

TDK-dolgozat

Máthé Edina

Veres Tímea-Bernadette

XXXVII. Országos Tudományos Diákköri Konferencia

Informatikatudományi szekció

2025

Vonalrajzok generálása fényképek alapján és a Line-us rajzoló robot vezérlése

Intézmények:

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Codespring

Szerzők:

Máthé Edina

**Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar,
Informatikai-matematika szak, III. év**

Veres Tímea-Bernadette

**Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar,
Informatikai-matematika szak, III. év**

Témavezetők:

dr. Simon Károly, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Farkas Eszter, szoftverfejlesztő

Codespring

Guti Zsombor, szoftverfejlesztő

Codespring



Vonalrajzok generálása fényképek alapján és a Line-us rajzoló robot vezérlése

Máthé Edina, Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar,
Informatikai-matematika szak, III. év
Veres Tímea-Bernadette, Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar,
Informatikai-matematika szak, III. év

dr. Simon Károly, egyetemi adjunktus
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar
Farkas Eszter, szoftverfejlesztő
Codespring
Guti Zsombor, szoftverfejlesztő
Codespring

A projekt célja egy gyerekeket és fiatalokat célzó telefonos applikáció létrehozása, amely alkalmas tudományszerűsítésre és szórakoztatásra. A Line-us robot egy robotizált rajzoló, amelynek irányításához egy felhasználóbarát felületet nyújt az alkalmazásunk.

Az applikáció a Line-us robottal WiFi kapcsolaton keresztül kommunikál, ezáltal lehetővé téve a felhasználónak, hogy rajzolásra bírja a robotot. A projekt keretein belül megvalósult fő funkcionálisok: kapcsolat létesítése a robottal és kommunikáció, valamint képek elkészítése, feldolgozása és a robot által értelmezhető formátumba alakítása.

A felhasználónak lehetősége van fényképet készíteni vagy egy képet betölteni a saját fájlrendszeréből. Ezután különböző műveleteket végezhet el a képpel, mint például rajzfilmesítési effektust alkalmazhat rá, kivághat belőle, vagy kinyerheti a kép vonalrajzát. A képfeldolgozási folyamatokhoz egy arcfelismerő is társul, mely arc észlelése esetén egy képkivágást javasol a felhasználónak. Mindezek mellett egy rajzfelület segítségével biztosítjuk a szabad rajzolás lehetőségét is. A képfeldolgozási funkcionálisok elérhetőek egy Line-us robot nélkül is, de ha papíron szeretnénk vizuálítani a rajzokat, akkor kapcsolatba léphetünk a robottal, és kirajzoltathatjuk a képből kinyert vonalrajzot, vagy a saját rajzunkat.

Tartalomjegyzék

Bevezető	1
1. A Line-us projekt	3
1.1. Funkcionális követelmények	3
1.2. Nem funkcionális követelmények	4
1.3. Architektúrális felépítés	4
2. Az alkalmazás felépítése	6
2.1. Android mobil alkalmazás felépítése	6
2.2. Android mobil alkalmazás funkcionalitásai	7
2.2.1. Kapcsolódás	7
2.2.2. Light és Dark mód	8
2.2.3. Képkészítés és képfeltöltés	8
2.2.4. Képszerkesztés	8
2.2.5. Rajzfelület	9
2.3. Kapcsolat és kommunikáció a Line-us robotkarral	12
2.3.1. Kapcsolat létesítése	13
2.3.2. A kommunikáció	14
3. Képfeldolgozás	17
3.1. Képmegjelenítés	17
3.2. Vonalrajz generálás	18
3.3. Képek stilizálása	21
3.3.1. Rajzfilmesítés OpenCV segítségével	21
3.3.2. Rajzfilmesítés GAN hálózatokkal	23
4. A fejlesztés módszertana és minőségbiztosítás	29
4.1. Fejlesztési folyamatok	29
4.2. Tesztek	30
4.3. Pipeline-ok	31
Következtetések és továbbfejlesztési lehetőségek	32

Bevezető

A gyermekek és fiatalok tudományos ismeretek iránti érdeklődését felkelteni és fenntartani kihívás, melynek megkönnyítésére sok játékos megközelítés született. A Line-us projekt is egy ilyen játékos, tudománynépszerűsítő alkalmazás létrehozását célozza, amely innovatív és szórakoztató módszerekkel közelít a probléma megoldásához. Az alkalmazás és az általa irányítható robot játékos interaktivitást biztosít és a tudományos szenvedély serkentését is szolgálja, szórakoztató formában.

A modern eszközök, mint például a robotok használata, gyors és játékos megoldást nyújtanak rajzok készítésére. Egy ilyen megoldás a Line-us, melynek egyedülálló funkciója, hogy közel valós időben alakít és módosít fotókat, illetve képeket vonalrajzokká, amelyeket a robot kirajzol papírra. Az alkalmazásban lehetőség van saját rajz készítésére egy rajzfelületen, és a fényképeket rajzfilmesíteni is lehet.

A Line-us robot egy kisméretű, hordozható eszköz, melynek rajzoló felületére papírt, míg a robot karjához íróeszközt rögzíthetünk. Rajzoláshoz szükséges egy WiFi kapcsolat kialakítása az alkalmazás és a robot között. Ehhez először a robotot áramforráshoz csatlakoztatjuk, majd az alkalmazásból megteremtjük a kapcsolatot. Sikeres kapcsolatfelépítés esetén a roboton pislogó fényt látunk, és amikor az alkalmazásból rajzolást indítunk, akkor a robot elkezd mozgatni a rajzoló kart.

A dolgozat további része a következő fejezetekből áll: az 1. fejezetben a Line-us projektben megjelenő funkcionális és nem funkcionális követelmények leírása, valamint a projekt architektúrális felépítésének ismertetése található; a 2. fejezet a Line-us alkalmazás felépítését tartalmazza, alaposan kitérve a megjelenő funkcionálisításokra és a kommunikációhoz szükséges kapcsolat megteremtésére; a 3. fejezetben a képfeldolgozás folyamatait ismertetjük, és bemutatjuk, hogy hogyan nyerjük ki a képekből a vonalrajzot, hogyan tároljuk és jelenítjük meg a képeket, valamint hogy hogyan jutunk el a képek rajzfilmesített változatáig; ezt követően a 4. fejezetben a munkafolyamat módszereit ismertetjük, és az applikációhoz készült teszteket, illetve pipeline-t mutatjuk be; végül sor kerül az alkalmazás továbbfejlesztési lehetőségeinek bemutatására és a következtetések levonására.

A projekt alapjait 2023 nyarán, a Codespring által szervezett nyári szakmai gyakorlat¹ keretén belül fektettük le. A szakmai gyakorlat alatt az volt a cél, hogy kapcsolatot tudjunk

¹<https://edu.codespring.ro/codespring-nyari-szakmai-gyakorlat-2023/>, elérés dátuma: 2024-04-30

létesíteni a Line-us robottal, kinyerjük a képekből a vonalrajzot és azt megfelelő formátumban elküldjük a robotnak. A 2023-2024-es egyetemi tanév csoportos projekt² tantárgy keretén belül tovább dolgoztunk a projekten, amikor három szaktársunk ideiglenesen csatlakozott csapatunkhoz, név szerint: Kis Aranka-Enikő, Szócs Bence-Kristóf és Varga Veronika-Katalin. Nekik köszönhető a rajzfelület néhány funkcionálitása, valamint a képkivágás arányosítása.

Ezúton szeretnénk köszönetet mondani szaktársainknak a hozzájárulásukért, mentorainknak: Farkas Eszternek és Guti Zsombornak, valamint témavezetőnknek, Dr. Simon Károlynak a szakmai segítségükért és támogatásukért. Köszönet illeti a Codespringet is a támogatásért, valamint a megfelelő fejlesztési infrastruktúra, illetve a Line-us robot biztosításáért.

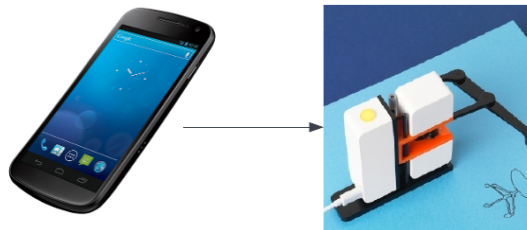
²https://www.cs.ubbcluj.ro/files/curricula/2023/syllabus/MIM_sem5_MLM5012_hu_s.kolumban_2023_8165.pdf, elérés dátuma: 2024-04-30

1. A Line-us projekt

A Line-us projekt egy felhasználóbarát mobilalkalmazást kínál a Line-us nevű rajzoló robothoz. Kapcsolatot teremt a Line-us robottal, képkészítést és képfeldolgozást valósít meg, valamint lehetőséget ad saját rajz készítésére az erre szánt felületen. Ennek a fejezetnek a célja a mobilalkalmazás funkcionális és nem funkcionális követelményeinek bemutatása, valamint az architektúrális felépítés ismertetése.

1.1. Funkcionális követelmények

Az alkalmazás kétféleképpen képes csatlakozni a Line-us robothoz (1. ábra): a robot saját WiFi hálózatán, vagy egy közös külső hálózaton keresztül. Az elsődleges kapcsolódási mód a robot saját WiFi hálózata. A kapcsolódást követően egy kommunikációs csatornát építünk fel, a robot irányításának érdekében. Az eszköz megfelelő vizuális visszajelzéseket nyújt a kapcsolat állapotáról. A felhasználónak lehetősége van arra, hogy megváltoztassa a Line-us hálózati kapcsolatát, egy külső hálózatra tudja csatlakoztatni a robotot. Ez a lehetőség azért szükséges, hogy a felhasználónak internet elérése is legyen az alkalmazás használata közben, mivel a Line-us WiFi elérési pontja nem szolgáltat internetet. Abban az esetben, ha a felhasználó az előbbi hálózatsváltást választja, az applikáció megkéri, hogy saját eszközét is csatlakoztassa ugyanarra a külső hálózatra, melyben ismét létrehozza a két eszköz közötti kapcsolatot.



1. ábra. Az alkalmazás képes csatlakoztatni a felhasználó eszközét a Line-us nevű robothoz.

A felhasználó képet tölthet be a saját fájlrendszeréből, valamint lehetősége van a készülék kamerájával is kép készítésére. Képkészítéskor az alkalmazás elindít egy arcfelismerő algoritmust, mely egy vagy több arc észlelése esetén képkivágást javasol a felhasználónak. Az elkészített, vagy galériából csatolt képeken további műveleteket tud végrehajtani a felhasználó. Az alkalmazás lehetőséget biztosít a kép kivágására, stilizálására, valamint vonalrajz generálására. Vonalrajz generálása esetén a felhasználó beállíthatja a rajzon megjelenő

vonalak sűrűségét, szabadon rajzolhat az elkészült vonalrajzra, és törölhet róla.

Az applikáció biztosít egy szabad rajzolásra szánt felületet is, melyen elérhetőek a törlés, visszavonás és visszaállítás műveletek. A felhasználó az elkészített rajzot, vagy a generált vonalrajzot a Line-us segítségével papírra rajzoltathatja.

Az alkalmazás megfelelő működéséhez szükség van a következő engedélyekre: helymeghatározás³, kamera használat, valamint a képekhez és tárhelyhez való hozzáférés. Ezek az engedélyek biztosítják, hogy a funkcionalitásokat teljes mértékben tudja kihasználni a felhasználó.

1.2. Nem funkcionális követelmények

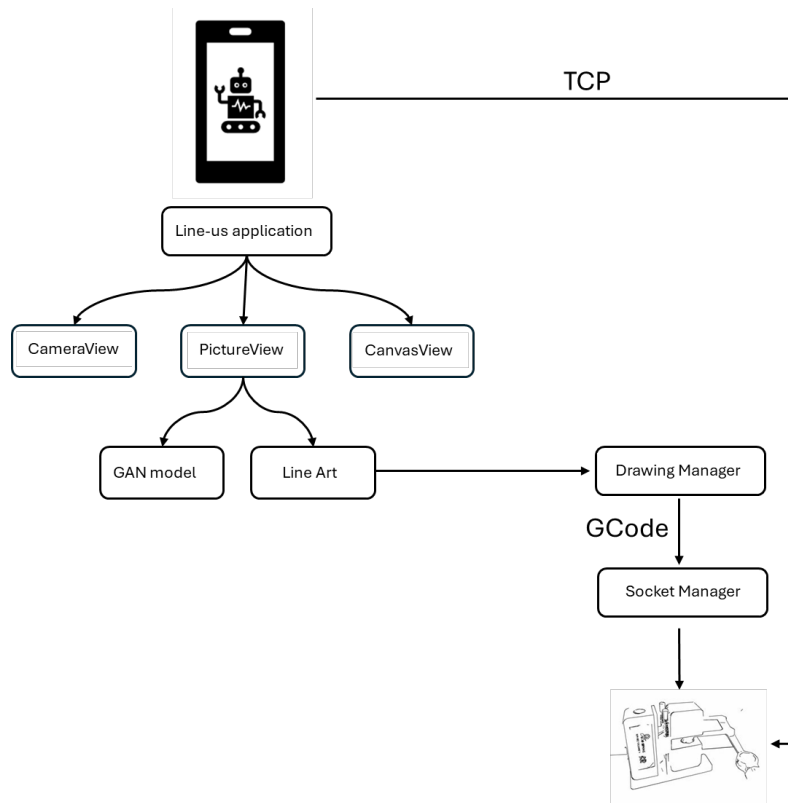
Az alkalmazásunk egy mobilapplikáció, amely Android eszközöket támogat. Kompatibilis a legújabb verziókkal⁴, valamint visszamenőleges kompatibilitást is biztosít. A megfelelő működéshez biztosítunk egy stabil kommunikációs csatornát a Line-us és a felhasználó eszköze között, a robot irányításának érdekében. A nem funkcionális követelményekhez tartozik, hogy a megjelenő felületek és oldalak intuitívak és könnyen kezelhetőek, mindig lehetőséget adnak arra, hogy a felhasználó visszalépjen az előző oldalra. Az alkalmazás közel valós időben válaszol a képformázási kérésekre. Képkivágás és stilizálás esetén biztosítja a szerkesztett kép előnézetét, így a felhasználó el tudja dönteni, hogy valóban szeretné-e az adott módosítást alkalmazni.

1.3. Architektúrális felépítés

Az alkalmazásunk teljes funkcionalitásainak kihasználásához szükségünk van két eszközre: egy Android eszközre és a Line-us robotra. Az alkalmazás lokálisan fut az Android eszközön, közvetlenül az eszköz erőforrásait használva, így biztosítva a gyors és megbízható működést külső szerverek bevonása nélkül. Az alkalmazás számos funkciót kínál, melyek a Line-us robot nélkül is elérhetők: a fényképkészítési, fényképkivágási, rajzfilmesítési, és rajzolási funkcionalitások. Ha az Android eszközünket WiFi kapcsolattal rácsatlakoztatjuk a robotra az alkalmazáson keresztül, akkor lehetővé válik a már felsorolt funkcionalitásokon kívül a vonalrajzok generálása és a rajzok papírra történő kirajzolása.

³Valójában nem használjuk fel a felhasználó helyadatait, hanem a WiFi élérés érdekében követeli meg ezt az engedélykérést az Android.

⁴Jelenleg utoljára kiadott Android verzió: Android 14: <https://developer.android.com/about/versions/>, elérés dátuma: 2024-04-30



2. ábra. Architektúra

A kommunikációt illetően TCP socket-en keresztül kapcsolódunk a Line-us robothoz. A fényképkészítési lehetőségeket a CameraView teremti meg, a CameraX függvénykönyvtár által biztosított funkcionalitásokkal. A fényképek megjelenítése a PictureView-n keresztül történik, ami összeköttetésben van a képmódosítási és stilizálási funkciókkal, beleértve a GAN modell alapú rajzfilmesítést is. A CanvasView a rajzfelület és a rajzolási funkciók hozzáférhetőségét teszi lehetővé. A PicturePreview segítségével generált vonalrajz, valamint a CanvasView-n készített vonalrajz koordinátáit a DrawingManager alakítja a robot által is értelmezhető G-code parancsokká, ezeket a SocketManageren keresztül továbbítjuk a robot felé. (2. ábra)

2. Az alkalmazás felépítése

2.1. Android mobil alkalmazás felépítése

Az általunk készített mobilapplikáció két fő funkcionalitással rendelkezik: képek kezelése, és kapcsolatteremtés a Line-us robottal. Az utóbbi szolgáltatás már meg volt valósítva 2018-ban, a robot megjelenése után, a KickStarter⁵ jóvoltából [7]. Ez a hivatalos Line-us applikáció sajnos már nem kompatibilis olyan eszközökkel, melyeken Android 10 fölötti verzió található. Így az Androidos eszközökkel rendelkező felhasználók nagy része nem tud hozzáférni az említett alkalmazáshoz. Az általunk készített mobilapplikáció ezt a problémát orvosolja, ugyanis ez egy Kotlinban írt, natív Android alkalmazás, mely elérhető a legújabb Android verziókon, és visszafelé is kompatibilis.

Az alkalmazásunk az MVVM (Model-View-ViewModel) [6] tervezési mintát követi, ami három alapvető komponensre épül: a modellre (Model), a nézetre (View), és a nézetmodellre (ViewModel). Ebben az architektúrában a *modell* felel az üzleti logika és az adatmodell képviseléséért. Az adatmodellünk a képekkel kapcsolatos információkat, mint például azok elérési útjait és méreteit tartalmazza, melyek függetlenek a felhasználói felülettől és a megjelenítési logikától. Az üzleti logika magába foglalja a képek kezelésének, az arcdetektálásnak, a képszerkesztési funkcióknak - például a fényképek kivágásának ajánlása és megvalósítása -, a rajzfilmstílus alkalmazásának, valamint a Line-us robottal való kapcsolat létrehozásának folyamatát.

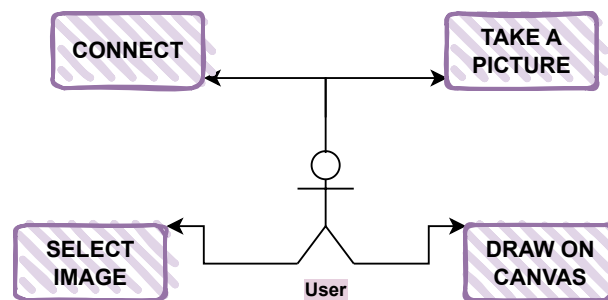
A *nézet*, mely az Androidon belül Activity-ket, Fragment-eket és layout-okat foglal magába, a felhasználói felületen megjelenő elemekért felelős. Ezek az elemek nem létesítenek közvetlen kommunikációt a *modell* adataival, ehelyett a *modell* és a *nézet* közötti kommunikációt a *nézetmodell* hivatott megvalósítani.

A *nézetmodell* közvetíti az adatokat a *modellből* a *nézethez*, ezáltal a felhasználói felület megjelenítéséhez szükséges átmeneti réteget biztosítva. Ez magában foglalja a felhasználói interfészhez kapcsolódó logikát, kezeli a felhasználói interakciókat, és a megfelelő adatokat kinyerve a *modellből*, frissíti a *nézetet*.

⁵<https://www.kickstarter.com/>, elérés dátuma: 2024-03-10

2.2. Android mobil alkalmazás funkcionalitásai

Az alkalmazásban jelen lévő funkcionalitások lehetővé teszik, hogy a felhasználók változtatásokat hajtsanak végre képeiken, valamint interakcióba lépjenek a Line-us robottal. Az egyik lényeges funkció a kapcsolatteremtés a Line-us robottal, amely felelős a robot és az eszköz közötti kommunikációs csatorna létrehozásáért az adatok átvitelének érdekében. Lehetőség van a felület témájának (sötét, világos) változtatására. A galériából való képbetöltés a már elkészült fényképek betöltését biztosítja, míg a képkészítés során a felhasználó az alkalmazáson belül tud fényképet készíteni, ami egy másik funkcionalitáshoz is kapcsolódik, amikor arcfelismerés alapján képkivágási keret ajánlást kap egy olyan képkivágásra, ahol az arc kerül előtérbe. Bármilyen fénykép esetén a fényképek képkivágása opcionális marad, ahogyan a rajzfilmesítés is. Másik funkció a vonalrajz generálás, amelyhez a vonalrajzok részleteinek finomítását is el lehet végezni, emellett jelen van a vonalgrafika kirajzoltatásának lehetősége, illetve egy rajzfelület használata is. A rajzfelületre történő rajzolás érdekében a *Draw on Canvas* gombot nyomja le a felhasználó, és ennek hatására megjelenik egy fehér rajzfelület, melyre fekete színnel lehet rajzolni. Ezen az oldalon öt különböző gomb található: a *Delete*, *Undo*, *Redo*, *Erase*, valamint a *Start Drawing*. A *Delete* opció segítségével a felhasználó teljesen törölni tudja a rajzát, az *Undo* és *Redo* a műveletek visszavonására és visszaállítására hivatottak. Az *Erase* gombot lenyomva, az érintett vonalak kitörölődnek, és ha már nem szeretne többet törölni a felhasználó, akkor ismét lenyomva ezt a gombot, újra rajzolhat. A *Start Drawing* segítségével a felhasználó kirajzoltathatja a rajzát a robottal. Az 3. ábra szemlélteti, hogy milyen lehetőségei vannak a felhasználónak, vagyis, hogy milyen gombok jelennek meg a főoldalon.



3. ábra. A főoldalon megjelenő gombok, amelyeket elérhet a felhasználó.

2.2.1. Kapcsolódás

Az alkalmazásunk képes kapcsolódni a Line-us robothoz, amely a robot által fenntartott WiFi hálózatra csatlakozik. A WiFi elérése érdekében helymeghatározás engedélyre van

szüksége az applikációnak. Amint rácsatlakozik a felhasználó eszköze a robotra, a háttérben kommunikációs csatorna épül fel. Mindezt a *Connect* gomb segítségével valósíthatja meg a felhasználó, melynek lenyomása egy párbeszédablakot jelenít meg. A felhasználó az alkalmazás segítségével olyan parancsot is tud küldeni a robotnak, amellyel változtatni lehet a Line-us hálózati kapcsolatán. Ez akkor fontos, ha szeretné a felhasználó, hogy a saját eszközének internet kapcsolata is legyen a robotkapcsolat fenntartása közben, ugyanis a Line-us WiFi hálózata nem szolgáltat internetet. Ha ezt az opciót választja, az alkalmazás egy párbeszédablakban megkéri, hogy csatlakoztassa rá a kiválasztott külső hálózatra saját eszközét is. Amint átsatlakozik a robot a külső hálózatra, megszűnik az eredetileg létrehozott csatorna, így a kapcsolatot újra létre kell hozni.

2.2.2. Light és Dark mód

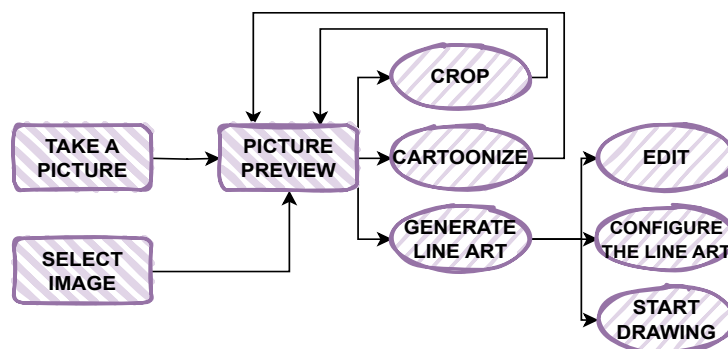
Egy jobb felhasználói élmény érdekében alkalmazásunkban implementáltuk a világos és sötét mód váltásának lehetőségét. Az alkalmazás alapértelmezetten világos módra van állítva, de a főoldalon elhelyezett gomb segítségével a felhasználók váltogathatnak a világos és sötét nézet között, preferenciájuknak megfelelően. A projektben a két mód bevezetése lehetővé teszi, hogy szélesebb felhasználói kör számára tudjuk testreszabni a felhasználói felület kinézetét. Mindkét mód esetében kiemelt figyelmet fordítottunk arra, hogy a feliratok és felhasználói felület elemei minden körülmények között jól olvashatóak maradjanak.

2.2.3. Képkészítés és képfeltöltés

A *Select Image* illetve a *Take a Picture* opciók kihasználása érdekében is engedélyt kell adnia a felhasználónak, hogy az applikáció hozzáférjen az eszköz tárhelyéhez, képekhez, videókhoz, valamint a kamerához. A *Select Image* gomb megnyomása után megnyílik a felhasználó galériája, és abból egy megfelelő formátumú (.jpg, .jpeg, .png) médiafájl kiválasztva betöltődik a kép előnézete. Képkészítés esetén is teljesen hasonló a helyzet, amennyiben nem azonosított arcot az alkalmazás, betölti a kép előnézetét, különben a minden felismert arcot tartalmazó ajánlott képkivágással felnyitott oldalra navigálja a felhasználót.

2.2.4. Képszerkesztés

A képelőnézet ablakban a felhasználó elérheti a kép szerkesztésére hivatott gombokat, pontosabban a kivágást, rajzfilmesítést, valamint vonalrajz generálást (lásd 4. ábra). A



4. ábra. A képfeldolgozás folyamata.

képkivágás és rajzfilmstílus alkalmazásakor egy új ablakban jelenik meg a módosított kép, és a felhasználó eldöntheti, hogy elfogadja ezeket a módosításokat, vagy visszalép az eredeti állapothoz. A vonalrajz generálását kiválasztva egy új oldalon nyílik meg a vonalrajz, ahol az ennek módosítására szolgáló gombok is megjelennek, ezek a szerkesztés, a vonalrajz konfigurálása, valamint a kirajzolás gombok. A szerkesztést választva a rajzfelületen nyitjuk meg a vonalgrafikát, melyre rajzolhatunk, illetve törölhetünk róla. Ha a felhasználó szeretné változtatni a vonalrajz generálás beállításait, akkor egy felugró párbeszédablakban módosíthatja a vonalak sűrűségét. A kirajzolást elindító gombot választva, megtörténik a vonalrajz kirajzolása a robot által.

Ugyan adatbázisra nincs szükségünk, de a képeket valamilyen formában el kell tárolnunk. Ennek érdekében az alkalmazás engedélyt kér a felhasználótól - ahogyan az előbbi bekezdésben is említettük, - hogy hozzáférjen a galériához, valamint az eszköz tárhelyéhez. Így az elkészült fotókat lementjük a felhasználó eszközére, és a továbbiakban egy URL⁶ segítségével tudunk hivatkozni a képre. Hasonlóan elmentjük a képet abban az esetben is, ha a felhasználó módosít rajta (rajzfilmesíti, kivág belőle, vonalrajzot generáltat), mentés után pedig frissítjük az aktuális URL címet.

2.2.5. Rajzfelület

Jelen fejezet az alkalmazás rajzfelületének bemutatásával foglalkozik. A rajzfelület alatt egy üres canvas-t értünk, amelyben vonalrajzok készíthetők úgy, hogy a felhasználó közvetlenül az eszköze képernyőjére rajzol.

⁶Az URL (*Uniform Resource Locator*) egy egyedi cím, mely pontosan meghatározza egy adott tartalom, például kép vagy weboldal helyét az interneten. Amikor egy eszközön, a saját tárhelyében lévő kép URL-jére hivatkozunk, akkor ezáltal megjelöljük a kép pontos elhelyezkedését az eszköz belső memóriájában. Tehát az URL lehetővé teszi számunkra, hogy könnyen és pontosan hivatkozzunk egy képre.

Az említett funkció elérhető az alkalmazás főoldaláról, egy Draw on Canvas gomb alatt. A funkció használatakor a felhasználó előtt egy fehér alapú üres vászon jelenik meg, alatta pedig a rajzolást segítő gombok, balról jobbra a következő sorrendben: Delete (törlés), Undo (visszavonás), Redo (visszaállítás), Erase (radírozás) és Start Drawing (rajzolás indítása). Az alábbiakban ezek működéséről lesz szó.

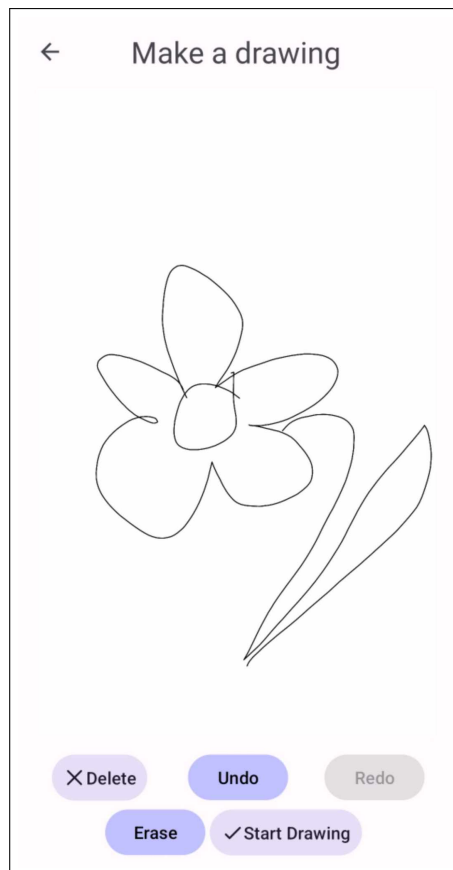
A Delete gomb, ahogy a neve is utal rá, a rajz végleges és teljes törlését végzi, azaz minden elmentett koordinátát kitörlünk.

Az Undo gomb a rajzolt vonalak visszavonásáért felelős. Minden kattintáskor törli az utoljára rajzolt vonalakat.

A Redo gomb ezzel ellentétben visszaállítja az Undo gomb által eltávolított vonalakat, abban a fordított sorrendben, ahogy ezek törölve voltak, tehát a legutoljára törölt vonal jeleníthető meg a leghamarabb.

Az Erase gomb és a Delete gomb különböző törlésekért felel. Míg a Delete az egész rajzot egy gombnyomással törli, addig az Erase csak azokat az összefüggő vonalakat tünteti el, amelyekre a felhasználó rávezeti az ujját, ezzel nagyobb rugalmasságot biztosítva a felhasználónak, mivel rajzoláskor gyakran csak bizonyos részeket kell javítani vagy eltüntetni, nem pedig mindig újakezdeni. Az Erase gomb működése különböző módon működik a felhasználó által rajzolt vonalakra, mint a képből generált vonalrajz esetében. Míg a törlés során érintett, felhasználó által húzott összefüggő vonalakat teljes egészében törli, addig a generált vonalak esetében részleges törlés történik, kitöröljük azokat a koordinátákat, amelyeket a törlés során érintettünk. Fontos megemlíteni, hogy a Redo és Undo gombok egymásra hatnak, azaz ha elvégeztük valamelyik gombbal a fent említett műveletek egyikét, a másik gombot is tudjuk alkalmazni. Ez viszont nem érvényes a Delete gombra, mert egyszer alkalmazva az előző rajz már nem hozható vissza.

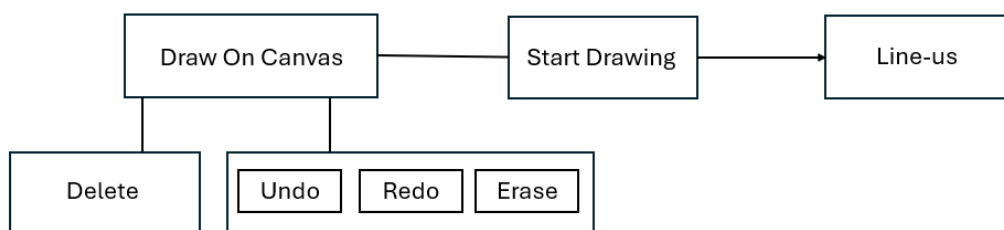
Az utolsó gombunk a Start Drawing, melynek lenyomásakor az alkalmazás ellenőrzi, hogy a Line-us robottal megvan-e a kapcsolat. A kapcsolat hiányában egy értesítést kap a felhasználó arról, hogy előbb szükséges a Line-us robothoz kapcsolódni. Ha megvan a kapcsolat, akkor az elmentett koordinátákat átalakítjuk G-code formátumba, majd egy TCP socketen keresztül elküldjük a Line-us robotnak, és a robot elkezdi a képernyőn megjelenített rajzot kirajzolni. Ugyanekkor történik a rajz elmentése is, amelyet Bitmap-ként tárolunk. Az eszköz külső tárhelyére való mentés garantálja, hogy a rajz megmarad, így a jövőben lehetőség lesz majd visszanézni a korábban elkészített rajzokat. A kirajzoltatás esetében fontos tudni hogy egy



5. ábra. A rajzfelület

rajzolás indításakor a képernyőn látható rajzot teljes egészében kirajzolja a robot. Tehát az utólagos hozzáadásokat, vagy törléseket részleteiben nincs lehetőség kirajzolni: javítás esetén javasolt egy tiszta papírlappal indítani. A G-code parancsok egy sor utasítást tartalmaznak, amelyek a rajzolási folyamat során a kar mozgását határozzák meg elejétől a végéig, így nincs lehetőség közbeiktatásra. A rajz egyes részeinek módosításakor az alkalmazásnak újra kell generálnia az egész G-code sorozatot és újra elindítani a rajzolás folyamatát. Egy másik korlátozás a robot rajzolási területéből fakad. A kirajzolási folyamat újra kezdésekor a rajz meglévő komponenseit a Line-us nem fogja egymásra rajzolni. Ez a viselkedés a képek skálázásának köszönhető, mivel a kép kirajzolásakor a vonalgrafikát a képméret alapján kicsinyíteni vagy nagyítani fogja.

A rajzolás során a vonalak mentése és tárolása kulcsfontosságú lépés a helyes G-code koordináták generálásához és ennek következményeként a kirajzoláshoz. Egy olyan vonalhúzást, amikor a felhasználó az ujját nem veszi fel a képernyőről, egy útvonalnak tekintünk. Az útvonalak x és y koordinátáit Descartes-féle koordinátarendszer szerint tároljuk. Fontos komponens a használt adatstruktúra, melynek képesnek kell lennie az összes pont



6. ábra. A rajzfelület komponensei

tárolására és gyors elérésére. Esetünkben egy `MutableList<List<Pair<Float, Float>>>` struktúrára esett a választás, mivel lehetőséget ad az útvonalban szereplő pontok gyors elérésére és csoportosítására, megőrizve a rajzolás sorrendjét. A `MutableList` használata lehetővé teszi a könnyű hozzáférést az elemekhez, azok módosítását és törlését. Dinamikus szerkezete miatt ideálisan megfelel céljainknak. A `MutableList` támogatja az elemek helyben történő módosítását, hozzáadását és eltávolítását anélkül, hogy a teljes adatstruktúra újraépítését igényelné, így megfelel a listák tárolására. A pontok tárolásához a `List` használata a rajzolási sorrend megőrzése miatt van alkalmazva, míg a `Pair` egy pont (x,y) koordinátáinak elmentésére szolgál. Tehát minden egyes képernyőkontaktus, illetve annak megszüntetése közötti útvonal egy pontpárokat tartalmazó listából áll, ahol minden pontot (x, y) koordinátával reprezentálunk.

Az érintési események kezelése, beleértve az érintést, a húzást és a felengedést, az `onTouchEvent` metóduson keresztül történik, amely lehetővé teszi a pontpárok listájának frissítését. A koordináták frissítésével, valamint az `invalidate` metódus hívásával biztosított, hogy a rajzfelületen megjelenjen az útvonal. Az `invalidate` metódus feladata, hogy megjelölje a Canvas-t, mint újrarajzolandó területet, ezzel aktiválva az `onDraw` metódust, amely frissíti a felhasználói felületet.

2.3. Kapcsolat és kommunikáció a Line-us robotkarral

A fejezet a Line-us robottal való kommunikációhoz szükséges kapcsolódási folyamatokat részletezi, különös figyelmet fordítva a kapcsolat létrehozására, az interakciókra, és az adatátviteli mechanizmusokra.

2.3.1. Kapcsolat létesítése

Az utasítások küldésének érdekében egy kapcsolatot szükséges létrehozni a robot és az alkalmazás között. A Line-us robot programozását számos könyvtár segíti, amelyek különböző programozási nyelveken érhetőek el. Ilyen például a Line-us Python modul, amely a robot vezérlésére szolgáló alapvető funkciókat és parancsokat teszi elérhetővé egy egyszerű interfészen keresztül. Ezen kívül van egy JavaScript könyvtár, amely a kapcsolódás és az üzenetsorba állítás (queueing) folyamatait kezeli. A PHP könyvtár pedig lehetővé teszi a Line-us robot használatát az említett nyelvvel. Projektünk során azonban ezekkel a könyvtárakkal nem foglalkoztunk, mivel Kotlin nyelvben nem készült könyvtár.

Dedikált könyvtár segítsége nélkül a kapcsolat két módon hozható létre: WebSocket vagy TCP (Transmission Control Protocol), mi az utóbbit választottuk projektünk megvalósításakor. Az adatok TCP socketen keresztüli küldésének folyamata több kulcsfontosságú lépésből áll. Először is, egy TCP kapcsolatot kell létrehozni az alkalmazás és a Line-us robot között. Létrehozunk egy kliens-oldali socketet és megvizsgáljuk, hogy van-e már létező socket kapcsolat, majd a robot IP címének és portszámának felhasználásával kísérletet teszünk a csatlakozásra. Minden Line-us robot gyárilag meghatározottan az 1337-es porton fogadja az adatokat, így egyszerre csak egy robothoz tudunk csatlakozni. Ha a kapcsolat létrejött, akkor sikeresen kommunikálhatunk az eszközzel, kudarc esetén hibakezelésre kerül sor.

A kapcsolat létrejöttét követően a sendData metódusunk biztosítja az adatok átvitelét G-code parancsok formájában. Minden elküldött adathoz egy lezáró karakterláncot fűzünk, amely jelzi az üzenet végét. Az adatfogadás során a receiveData metódus feladata várni a Line-us-tól érkező válaszokat.

Az alkalmazásban két kapcsolódási formát valósítottunk meg. A két különféle mód közül a felhasználó választhatja ki a számára megfelelőbbet. Az alapértelmezett, egyszerűbb kapcsolattípusnak megfelelően, a felhasználó a Line-us által létrehozott és sugárzott WiFi-hálózatra tud csatlakozni. A felhasználói élmény javításának érdekében az alkalmazás a Connection gomb lenyomása után elvégez egy hálózati szűrést név alapján, és kiválasztja a Line-us-Setup SSID-val rendelkező WiFi-t, amely egy dialógusablakban jelenik meg. Ha a felhasználó elfogadja valamelyik Line-us WiFi-re a csatlakozást, akkor a két eszköz között létrehozuk a kapcsolatot. Ha WiFi-hez való csatlakozás sikeres, továbblépünk, egyébként hibakezelés indul.

A második lehetőség egy alternatív kapcsolatfelvételt kínál a Line-us robottal. Ez

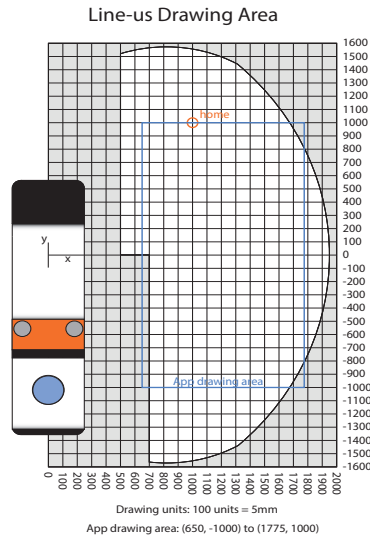
a lehetőség akkor adott, ha a felhasználó már rácsatlakozott a Line-us által sugárzott WiFi-hálózatra. Ekkor megváltoztathatjuk a Line-us által használt hálózatot és a robotot egy külső hálózatra csatlakoztathatjuk. Ez lehetővé teszi, hogy ezt követően ne közvetlenül a Line-us WiFi hálózatra csatlakozzunk, hanem alternatív megoldásként a mobil eszközüket és a Line-us robotot is ugyanarra a hálózatra csatlakoztatva hozzuk létre a kapcsolatot. Ennek előnye az, hogy lehetővé teszi a felhasználó számára, hogy az internetkapcsolatához is hozzáférjen a robot használata közben. Ha a hálózat nyitott, nem szükséges jelszó, viszont jelszóval védett hálózatok esetében a felhasználót az alkalmazásban megjelenő dialógusablakon keresztül kérjük, hogy adja meg a csatlakozáshoz szükséges jelszót a robot számára. Ekkor szükséges a felhasználónak rácsatlakoznia ugyanarra a Wi-Fi hálózatra az eszköz WiFi-beállításainál, hogy létrejöhessen a kapcsolat.

2.3.2. A kommunikáció

A fényképek kirajzolásának megvalósításához szükséges az adatok továbbítása a robot felé, a kommunikációs csatornán keresztül. A Line-us robotot, hasonlóan a CNC gépekhez, G-code utasításokkal lehet irányítani. A G-code, egy széles körben használt programozási nyelv, melynek alkalmazásai kiterjednek a 3D nyomtatástól kezdve a szerszámgépekig. Mivel a Line-us G-code utasításokat fogad, és mi vonalrajzot szeretnénk kirajzolni, a vonalrajzokat G-code formátumba kell alakítani.

A fényképek feldolgozásának eredményeként rendelkezésünkre áll egy éleket reprezentáló vonalrajz. Az alkalmazásbeli modell tér és a tényleges rajzolási felület különbözősége miatt szükség lesz egy koordináta transzformációra. A `scaleContoursToDrawingArea` funkció felelős azért a folyamatért, amelyben a kontúrokat átméretezzük a Line-us rajzterületéhez. Enélkül a lépés nélkül a robot nem tudná a kép összes részletét kirajzolni, mivel egyes részek a mozgásterületén kívül esnének. A `scaleContoursToDrawingArea` metódus működése során meghatározzuk a kontúrok x és y koordinátáinak minimumát és maximumát, annak érdekében, hogy megtaláljuk a kirajzolandó vonalgrafikák valós méreteit.

Figyelembe véve a Line-us rajzterületének előre meghatározott méreteit, amely x koordináta szerint 650 és 1750, y koordináta szerint pedig -1000 és 1000 között van, újra skálázzuk a kontúrokat, hogy azok illeszkedjenek a rajzterületre, miközben megőrizzük a kép arányait, 7. ábra. Meghatározzuk a kontúrok által elfoglalt téglalap alakú területet, azaz meghatározzuk a kontúrok pontjaiból a minimum és maximum értékeket mind az x, mind az y koordináták



7. ábra. Az ábrán látható a Line-us robot rajzolóterületének méretei. Képforrás: <https://github.com/Line-us/Line-us-Programming/blob/master/Documentation/LineUsDrawingArea.pdf>, elérés dátuma: 2024-03-24

szerint. A skálázáshoz szükséges a skálázási tényezők ($scaleX$ és $scaleY$) kiszámítása, melyekből a kisebbik értéket alkalmazzuk mindkét tengelyre, ezzel elkerülve a kép esetleges torzulását (lásd 2.3.2. képletek, ahol $lineUsMaxX$, $lineUsMaxY$, $lineUsMinX$, $lineUsMinY$ a robot rajzolóterületének határpontjai, míg a $maxX$, $maxY$, $minX$, $minY$ a kinyert kontúrok határpontjai).

$$\begin{aligned}
 scaleX &= \frac{lineUsMaxX - lineUsMinX}{maxX - minX} \\
 scaleY &= \frac{lineUsMaxY - lineUsMinY}{maxY - minY} \\
 scale &= \min \{scaleX, scaleY\}
 \end{aligned} \tag{1}$$

A kontúrok átskálázása után kezdődhet meg a konkrét G-code parancsok generálása. A G-code parancsok betűvel kezdődnek, amelyeket további szám és/vagy betűből álló karaktersorozatok követnek, így alkotva egy utasítást, amely meghatározza a robot műveletét. Minden rajzolási folyamat során első lépésként a robotkart a kiinduló pozícióba helyezzük, melynek parancsa **G28**. Ezt követően a kontúrok mentén haladva generáljuk az egyes G-code

parancsokat, amelyek lehetővé teszik a robotkar számára, hogy kövesse a kontúrok által meghatározott útvonalat. Az első pont meghatározásakor a robot gyorsasági beállítást (G00) és rajzoló parancsokat (G01) kap, Minden további vonal megrajzolása esetén csak rajzoló parancsokat használunk, amelyek formátuma G01 X{point.x} Y{point.y} Z0, ahol X és Y a kontúrpontok koordinátái, míg a Z0 a karfelemelés hiányát jelöli. Minden összefüggő vonal megrajzolása után a G01 Z1000 parancsot küldjük, ami a kar felemelését jelzi, hogy az újabb koordinátapár helyére tudjuk mozgatni a kart, ahol egy új összefüggő vonalat kell rajzolni. A végén a G28 parancsot használva a kart visszaállítjuk az eredeti pozícióba. Az említett G-code utasítások elküldését a robotnak a TCP socket kommunikáción keresztül valósítjuk meg.

3. Képfeldolgozás

A képfeldolgozás fejezetben bemutatjuk, hogy milyen technológiákkal valósítottuk meg a képek kezelését, szerkesztését, stilizálását és hogy hogyan jutottunk el a képektől a G-code parancsokig.

3.1. Képmegjelenítés

Ahogy a 2.1 fejezetben is említettük, az elkészült vagy módosított képeket elmentjük a felhasználó saját fájlrendszerébe. Ennek köszönhetően a továbbiakban az URL címükre hivatkozva tudjuk elérni a képeket, melyek betöltésére a harmadik féltől származó **Glide** könyvtárat használunk. [5] A Glide egy hatékony képkezelő és képbetöltő keretrendszer, mely Android platformon használható. Fő célja, hogy egyszerűbbé és gyorsabbá tegye a képek kezelését az Android alkalmazásokban. Mivel a felhasználó eszközének tárhelyéről kell betöltenünk a képeket, így szükséges a `READ_EXTERNAL_STORAGE` nevű engedély. A Glide kezeli az általa használt memóriát, vagyis nem a fejlesztők felelőssége a betöltött kép memóriában történő kezelése. A memória felszabadítást akkor végzi el, amikor a hozzárendelt *Activity*⁷ vagy *Fragment*⁸ objektum megsemmisül. Ezenkívül be lehet állítani azt is, hogy hova szeretnénk betölteni a képet, vagy milyen formátumban szeretnénk megkapni azt, valamint lehetőségünk van néhány memóriakezelési beállításra is: ezt kihasználva érjük el, hogy a Glide ne mentse a gyorsítótárba a képeket, hanem mindig töltsen újra. Ez az opció azért előnyös, mert gyakran változhatnak a képek, ezáltal az URL címük is, és nem szeretnénk sok helyet elfoglalni a felhasználó eszközének gyorsítótárából.

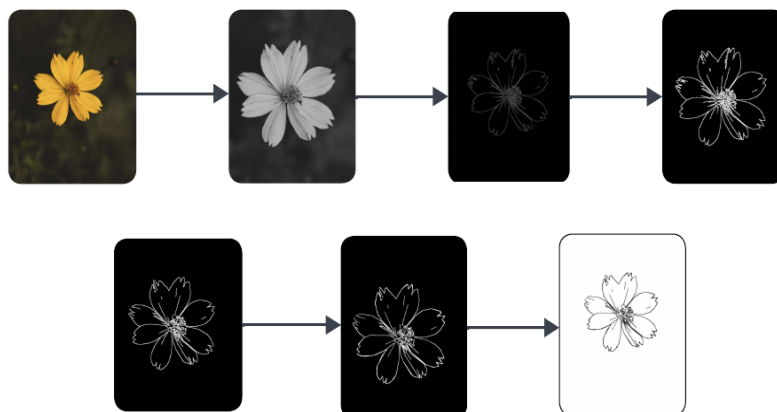
A kép megjelenítéséhez is használjuk a Glide könyvtárat, amelyet hozzákötünk a kép betöltését intéző *Activity*-hez, megadjuk a betölteni kívánt kép URL címét, és megmondjuk, hogy melyik *ImageView*⁹-ba helyezze el a képet. A Glide-ot arra a célra is használjuk, hogy egy URL alapján a képet *Bitmap*¹⁰ formátumban kapjuk vissza. A visszakapott bitmap-en különböző módosításokat végezhetünk el. A Glide-nak ezt a funkcióját akkor használjuk, ha

⁷<https://developer.android.com/reference/kotlin/android/app/Activity>, elérés dátuma: 2024-03-21

⁸<https://developer.android.com/reference/kotlin/android/app/Fragment>, elérés dátuma: 2024-03-21

⁹<https://developer.android.com/reference/android/widget/ImageView>, elérés dátuma: 2024-03-21

¹⁰A Bitmap egy képmegjelenítési formátum, amely alapvetően egy kétdimenziós rácsot alkot, és meghatározott szélességű, magasságú és színinformátumú képpontokból áll.



8. ábra. Balról jobbra haladva a képfeldolgozás állapotait láthatjuk rendre: eredeti -, szürkeárnyalatosított -, éldetektált kép, kinyert kontúrok, szűrt kontúrok, egyszerűsített kontúrok, végül az invertált kép.

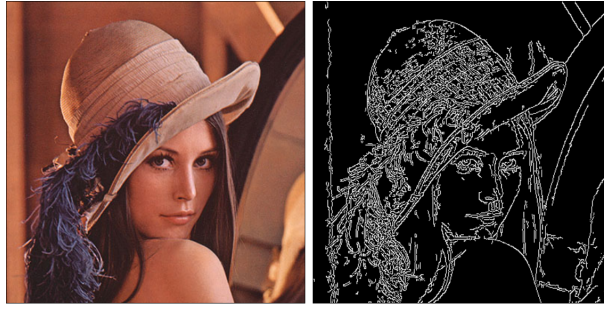
még valamilyen szerkesztéseket szeretnénk elvégezni a képen a megjelenítés előtt, melyeket a Glide alapértelmezetten nem támogat.

3.2. Vonalrajz generálás

A Line-us rajzoló robot vonalakat képes rajzolni egy papírlapra, a "karjára" rögzített írószer segítségével. A vonalakat G-code parancsok formájában tudja értelmezni, éppen ezért szükséges a képekből vonalrajzot generálni, melyeket könnyen G-code parancsokká lehet alakítani. A vonalrajz kinyerése érdekében néhány fontos műveletet kell végrehajtanunk a képen, ezek a következők: szürkeárnyalatosítás, éldetektálás, kontúrok kinyerése, szűrése és egyszerűsítése, majd bitinverzió alkalmazása, annak érdekében, hogy fehér alapon, fekete vonalakat kapjunk (8. ábra). Mindezek megvalósítására az **OpenCV** (Open Source Computer Vision Library)[8] nyílt forráskódú számítógépes látás szoftverkönyvtár osztályainak különböző függvényeit használjuk.

Az OpenCV több mint 2500 algoritmusával magas szintű interfészt kínál többek között a kép- vagy videóbemenetek feldolgozásához, és objektum vagy valamilyen képen megjelenő anomália detektálásához. A vonalrajz generálásához első sorban betöltjük a képet, amelyre az OpenCV *Imgcodecs*¹¹ nevű osztályának *imread* függvényét használjuk. Ez a metódus betölt egy képet a paraméterként megadott fájlból. A kép típusát a tartalma alapján határozza meg, és visszaadja azt Mat típusú objektumként. A Mat az OpenCV által használt alapvető adattípus a

¹¹<https://docs.opencv.org/4.x/javadoc/org/opencv/imgcodecs/Imgcodecs.html>, elérés dátuma: 2024-03-22



9. ábra. A képen látható az OpenCV Canny() metódusának eredménye, mely éleket detektál. Képforrás: https://docs.opencv.org/4.x/d7/de1/tutorial_js_canny.html, elérés dátuma: 2024-03-24

képek tárolására és kezelésére, mely tulajdonképpen egy n-dimenziós sűrű tömb.

Miután sikeresen betöltjük a képet egy Mat objektumba, következnek a szerkesztési folyamatok. A **Canny Edge Detection**[1], optimális detektorként is emlegetett, többlépcsős algoritmust használjuk a képen lévő élék észlelésére. Ezt az algoritmust John F. Canny fejlesztette ki 1986-ban azzal a céllal, hogy egy szigorúan meghatározott módszert adjon, amely jó és megbízható éldetektálást biztosít. Az algoritmus szürkeárnyalatos képekkel működik, és négy lépést tartalmaz: zajeltávolítás, intenzitás-gradiens kiszámítása, nem-maximumok eltávolítása, élék nyomon követése. A zajeltávolítás egy fontos lépés annak érdekében, hogy tiszta éleket kapjunk eredményül, ehhez a *Gaussz-szűrő*-t használják. A második lépésben ki kell számítani a vízszintes és függőleges gradienseket (G_x és G_y), majd a gradiens erősséget és az irányát a $G = \sqrt{(G_x^2 + G_y^2)}$, $\theta = \arctan(\frac{G_y}{G_x})$ képletekkel. A következőkben el kell távolítani azokat a felesleges képpontokat, melyek nem részei egy élnek sem. Végül a legfontosabb lépés az élék nyomon követése, ami egy küszöbparaméter segítségével valósul meg.

Mindezeket a folyamatokat az OpenCV egyetlen függvényben valósította meg, amely a *Canny()*¹². Mielőtt alkalmazzuk ezt a metódust a programunkban, átméretezzük a képeket, és az *Imgcodecs.cvtColor* függvény segítségével szürkeárnyalatosá alakítjuk őket. A *Canny* metódusnak megadjuk paraméterként a szürkeárnyalatos képet és egy Mat objektumot, az eredmény tárolásáért, valamint két küszöbértéket, melyek az élszűrés minőségét és mennyiségét befolyásolják. A metódus alkalmazása után az eredeti képünknek egy olyan változatát kapjuk, amely fekete alapon tartalmazza a kép éleit (lásd 9. ábra). Ez azonban nem elegendő nekünk, hiszen szükségünk van az egybefüggő kontúrokra. Tehát a következő feladatunk a kontúrok kinyerése az éldetektálás eredményéből. Az OpenCV erre is megoldást nyújt nekünk a

¹²https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html, elérés dátuma: 2024-03-22

*findContours()*¹³ függvényével. A *findContours* azokat a görbéket keresi, amelyek összekötik az azonos színű vagy intenzitású, összefüggő pontokat. Pontosabban fekete háttérből keresi ki a fehér színű élvonalakat. Esetünkben a *findContours* módszernek megadjuk paraméterként az élvonaldetektálás eredményét, valamint beállítjuk a kontúr visszakereső módot, hogy csak a külső kontúrokat kérje le, és az egyszerű közelítő módot, hogy a vízszintes, függőleges és átlós részeket tömörítse, és csak a végpontjaikat hagyja meg. Eredményül egy *MatOfPoint*¹⁴ típusú elemekből álló listát kapunk, mely tartalmazza a kontúrokat pontvektorként, valamint egy *Mat* objektumot, mely információkat tartalmaz a kép topológiájáról (lásd 1. kódrészlet).

Ezek után megsűrjűk a kapott kontúrokat a területeik alapján. Lekérjük minden egyes kontúr területét, és csak azokat tartjuk meg, melyek beleesnek az általunk meghatározott kontúrterület-korlátozó intervallumba. Ez azért szükséges, hogy csak a releváns körvonalakkal dolgozzunk tovább. Egyéb finomhangolást is végzünk a kontúrokkal, egyszerűsítjük őket azáltal, hogy egy rögzített pontossággal görbéket szerkesztünk hozzájuk. Ezek a görbék kevesebb csúcsot tartalmaznak, így velük dolgozunk a továbbiakban. Ezt a közelítést az *Imgproc approxPolyDP*¹⁵ függvényével valósítjuk meg. Ez csökkenti a feldolgozás során használt memória és erőforrás mennyiséget valamint gyorsabbá és hatékonyabbá teszi a vonalrajz generáló algoritmusunkat. A következőkben ezeket az egyszerűsített kontúrokat kell megrajzoljuk egy *Mat* típusú objektumba, ez a változó tartalmazza a körvonalaink véglegesített változatát. Erre az *Imgproc drawContours* nevű módszerét használjuk.

Az utolsó lépésekben bitenkénti inverziót hajtunk végre a véglegesített kontúrokat tartalmazó objektumon, annak érdekében, hogy fehér alapon kapjuk meg a kinyert

```
val contours: List<MatOfPoint> = ArrayList()
val hierarchy = Mat()
Imgproc.findContours(
    cannyOutput,
    contours,
    hierarchy,
    Imgproc.RETR_EXTERNAL,
    Imgproc.CHAIN_APPROX_SIMPLE
)
```

1. kódrészlet. Kontúrok kinyerése egy *Mat* objektumból.

¹³https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html, elérés dátuma: 2024-03-24

¹⁴<https://docs.opencv.org/4.x/javadoc/org/opencv/core/MatOfPoint.html>, elérés dátuma: 2024-03-24

¹⁵https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html, elérés dátuma: 2024-03-24

körvonalakat, fekete színben. Majd frissítjük az osztályszintű *contours* változónkat, melyben mindig az aktuálisan megjelenített vonalrajz kontúrjai vannak, pontvektor formájában. Utolsó lépésként egy bitmap-et hozunk létre az eredményül kapott vonalrajzból, hogy azt .jpeg formátumba le tudjuk menteni a felhasználó eszközére. Ha ki szeretné rajzoltatni a kigenerált vonalrajzot a felhasználó, lekérjük az elmentett körvonalakat és G-code parancsokat formálunk belőlük, majd elküldjük a robotnak.

Amennyiben a felhasználó módosítani szeretné a vonalrajzot a *Configure the Line Art* gomb segítségével, és új paramétereket állít be a felugró panelen, a vonalrajzot újra kell generálnunk a háttérben. A felhasználó által beállított paraméterek tulajdonképpen a *Canny* metódus küszöbértékei lesznek. A két érték közül a legkisebb az élek összekapcsolásáért felelős, míg a másik érték az erős élek kezdeteinek keresésére szolgál. Amint jóváhagyta a felhasználó, hogy új vonalrajzot kér a beállított paraméterekkel, végrehajtjuk a fentiekben leírt lépéseket az eredeti képen, figyelve arra, hogy a *Canny* függvényt ne az alapértelmezett küszöbértékekkel hívjuk meg, hanem a felhasználó által választottakkal. Az általunk rögzített alapértelmezett értékek a 25.0 és 75.0 küszöbszámok, ezeket találtuk a legmegfelelőbbnek a kísérletezések során.

3.3. Képek stilizálása

3.3.1. Rajzfilmesítés OpenCV segítségével

Ebben a fejezetben bemutatjuk, hogy hogyan lehet átalakítani az elkészített vagy galériából csatolt képeket az alkalmazásunk segítségével. Az 1. fejezetben vázolt képkivágáshoz, arcfelismeréshez, valamint a rajzfilmesítéshez többféle technológiát használunk. Felhasználjuk az **ML Kit**[2] arcfelismerő alkalmazásprogramozási felületét (API-ját) és az **Android Image Cropper**¹⁶ harmadik féltől származó könyvtárat, mely a képek kivágásában segít. A felhasznált technológiák sorát bővítik a **Generative Adversarial Network (GAN)**[4], amelyet a képek rajzfilmesítéséhez használunk fel, valamint a **pytorch LiteModuleLoader**¹⁷ osztálya, amely a GAN integrálását teszi lehetővé az applikációnkba.

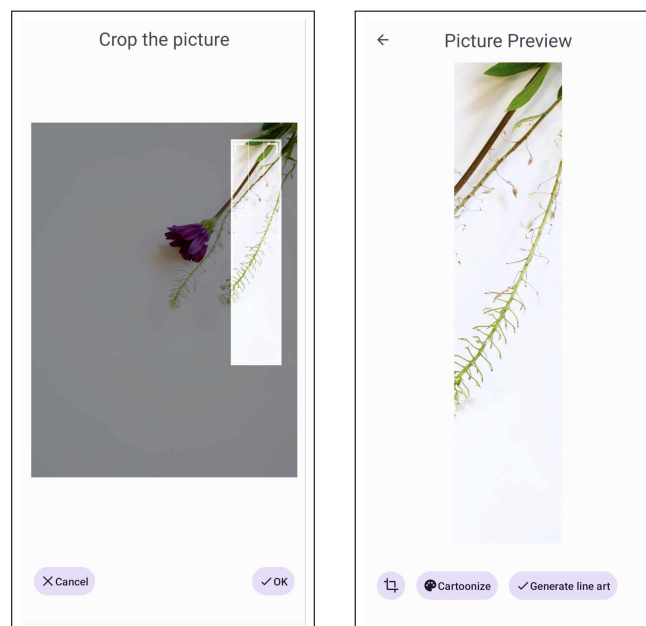
Az ML Kit egy szoftverfejlesztői készlet (SDK), amely hozzáférést nyújt a Google gépi tanulási modelljeihez. Pontosabban a mobiltelefonokra optimalizált modellek érhetőek el általa

¹⁶<https://github.com/CanHub/Android-Image-Cropper>, elérés dátuma 2024-04-25

¹⁷<https://pytorch.org/javadoc/1.9.0/org/pytorch/LiteModuleLoader.html>, elérés dátuma 2024-04-25

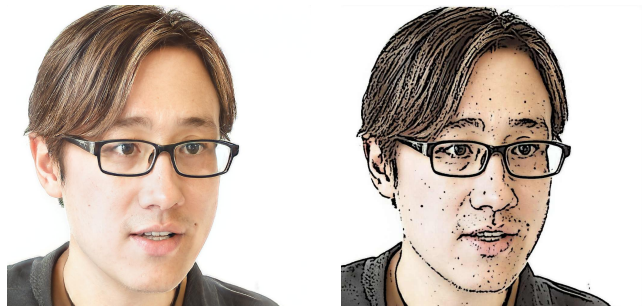
és ezek egyszerű integrálását is lehetővé teszi Android és iOS alkalmazásokba. Az ML Kit minden egyes alkalmazásprogramozási felülete a mobileszközön fut, ezáltal valós idejű feladatokat is meg tud oldani internetkapcsolat nélkül is. Az arcfelismerő API azonosítani tudja a képen megjelenő fontosabb arcvonásokat, megtalálja az arcok körvonalait, ezáltal fel tudja térképezni a képet. Nem ismeri fel az embereket, mivel csak észleli az arcot, viszont meg tudja határozni minden arc esetén a szemek, az orr, a fülek és a száj koordinátáit. Arc kifejezéseket is fel tud ismerni, meg tudja határozni például, hogy a képen megjelenő személy mosolyog, vagy csukva van a szeme. Az ML Kit olyan képeket vár bemenetként, amelyek mérete legalább 480x360 képpont, valamint azokat az arcokat képes felismerni, amelyek több mint 100x100 képponton találhatóak.

Ahogy a fejezet elején is említjük a képkivágást az *Android Image Cropper* harmadik féltől származó könyvtárral valósítjuk meg. Létrehozunk egy *CropImageView* objektumot, amely egy egyéni nézet és amelynek különböző beállításokat adunk meg a kép kivágására vonatkozóan. Elsőként megadjuk a vágásra szánt kép URL címét, engedélyezzük a *multitouch* funkcionalitást, valamint megadjuk a vágást jelző keret típusát. Ezek után már csak annyi a dolgunk, hogy meghívjuk ezeknek az objektumoknak a *getCroppedImage* függvényét, amely visszatéríti a kivágott képet Bitmap formátumba. A szerkesztett képet megjelenítés előtt átméretezzük a nézet objektumunknak megfelelően, így akárhogyan vágunk ki a képből egy részt, nem kapunk eltorzított eredményt. (lásd 10. ábra)



10. ábra. Az ábrán látható a képkivágás eredménye egy nem arányos vágás esetén.

A projektünk első verziójában a rajzfilmesítés lehetőségét az OpenCV függvénykönyvtár segítségével valósítottuk meg (11. ábra). A vonalrajz generálásához hasonlóan több függvényt használtunk fel, amelyek segítségével úgy alakítottuk a képet, hogy annak egy rajzfilmesített változatát kapjuk. Legelőször szürkeárnyalatossá tettük a képet, majd alkalmaztuk az *Imgproc.GaussianBlur*¹⁸ függvényt, mely segítségével csökkentettük a képen megjelenő zajokat. Ezeket az előfeldolgozásokat követte az *Imgproc.bilateralFilter*¹⁹, mely kétoldali szűrést alkalmaz a képre. Ez a szűrés valósította meg tulajdonképpen a rajzfilmhatást, mivel csökkentette a nem kívánt zajt és a kontúrokat meglehetősen élesre állította. A következő művelet, amelyet végrehajtottunk, az adaptív küszöbérték meghatározás, amelyet az *Imgproc.adaptiveThreshold*²⁰ függvény segítségével valósítottunk meg. Utolsó lépésekben visszakonvertáltuk a képet színessé, majd bitenkénti műveletet hajtottunk végre, hogy csak az adaptív küszöbérték-meghatározás eredményeként kapott fehér pixeleket tartsuk meg a színes képen, a többi rész halványuljon el. Ezt a módszert cseréljük ki a továbbiakban egy GAN modellre, hogy javítsunk a képek minőségén, és hogy jobban befolyásolni tudjuk a generált kép stílusát.



11. ábra. Az OpenCV segítségével megvalósított rajzfilmesítés.

3.3.2. Rajzfilmesítés GAN hálózatokkal

A generatív ellenséges hálózat (Generative Adversarial Network) vagy egyszerűen csak **GAN** egy gépi tanulási modell²¹. A GAN-ok jelentős figyelmet 2014 óta kapnak, Ian Goodfellow²² és munkatársai úttörő munkája után, melyet a *Generative Adversarial Nets*

¹⁸Elhomályosítja a képet Gauss-szűrő használatával.

¹⁹https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html, elérés dátuma 2024-04-25

²⁰https://docs.opencv.org/4.x/d7/dd0/tutorial_js_thresholding.html, elérés dátuma 2024-04-25

²¹A gépi tanulási modell egy számítógépes program, amely az adatminták felismerésére vagy előrejelzések készítésére szolgál és különböző gépi tanulási algoritmusokból áll.

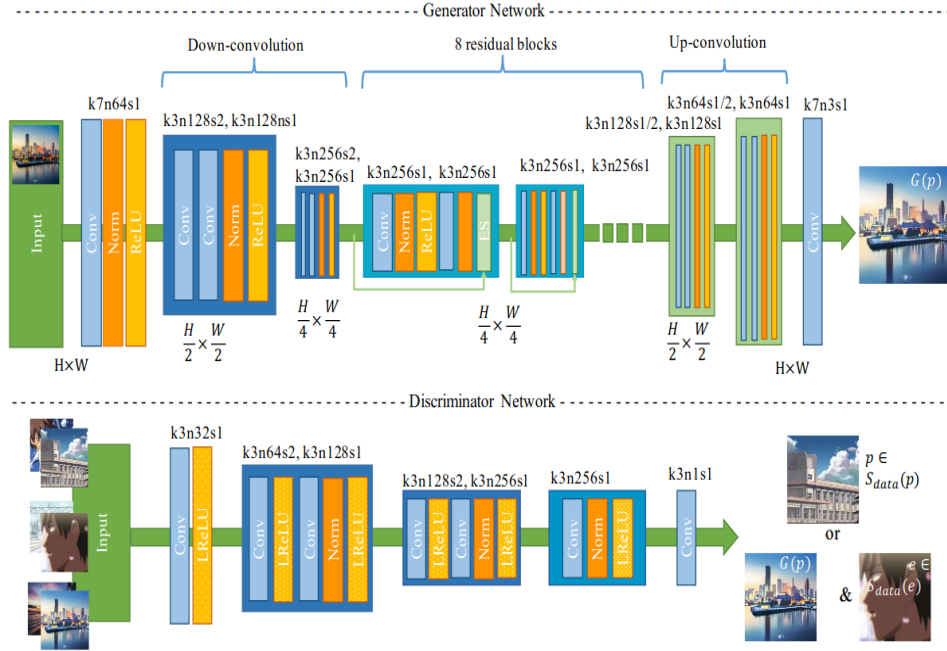
²²Ian Goodfellow mesterséges intelligencia kutató, kiemelkedően hozzájárul a mélytanulás és a generatív modellek területéhez.

[4] című cikkben mutatnak be. Ez a tanulási modell innovatív hatással van a képgenerálás és a kreatív AI alkalmazások világára. A GAN egy generatív modell, amely új adatok létrehozására hivatott. Számos területen alkalmazzák, például képgenerálásban (2D vagy akár 3D), stílusátvitelben vagy szöveggenerálásban stb. A legtöbb felhasználás a szórakoztatás céljából valósul meg, mint például a *ThisPersonDoesNotExist* oldal, mely újabb és újabb nem létező emberi arcokat generál.

A GAN modell két különálló neurális hálóból áll: egy generátorból és egy diszkriminátorból. A generátor feladata, hogy meggyőző adatokat állítson elő. Arra törekszik, hogy olyan valószínűségi eloszlást tanuljon meg, amely közelíti a bemeneti adatok eloszlását. A diszkriminátor feladata, hogy eldöntse egy bemenetről, hogy az a valós adathalmazból származik-e, vagy a generátor hozta létre. A tanítás során a generátor folyamatosan próbálja legyőzni a diszkriminátort, azáltal, hogy egyre valószínűbb hamisítványokat generál, míg a diszkriminátor azon dolgozik, hogy jobb nyomozóvá váljon, és helyesen osztályozza a valódi és hamis képeket. Ideális esetben idővel mindkét fél jól teljesít, ami meggyőző eredményeket produkál. Ez az ellenséges játszma és folyamatos versengés képezi a GAN-ok alapját.

Számos típusa van a GAN hálóknak, melyek mind különböző feladatokra és adattípusokra vannak optimalizálva. A rajzfilmesítés megvalósítására 2018-ban Yang Chen, Yu-Kun Lai és Yong-Jin Liu egy új generatív ellenséges hálózati keretrendszert javasol, a **CartoonGAN**-t [9]. Az eljárás során szükségünk van két adathalmazra a modell tanításához: az egyik az eredeti, valós képeket tartalmazza, míg a másik a rajzfilmesített, vagy valamilyen művész stílusát követő képeket. A módszer egyik erőssége, hogy az adatok nem kell párban legyenek, nem szükséges semmilyen megfeleltetés a két adathalmaz között. A CartoonGAN esetében a generátor hálózat feladata a képek rajzfilmesítése. A generátor hálózat egy konvolúciós réteggel kezdődik, amelyet két különálló lefele irányuló konvolúciós blokk (*down-convolution*) követ a képek összepréselése érdekében. Ebben a szakaszban nyerjük ki a képekből a lokális vonásaikat. Ezek után a generátor architektúrájában 8 blokk következik (*residual blocks*), melyek feladata a kép tartalmának összerakása. Végül az utolsó részek - a felfele irányuló konvolúciós rétegek (*up-convolution*) - és egy különálló konvolúciós réteg, a kép újjáépítéséért felelősek. Jelen esetben a diszkriminátor feladata, az, hogy eldöntse egy adott képről, hogy azt a generátor készítette-e, vagy az eredeti rajzfilmesített adathalmazból való. A stílus felismerése érdekében a diszkriminátor a kép lokális jellemzőit vizsgálja, melyeket a léptetett (*strided*) konvolúciós rétegek alkalmazása segítségével tud kódolni. Az utolsó részben egy építő blokk és egy 3x3-as

konvolúciós réteg található, melyek a diszkriminátor választ építik fel (12. ábra).



12. ábra. A generátor és diszkriminátor hálózatok felépítése a CartoonGAN esetében. k a kernel méretet, n a jellemzőterképek számát, s a lépést az egyes konvolúciós rétegekben, a *norm* a normalizációs réteget, az *ES* pedig a elemenkénti összeget jelöli. Képforrás: <https://gmarti.gitlab.io/deep-learning/2020/03/22/cartoongan-hk-landscapes.html>, elérés dátuma: 2024-04-05.

Az architektúrális felépítés mellett a CartoonGAN sajátossága a veszteség függvényekben és az inicializáló fázisban rejlik. A hibafüggvény két részből tevődik össze: ellenséges veszteség (*adversarial loss*) - $\mathcal{L}_{adv}$ és tartalmi veszteség (*content loss*) - $\mathcal{L}_{con}$. A végső hibafüggvény alakja a 3.3.2 képlettel számítható ki, ahol az ω súly segítségével tudjuk beállítani az arányt a stílus és a tartalommegőrzés között. A CartoonGAN szerzői a jó eredmény elérése érdekében a 10-es értéket ajánlják az ω -nak. Az $\mathcal{L}_{adv}$ függvényt a generátor és a diszkriminátor hálózatokra is alkamazzuk, célja az, hogy segítse a generátor tanulási folyamatát. A $\mathcal{L}_{con}$ függvény a bemeneti képek szemantikai tartalmának megőrzéséért felelős.

$$\mathcal{L} = \mathcal{L}_{adv} + \omega \mathcal{L}_{con} \quad (2)$$

A CartoonGAN modell konvergenciájának biztosítása érdekében a szerzők egy olyan inicializáló fázist ajánlanak, ahol a bemeneti képek tartalmát építjük fel. Így a generátor tanulási folyamata az inicializáló fázisban rekonstruált képekből indul ki, nem pedig véletlenszerű

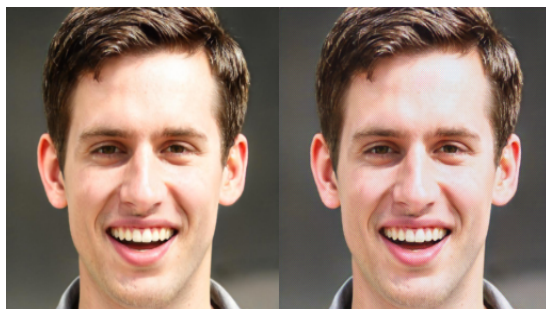
értékekből csökkentve a lokális minimumban ragadás esélyét.

Ezt a rajzfilmesítésre optimalizált CartoonGAN modellt valósítja meg Hyeonwoo Kang²³ a pytorch-CartoonGAN²⁴ nevű repójában. Az általa megírt programrészeket használjuk fel az applikációnkhoz. A programrészek Python nyelvben vannak megírva és a modell implementálásához a Pytorch²⁵ technológiát használják fel. A modell tanítására az adatokat a Kaggle²⁶ oldaláról gyűjtjük be.



13. ábra. Példa az általunk felhasznált adathalmaz elemeiről.

Szükségünk van valós és rajzfilmesített képekre is a modell betanításához. A *Comic Faces (paired, synthetic)* v2 néven elérhető adathalmazt használjuk, amely megtalálható a Kaggle oldalán. Párosított képeket tartalmaz, minden valós képhez megtalálható az adathalmazban egy hozzá tartozó rajzfilmesített változat is. A készlet 10000 darab, jpg formátumú képpárrt tartalmaz, a 13. ábrához hasonlóan.



14. ábra. Bal oldalon az eredeti kép, míg jobb oldalon az előtanítás eredménye 10 epoch után.

A CartoonGAN dokumentációnak megfelelően egy előtanítást, inicializálást végzünk az optimális hiperparaméter kezdőérték beállításához. Az előtanítás célja, hogy olyan paraméterbeállítást találjunk, mely esetén a generátor képes rekonstruálni a bemenetként kapott képeket. A kívánt eredményt 10 epoch után érjük el, amely a 14. ábrán látható. A tanulási

²³<https://github.com/znxlwm>, elérés dátuma 2024-04-25

²⁴<https://github.com/znxlwm/pytorch-CartoonGAN>, elérés dátuma 2024-04-25

²⁵<https://pytorch.org/>, elérés dátuma 2024-04-25

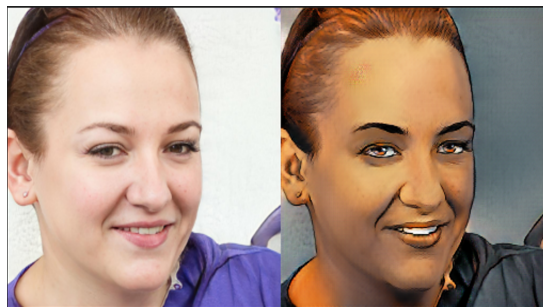
²⁶<https://www.kaggle.com/datasets>

folyamat néhány állapotát a 15. ábrán szemléltetjük, a véglegesnek nyilvánított eredményt a 300. epoch után érjük el, amely a 16. ábrán látható.



15. ábra. Az (a) képen a tanítás eredménye 100 epoch után. A (b) képen a tanítás eredménye 200 epoch után.

A 17. ábrán szemléltetjük a betanításhoz szükséges fontosabb számértékeket és összehasonlítjuk őket a CartoonGAN dokumentációban megjelenőekkel.



16. ábra. Bal oldalon az eredeti kép, míg jobb oldalon a tanítás eredménye 300 epoch után.

Mindezek után be kell építenünk az applikációnkba az elkészült modellt, amelyre a Pytorch `torch.jit.script` és `torch.utils.mobile_optimizer.optimize_for_mobile` függvényeit használjuk. A `torch.jit.script` lehetővé teszi a modell konvertálását TorchScript-be. A TorchScript egy olyan reprezentáció, amely segítségével a PyTorch modelleket futtatni tudjuk különböző környezetekben, vagyis a Pythonon kívüli környezetekben is. Ezt a lehetőséget kihasználva átalakítjuk a modellünket TorchScript-be. Az átalakítás után a `torch.utils.mobile_optimizer.optimize_for_mobile` függvényt használjuk, amely mobil eszközökön való futtatásra optimalizált modellt hoz létre, ezáltal kisebb, gyorsabb, és hatékonyabb lesz a modellünk. Ezt a változatot elmentjük .ptl formátumban, és felhasználjuk az alkalmazásban.

A konkrét rajzfilmesítési folyamat a felhasználó szemszögéből nem változik. Betöltjük a modellt a pytorch **LiteModuleLoader** osztályának segítségével. A LiteModuleLoader osztály

	CartoonGAN dokumentációban vázolt betanítás	Saját modellünk betanítása
adathalmazban lévő képek méretei	256 x 256	256 x 256
eredeti képekből álló adathalmaz mérete	6.153	10.000
rajzfilmesített képekből álló adathalmaz mérete	4.573 (Makoto Shinkai stílus)	10.000
előtanítás	10 epoch	10 epoch
tanítás	200 epoch	300 epoch

17. ábra. Párhuzamot vonunk a saját modellünk betanítása és a CartoonGAN dokumentációban leírt betanítási folyamat között.

két statikus load függvénnyel rendelkezik, melyből mi azt a verziót választjuk, ahol a függvény paramétereként a modell elérési útvonalát kéri. Átigazítjuk a kép méretét a modell által elvártra, majd átadjuk, és eredményül megkapjuk annak rajzfilmesített változatát. (18. ábra)



18. ábra. Az alkalmazásba beépített, CartoonGAN modell segítségével megvalósított rajzfilmesítés.

4. A fejlesztés módszertana és minősbiztosítás

Az alábbiakban a mobilalkalmazás-fejlesztésben alkalmazott eszközök és munkamódszerek kerülnek ismertetésre, illetve a kód és a mobilalkalmazás minőségének fenntartására használt eszközök, kiemelve funkciójukat és jelentőségüket a fejlesztési folyamat során,

4.1. Fejlesztési folyamatok

A verziókövetéshez használt eszköz a Git²⁷, amely segíti a kódmódosítások követését és a projekten párhuzamosan történő fejlesztést, illetve a fejlesztés során felmerülő konfliktusok megoldását. Funkcionalitásai megkönnyítik az együttműködést, a projekt összehangolását és menedzselését.

A GitLab²⁸ többek között kódellenőrzésre nyújt megfelelő felületet. Lehetőséget ad arra, hogy a csapattagok egymás kódját áttekinthessék és megjegyzéseket hagyjanak a kódsoroknál. Továbbá, könnyíti a feladatok követését, és több további funkcionálisással támogatja a fejlesztési folyamatot.

A csapat a Scrum munkamódszert választotta a projekten belüli hatékony együttműködésre. A Scrum munkamódszer jellemzői az iteratív fejlesztés és a különböző szerepkörök meghatározása: scrum master, product owner, development team. A munkamódszer szerint a csapat kéthetes Sprintekben dolgozott, amelyek egy tervezési fázissal kezdődtek. Ebben a fázisban a user story (felhasználói történetek) formában megfogalmazott követelmények prioritási sorrendben voltak válogatva, és így voltak kiválasztva a következő iterációban megoldandó feladatok.

A projekt feladatainak követését és rendszerezését a GitLab Kanban táblája segítette elő. A rendszeren belül a feladatoknak különböző állapotai lehetnek: open, to do, doing, review, merged, closed. Amikor egy feladat a Sprint kezdetekor kiválasztásra kerül, akkor az open listából (ahol a várakozó feladatok, azaz backlogban lévő feladatok vannak) átkerült a to do listába. Ha egy csapattag dolgozott egy feladaton, akkor a feladat az In Progress oszlopba került, elkerülve, hogy több csapattag ugyanazon a teendőn dolgozzon. A munka befejeztével a feladatok az In Review oszlopba kerültek át, ahol a kód végelgesítésére és a kód összefésülésére várakoztak. A már teljesített feladatok a Closed oszlopban jelentek meg.

A fejlesztés során a branchek kezelése a következő logika mentén történt: a main ág

²⁷<https://git-scm.com/>, elérés dátuma: 2024-04-30

²⁸<https://about.gitlab.com/>, elérés dátuma: 2024-04-30

tartalmazza a telepítésre alkalmas verziókat, míg a fejlesztési és hibajavítási munkálatok külön brancheken történnek. A branchek azonosítására a következő névkonvenció volt bevezetve: a branch nevében a két kezdeti betű jelzi a feature-on dolgozó személy nevét, amit követ a feature két-három szavas leírása.

4.2. Tesztek

Az alkalmazásunkhoz tesztek is társulnak, melyek segítségével biztosítani tudjuk az alkalmazás minőségét és stabilitását. A kifogástalan felhasználói élmény szolgáltatásának érdekében **UI** tesztek készítettünk, amelyek a felhasználói felületen megjelenő ikonok, gombok megjelenését és funkcionalitását ellenőrzik. Ezen kívül **unit** tesztek segítségével biztosítottuk, hogy az applikáció egyes részei rendeltetésszerűen működjenek.

A unit tesztek esetében lebontottuk a kódot kisebb funkcionális egységekre, és ezeket teszteltük külön-külön, kiragadva őket a környezetükből. Tesztelés során a függvények helyes működését ellenőriztük, minden lehetséges elágazáson. Mindezeket a *JUnit*, *Robolectric* és a *Mockk* technológiákkal oldottuk meg.

A JUnit egy tesztelési keretrendszer, amellyel automatizált tesztek íratunk. Egy megbízható környezetet szolgáltat a tesztek megírásához és végrehajtásához. Számos annotációt tartalmaz, melyek közül felhasználtuk a Before és After elő- és utófeltételekre vonatkozó annotációkat, a Test-et, mely jelzi a teszt törzsét, valamint a RunWith annotációt, mely segítségével meghívtuk a RobolectricTestRunner osztályt a tesztek futtatásához. A Robolectric keretrendszer segítségével a tesztek futtatása egy szimulált Android futási környezetben valósult meg. A Mockk keretrendszer segítségével hamis objektumokat hozunk létre, melyekkel helyettesítettük a valódi objektumokat, hogy a tesztelendő funkcionalitást egy elkülönített környezetben tudjuk megvizsgálni.

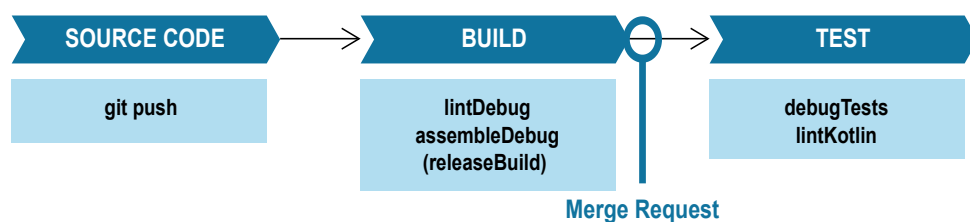
Az UI tesztek biztosítják a felhasználói felület helyes működésének ellenőrzését. Ezeknek a teszteknek a célja, hogy ellenőrizzük a felhasználói élmény és interfész minőségét, annak érdekében, hogy azok megfeleljenek a terveknek és a funkcionális követelményeknek. Alkalmazásunk teszteléséhez az Espresso keretrendszert használtuk, amely lehetővé teszi a tesztek automatizálását, ezzel csökkentve, és helyenként kiküszöbölve a manuális tesztelés szükségességét. Az Espresso integrálása az Android Studio-val nagyban egyszerűsítette a használatát. Projektünkben egy E2E teszt is megvalósult, amely a UI tesztek egyik típusa. Ekkor az egész felhasználói folyamat tesztelése történik úgy, hogy egy szimuláció készül a felhasználó

és az alkalmazás közötti interakcióról.

4.3. Pipeline-ok

Alkalmazásunkban **CI/CD**²⁹ (folyamatos integráció és folyamatos telepítés) szoftverfejlesztési folyamatokat használunk annak érdekében, hogy automatizáljunk különböző feladatokat, és biztosítsuk, hogy helyes kód kerül be a GitLab-on tárolt kódbázisba.

A **GitLab CI/CD**[3] szoftverfejlesztési eszközt használjuk applikációnkban a tesztek automatikus futtatására, valamint APK³⁰-k kinyerésére. Ennek érdekében egy `.gitlab-ci.yml` nevű fájlt hoztunk létre, amely tartalmazza a CI/CD munkafolyamatok konfigurációját, ezáltal meghatározza a pipeline-ok során végbemenő lépéseket. A fájl segítségével beállítjuk, hogy milyen szakaszai legyenek az ellenőrző pipeline-nak. Két szakaszt különítünk el: *build* és *test*. A build szakasz esetén több lépés (*job*) kell megvalósuljon. Az első ilyen *job* a kód stílusának és minőségének ellenőrzése, statikus kódelemző által. Ezek után az alkalmazás *debug* APK-jának felépítése következik, majd a *release* verzió összeállítása, amely csak minden Sprint forduló végén hajtódik végre. Minden lépés sikeresen végbe kell menjen ahhoz, hogy a pipeline a következő szakaszba lépjen. A fentebb említett `.yml` fájlban, úgy állítjuk be a *test* szakaszt, hogy kizárólag *merge request*-ek esetén hajtódjon végre. Két lépésből áll: unit tesztek futtatása és Kotlin kód stílusának ellenőrzése. (19. ábra)



19. ábra. Pipeline

Mindezek a lépések és feladatok biztosítják a kód minőségét és stabilitását. Egy kódváltoztatás akkor lesz elfogadott, ha a pipeline minden egyes szakasza sikeresen végrehajtotta feladatait. Ez azt eredményezni, hogy a GitLab-on tárolt kódok struktúrája és stílusa konzisztens lesz, valamint eleget tesz a megírt unit teszteknek.

²⁹Continuous Integration/Continuous Deployment

³⁰Android Package: Android applikációk csomagolási formátuma.

Következtetések és továbbfejlesztési lehetőségek

A Line-us projekt fejlesztése során sikerült egy olyan alkalmazást létrehozni, amely képes képek készítésére, módosítására és kirajzoltatására, ezzel elérve a projekt fő célkitűzéseit. Az alkalmazás egy felhasználóbarát felületet kínál azoknak, akik képeiket vonalrajzzá szeretnék alakítani, és ezeket a Line-us robot segítségével papírra szeretnék átvinni. Ezen kívül rajzfilmszerű stílusban történő képszerkesztési lehetőségekkel is bővítettük a funkciók körét.

Funkcionalitás szempontjából az alkalmazás több továbbfejlesztési lehetőséget is rejt magában. Egy ilyen lehetőség a karikatúragenerálás. A cél, hogy az arcokból, a Stable Diffusion segítségével, egy külső API-on keresztül karikatúrákat készíthessünk, amelyek kiemelik az arc karakterisztikus vonásait, miközben az ábrázolt személy még felismerhető marad. Ez nagyban hozzájárulna az alkalmazás szórakoztató jellegének növeléséhez.

Ugyanakkor, ha már a képmódosítási funkciókról beszélünk, szeretnénk tovább finomítani a GAN-alapú rajzfilmszerűsítést, jobb minőségű képek generálásának érdekében, amit további tréninggel és paraméterfinomítással érnénk el. Ezen kívül felmerült az ötlet, hogy a GAN modellt más adathalmazokon is tanítsuk, például egy népmesefigurák stílusában történő rajzfilmszerűsítésre. Ezzel növelhetnénk például a magyar népmesék népszerűségét is.

Annak érdekében, hogy a robot által készített rajzok még jobbak legyenek, és így a felhasználók számára érdekesebbek, szükséges még a vonalrajz generálásának fejlesztése, hogy a kontúrok élesebbé és könnyebben felismerhetővé váljanak, különösen a portrék esetében.

Ezek a fejlesztési lehetőségek bővítenék az alkalmazás funkcionalitását és javítanák a felhasználói élményt is. Általuk tovább növelhetjük az alkalmazás használhatóságát, megfelelően a felhasználói igényeknek és elvárásoknak.

Hivatkozások

- [1] Cuneyt Akinlar Cihan Topal. „Edge Drawing: A combined real-time edge and segment detector”. *Journal of Visual Communication and Image Representation* 23 (2012), 862–872. oldal.
- [2] Google for Developers. *Machine learning for mobile developers*. URL: <https://developers.google.com/ml-kit> (elérés dátuma: 2024. márc. 26.)
- [3] GitLab. *CI/CD*. URL: <https://about.gitlab.com/topics/ci-cd/> (elérés dátuma: 2024. ápr. 12.)
- [4] Ian Goodfellow és mások. „Generative Adversarial Nets”. *Advances in Neural Information Processing Systems*. Szerkesztette Z. Ghahramani és mások. 27. kötet. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.
- [5] Sam Judd. *Glide v4 Fast and efficient image loading for Android*. 2014. URL: <https://bumptech.github.io/glide/> (elérés dátuma: 2024. márc. 21.)
- [6] Microsoft. *Model-View-ViewModel - .NET*. 2022. URL: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm> (elérés dátuma: 2024. ápr. 18.)
- [7] Durrell Bishop és Robert Poll. *Download the Line-us App!* 2018. URL: <https://www.kickstarter.com/projects/line-us/line-us-the-little-robot-drawing-arm/posts/2102046> (elérés dátuma: 2024. márc. 12.)
- [8] OpenCV team. *OpenCV*. URL: <https://opencv.org/> (elérés dátuma: 2024. márc. 22.)
- [9] Yong-Jin Liu Yang Chen Yu-Kun Lai. „CartoonGAN: Generative Adversarial Networks for Photo Cartoonization”. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.