Gépi látás

GKNB_INTM038

Kapcsolási rajz konvertálás KiCad szerkesztőbe

Bicsák Bence

NWHTNA

Győr, 7. félév

Tartalomjegyzék

1 Bevezetés

A dokumentum lényege, hogy ismertesse a szoftver működési elvét, megoldások lépéseinek bemutatása, a szoftver milyen pozitívumokkal, negatívumokkal rendelkezik.

A program a manapság sokszor előforduló problémára ad megoldást: egy kép tartalmának szerkeszthető változatát.

Egy gyártó vagy javító cégnél sűrűn előfordul, hogy a fejlesztő mérnököknek az adott eszközt kell fejleszteni és van, hogy a kapcsolási rajza csak papírón, vagy digitális formában van meg

Egy ilyen példával találkoztam én is szakmai gyakorlatom során. Egy terminál szervizelésénél alkalmazott panel kapcsolási rajzát kellett volna módosítani, de mivel csak papíron volt meg a kapcsolási rajza, ezért a rajzolással szoftveresen egy órát vett igénybe. Milyen egyszerű lett volna ha van egy szoftver amely beolvassa a kapcsolási rajzot, felismeri rajta a kapcsolási elemeket és ezeknek a pozíciója alapján átkonvertálja KiCad-be szerkeszthető formába.

Másik probléma az lehet hogy egy kézzel rajzolt, 2D rajzot olyan valaki is beolvastathatja a szoftverrel aki nem feltétlen ért vagy még csak ismerkedik a kapcsolási rajzok készítésével.

Még egy ilyen probléma lehet, hogy szabványeltérések is lehetnek. Előfordulhat, hogy a fejlesztőmérnök rosszul értelmezi a kapcsolási rajz elemét, így hibázhat és ezt a lehetőséget kiküszöbölhető a szoftverrel.

2 Elméleti háttér

A program megoldásához szükséges az élek és egyenesek detektálása, a objektumok mintakeresés módszere elmélet valamint a KiCad sch fájl felépítésének az ismerete.

2.1 Éldetektálás

Él ott található egy képen, ahol az intenzitás értéke nagymértékben, hirtelen változik meg. Általában az objektumok határvonalának meghatározására alkalmazzuk, viszont nem csak a objektumoknak van éle, hanem árnyéknak, törésvonalnak vagy objektumon belüli objektumnak.[4]

A megoldáshoz a bemenő jelet deriváltjának az abszolút értékét. A képet diszkrétként kezelve a deriváltat differencia számítással közelítünk. Ahol folytonosnak ott konvolúciós megközelítést alkalmazunk.
Azonban a deriválás hátránya, hogy kiemeli a zajokat, amire ha alkalmazzuk a Gaus szűrőt akkor a zaj eltűnik, de lényeges részek a képről úgyszintén.
Viszont a második derivált sokkal jobb eredményt ad amit Laplace szűrővel közelítünk.[4]

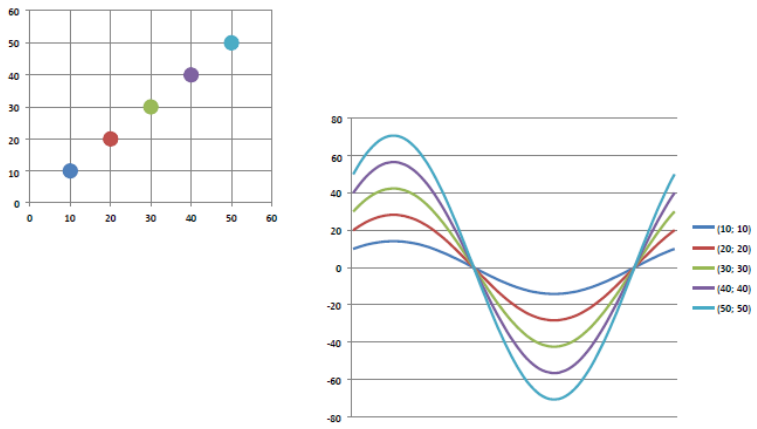
A legismertebb és itt is alkalmazott éldetektor a Canny éldetektor.
Canny működési elve:

- szürkeárnyalatossá alakítjuk a vizsgálandó képet.
- Gaus simítás
- Differencia számítás
- Nem-maximum vágás
- Kettős küszöbölés
- Hiszterézis küszöbölés[4]

2.2 Egyenes detektálás

Digitális képek esetén végesszámú pontok halmaza, ami egy adott tűréssel illeszkedik az egyenesre. Egyenes lehet egy objektum határvonal is[2].
Hough-transzformáció:

- A valós képtérről transzformáljuk a pontokat Hough térbe.
- Megadja egy ponton átmenő összes egyenes egyenletét.



1. ábra: Hough-transzformáció[2]

A Hough-transzformációt a következő esetekben használjuk:

- Éldetektálás
- Binarizáció
- Morfológia és zajszűrés

A Hough térben a világos pontokat vizsgáljuk. Diszkrét térképet általában a Hough-tér. Egyenesek súlyozása szavazásos módszerrel történik és fontos a jól megválasztott tűrés határérték.[2]

2.3 Mintakeresés

Mintakeresésnél 3 eljárást különbözetünk meg és ezek normalizált módjait:

- Négyzetes különbség
- Kereszt- korreláció
- Korreláció koefficiens

Ezeket a módszereket piramis módszerre javítható a műveleti sebesség, de továbbra is a hátrány, hogy nem skála- invariáns és nem forgatás- invariáns.[3]

Alakzatjellemzők: objektumok határvonalakkal jól jellemezhetők. Ilyen jellemző a határpont, lánckód.[3]

Határpont: Az a pont az objektumon, amely rendelkezik N db nem objektum részeként szereplő szomszédpontokkal.[3]

Lánckód:

- határpontok összefűzése a lánckód
- órajárásával ellentétes irányú számozás
- Elemeit az elmozdulások alkotják
- Van 4-es és 8-as lánckódok
- Különbségkód: A lánckód első deriváltja
- Normalizálás: Lánckód permutációjának a minimuma
- Alakleíró: Normalizált különbségkód, független a kezdőponttól.
- Kerület, Terület, Kompaktság, Cirkularitás és Rektangularitás

Előnye a lánckódnak, hogy eltolás-invariáns, gyorsanszámítható és könnyen előállítható alakzat. Hátránya viszont, hogy nem forgatás- és skála- invariáns, maximum képpontnyi pontossággal dolgozik és zaj érzékeny módszer.[3]

2.4 KiCad formátum

A KiCad egy nyílt forráskódú szoftver. A szoftver elektronikus vázlatos diagramok és NYÁK rajzok készítésére alkalmas. A mi megoldásunknál a program egy Eeschema vázlatszerkesztő és komponensszerkesztő kódját fogja legenerálni. A generált fájlt majd egy új projektben megnyitva fogjuk társítani a Eeschema szerkesztő felülethez.

3 A megvalósítás terve és kivitelezése

A programot Python nyelven valósítom meg Visual Studio Code fejlesztőkörnyezetben. A megvalósításhoz importálni kell az Opencv és a Numpy kiegészítőket.
A tervezett lépések:

- A vizsgálandó kép beolvasása.
- A képet szürkévé konvertálás.
- A keresett elemek megkeresése a képen.
- A talált elemeket kiírjuk a fájlba a középpontjukat megadva inchben.
- A következő lépésben az egyeneseket detektáljuk.
- A talált egyeneseket a fájlba kezdő és a végpontokat megadva adjuk meg inchben.

3.1 Kép beolvasás

A képek beolvasás a teszt mappából történnek az *„cv2.imread(”teszt1.png”)”* kód segítségével. A képek 200x 200 körüliek.

3.2 Kép szürkévé konvertálás

A szürkévé konvertálás azért szükséges, mert egyenesek detektálása és objektum felismerésnél segítségünkre lesz. A konvertáláshoz unit8 formátumba kell alakítani, majd a „cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)” kóddal konvertálunk. Az I a képünk.

3.3 A keresett elemek megkeresése a képen

Alapesetben az alábbi képeket ismeri fel a program:

5. ábra: Volt

4. ábra: Kapcsoló

3. ábra: Megszakító

2. ábra: Lámpa

6. ábra: Amper

7. ábra: Ellenállás

A elemeket beolvassuk a *„cv2.imread(”teszt1.png”)”* paranccsal majd szürkeárnyalatossá konvertáljuk őket a *„cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)”*.
Ezekután a *„cv2.matchTemplate(I, T.astype(np.uint8),cv2.TM_CCOEFF)”* függvénnyel keressük meg a képen az elemeket. A TM_CCOEFF összehasónlító módszert alkalmazom, mert ez adja a legjobb értéket.

matchTemplate():
Egyszerűen csúsztatja a sablon képet a bemeneti kép fölé (mint a 2D konvolúcióban), és összehasonlítja a bemeneti kép sablonját és javítását a sablon kép alatt. Az OpenCV-ben számos összehasonlítási módszer valósul meg. Szürkeárnyalatos képet ad vissza, ahol minden pixel azt jelzi, hogy az adott pixel szomszédsága mennyire egyezik a sablonnal. Miután megkapta az eredményt, a”cv2.minMaxLoc ()”függvény segítségével megtudhatja, hol van a maximális / minimális érték.[1]

A *„minMaxLoc()”* függvényt felhasználva megállapítjuk a keresés eredményét.

3.4 A talált elemeket kiírjuk a fájlba a középpontjukat megadva inchben.

Sajnos a *„minMaxLoc()”* függvény képes olyan elemet is megtalálni ami igazából nincs a képen vagy csak minimálisan hasonlít rá. Ezért alkalmazunk egy küszöbértéket. A mi esetünkbe a *„threshold”* változó tartalmazza. Az értéket összehasonlítjuk a *„minMaxLock()”* értékével és ha vele egyenlő vagy nagyobb akkor az azt jelenti, hogy a kép nagy valószínűséggel tényleg rajta van.

```
if(np.amax(M1) >= threshold):
    
    Rc="R1"
    Rn="R_PHOTO"
    x=int((maxloc[0] + Ellenallas.shape[1] // 2)*10.4166667)
    y=int((maxloc[1] + Ellenallas.shape[0] // 2)*10.4166667)

    f.write("$Comp\n")
    f.write("L Device:R_PHOTO R1\n")
    f.write("U 1 1 6003C928\n")
    f.write("P "+str(x)+" "+str(y)+"\n")
    f.write("F 0 "+str(i+Rc+i)+" V "+str(x+len(Rc))+" "+str(y+len(Rc))+" 50  0000 C CNN\n")
    f.write("F 1 "+i+Rn+i+" V "+str(x+len(Rn))+" "+str(y+len(Rn))+" 50  0000 C CNN\n")
    f.write("F 2 "+i+i+" V "+str(x)+" "+str(y+60)+" 50  0001 L CNN\n")
    f.write("F 3 "+i+h+i+" H "+str(x)+" "+str(y+60)+" 50  0001 C CNN\n")
    f.write("   1    "+str(x)+" "+str(y)+"\n")
    f.write("   1    -1    -1    0\n")
    f.write("$EndComp\n")
```

8. ábra: Elem kiíratás

Az x és y változóba meghatározzuk az elem középpontját inchben. Mivel a KiCad saját maga állítja be a tizedes vesszőt, ezért az inch váltószámot meg kellett szorozni ezerrel. Az így kapott számot egésszé kell alakítani, erre szolgál az int().

Egy elem kiíratását a *„$Comp”* sorral kezdünk és a *„$EndComp”* sorral zárjuk.

Az kiíratásnál a *„P”* kezdősorú kiíratásnál kezdődik. A *„P”* sorban a koordinátákat adjuk meg, hogy hol fog az elem elhelyezkedni. Mivel szám értéket nem lehet kiíratni, ezért str()- rel string típussá alakítom.

Az *„F 0”* sorban az Rc változó értékét iratom ki, aminek a koordinátája az elem koordinátája plusz a Rc érték hossza lesz.

Az *„F 1”* sorban ugyanaz a eljárás csak ott a Rn változó értékét helyezzük el és használjuk fel pozíciónálásnál.

Az *„F 2”* esetén a *” ”* értéket és a *„F 3”* esetén a *„~”* értéket helyezzük el az y koordináta + 60 helyen.

A következő két sor a koordinátát tartalmazza ismét valamint az elem elforgatási adatát. „-1 -1 0” érték esetén van alaphelyzetben az elem.

3.5 Az egyeneseket detektáljuk.

```
dst = cv2.Canny(I_sz, 50, 200, None, 7)
linesP = cv2.HoughLinesP(dst, 1, np.pi / 180, 50, None, 4, 4)
```

9. ábra: Egyenes detektálása

Az egyenes detektálásához a forrás képet szürkévé kell konvertálni amit már a mintaillesztés résznél megtettünk. A szürkeállományos képet megkapja a *„Canny”*. A képen kívül a alsó és a felső küszöb értéket valamint a rekeszméretet.

Ezekután alkalmazzuk a *„HoughLinesP()”* függvényt amelynek a paraméterei a következők:

- Adott kép amin keressük az egyenest
- Vonalnak szánt vektor mérete
- Távolság pixel felbontásban
- A minimális küszöb amire „ráfogható”, hogy egyenes
- A minimális és maximális vonalhosszok

3.6 A talált egyenesek fájlba írása

```
if linesP is not None:
    for i in range(0, len(linesP)):
        l = linesP[i][0]
        f.write("Wire Wire Line\n")
        f.write("   "+str(int(l[0]*10.4166667))+" "+str(int(l[1]*10.4166667))+" "+str(int(l[2]*10.4166667))+" "+str(int(l[3]*10.4166667))+"\n")
```

10. ábra:Egyenesek kiírása fájlba

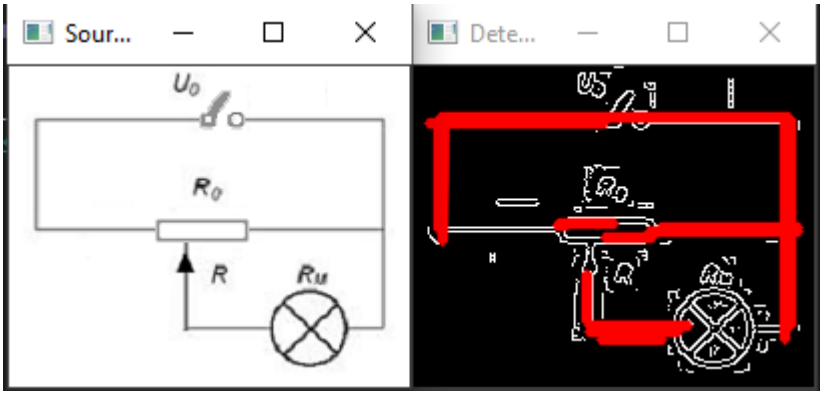
A *„HoughLinesP()”* által kapott értékeket a *„linesP”* változóban tároltuk el. Ezután egy *„if”* segítségével megnézzük talált-e egyenest a *„HoughLinesP()”* függvény. Ha talált, akkor indítunk egy *„for”* ciklust ami *„linesP”* hosszáig megy 0-tól. Az „l” változót beállítjuk *„linesP[i][0]”* értéknek.

KiCad a vonalakat úgy olvassa be, hogy a *„Wire Wire Line”* után egy tabulátorral beljebb kezdve tárolja az egyenes kezdésének koordinátáját és a végpont koordinátáját inchben.

Az egyenes kezdő és végpontját úgy kapjuk meg, hogy az *„l”* változó 0. és 1. indexű eleme lesz a kezdő pont, míg a 2. és 3. indexű elem a végpontja lesz az adott egyenesnek. Ezeket a koordináta pontokat átváltva inchbe majd 1000-el megszorozva megkapjuk azt az értéket amit a KiCad értelmezni tud és ezt az értéket íratjuk bele a fájlba.

4 Tesztelés

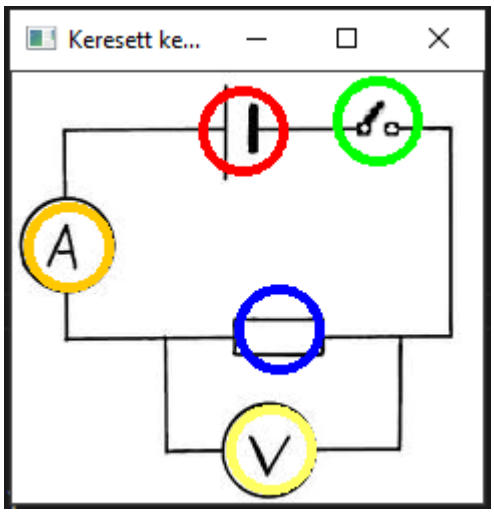
4.1 Egyenes tesztelés



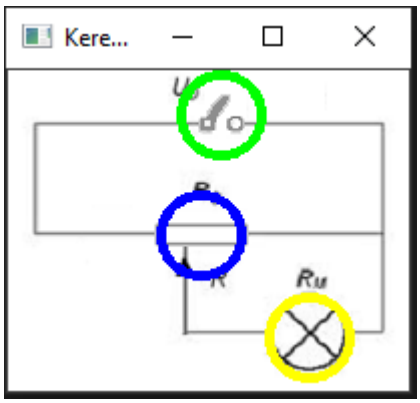
11. ábra: Egyenes detektálás eredménye

Az egyenes detektálása esetén elmondható pozitívnak, hogy az egyenesek nagy részét felismeri. A negatívum viszont az, hogy egyes elemekre mint például a kapcsolónál egyenesnek vette. Ez a HoughLinesP() függvény küszöbértékének növelésével lehet javítani, de akkor kevesebb egyenest detektál és ezáltal hiányosabb lesz a eredmény.

4.2 Mintafelismerés tesztelés

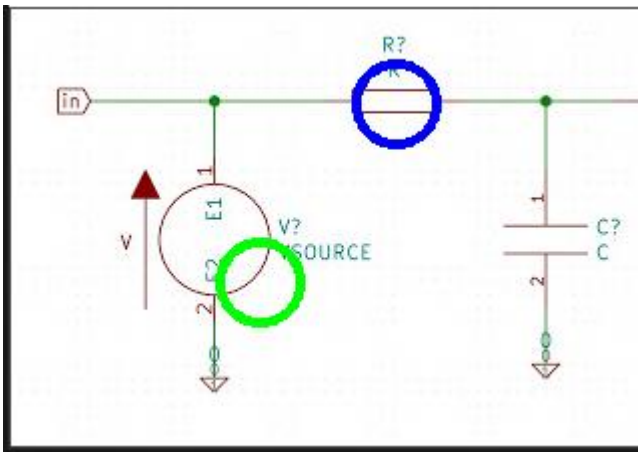


13. ábra: Mintafelismerés eredménye



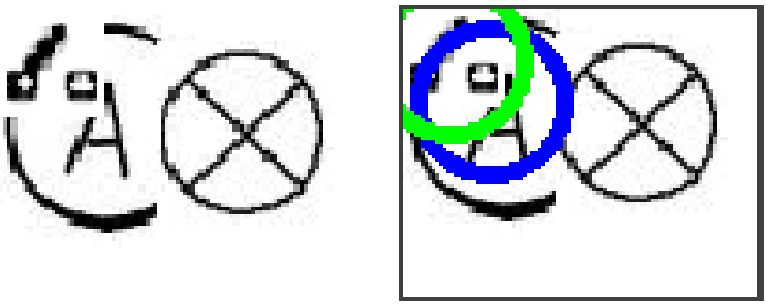
12. ábra: Mintafelismerés eredménye

A két képen jól látható, hogy az ismert elemeket szépen felismeri a program.



14. ábra: Nem ismert elemre ráteszi a kapcsolási elem detektálását

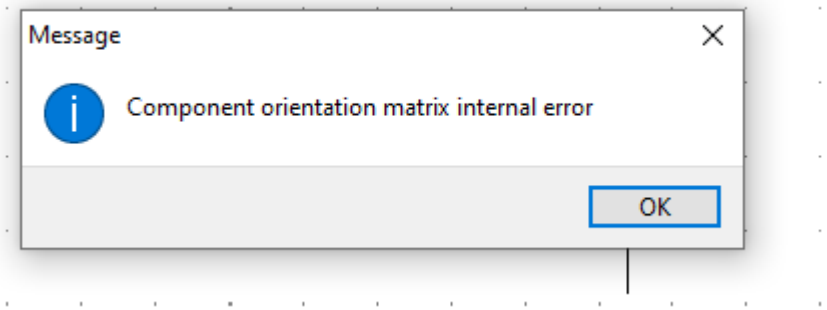
Sajnos a nem ismert elemnél a program a kapcsoló szára miatt ráillesztette a kör szélére. Ez az egyik hiba ami fellépett tesztelés során. Megoldásra az új elem felvétele és nagyobb tűrésérték beállítása lehetséges. De pozitívum hogy az ismeretlen elemre és a nyílra nem illesztette rá a megszakító elemet, valamint az Ampert, Volt és Lámpa elemet nem helyezte rá az új elemre.



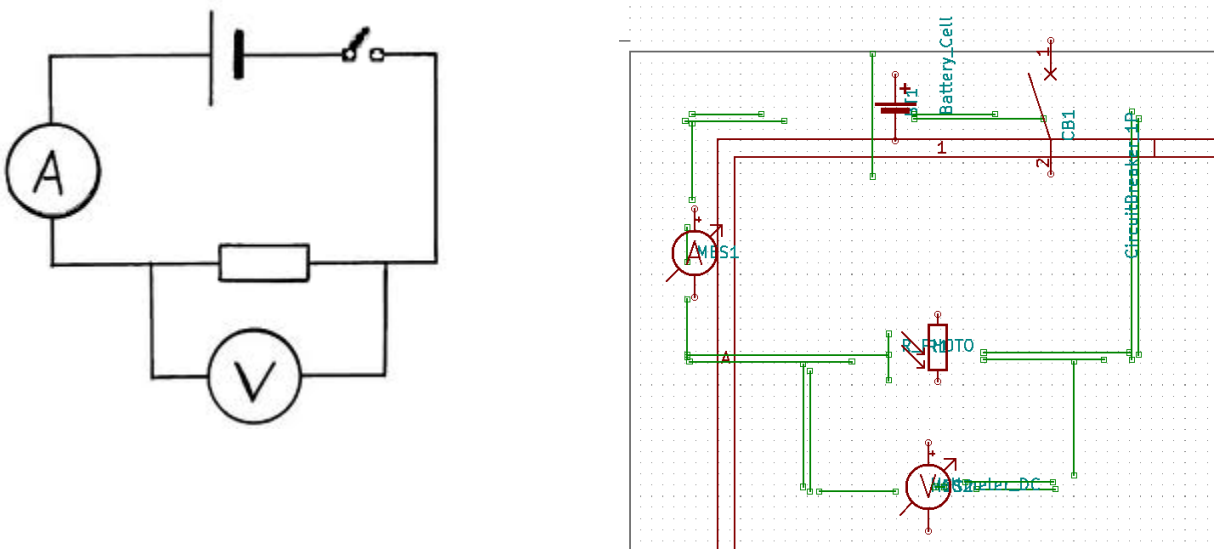
16. ábra: Hibás rajz

A képen a takarásban lévő ampert és a rajta levő kapcsolót felismerte, viszont a lámpa elemet nem érthető okokból nem detektálta. A küszöb érték állításával lehetséges a jobb eredmény.

4.3 KiCad importálás tesztelés és végeredmény



15. ábra: Importálás során fellépő hiba



16. ábra Eredeti és a program által előállított kép

A KiCad sajátos problémája, hogy a saját alkatrész tárából való elemeket nem szeretné elfogadni ahogy a felugró hibaüzenet ablak is jelzi. Viszont a 4. ok gomb lenyomására eltűnik. és megjelenik a rajz.

Az eredeti és a végeredmény összehasonlítása:

Az objektumok pozíciója pontosan ugyanott helyezkedik el. A forgatást sajnos a KiCad a saját általa készített rajzoknál is más projektbe való importálása során is rosszul állítja be.

Az egyenesek esetén mindenhol szinte kettőt láthatunk. Ez azért lehetséges, mert az egyenes detektálása során nem alkalmazzuk a *„blur()”* algoritmust. Az algoritmus elmossa a képet és egy él lesz látható, viszont a *„HoughLinesP()”* így rosszabb eredményt adna és például a Volt elemen blur-ral több egyenest is rajzolt volna.

Összefoglalva: Bár nem a legszebben adja vissza a rajzot , de minden fontos egyenes, objektum rajta van.

5 Felhasználói leírás

A program használatához következő lépéseket kell tenni:

1. Nyissa meg a forráskódot a *„ToKiCad.py”*-t egy szerkesztőben.
2. Az „I” értékadásnál az *„imread(” ”)”* belülre adja meg a beolvasandó kép nevét kiterjesztéssel.
3. Ezekután a 42. sorban megtalálja az *„f=open(”valami.sch”,”w”)”* kódsort, ahol a valami helyére azt adhatja meg, hogy milyen néven mentse el a konvertált fájlt.
4. Ha előző lépéseket megtette futtassa a forráskódot.
5. Futtatás befejezése után a forráskód könyvtárában megtalálja a generált fájlt.
6. Nyissa meg a KiCad szerkesztő programot, majd hozzon létre egy új projektet.
7. Projekt létrehozása után megjelenik baloldalt a tálcán egy projektnéven elnevezett sch kiterjedésű fájl, azt meg kell nyitni.
8. Megnyitás után a Fájl→ Kapcsolási rajz hozzáfűzése menüpont megnyitás.
9. Itt kiválasztja a generált fájlt.
10. Kiválasztás után felugró ablakon *„Use Relative Path”* majd *„Continue Load”*.
11. Itt feljön egy kis ablak amire 6x kattintva eltűnik és megjelenik a rajz.

6 Felhasznált irodalom

[1] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html
[2] https://drive.google.com/file/d/1AJH5m-dn1QKFUnnvMGJKf9sjp9xt3Ee8/view?usp=sharing
[3] https://drive.google.com/file/d/1DiwzgIX1_ssNL3VehNj4xEOQKwNcZkvw/view?usp=sharing
[4] https://drive.google.com/file/d/1-NtmLfnEHb77SLTbpoMlbywRY_EBJxhq/view?usp=sharing