



PROJEKTMUNKA 2

GKLB_INTM005

Objektumok távolságát meghatározó algoritmus fejlesztése és tesztelése

**HORNYÁK CSABA (OV216Y) &
SOÓS PÉTER (CKBD0A)**

GYŐR, 2020/21 1. FÉLÉV

Tartalomjegyzék

1	Bevezetés	2
1.1	A projekt célja és neve	2
2	Lehetséges módszerek.....	3
3	Tervezés	4
3.1	Mockup / Scratch Terv	5
3.2	A rendszer folyamatábrája	6
3.3	A rendszer felépítése, szükséges összetevők.....	7
4	Megoldás lépései [1][2][3]	8
4.1	Menü létrehozása.....	8
4.2	A szükséges összetevők importálása.....	8
4.3	Mintakép alapján fókusztávolság meghatározás a további számításokhoz.....	8
4.3.1	Kontúr keresés lépései	9
4.3.2	Fókusz távolság meghatározása a mintaképen	10
4.4	Képek, képkockák beolvasása.....	10
4.5	Kontúr keresés a mintaképhez hasonlóan	10
4.6	A távolság kiszámítása a mintakép alapján.....	10
4.7	A megtalált kontúr kirajzolása	11
4.8	A kiszámított távolság képre illesztése	11
4.9	Szükség esetén a kép átméretezése és megjelenítése	11
5	Tesztelés lépései.....	12
5.1	Teljesítmény teszt és a lehetőségekhez mérten ennek csökkentése.	12
5.2	Mérések elvégzése lézeres távolságmérő segítségével.	13
5.3	Kamera vagy objektum hatásainak vizsgálata	13
5.3.1	Dőlés	13
5.3.2	Fényerő változás	14
5.3.3	Elforgatás	14
5.4	Pontosság meghatározása a megfigyelések alapján	15
6	Grafikai Felület	16
7	Felhasznált irodalom	17

1 Bevezetés

A projekt hivatalos nyelve magyar. Ez magába foglalja, hogy a dokumentáció is magyar nyelven íródik. A projekt a későbbiekben, távlati tervekben természetesen más nyelveken is megérkezhet és elérhetővé válik.

1.1 A projekt célja és neve

Projekt neve: **Objektumok közötti távolságmérő**

Projekt célja:

Felhasználhatósága napjainkba akár az emberek távolságának meghatározására (Covid-19) vagy pozíciók meghatározására is lehetséges gyárakban. Természetesen egy ilyen komplex rendszer megtervezése, programozása és tesztelése egy hosszú folyamatot venne igénybe. Kicsit egyszerűsítve a programot, a jelenlegi feladatban a cél, hogy a gépi látás segítségével bemutassunk egy olyan a kamerától mért távolság meghatározására alkalmas algoritmust, ami egy meghatározott méretekkel és tulajdonságokkal rendelkező objektum esetében használható. Ezt fotók, videófelvétel és akár kamera kép alapján történő feldolgozás esetén is be lesz mutatva. Egy A4-es sárga lap kamerától mért távolságának meghatározásával kerül bemutatásra. Megnézünk 1-2 lehetőséget a távolság meghatározásnál fellelhető módszerekről (a teljesség igénye nélkül) és ezek közül egy irány mentén haladunk tovább. Bemutatásra kerül a tervezés lépései, a program részletes felépítése és a tesztelés menete.

A dolgozat részét képezi az algoritmus is. Amely a GitHub-ra feltöltésre kerül.

GitHub

Repository

elérése:

https://github.com/hornyacs/bead_Projektmunka2_Tavmeres

2 Lehetséges módszerek

Vegyük sorra milyen módszerek állnak rendelkezésre jelenleg a távolság meghatározására.

A kontakt mérési módszer, mint pl.: tolómérő, mérőszalag, vonalzó stb.

A nem kontakt módon történő mérések, mint pl.: lézeres távolságmérő, ultrahangos távolság meghatározás, GPS alapú távolság mérés és ide tartozik a gépi látás segítségével történő távolság meghatározások is.

Gépi látás segítségével távolság meghatározás: pl.: az egy kamera általi távolság meghatározás, a két kamera általi távolság meghatározás és a három vagy több kamera általi távolság meghatározás.

Természetesen ezen mérési módszerek és a hozzá tartozó algoritmusok ennek megfelelően eltérnek egymástól. Azt azonban leszögezhetjük, hogy a kamerák számától függően egyre pontosabb és jobb eredményeket kaphatunk. Gondoljunk csak arra, hogy több kamera más fényviszonyokhoz beállítva jobb felismerési eredményhez vezet. A 3D-s objektumok is több kamera használata esetén jobb felismerési arányhoz vezet. Természetesen a minél több kamerának akár hátrányai is lehetnek. Ilyen lehet a számítási gyorsaság csökkenés a nagy mennyiségű adat párhuzamos feldolgozása miatt. Ezért ennek hatékony működése nagyobb erőforrás igényű.

3 Tervezés

1. A rendelkezésre álló ill. szükséges eszközök felmérése. A programozáshoz egy ASUS ZenBook UX410U laptop ált rendelkezésre, amely Windows 10 operációs rendszert, webkamera legjobb felbontása 1280x720-as tartalmaz. Méréshez mérőszalagot és egy Parkside HG06724-es lézeres távolságmérőt alkalmaztam.
2. A bemutatáshoz egy vagy több A4-es sárga parírlap lett választva a felismerendő objektumnak. Ezt egy egyszerűbb 2D-s módszer megvalósítását tette lehetővé.
3. Ezt figyelembe véve egy 1 kamerás rendszer tervezése lett a cél.
4. A programozáshoz és más tervezéshez szükséges szoftverek beszerzése.
 - a. Szóba jött a C++ és a Python nyelv. Mivel több információ és anyag lelhető fel az OpenCV használatára Python nyelven, ezért erre dönt a mérleg. Források:[1][3].
 - b. A programozáshoz a Visual Studio 2017 szoftver került használatra. Ez segít a program gyors fordításában és futtatásában. Forrás: [4]
 - c. Balsamiq Mockup
5. A program felépítésének tervezési lépései:
 - a. Egy menü összeállítása, ami a kép, videó és webkamera feldolgozást különíti el.
 - b. Mindhárom esetben egy mintakép alapján meghatározott fókusztávolságot alapul véve szeretnék tényleges távolságot számítani.
 - c. Egyenként végig nézni a képeket, ill. videó és webkamera esetében a képkockákat.
 - d. Az A4-es lap tulajdonságait alapul véve szűrési funkciók beiktatása a minél pontosabb találat érdekében. Szín, sarokpontok száma, élek hosszának aránya.
 - e. A programozás és tesztelések során jöttek olyan felismerések, hogy mi van, ha több lapot vizsgálok egyszerre. Ezzel is át lett tervezve a program, hogy megfeleljen a céloknak.
 - f. Az eredmény egyértelmű kijelzése érdekében a megtalált objektumokat keretezzük, megszámozzuk és a számozásnak megfelelően a távolságokat folyamatosan feliratozzuk a képen.
6. Tesztelés lépései
 - a. Teljesítmény teszt és a lehetőségekhez mérten ennek csökkentése.
 - b. Mérések elvégzése lézeres távolságmérő segítségével.
 - c. Kamera vagy objektum hatásainak vizsgálata:
 - i. Dőlés
 - ii. Fényerő változás
 - iii. Elforgatás
 - d. Pontosság meghatározása a megfigyelések alapján.

3.1 Mockup / Scratch Terv

Ha szó szerint szeretnénk lefordítani a „mockup”-ot, azt kapnánk, hogy makett, vagy mintadarab. A grafikai tervezés világában viszont inkább egy kreatív sablonnak nevezzük, amely látványtervek elkészítésére alkalmas.

Amikor ezeket a terveket, vázlatokat, maketteket bemutatjuk a megrendelőnek, akkor értelemszerűen nem valósítjuk meg az összes prezentálandó, éles, kész ötletet, viszont a Photoshop, Illustrator, Mockup programoknak köszönhetően meglehetősen gyorsan és egyszerűen, teljesen élethű látványterveket tudunk készíteni.

A kezdeti látványtervek ugyanakkor nem csak az ügyfélnek jó, hanem belső embereknek is, főleg a site buildereknek, hogy milyen irányba kellene elmenni amikor építik fel lépésről lépésre az oldalt.

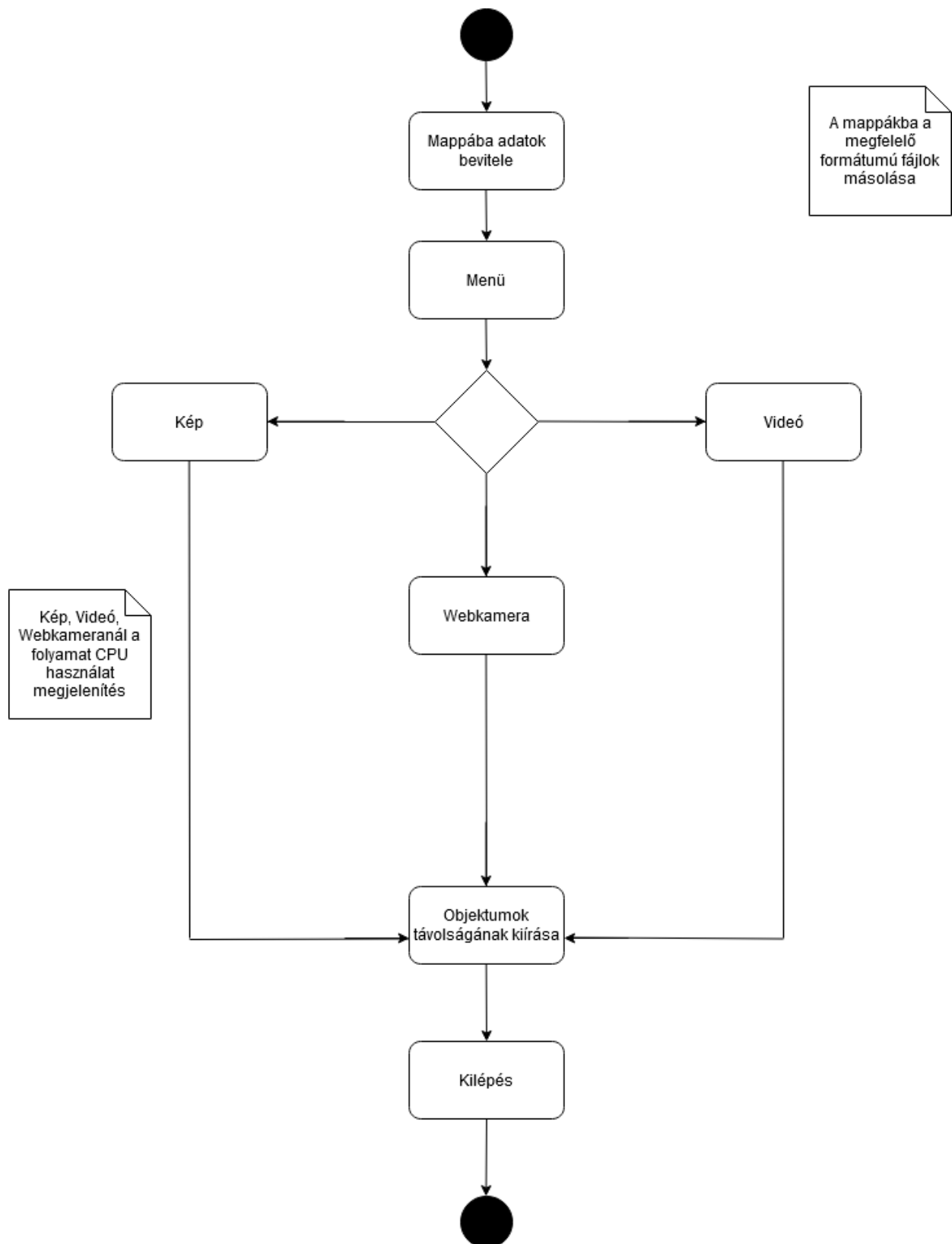
Dokumentációnk következő oldalain bemutatjuk lépésről – lépésre a software-ünk minden egyes látványelemét. Látni fogjuk, hogy nagyjából milyen is lesz a kész software-ünk grafikus, felhasználóbarát alakja.

Az aktuális mockup tervet a Balsamiq Mockup nevezetű szoftverrel készítettük el.












Felhasználóbarát kezelőfelület, a szoftver előzetes látványterve.



3.2 A rendszer folyamatábrája



3.3 A rendszer felépítése, szükséges összetevők

	.git	Fájlmappa
	__pycache__	Fájlmappa
	icon	Fájlmappa
	images	Fájlmappa
	ingredients	Fájlmappa
	samples	Fájlmappa
	video	Fájlmappa
	.gitattributes	GITATTRIBUTES fájl
	.gitignore	GITIGNORE fájl
	Távolságmérés.exe	Alkalmazás
	Távolságmérés.py	Python File

1. ábra A programhoz szükséges összetevők

- Az alkalmazást a TavolsagmeresAppStart.exe fájlal lehet elindítani.
- A Távolságmérés.py a python kódot tartalmazza.
- Az icon mappa a felhasznált ikonokat tartalmazza.
- Az images mappa tartalmazza a feldolgozandó képeket.
- Az ingredients a használatához szükséges további .py fájlokat.
- A samples a mintaképeket, amit kép, videó és webkamera esetében megkülönböztetünk
- A video mappában a feldolgozandó minta videó és egy teszt videó (Távolságmérés_app_bemutató.mp4) is található, ami bemutatja a működést.

A használatbavételhez csak annyit kell tennünk, hogy a Github-ról a mellékelt linkről letöltjük az appot, kicsomagoljuk a tömörített állományt és elindítjuk a Távolságmérés.exe fájlal.

GitHub Repository Download Link:

https://github.com/hornyacs/bead_Projektmunka2_Tavmeres/archive/master.zip

4 Megoldás lépései [1][2][3]

A feladat megoldásához a Python programozási nyelv lett választva. A továbbiakban a program felépítésének lépésein megyünk végig.

4.1 Menü létrehozása

A fotók, videó és élő webkamerakép feldolgozás választási lehetőség miatt első körben, a kezdeti fázisban egy menü lett elkészítve, ami a következőképpen nézett ki.



2. ábra Menü felépítése

4.2 A szükséges összetevők importálása

1. Összetevők importálása

```
import os, cv2, time, threading, psutil
import platform
import ingredients.definitions as df
import numpy as np
import tkinter as tk
from tkinter import *
from PIL import Image, ImageTk
from tkinter import messagebox
import tkinter.ttk as ttk
```

Az os-re csak a képfeldolgozásnál van szükség a könyvtár kezeléshez. A numpy csak a keret típuskonverzióhoz és tömb képzéshez van használva. Az OpenCV végzi a feladat oroszánrészét. Ebből felhasznált funkciókat a továbbiakban kerül kifejtésre. A tkinter az UI kivitelezése miatt szükséges. A Pil a képek kezelése miatt. A definitions a 3 menüpont közös feladatainak elvégzésére hoztam létre.

4.3 Mintakép alapján fókusztávolság meghatározás a további számításokhoz

Első lépésben a mintaként készített képek adatainak letárolását végeztük el. A mintaképet mérőszalag ill. lézeres távolságmérő segítségével meghatározott távolságban készítettük, ami a képfeldolgozás esetében 50cm a videónál 100cm a kamerakép esetén pedig 120cm volt, ami letárolásra került a „Minta_tavolsag” változóban. A minta szélesség értékét a „Minta_szélesség” változóban tárolódik, amit egy A4-es papírlap szélesség értékére lett állítva, azaz 29,7cm-re. A mintakép betöltésével és a kontúr keresés függvény hívással folytatjuk a kódot.

2. Mintakép betöltése

```
mintakep = cv2.imread("samples/_sampleimage.jpg")
c = max(df.konturkeres(mintakep), key = cv2.contourArea)
marker = cv2.minAreaRect(c)
```

4.3.1 Kontúr keresés lépései

Mivel ezeket a lépéseket többször kell alkalmazni a program futása során, ezért külön fájlba lett szervezve (definition.py). Erre szükség van mindhárom menüpont esetében és a mintakép, a mérendő képek, videó és webkamera képkockák esetében egyaránt.

4.3.1.1 Gauss szűrés

Első lépésben egy szűrő lett alkalmazva a GaussianBlur() függvénnyel, ami egy kis elmosódást eredményez a képen a jobb élkeresés érdekében. Kísérletezés is volt a bilateralFilter() függvénnyel is, ami annyival jobb lett volna, hogy az éleket jobban kiemeli, de ennek használata már látható teljesítményromlást eredményezett, ezért nem került bevezetésre.

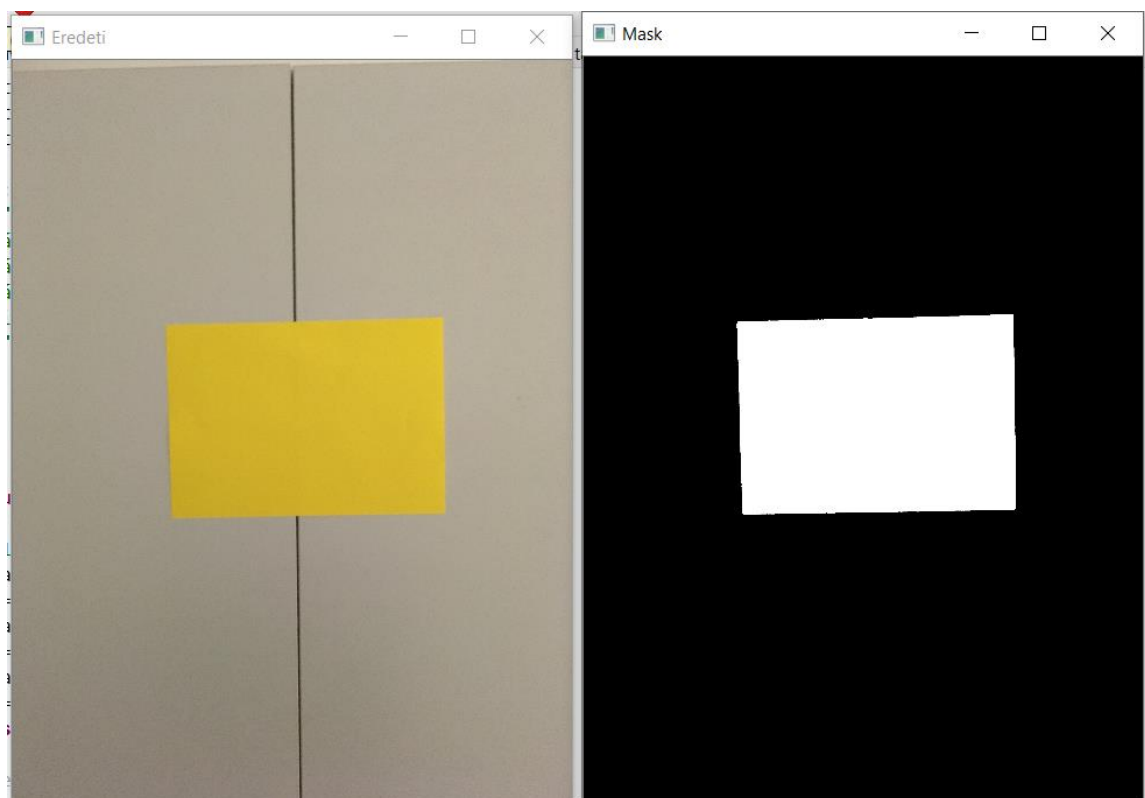
4.3.1.2 HSV színtér konverzió, szín szűrés és maszkolás

A következő lépésként a színek átalakítását a cv2.cvtColor() függvénnyel került megvalósításra, melynek a cv2.COLOR_BGR2HSV argumentuma szolgál a BGR kép HSV színtérre való konverzióhoz. A színszűrést és a maszkolást a következő kódrészlet tartalmazza.

3. Színszűrés és maszkolás

```
color_min = np.array([10, 130, 128])  
color_max = np.array([40, 255, 255])  
mask = cv2.inRange(hsv, color_min, color_max)
```

A minimum és maximum értékek megállapításához, különböző beállítások végig zongorázása után alakult ki. A legjobb felismerés érdekében a maszkolás csak a keresett színt tartalmazza különböző beállítások mellett.



3. ábra Eredeti és a maszkolt kép összehasonlítása

4.3.1.3 A maszkolt képen kontúrok keresése

Erre az OpenCV a `cv2.findContours()` függvény nyújt nekünk segítséget. Első argumentuma a bemeneti kép, ami esetünkben a maszkolt kép, a második a `cv2.RETR_EXTERNAL` csak a legkülső kontúrokat veszi figyelembe és a harmadik a `cv2.CHAIN_APPROX_SIMPLE` a `cv.CHAIN_APPROX_NONE`-al szemben nem az összes él menti pontot, hanem csak a sarokpontokat veszi figyelembe. Ezzel memóriát spórol, ami gyorsabb működést eredményez.

4.3.1.4 A4-es lapnak megfelelő méretek szűrése

Végig nézzük a megtalált kontúrokat és ezek közül kiválogatjuk egy A4-es lapnak megfelelő tulajdonságokkal rendelkezőket. Erre a `cv2.approxPolyDP()` függvény lett használva, ami az előre megadott pontosságnak megfelelően közelítéssel adja vissza az alakzatunk csúcsszámát. Természetesen a 4 csúccsal rendelkező kontúrokat vesszük csak figyelembe. A `cv2.minAreaRect()` függvény kiszámítja és visszaadja a minimális területet határoló téglalapot (esetleg elforgatva) egy megadott pontkészlethez [3]. Ezt követően a kontúr hosszúság és szélesség arány került kiszámításra, hogy kiszűrjük a nem megfelelő arányú téglalapokat. A megfelelő aránnyal rendelkező kontúrokat megtartjuk (A4-es lapnál ez: $2,97/2,1=1,414$). Természetesen ennek lett adva egy min - max tartomány (1,35-1,48).

4.3.1.5 A megtalált kontúr visszaadása

Az így összegyűjtött kontúrokat (ha van ilyen) visszaadjuk a meghívott függvénynek. Abban az esetben, ha nincs ilyen tulajdonságú kontúr, akkor 0 kerül visszaadásra.

4.3.2 Fókusz távolság meghatározása a mintaképen

Itt ismét alkalmazzuk a korábban 4.3.1.4-ben említett `cv2.minAreaRect()` függvényt. Ennek felépítése a következő: [a középpont x,y koordináták] , [szélesség, magasság] , elfordulási szög

A szélesség adat figyelembevételével megállapítjuk a fókusz távolságot.

4. Fókusz távolság számítása

$$\text{focalLength} = (\text{marker}[1][0] * \text{Minta_tavolsag}) / \text{Minta_szelesseg}$$

4.4 Képek, képkockák beolvasása

Képek esetében az „images” mappából beolvassuk egy for ciklus segítségével és az `os.listdir()` használatával a képeket. Videó és webkamera esetében képkockákra bontás egy while ciklussal valósul meg. A képkocka betöltésére a `ret, frame = cap.read()` szolgál. Természetesen, ha nincs több kép, akkor kilépünk a ciklusból `break` segítségével.

4.5 Kontúr keresés a mintaképhez hasonlóan

Itt ismét elvégezzük a vizsgálandó képen vagy képkockán a kontúr keresés lépéseit, amit a 4.3.1 fejezetben került kifejtésre. A visszkapott kontúrokon dolgozunk tovább egy ciklusban. Ha a mintaképtől eltérő pozíciójú képet kapunk, akkor megcseréljük a szélesség és hosszúság adatokat a későbbi helyes számítás miatt.

4.6 A távolság kiszámítása a mintakép alapján

A „cm” változóban a mintakép alapján letárolt fókusz távolság figyelembevételével kiszámítjuk a vizsgált képen fellelhető leszűrt kontúrok távolságát cm-ben. A jobb hatékonyság érdekében csak a 4 méternél közelebbi eredményeket vesszük figyelembe.

5. Távolság számítás meghívása

```
cm = df.tavolsag(Minta_szelesseg, focalLength, marker[1][0])
```

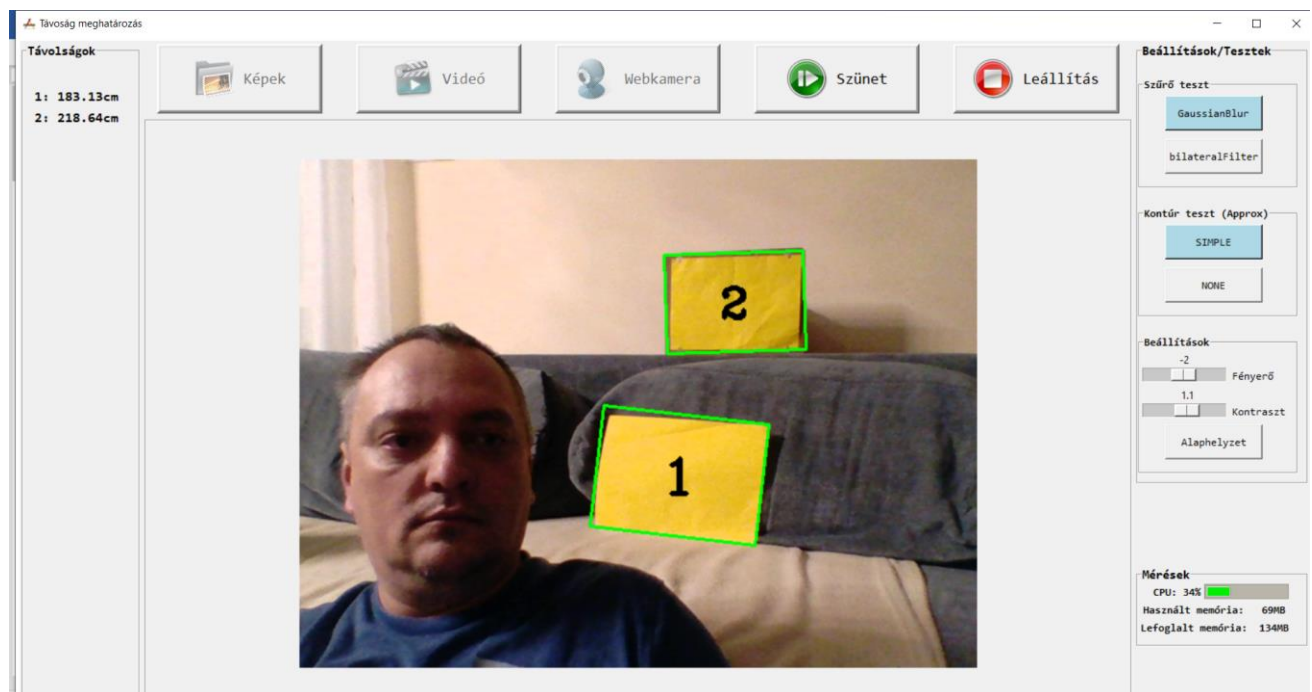
A távolság függvényünk a $(Minta_szelesseg * focalLength) / marker[1][0]$ eredményt szolgáltatja vissza nekünk. Mivel ennek a képletnek a használata is többször jön elő, ezért ezt is a definition.py fájlba helyezzük el.

4.7 A megtalált kontúr kirajzolása

A sarokpontok ismeretében a cv2.boxPoints() és cv2.drawContours() segítségével zöld színű kontúrt rajzolunk a jó láthatóságnak megfelelő vastagságban.

4.8 A kiszámított távolság képre illesztése

Ezt több lépésben valósul meg. Elsőként megszámozzuk a talált objektumokat, annak közepére írt sorszámmal, ahol a távolságot, betűméretet és a betűvastagságot is figyelembe kellett venni a megfelelő megjelenítés érdekében. Ezt a számozást használva a Távolságok mezőben sorban lefelé megjelenítjük a hozzá tartozó távolság értékeket (4.ábra). Amennyiben nem található megfelelő objektum, akkor a Not detekted felirat jelenik meg a távolság helyén. A szöveg képre illesztéséhez az OpenCV cv2.putText() függvényét alkalmazzuk.



4. ábra A kiszámított távolságok képre illesztése

4.9 Szükség esetén a kép átméretezése és megjelenítése

Annak érdekében, hogy a képernyőn elférjen a videónk vagy képeink, szükség lehet átméretezni és úgy megjeleníteni az eredményünket. Erre az OpenCV a cv2.resize() függvényt használjuk, ahol a bemeneti képet és a kívánt felbontást kell csak megadnunk. A képek megjelenítésének folyamatossága miatt egy 2ms késleltetéssel történik a következő kép beolvasása a mappánkból. Videó és webkamera esetében a cap.release() segítségével felszabadítjuk a videót. Majd a programunk végén bezárhatjuk a windows ablakot.

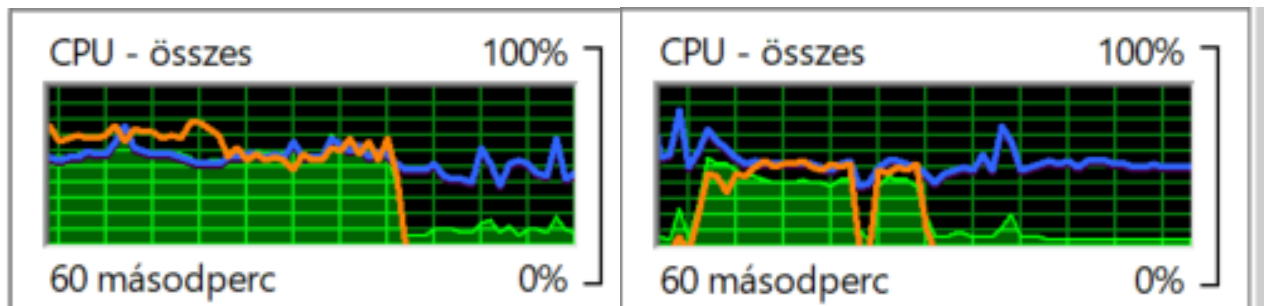
5 Tesztelés lépései

5.1 Teljesítmény teszt és a lehetőségekhez mérten ennek csökkentése.

Vegyünk végig néhány teljesítményt befolyásoló beállítást és annak hatásait. Egyik a teljesítményt nagy mértékben befolyásoló tényező a szűrő használatának kiválasztása. A `cv2.bilateralFilter(bemeneti kép, 15, 75, 150)` beállítás az éleket kiemeli a háttérrel elmossa, ami pontosabb felismerést eredményez. Viszont a teljesítmény látható módon romlik. A videó képkockái lassabban jelennek meg (akadozik). A képek később töltődnek be. A mellékelt videó feldolgozása így 250795.89190000002 ezredmásodpercet vett igénybe, ez Gauss szűrő esetében a `cv2.GaussianBlur(bemeneti kép, (5, 5), 0)` beállítások mellett 23665.2625 ezredmásodpercig tartott, ami 10-ed részére csökkentette a feldolgozási időt.



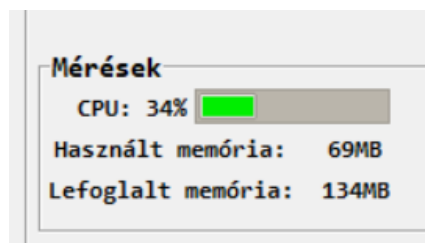
5. ábra Eredeti és bilateralFilter szűrő összehasonlítása



6. ábra BilateralFilter és a Gauss szűrő összehasonlítása CPU használatára

A CPU használat is hosszabb ideig tartó és magasabb terhelést eredményezett. A bilateralfilter esetében ez 80%-os leterheltséghez vezetett közel 10x hosszabb időtartamban a Gauss szűrőhöz viszonyítva ahol a CPU terhelés csak 50-60% között mozgott (6.ábra).

Az felhasználói felületen folyamatosan megjelenítésre kerül az alkalmazásunk CPU és memória használatára (7.ábra).



7. ábra A folyamat CPU és memória használatának monitorozása

A következő ilyen a kontúr keresés beállításában rejlik. Nézzük meg a `cv2.findContours()` `cv2.CHAIN_APPROX_NONE` és `cv2.CHAIN_APPROX_SIMPLE` argumentum beállításainka hatását a feldolgozási idő függvényében. Több mérést alapul véve és az eredményeket átlagolva a következő eredmény jött ki:

cv2.CHAIN_APPROX_SIMPLE: átlagosan 23400 ezredmásodperc.

cv2.CHAIN_APPROX_NONE: átlagosan 23700 ezredmásodperc.

Ennek hatása tehát közel 300 ezredmásodperc megtakarításhoz vezetett, ami elenyésző különbségnek bizonyult.

5.2 Mérések elvégzése lézeres távolságmérő segítségével.

A mérési eredményeket, melyet a 7.ábra tartalmaz a véglegesen kialakult kód alapján az ideális feltételek mellett értékeltük ki a webkamera kép alkalmazása esetén.

Kijelzett távolság [mm]	Lézeres távolságmérővel mért távolság [mm]	Eltérés Abszolút értéke [mm]
501	499	2
500	498	2
502	500	2
499	499	0
500	499	1
1001	999	2
1001	998	3
999	997	2
1000	997	3
1001	998	3
1499	1501	2
1502	1501	1
1503	1507	4
1504	1502	2
1510	1501	9
1997	1986	11
1994	1985	9
1995	1985	10
1997	1986	11
1994	1986	8
2478	2488	10
2497	2493	4
2513	2488	25
2497	2491	6
2496	2490	6
2992	2984	8
2991	2983	8
2992	2983	9
3001	2988	13
3018	2992	26

7. ábra Mért eredmények táblázata

A tesztelés alatt próbálgattuk, hogy mi az a kritikus távolság, aminél az objektum felismerés már bizonytalanává válik. 7 méter fölött kezdett jelentkezni először a detektálási hiba és 8 méter körül már nagyon gyakorivá vált.

5.3 Kamera vagy objektum hatásainak vizsgálata

5.3.1 Dőlés

Mind az objektum, mind a kamera dőlése hatással van a felismerés pontosságára. A kódból adódóan vagy nem ismeri fel az objektumot az oldal arányok megsértése miatt, vagy egyre pontatlanabb mérést eredményez ill. hibás kontúr találathoz vezet. Ezt szemlélteti a 8. ábra.



8. ábra Dőlésből adódó hibás kontúr találat

5.3.2 Fényerő változás

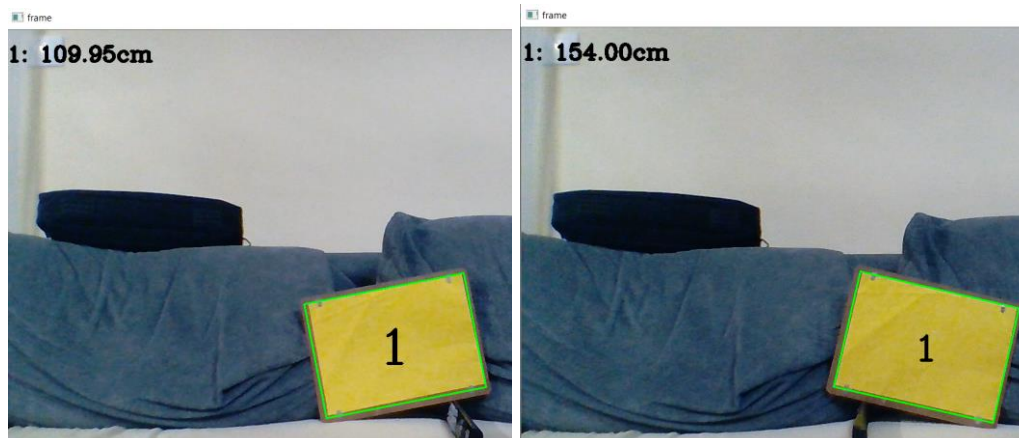
A fényerő változás nagy hatással van a felismerésre. A szín szűrés az, amit a leginkább befolyásol. A fényerő változást a szín szűrés határainak növelésével kell ellensúlyozni, de így több olyan tárgy is felismerhető volt, ami nem a kívánt objektum. A fényviszonyok változása következtében történő objektum felismerés hatását a 9. ábra szemlélteti. Az UI-n real-time módosíthatjuk és nyomon tudjuk követni hatását.



9. ábra Fényviszonyok hatása a felismerésre

5.3.3 Elforgatás

Első körben a kamera vagy az objektum elforgatása több problémát is felvetett. Az egyik ilyen a vízszintes és függőleges képeket nem megfelelően keretezte, ezért meg kellett cserélni a szélesség és magasság adatokat. A másik a cv2.minAreaRect() függvény elfordulási szög okozta, ami negatív értéknél okozott problémát (10. ábra). Ezt a szélesség és magasság nagyságának vizsgálata után át lett számítva 90+eredeti elfordulási szög formában. Így nem adott negatív eredményt és helyes megjelenítéshez vezetett.



10. ábra Bemutatja az azonos távolságban elhelyezett objektum elfordulásból adódó helytelen távolság számítását

5.4 Pontosság meghatározása a megfigyelések alapján

A fellelhető objektum merőlegesen a kamerára helyezkedik el. A fényviszonyok megfelelőek és nincs egyéb zavaró tényező. Abban az esetben a több mérésen alapulva a pontosságról az mondható el, hogy a kamerától egyre inkább távolodva a felismerés és a pontosság romlik. Természetesen ez függ a felbontástól is, ami minél nagyobb annál kevésbé szórnak az adatok. A következtetéseim levonásában a statisztikai adatok voltak segítségünkre, amit a 11. ábra tartalmaz. Van azonban egy sokkal nagyobb probléma is ezzel a módszerrel.

Abban az esetben, ha egy A5-ös vagy mindegy, de az A4-es lap arányait megtartva akarok felismerni objektumot, akkor hibás eredményre jut az algoritmus. Ugyan a felismerés megvan, de nem a helyes távolságot adja eredményül.

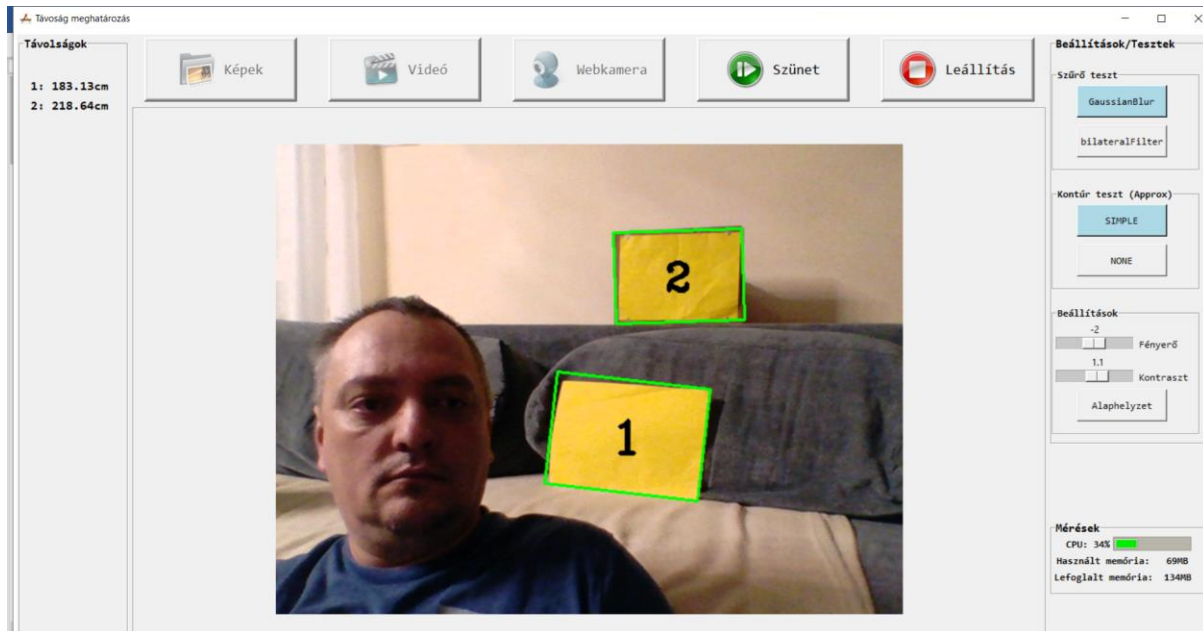
A pontosság a mért eredmények figyelembevételével úgy alakul, hogy a távolság növelésével a pontosság romlik. Ettől függetlenül meglepően pontosnak bizonyul. A mérőműszer 2mm-es pontatlanságát figyelembe véve 1,5-2 méter fölött van csak érdemi eltérés.

Távolság	Átlag eltérés	Eltérések szórása
50	1,4	0,894427191
100	2,6	0,547722558
150	3,6	3,209361307
200	9,8	1,303840481
250	10,2	8,555699854
300	12,8	7,661592524

11. ábra Statisztikai mutatók a távolságok függvényében

6 Grafikai Felület

Felhasználóbarát kezelőfelületű, a szoftver végleges látványterve, grafikusan.



7 Felhasznált irodalom

- [1] How to Think Like a Computer Scientist: Learning with Python 3 Documentation Release 3rd Edition
Peter Wentworth, Jeffrey Elkner, Allen B. Downey and Chris Meyers
<https://buildmedia.readthedocs.org/media/pdf/howtothink/latest/howtothink.pdf>
- [2] A gépi látás és képfeldolgozás párhuzamos modelljei és algoritmusai Rövid András, Sergyán Szabolcs,
Vámosy Szabolcs
- [3] <https://opencv.org/sdf>
- [4] <https://docs.microsoft.com/en-us/visualstudio/python/tutorial-working-with-python-in-visual-studio-step-01-create-project?view=vs-2019>