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

Nyomtatott áramköri lemez vizuális analízisét segítő alkalmazás fejlesztése

Készítette
Ossik László

Konzulens
Dr. Orosz György

2020

TARTALOMJEGYZÉK

HALLGATÓI NYILATKOZAT

Alulírott Ossik László, szigorló hallgató kijelentem, hogy ezt a diplomaterv dokumentációt 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, 2020. 12. 18.

.....

Ossik László

Kivonat

Életünket az elektronikai eszközök teljes mértékben átszövik. Számos terület azok felhasználására és megértésére fókuszál, mint például az elektronikai szakemberek, villamosmérnökök. A megértéshez azonban nagyban hozzájárul az eszköz dokumentáltsága, amely sok esetben nehezen hozzáférhető. A dokumentáltság jelen esetben nyomtatott áramköri lemez szintjén értendő.

Az egyszerűbb áramkörök esetében a megfelelő analízis eléréséhez egyetlen kép felhasználása elegendő, amelyen jól kivehetők a huzalok, illetve a furat vagy felületszerelt alkatrészek. Ezek különböző képfeldolgozási technikák segítségével értelmezhetők. A dolgozat egy olyan alkalmazás fejlesztésén vezeti végig az olvasót, amely képes félautomatikus módon a felhasználóval közösen detektálni az áramköri lemezen a vezetékeket, beültetett áramköri elemeket, illetve az azok közötti kapcsolatokat. A folyamatot nyomtatott áramköri lemez visszafejtésnek, angolul (PCB reverse engineering-nek) nevezzük.

A dolgozat érinti a képfeldolgozás alapvető technikáit, hogyan működik a képszegmentáció azon belül a *Selective Search*, hogyan lehetséges egy képen előfeldolgozást végezni olyan szűrőkkel, mint *Gauss* vagy *Bilaterális* szűrő és hogyan tudunk létrehozni egyszerű, de jól átlátható felhasználói interfészt *Tkinter* segítségével. Végezetül pedig a félév során elért eredmények kerülnek bemutatásra.

Abstract

Our lives are completely interwoven with electronic devices. Many areas focus on using and understanding them, such as electronics professionals, electrical engineers. The documentation of the device, which in many cases is difficult to access, contributes greatly to its understanding. In this printed circuit level schamatics are meant by documentation.

For simpler circuits, it's sufficient to use only a single image, where the wires, through-holes or the surface mounted components can be easily observed to achieve proper analysis. These can be interpreted using different kind of image processing techniques. The dissertation guides the reader through the development of an application, which is able to detect wires, components and the connection between them in a semi-automatic way working together with the user. The process is called printed circuit board reverse engineering.

The dissertation touches the basic techniques of image processing, how image segmentation works within Selective Search, how it is possible to preprocess an image with filters such as Gaussian or Bilateral filters and how we can create a simple but clear user interface using Tkinter.

1. Bevezetés

Életünk minden ágát átszövik az elektronikai eszközök. Ott vannak otthonainkban, munkahelyünkön, még a zsebünkben is. Ezek az eszközök meghibásodnak, elhasználódnak, így számos olyan területre van szükség, amely a meglévő eszközöket analizálja, felméri annak állapotát és ennek ismeretében további folyamatokat indít el. Ilyen például gyártás során a különböző hibadetektáló berendezések, amelyek képesek komponensek közötti vezetékek, vagy esetleg rossz forrasztásból eredő hibát detektálni. A másik egyik legfontosabb terület az újrahasznosítás, hiszen sok menthető alkatrész található egy már nem működő áramköri elemen. Ezek megmentésével csökkentjük a környezeti szennyezettséget, a termék újbóli legyártásából származó költségeket, illetve ráfordított időt. 1 millió laptop megmentése körülbelül annyi energia megtakarításával ér fel, amelyet 3657 lakás fogyasztana Amerikában egy év alatt [1]. Illetve számos komponens kémiai anyagot tartalmaz, amelyek veszélyesek a környezetre.

A rendszer megalkotásának legfőbb indoka, hogy egy olyan lehetőséget teremtsünk, amely segítségével a felhasználó képes már elkészült nyomtatott áramköri lemez felépítését analizálni a mélyebb megismerés érdekében, annak ellenére is, hogy konkrét kapcsolási rajz nem áll rendelkezésre. Természetesen a rendszernek korlátokat szab számos tényező, például több rétegű nyák felismerése nem kivitelezhető, hiszen csak képfeldolgozási technikákkal dolgozik az alkalmazás. Emiatt fontos a megfelelő minőségű kép biztosítása, mert ez az egyetlen információforrás, amelyből az alkalmazás táplálkozik. A rendszer félautomatikus módon működik, amely azt jelenti, hogy a felhasználó közreműködése szükséges, az alkalmazás pedig bizonyos dolgokat automatikus módon hajt végre. A rendszer megfelelő kezelése érdekében elengedhetetlen egy könnyen átlátható felhasználói interfész, amely egyszerűen kezelhető, és a megfelelő parancsok könnyen végrehajthatóak.

A cél elérése érdekében különböző képfelismerési eljárást tanulmányoztam, amelyek közül a *Selective Search* és a *Non-maximum suppression* közös használata ígérkezett a legmegfelelőbbnek. A dolgozat részletesen taglalja a rendszer működését, továbbá a *Tkinter* segítségével előállított felhasználói interfész felépítését és használatát.

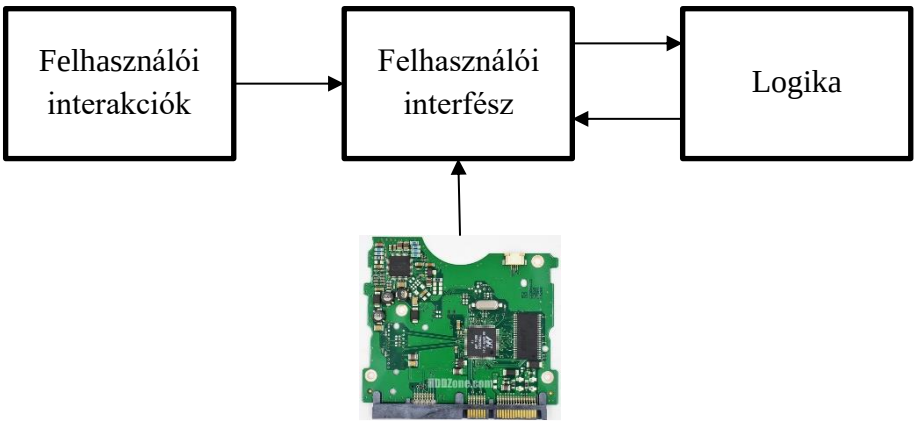
2. A megvalósított szoftver rendszerterve

A szoftver két fő részből tevődik össze. Az egyik ilyen fő rész a felhasználói interfész, amely segítségével a felhasználó képes interaktívan kezelni az alkalmazás nyújtotta lehetőségeket, míg a másik fő rész feladata a felhasználó által kért feladatok kiszolgálása. A megvalósítandó alkalmazás implementációjához Python [2] programozási nyelv került felhasználásra. Előnyei közé sorolhatók az alábbi tulajdonságok: platformfüggetlen, képfeldolgozás szempontjából sok beépített csomaggal rendelkezik, könnyen kezelhető, egyszerűen futtatható. További nagy előnye, hogy a Python lehetőséget nyújt Tkinteren [3] keresztül a felhasználói interfész megvalósítására, emiatt a rendszer logikai működése és annak képi megnyilvánulása egyetlen nyelv felhasználásával megvalósítható. A kialakított interfészről a 4. fejezetben részletesebb információk is megtalálhatók.

Az alkatrészek változatossága, a feldolgozandó kép minősége mind olyan tényezők, amelyek nehezítik a tökéletes felismerés elérését. A jelen diplomatervezés tárgy keretén belül nincs lehetőség egy olyan rendszer megalkotására, amely teljesen automatikus módon képes detektálni a képen látható objektumokat, így olyan funkcionalitások kerültek a rendszerbe, amely segítségével az automatikus döntések is felülbírálhatóak, illetve kiegészíthetőek felhasználó barát módon. Az alkalmazás a következő lehetőségeket kínálja:

- Jól átlátható kezelői felület, amely segítségével egyszerűen végrehajthatók a különböző műveletek,
- Nyomtatott huzalozású lemezekről készített képek beolvasása,
- Manuális segítséggel vezetékek detektálása,
- Alkatrészek forrszemeihez kapcsolódó vezetékek végpontjainak megtalálása,
- Az áramkörben fellelhető különböző komponensek detektálása,
- Adatbázisba beregisztrált komponensek, vezetékek, végpontok eltávolítása
- Alkatrészek közötti kapcsolatok beazonosítása és megjelenítése.

A felhasználói interfész a végrehajtani kívánt műveletek fogadása után, továbbítja azt az alkalmazás logikai része felé, amely értelmezi és végrehajtja a feladatot, majd a kívánt eredménnyel visszatér a felhasználói interfész irányába. Ezt a fajta működést az 1. ábra jól mutatja.



1. ábra - Az alkalmazás rendszerterve

A teljes rendszer eseményvezérelt. Az egyes események a felhasználó és a felhasználói interfész közötti interakcióból generálódnak. A logikai blokk egy állapotgépből áll, amelynek állapotát a generált esemény vezérli. A vezérlés történhet kattintással, illetve a billentyűzet adott gombjának lenyomásával is. Az egyes állapotokban a kívánt műveletek hajtódnak végre. Itt kapnak helyet az objektum detektálást végző algoritmusok is. A művelet elvégzése után pedig a felhasználó által látott képre maszkolások segítségével vetítődnek rá az eredmények.

3. Objektumdetektálás

Objektumdetektálás során a vizsgálandó tárgyról előzetes információkkal rendelkezünk, amely segítségével beazonosíthatóak az egyes képen látható komponensek. E művelet elvégzése tekintetében nagyon sok rendelkezésre álló eszközből válogathatunk. Előszeretettel használnak előre betanított mesterséges intelligenciát, ahol a felismerés mértéke a tanítás mértékétől függ. Azonban ennek a módszernek a legnagyobb hátránya a tanítandó minták összegyűjtése. Ez esetben több ezer képről van szó, és a kiépített adatbázisok hiányában nehezen kivitelezhető a megfelelő felismerés elérése. Azonban nem szükséges csak a mesterséges intelligencia által nyújtotta lehetőségekre szorítkozni. Számos olyan módszer használható, amely megfelelő mértékben képes elvégezni a kitűzött célt. Ebben a fejezetben ezek a felhasznált algoritmusok kerülnek bemutatásra, azok működése és alkalmazása.

3.1. Kép előfeldolgozása

Az objektumdetektálás elvégzése előtt szükséges némi előfeldolgozást végezni a képen, amely nagyban hozzájárul a felismerés mértékének növeléséhez. Ezt a lépést szinte minden képfeldolgozó rendszer használja. A feldolgozandó kép általában nagymértékű zajjal terhelt, amelynek forrása lehet fizikai eredetű. Ilyen fizikai forrás például a képet készítő készülék mechanikai problémái, az elektronok által keltett termikus zaj, amelyet a szenzor hűtésével csökkenthetünk, illetve a környezetből eredő rossz fényviszonyok is [4]. A kép minősége szintén tovább romlik, amikor az elektromos jelből digitális adattá válik a kép. A zaj csökkentése érdekében számos modell készült, amely szimulál különböző zajforrásokat, és ennek segítségével olyan szűrők kerültek előállításra, amelyek megfelelően képesek csökkenteni az előforduló degradációt.

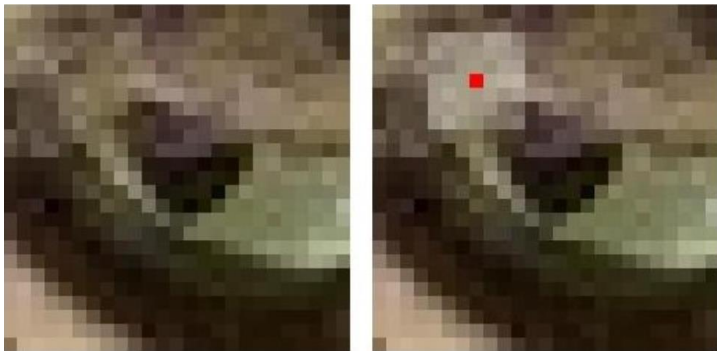
3.1.1. Zajcsökkentési módszerek

A képen detektálható zaj nem más, mint a fényesség és a színinformációk véletlen változása. Megjelenésének egyik főbb formája, hogy a zajjal terhelt kép pixelei világosabbak az eredetinél. Az alábbi 2. ábra jobb oldalán egy általam Matlabban generált 0.1 varianciájú gaussian zajjal terhelt kép figyelhető meg, míg baloldalt az eredeti kép helyezkedik el.

2. ábra – Bal oldalt zajnélküli, míg jobb oldalt a zajjal terhelt kép

3.1.1.1. Gaussian Blur

A Gaussian Blur szűrő segítségével eltávolíthatók a képről azok a túlzottan intenzív képpontok, amelyek nagy valószínűséggel a zajt tartalmazzák. Eredményül egy enyhén elmosódott képet kapunk. Aluláteresztő szűrő felhasználásával is hasonló eredmény érhető el. Ahogy az alábbi 3. ábra is mutatja egy úgynevezett kernel segítségével a kép egyes pixelein csúszóablakos módszerrel végig haladva alkalmazzuk a szűrőt, amelynek hatása mindig csak a pirossal megjelölt pixelre fejti ki hatását. A kernel magasságának és szélességének mindig páratlannak kell lennie, hogy meglehessen állapítani a kernel középpontját [9].



3. ábra – Kernel alkalmazása [9]

Gaussian Blur szűrő alkalmazása során a piros középponthoz közel elhelyezkedő pixelek nagyobb súllyal lesznek figyelembe véve, míg a távolabb lévők elhanyagolhatóbb mértékben. A Gauss függvényt az alábbi egyenlet írja le kétdimenzióban [10]:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2-y^2}{2\sigma^2}} \quad (1)$$

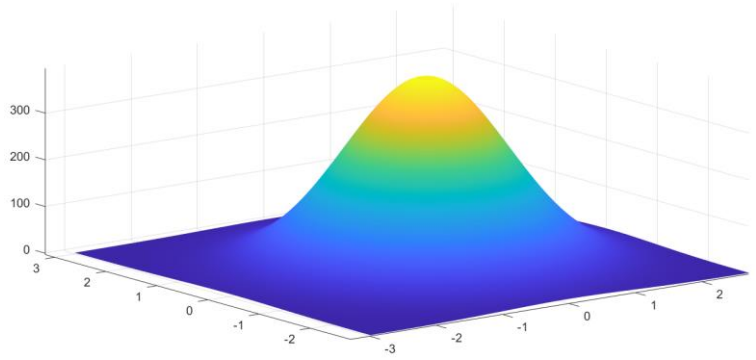
ahol

x, a vízszintes tengelyen megtett távolság

y, a függőleges tengelyen megtett távolság

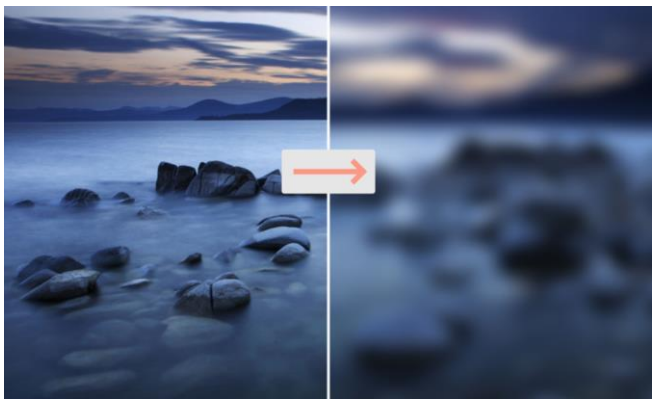
σ, a Gauss-eloszlás szórása (szigma)

A függvény csúcsosságát a szigma változó befolyásolja. Kisebb érték mellett sokkal csúcsosabb, míg nagyobb értéknél laposabb függvényt kapunk. A Gauss függvényt a 4. ábra mutatja háromdimenzióban [5].



4. ábra – 3D Gaussian függvény

Alkalmazásának eredményét az 5. ábra mutatja. Bal oldalt az eredeti kép, jobb oldalt pedig a szűrt kép.



5. ábra – Gaussian Blur alkalmazásának eredménye

3.1.1.2. Bilaterális Szűrő

A bilaterális vagy másnéven kétoldalas szűrőt előszeretettel használják olyan esetekben, ahol a cél az élek megtartása és a zaj csökkentése egyaránt. A korábban bemutatott eljárás nem teszi lehetővé az élek megtartását. A pixelek intenzitását a hozzájuk közel elhelyezkedő pixelek intenzitásának súlyozott átlaga alkotja, ahol a súly a korábban em-

lített Gauss-eloszláson alapul. Az élek megtartása érdekében a súlyok nem csak az euklideszi távolságtól függenek, hanem a képen található színintenzitások és mélységtávolságoktól is [11]. A kétoldalú szűrőt az alábbi egyenletekkel definiálhatjuk [11]:

$$I^{filtered}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|) \quad (2)$$

$$W_p = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|) \quad (3)$$

ahol,

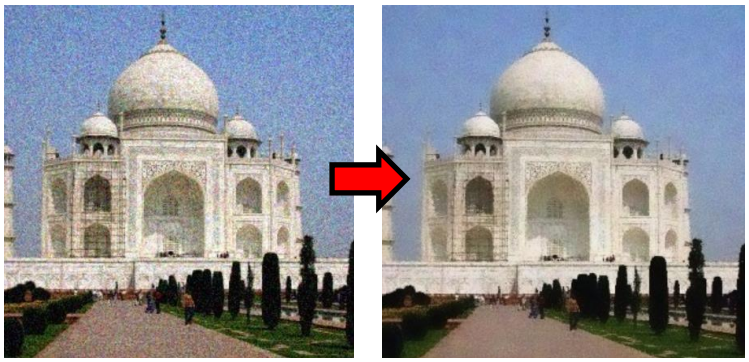
$I^{filtered}$, a szűrt kép

x, az aktuális szűrendő pixel koordinátái

f_r, az intenzitásbéli különbségek kiegyenlítésére szolgáló függvény

g_s, a térbeli vagy tartománybeli kernel a koordináták közötti különbségek simítására.

Alkalmazásának eredményét az alábbi 6. ábra szemlélteti.



6. ábra – Bilaterális szűrő alkalmazásának eredménye

3.2. Képszegmentálási módszerek

Képfeldolgozás során az egyik legfontosabb alapprobléma a képszegmentálás, amely során bizonyos tulajdonságok alapján az egymáshoz hasonló pixelek kerülnek csoportosításra. A probléma abban rejlik, hogy a szegmentálási folyamat szubjektív és a képről meglévő előzetes ismereteinktől függ [6].

Szegmentálás során 3 fő szintet különböztethetünk meg [6]:

1. Alacsony szintű képfeldolgozás:

- Ez esetben a képhez szervesen tartozó tulajdonságokat nyerhetjük ki és jeleníthetjük meg, általában külön képen, ilyen például a szín, élek, textúra.

2. Középső szint:
    - Ezen a szinten a képen észlelhető objektumok tulajdonságainak meghatározása a cél. Azt az információt, amelyből dolgozhatunk az alacsonyszintű képfeldolgozás biztosítja számunkra. Eredményként az objektumra vonatkozó információk (kontúr vonalak, egybezárt felületek, mozgás) fedezhetők fel.
3. Felső szint:
    - A kép tartalmának értelmezése, objektumok felismerése történik ezen szinten.

A képszegmentálást széles körben alkalmazzák. Egyik ilyen nagy terület az orvostechnológia, ahol ezt a technikát csontok, erek, szervek elkülönítésére használják [7]. További nehézségként jelenik meg az objektumok között elmosódott határok, intenzitás egyenlőtlenség, vagy akár a kép nem megfelelő felbontása.

3.2.1. Selective Search

Egy olyan módszer megtalálása volt a cél, amely előzetes tanítás hiányában is képes a képi információk alapján robosztus működésre. Az objektumok méretei, színei széles skálán mozoghatnak. A kép készítése során nem mindig egységesen fókuszált a kép, előfordulhatnak homályosabb részek, ahol az élek kevésbé kivehetők. A Selective Search az objektumokat hierarchikus módon keresi, amelyről részletesebb információk a 3.2.1. fejezetben találhatók. Előfordulnak olyan esetek is, hogy a detektálandó részek akár szín vagy textúra alapján is alkothatnak egy közös objektumot, továbbá a fényviszonyok mint például az árnyékok, és a megvilágító fény színe, mind befolyásolják a feltételezett objektumról alkotott képet. Egyszerre több algoritmus használata helyett a Selective Search a fentebb említett problémák mindegyikével felveszi a harcot, és egy egységes algoritmusként próbál javaslatokat tenni azokra a régiókra a képen, ahol a feltételezett objektum található [27].

3.2.1.1. Hierarchikus Csoportosító Algoritmus

A Selective Search hierarchikus csoportosítást alkalmaz ennek köszönhetően az objektumokat különböző méretben képes megtalálni és egyesíteni azokat [27].

7. ábra – A hierarchikus csoportosítás folyamata [29]

A teljes folyamat egy inicializálási fázissal kezdődik, amely során azok a régiók jönnek létre, ahol feltételezhetően elhelyezkedhetnek objektumok. Ez az úgynevezett Graph-Based algoritmus segítségével érhető el. Részletesebb információk a 3.2.1.2. fejezetben olvashatók. Az inicializálást követően a leghasonlóbb régiókat megkeresve egyet alkotunk belőlük és újabb hasonlóságok kerülnek kiszámításra az új régió, illetve a szomszédok között [27]. A teljes folyamat mind addig tart, amíg az apróbb részek egy nagy egységes objektumot nem alkotnak. Az algoritmus részletesebb működését az alábbi folyamat írja le [27]:

Bemenet: Színes kép

Kimenet: feltételezett objektumok (*R*)

1. *Előállítjuk a kezdeti szegmenseket* R = {$r_1, \cdots, r_n$} (Graph-based szegmentáció)
2. A hasonlóságot jelezzük *S*-el, amelynek kezdeti értéke 0

foreach minden szomszédos páron (r_i, r_j) **do**

hasonlóság meghatározása $s(r_i, r_j)$

$S = S \cup s(r_i, r_j)$

while $S \neq 0$ **do**

Keressük meg a leghasonlóbb részeket $s(r_i, r_j) = \max(S)$

Egyesítsük az összetartozó részeket: $r_t = r_i \cup r_j$

Távolítsuk el az r_i-hez tartozó hasonlóságokat: $r_i = S \setminus s(r_i, r_*)$

Távolítsuk el az r_j-hez tartozó hasonlóságokat: $r_j = S \setminus s(r_*, r_j)$

Határozzuk meg S_t -t az alapján, hogy mennyire hasonló r_t a szomszédaihoz képest

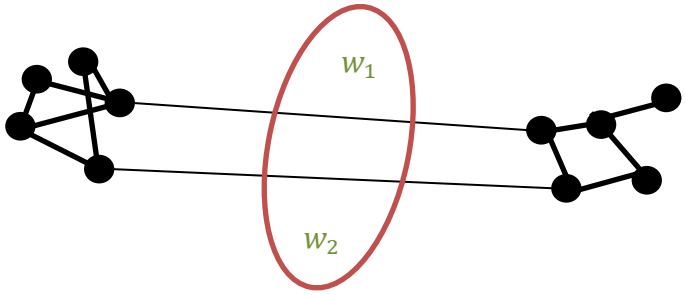
$S = S \cup S_t$

$R = R \cup R_t$

Az eredményt pedig az *R* tartalmazza, amelyben a megtalált régiók találhatók.

3.2.1.2. Graph-Based Algoritmus

A Graph-Based képszegmentációs technika a problémát gráfokban ábrázolja $G = (V, E)$, ahol minden egyes csomópont $v_i \in V$ egy pixelhez tartozik a képen, míg az éleket E reprezentálja és kapcsolja össze a szomszédos pixeleket. Az élek súlyait a pixelek közötti intenzitás határozza meg. A legelső Graph-Based algoritmus egyszerű küszöbértéket használt a különböző szegmensek elkülönítésére [29]. Zahn munkája során [28] azonban már a maximum folyam minimális vágás metódust használta. Ez a metódus mind a pontok csoportosítására és a képszegmentációra is alkalmas volt. Ennél a módszernél az élek súlyainak a pixelek közötti intenzitás különbséget választotta.



8. ábra – Minimális vágás során megtalálandó súlyok

Az algoritmus működése [29]:

A bemenet egy gráf $G = (V, E)$, ahol n a csúcsok száma, m pedig az élek száma. A kimenet pedig V szegmentálása $S = (C_{1,} \dots, C_r)$ komponensekbe.

1. Rendezzük E-t $\pi = (o_1, \dots, o_m)$-be nem szigorú monoton növekvő sorrendben.
2. A szegmentációt S^0-al kezdjük, ahol minden egyes csúcs v_i egy komponenst alkot.
3. Ismételjük meg a 4. lépést $q = 1, \dots, m$ lépésig.
4. Hozzuk létre S^q-t S^{q-1}-ből a következő módon. v_i és v_j-vel jelölt csúcsok legyenek összekapcsolva a q. éllel és jelöljük ezt $o_q = (v_{i,} v_j)$-vel. Ha v_i és v_j különálló elemei S^{q-1}-nek és $w(o_q), ami\ a\ v_i$ és v_j csúcsokat összekötő él súlya kisseb az egyes komponensekben megtalálható élekhez képest, akkor egyesítjük a különálló komponenst, ellenkező esetben nem teszünk semmit. Formálisan legyen C_i^{q-1} a S^{q-1} komponense, amely tartalmazza v_i-t és C_j^{q-1} komponens tartalmazza v_j-t. Ha C_i^{q-1} és C_j^{q-1} nem egyezik és $w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1}),\ akkor\ S^q$

nemmásból, mint S^{q-1}-ból származik, mégpedig úgy, hogy C_i^{q-1}-et és C_j^{q-1}-et egyesítettük. Máskülönben S^q egyenlő S^{q-1}-vel.

5. A visszatérési érték $S = S^m$

A 4. pontban megtalálható volt $MInt\left(C_i^{q-1}, C_j^{q-1}\right)$, ami nem más mint:

$$MInt(C_1, C_2) = \min\left(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2)\right) \quad (4)$$

$$Int(C) = \max_{e \in MST(C,E)} w(e) \quad (5)$$

$$\tau(S) = k \,/\, |C| \quad (6)$$

ahol

MST, Minimális feszítőfa (Minimum Spanning Tree)

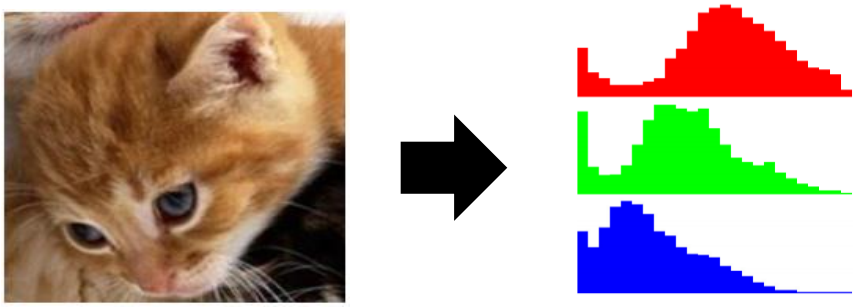
k, egy tetszőlegesen megválasztható konstans

$|C|$, a komponens mérete.

3.2.1.3. Tulajdonságok

A Selective Search az egyes régiók vizsgálata során számos tulajdonságot vesz figyelembe, amely segítségével képes megállapítani, melyek azok a területek, amik egymáshoz szorosan kapcsolódnak.

Az egyik legfontosabb jellemző a szín, amelyet jelöljünk $s_{colour}(r_i, r_j)$. Minden egyes régiónál meghatározunk egy egydimenziós hisztogramot annak minden szín csatornájára. Az egyes csatornákat pedig felosztjuk 25 részre [27]. Alább egy példa is látható:



9. ábra – Különböző csatornákra meghatározott hisztogram RGB esetén [27]

Így kapunk egy szín hisztogramot minden egyes régióhoz r_i, amelyet jelöljünk $C_i = \{c_i^1, \ldots, c_i^n\}$. Ennek dimenziója $n = 75$ lesz, hiszen 3 csatornánk van és 25-ös felosztást

alkalmaztunk minden egyes csatornára. Minden egyes hisztogram L1 szerint van normalizálva [27]. A színbéli hasonlóságot az alábbi egyenlet definiálja:

$$s_{colour}(r_i, r_j) = \sum_{k=1}^{n} \min\,(c_i^k, c_j^k) \tag{7}$$

A kapott új régió hisztogramja az alábbi módon számolható [27]:

$$C_t = \frac{size(r_i)\ \times\ C_i + size(r_j)\ \times\ C_j}{size(r_i) + \ size(r_j)} \tag{8}$$

Az új terület nagysága pedig egyszerűen az azt alkotó régiók nagyságának összege: $size(r_t) = \ size(r_i) + \ size(r_j)$

A következő legfontosabb mérték az objektumok textúrájának hasonlósága. Ezt jelöljük: $s_{texture}(r_i, r_j)$-el. Az anyagfelismeréshez általában az úgynevezett SIFT (Scale-invariant feature transform) algoritmussal történik [27]. Emellett minden egyes szín csatornára vesszük a Gauss deriváltját nyolc irányból $\sigma = 1$ mellett. Ez esetben az s_{colour}-hoz képest az egyes csatornák hisztogramjai tíz részre kerültek felosztásra. Így kapjuk meg végül a textúra hisztogramot $T_i = \{t_i^1, \dots, t_i^n\}$ a három különböző csatornára. A dimenziók száma ezesetben $n = 240$. A régiók szintén r_i-vel kerültek jelölésre. A normalizálás pedig szintén L1-el történik. A végső kifejezést az alábbi egyenlet definiálja [27]:

$$s_{texture}(r_i, r_j) = \sum_{k=1}^{n} \min\,(t_i^k, t_j^k) \tag{9}$$

Ahogy korábban említésre került egy képen az objektumok különböző méretekben fordulhatnak elő, és az algoritmusnak erre is fel kell készülnie. A következő fontos szempont a méret figyelembevétele. Ez esetben az algoritmus a kisebb régiókat hamarabb olvasztja egybe egymással. Ennek köszönhetően elkerülhető az a probléma, hogy a nagyobb területek magukba olvasztják a kisebb régiókat. A méret $s_{size}(r_i, r_j)$ a kép egy adott részére vonatkozik, amelyet r_i és r_j töltenek ki [27]:

$$s_{size}(r_i, r_j) = 1 - \frac{size(r_i)\ + size(r_j)}{size(im)} \tag{10}$$

ahol $size(im)$ a kép mérete pixelben mérve.

A szín, textúra és méret mellett az algoritmus figyelembe veszi, hogy r_i és r_j mennyire olvaszthatók egymásba úgy, hogy mekkora hézagokat hagynak egymás között. Definiálunk egy szűk négyzetet, amely körülöleli r_i és r_j területet. Nevezzük ezt a négyzetet BB_{ij}-nek. Ezt a teljes képhez és a szűk négyzethez viszonyított kitöltöttséget pedig az alábbi egyenlet írja le (11) [27]:

$$s_{fill}(r_i, r_j) = 1 - \frac{size(BB_{ij}) - size(r_i) - size(r_j)}{size(im)} \tag{11}$$

A végleges hasonlóságot pedig r_i és r_j között egy egyszerű összeadás írja le:

$$\begin{aligned} s(r_i, r_j) = {} & a_1 s_{colour}(r_i, r_j) + a_2 s_{texture}(r_i, r_j) + a_3 s_{size}(r_i, r_j) \\ & + a_4 s_{fill}(r_i, r_j) \end{aligned} \tag{12}$$

ahol $a_i \in \{0, 1\}$ azt adja meg, hogy az adott hasonlósági faktor használatban van-e vagy sem.

3.2.2. Canny éldetektáló módszer

Elmondható, hogy azok a pontok a képen, amelyek fényereje élesen megváltozik, általában görbe vonalszakaszokba rendeződnek. Ezeket az összefüggő pontokat éleknek nevezzük. Az élek ismeretében pedig megállapíthatóak sok esetben a képen előforduló objektumok helyzete. A Canny a színes kép helyett fekete fehér képpel dolgozik, így az algoritmus használata előtt szükséges a kép átkonvertálása.

A Canny éldetektáló módszer négy fő lépésből tevődik össze [12]:

- Kép kisimítása Gauss szűrővel
- Gradiens nagyságának és irányának meghatározása a képen.
- Non-maximal suppression (Nem-maximális elnyomás) alkalmazása a gradienseket tartalmazó képen.
- Hiszterézises küszöbölés használata a megfelelő élek megtartásához.

Ahogy a 3.1.1.1. fejezetben már bemutatásra került Gauss szűrő az egyik legalkalmasabb módszer a képen megjelenő zaj kiszűrésére. Ehhez egy 3x3, 5x5, 7x7 stb… mátrix használható, amelynek mérete attól függ, hogy milyen mértékben kívánjuk kisimítani a képet. Minél kisebb a mátrix mérete, annál kevésbé lesz homályos a feldolgozandó kép.

Hasonló hatással van a σ értéke is. Nagyobb érték mellett a zaj jobban elnyomódik, de figyelembe kell venni azt is, hogy a vékonyabb élek így eltűnhetnek a képről.

A gradiens számítása során úgynevezett éldetektáló operátorok használhatók, amelyek segítségével meghatározhatók az élek intenzitása és azok iránya. A megfelelő szűrők által megkaphatóak az intenzitás változások mind *x* és mind *y* irányban egyaránt. Ehhez az úgynevezett *Sobel* operátorok használhatók. A horizontális irányú intenzitásváltozás meghatározásához a (13) egyenlet, míg vertikális irányhoz a (14) egyenlet használható [13].

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \tag{13}$$

$$S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{14}$$

A mátrixokat a képpel konvolválva megkapjuk I_x-et és I_y-t. A konvolválás az alábbi módon működik:

100	100	200	100
100	100	200	200
100	100	200	200
100	200	200	100

-1	0	1
-2	0	2
-1	0	1

Bal oldalt egy a képet reprezentáló mátrix, míg jobb oldalt az S_x látható. Az egymással átlapolásban lévő cellák összeszorzásra kerülnek, majd az egyes számokat összeadjuk. Ennél a példánál az eredményt, így kapjuk: -100-200-100+200+400+200=400. A kialakult 2x2-es mátrix pedig az alábbi módon nézne ki:

400	300
400	200

I_x és I_y meghatározása után a gradiens nagyságát és meredekségét a következő egyenletekkel fejezhetők ki [12]:

$$|G| = \sqrt{I_x^2 + I_y^2} \tag{15}$$

$$\theta(x, y) = \arctan\left(\frac{I_y}{I_x}\right) \tag{16}$$

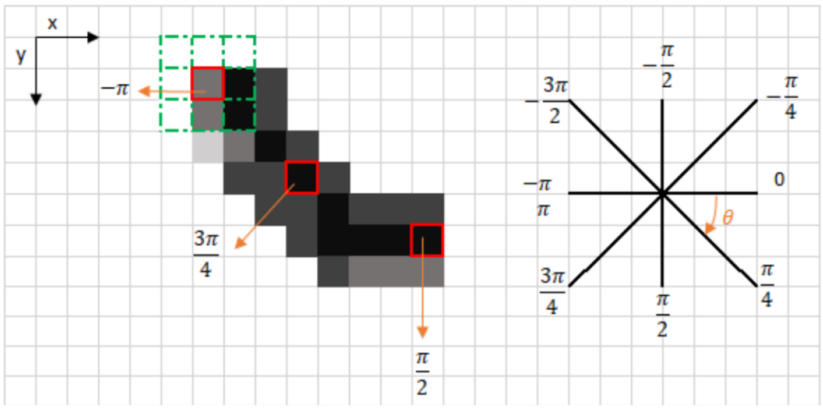
A gradiens nagyságának és meredekségének meghatározása után előállított eredményt a 10. ábra szemlélteti. Megfigyelhető, hogy előfordulnak erősebb, illetve vékonyabb élek is. Annak érdekében, hogy egységesen eldönthetővé váljon mik a tényleges élek Non-maximum suppression-t alkalmazunk.

10. ábra – Gradiens nagyságát tartalmazó kép [13]

Az algoritmus az alábbi módon működik [12]:

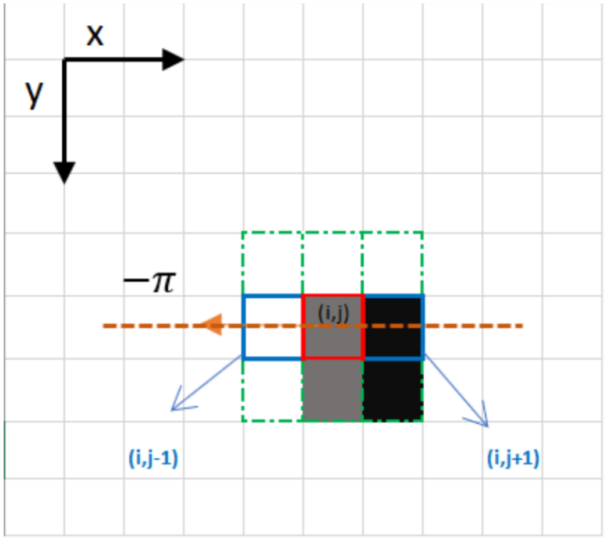
- A gradiens mátrix nagyságával megegyező mátrixot inicializál 0-kal.
- Meghatározza az élek irányát a szögmátrix segítségével.
- Megvizsgálja, hogy az irány mentén a szomszédos pixelek intenzívebb értékkel rendelkeznek-e vagy sem. Ha igen, akkor 0-ra állítja az aktuális pixel értékét, ha azonban az aktuális pixel intenzívebb, akkor nem változtat rajta. Ezzel a módszerrel halad végig a pixeleken.

Az alábbi példát megfigyelve látható, milyen irányban halad a feltételezett él, amelyet jelenleg a sötét pixelek alkotnak. Ezek a pixel értékek a gradiens mátrixból származnak, míg a narancssárgával jelölt nyilak pedig azok a szögek, amik a korábban kiszámolt szögmátrixban találhatók.



11. ábra – Gradiens mátrix és irány mátrix vizsgálata [12]

Vizsgáljuk meg közelebbről egy szeletét a fenti képnek. A vizsgálandó pont *(i,j)* piros négyzettel kiemelve látható. Az irányt a narancssárga egyenes mutatja. Ezek alapján a két megfigyelendő szomszéd *(i, j-1)* és *(i, j+1)* kék négyzettel vannak jelölve. A következő lépés, hogy az algoritmus megállapítsa, hogy mely pixel intenzívebb a két szomszéd közül. Egyértelműen látható, hogy az *(i, j-1)* pixel a sokkal intenzívebb. Mivel sikerült olyan szomszédos pixelt találni, amely magasabb értékkel rendelkezik *(i,j)* értékénél, így 0-ra állítjuk azt, amely a fekete színnel egyenértékű.



12. ábra – Egy pixel és környezetének vizsgálata [12]

A Non-maximum suppression alkalmazása után az alábbi eredmény látható a képen.

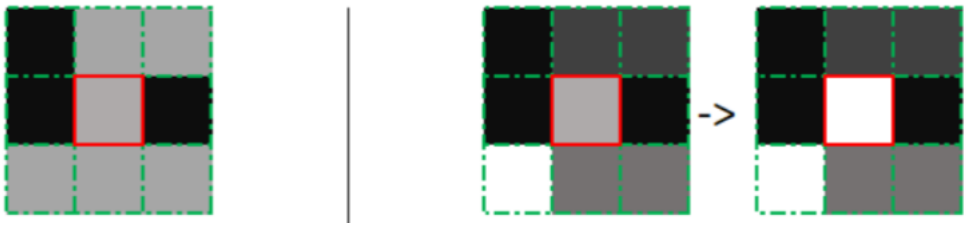
13. ábra – Non-maximum suppression alkalmazásának eredménye [13]

Az figyelhető meg, hogy a korábban vastagabb élek vékonyabbak lettek és még mindig van jelen zaj a képen. Megfelelő küszöbérték megtalálásával, azonban ezek a részek is eliminálhatók.

A következő lépésben tehát alkalmazunk két küszöbértéket, amely segítségével detektálhatjuk a különböző fajta vonalakat. Három fajta pixel típust különítünk el [12].

- Erős pixelek: Azok az élek, amelyekről egyértelműen eldönthetők, hogy hasznos információval rendelkeznek számunkra és megtartásuk elengedhetetlen.
- Gyenge pixelek: Nem elég intenzívek ahhoz, hogy erősnek legyenek mondhatók, de nem is túl kis értékűek, hogy ne legyenek eldobhatók.
- Nem releváns pixelek: Azok a pixelek, amelyeket eldobhatunk. Nem rendelkeznek többletinformációval.

Ezek alapján a küszöbértékeket úgy kell megválasztani, hogy a felső küszöbérték felett az erős pixelek helyezkedjenek el, a nem releváns pixelek pedig az alsó küszöbérték alatt. A hiszterézises élkereső módszer segítségével lehetőség nyílik a gyengébb éleket is erősebb élekhez kapcsolni. A módszer úgy működik, hogy az egyes pixeleken végig haladva, ha annak szomszédos pixele erősek, akkor az aktuális pixel is erőssé válik, míg erős pixel szomszéd hiányában az aktuális pixel gyenge marad, ahogy azt a 14. ábra is mutatja.



14. ábra – Gyenge pixelek erőssé válásának feltétele [12]

A Canny algoritmus végeredménye pedig alább látható:

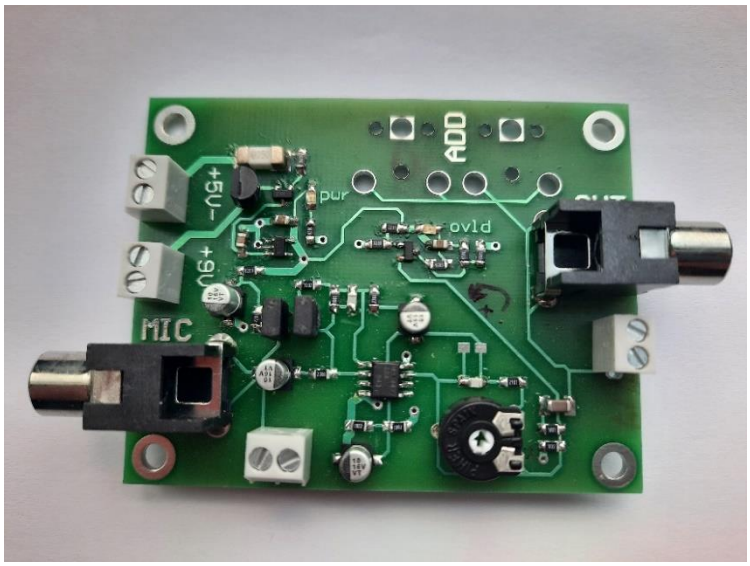
15. ábra – Canny éldetektáló algoritmus eredménye [13]

3.3. Vezetékek detektálása

Különböző éldetektáló algoritmusok megismerése után az elsődleges és egyik legfontosabb feladat a vezetékek felismerése volt. A cél ezesetben megtalálni, hogy melyik vezetéket, mely pixelek alkotják, illetve beazonosítani, hogy hol találhatók a vezetékekhez tartozó végpontok. A feldolgozandó kép esetében fehér hátteret próbáltam használni, hogy a NYÁK (Nyomtatott áramköri lemez) minél jobban kivehető legyen a háttérből.

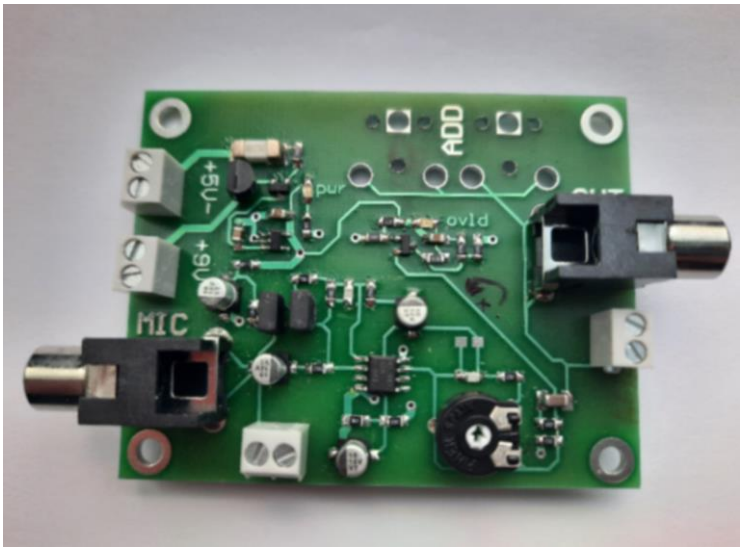
3.3.1. Éldetektálási módszer alkalmazása

A kép vizsgálata előtt érdemes némi zajcsökkentést alkalmazni. Hasonlítsuk össze az eredeti képhez képest milyen eredmény érhető el Gauss, illetve bilaterális szűrő segítségével. A feldolgozandó kép az alábbi 16. ábra illusztrálja.



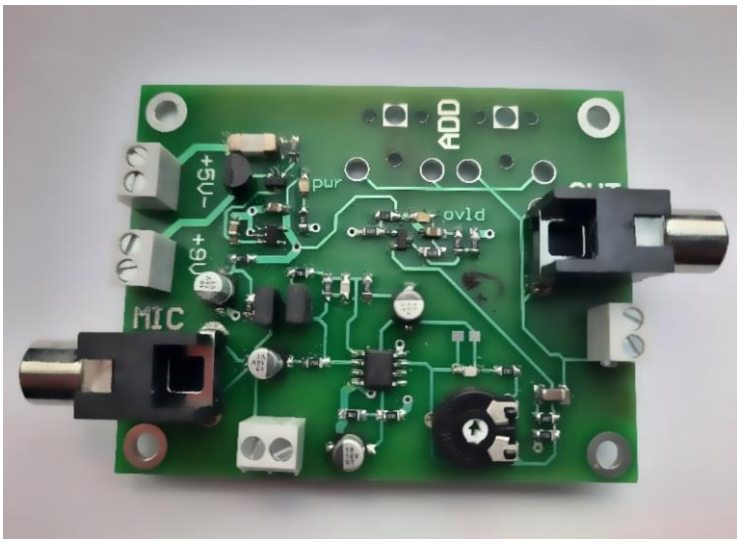
16. ábra – Feldolgozandó kép

Gauss szűrő esetében egy 5x5-ös mátrix került felhasználásra, amelynek méretét az OpenCV-ből [8] hívható cv2.GaussianBlur [14] függvénynek kell paraméterül átadni. Az eredményt alábbi 17. ábra mutatja.



17. ábra – Gaussian szűrő alkalmazása

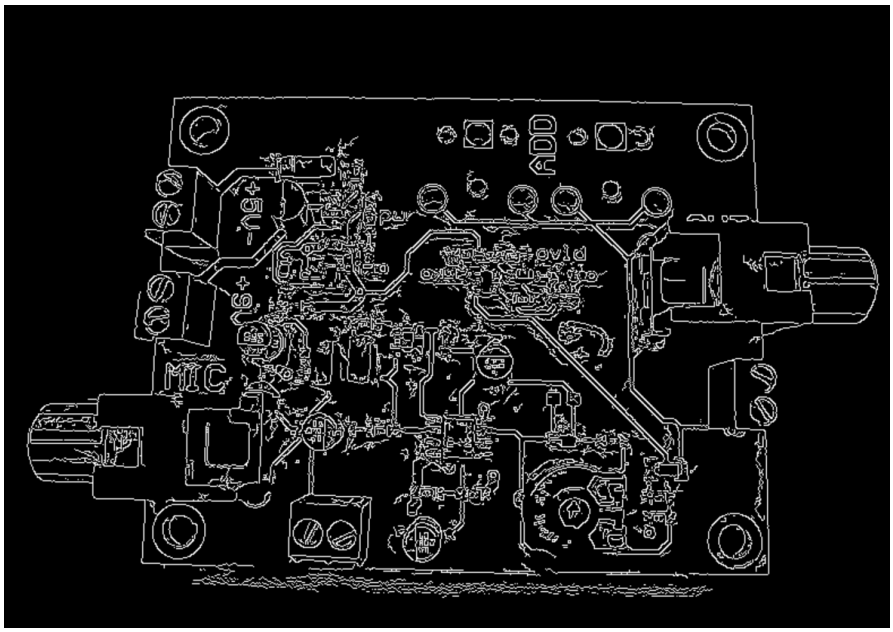
Ahogy az látható a kép teljes egésze elhomályosodott, amely zajcsökkentés szempontjából hasznos, azonban a kontúrok detektálásának hatását minimálisan rontja. A bilaterális szűrő esetében azonban az éleket megtartva az egybefüggő régiók kisimultak. Eredménye alább látható.



18. ábra – Bilaterális szűrő alkalmazása

Mivel az élek számunkra fontos információ tartalommal bírnak, így a továbbiakban a bilaterális szűrővel dolgoztam tovább a Gauss szűrő helyett.

A Canny használatához a képet fekete fehérré kell átkonvertálni, majd az algoritmus futtatható is. OpenCV-ben erre is található beépített függvény, ahol elsősorban paraméterként a fekete fehér képet, a hiszterézises küszöbérték alsó, illetve felső határát adhatjuk meg. A Canny algoritmus által előállított eredményen a kontúrok megtalálása a következő feladat, amelyet az OpenCV-ben elérhető cv2.findContours [15] valósítok meg. A függvény a [16] algoritmus alapján működik. Az alábbi képen a Canny által szolgáltatott éldetektáció eredménye figyelhető meg.



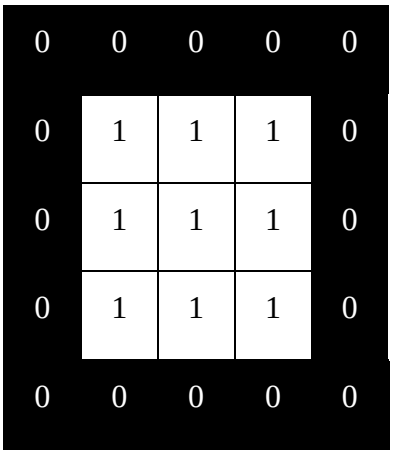
19. ábra – Canny éldetektálás eredménye

Ahogy a 19. ábra is mutatja annak ellenére, hogy szemmel jólkivehetőek az egyes komponensek illetve vezetékek, az árnyékolások miatt a kép nagy részén az előfeldolgozás ellenére is nagymértékű zaj van jelen.

20. ábra – Árnyékolás miatt jelenlévő zaj éldetektálás után

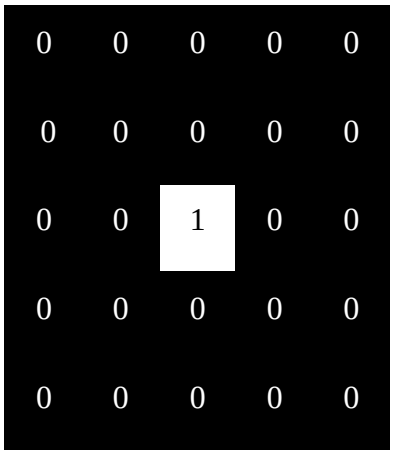
Az OpenCV biztosít számunkra további lehetőségeket a nemkívánatos zaj eltüntetésére. A két leggyakoribb morfológiai transzformáció a dilatáció és az erózió. Nemcsak zajcsökkentés, de kisebb területek egybeolvasztására is használhatók.

Az erózió és dilatáció esetében is egy kernelt definiálunk, amelyet az előállított képünkkel konvolválunk. Konvolválásról a 3.2.2 fejezetben esett szó. A kernel a képen végig haladva a középpontjában elhelyezkedő pixel értékét változtatja különböző feltételek mellett. Erózió esetén azokat a pixeleket hagyja meg ’1’-esként, ahol a kernel egészére vetítve mindenhol ’1’ szerepel. Minden más esetben ’0’ érték kerül beállításra. Dilatáció estében azonban azokon a helyeken lesz ’1’ a pixel értéke, ahol a vizsgálandó kép és a kernel átlapolódása során legalább egy pixel azonosan ’1’-es értéket vesz fel. Az alábbi példán bal oldalt található a feldolgozandó kép mátrix formájában, míg jobb oldalt az előállított 3x3-as kernel figyelhető meg.

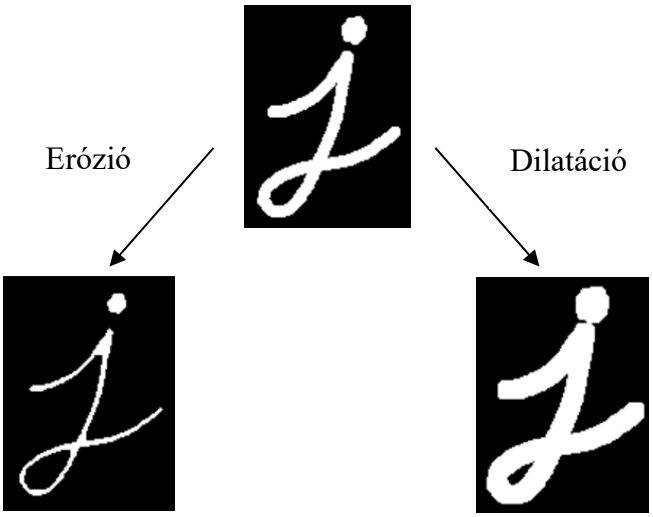


0	1	0
1	1	1
0	1	0

A kettő konvolválásának eredménye erózió, illetve dilatáció esetében:



0	1	1	1	0
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
0	1	1	1	0



21. ábra – Erózió és dilatáció hatása [17]

E két operáció variációiból számos további művelet végrehajtható, mint például opening, closing, morphological gradient, top hat és black hat [17]. Ezek közül az opening és a closing került felhasználásra. Az opening esetében először egy erózió, majd dilatáció

kerül végrehajtásra. Ez a művelet elsősorban további zajcsökkentés szempontjából alkalmazandó. A 22. ábra ennek eredményét mutatja.

22. ábra – Opening hatása [17]

A closing pedig ennek ellentétje. Előbb dilatáció, majd erózió kerül alkalmazásra. Ennek segítségével nem teljesen egybefüggő részek összeolvasztása valósítható meg. Eredményét a 23. ábra mutatja.

23. ábra – Closing hatása [17]

A zaj nagy kiterjedtsége miatt annak eltűntetése érdekében az opening alkalmazása nem csak a zajt, de a hasznos kontúrok java részét is eltűntette. További nehézségként merült fel, hogy a zaj sikeres eltávolítása után, milyen tulajdonságok alapján lehetne csak a vezetékeket detektálni. A cél elérése érdekében a problémát más oldalról kellett megközelíteni.

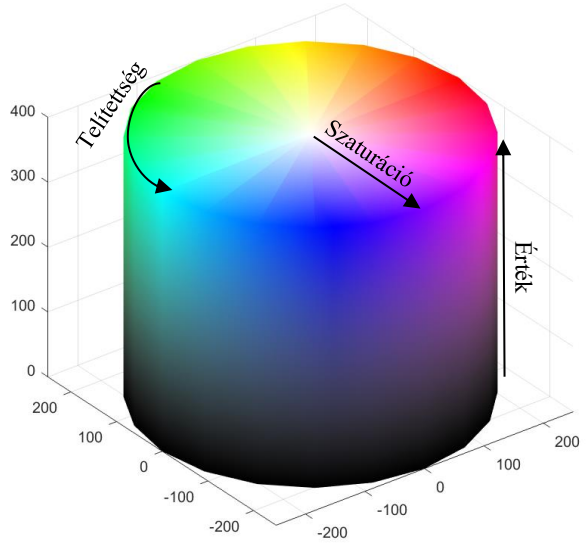
3.3.2. Régió-növesztő algoritmus bemutatása

A régió-növesztés a régió alapú szegmentálási eljárások közé tartozik. A régió alapú szegmentálás során a rendelkezésre álló *K* képet *n* darab összefüggő homogén $L_1, \dots, L_n$ szegmensre osztjuk [18]. Ehhez definiálhatunk egy homogenitási kritériumot, amely megadja, hogy egy *L* régión belül található képelemek hasonló tulajdonságokkal bírnak-

e vagy sem. Abban az esetben, ha hasonlóak akkor homogénnek tekinthető a régió, ellenkező esetben nem. Többféle kritérium is előírható, például a régió homogén, ha az alábbiak egyike legalább teljesül. Jelöljük a régión belüli részek értékét P-vel [18]:

- A régión belül elhelyezkedő pixel értéke minimálisan tér el a régió átlagos értékétől. $| P_i - L_{k,mean} | < P_{maxdiff}$
- Ha $| P_{max} - P_{min} | < P_{maxdiff}$
- Ha a σ_L szórás a régióban kis értéket vesz fel.

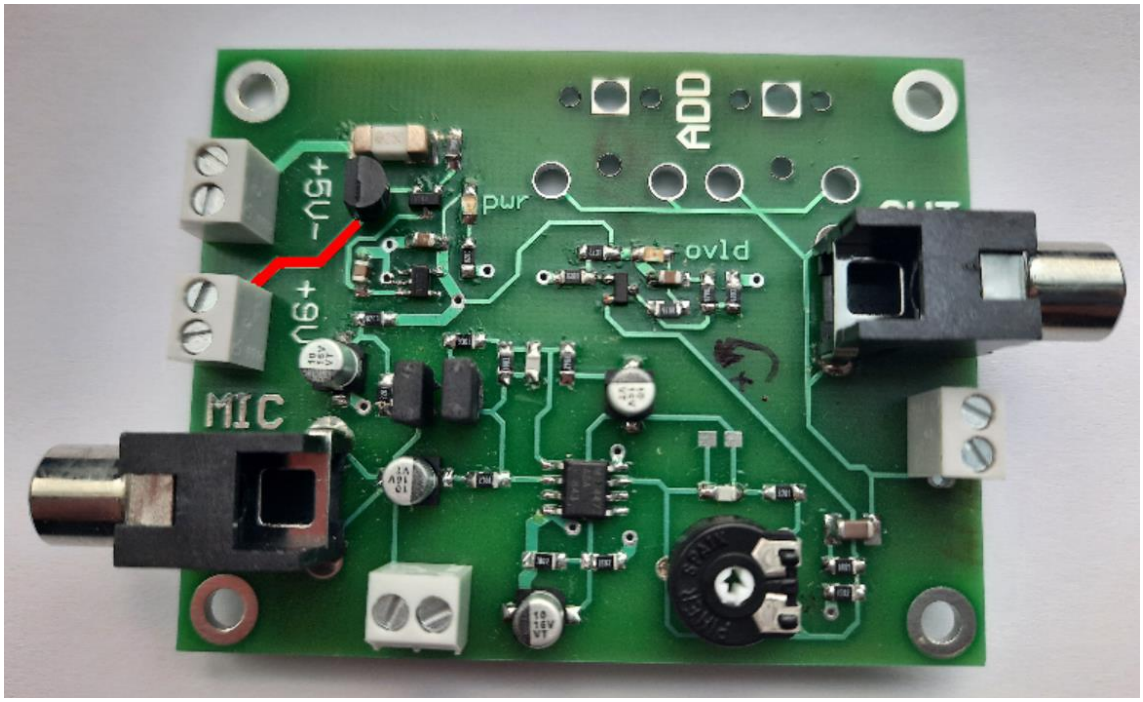
A régióban található pixelek összehasonlításakor nem a megszokott RGB (Red, Green, Blue) kép kerül feldolgozásra, hanem a kép beolvasása során HSV-be (Hue/Telítetség, Saturation/Szaturáció, Value/Érték) konvertáljuk át. A HSV szín alapú összehasonlítása sokkal előnyösebb, hiszen itt a szín információ elkülönül a különböző fényviszonyokból eredő információktól. A homogenitás kritériumban található pixel értékek esetében mind a H, S, V-re is külön küszöbérték került definiálásra, amelyet $P_{maxdiff}$ jelképez.



24. ábra – HSV cilinder

Régió-növesztés esetében a kiindulási pont egy vagy több magpont, amelyet bizonyos tulajdonság alapján megválaszthatunk a képen. Ez a magpont egyetlen pixelnek feleltethető meg. Definiálható, hogy 4 vagy 8-as szomszédságot vizsgálunk. 4-es szomszédság esetében a pixel éleihez kapcsolódó pixelekről beszélünk, míg 8-as szomszédság esetében a képpont csúcsaihoz tartozó pixelek is hozzátartoznak. Kezdetben az algoritmus a magpont és a szomszédok összehasonlítását végzi el. Rekurzív módon az újonnan régióhoz

csatolt pontok szomszédjait tovább vizsgáljuk, majd ezek szomszédjait is mind addig, míg el nem jutunk egy olyan pixelig, amely már nem hasonló a korábbi elemekhez. A régió növekedésével párhuzamosan az eddig megtalált pontok HSV értékeinek szórását és átlagát folyamatosan számítja az algoritmus, hogy mint plusz információ segítsen a megfelelő pontok megtalálásában. A kezdeti magpont kialakítása a felhasználó segítségével történik. Ezesetben egy olyan helyre kattint, amely biztosan vezetékhez tartozó pixel. Ebből terjed ki a régió. Az alábbi ábrán a vezetékre kattintva, megadva a kezdeti magpontot, a régi kiterjedésének eredménye látható.



25. ábra – Régió-növesztés eredménye egy vezetékre kattintva

A módszer segítségével legalább annyiszor kell a felhasználó által kattintani, ahány darab vezeték előfordul a NYÁK-on. A folyamat gyorsítása érdekében az első régióból meghatározott átlag és szórás értékekből a kép teljes egészére újabb magpont javaslatokat alkothatunk, amelyre ismételten alkalmazva a régió-növesztés módszert más vezetékeket is detektálhatunk. A nehézség az alábbiakban rejlik:

- Azon tulajdonságok megtalálása, amely a legjobban leírják az egy régióba tartozó pixeleket.
- A megfelelő $P_{maxdiff}$ megválasztása.
- Javaslat tétel más magpontok helyére minimális hiba mellett.

Az újabb magpontok azonban sok esetben olyan pontok, amelyek nem vezetékekhez tartoznak. Általában a rossz javaslatok a NYÁK széleihez közel helyezkedtek el. Az ezekből történő kiterjedés során pedig sokszor a háttérként szolgáló fehér terület került egy régióként felismerésre. A hibás javaslatok csökkentése érdekében 3.3.1 fejezetben megismert morfológiai opening-et alkalmaztam. Ezesetben a hibás pontok nagyrésze eliminálódott, azonban ezzel együttesen számos jó javaslat is törlésre került. Egy másik ötlet a rossz magpontok eldobására, hogy minden magpont egy külön osztályhoz tartozik. Az egyes pontokból kiterjedünk és a hozzá hasonló pixeleket szintén azonos címkével látjuk el. A terjedést folyamatosan figyelemmel kísérve, ha a régió elér egy bizonyos nagyságot, akkor elmondható, hogy valószínűleg a kiindulási magpont egy rossz javaslat volt és az összes pixelt, amelyek ehhez a régióhoz tartoznak a címke alapján töröljük. Annak ellenére, hogy a nagyobb régiókká növekvő rossz javaslatok eltűntek, további problémaként merült fel, hogy azok a területek, amelyek kisebb részekben az előírt határérték alatt maradtak és zajként jelentek meg, az algoritmus nem volt képes detektálni és eltávolítani azokat. Ezokból kifolyólag a tiszta és egyértelmű működés érdekében újabb magpontokat a szoftver nem javasol, ezzel csökkentve a plusz aktivitást a felhasználó részéről, amely a rossz pontok eltávolítására irányul.

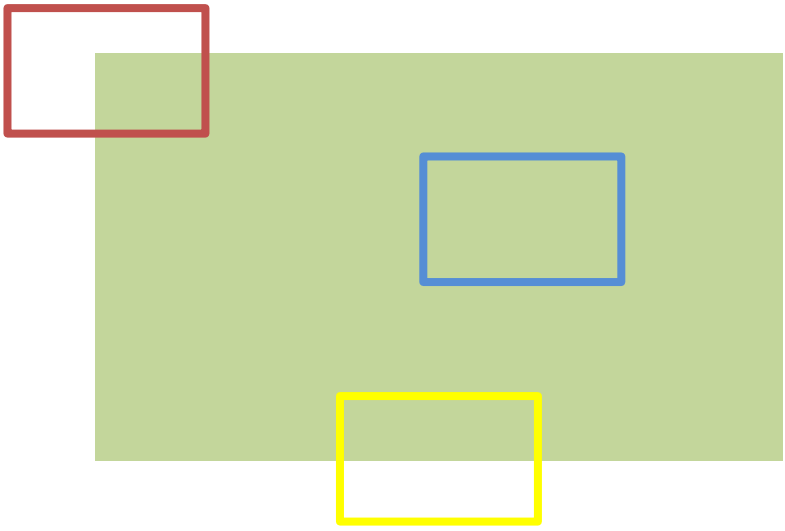
3.3.3. Végpontok felismerése

A célunk a kapcsolatok detektálása az egyes objektumok között. A kapcsolatok megteremtéséhez azonban tudnunk kell, hogy mely vezetékek kötik össze az egyes komponenseket. A következő 3.3.4. fejezetben bemutatásra kerül, hogyan kivitelezhető az objektumok felismerése, míg ebben a fejezetrészben a vezetékek végpontjainak detektálására szolgáló algoritmus kerül ismertetésre.

A régiónövesztés során a vezetékhez tartozó pixeleket megállapítjuk, címkét rendelünk a csoporthoz, amely segítségével hivatkozhatunk rájuk és végezetül egy maszk generálás történik meg, amelyet az alábbi 26. ábra jól mutat. Ezt kapja bemenetként a végpontokat meghatározó algoritmus.

26. ábra – Maszk generálás eredménye régió-növesztés után

A felhasznált Good Feature to Track névre hallgató algoritmus J. Shi és C. Tomasi nevéhez fűződik, amely a Harris sarokdetektáló algoritmus továbbfejlesztett változata [19]. A sarkok olyan régiók a képen, amelyek nagy intenzitás változással rendelkeznek. Az alábbi példa azt szemlélteti, hogy mint tulajdonság a képen milyen egyszerű módon detektálható.



A kék, sárga, és bordó színnel jelzett régiókat vizsgáljuk meg. Ha a régiókat a zöld négyszög megfelelő helyeire kellene elhelyeznünk mi alapján tudnánk ezt megtenni? A kék téglalap a képen minden irányba mozgatva ugyanazt a képet kapjuk, tehát elmond-

ható, hogy semmi fajta hely információ nem nyerhető ki, ha annak környezetében vizsgálódunk. A sárga négyszög esetében elmondható, hogy létezik egy olyan orientáció, amely mellett nem kapunk többlet információt a régió elhelyezkedéséről. Ez akkor következik be, ha a régiót párhuzamosan mozgatjuk arra az élre, amin elhelyezkedik a sárga téglalap. A bordó négyszög esetében azonban bármilyen irányba mozgatva azt, változik a régióban látott képünk így könnyen beazonosítható, hogy egy sarokpontról van szó.

A Harris sarokdetektáló algoritmus a képen lévő nagy intenzitás különbségeket detektálja. Ez látható az alábbi egyenleten is, amely egy pont (u, v) környezetében minden irányban vizsgálja az intenzitás különbségeket [19]:

$$E(u,v) = \sum_{x,y} w(x,y)\,[I(x+u, y+v) - I(x,y)]^2 \tag{17}$$

ahol

$w(x,y)$, az ablakozó függvény, amely lényegében egy maszkként feleltethető meg

$I(x+u, y+v)$, eltolt ablak intenzitása

$I(x,y)$, $az\ (x,y)$ pontban elhelyezkedő ablak intenzitása.

Az ablakozó függvény lehet négyszögletes vagy Gauss. Gauss esetében a pixelekhez súlyokat rendelünk. A cél az $E(u,v)$ pontosabban a (17) kifejezés $\sum_{x,y}[I(x+u, y+v) - I(x,y)]$] részének maximalizálása. Ezen a ponton érhető el a legnagyobb intenzitáskülönbség. A (17) egyenlet átalakítása és néhány matematikai művelet [20] végrehajtása után juthatunk el a következő formához [19]:

$$E(u,v) = [\,u\ v\,]\ M \begin{bmatrix} u \\ v \end{bmatrix} \tag{18}$$

ahol

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix} \tag{19}$$

I_x és I_y a kép *x* és *y* iránymenti deriváltjai. A sarokpontok meghatározásához az M sajátértékeinek kiszámítása szükséges. Chris Harris és Mike Stephens megalkották azt az

egyenletet, amely a sajátértékek felhasználásával képes eldönteni, hogy egy adott régióban sarokpontról van-e szó vagy sem. Az egyenlet pedig az alábbi (20) módon néz ki [19]:

$$R = \det(M) - k\big(trace(M)\big)^2 \tag{20}$$

$$\det(M) = \lambda_1\lambda_2 \tag{21}$$

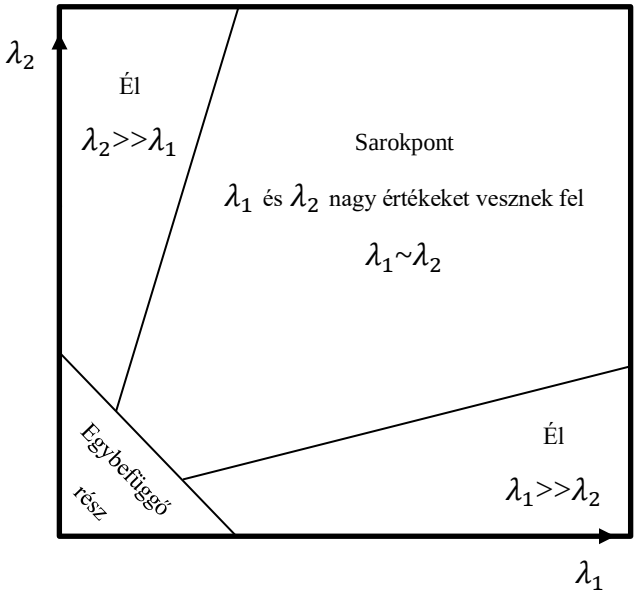
$$trace(M) = \lambda_1 + \lambda_2 \tag{22}$$

ahol

λ_1 és λ_2, az M sajátértékei

k, egy szabadparaméter

A sajátértékek segítségével tehát megmondható, hogy egy régión belül élt, egybefüggő részt, vagy sarokpontot találtunk. A 27. ábra szemléletesen bemutatja az egyes eshetőségeket.



27. ábra – Sajátértékek alapján a detektált régiótípus [19]

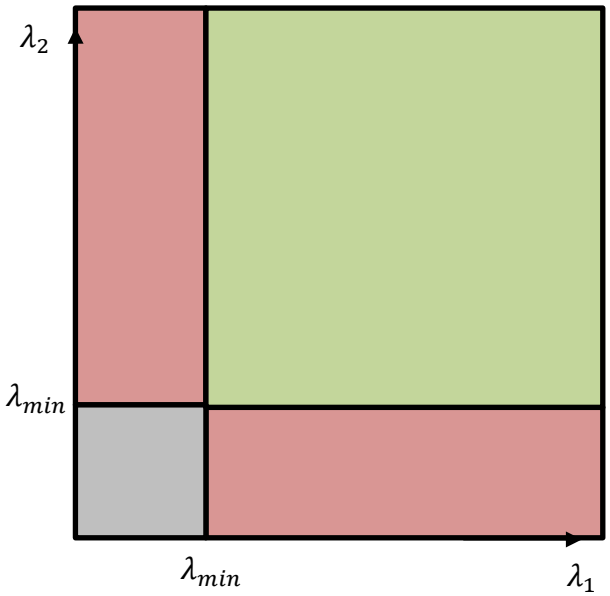
Ahogy az a 27. ábra is mutatja három helyzetet különböztetünk meg [19]:

- Ha $|R|$ kis értéket vesz fel, ami akkor következik be, ha λ_1 és λ_2 is egyaránt kicsi, ekkor az adott része a képnek egybefüggő.
- Ha $R < 0$, ami akkor történik meg ha $\lambda_2 \gg \lambda_1$ vagy fordítva, akkor az adott képrészen él található.

- Ha *R* értéke nagy, ami akkor következik be, ha λ_1 és λ_2 is nagy és $\lambda_1 \sim \lambda_2$, akkor az adott régióban sarokpont található.

Az alábbi koncepciót egy minimálisan dolgozta át J. Shi és C. Tomasi [22]. A modifikáció az *R* kifejezésben történt meg. Bevezetésre került egy λ_{min} küszöbérték, amelyet meghaladva a régió sarokpontnak tekinthető. Az *R* kifejezést az alábbi (23) egyenlet írja le, míg a 28. ábra pedig szemlélteti azt.

$$R = \min(\lambda_1, \lambda_2) \tag{23}$$

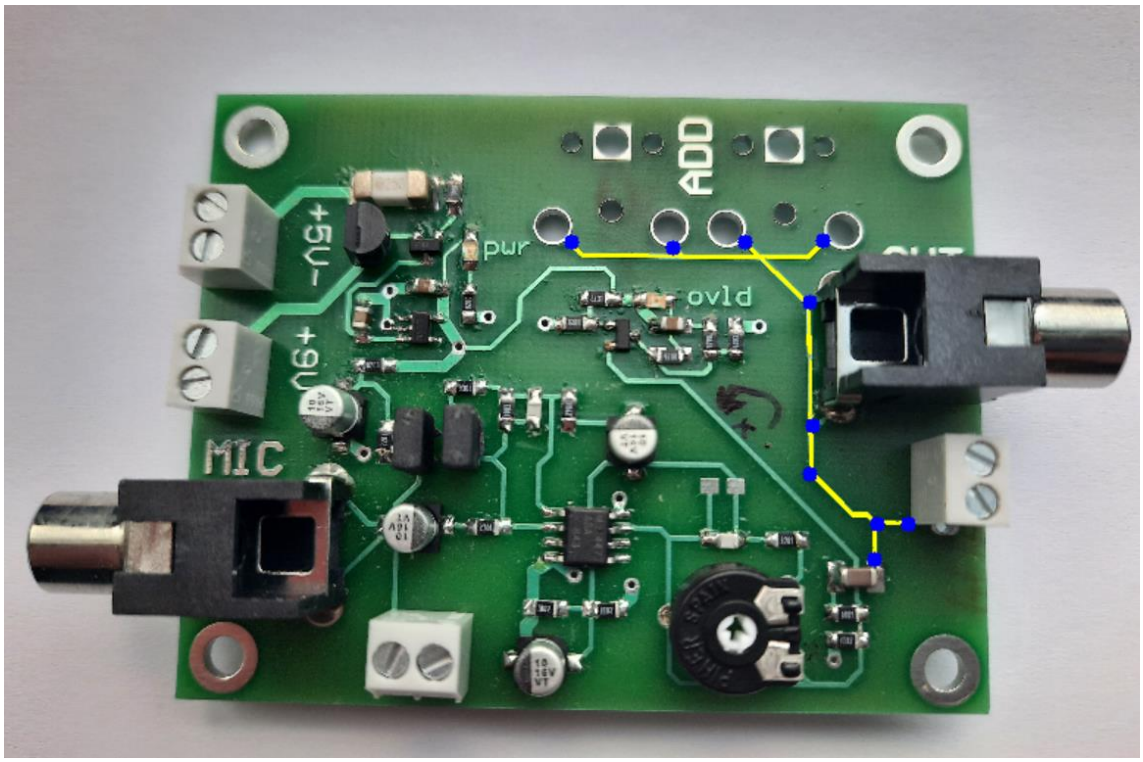


28. ábra – Good Feature to Track régiófelismerés sajátértékek alapján [21]

Az algoritmus egyszerűen használható a beépített OpenCV függvény segítségével. A függvény az alábbi néven cv.goodFeaturesToTrack érhető el [21]. Paraméterként az alábbiakat várja:

- *N* számot, amely a keresendő sarokpontokkal egyezik meg.
- λ_{min} értékét, amelynek 0 és 1 között kell lennie.
- Minimum euklideszi távolságot az egyes sarkok között.

A folyamat során az *N* legerősebb sarokpont kerül megtartásra. Azok a pontok, amelyek az euklideszi távolságon belül vannak eldobásra kerülnek. A folyamat eredménye az alábbi 29. ábra szemlélteti. A sárga szín a kijelölt vonalat jelképezi, míg a kék pontok az algoritmus által javasolt sarok pontok a definiált kritériumok mellett.



29. ábra – Good Feature to Track alkalmazásának eredménye

3.3.4. Hough transzformáció

A komponensek egy részének detektálása érdekében Hough transzformáció került felhasználásra, ezen belül is a kör Hough transzformáció. Ennek segítségével olyan objektumok ismerhetők fel, amelyek kör alakúak.

Kétdimenzióban a kör az alábbi egyenlettel írható le:

$$(x - a)^2 + (y - b)^2 = r^2 \qquad (24)$$

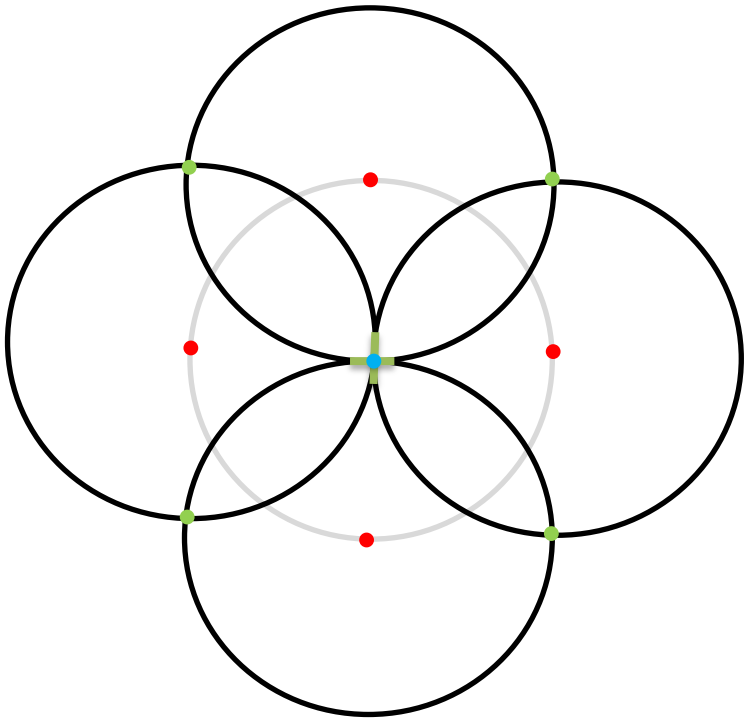
ahol

a, b a kör középpontjának koordinátái

x, y pedig a kört alkotó pontok koordinátái

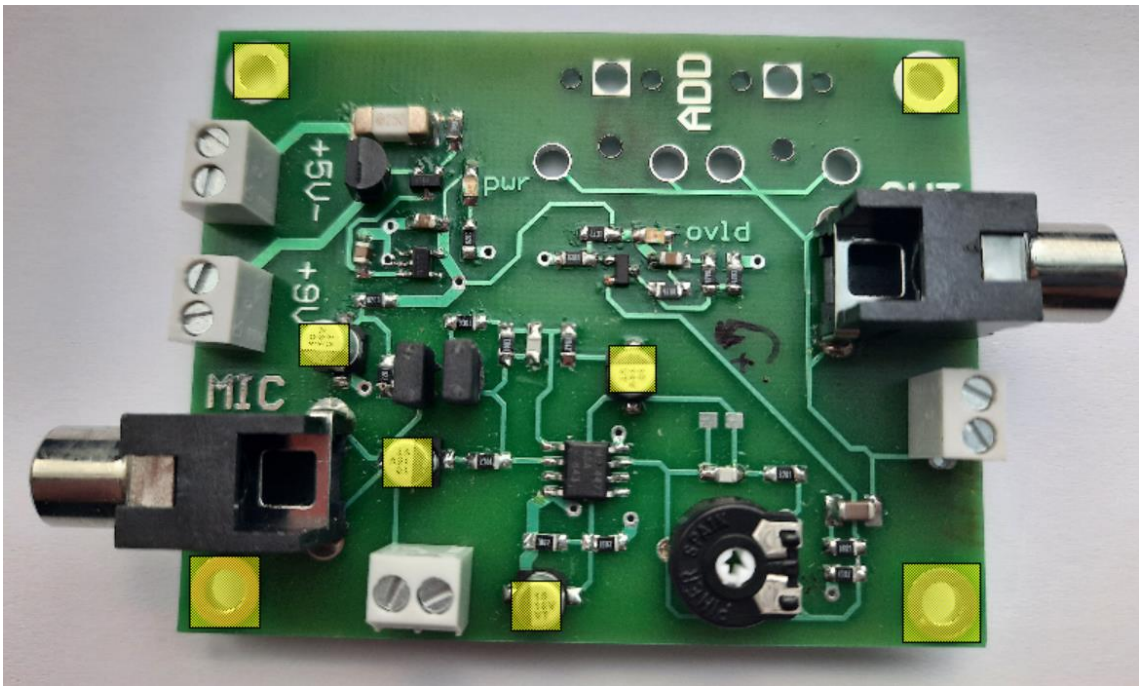
x és *y* fixálása esetén meghatározhatóak azon paraméterek, amelyek kielégítik a (24) egyenletet. Ez esetben a paramétertér háromdimenziós lenne (a, b, r). Az összes olyan paraméter, amely *x* és *y*-t kielégíti, egy derékszögű kúp felületén fekszik, amelynek csúcsa nem más mint (x, y, 0) [23]. A Hough transzformáció során kezdetben az *R* sugarat rögzítjük és a paramétertérben az *a* és *b* értékének megtalálása a cél. Az alábbi 30. ábra esetében a szürkével jelzett kör középpontja keresendő. Pirossal jelölt pontokból egy-egy

R sugarú kört generálunk. A metszéspontok követésére akkumulátormátrixot használ az algoritmus. Az akkumulátor mátrix minden egyes eleme egy rácsponthoz tartozik, amelynek értékét a rácsponton áthaladó körök száma határozza meg. Ezt a számot „szavazási számnak" hívják [23]. A mátrix minden elemét kezdetben nullával inicializáljuk, majd egy éldetektáló algoritmus (pl: Canny) által detektált éleken végig haladva annak minden egyes pontján definiálunk egy kört. Ekkor növelhetjük azon rácspontok értékét, amelyen áthalad az adott kör. Ezt nevezzük „szavazásnak" [23]. A folyamat befejeztével a mátrixon belül maximumokat keresünk. Ezeken a pontokon feltételezhető a keresendő kör középpontja a képen.



30. ábra – Kör Hough transzformáció működési elve

Lehetőség van továbbá különböző *R* sugarak mentén keresni. Ezesetben az akkumulátor mátrix háromdimenziós lesz. A fentebb ismertetett folyamatot minden *R* esetében el kell végezni, illetve a maximumokat meg kell határozni. Az algoritmus használatához létezik beépített OpenCV függvény: cv.HoughCircles [24], amelynek paraméterül megadható a keresendő *R* sugár tartománya. A 31. ábra a cv.HoughCircles függvény alkalmazásának eredményét mutatja.

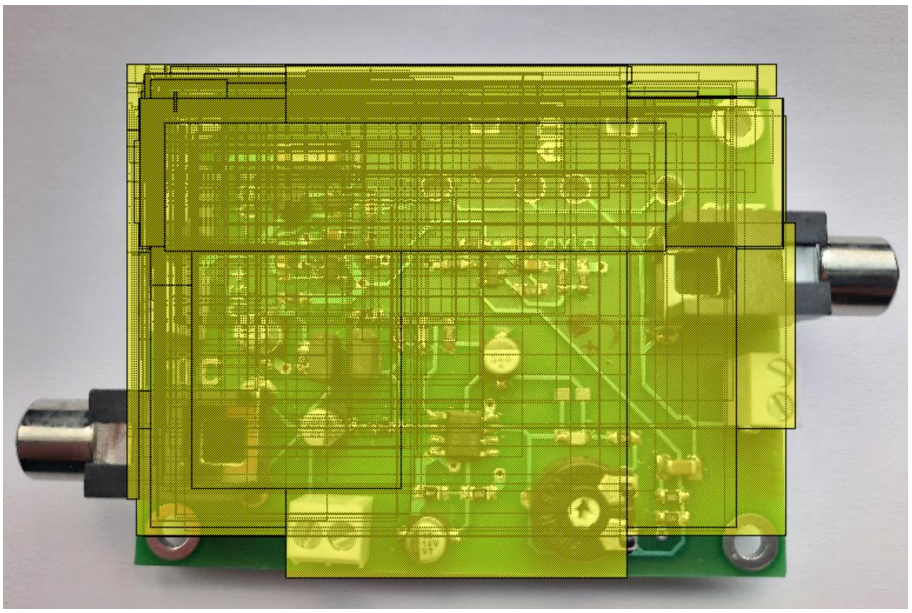


31. ábra – Kör kereső Hough algoritmus alkalmazásának eredménye

3.3.5. Selective Search és Non-maximum suppression alkalmazása

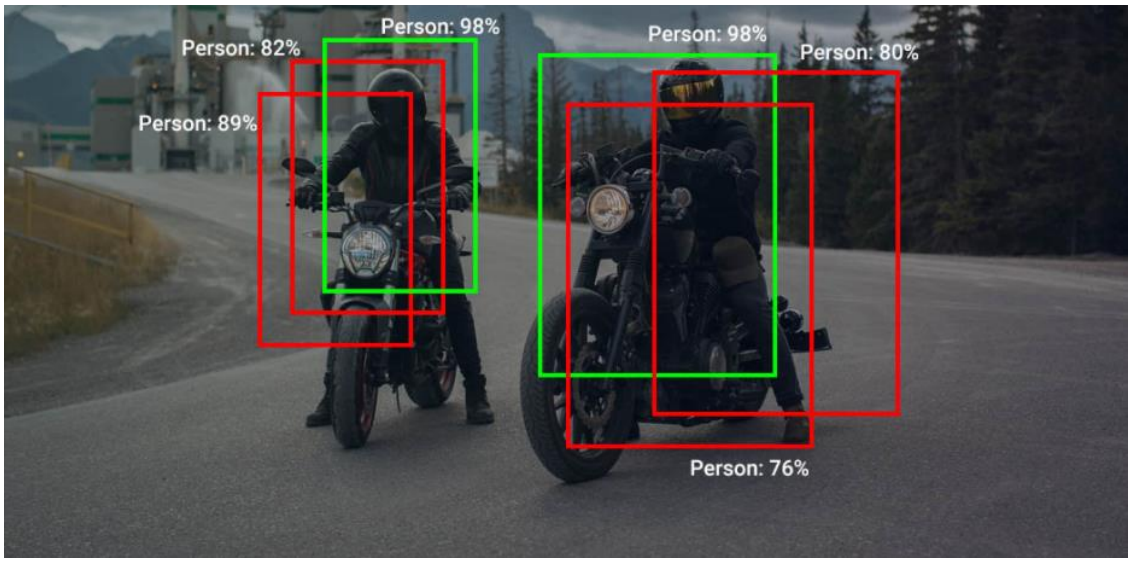
A NYÁK több négyszögletű komponenst tartalmaz, mint kör alakút, így célszerű egy olyan megoldást találni, amely képes ezen alkatrészek közül is minél többet felismerni. Az egyik legáltalánosabb megoldás objektumok alakjának detektálására az úgynevezett Ramer-Douglas-Peucker algoritmus [25]. Az algoritmus előre meghatározott kontúrokkal dolgozik, és a fő célja az objektumot körül határoló kontúr méretének csökkentése. Az algoritmus végeredményeként látható, hogy hány darab egyenes kellett az objektum körülhatárolására, így meghatározható annak alakja is, például 3 egyenes esetében háromszögről van szó, 4 esetében négyzet vagy téglalapról. A probléma ezzel a megoldással, hogy a feldolgozandó képen az árnyékok, a zaj mind hibás kontúrokat generálnak, ami miatt több objektumot detektál az algoritmus. A probléma az éldetektálás szempontjából is előjött, amelyről részletesebb információk a 3.3.1. fejezetben olvashatóak.

Legeredményesebb megoldásnak a Selective Search és a Non-maximum suppression alkalmazása bizonyult. A Selective search algoritmus működési elve a 3.2.1. fejezetben, míg a Non-maximum suppression algoritmus a 3.2.2. fejezetben került ismertetésre. A Selective Search segítségével különböző szempontok alapján olyan régiókat állapíthatunk meg, amelyen feltételezhetően valamilyen objektum található. A képen számos különböző méretű négyszög kerül legenerálásra, amely ezeket a javaslatokat reprezentálják. A selective search algoritmus kimenetét a 32. ábra mutatja.



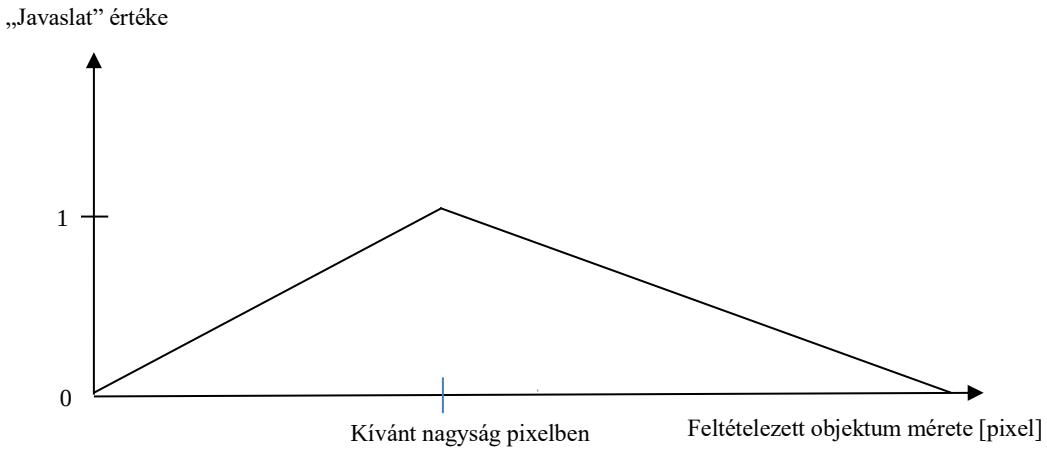
32. ábra – Selective Search által javasolt objektumok bekeretezve

Általában a selective search és a non-maximum suppression algoritmus közé valamilyen tanított hálózat kerül elhelyezésre, amely az egyes négyszögeket képes úgynevezett „javaslatokkal” ellátni. Ez az érték 0 és 1 közötti szám és annál nagyobb minél nagyobb mértékben tartalmaz a kép egy helyesen felismert objektumot. A non-maximum suppression segítségével pedig megtarthatóak azok a téglalapok, amelyek a legjobb mértékben fedik le a keresett objektumokat. A kiválasztás folyamatának egy példáját a 33. ábra szemlélteti.



33. ábra – Non-maximum suppression alkalmazásának hatása [26]

Betanított neurális háló hiányában egy olyan megoldás megtalálása volt a cél, amely valamilyen mértékben képes az objektumok beazonosítására és ezek alapján „javaslatokkal” ellátni az egyes téglalapokat. Az egyik legszembetűnőbb tulajdonság az alkatrészek méretei. Általában egy adott típusú komponensből egy NYÁK lapon több azonos is beültetésre kerül. Azon objektumok között, amelyek méretei egy bizonyos érték között mozognak nagyobb javaslati értéket kapnak, ellenben azokkal, amelyek nagyobb mértékben térnek el a kívánt értéktől. Ehhez definiálásra került két egyenes egyenlet. A találkozási pontjukban a kívánt területnagyság érhető el. Ez esetben a javaslati érték 1. Ettől a ponttól távolodva azonban mind a két irányban monoton csökkenő tendencia tapasztalható, amelynek alsó korlátja a 0. Az alábbi 34. ábra a fentebb említett kritériumokat mutatja be szemléletesen. Az objektum mérete kifejezés alatt annak a téglalapnak a méretét kell érteni, amelyet a selective search legenerált.

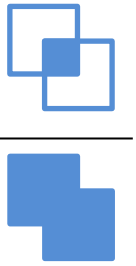


34. ábra – Objektum méretének és a „Javaslat” értékének kapcsolata

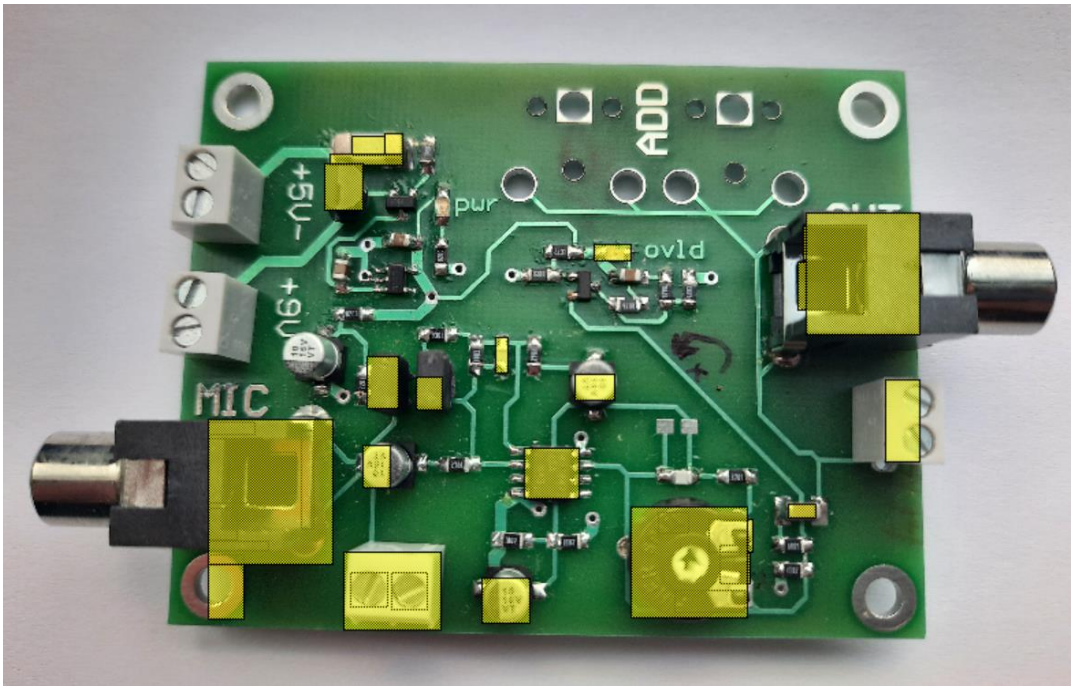
Csak a méret alapján azonban több fals javaslat készült, mint helyes, így további tulajdonságot kellett figyelembe venni. Ez a plusz tulajdonság a szín, hiszen a PCB nagyrészén a zöld különböző árnyalatai jelennek meg. További feltételként tehát a kijelölt téglalap méretei mellett figyelemmel kell kísérni, hogy milyen arányban tartalmaz zöldes, illetve más színű pixeleket. A HSV alkalmazásának köszönhetően, amelyről korábban a 3.3.2. fejezetben esett szó, lehetőség nyílik egy olyan körcikk definiálására a HSV cilinderen, amely a zöld minden árnyalatát magában foglalja. Abban az esetben, ha a négyszögre kiszámított terület, illetve szín arányok „javaslati” értékei meghaladnak egy bizonyos küszöbértéket akkor elmentésre, a feltételt nem teljesítő téglalapok pedig eldobásra

kerülnek. Ezzel a módszerrel szűkíthető a rossz javaslatok száma. Ezt követően mindegyik négyszög rendelkezik egy hozzárendelt értékkel, amely alapján a Non-maximum suppression az egymással átlapolásban lévők közül a legnagyobb értékűt kiválasztja, majd az IoU (Intersection over Union) alapján meghatározható, hogy az átlapolt másik négyszög része-e a detektált objektumnak, vagy az már egy másik komponenshez tartozik. Az IoU meghatározását az alábbi egyenlet definiálja:

$$IoU = \frac{Területek\ metszete}{Területek\ uniója} \qquad (25)$$



A Non-maximum suppression használata során paraméterként megadható egy előre definiált érték, amelynek 0 és 1 között kell lennie. Minden olyan négyszög, amelynek az IoU-ja kisebb, mint a definiált érték, külön négyszögként megtartjuk, míg ellenkező esetben eldobjuk azt. Ahogy azt a 35. ábra is mutatja, már ez a két tulajdonság is elegendő volt ahhoz, hogy az objektumok egy részét a szoftver képes legyen felismerni, további tulajdonságok hozzáadásával a felismerés hatásfoka még jobban növelhető.



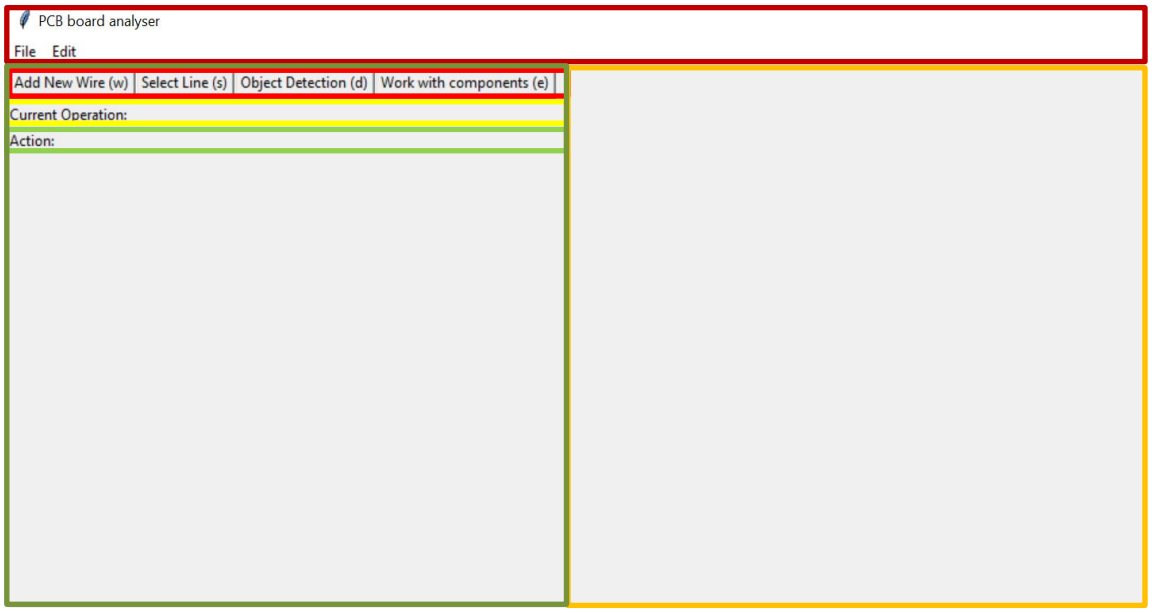
35. ábra – Selective Search és Non-maximum suppression alkalmazásának eredménye

4. Felhasználói interfész

Egy rendszer jobb felhasználhatósága érdekében sokszor előnyösebb, ha rendelkezik valamilyen felhasználói interfésszel, amely a vizuális megjelenésnek köszönhetően sokszor jobban értelmezhető. Az alkalmazás logikai részének megvalósítása után, egy olyan felületet kialakítása volt a cél, amely egyszerűen átlátható, kezelése nem túl komplex a felhasználó számára. A python lehetőséget nyújt felhasználói interfész megalkotására is a Tkinter segítségével. Az alábbi fejezetben bemutatásra kerül az interfész felépítése és a rendelkezésre álló funkcionalitások.

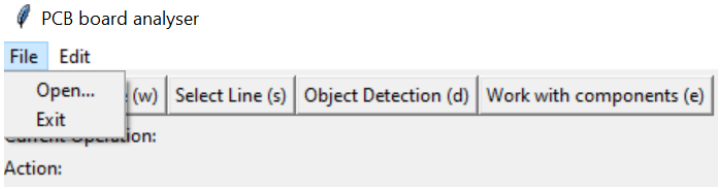
4.1. A felhasználói interfész felépítése

Az alkalmazás indulásakor egy fő ablak jelenik meg a felhasználó számára, amelynek mérete hozzáigazodik a megjelenítendő eszköz felbontásához. Az alábbi ábrán ez a megjelenő fő ablak látható, a színekkel pedig az egyes részek kerültek kiemelésre. A bordóval jelölt részen található a címsor, illetve a menüsor, amelyet lenyitva további funkciók érhetők el. Sötétzölddel és narancssárgával két keret került kiemelésre. Ezek tartalmazzák az egyes címkéket, amelyek piros, sárga és zöld színnel láthatóak. A piros színű címkében azok a gombok kerültek elhelyezésre, amelyek viszonylag gyakran fog használni a felhasználó. A sárga színnel jelölt címkében az aktuális műveletet láthatjuk, míg az alatta lévő Action elnevezésűben utalást kapunk arra milyen cselekedeteket végezhetünk el. A narancssárga kereten belül található majd a feldolgozás céljából betöltött kép.

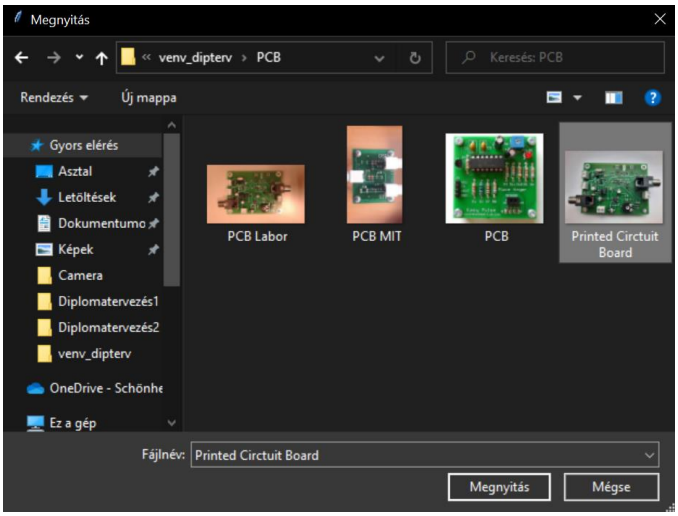


36. ábra – Felhasználói interfész felépítése

A menüsoron végig haladva a legelső elérhető opció a felhasználó számára a *'File'* névvel jelölt lenyíló menü. Ezen a helyen képes a kiválasztott kép betöltésére az *'Open'*-re kattintva. Ekkor egy dialógus ablak jelenik meg, amely segítségével kiválaszthatja azt a képet, amelynek tartalmát betölteni szeretné. A megjelenő dialógus ablakot a 38. ábra szemlélteti. A '*File'* menüponton belül az *'Exit'*-re kattintva az ablak bezárására van lehetőség. A '*File'* menü tartalmát az alábbi 37. ábra mutatja.



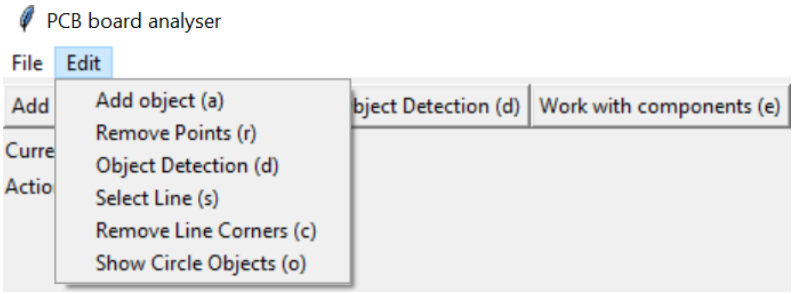
37. ábra – '*File'* menüpont



38. ábra – Felugró dialógus ablak fájl megnyitására

39. ábra – Fájl megnyitása utáni megjelenés

Ahogy az a fenti 39. ábra is mutatja a dialógus ablakban a kép kiválasztása után az a jobb oldalt elhelyezkedő keretbe kerül. Az *'Edit'*-re kattintva azok a funkcionalitások láthatók, amelyet a betöltött képen végezhetünk el. Ezeket a lehetőségeket a 40. ábra szemlélteti. Az egyes műveletek nevei mellett zárójelben egy karakter található. A szoftvert állapotát nem csak kattintással, hanem bizonyos billentyűgombok lenyomásával is vezérelhetjük. A következő fejezetekben ezen műveletek használata kerül részletesen bemutatásra.



40. ábra – „Edit" menüpont

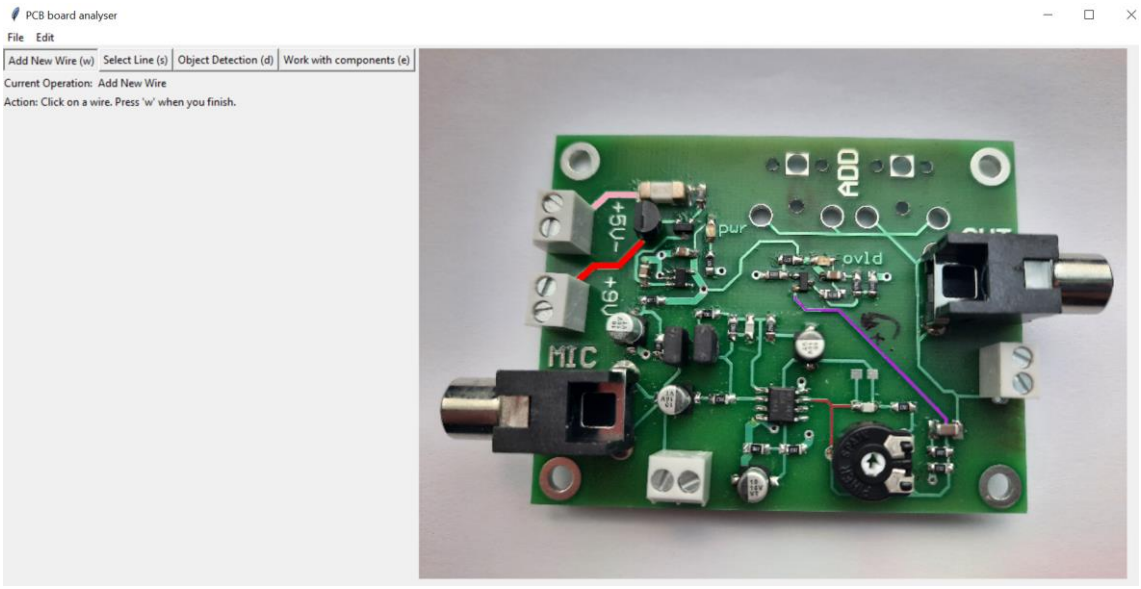
4.2. Vezetékekkel kapcsolatos interakciók

A vezetékek a NYÁK felszínének nagy hányadát lefedik, emiatt fontos, hogy a lehető leggördülékenyebb módon tudja elvégezni a felismerést a felhasználó, a lehető legkevesebb utómunkával.

4.2.1. Vezeték felvétele az adatbázisba

A vezetékek felismerése az úgynevezett régió-növesztés elve alapján működik. Erről a 3.3.2. fejezetben található bővebb információ. A kezdeti magpont kialakításához szükség van a felhasználó beavatkozására. Ezesetben egy a képen látható vezetékre kattint, miután beállította az *'Add New Wire'* állapotot. Az állapotok közötti váltáshoz egyszerűen csak a kívánt operációhoz tartozó gombra kell kattintani, vagy a hozzárendelt zárójelben szereplő billentyűzetgombot lenyomni. Az *'Add New Wire'* gyors gombként került elhelyezésre, hogy a felhasználó gördülékenyebben használhassa. A lenyomásakor az nyomva marad ezzel jelezve, hogy a kijelölt részek egy összefüggő vezetékhez tartoznak. Ennek megszüntetése a gomb ismételt megnyomásával lehetséges. A funkció használata során minden vezetékszakaszhoz egy szín rendelődik. Az azonos színű vezetékszakaszok egy csoportba tartoznak. A gomb lenyomása után látható a felhasználói interfészen, hogy a *'Current Operation'* (Aktuális művelet) megváltozik és az megegyező lesz a lenyomott gomb nevével. Ez egy plusz visszacsatolás a felhasználó számára, hogy láthassa milyen

állapotban van jelenleg a rendszer. Az *'Action'* (Akció) rész mellett feltüntetett szöveg segít tájékozódni mit is lehet tenni a művelet során. Az alábbi 41. ábra jól szemlélteti, hogy az *'Add New Wire'* állapotban van a rendszer és ahogy az az *'Action'* mező mellett látható vezetékekre kattinthatunk. A jobb oldalt pedig az látható, hogy négy vezeték került hozzáadásra az adatbázishoz, amelyeket a rendszer külön színnel jelöl.



41. ábra – Vezetékek regisztrálása

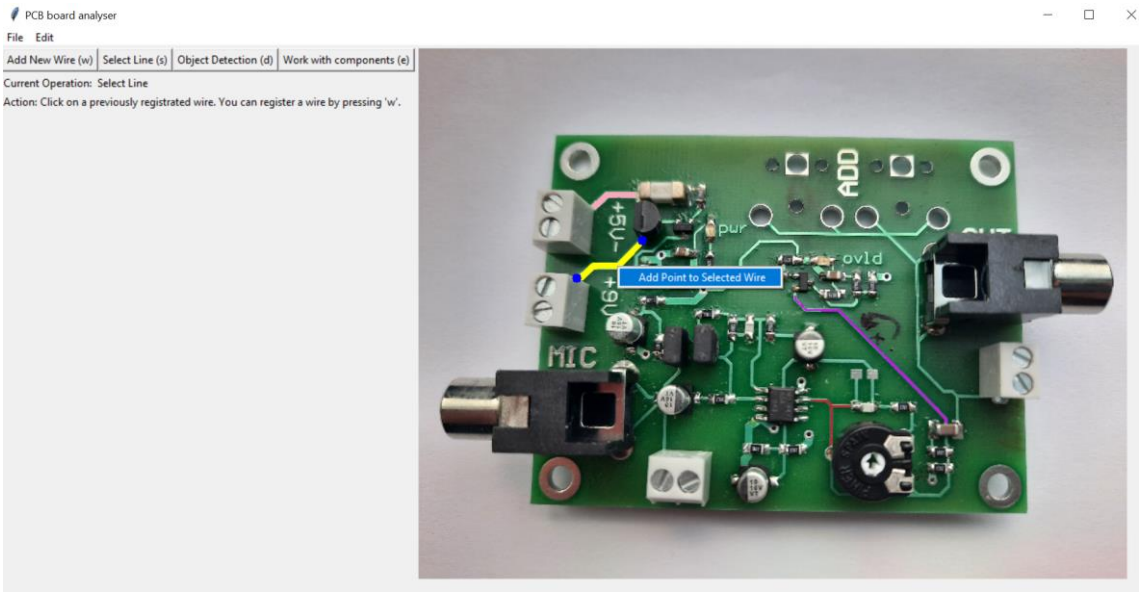
4.2.2. Vezeték kiválasztása

Olyan vezeték kiválasztására van lehetőség, amely már korábban beregisztrálásra került. Kiválasztás során egyszerre csak egy vezetékre kattinthat a felhasználó. Kattintás után a kiválasztott huzal színe sárgára változik és megjelennek a végpontokat detektáló algoritmus által meghatározott pontok sötétkék színben. A végpontokat előállító algoritmusról az alábbi 3.3.3. fejezetben részletesebb információk is megtalálhatók. A vezeték kiválasztása előtt az *'s'* vagy a '*Select Line*' megnyomása szükséges. A 42. ábra egy vezeték kiválasztás utáni állapotot mutat.

42. ábra – Vezeték választásának eredménye

4.2.3. Vezetékhez tartozó végpontok változtatása

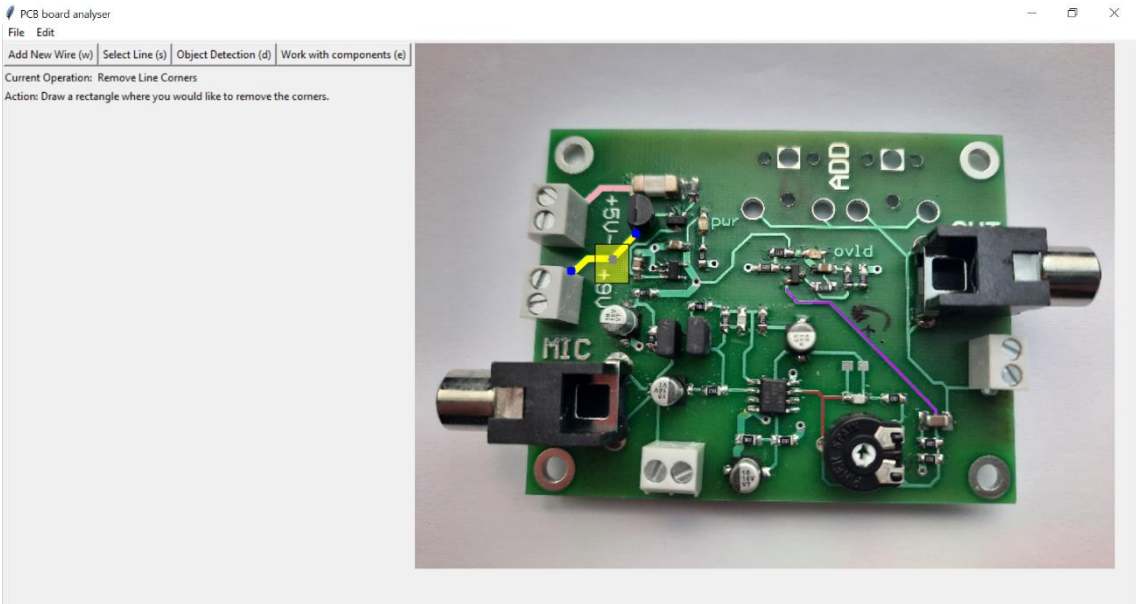
A végpontok változtatásához *'Select Line'* módban kell lennie a rendszernek. Ez az *'s'* vagy a *'Select Line (s)'* gomb lenyomásával lehetséges. Ekkor lehetőség van kiválasztani egy vezetéket és ezek után módosítani a hozzá tartozó végpontokat. Újabb pont hozzáadása során a kurzort a vezeték azon pontjára kell mozgatni, ahová az új pontot szeretné a felhasználó felvenni, majd jobb kattintással egy kontext menü jelenik meg, ahol egy kiválasztható opció szerepel. Erre kattintva az aktuális pontba egy végpont kerül elhelyezésre. A folyamatot a 43. ábra, míg annak eredményét a 44. ábra mutatja.



43. ábra – Végpont hozzárendelése egy kiválasztott vezetékhez

44. ábra – Új végpont hozzáadásának eredménye

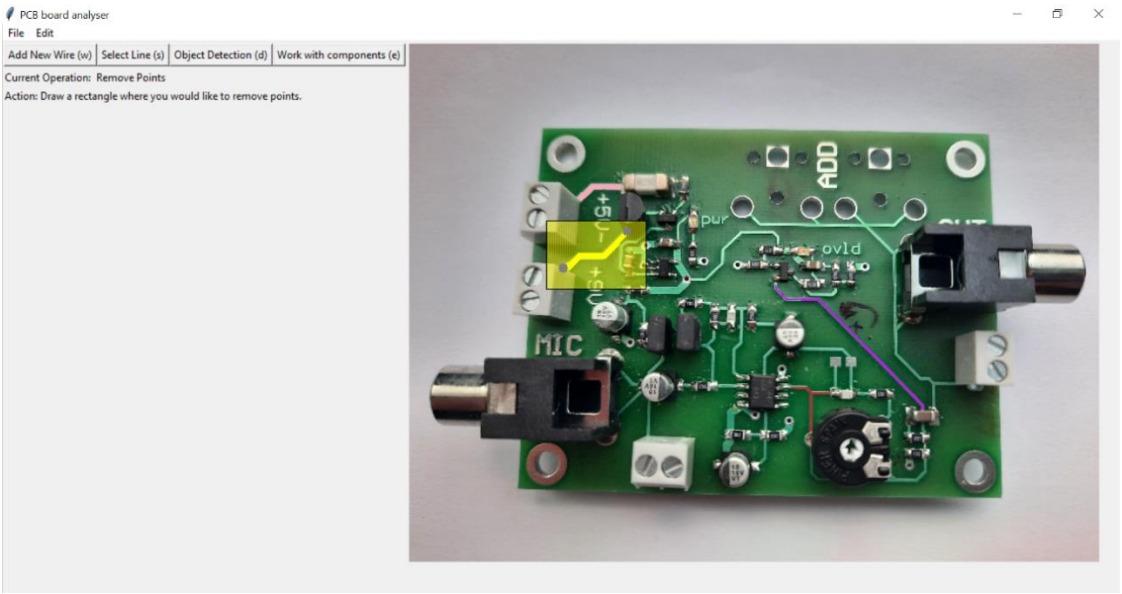
Lehetőség van továbbá egy kiválasztott vezeték sarokpontjait eltűntetni. Ehhez vagy a *'c'* karakter lenyomása, vagy az *'Edit'* legördülő menün belül a *'Remove Line Corners'* kiválasztása szükséges. A megváltozott állapot megfigyelhető a *'Current Operation'* melletti részen. Annak érdekében, hogy a felhasználó ne többszörös kattintással próbálja megtalálni azt a pixelt, amelyen pontosan a sarokpont elhelyezkedik, lehetőség nyílik egy tetszőleges méretű négyszöget rajzolni, amely minden belül elhelyezkedő pontot eltávolít a kiválasztott vezetéken. A négyszög rajzolásához elegendő a bal klikk folyamatos nyomva tartása, majd az egér mozgatása. A folyamatot a 45. ábra szemlélteti.



45. ábra – Kiválasztott vezetékhez tartozó sarokpont törlése

4.2.4. Vezetékszakasz törlése

Rosszul felismert vezetékek korrigálására is van lehetőség, ezt a műveletet mindenképp egy vezeték kiválasztás meg kell előzze. Ez segít megakadályozni a felhasználónak, hogy ne törölhessen ki más korábban regisztrált huzalozást. Az *'r'* karakter lenyomása vagy az '*Edit'* menün belül található *'Remove Points'* kiválasztásával aktiválhatja a felhasználó ezt az állapotot. Az előző 4.2.3. fejezet végén megismert módon egy négyszög rajzolásával jelölhető ki az a tartomány, amelyen belül minden korábban regisztrált információt törlünk. Ezalatt a regisztrált végpontok és vezetékek egyaránt érthetőek. A folyamatot a 46. ábra, míg az eredményt a 47. ábra szemlélteti.



46. ábra – Kijelölt vezeték törlésének folyamata

47. ábra – Kijelölt vezeték törlésének eredménye

4.3. Komponensekkel kapcsolatos interakciók

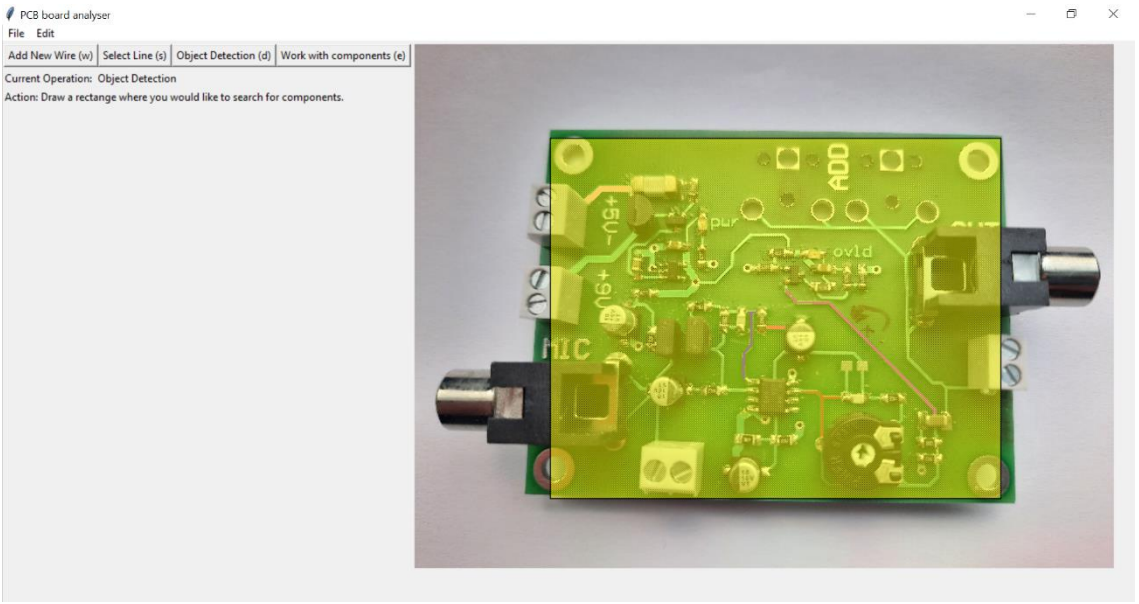
A vezetékek sikeres felismerése után a következő fontos detektálandó elem az alkatrészek. Az alkatrészek beazonosításához korábban két algoritmus került bemutatásra. A kör Hough transzformáció segítségével kör alakú objektumok detektálhatók, erről részletesebb információ a 3.3.4. fejezetben olvashatók. A selective search és non-maximum suppression módszer pedig alaktól függetlenül képes komponensek megtalálására. További részletek a 3.3.5. fejezetben találhatók.

4.3.1. Komponensek detektálása

A kör Hough transzformáció alkalmazásához egyszerűen az *'o'* karakter vagy az *'Edit'* menüponton belül a *'Show Circle Objects'* opcióra történő kattintás szükséges. Ekkor minden detektált objektumra egy áttetsző sárga négyzet kerül. Az alábbi 48. ábra jól mutatja, hogy a '*Work with components'* állapotban vagyunk. Itt tekinthetőek meg a detektált elemek és hajthatunk végre velük különböző műveleteket. Gyorsgombként is elhelyezésre került, de az *'e'* karakter lenyomásával is elérhető.

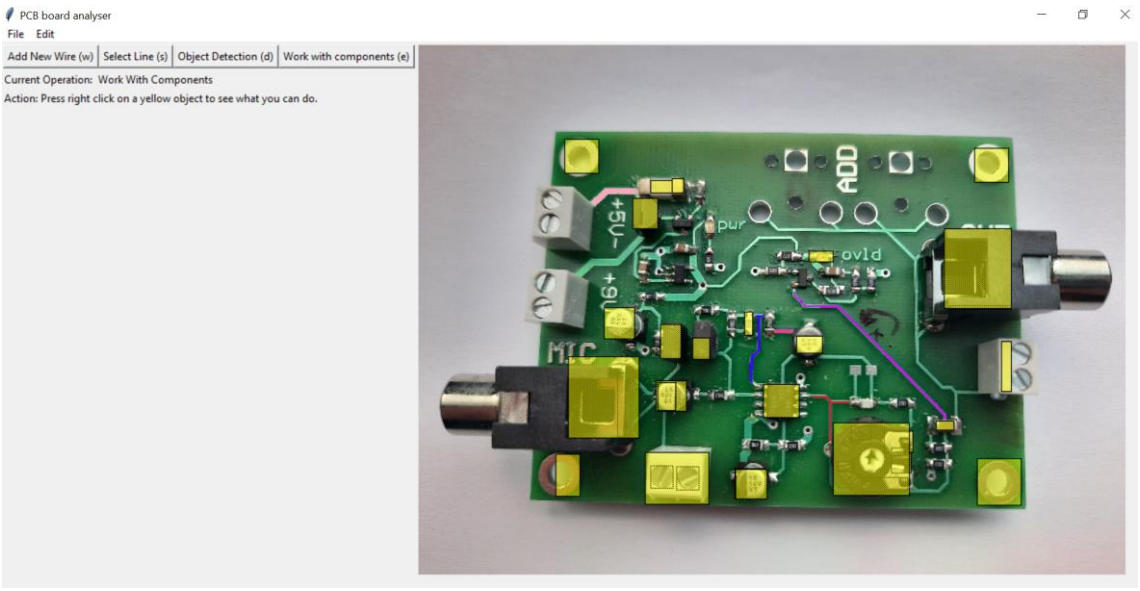
48. ábra – Kör alakú objektumokat kereső algoritmus alkalmazásának eredménye

A selective search használata előtt az *'Edit'* menüpont alatt az *'Object Detection'* kiválasztása szükséges vagy elegendő a *'d'* karakter lenyomása. Ezt követően az egér segítségével egy négyzetet kell rajzolni a képen, amelyen belül az objektumokat kívánja a felhasználó detektálni. A folyamatot jól mutatja a 49. ábra.



49. ábra – Selective search használatához szükséges terület megadása

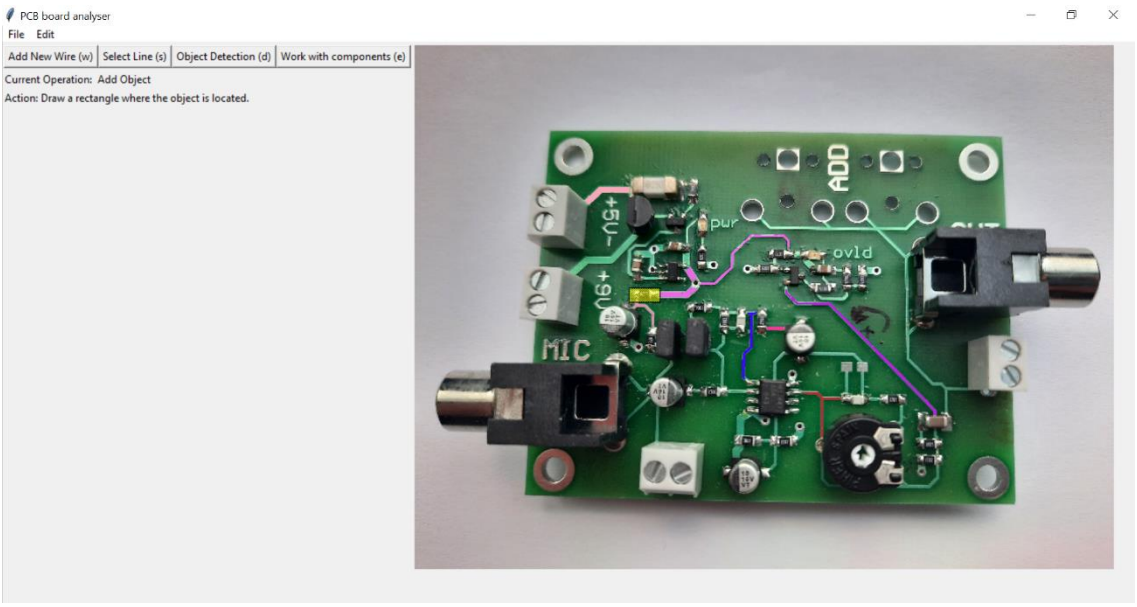
Ahogy az a kör alakú objektumok felismerésénél is történt, ez esetben is transzparens sárga négyzettel került jelölésre az algoritmus által javasolt objektumok pozíciói. Ennek eredményét a 50. ábra szemlélteti.



50. ábra – Objektum detektálás eredménye egy kijelölt régión

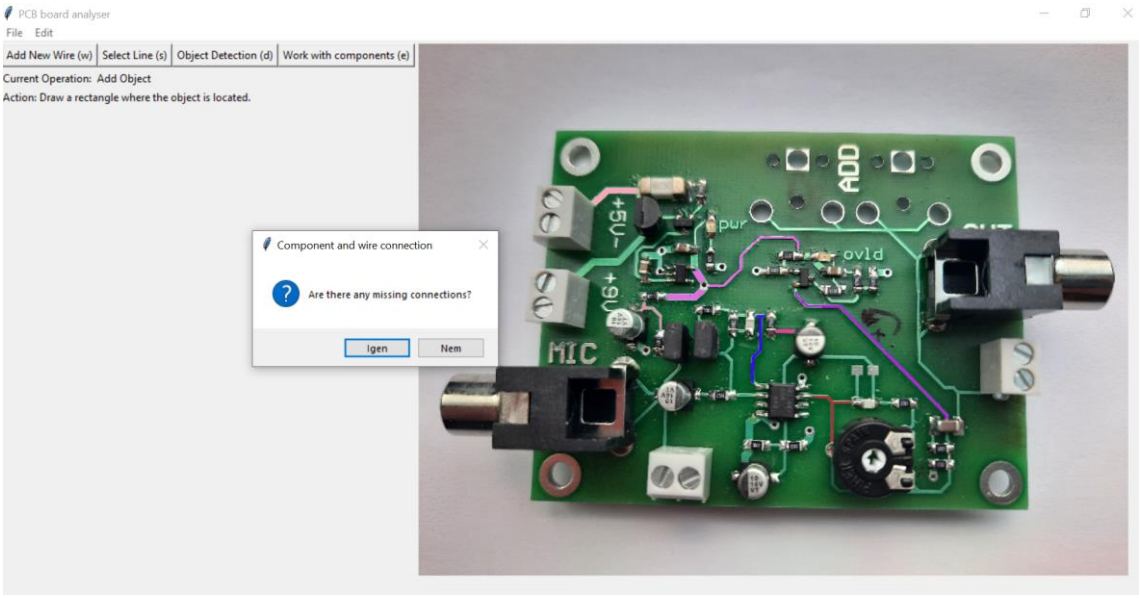
4.3.2. Komponensek kézzel történő felvétele

Az algoritmus, ahogyan azt az 50. ábra is mutatja nem minden komponenst képes felismerni. A félautomata működés ezen a ponton szintén előtérbe kerül hiszen a felhasználó részéről különböző korrekciók szükségesek, hogy a rossz javaslatokat megszűntesse és a nem detektált elemeket az adatbázisba beregisztrálja. Az új elem felvételéhez szintén egy külön állapot létezik, amelybe az *'a'* karakter vagy az *'Edit'* menüponton belül az '*Add object'* lenyomásával juthat a rendszer. A korábban megszokott módon a regisztrálni kívánt komponens köré egy téglalap rajzolása szükséges. A téglalap koordinátái ekkor elmentésre kerülnek és automatikusan megkezdődik az objektumhoz kapcsolódó végpontok detektálása. A kijelölés folyamata az alábbi ábrán látható.

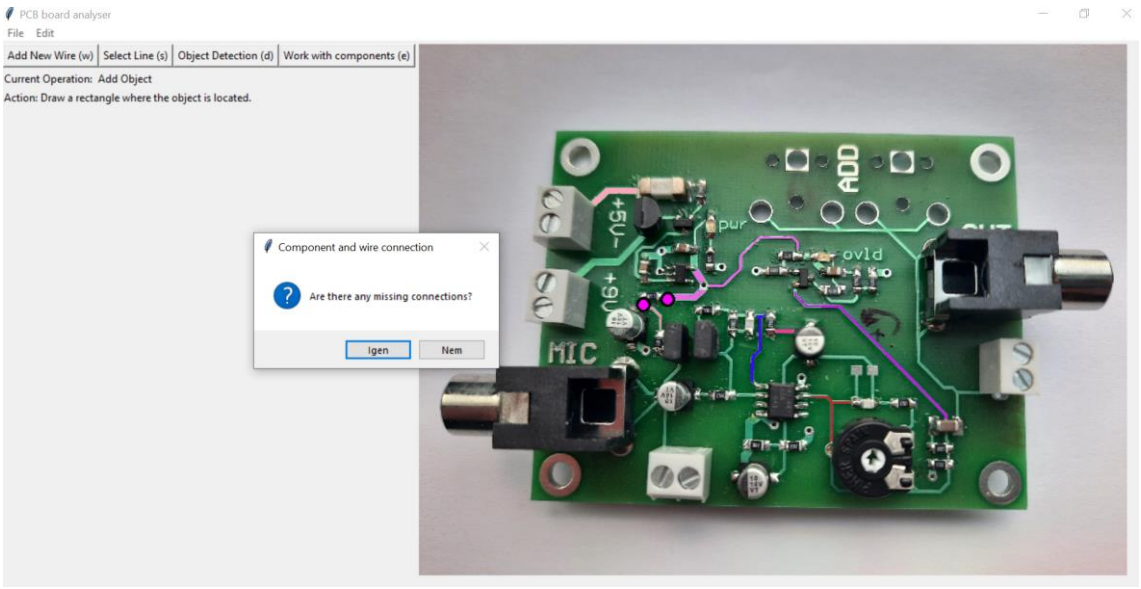


51. ábra – Objektum kézzel történő felvétele

A komponenshez tartozó végpontok megtalálása során a kezdeti négyszögön belül keres az algoritmus végpontok után. Ezt követően egy felugró üzenet jelenik meg, amelyen a felhasználót arról kérdezik, hogy lát-e olyan végpontot, amelyet még nem talált meg a rendszer. Abban az esetben, ha van ilyen akkor az igenre, ellenkező esetben a nemre kattinthat. Ez a folyamat mind addig tart, amíg az összes végpont nem kerül megtalálásra. A felismert végpontokat rózsaszínnel jelöli a rendszer. Az adatbázisba felvett objektum környezetében csak olyan vezetéken képes a rendszer végpontokat keresni, amely korábban szintén rögzítésre került már. A 52. ábra azt az állapotot mutatja, ahol még nem került felismerésre végpont, azonban megfigyelhető, hogy az objektum környezetében két vezeték felvételre került, így a felhasználó feltehetően az *„igen”* választ választja a felugró ablakon. Ekkor az algoritmus egy előre definiált epszilon mérettel növeli az objektum környezetét és megismétli a keresést. A 53. ábra szemlélteti hogyan is jelennek meg a detektált végpontok. Ebben az esetben az alkalmazás ismételten megkérdezi a felhasználót, hogy van-e további hiányzó végpont. A *„nem”* válaszra történő kattintással befejezhető a művelet és az objektum a megtalált végpontokkal kerül elmentésre.



52. ábra – Végpontok keresése beregisztrált objektumhoz



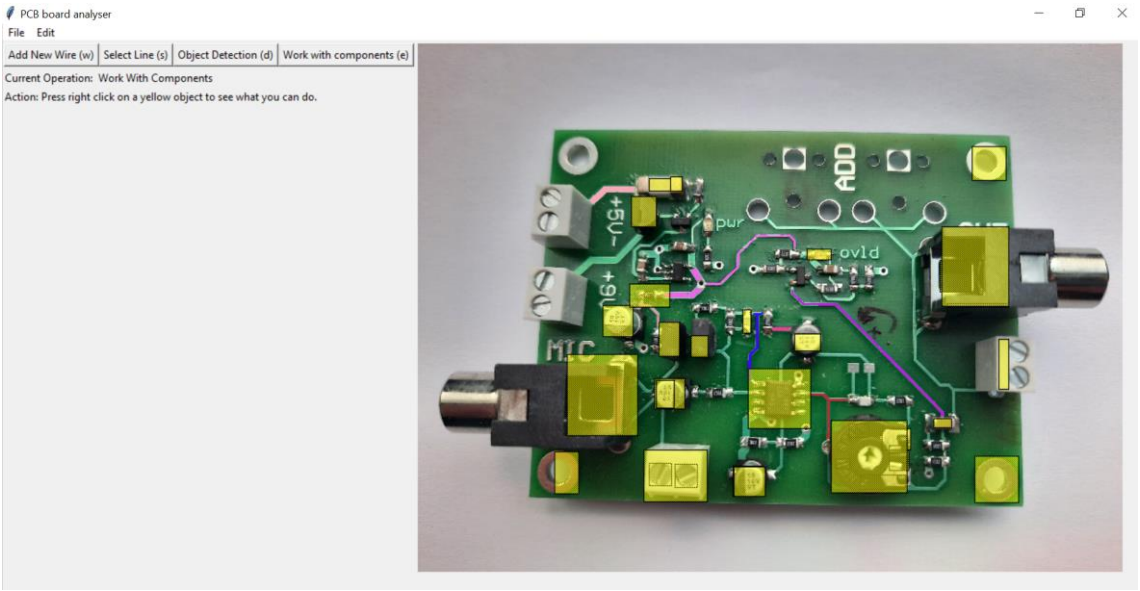
53. ábra – Talált végpontok megjelenítése

4.3.3. Komponensek törlése

Az objektum detektálást végző algoritmus nem minden esetben képes helyes döntést hozni. Bizonyos esetekben több téglalap is megjelenítésre kerül, mint ahány komponens az adott kép egy részén látható. A felhasználónak lehetősége van korrigálni ezeket a hibákat is. A művelet elvégzése előtt a *'Work with components'* gyorsgomb, vagy az *'e'* karakter lenyomása szükséges. A detektált objektumok megjelenése után a nem kívánt elemre jobb kattintással egy kontext menü jelenik meg, ahol láthatóak az elvégezhető

műveletek. Az utolsó lehetőségre kattintva, ahogy azt az 54. ábra is mutatja az elem eltávolításra kerül. Ennek eredménye szintén alább látható.

54. ábra – Detektált objektum eltávolítása



55. ábra – Detektált objektum eltávolításának eredménye

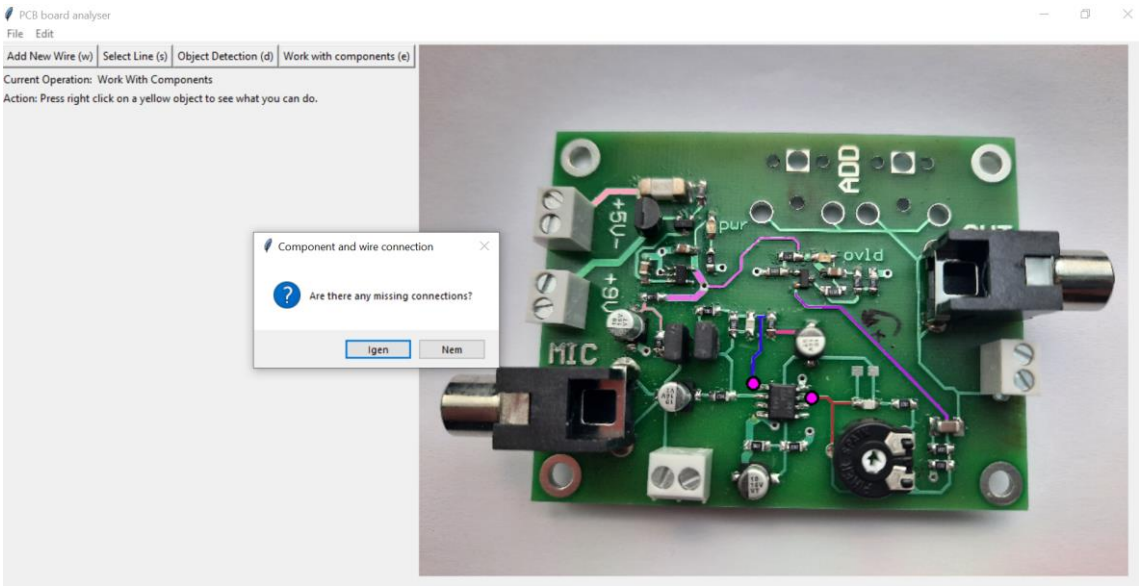
4.3.4. Komponensekhez kapcsolódó végpontok regisztrálása, megjelenítése

A *'Show Circle Objects'* vagy *'Object Detection'* használata során az egyes objektumokhoz végpontok nem rendelődnek. A felhasználó feladata eldönteni mik azok a detektált részek, amelyek helyesek és megtartandóak és miket kell eldobni. A megmaradt alkatrész halmazhoz hozzárendelhető kézzel egyesével a végpontok. Ehhez a korábban említett módszert kell alkalmazni, miszerint a rendszer *'Work with components'* állapotában

jobb kattintással egy regisztrált alkatrészen láthatóak a rajta elvégezhető műveletek. Ezt a 56. ábra mutatja. Itt az *'Add New Connection points'* segítségével, ahogy az kézzel történő komponens felvételnél is tapasztalható volt, a kijelölt területen belül az algoritmus végpontokat kezd el keresni és mind addig míg a felhasználó nem elégedett az eredménnyel, a régió növesztésével nagyobb területeken próbál vizsgálódni.

56. ábra – Regisztrált objektum végpont hozzárendelése

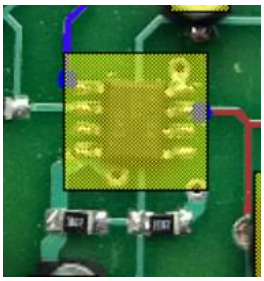
A megtalált végpontokat szintén rózsaszínnel jelzi az alkalmazás a felhasználó számára. Ez az alábbi ábrán is jól látható.



57. ábra – Megtalált végpontok jelzése a felhasználó számára

Lehetőség van továbbá ellenőrizni, hogy egy adott elemhez került-e korábban vezeték kapcsolva. Ez a funkció a *'Work with components'* állapotban érhető el. Egy regisztrált alkatrészre kattintva jobb klikkel a *'Show Connection Points'* segítségével megjeleníthetők a kapcsolati pontok. Ennek folyamata és eredménye az alábbi ábrákon figyelhetőek meg.

58. ábra – Komponenshez regisztrált végpontok megtekintése

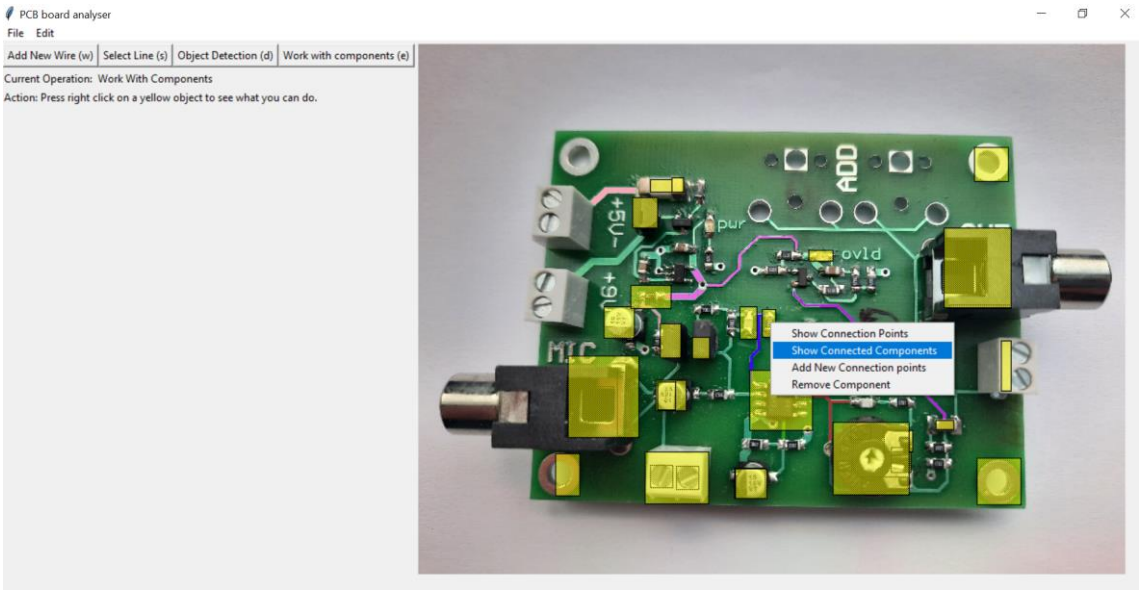


59. ábra – *'Show Connection Points'* opció kiválasztásának eredménye

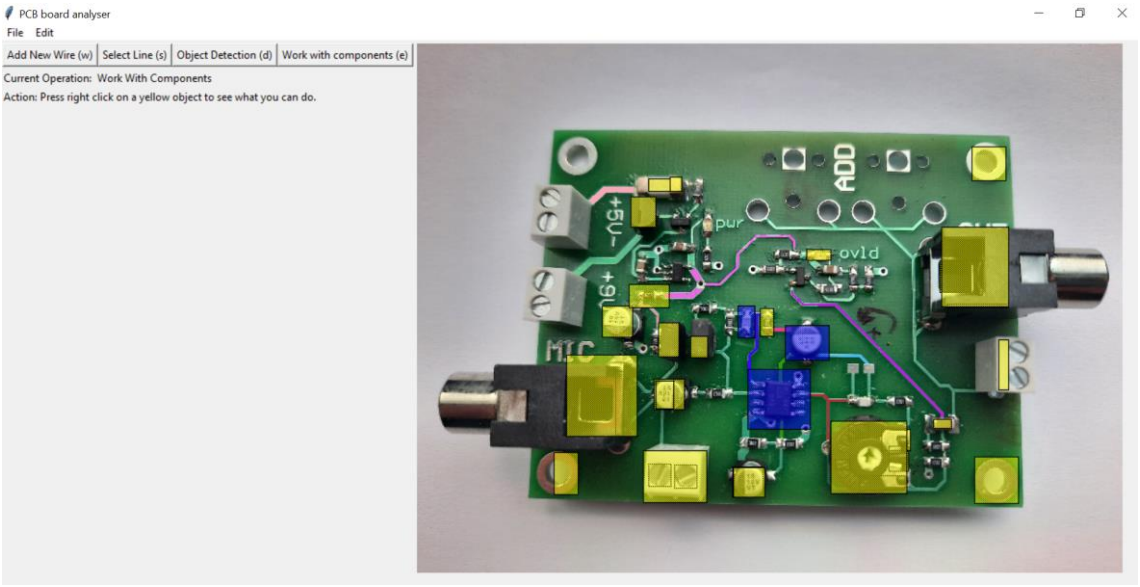
4.3.5. Komponensek közötti kapcsolatok megjelenítése

Annak érdekében, hogy a későbbiekben lehetőség legyen egy kapcsolási rajz legenerálásához szükséges tudni milyen komponens milyen más komponensekkel van kapcsolatban. Az egyes alkatrészekhez történő végpontok beregisztrálásával ez meg is történt. Egy komponenshez tartozó végpontból beazonosítható, hogy melyik vezetékhez tartozik. Ezek alapján pedig a többi végpontot vizsgálva detektálhatók a kapcsolatban álló további komponensek is. A funkció a *'Work with components'* állapotban érhető el, egy elemre

kattintva a *'Show Connected Components'* segítségével láthatók a kapcsolatban álló további elemek. Ezt az alkalmazás transzparens kék színnel jelzi, amelyet a 61. ábra is jól szemléltet.



60. ábra – Kapcsolatok megjelenítését segítő funkció kiválasztásának folyamata

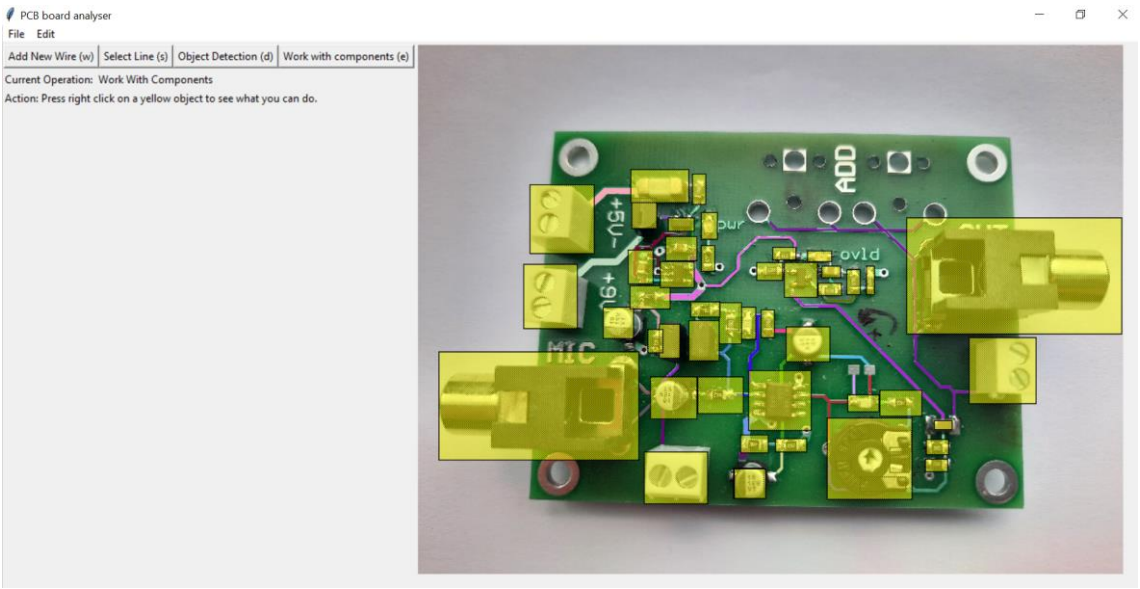


61. ábra – Kiválasztott komponens kapcsolatainak megjelenítése

5. Konklúzió, eredmények, továbbfejlesztési lehetőségek ismertetése

A diplomatervezés tárgy keretén belül egy olyan alkalmazás megvalósítása volt a cél, amely nyomtatott áramköri lemezeken található komponensek, huzalozások felismerését képes félautomatikus módon elvégezni. A rendszer megalkotása során megismerkedtem az alapvető képfeldolgozási technikákkal, hogyan kivitelezhető objektumok detektálása betanított neurálishálók nélkül, milyen számos lehetőség nyílik a python programozási nyelvben, és hogyan alkothatunk meg egy egyszerű felhasználói interfészt, amellyel sokkal gördülékenyebb egy alkalmazás használata.

A végső kialakított rendszer képes a fent említett célokat megvalósítani. A vezetékek és komponensek hibás javaslata esetén is van lehetőség az eredmények korrekciójára. Képes az alkalmazás egy olyan végleges eredményt előállítani, ahol a vezetékek különböző színnel elkülönülnek egymástól, míg a komponensek egységesen transzparens sárga színnel jelenítődnek meg. Egy ilyen végleges eredményt mutat az alábbi 62. ábra is.



62. ábra – Egy NYÁK analizálásának eredménye

Továbbfejlesztési célként az alábbiak tűzhetők ki:

- Az felhasznált algoritmusok finomhangolása, illetve újak megismerése.
- A regisztrált huzalozások és komponensek kapcsolási rajzának legenerálása. Ehhez azonban előtte szükséges beazonosítani az egyes alkatrészek lábainak elhelyezkedését.

- Megismerkedni különböző szövegfelismerő alkalmazásokkal, mint például Tesseract, és ennek segítségével a detektált alkatrészek feliratai alapján a dokumentációk begyűjtése.
- Az automatikus működés mértékének növelése.
- Felhasználói interfész megjelenésének fejlesztése.

Köszönetnyilvánítás

Szeretném megköszönni konzulensemnek, dr. Orosz Györgynek az egész éves kitartó támogatását, segítőkészségét, hogy tudásával hozzájárult a kitűzött célom eléréséhez.

Irodalomjegyzék

[1] U.S. Environmental Protection Agency. "Wastes - Resource Conservation - Common Wastes & Materials - eCycling.” [utolsó hozzáférés: 2020 december 08.]

[2] Python. 2020. python.org. URL: https://www.python.org/ [utolsó hozzáférés: 2020 december 08.]

[3] Tkinter, 2020. python.org. URL: https://docs.python.org/3/library/tkinter.html [utolsó hozzáférés: 2020 december 08.]

[4] Thibaut Julliand, Vincent Nozick, Hugues Talbot, „Image Noise and Digital Image Forensics” [utolsó hozzáférés: 2020 december 08.

[5] Wikipedia. 2020. Gaussian function. URL: https://en.wikipedia.org/wiki/Gaussian_function [utolsó hozzáférés: 2020 december 08.]

[6] Miklós Árpád, Dr. Vámossy Zoltám, „Képfeldolgozás és párhuzamosíthatóság”, [utolsó hozzáférés: 2020 december 08.]

[7] Czúni László, Tanács Attila, „Képi információ mérése”, 7. fejezet

[8] OpenCV. 2020. opencv.org. URL: https://opencv.org/ [utolsó hozzáférés: 2020 december 08.]

[9] Gaussian Blur: URL: https://datacarpentry.org/image-processing/06-blurring/ [utolsó hozzáférés: 2020 december 08.]

[10] Wikipedia, 2020, Gaussian Blur, URL: https://en.wikipedia.org/wiki/Gaussian_blur [utolsó hozzáférés: 2020 december 08.]

[11] Bilateral filter, 2020, hu.qaz.wiki, URL: https://hu.qaz.wiki/wiki/Bilateral_filter [utolsó hozzáférés: 2020 december 08.]

[12] Sofiane Sahri, Canny Edge Detection Step by Step in Python – Computer Vision, 2019, URL:https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123 [utolsó hozzáférés: 2020 december 08.]

[13] Sofiane Sahir, „Canny Edge Detection”, URL: http://justin-liang.com/tutorials/canny/#suppression [utolsó hozzáférés: 2020 december 08.]

[14] Python filtering, 2020. opencv-python-tutroals.readthedocs.io URL: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html [utolsó hozzáférés: 2020. december 08.]

[15] Contours: Getting started, 2020, docs.opencv.org, URL: https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html [utolsó hozzáférés: 2020.december 08.]

[16] Satoshi Suzuki. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[17] Morphological Transformations, 2020, docs.opencv.org, URL: https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html [utolsó hozzáférés: 2020.december 08.]

[18] Csetverikov Dmitrij. ELTE, „Digitális képelemzés alapvető algoritmusai”

[19] Harris Corner Detection, 2020, docs.opencv.org URL:https://docs.opencv.org/master/dc/d0d/tutorial_py_features_harris.html [utolsó hozzáférés: 2020.december 08.]

[20] Harris Corner Detector, AI Shack, URL:https://aishack.in/tutorials/harris-corner-detector/ [utolsó hozzáférés: 2020.december 08.]

[21] Shi-Tomasi Corner Detector & Good Features to Track, 2020, docs.opencv.org, URL:https://docs.opencv.org/4.0.0-beta/d4/d8c/tutorial_py_shi_tomasi.html [utolsó hozzáférés: 2020.december 08.]

[22] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[23] Circle Hough Transform, 2020, hu.qaz.wiki, URL: https://hu.qaz.wiki/wiki/Circle_Hough_Transform [utolsó hozzáférés: 2020.december 08.]

[24] Feature Detection – Hough Circles, 2020, docs.opencv.org, URL:https://docs.opencv.org/master/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d [utolsó hozzáférés: 2020.december 08.]

[25] Wikipedia. 2020. Ramer-Douglas-Peucker algortimus, URL:https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm [utolsó hozzáférés: 2020.december 08.]

[26] Aishwarya Singh, Selecting the right bounding box using Non-maximum suppression, 2020, analyticsvidhya.com URL:https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/ [utolsó hozzáférés: 2020.december 08.]

[27] J.R.R. Uijlings, K.E.A. van de Sandle, T. Gevers, and A.W.M Smeulders. Selective Search for Object Recognition, 2012.

[28] C.T. Zahn. Graph-theoretic methods for detecting and describing gestalt clusters. IEEE Transactions on Computing, vol 20, pag es 68-86, 1971.

[29] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient Graph-Based Image Segmentation. *IJCV*, 59:167–181, 2004. 1, 3, 4, 5, 7