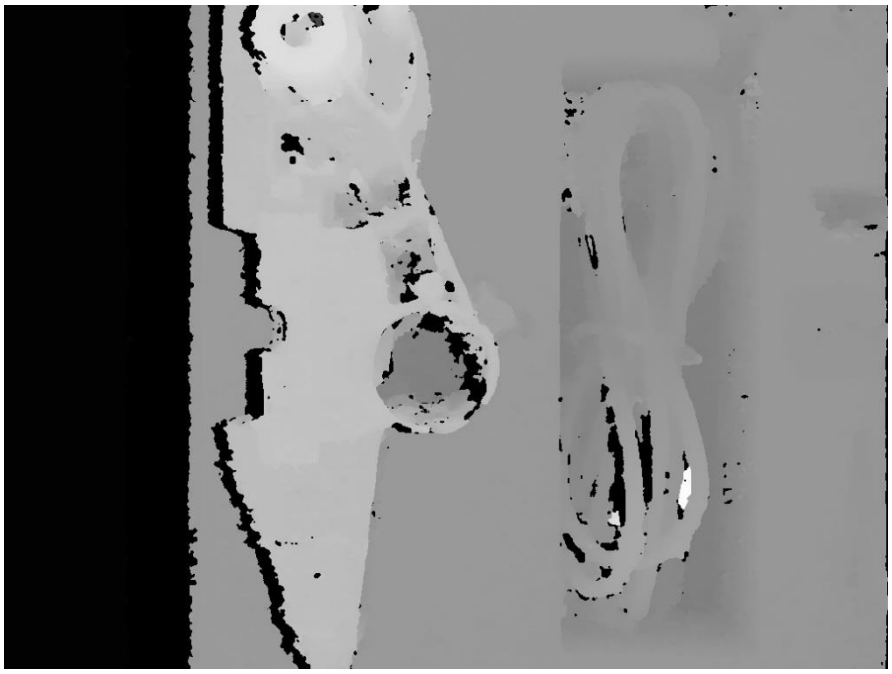
Megfigyelhetőek a fent említett hiányosságok. A szőnyegen egy összetartozó pont sem található, a szivacson pedig még pár nélküli is alig. Látható, hogy külön-külön számos feature pont található a képeken, azonban, a Flann-módszerrel ezek közül összesen 12 összetartozót sikerült találnom, ami nagyon kevésnek számít. Ezeket egyenesekkel kötöttem össze a jobb és bal kamera képén. Előfordult, hogy ezek az egyenesek nem lettek horizontálisak a két képen, így szűkítettem a maximális ponttávolság sugarát. Azért fontos, hogy elkerüljük a nem horizontális vonalakat, mert ezek teljesen elrontják a rektifikációt.

Az összetartozó pontpárok segítségével már meghatározható a fundamentális mátrix, ami a sztereó képek közötti epipoláris kényszer kifejezésére alkalmas. Ennek ismeretében már csak egy mátrixszorzás meghatározni a keresett sík-sík leképzést a rektifikációhoz. Az algoritmus szempontjából ez két függvény meghívását jelenti, amik a *findFundamentalMat* és a*stereoRectifyUncalibrated*. Ezzel megkaptam azt a sík-sík leképzést, amely segítségével megkaptam a rektifikált képeket (5.4. ábra).



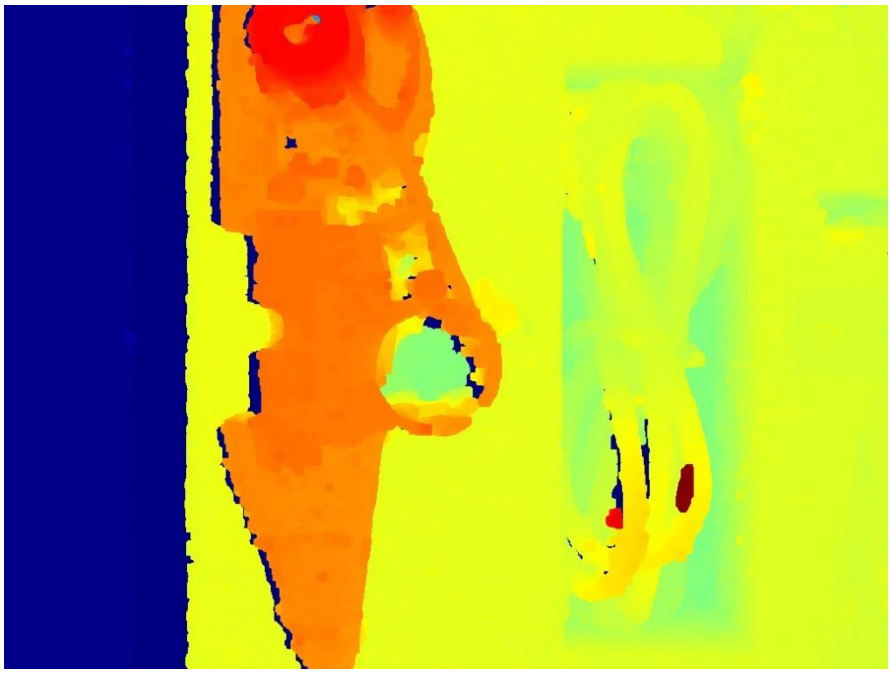
5.4. ábra A bal és jobb kamera rektifikált képei

A rektifikált képeken már elvégezhető a blokk-illesztés. Erre található az OpenCV-ben a *StereoBM* és *StereoSGBM* osztály. Míg az első gyorsabb és kissé pontatlanabb diszparitás képet ad, addig a második sokkal lassabb és szinte ugyan olyan eredményt sikerült elérnem vele.



5.5. ábra A tálcáról készült diszparitás kép

Látható, hogy néhány helyen zajos a kép, ott az algoritmus nem talált diszparitásértékeket a pixelekhez. Ezeket dilatációval javítottam. Hogy az emberi szem számára jobban felfoghatóak legyenek az intenzitásértékek, colormap-et rendeltem a diszparitás képekhez. Így a kamerához közelebb lévő pontok pirosak, míg a távolabbiak kékes árnyalatba mennek át.



5.6. ábra Dilatált, színezett diszparitás kép

5.2 Betanított algoritmus

5.2.1 Régiókra bontás

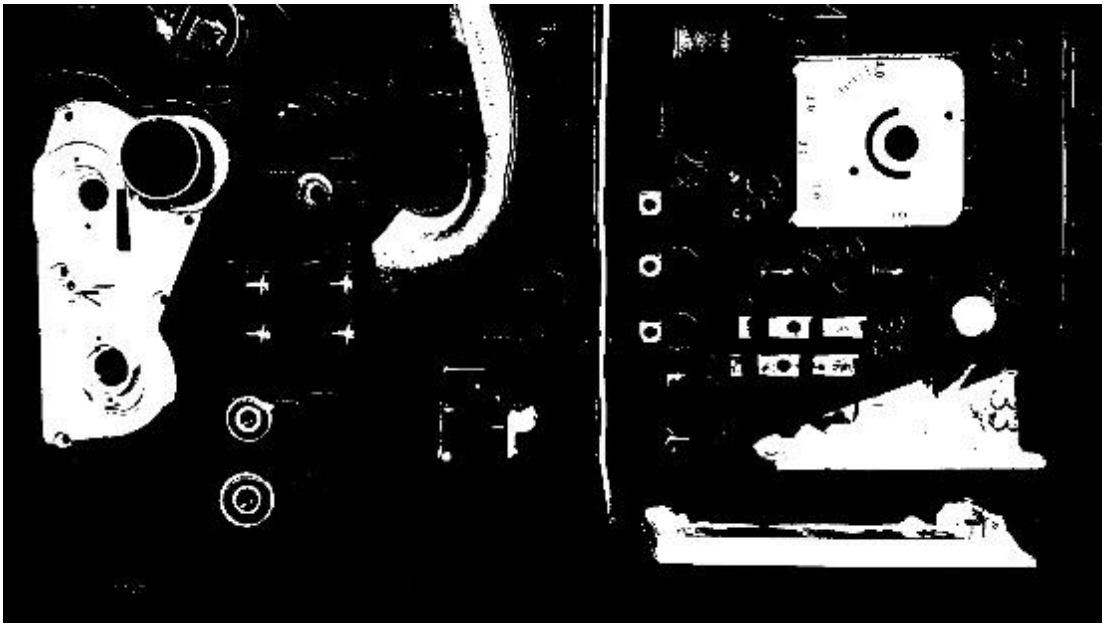
Mint azt a harmadik fejezetben említettem, az általam használt módszerekhez szükség van a forrásképen keresett alkatrészek pozíciójára és mintájára. Kezdetben ezeket szoftveresen szerettem volna szegmentálni a háttértől, de ennek a bonyolultsága magában egy külön szakdolgozat témája lehetne. Az elképzelésem az volt, hogy az alkatrészekkel teli tálcáról készült képet kivonom a teljesen üres tálcából, így megkapva az alkatrészek maszkját.

Első lépésként a teli és üres tálcáról készült RGB képeket szürkeárnyalatossá konvertálom, így innentől intenzitásokkal lehet számolni, nem pedig az RGB értékek megváltozásával. A konvertálás után a képeket kivonva egymásból, a következőt kaptam (5.7. ábra):

5.7. ábra Egyszerű különbségi kép

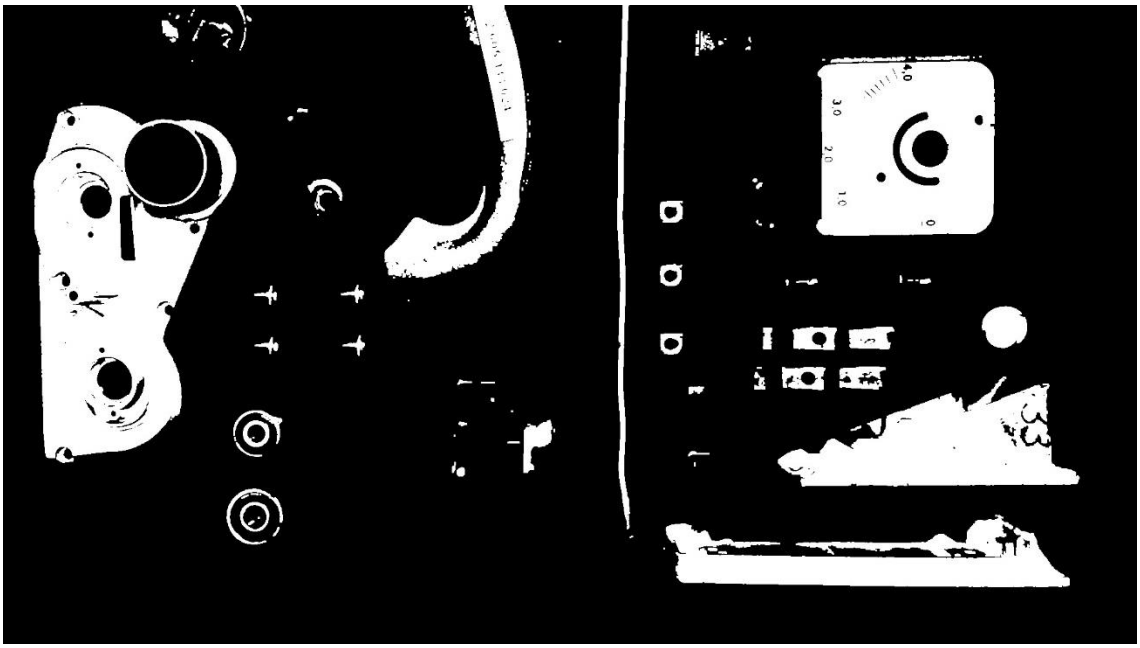
A különbségi képen láthatóan elkülönülnek az alkatrészek, azonban nem odaillő zajok is megjelennek. Ezek a kamera pontatlanságából, a tálca elmozdulásából adódnak. A zajok szűrésére számos hatékony megoldás létezik. Első szűrési eljárásként a zajok eltűntetésére egy bináris thresholdolást alkalmaztam. Ennek folyamata, hogy az algoritmus megvizsgálja a kép összes pixelét, és ami egy bizonyos határérték (*threshold*)

alatt van, annak nulla értéket ad (fekete), ami felette annak pedig fehéret. A threshold értéke alkalmazástól, fényviszonyoktól, háttértől függően más és más, ezt nekünk kell tesztek során meghatároznunk, nincs egy univerzálisan használható érték. Én az itt látható szekvenciák során 40-es thresholdot használtam, mert efelett már alig láthatóak a háttérhez hasonló színű alkatrészek, ezalatt pedig túlzottan zajos a kép. A következő ábrán (5.8. ábra) látszik a thresholdolt különbségi kép.



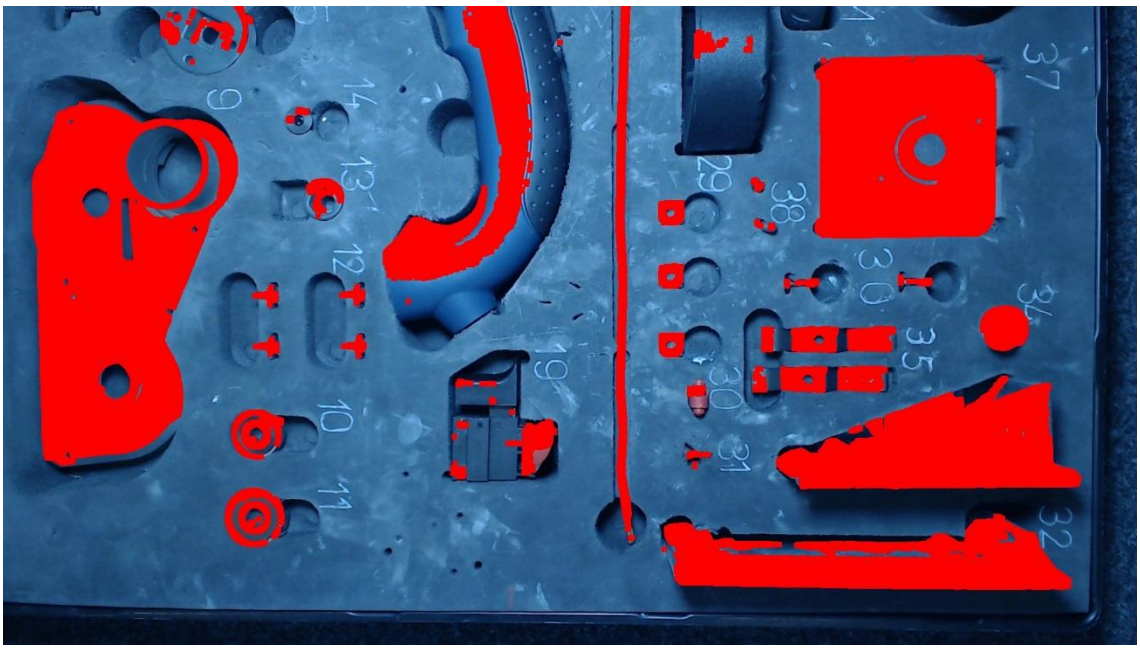
5.8. ábra Thresholdolt különbségi kép

Látható, hogy a folyamat során a zajok nagy része eltűnt, ezt azonban lehet még javítani. A megmaradt zajokat a morfológia képfeldolgozásban használt nyitás művelettel tűntettem el. A nyitás két lépésből álló művelet, amik az erózió és a dilatáció. Ennek fordítottja a zárás. Strukturáló elemnek 5*5-ös négyzetet választottam, így az egy pixelnél nagyobb zajok is eltűntethetőek vele, viszont nem túl nagy ahhoz, hogy a kisebb alkatrészek is törlésre kerüljenek. Végeredményképpen csökkenti a zajokat a képen. Mellékhatásként a lényeges alkatrészek mérete is csökken, ám ez dilatációval javítható. Az így kapott kép (5.9. ábra):



5.9. ábra Az alkatrészek nyitás után

A nyitás után már egészen láthatóvá válnak az alkatrészek. Ahhoz hogy növeljem az alkatrészek méretét, végrehajtottam egy dilatációt. Így megkapva a végleges különbségi képet, azt visszarajzolva az eredeti, alkatrészekkel teli képre, a következőt kaptam (5.10. ábra):

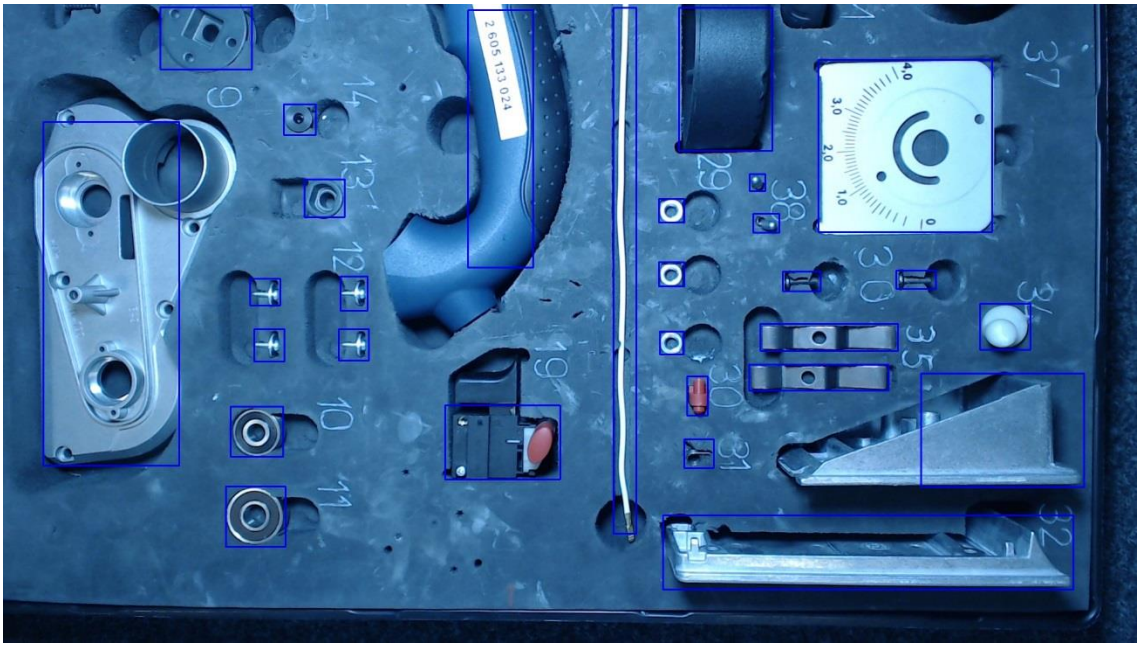


5.10. ábra A különbségi képpel kapott alkatrészek elkülönítése, visszarajzolva az eredeti képre

Látható, hogy a háttértől jól elkülöníthető alkatrészek jól látszanak, míg a műanyagoknak csak részletei. Az alkatrészek szoftveres elkülönítése a háttértől ezzel a módszerrel azért nehezen megoldható, mert ahol ezek az alkatrészek vannak, ott

elhanyagolható lesz az intenzitásváltozás. Már a különbségi kép létrehozásánál eltűnik a problémás alkatrészeknek bizonyos százaléka, és ezen a thresholdolás csak ront. Megpróbáltam a megvilágításon változtatni, remélve a műanyag alkatrészek jobb láthatóságát, azonban nem jutottam jobb eredményre, ezért döntöttem a kézi szegmentálás mellett.

Erre készítettem egy egyszerű scriptet, amely megjeleníti az alkatrészekkel teli tálcát, majd kézzel behúzhatóak az alkatrészeket befoglaló négyzetek. Az összes alkatrész megjelölése után az algoritmus egy xml file-ba menti a befoglaló négyzetek pozícióját/méretét és elmenti a mintának szánt képeket. Ez a megoldás azért előnyös, mert így nem csak az alkatrészek template-jeit kapjuk meg, hanem azok elvárt pozícióját is. A következő alfejezetben részletezem, hogy ez miért előnyös. Hátránya, hogy egyszer kézzel kell meghatározni az alkatrészek pozícióját. A kézzel történő alkatrész kijelölés a következő (5.11. ábra) ábrán látható.



5.11. ábra Kézzel bejelölt alkatrész pozíciók

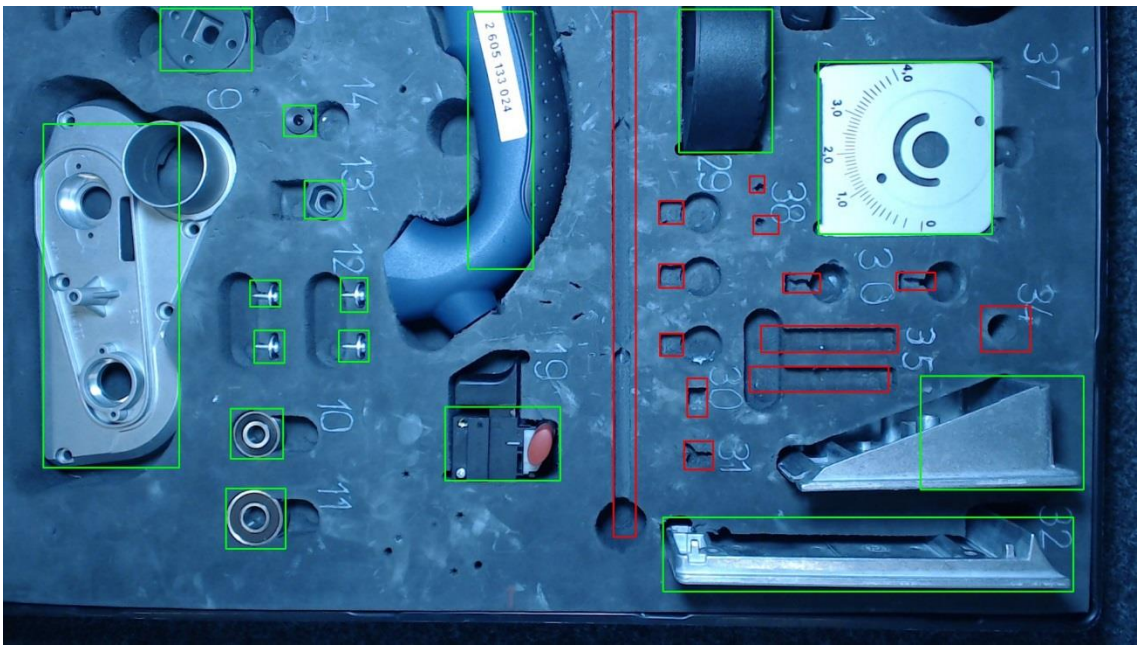
A hasonló alkatrészeknél (mint például egy anyacsavar) akár egy template-tel is lehetne dolgozni. Én azért jelöltem ki mindent külön, hogy ismert legyen az összes alkatrész pozíciója és így nagyobb egyezéssel lehessen keresni a mintákat a képen.

5.2.2 Template matching

A mintákkal már elkezdhettem tesztelni a template matching algoritmus pontosságát. Kezdetben az egész képen kerestem a mintákat, és ez jól is működött, de

mint azt az elméleti összefoglalóban is említettem, ez nagyon időigényesnek bizonyult. A tesztek során az algoritmus körülbelül 37 másodpercig futott, ami semmiképpen sem megengedhető.

Hogy csökkentsem az algoritmus futásidejét, a korábban elmentett minták pozícióját felhasználva egy 50 pixeles keretet húztam az alkatrészek köré és csak ezen a kereten belül keresem az alkatrészeket. Így kevesebb, mint egy másodperc alatt végigfut az algoritmus és még pontosabb is, mivel kisebb az esélye a false positive találatnak. A következő ábrán (5.12. ábra) látszik az algoritmus működése egy olyan tesztképen, amiből véletlenszerűen ki van szedve néhány alkatrész.



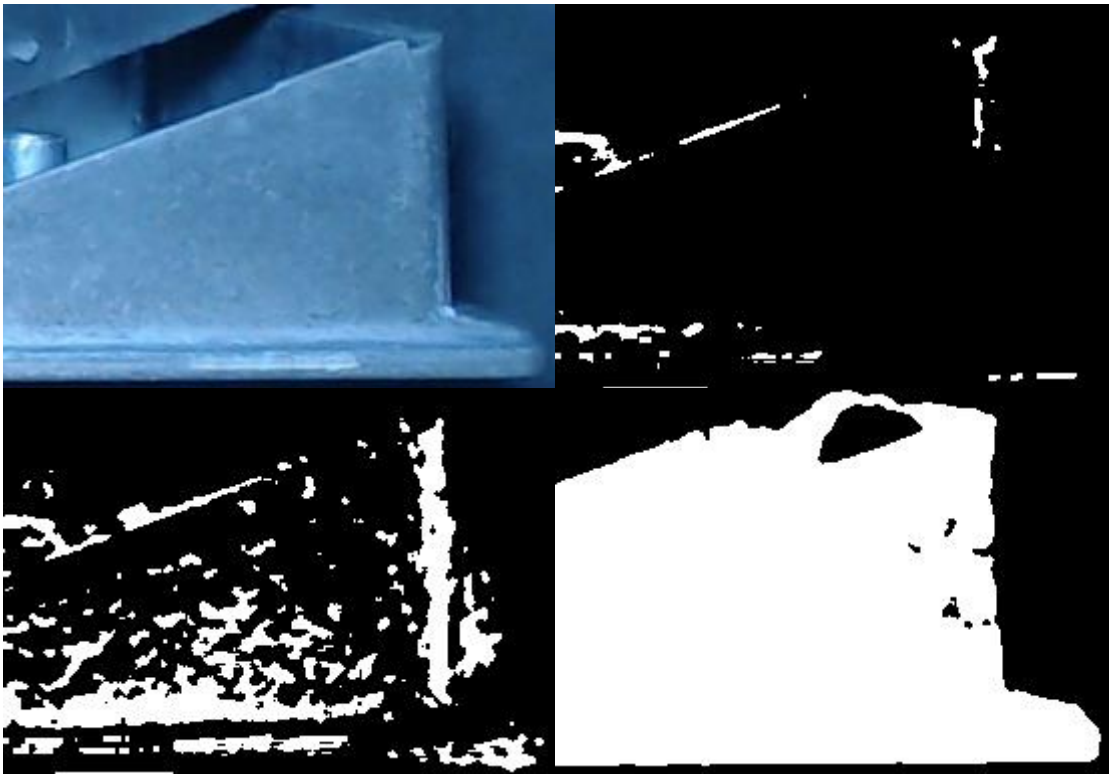
5.12. ábra A template matching futtatása után a megtalált alkatrészek zölddel, a hiányzóak pirossal

A meglévő alkatrészeket zölddel, a hiányzóakat pirossal keretezi be az algoritmus.

5.2.3 Regionális intenzitásváltozás

Szintén statikus kamera és tálca elrendezést feltételezve, az alkatrészek pozícióinak és sablonjainak ismeretében a regionális intenzitásváltozás eljárását alkalmaztam. A vizsgálandó képen csak a korábban megjelölt alkatrész régiókkal foglalkozom. Az alkatrészek régióit kivágtam a vizsgálandó képből, majd ezen régiókból különbségi kép képzésével megnéztem az intenzitásváltozást az ott lévő alkatrészhez képest. Ha az alkatrész hasonló pozícióban van jelen, akkor az intenzitás csak kissé változik meg, ha nincs ott, akkor számottevően. Az intenzitás megváltozását a különbségi képen úgy számolom, hogy összeadom a pixelek intenzitását, majd elosztom a

pixelszámmal. Így egy átlagértéket kapok, ami nem függ a kép méretétől. Ez nagyon hasonlít ahhoz a módszerhez, mint ahogyan el szerettem volna választani az alkatrészeket a háttértől. Ebben az esetben azért használhatóbb ez a megoldás, mert nem függ annyira a környezeti viszonyok megváltozásától, hiszen nem a teljes képpel foglalkozom, csak egyes részeivel. Egy példán keresztül illusztrálom, hogy milyen nehézségek kerülhetnek szóba ennél a módszernél. Három állapotot különböztettem meg: amikor a fényviszonyok változnak, amikor elmozdul az alkatrész, de a helyén van, és amikor hiányzik. Ezek láthatóak a következő ábrán (5.13. ábra):

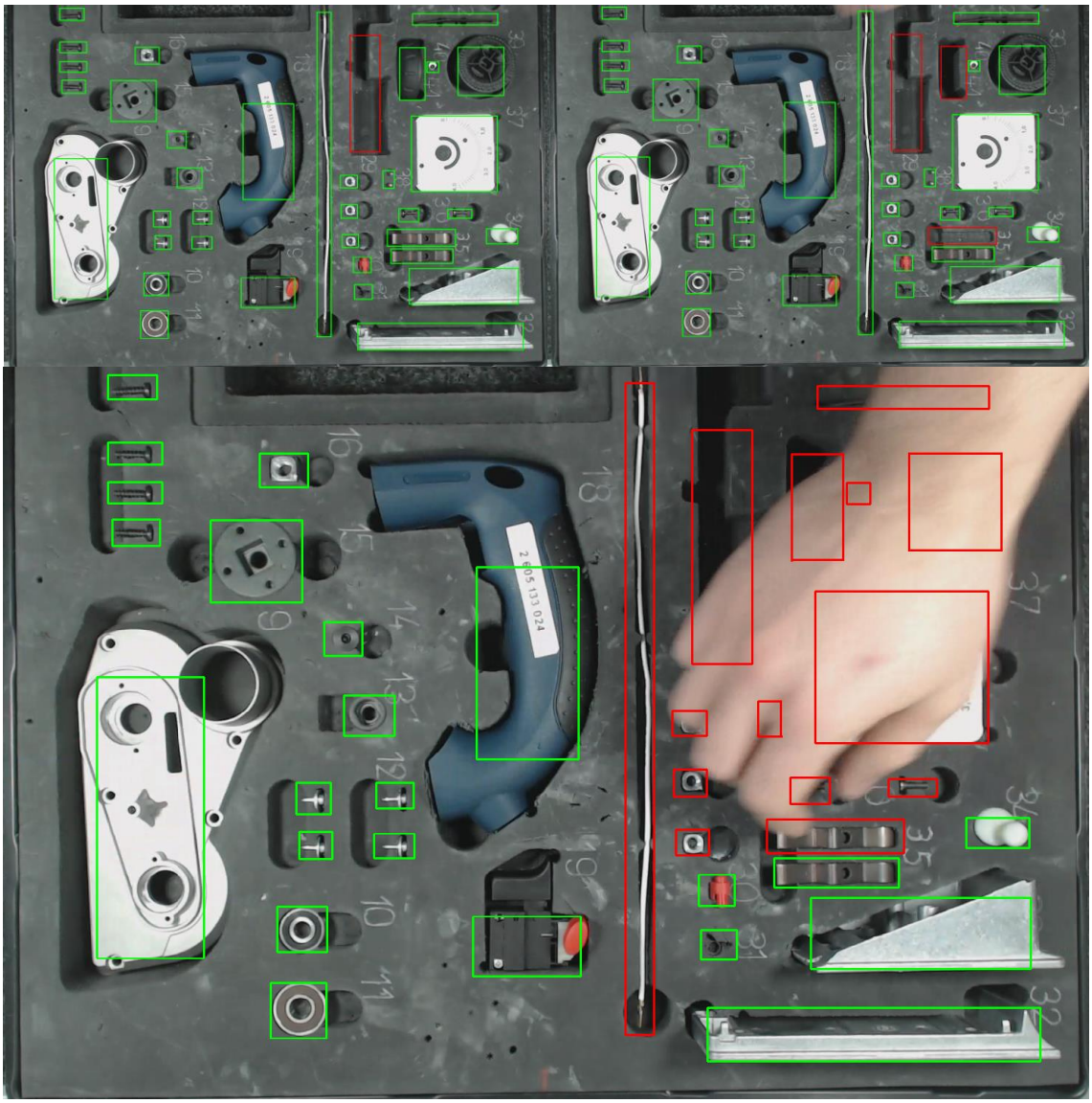


5.13. ábra A bal felső képen a minta, a jobb felsőn a fényviszony megváltozása, bal alsón a pozíció megváltozása, jobb alsón pedig az alkatrész hiánya látható

Az átlag intenzitásváltozás ott a legkisebb, ahol csak a fény változik, ennél nagyobb, ahol elmozdult az alkatrész és a legnagyobb, ahol hiányzik. Rendre 3.2, 7.5 és 57 az átlag intenzitásváltozás számértékekben kifejezve. Látszik, hogy az elmozdulás és a hiány között számottevő az ugrás. Megjegyzendő, hogy a fenti képeken a jobb láthatóság kedvéért maximális intenzitással rajzoltattam ki az eredményeket, a számértékek azonban az eredeti képeken mérve értendőek. Sajnálatos módon, ennél a módszernél a háttérhez hasonló színű alkatrészek problémája újból előkerül. Míg a többi alkatrésznél nagyságrendbeli ugrás történik az alkatrész hiányánál, úgy a következő ábrán látható alkatrész elmozdulása és hiánya között csupán 2.6-ról 6.1-re változik az

átlagintenzitás. Így a három problémás alkatrésznél egyedi intenzitásértékeket definiáltam

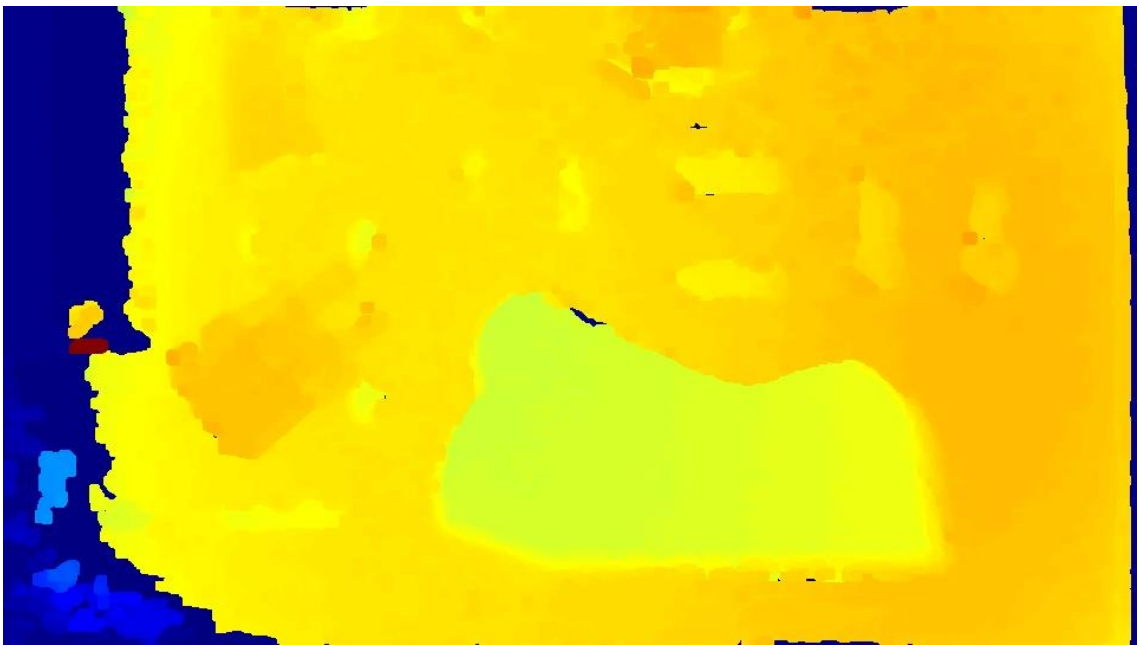
Ez a megoldás a korábban említettekhez képest mérve magasan a leggyorsabb, 11*ms* alatt végigfut mind a 29 alkatrészen és rárajzolja az eredeti képre a hiányzó alkatrészek helyét. Így ezt a módszert valósidejű videón is tudtam tesztelni. Ezt úgy hajtottam végre, hogy a kamera élő képére futtattam az algoritmust és közben kézzel vettem ki az alkatrészeket. Ennek eredményéről mutatnám a következő néhány képet (5.14. ábra):



5.14. ábra A valósidejű teszt eredményei

5.3 Összehasonlítás

Összességében sokkal jobb eredményeket sikerült elérnem a betanított algoritmusokkal, mint sztereóval. Ennek oka, hogy viszonylag nagy felületen (egy méterszer fél méter) szeretném kis alkatrészek hiányát/meglétét detektálni és ehhez olyan közelről kéne készítenem a képeket a blokk-illesztéshez, hogy ez a két kamera mozgatása nélkül nem lehetséges. Ráadásul sokkal időigényesebb a diszparitás számolása, mint a betanított algoritmusok számolásai. Pontosabban: 10cm magasról 3cm-es bázisvonallal sikerült elfogadható diszparitásképet készítenem, ahol már kivehetőek (de nem túl jól - 5.15. ábra) a kisebb alkatrészek. Ezekkel az adatokkal viszont legalább 30 különböző helyről kéne képeket készíteni az egész tálcáról. Továbbá a futásideje sem bizonyult elég gyorsnak.



5.15. ábra Sztereó konstrukció, ahol már a kisebb alkatrész mélyedések is látszanak

A betanított algoritmusoknál mindkét ismertetett módszernek megvannak az előnyei és hátrányai. A template matching hátrányai az amorf (változó alakú) és forgatható alkatrészeknél jönnek elő. Hiszen egy amorf vezeték vagy egy forgatható kör alakú alkatrész, mint például egy csapágy, végtelen pozíciót vehet fel, ezért ennek biztos felismerése ezzel az algoritmussal szinte lehetetlen. Ezeknél az eseteknél lehetne kisebb thresholdot használni vagy esetleg több template-tel dolgozni. A fenti példáknál 0.9-es thresholdot használtam és minden alkatrész meglétét egy template-tel vizsgáltam. Az alkatrészek pozíciójának ismeretében kizárható, hogy egy véletlenszerű alkatrész került

a szivacs azon részére, ahol nem kéne alkatrésznek lennie. Azzal az esettel, hogy egy alkatrész helyére hasonló került, de mégsem az odaillő, nem foglalkoztam. Ezektől az esetektől eltekintve 20 mintaképre tesztelve, mindent megtalált és a futásideje is teljesen elfogadható, ami kisebb, mint egy másodperc

A regionális intenzitásváltozás módszerével a fent említett amorf és forgatható alkatrészek is felismerhetőek. Hátrányai a háttérhez hasonló színű alkatrészeknél és az erőteljes fényviszony változásnál tapasztalhatóak. Utóbbi gyári körülmények között biztosítva lesz, azonban a tesztek során rengetegszer előjött ez a probléma, hogy a késő délutáni felvételek teljesen máshogy sikerültek, mint a délelőttiek, ahol a nap sokkal erőteljesebben sütött. Nem beszélve arról az esetről, amikor egy kolléga éppen elállta a fényt. A későbbi felvételeknél ezekkel az effektusokkal számoltam és két fényforrással világítottam meg külön a tálcát, de ezek kezdetben sok fejtörést okoztak. Mind a futásidőt, mind a pontosságot figyelembe véve ez a megoldás bizonyult a legjobbnak. Nagyságrenddel gyorsabbnak bizonyult, mint a korábban tesztelt megoldások.

6 Továbbfejlesztési lehetőségek

Dolgozatom leadása után a fejlesztés folytatódik, és további problémákkal is foglalkozom. Több lehetséges eset is előkerült, amikkel nem foglalkoztam korábban, azonban gyári körülmények között elkerülhetetlen az esettel foglalkozni. Ilyen például, ha egy alkatrész helyére egy hasonló kerül.

A sztereó megoldáshoz lehetne konstruálni egy sínt, amin csúsztathatóak lennének a kamerák, így akármekkora bázisvonallal lehetne számolni és tesztelni, hogy melyik bizonyul a legjobbnak.

A betanított algoritmusok érzékenyek a fényviszony megváltozására, árnyékok megjelenésére. Erre például egy ledsort lehetne alkalmazni, amely árnyékolásmentesen tudná megvilágítani a tálca összes alkatrészét.

Az alkatrészeket tároló szivacs lehetne sokkal előnyösebb színű. Például fehér, ami sokkal jobban elüt az alkatrészek színétől. Ezzel a szoftveres szegmentálás is sokkal egyszerűbb lenne és mindkét megoldás pontosabbá válna.

A tálca pozíciójának ellenőrzésére lehetne készíteni egy alkalmazást, ami három megjelölt pont alapján kiszámolná a tálca pontos pozícióját. Így lehetne viszonylagos értékekkel számolni és elég lenne azt megadni, hogy a tálca széleitől hol helyezkednek el az alkatrészek.

A GUI-t felhasználó barátiabbá lehetne tenni.

Irodalomjegyzék

[1] OpenCV: *Samples,* https://github.com/opencv/opencv/tree/master/samples/cpp (April 19, 2017)

[2] Wikipedia: *Computer stereo vision,* https://en.wikipedia.org/wiki/Computer_stereo_vision (revision 23:37, 22 Nov 2016)

[3] MathWorks: *What is camera calibration,* https://mathworks.com/help/vision/ug/camera-calibration.html (2017)

[4] Olga Krutikova, Aleksandrs Sisojevs, Mihails Kovalovs. *„Creation of a Depth Map from Stereo Images of Faces for 3D Model Reconstruction”,* Procedia Computer Science Volume 104, 2017, pp. 452-459

[5] George Brindeiro: *How to calibrate stereo camera,* http://wiki.ros.org/camera_calibration/Tutorials/StereoCalibration (Aug 31, 2012, 14:09)

[6] Wikipedia *Epipolar Geometry,* https://en.wikipedia.org/wiki/Epipolar_geometry (Feb 04, 2017, 18:58)

[7] OpenCV: *CPU vs GPU,* http://opencv.org/platforms/cuda.html (Last updated on May, 2011)

[8] OpenCV: *Camera Calibration and 3D Reconstruction,* http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html (Last updated on Jun 01, 2012)

[9] Leow Wee Kheng: *Camera Models and Imaging,* https://www.comp.nus.edu.sg/~cs4243/lecture/camera.pdf (Oct 19, 2012, 10:41)

[10] Kató Zoltán: *Ipari Képfeldolgozás,* http://www.inf.u-szeged.hu/~kato/teaching/IpariKepfeldolgozas/03-StereoCamera.pdf (Last modified: Oct 28, 2014, 21:01)

[11] Varga Péter: *Qt strandkönyv,* http://szabadszoftver.kormany.hu/letoltesek/konyveink/qt_strandkonyv.v2013-12.pdf (Dec 17, 2013)

[12] NASA: *Anagliph picture,* https://www.jpl.nasa.gov/spaceimages/details.php?id=PIA01427 (1998)

[13] Csetverikov Dmitrij: *Digitális képelemzés alapvető algoritmusai,* http://people.inf.elte.hu/redragon/Digit%E1lis%20k%E9pelemz%E9s/lec10_bin2morf_prn_4.pdf

[14] Myron Z. Brown, Member, Darius Burschka, Member, and Gregory D. Hager, Senior Member: „*Advances in Computational Stereo"*, IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 25, Issue: 8, Aug. 2003)

[15] Dibyendu Mukherjee, Guanghui Wang, Q.M. Jonathan Wu: *„Stereo correspondence based on curvelet decomposition, support weights, and disparity calibration"*, Optical Engineering on Machine Vision, Pattern Recognition (Volume 49, Issue 4, April 13, 2010)