Sustav za generiranje i detekciju ArUco oznaka

Bošnjak, Filip

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Filip Bošnjak

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Filip Bošnjak

Zagreb, 2024.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem mentoru profesoru Tomislavu Stipančiću na neiscrpnoj podršci tijekom izrade završnog rada.

Zahvaljujem svima koji su mi pružali podršku tijekom studiranja.

Posebno zahvaljujem svojoj obitelji na neizmjernoj podršci i čeličnim živcima.

Filip Bošnjak

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Filip Bošnjak** JMBAG: **0035227497**

Naslov rada na hrvatskom jeziku: **Sustav za generiranje i detekciju aruco oznaka**

Naslov rada na engleskom jeziku: **System for generating and detecting aruco markers**

Opis zadatka:

U primjenama računalnog vida, procjena položaja je vrlo važan i težak zadatak. ArUco oznake su binarne kvadratne slike koje se koriste za procjenu položaja, kalibraciju kamere, navigaciju robota te primjenu proširene stvarnosti. Najčešće imaju crni rub, a unutarnji dio ima bijelu boju koja se mijenja ovisno o oznaci. ArUco rječnik odnosi se na predefinirani skup ArUco oznaka, gdje svaka oznaka ima jedinstveni identifikator kodiran unutar nje. Svaki rječnik specificira binarne uzorke koji čine različite ArUco oznake dozvoljene unutar tog specifičnog skupa.

U radu je potrebno:

- razviti robusni sustav temeljen na Python programskom jeziku te OpenCV biblioteci za stvaranje i detekciju ArUco oznaka koji će biti primijenjen na statičkim slikama
- odabrati odgovarajući ArUco rječnik uzevši u obzir broj jedinstvenih vrijednosti u rječniku, rezoluciju slika te udaljenost između korištenih oznaka.

U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan: 24. 4. 2024.

Zadatak zadao: izv. prof. dr. sc. Tomislav Stipančić

Datum predaje rada:
2. rok (izvanredni): 11. 7. 2024.
3. rok: 19. i 20. 9. 2024.

Predviđeni datumi obrane:
2. rok (izvanredni): 15. 7. 2024.
3. rok: 23. 9. – 27. 9. 2024.

Predsjednik Povjerenstva: prof. dr. sc. Damir Godec

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

Oznaka	Jedinica	Opis
AI	-	Umjetna inteligencija
CV	-	Računalni vid
GPS	-	Globalni sustav pozicioniranja
GUI	-	Grafičko korisničko sučelje
IDE	-	Integrirano razvojno okruženje
ML	-	Strojno učenje

SAŽETAK

Sustav za generiranje i detekciju ArUco oznaka temelji se na izradi i postavljanju projekta u integriranom razvojnom okruženju PyCharm, te programiranju aplikacije za učitavanje statičke slike, generiranje i detekciju ArUco oznaka koristeći programski jezik Python.

ArUco oznake su fiducijalne oznake koje sadrže vanjsku crnu granicu, omogućujući lakšu detekciju objekata prijelazom s pozadine slike na oznaku, i unutarnji binarni zapis koji sadrži identifikator. Tip rječnika ArUco oznake definiran je kombinacijom ukupne veličine i veličine rešetke oznake, te su time određene dimenzije granice i binarnog zapisa oznake.

Prilikom učitavanja statičke slike u program, potrebno je očitati rezoluciju slike te potom odrediti koji tip rječnika ArUco oznake je najprikladniji za primjenu i gdje ga postaviti, vodeći računa o preglednosti sadržaja slike i mogućnosti detekcije ArUco oznake na temelju udaljenosti oznake na slici, kako bi se izbjegao gubitak vidljivosti sadržaja oznake. Identifikator se dodjeljuje nasumično. Zatim se slika sprema u istu datoteku s novim sadržajem, odnosno dodanim ArUco oznakama na slici. Nakon toga, primjenjujemo detekciju ArUco oznaka na slici koja sadrži oznake. Program će se izvršiti čak i ako slika ne sadrži ArUco oznake, ali oznake neće biti detektirane.

Ključne riječi: programiranje, programski jezici, Python, prevođenje programa, integrirano razvojno okruženje, PyCharm, izrada projekta, postavljanje projekta, računalni vid, umjetna inteligencija, detekcija objekata, fiducijalne oznake, ArUco oznake, primjena ArUco oznaka na statičkim slikama, glavni program, učitavanje statičke slike, generiranje ArUco oznaka, detekcija ArUco oznaka, sustav za generiranje i detekciju ArUco oznaka

SUMMARY

The system for generating and detecting ArUco markers is based on the creation and setup of a project in the integrated development environment PyCharm, and the programming of an application for loading static image, generating and detecting ArUco markers using the Python programming language.

ArUco markers are fiducial markers that feature an outer black border, facilitating easier detection of objects by transitioning from the image background to the marker, and an internal binary code containing an identifier. The type of ArUco marker dictionary is defined by the combination of the overall size and the grid size of the marker, thereby determining the dimensions of the border and the binary code of the marker.

When loading a static image into the program, it is necessary to read the image resolution and then determine which type of ArUco marker dictionary is most suitable for application and where to place it, taking into account the clarity of the image content and the possibility of detecting the ArUco marker based on its distance in the image, to avoid losing the visibility of the marker's content. The identifier is assigned randomly. The image is then saved in the same file with the new content, i.e., the added ArUco markers on the image. After that, we apply ArUco marker detection on the image containing the markers. The program will execute even if the image does not contain ArUco markers, but the markers will not be detected.

Key words: programming, programming languages, Python, program translation, integrated development environment, PyCharm, project creation, project setup, computer vision, artificial intelligence, object detection, fiducial markers, ArUco markers, application of ArUco markers on static images, main program, loading static images, generating ArUco markers, detecting ArUco markers, system for generating and detecting ArUco markers

1. UVOD

Ideja za završni rad na temu sustava za generiranje i detekciju ArUco oznaka proizašla je iz potrebe za razvojem kvalitetnih udžbenika, priručnika i sličnih pomagala koji će omogućiti jednostavniju izradu različitih aplikacija na često korištenim modelima. Sustav za generiranje i detekciju ArUco oznaka pokazao se kao izvrstan model s mogućnošću nadogradnje, budući da je trenutno baziran na relativno jednostavnoj primjeni na statičkim slikama, ali sadrži mnoge komponente koje se mogu iskoristiti za izradu virtualnog okruženja i primjenu dijelova koda u manjim, sličnim ili većim projektima.

Kroz rad s minimalnom količinom teorije, nastojalo se integrirati model u kojem je teorijski dio osnova, a praktični dio slijedan toj osnovi. Ovaj rad, temeljen na primjeni ArUco oznaka na statičkim slikama, pruža sažet pregled teorije programiranja, programskih jezika i prevoditelja, koji je nadovezan kratkom teorijom o integriranim razvojnim okruženjima te primjenom istih u izradi projekta i postavljanju radnog okruženja. Prije početka programiranja, obrađuje se poglavlje o računalnom vidu kako bi čitatelj imao jasniji uvid u temu prije nego što se krene s praktičnom primjenom.

Ako čitatelj želi saznati više o određenim pojmovima, preporučuje se korištenje interneta, koji obiluje informacijama. Međutim, molimo da se obrati pažnja na izvore informacija kako bi se izbjeglo korištenje netočnih i nepotpunih podataka. U ovom radu prikazan je jedan od mogućih načina izrade zadatka, koji je pokušan biti izveden optimalno.

2. PROGRAMIRANJE

Programiranje je pisanje uputa računalu što i kako učiniti, a izvodi se u nekom od programskih jezika. Programiranje je umjetnost i umijeće u stvaranju programa za računala. Stvaranje programa sadrži u sebi pojedine elemente dizajna, umjetnosti, znanosti, matematike kao i inžinjeringa. [1]

Slika 2.1 Faze programiranja [2]

Slika 2.1 prikazuje faze programiranja. Faze programiranja sadrže definiranje i analizu problema što može podrazumijevati uporabu različitih alata poput provođenja intervjua, anketa, snimanje ili uslikavanje situacije kako se određeni proces trenutno izvršava. U početku je bitno postaviti dobre temelje to jest dobro proanalizirati problem i poopće definirati problem.

Tijekom faze razrade zadataka i planiranja naša početna analiza i definiranje problema počinju poprimati realne vrijednosti, što bi značilo da se razgledavaju i izrađuju moguća rješenja problema i u obzir se uzimaju moguće izvedbe i prepreke tih zahtjeva. No, nužno je ne ograničiti se već gotovim rješenjima.

Nakon što smo postavili okvir razrade zadatka i njegov plan izvedbe, započinjemo izradu algoritma. Ukratko, algoritam je definiran kao postepeni slijed kako riješiti problem. Postoje različiti načini kako izraditi algoritam, a neki od načina su s pomoću sekvencijalnog dijagrama

i s pomoću pseudokoda. Dobro izrađen algoritam je poput slagalice svaki sljedeći korak se nadopunjuje neprekidno bez dodatnih rupa ili preklapanja.

Ako algoritam nije moguće ostvariti kako kupac/naručitelj posla želi, potrebno je ponoviti proces iz početka. Ako je proces moguće izvesti, vrijeme je da to znanje „pretočimo“ u kod. Kodiranje, ili programiranje (kao što je opisano na početku 2. poglavlja), je srž navedenih faza. Programer ima jasno postavljen zadatak pred sobom i treba naći način kako da algoritam, koji je odredio na koji način i s pomoću kojeg programskog jezika (podrobnije o programski jezicima bit će opisano u 3. poglavlju) treba biti napisan, provede u stvarnost.

Nadalje, završetkom izrade programa (program je skup redaka kodova koji se izvršavaju i tvore smislenu cjelinu) potrebno je testirati kod. Testiranje koda znači provjeriti ispravnost koda i utvrditi postoje li greške prilikom rada programa. Kako je i prikazano na slici 2.1, greške mogu biti sintaksne i logičke. Sintaksne greške su greške koje naznačuju krivo napisanu liniju/redak koda, a logičke greške ukazuju na neispravno korištenje funkcija ili nesmislenost korištenih izraza. Programske funkcije i izrazi će podrobnije biti opisani u primjeni koda završnog zadatka.

Nakon što smo ustvrdili funkcionalnost koda, potrebno je napraviti tehničku i korisničku dokumentaciju. Tehnička dokumentacija opisuje detalje funkcionalnosti koda i zakonske regulative, a korisnička dokumentacija kako se rukuje navedenim kodom to jest kako ga upotrijebiti.

Na kraju, poželjno je održavati program kako ne bi došlo do neželjenih kretnji stroja i do mogućih neželjenih oštećenja.

3. PROGRAMSKI JEZICI

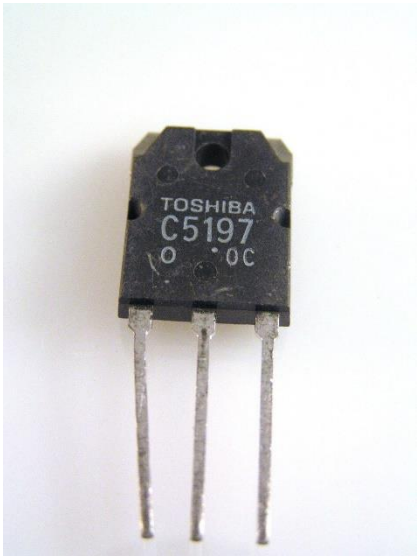
Programski jezici su alati, s pomoću kojih pišemo instrukcije, koje računalo prati. Računala „razmišljaju“ binarno u nulama i jedinicama. Programski jezici omogućuju da iste te nule i jedinice prevedemo u nešto što je nama ljudima razumljivo i što možemo zapisati. Programski jezici se sastoje od niza simbola koji služe kao most koji omogućuje ljudima da provedu svoju misli u instrukcije koje računalo razumije. [3]

Bazična podjela programskih jezika je na programske jezike niske i visoke razine.

3.1. Programski jezici niske razine

Programski jezici niske razine koriste binarni zapis koji stroj može razumjeti i izvršiti. Kada kažemo da stroj „razumije“, mislimo na njegovu logiku koja se izvršava putem digitalno-elektroničkih sklopova koji sadrže elemente poput tranzistora (vidjeti sliku 3.1).

Programski jezici niske razine nazivaju se i strojnim jezicima. Međutim, prilikom opće klasifikacije programskih jezika, u ovu kategoriju uključujemo i asemblerski jezik. Asemblerski jezik kombinira elemente jezika niske i visoke razine. Kao jezik niske razine, specifičan je za hardversku arhitekturu računala na kojem se pokreće. Kao jezik visoke razine, koristi simbole umjesto binarnog koda te zahtijeva prevoditelja.



Slika 3.1 Tranzistor proizvođača Toshiba [4]

Tranzistori (slika 3.1) su aktivni poluvodički elementi od kojih su izgrađeni mnogi elektronički sklopovi i integrirani krugovi. U analognim sklopovima koriste se za pojačanje električnog signala, a u digitalnim sklopovima kao sklopka. Glavni princip rada tranzistora je da manjom

ulaznom strujom ili naponom na upravljačkoj elektrodi upravlja većom izlaznom strujom. Razlikujemo dvije vrste tranzistora. Ako tranzistorom upravljamo pomoću struje, govorimo o bipolarnom tranzistoru, a ako upravljamo pomoću napona, onda govorimo o unipolarnom tranzistoru. [5]

3.2. Programski jezici visoke razine

Programski jezici visoke razine koriste složene izraze koje je potrebno prevesti u binarni zapis. Budući da se određene aplikacije mogu preklapati, u programiranju postoje takozvane biblioteke (engl. *libraries*). Biblioteke su skupovi unaprijed definiranih funkcija koje se mogu pozvati i koristiti u programu. Primjenu biblioteka i ostale nerazjašnjene segmente objasnit ćemo u poglavlju 7, koristeći Python, programski jezik visoke razine, u kontekstu računalnog vida.

3.2.1. Python

Python je jedan od najkorištenijih i najpopularnijih programskih jezika na svijetu. Snažan je, svestran i jednostavan za učenje. Python ima široku primjenu u raznim aplikacijama, uključujući:

- web razvoj,
- znanost o podacima,
- analizu podataka,
- strojno učenje,
- umjetnu inteligenciju,
- skriptiranje i alate.

Često se kaže da Python dolazi s „uključenim baterijama“, što znači da uključuje sveobuhvatnu standardnu biblioteku. Osim toga, zbog velike popularnosti Pythona, dostupne su stotine tisuća visokokvalitetnih biblioteka i okvira koji omogućuju brzo i jednostavno rješavanje različitih zadataka. [6]

Slika 3.2 Python, programski jezik visoke razine [7]

Python je objektno orijentirani programski jezik visoke razine koji koristi automatsko upravljanje memorijom, za razliku od drugih programskih jezika visoke razine poput C++-a. Automatsko upravljanje memorijom omogućuje brži razvoj aplikacija, ali može usporiti izvršenje programa i bespotrebno opteretiti memoriju.

3.2.2. C++

C++ je generički programski jezik namijenjen razvoju softvera. C++ je objektno orijentirani jezik, što znači da naglašava korištenje podatkovnih struktura s jedinstvenim atributima, poznatim kao objektima, umjesto logike ili funkcija. Primjer objekta je korisnički račun na web stranici, koji obično sadrži povezane podatke kao što su ime, prezime i adresa e-pošte. Grupiranje tih informacija u objekt olakšava repliciranje procesa stvaranja novog računa. [8]

Bjarne Stroustrup razvio je C++ kao proširenje programskog jezika C. Ključna razlika između C-a i C++-a je u tome što C++ podržava klase i objekte, dok C ne podržava.

Slika 3.3 C++, programski jezik visoke razine [9]

3.2.3. Java

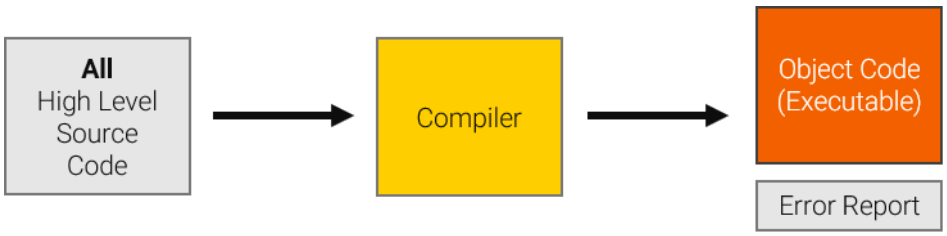
Sun Microsystems je 1995. godine objavio Javu, programski jezik i računalnu platformu. Od svojih skromnih početaka, Java je izrasla u ključnu tehnologiju koja pokreće značajan dio današnjeg digitalnog svijeta, pružajući pouzdanu platformu za mnoge usluge i aplikacije. Java se i dalje koristi za razvoj novih, inovativnih proizvoda i digitalnih usluga budućnosti. [10]

Slika 3.4 Java, programski jezik visoke razine [11]

4. PREVOĐENJE PROGRAMA U BINARNI OBLIK

Kada napišemo program u nekom od programskih jezika visoke razine ili u asemblerskom jeziku, potrebno ga je prevesti u strojni jezik kako bi računalo moglo izvršiti određene fizičke ili digitalne radnje. Postoje tri vrste prevoditelja: kompajleri, interpreteri i asembleri.

4.1. Kompajleri

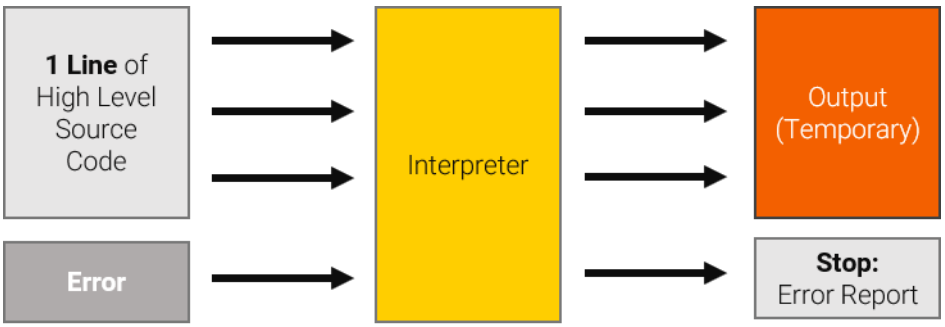


Slika 4.1 Princip rada kompajlera [12]

Kompajleri (vidjeti sliku 4.1) se koriste za prevođenje programa napisanih u programskom jeziku visoke razine u strojni jezik (objektni kod). Nakon kompajliranja (u jednom koraku), prevedena programska datoteka može se izravno koristiti na računalu i samostalno je izvršna. Iako kompajliranje može potrajati, prevedeni program može se koristiti više puta bez potrebe za ponovnim kompajliranjem.

Izvješće o pogreškama obično se generira nakon što je cijeli program preveden. Pogreške u programskom kodu mogu uzrokovati pad sustava. Te se pogreške mogu ispraviti samo izmjenom izvornog koda i ponovnim kompajliranjem programa. [12]

4.2. Interpreteri



Slika 4.2 Princip rada interpretera [12]

Interpreteri (vidjeti sliku 4.2) mogu čitati, prevoditi i izvršavati kod programskog jezika visoke razine liniju po liniju. Zaustavljaju se odmah kada naiđu na grešku u kodu. Često se koriste u razvoju programa jer provjeravaju kod liniju po liniju prije izvršavanja, što olakšava ispravljanje pogrešaka.

Interpretirani programi pokreću se odmah, ali mogu raditi sporije od kompajliranih datoteka. Ne stvara se izvršna datoteka; program se interpretira iz početka svaki put kada ga pokrenete. [12]

4.3. Asembleri

Asembleri se koriste za prevođenje asemblerskih jezika, koji su programski jezici niske razine, u strojni jezik (objektni kod) koji računalo može izvršiti. Nakon prevođenja, programska datoteka može se ponovno koristiti bez ponovnog prevođenja. [12]

5. INTEGRIRANO RAZVOJNO OKRUŽENJE

Integrirano razvojno okruženje IDE (engl. *integrated development environment*) je softver koji kombinira često korištene razvojne alate u kompaktno grafičko korisničko sučelje GUI (engl. *graphical user interface*). IDE uključuje alate kao što su uređivač koda, kompajler i program za ispravljanje pogrešaka koda s integriranim terminalom.

Integriranjem značajka kao što su uređivanje, izrada, testiranje i pakiranje softvera u alat jednostavan za korištenje, IDE pomaže u povećanju produktivnosti razvojnih programera. Programeri i softverski inženjeri često koriste IDE kako bi olakšali i ubrzali svoj razvojni proces. [13]

Detaljno ćemo predstaviti integrirano razvojno okruženje kroz izradu završnog rada. Za izradu završnog rada korišteno je integrirano razvojno okruženje PyCharm koje je primarno namijenjeno programskom jeziku Python. Neka od poznatijih integriranih razvojnih okruženja su: Android Studio, Arduino IDE, Eclipse, IntelliJ IDEA, NetBeans, PhpStorm, RubyMine, Visual Studio, WebStorm i Xcode. Iako neka od navedenih okruženja nude podršku za Python, većina nije primarno namijenjena za taj programski jezik.

5.1. PyCharm Community Edition 2024.1

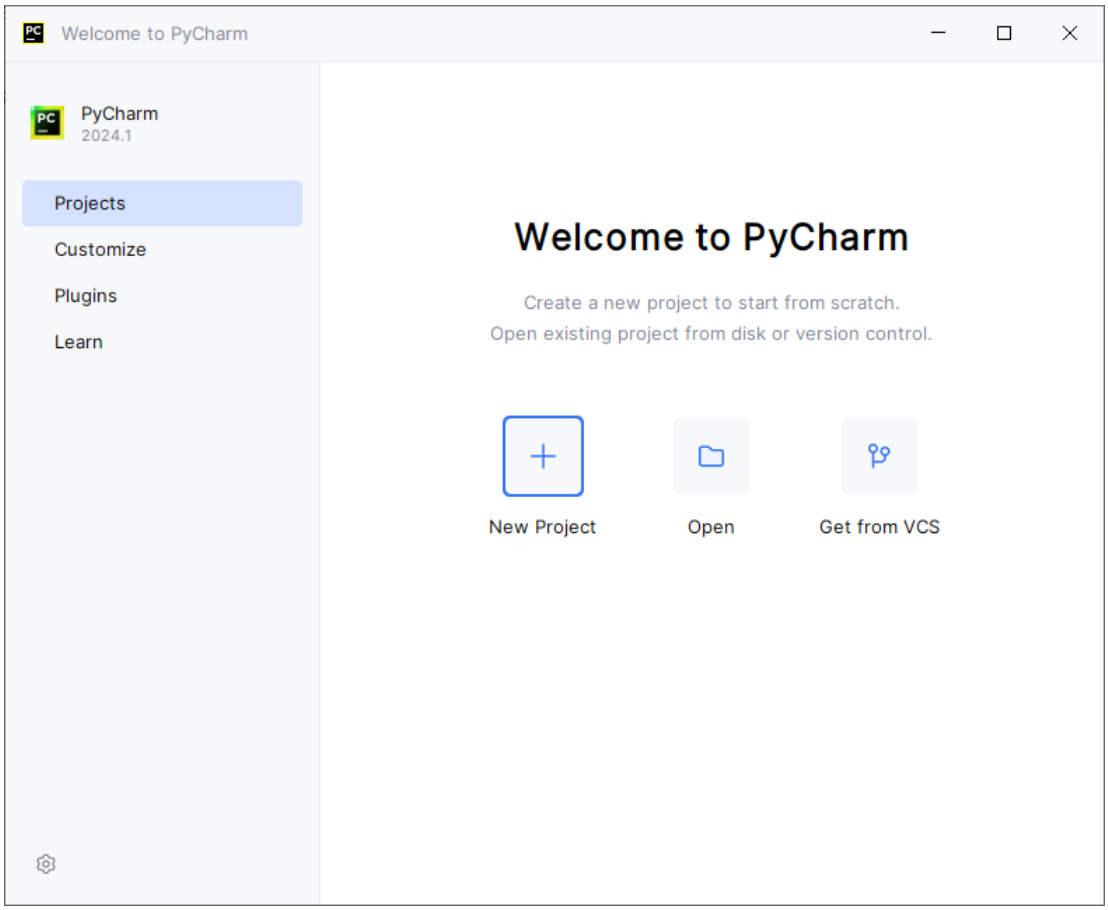
Slika 5.1 PyCharm Community Edition 2024.1, integrirano razvojno okruženje (IDE)

Slika 5.1 prikazuje integrirano razvojno okruženje PyCharm koje je jedno od najkorištenijih za izradu programa u programskom jeziku Python.

PyCharm je poznati IDE za poboljšanje Pythona jer pruža širok raspon mogućnosti koje pomažu programerima da učinkovitije uređuju, otklanjaju pogreške i testiraju svoj kod. Neke od važnih mogućnosti PyCharma sastoje se od:

- **Pametnog dovršavanja koda:** PyCharm predstavlja pametan kod u cjelini koji vam omogućuje brže i ispravnije uređivanje koda. Može ponuditi ključne riječi, varijable, značajke i druge elemente koda koji se prvenstveno temelje na kontekstu vašeg koda.
- **Inspekcije koda:** PyCharm može pregledati vaš kod za pogreške i probleme s kapacitetom. Može uočiti sintaksne pogreške kapaciteta, tipske pogreške i različite logičke pogreške.
- **Otklanjanja pogrešaka:** PyCharm pruža učinkovit program za ispravljanje pogrešaka koji će vam omogućiti da prođete kroz liniju po liniju koda, ispitate vrijednosti varijabli i postavite prijelomne točke.
- **Testiranja:** PyCharm uključuje poznate Python okvire za testiranje, uključujući unittest i pytest. To vam omogućuje da odmah pokrenete svoje testove iz IDE-a.
- **Kontrole verzije:** PyCharm uključuje popularne strukture za upravljanje modelima, uključujući Git i Mercurial. To vam omogućuje da sačuvate svoje izmjene na svom kodu i surađujete s drugim graditeljima koda. [14]

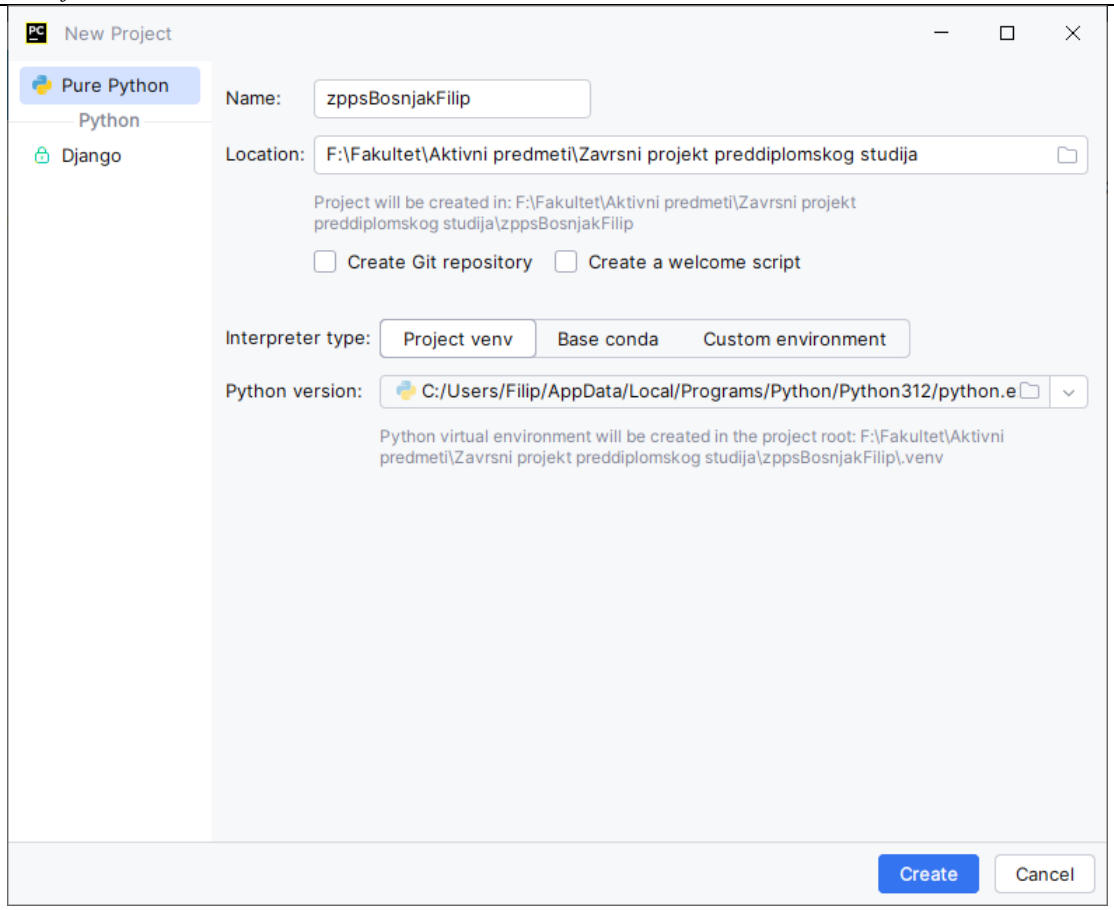
5.1.1. Izrada projekta



Slika 5.2 Izrada novog projekta 1. korak, PyCharm IDE

Nakon što smo uspješno preuzeli i instalirali softversko rješenje PyCharm, prateći upute za instalaciju (engl. *installation guide*) sa službene stranice proizvođača JetBrains, [15] pokrećemo program. Prvo ugledamo sliku 5.1, a nakon učitavanja početnog zaslona odmah se prikazuje zaslon sa slike 5.2.

Kao što je naznačeno na slici 5.2, odaberemo karticu *Projects*, a zatim odaberemo *New Project* kako bismo stvorili novi projekt. Za izradu ovog projekta nije nužno otvarati kartice *Customize*, *Plugins* i *Learn*, no preporučuje se pregledati. Isto tako, važno je napomenuti da se određene stvari postavljanja projekta mogu napraviti na više načina i ponekad ne postoji optimalan način.



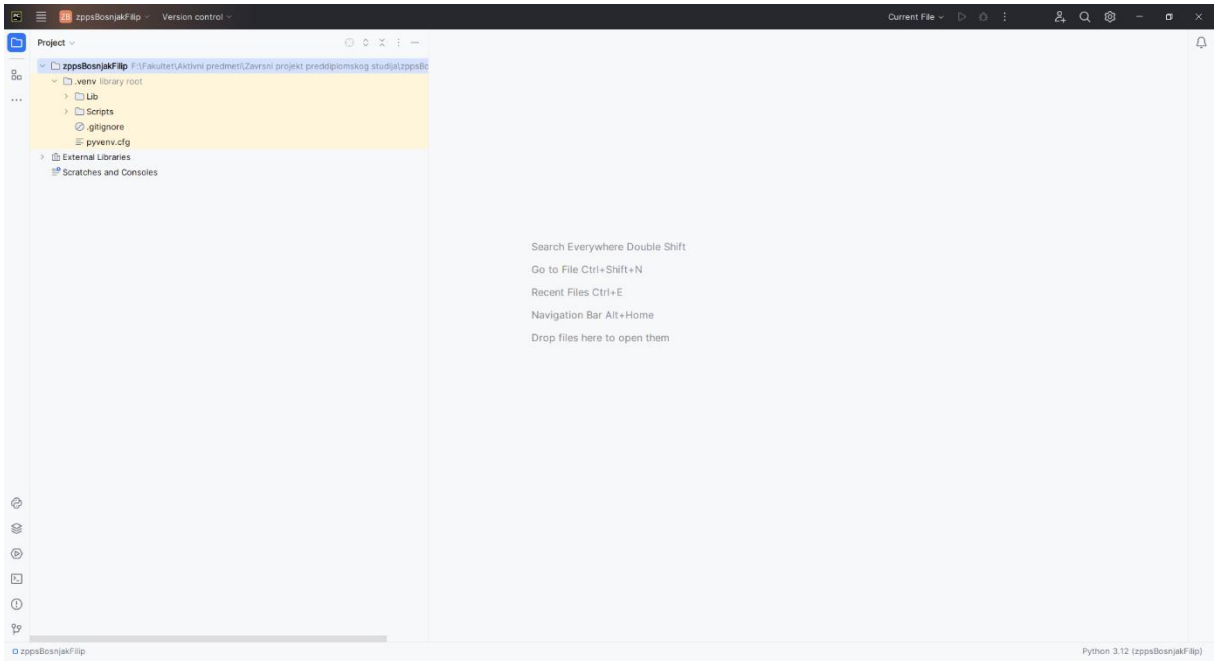
Slika 5.3 Izrada novog projekta 2. korak, PyCharm IDE

Nakon što smo pritisnuli gumb *New Project* za stvaranje novog projekta, potrebno je unijeti još par podataka o samom projektu. Sa slike 5.3 vidljivo je da su to:

- ***Name*** (ime projekta)**:** kako želimo da se naš projekt naziva.
- ***Location*** (lokacija)**:** gdje želimo da se naš projekt nalazi na računalu.
- ***Create Git repository*** (izrada Git repozitorija)**:** odabirom ove kućice, program izrađuje online verziju našeg koda kojoj mogu pristupiti svi korisnici na internetu, također je moguće dodijeliti dozvole za uređivanje koda i mnoge druge funkcionalnosti.
- ***Create a welcome script*** (izrada skripte dobrodošlice)**:** ovo je skripta koja ukratko objašnjava neke od osnovna korištenja programa, ako odaberemo ovu kućicu ta će se skripta stvoriti.
- ***Interpreter type*** (tip interpretera)**:** koji tip interpretera želimo odabrati.
- ***Python version*** (inačica Pythona)**:** koju inačicu Pythona želimo koristiti.

Od navedenih podataka, važno je odabrati ime, lokaciju i provjeriti verziju Pythona kao na slici 5.3. U lijevom izborniku (na slici 5.3) postoji opcija odabira Django projekta, no Django se koristi za izradu web stranica i trenutno nam nije relevantan.

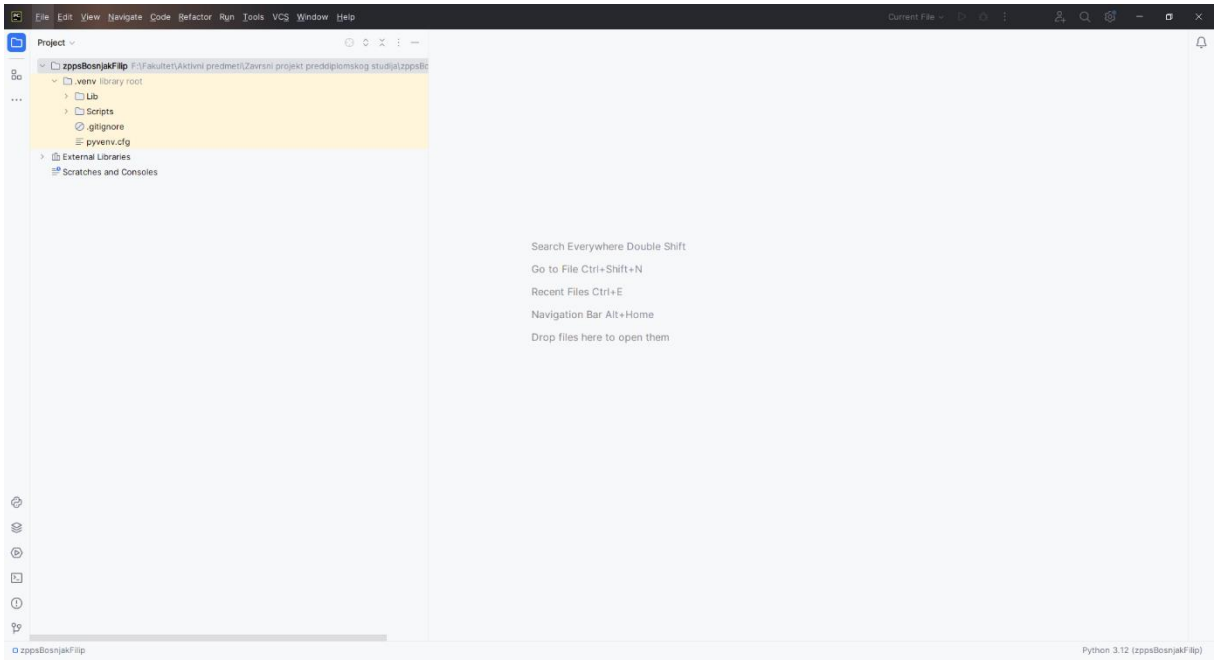
Kada smo unijeli potrebne podatke, pritisnemo tipku *Create*.



Slika 5.4 Izrađeni novi projekt, PyCharm IDE

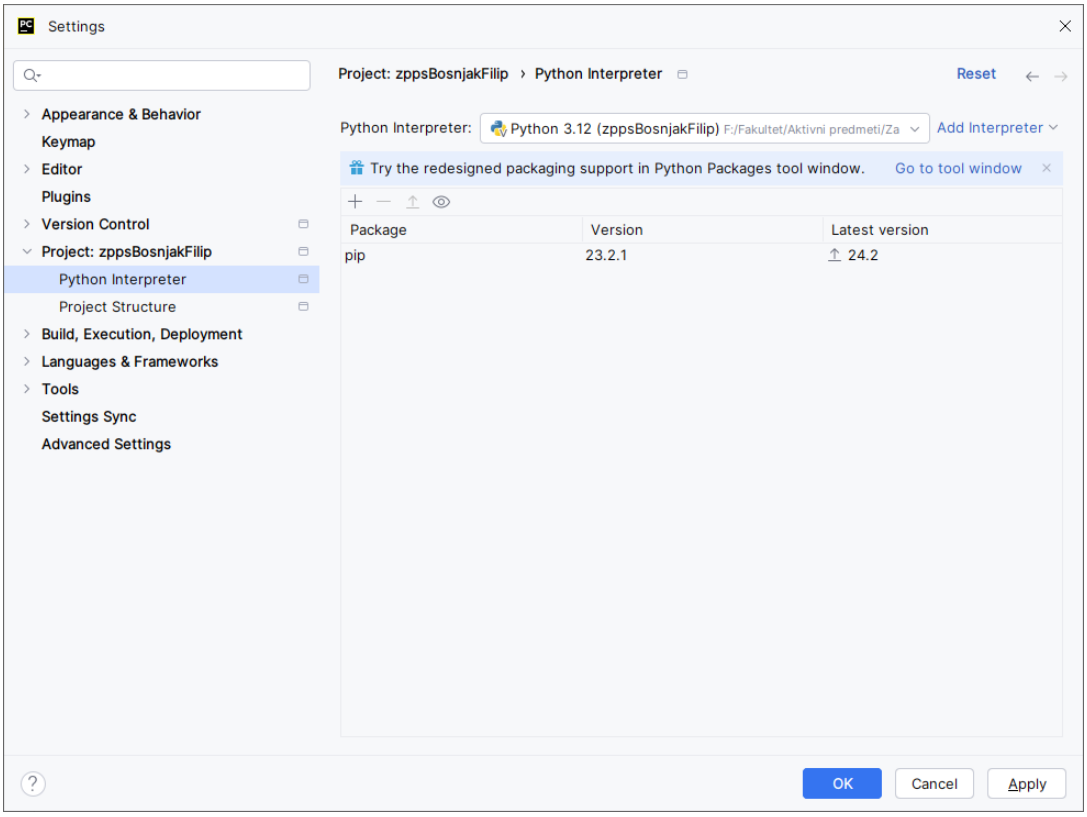
Nakon pritiska na tipku *Create*, otvara se novi prozor (vidjeti sliku 5.4) s prikazom novog projekta. Nadalje, potrebno je postaviti projekt za izvršavanje i testiranje koda.

5.1.2. Postavljanje projekta



Slika 5.5 Postavljanje projekta 1. korak, PyCharm IDE

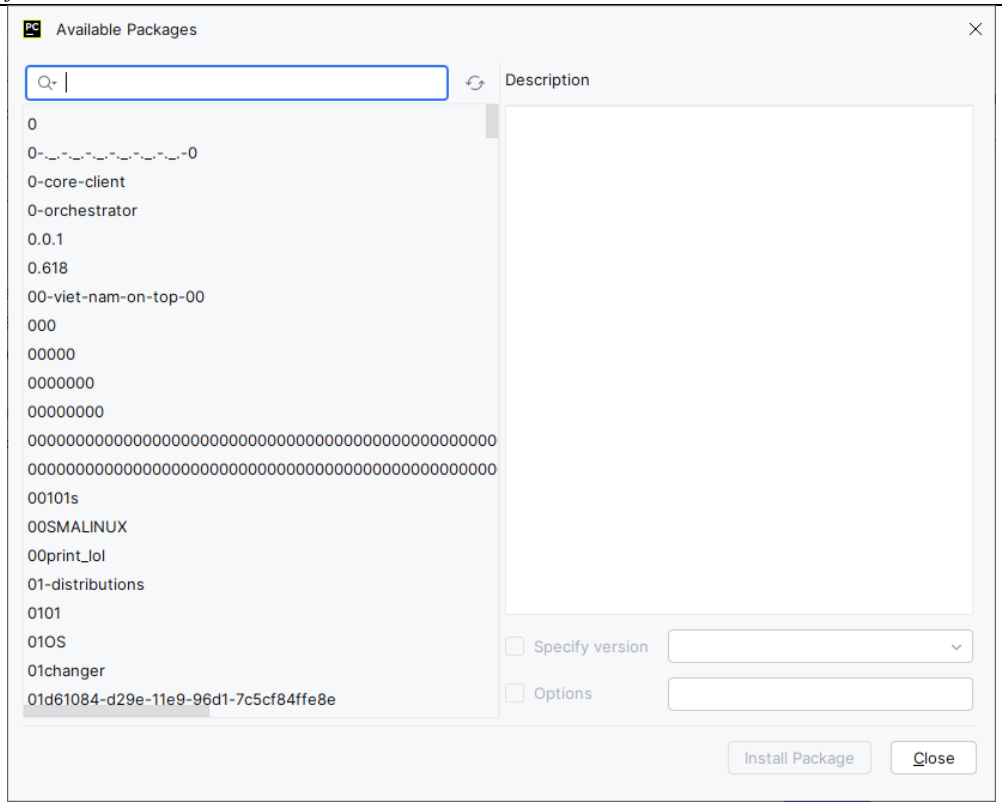
Pritiskom na ikonu s četiri crtice u gornjem lijevom kutu (vidjeti sliku 5.4), otvara se izbornik s raznim opcijama (vidjeti sliku 5.5). Za postavljanje projekta, odaberite opciju *File*. Zatim, u padajućem izborniku, odaberite opciju *Settings*.



Slika 5.6 Postavljanje projekta 2. korak, PyCharm IDE

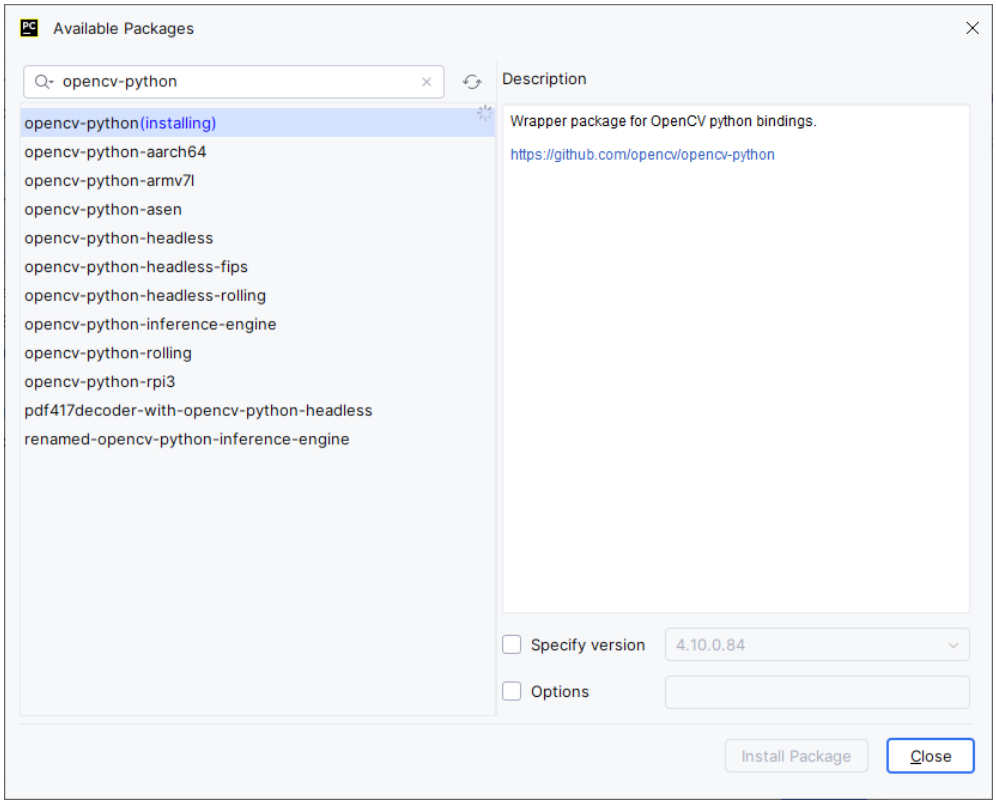
Nakon odabira opcije *Settings*, pojavit će se prozor prikazan na slici 5.6. Program inicijalno prikazuje opciju *Appearance & Behavior* u lijevom izborniku. Međutim, potrebno je odabrati opciju *Project: zppsBosnjakFilip* (ili *Project: odabranoImeProjekta*) i zatim podopciju *Python Interpreter* (vidjeti sliku 5.6).

Python interpreter se automatski postavlja na početku izrade projekta. Ovdje želimo dodati biblioteke koje će naš projekt koristiti. U tablici, prikazanoj na slici 5.6, nalaze se polja *Package*, *Version* i *Latest Version*. Iznad polja *Package* nalazi se znak plus koji ćemo odabrati kako bismo dodali preostale biblioteke potrebne za izradu ovog projekta. Dodatno, primijetimo da postoji paket nazvan pip. Taj paket je ponekad već instaliran i može se koristiti za instaliranje drugih paketa ili biblioteka putem terminala.



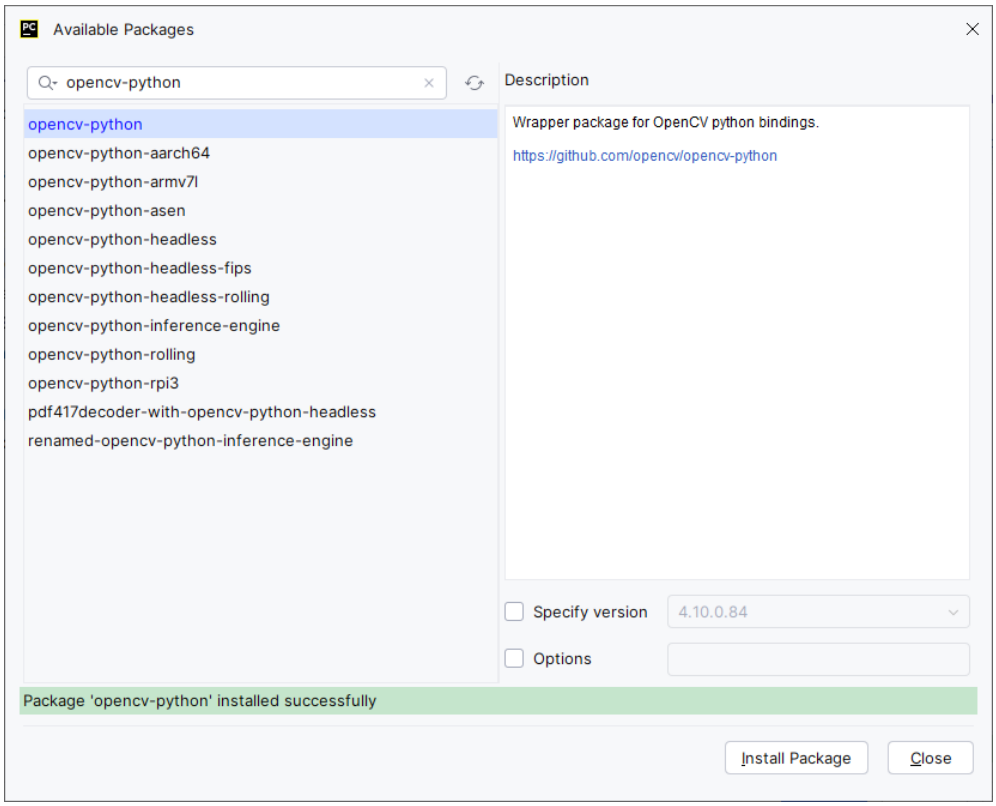
Slika 5.7 Postavljanje projekta 3. korak, PyCharm IDE

Nadalje, pojavit će se prozor u kojem je potrebno unijeti naziv željene biblioteke u gornji lijevi okvir za unos teksta (označen plavim rubom; kao što je vidljivo na slici 5.7).



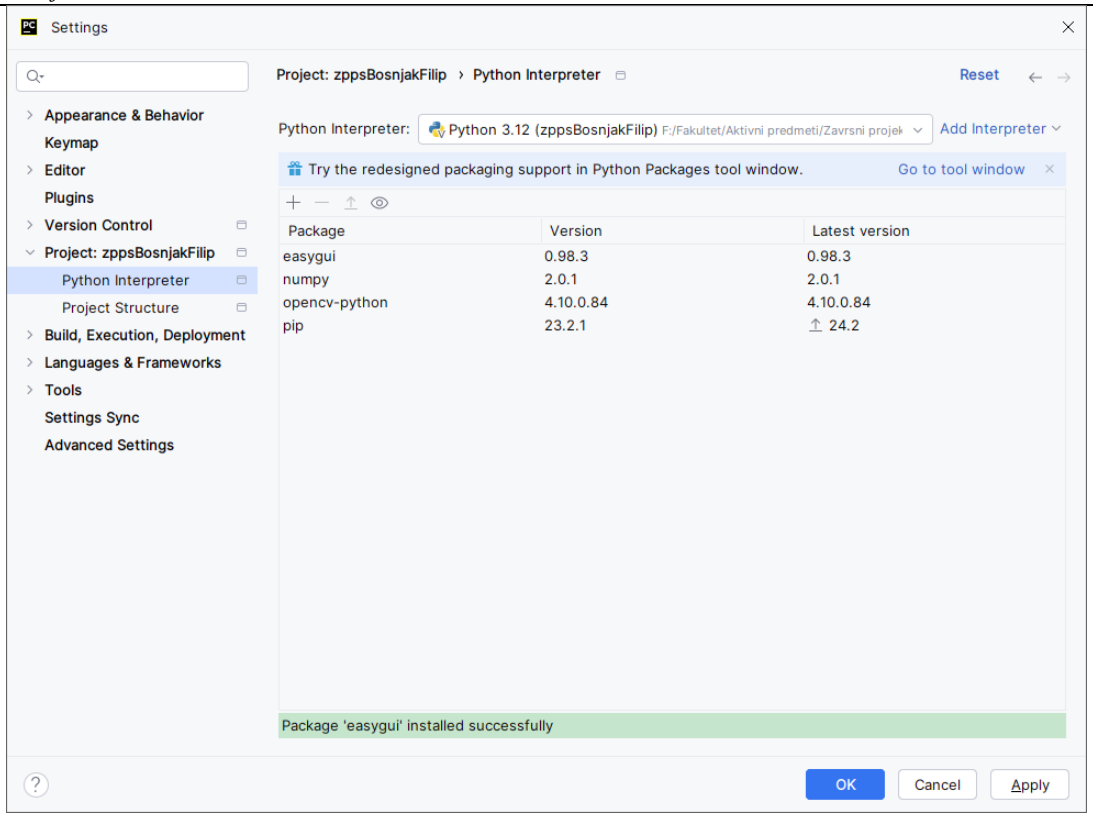
Slika 5.8 Postavljanje projekta 4. korak, PyCharm IDE

Za zadani projekt želimo instalirati biblioteke *opencv-python*, *numpy* i *easygui*. Da bismo to učinili, potrebno je unijeti naziv biblioteke (kako je opisano u prethodnom koraku). Program će prikazati popis mogućih biblioteka s istim ili sličnim nazivom. Zatim je potrebno odabrati željenu biblioteku i pritisnuti gumb *Install Package*. Nakon pritiska na taj gumb, zaslon će izgledati kao na slici 5.8, što znači da se biblioteka instalira.



Slika 5.9 Postavljanje projekta 5. korak, PyCharm IDE

Nakon instalacije biblioteke, zaslon će izgledati kao na slici 5.9. Ovaj postupak je potrebno ponoviti za sve preostale biblioteke. Detaljan opis korištenih biblioteka bit će dan u 7. poglavlju gdje ćemo opisivati kod za generiranje i detekciju ArUco oznaka.



Slika 5.10 Postavljeni projekt, PyCharm IDE

Kada smo instalirali biblioteke, prozor bi trebao izgledati kao na slici 5.10. Ako je sve u redu, možemo pritisnuti gumb *OK*, koji se nalazi u donjem desnom kutu i označen je plavom bojom na slici 5.10.

Prije negoli započnemo s programiranjem, u 6. poglavlju nalazi se pregled teme računalnog vida kako bismo podrobnije razumjeli što točno programiramo u zadanom završnom zadatku.

6. RAČUNALNI VID

Računalni vid (engl. *computer vision*; skraćeno CV) je područje umjetne inteligencije koje koristi strojno učenje i neuronske mreže za obučavanje računala i sustava da izvlače smislene informacije iz digitalnih slika, videozapisa i drugih vizualnih ulaza. Ovi sustavi mogu davati preporuke ili poduzimati radnje kada uoče nedostatke ili probleme. [16]

Umjetna inteligencija (engl. *artificial intelligence*; skraćeno AI) je tehnologija koja omogućuje računalima i strojevima simulaciju ljudske inteligencije i sposobnosti rješavanja problema. Samostalno ili u kombinaciji s drugim tehnologijama (npr. senzorima, geolokacijom, robotikom), AI može obavljati zadatke koji bi inače zahtijevali ljudsku inteligenciju ili intervenciju. Digitalni asistenti, GPS navođenje, autonomna vozila i generativni AI alati (poput OpenAI-jevog ChatGPT-a) samo su neki od primjera umjetne inteligencije u svakodnevnim vijestima i našem svakodnevnom životu. [17]

Ako umjetna inteligencija omogućuje računalima da razmišljaju, računalni vid im omogućuje da vide, promatraju i razumiju. Računalni vid funkcionira slično kao i ljudski vid, osim što ljudi imaju prednost životnog iskustva. Ljudski vid koristi životno iskustvo za razlikovanje predmeta, procjenu udaljenosti, prepoznavanje kretanja i uočavanje nepravilnosti na slici.

Računalni vid obučava strojeve za obavljanje ovih funkcija, ali to mora učiniti u mnogo kraćem vremenu koristeći kamere, podatke i algoritme, umjesto mrežnica, vidnih živaca i vidnog korteksa. Sustav obučen za pregled proizvoda ili nadzor proizvodnih procesa može analizirati tisuće proizvoda ili procesa u minuti, uočavajući neprimjetne nedostatke ili probleme, čime brzo nadmašuje ljudske sposobnosti. [16]

Strojno učenje (engl. *machine learning*; skraćeno ML) je grana umjetne inteligencije i računalne znanosti koja se fokusira na korištenje podataka i algoritama kako bi se umjetnoj inteligenciji omogućilo da oponaša način na koji ljudi uče, postupno poboljšavajući njezinu točnost. [18]

Neuronska mreža (engl. *neural network*) je program strojnog učenja, ili model, koji donosi odluke na način sličan ljudskom mozgu, koristeći procese koji oponašaju način na koji biološki neuroni rade zajedno kako bi identificirali pojave, odvagnuli mogućnosti i došli do zaključaka. [19]

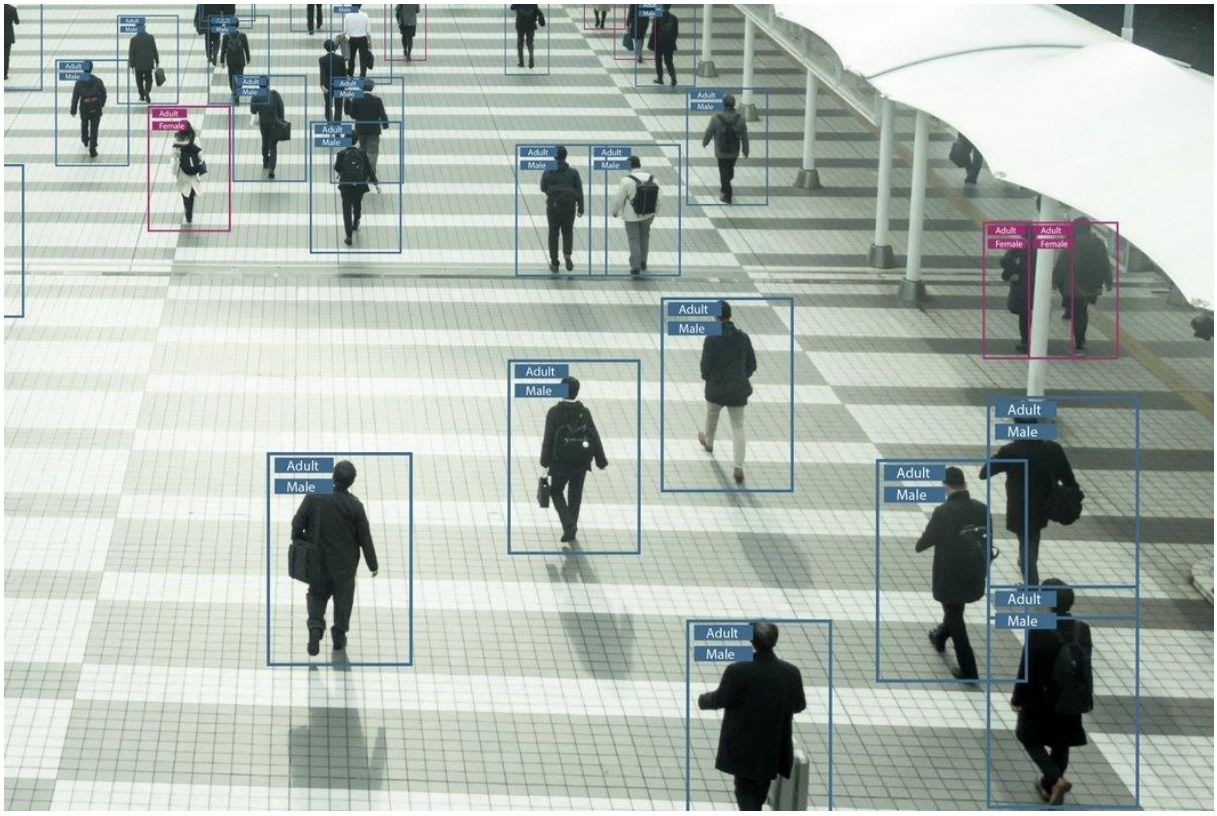
Računalni vid je složen zbog kvalitete i količine podataka, varijabilnosti slika, obrade u stvarnom vremenu, rubnih slučajeva, integracije s drugim sustavima te etičkih i osobnih pitanja.

Detaljno o teorijsko podlozi kako računalni vid funkcionira i gdje se sve koristi možete proučiti na stranici OpenCV-ija. [20] Mi ćemo se primarno fokusirati na primjenu računalnog vida.

6.1. Otkrivanje objekata

U središtu računalnog vida nalazi se klasifikacija slika, osnovni zadatak koji uključuje kategorizaciju ulazne slike u unaprijed definirane klase ili kategorije. Zamislite sustav sposoban razlikovati mačku, psa ili nijedno, jednostavnom analizom slike. Ova osnovna sposobnost predstavlja temelj za brojne druge primjene računalnog vida, otvarajući put naprednom vizualnom prepoznavanju.

Prelazeći s klasifikacije na otkrivanje objekata, dodaje se dodatni sloj složenosti. Ovaj proces uključuje identifikaciju objekata unutar slike i precizno određivanje njihove lokacije putem crtanja graničnih okvira. Zamislite autonomna vozila koja prepoznaju pješake i druga vozila, sigurnosne sustave koji detektiraju uljeze ili maloprodajne aplikacije koje prate proizvode na policama trgovina. Otkrivanje objekata omogućuje strojevima učinkovitiju navigaciju i interakciju s okolinom. [20]



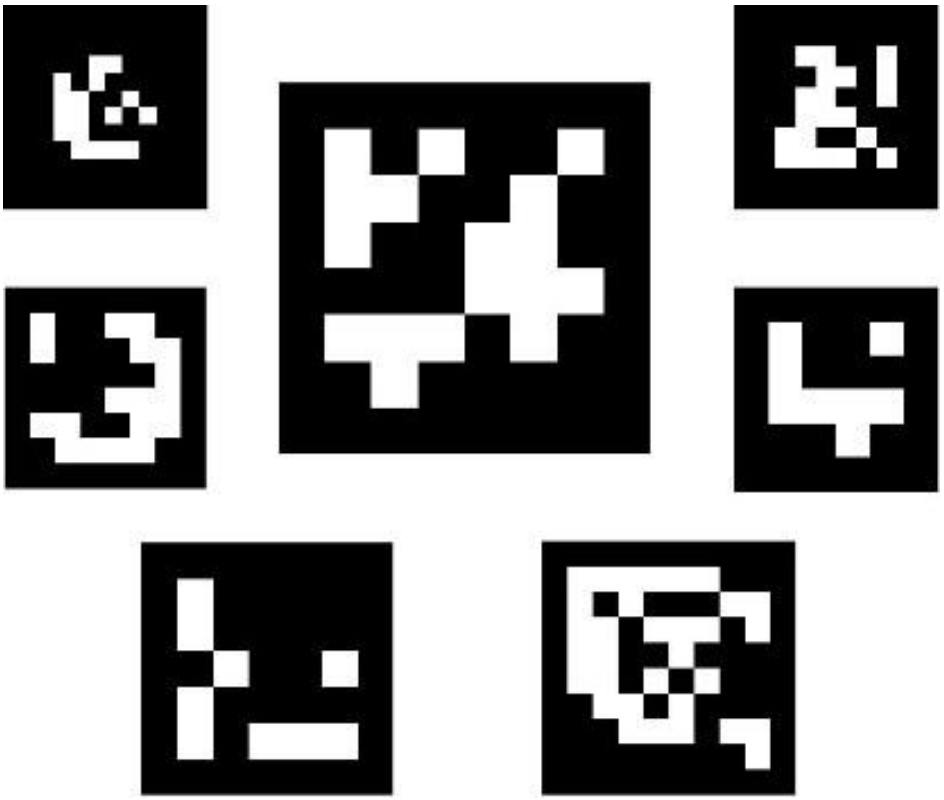
Slika 6.1 Detekcija objekata primjenom YOLOv3 metode dubokog učenja [21]

Otkrivanje objekata (vidjeti sliku 6.1) može se provoditi različitim metodama, uključujući metode podudaranja predložaka, metode temeljene na značajkama, detekciju temeljenu na strojnom učenju, detekciju temeljenu na dubokom učenju, detekciju rubova, metodu optičkog protoka, 3D detekciju objekata, detekciju fiducijalnih markera i druge metode. Svaka metoda ima svoje prednosti i prikladna je za različite primjene. Na primjer, metode detekcije temeljene na dubokom učenju iznimno su precizne, ali zahtijevaju značajne računalne resurse, dok su metode temeljene na značajkama brže i mogu se koristiti u aplikacijama u stvarnom vremenu.

Razmotrit ćemo metodu detekcije fiducijalnih markera, s posebnim naglaskom na ArUco oznake.

6.1.1. Fiducijalne oznake

Fiducijalni markeri, ili oznake, koriste se u različitim aplikacijama kako bi identificirali objekte. Njihova svrha je omogućiti prepoznavanje objekata putem određenih metoda. Ovisno o definiranom skupu oznaka, fiducijalni markeri omogućuju procjenu pozicije, kalibraciju kamere, navigaciju robota i primjenu proširene stvarnosti. Postoje različite vrste fiducijalnih oznaka kao što su CALTag, STag, WhyCon, WhyCode, ArUco, AprilTag i mnoge druge.



Slika 6.2 Primjer ArUco oznaka [22]

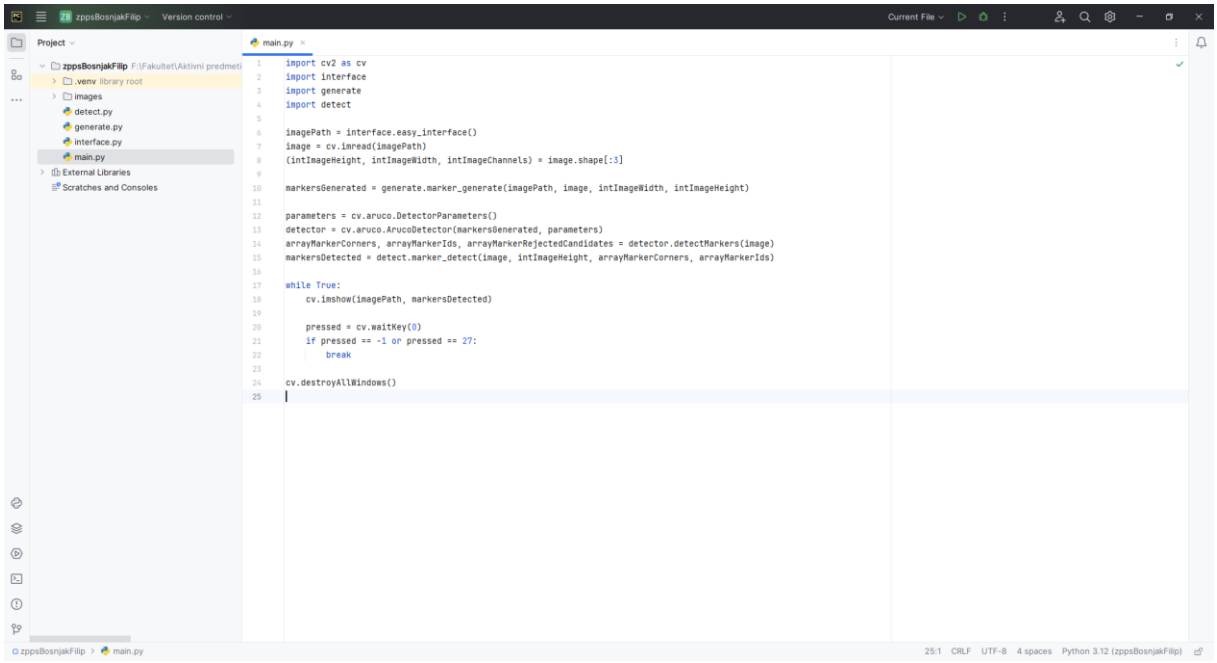
ArUco oznake (vidjeti sliku 6.2) predstavljaju unaprijed definirani skup rječnika te sadrže vanjsku crnu granicu i unutarnji bijeli jedinstveni binarni uzorak koji određuje identifikator oznake. Upravo ih ova struktura čini pogodnima za procjenu položaja, kalibraciju kamere, navigaciju robota i primjenu u proširenoj stvarnosti.

7. PRIMJENA ARUCO OZNAKA NA STATIČKIM SLIKAMA

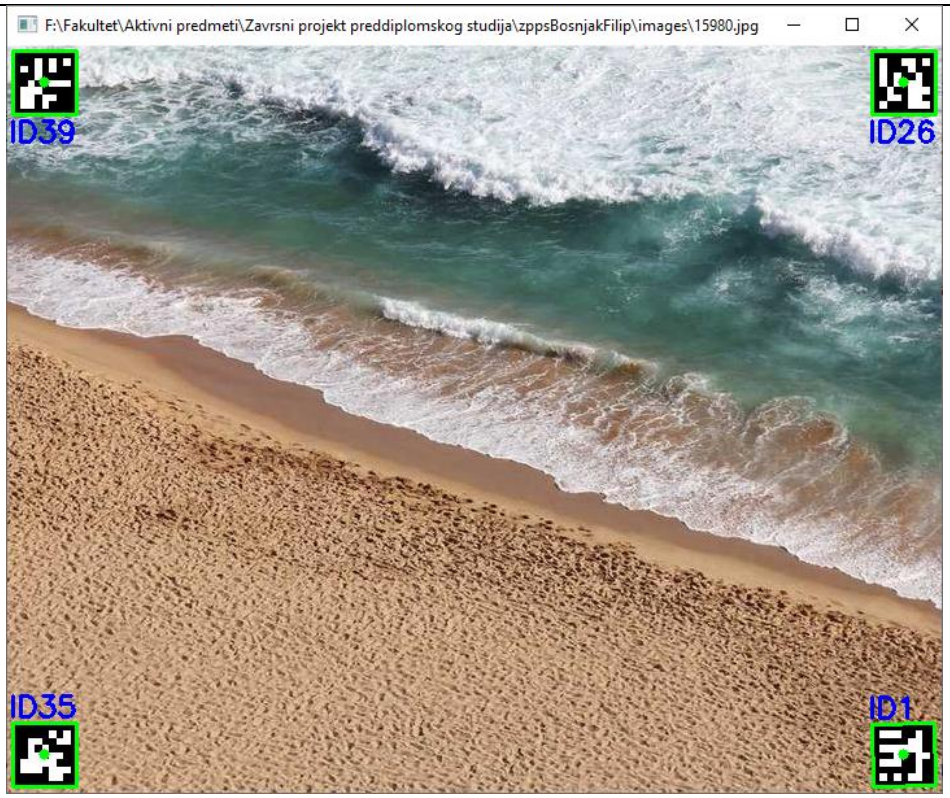
Prije nego što započnemo s programiranjem, važno je razjasniti nekoliko ključnih aspekata vezanih uz izradu zadatka. Kao što naslov poglavlja sugerira, primijenit ćemo ArUco oznake na statičkim slikama na sljedeći način. Prvo ćemo učitati datoteku i provjeriti je li to zaista slika formata koji je primjenjiv za statičke slike. Zatim ćemo generirati ArUco oznake na slici. Poznavajući rezoluciju slike, odabrat ćemo ArUco rječnik odgovarajuće veličine (engl. *size*) i odgovarajuće veličine rešetke (engl. *grid*; gdje se nalazi binarni uzorak) te oznaku jedinstvenog identifikatora. Nakon stvaranja markera, otkrit ćemo ArUco oznake, ali nećemo koristiti informacije koje smo sami stvorili, već ćemo detektirati ArUco oznake kao da ne znamo gdje se nalaze, tj. kao da ih nismo sami stvorili i nemamo pristup tim podacima. Ova funkcija bi bila korisna kao odvojeni segment koda, jer kada sami generiramo markere, možemo lakše doći do njihovih podataka budući da smo ih sami stvorili.

Projekt obuhvaća četiri programa: jedan glavni i tri koji se pozivaju unutar glavnog programa. U nastavku ćemo detaljno opisati svaki program zasebno.

7.1. Glavni program



Slika 7.1 Programski kod glavnog program (main.py)



Slika 7.2 Krajnji ishod glavnog programa

Glavni program (vidjeti slike 7.1 i 7.2), nazvan main.py, pokreće se klikom na strelicu zelenog obruba u gornjem desnom kutu slike 7.1 te poziva sve ostale potprograme. Programski kod izgleda ovako:

```
import cv2 as cv
import interface
import generate
import detect
```

Naredba *import cv2 as cv* poziva biblioteku OpenCV koja nam je potrebna za funkcionalnosti računalnog vida. Dio koda *as* omogućuje korištenje kratice *cv* umjesto *cv2* za pozivanje biblioteke što je korisno u slučaju nadogradnje na novu verziju OpenCV-a (npr. *cv3*). Tako svi pozivi biblioteke u kodu ostaju konzistentni i nije potrebno mijenjati liniju po liniju koda.

Naredbe *import interface*, *import generate* i *import detect* pozivaju potprograme koji se koriste za korisničko sučelje, stvaranje i otkrivanje ArUco oznaka.

```
imagePath = interface.easy_interface()
image = cv.imread(imagePath)
(intImageHeight, intImageWidth, intImageChannels) = image.shape[:3]
```

Varijabla *imagePath*, s pomoću funkcije *easy_interface* iz potprograma *interface*, pokreće jednostavno korisničko sučelje i dohvaća direktorij odabrane slike te nam vrača tip podatka string putanje datoteke.

Varijabla *image* koristi funkciju *imread*, iz biblioteke *cv*, za učitavanje matrice vrijednosti piksela slike čiju smo putanju datoteke dohvatili s pomoću varijable *imagePath*.

Linija koda *(intImageHeight, intImageWidth, intImageChannels) = image.shape[:3]* kreira varijable *intImageHeight*, *intImageWidth*, *intImageChannels* i u njih posprema redom vrijednosti: visine slike (u pikselima), širine slike (u pikselima) i broja kanala slike koji nam ukazuje radili se o RGB-u ili nekom drugom kanalu.

```
markersGenerated = generate.marker_generate(imagePath, image, intImageWidth,
intImageHeight)
```

Varijabla *markersGenerated* koristi funkciju *marker_generate*, iz potprograma *generate*, kojoj smo dodali argumente *imagePath*, *image*, *intImageWidth* i *intImageHeight*, kako bismo uspješno stvorili ArUco oznake na statičkoj slici. Detaljno objašnjenje bit će pruženo kasnije u tom potprogramu.

```
parameters = cv.aruco.DetectorParameters()
detector = cv.aruco.ArucoDetector(markersGenerated, parameters)
arrayMarkerCorners, arrayMarkerIds, arrayMarkerRejectedCandidates =
detector.detectMarkers(image)
markersDetected = detect.marker_detect(image, intImageHeight, arrayMarkerCorners,
arrayMarkerIds)
```

Varijabla *parameters* koristi objekt *DetectorParameters()*, iz biblioteke *cv* i *aruco* modula, kako bi postavila parametre za pronalaženje ArUco markera. S pomoću objekta *DetectorParameters* možemo mijenjati različite parametre poput pragova (engl. *threshold*). *Thresholding* je često korištena metoda za pretvaranje slika u sivim tonovima u binarne slike. Ona nam pomaže u odvajanju objekata od njihove pozadine, što pojednostavljuje analizu i obradu slike.

Varijabla *detector* koristi konstruktor klase *ArucoDetector()*, iz biblioteke *cv* i *aruco* modula, i sadrži argumente *markersGenerated* i *parameters*, kako bi postavili rječnik ArUco oznaka i parametre za pronalaženje istih.

Korištenjem jednadžbe *arrayMarkerCorners*, *arrayMarkerIds*, *arrayMarkerRejectedCandidates* = detector.*detectMarkers(image)* kreiramo tri varijable koje redom sadrže nizove: kutova oznaka, identifikatora oznaka i odbijenih kandidata oznaka sa

željene slike. Odbijeni kandidati su oni koji su ispunjavali određene kriterije, ali nisu zadovoljili sve potrebne uvjete.

Kreiranjem objekta *markersDetected* s pomoću naredbe *detect.marker_detect* i potrebnih argumenata obilježujemo otkrivene ArUco oznake.

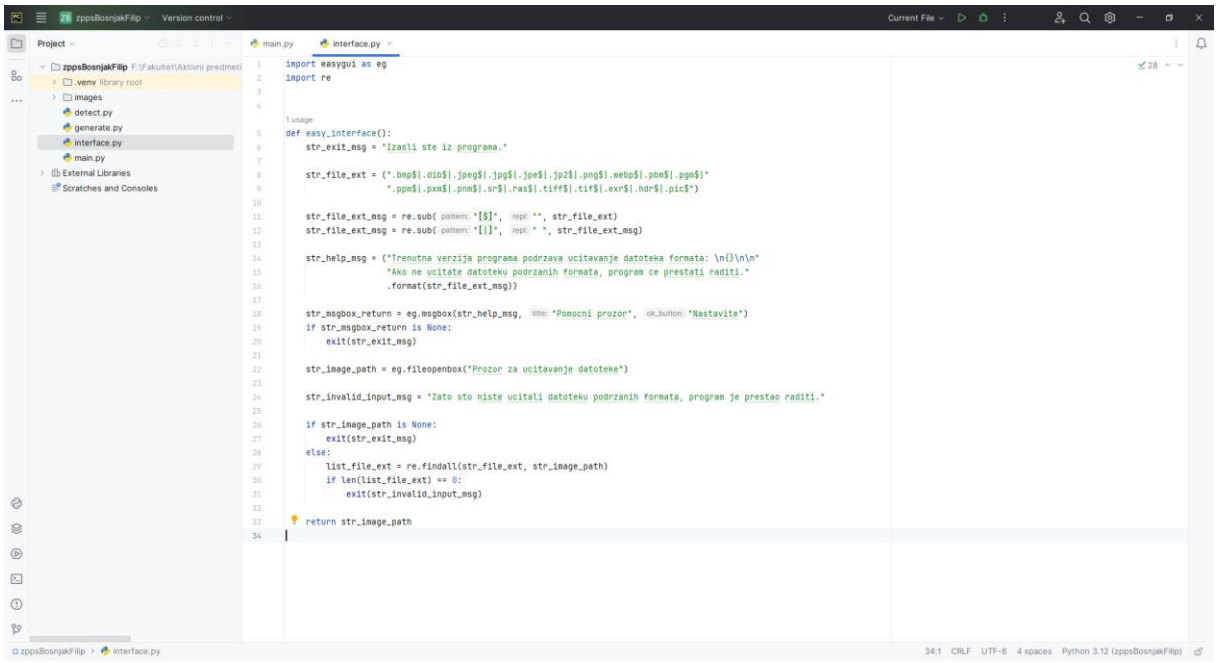
```
while True:
    cv.imshow(imagePath, markersDetected)

    pressed = cv.waitKey(0)
    if pressed == -1 or pressed == 27:
        break
```

Korištenjem gore navedenog koda osiguravamo kontinuiranu aktivnost programa, prikaz željene slike te stvorenih i otkrivenih markera, sve dok korisnik ne pritisne tipku *escape* ili gumb za zatvaranje prozora.

```
cv.destroyAllWindows()
```

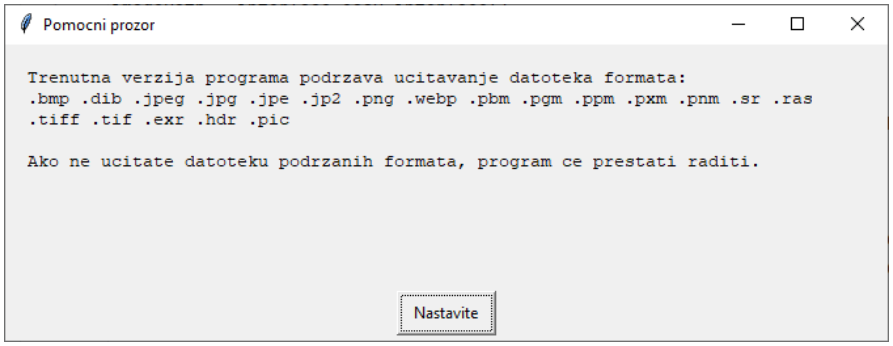
Nakon zatvaranja prozora, korištenjem naredbe *cv.destroyAllWindows()*, osiguravamo da se svi preostali aktivni prozori povezani s našim programom zatvore. I to je kraj glavnog programa sada ćemo svaki potprogram pojasniti kako bi riješili nekolicinu nedoumica.

7.2. Učitavanje statičke slike

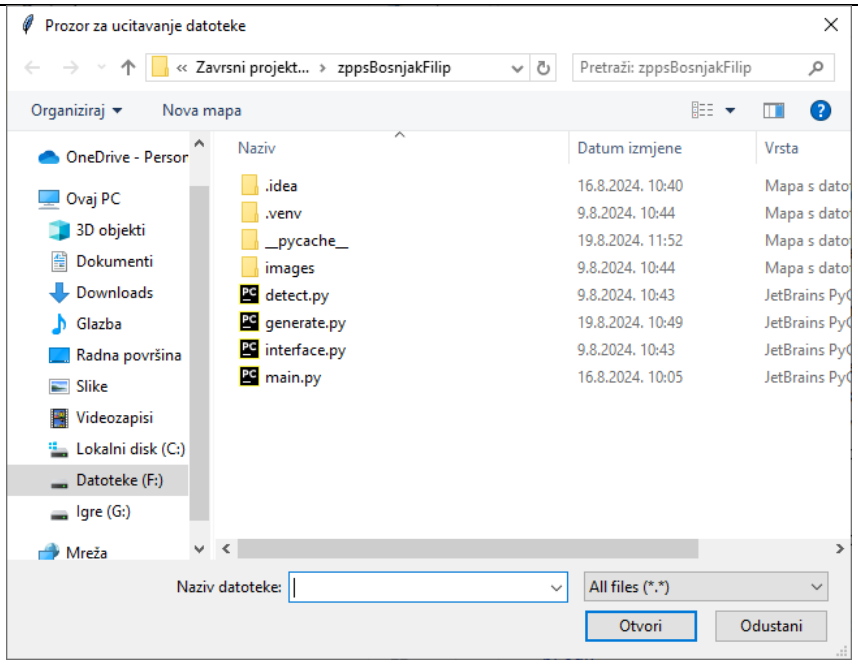


```
import easygui as eg
import re


1 usage
def easy_interface():
    str_exit_msg = "Izasli ste iz programa."

    str_file_ext = (*.bmp$|.dib$|.jpeg$|.jpg$|.jpe$|.jp2$|.png$|.webp$|.pbm$|.pgm$|"
                    "*.ppm$|.pxm$|.pnm$|.sr$|.ras$|.tiff$|.tif$|.exr$|.hdr$|.pic$")

    str_file_ext_msg = re.sub( pattern: "[$]", repl: "", str_file_ext)
    str_file_ext_msg = re.sub( pattern: "[|]", repl: " ", str_file_ext_msg)

    str_help_msg = ("Trenutna verzija programa podrzava ucitavanje datoteka formata: \n{}\n\n"
                    "Ako ne ucitate datoteku podrzanih formata, program ce prestati raditi."
                    .format(str_file_ext_msg))

    str_msgbox_return = eg.msgbox(str_help_msg, title: "Pomocni prozor", ok_button: "Nastavite")
    if str_msgbox_return is None:
        exit(str_exit_msg)

    str_image_path = eg.fileopenbox("Prozor za ucitavanje datoteke")

    str_invalid_input_msg = "Zato sto niste ucitali datoteku podrzanih formata, program je prestao raditi."

    if str_image_path is None:
        exit(str_exit_msg)
    else:
        list_file_ext = re.findall(str_file_ext, str_image_path)
        if len(list_file_ext) == 0:
            exit(str_invalid_input_msg)

    return str_image_path
```

Slika 7.3 Programski kod za učitavanje statičke slike (interface.py)



Slika 7.4 Prvi prozor za učitavanje slike



Slika 7.5 Drugi prozor za učitavanje slike

Potprogram za učitavanje statičke slike (vidjeti slike 7.3, 7.4 i 7.5), nazvan interface.py, koristi sljedeće biblioteke: *easygui* i *re*.

```
import easygui as eg
import re
```

Biblioteka *easygui* omogućuje brzu i učinkovitu implementaciju jednostavnih komponenti grafičkog sučelja, dok se biblioteka *re* koristi za regularne izraze (poznate kao regex). Regex predstavlja uzorak specifičnih brojeva i znakova koji su dozvoljeni za upotrebu unutar polja za unos teksta (engl. *textbox*) i drugih sličnih polja za unos (engl. *input field*).

```
def easy_interface():
    str_exit_msg = "Izasli ste iz programa."

    str_file_ext = (".bmp$|.dib$|.jpeg$|.jpg$|.jpe$|.jp2$|.png$|.webp$|.pbm$|.pgm$|"
                    ".ppm$|.pxm$|.pnm$|.sr$|.ras$|.tiff$|.tif$|.exr$|.hdr$|.pic$")

    str_file_ext_msg = re.sub("[$]", "", str_file_ext)
    str_file_ext_msg = re.sub("[|]", " ", str_file_ext_msg)
```

Izrazom *def easy_interface():* kreiramo funkciju *easy_interface* koja u svojem tijelu (engl. *body*) sadrži sljedeće.

Varijabla *str_exit_msg* sadrži poruku o grešci koja se prikazuje kada izađemo iz programa bez odabira slike na koju ćemo generirati i detektirati markere.

Varijabla *str_file_ext* sadrži sve ekstenzije koje će naš program podržavati prilikom pokretanja. Unutar zagrada mogu se primijetiti znakovi dolara i okomite crte koji omogućuju provjeru podržanih ekstenzija pomoću regularnih izraza (regex).

Oba izraza *str_file_ext_msg* omogućuju jasniji prikaz teksta podržanih ekstenzija koje će program ispisati bez prethodno navedenih znakova u zagradama.

```
    str_help_msg = ("Trenutna verzija programa podrzava ucitavanje datoteka formata:
\n{}\n\n"
                    "Ako ne ucitate datoteku podrzanih formata, program ce prestati raditi."
                    .format(str_file_ext_msg))

    str_msgbox_return = eg.msgbox(str_help_msg, "Pomocni prozor", "Nastavite")
    if str_msgbox_return is None:
        exit(str_exit_msg)
```

Varijabla *str_help_msg* sadrži uvodnu poruku programa koja specificira podržane i nepodržane funkcionalnosti.

Varijabla *str_msgbox_return* sadrži funkciju *msgbox*, iz biblioteke *easygui*, koja kreira jednostavno sučelje za prikaz uvodne poruke s gumbom za nastavak ili zatvaranje te pohranjuje odgovor.

Naredbama *if str_msgbox_return is None:* i *exit(str_exit_msg)* specificiramo da, ako je odabrano zatvaranje prozora, program će se prekinuti i ispisati poruku sadržanu u varijabli *str_exit_msg*.

```
    str_image_path = eg.fileopenbox("Prozor za ucitavanje datoteke")

    str_invalid_input_msg = "Zato sto niste ucitali datoteku podrzanih formata, program
je prestao raditi."
```

Varijabla *str_image_path* pokreće funkciju *fileopenbox* koja sadrži jednostavno sučelje za odabir datoteke te vrača njezinu putanju.

Varijabla *str_invalid_input_msg* sadrži poruku koja se prikazuje kada nije učitan podržani datotečni format.

```
    if str_image_path is None:
        exit(str_exit_msg)
    else:
        list_file_ext = re.findall(str_file_ext, str_image_path)
        if len(list_file_ext) == 0:
            exit(str_invalid_input_msg)

    return str_image_path
```

Linije *if str_image_path is None:* i *exit(str_exit_msg)* znače da, ako je odabrano zatvaranje prozora, program će se prekinuti i ispisati poruku sadržanu u varijabli *str_exit_msg*.

Naredbom *else:* zahtjevamo da, ako prethodno navedeni uvjet nije zadovoljen, program treba izvršiti sljedeće korake. Funkcijom *findall* pronalazimo sva datotečna proširenja sadržana u nazivu odabrane slike, koristeći izraz unutar varijable *list_file_ext*.

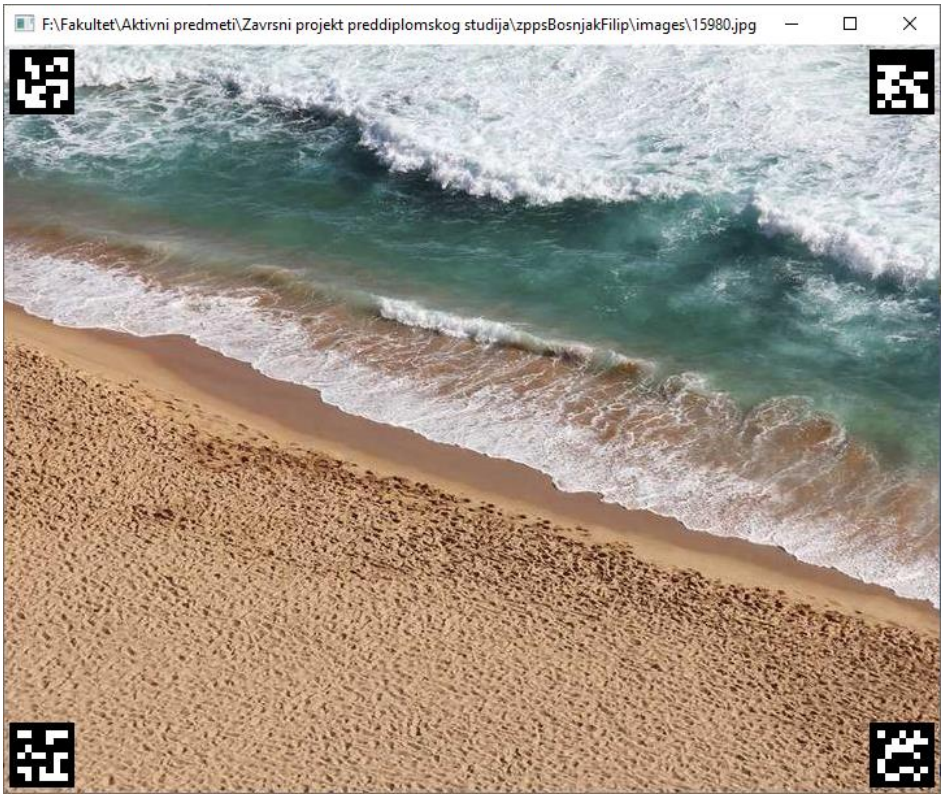
Nadalje, izrazima *if len(list_file_ext) == 0:* i *exit(str_invalid_input_msg)* naznačujemo da, ako nijedna datoteka nije pohranjena u listi, program ispisuje grešku sadržanu u varijabli *str_invalid_input_msg*.

Potprogram završava linijom koda *return str_image_path*, što znači da funkcija *easy_interface* vraća vrijednost varijable *str_image_path*, a to nam je putanja datoteke odabrane slike.

7.3. Generiranje ArUco oznaka

```python
def marker_generate(str_image_path, array_image, int_image_width, int_image_height):

    int_marker_size = list_marker_size[list_difference.index(min(list_difference))]

    if int_image_size_min <= 720:
        str_marker_grid = "7x7"
    elif 720 < int_image_size_min <= 1080:
        str_marker_grid = "6x6"
    elif 1080 < int_image_size_min <= 1440:
        str_marker_grid = "5x5"
    elif 1080 < int_image_size_min <= 2160:
        str_marker_grid = "4x4"
    else:
        str_marker_grid = "aruco"

    if int_marker_size == 1024:
        str_marker_type = "dict_" + str_marker_grid + "_original"
    else:
        str_marker_type = "dict_" + str_marker_grid + "_" + str(int_marker_size)

    dict_marker = cv.aruco.getPredefinedDictionary(dict_marker_dict[str_marker_type])
    array_marker = np.zeros( shape: (int_marker_size, int_marker_size, 1),  dtype: "uint8")

    int_pos_x = [4, (int_image_width - int_marker_size) - 4]
    int_pos_y = [4, (int_image_height - int_marker_size) - 4]
    for i in range(len(int_pos_x)):
        for j in range(len(int_pos_y)):
            int_marker_id = np.random.randint( low: 0, int_marker_size - 1)
            cv.aruco.generateImageMarker(dict_marker, int_marker_id, int_marker_size, array_marker,  borderBits: 1)

            array_image[int_pos_y[j]:int_pos_y[j] + array_marker.shape[0],
            int_pos_x[i]:int_pos_x[i] + array_marker.shape[1]] = array_marker

    cv.imwrite(str_image_path, array_image)

    return dict_marker
```

Slika 7.6 Programski kod za generiranje ArUco oznaka (generate.py)



Slika 7.7 Krajnji ishod generiranja ArUco oznaka

Potprogram za generiranje ArUco oznaka (vidjeti slike 7.6 i 7.7), nazvan generate.py, koristi sljedeće biblioteke: *cv2* i *numpy*.

```
import cv2 as cv
import numpy as np
```

Biblioteka *numpy* se koristi za napredno rukovanje nizovima, pruža dodatne matematičke metode za upravljanje nizovima, osigurava brzo i učinkovito izvršavanje programa te izvrsno funkcionira s drugim bibliotekama.

```
def marker_generate(str_image_path, array_image, int_image_width,
int_image_height):
    dict_marker_dict = {
        "dict_4x4_50": cv.aruco.DICT_4X4_50,
        "dict_4x4_100": cv.aruco.DICT_4X4_100,
        "dict_4x4_250": cv.aruco.DICT_4X4_250,
        "dict_4x4_1000": cv.aruco.DICT_4X4_1000,
        "dict_5x5_50": cv.aruco.DICT_5X5_50,
        "dict_5x5_100": cv.aruco.DICT_5X5_100,
        "dict_5x5_250": cv.aruco.DICT_5X5_250,
        "dict_5x5_1000": cv.aruco.DICT_5X5_1000,
        "dict_6x6_50": cv.aruco.DICT_6X6_50,
        "dict_6x6_100": cv.aruco.DICT_6X6_100,
        "dict_6x6_250": cv.aruco.DICT_6X6_250,
        "dict_6x6_1000": cv.aruco.DICT_6X6_1000,
        "dict_7x7_50": cv.aruco.DICT_7X7_50,
        "dict_7x7_100": cv.aruco.DICT_7X7_100,
        "dict_7x7_250": cv.aruco.DICT_7X7_250,
        "dict_7x7_1000": cv.aruco.DICT_7X7_1000,
        "dict_aruco_original": cv.aruco.DICT_ARUCO_ORIGINAL
    }
    list_marker_size = [50, 100, 250, 1000, 1024]
```

Prvo kreiramo funkciju *marker_generate* s argumentima *str_image_path*, *array_image*, *int_image_width* i *int_image_height*. Navedeni argumenti se točno navode u glavnom programu. Argumenti navedeni iznad potrebni su nam kako bismo odredili putanju datoteke slike, matrični zapis slike te širinu i visinu slike u pikselima.

Rječnik *dict_marker_dict* sadrži rječnike tipova oznaka koji se pozivaju iz biblioteke *cv2*. Za rječnik *dict_4x4_50* s naredbom *cv.aruco.DICT_4X4_50*, naznačujemo da je to ArUco rječnik rešetke 4x4 i veličine 50, što znači da je marker širok i visok 50 piksela. Rešetke gdje se nalazi binarni zapis su 4x4 unutar tih 50 piksela, dok je važno naglasiti da je stvarna rešetka 5x5, ali se koristi 4x4 jer gledamo binarni zapis, a ne dodatne crne piksele koji su zapravo granica (engl. border) markera. Rječnik *dict_aruco_original* je veličine 1024.

Varijabla *list_marker_size* sadrži moguće predefinirane veličine tipova ArUco rječnika. Također, ovim radom obuhvaćeni su svi tipovi ArUco oznaka iz standardne biblioteke.

Sljedeći dijelovi koda posvećeni su određivanju tipa ArUco oznake na temelju rezolucije učitane slike. Imajući na umu sljedeće činjenice objašnjene u priloženom videu u literaturi. Ako su dimenzije rešetke markera manje, slika se može lakše detektirati s veće udaljenosti jer su binarni zapis i granica markera uočljiviji. Isto tako, to smanjuje broj tipova markera koje možemo koristiti. Povećanjem dimenzija rešetke markera možemo povećati broj tipova markera koje možemo koristiti. Međutim, ako markere udaljimo od kamere, bitovi će biti premali da bi ih kamera prepoznala. [23]

Iako se navedeno primarno odnosi na slike koje će kamera detektirati, princip se može primijeniti i na statičke digitalne slike, čime se može dodatno proširiti primjena programa u detekciji ArUco oznaka s pomoću kamera.

```
int_image_divisor = 10
    if int_image_width >= int_image_height:
        float_image_quotient = int_image_height / int_image_divisor
        int_image_size_min = int_image_height
    else:
        float_image_quotient = int_image_width / int_image_divisor
        int_image_size_min = int_image_width
```

Stoga, eksperiment je pokazao da je vrijednost 10 (sadržana u varijabli *int_image_divisor*) optimalna za podjelu manje dimenzije slike, bilo širine ili visine slike, kako bi marker u potpunosti stao u sliku.

Prva uvjetna naredba *if int_image_width >= int_image_height:* određuje da, ako je širina slike veća od visine, treba izračunati varijablu *float_image_quotient* prema visini slike i postaviti varijablu *int_image_size_min* na najmanju vrijednost jednaku visini slike. Naredba *else:* primjenjuje suprotnu logiku, ako je širina slike najmanja vrijednost, izvršava se sve prethodno navedeno, ali koristeći širinu slike umjesto visine.

```
    list_difference = []
    for i in range(len(list_marker_size)):
        list_difference.append(abs(float_image_quotient - list_marker_size[i]))

    int_marker_size = list_marker_size[list_difference.index(min(list_difference))]
```

Usporedbom vrijednosti varijable *float_image_quotient* s najmanjom vrijednošću dobivenom oduzimanjem varijable *float_image_quotient* od svake pojedinačne vrijednosti iz niza

list_marker_size, možemo odrediti broj s najmanjom razlikom i odabrati ga te tako postići optimalnu veličinu markera za vidljivost sadržaja slike i detekciju markera.

Varijabla *int_marker_size* pohranjuje vrijednost niza *list_marker_size* s pomoću indeksa istog koji sadrži indeks najmanje razlike unutar niza *list_difference*. Indeks predstavlja poziciju elementa unutar niza. U programiranju, indeksiranje započinje od nule.

```
if int_image_size_min <= 720:
    str_marker_grid = "7x7"
elif 720 < int_image_size_min <= 1080:
    str_marker_grid = "6x6"
elif 1080 < int_image_size_min <= 1440:
    str_marker_grid = "5x5"
elif 1080 < int_image_size_min <= 2160:
    str_marker_grid = "4x4"
else:
    str_marker_grid = "aruco"
```

Nakon određivanja veličine markera, uvjetnim naredbama definiramo rešetku markera. Ako je manja dimenzija slike, dodjeljuje se veća rešetka markera (razlog tome je prethodno dan). Budući da se radi o statičkim digitalnim slikama, rezolucije su standardizirane, no ovom logikom pokrivene su i neke nestandardne rezolucije.

```
if int_marker_size == 1024:
    str_marker_type = "dict_" + str_marker_grid + "_original"
else:
    str_marker_type = "dict_" + str_marker_grid + "_" + str(int_marker_size)
```

Ovim dijelom koda, budući da originalni tip ArUco oznaka ima veličinu 1024, a to nije navedeno u imenu, izrađujemo logiku kojom se ArUco oznaci s imenom *original* dodjeljuje veličina 1024. U ostalim slučajevima, tip se dodjeljuje na sljedeći način: *str_marker_type = "dict_" + str_marker_grid + "_" + str(int_marker_size)*, što znači da se varijabli *str_marker_type* dodjeljuje točan naziv kako je prethodno izračunato.

```
    dict_marker =
cv.aruco.getPredefinedDictionary(dict_marker_dict[str_marker_type])
    array_marker = np.zeros((int_marker_size, int_marker_size, 1), "uint8")
```

Varijablom *dict_marker* određujemo konačan tip rječnika za našu oznaku koristeći funkciju *getPredefinedDictionary* iz biblioteke *cv2*. Postavljamo argumente kako bismo iz glavnog rječnika *dict_marker_dict*, koji sadrži sve tipove oznaka, dohvatili instancu koja nam je potrebna (s pomoću varijable *str_marker_type*).

Varijabla *array_marker* koristi biblioteku *numpy* i funkciju *zeros* koja stvara matricu zadane veličine *(int_marker_size, int_marker_size, 1)* ispunjenu nulama. Argument *"uint8"* određuje tip podataka koji će se koristiti za niz.

```
    int_pos_x = [4, (int_image_width - int_marker_size) - 4]
    int_pos_y = [4, (int_image_height - int_marker_size) - 4]
    for i in range(len(int_pos_x)):
        for j in range(len(int_pos_y)):
            int_marker_id = np.random.randint(0, int_marker_size - 1)
            cv.aruco.generateImageMarker(dict_marker, int_marker_id, int_marker_size,
array_marker, 1)
```

Listama *int_pos_x* i *int_pos_y* definiramo granice unutar kojih se markeri mogu postaviti na slici. Prvu vrijednost postavljamo na 4 piksela od ruba po x i y osi (kako je vidljivo iz uglatih zagrada), dok drugu vrijednost određujemo ovisno o x i y osi koristeći širinu ili visinu slike, umanjujući ju za 4 piksela. Ova 4 piksela su uzeta kako marker ne bi bio na samom rubu slike, čime se osigurava vidljiv kontrast između slike i markera. Markere ćemo postavljati uz rubove definirane granice kako bi se izbjegao gubitak sadržaja slike, koji je u većini slučajeva centriran. Petlja *for i in range(len(int_pos_x)):* i ugniježđena petlja *for j in range(len(int_pos_y)):*, budući da su duljine varijabli *int_pos_x* i *int_pos_y* jednake vrijednosti 2, omogućuju generiranje markera uz prethodno definirane rubove granice slike. Unutar petlji se konačno generiraju markeri jer uz tip rječnika markera definiramo i identifikator, što je posljednji potreban korak pri definiciji markera kako bi svaki marker imao jedinstven identifikator. Stoga se ta naredba koristi unutar petlji, tj. svakim prolazom petlje definira se nasumično drugi identifikator i stvara se konačno definirani marker na slici s pomoću sljedeće dvije naredbe: *int_marker_id = np.random.randint(0, int_marker_size - 1)* za generiranje identifikatora markera i *cv.aruco.generateImageMarker(dict_marker, int_marker_id, int_marker_size, array_marker, 1)* za generiranje konačnog markera.

```
            array_image[int_pos_y[j]:int_pos_y[j] + array_marker.shape[0],
            int_pos_x[i]:int_pos_x[i] + array_marker.shape[1]] = array_marker
```

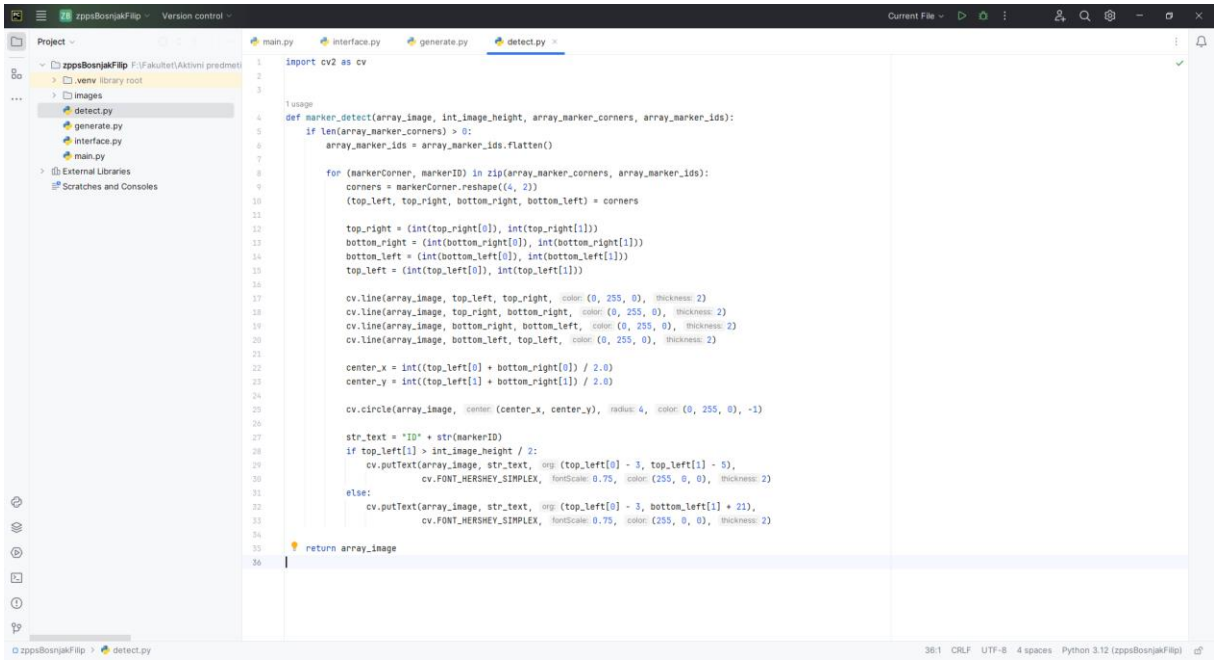
Varijabla *array_marker* sadrži matricu slike u koju se pohranjuju novo stvorene ArUco oznake. Oznake se pohranjuju s njihovim lokacijama u matricu slike, čime se briše prethodni sadržaj slike. To znači da slika više nije originalna, već je prepisana oznakama, što onemogućava povratak na izvorni sadržaj slike ako imamo samo datoteku slike u formatu slike. Time je petlja završena i oznake su stvorene na željenoj učitanoj slici.

```
        cv.imwrite(str_image_path, array_image)

        return dict_marker
```

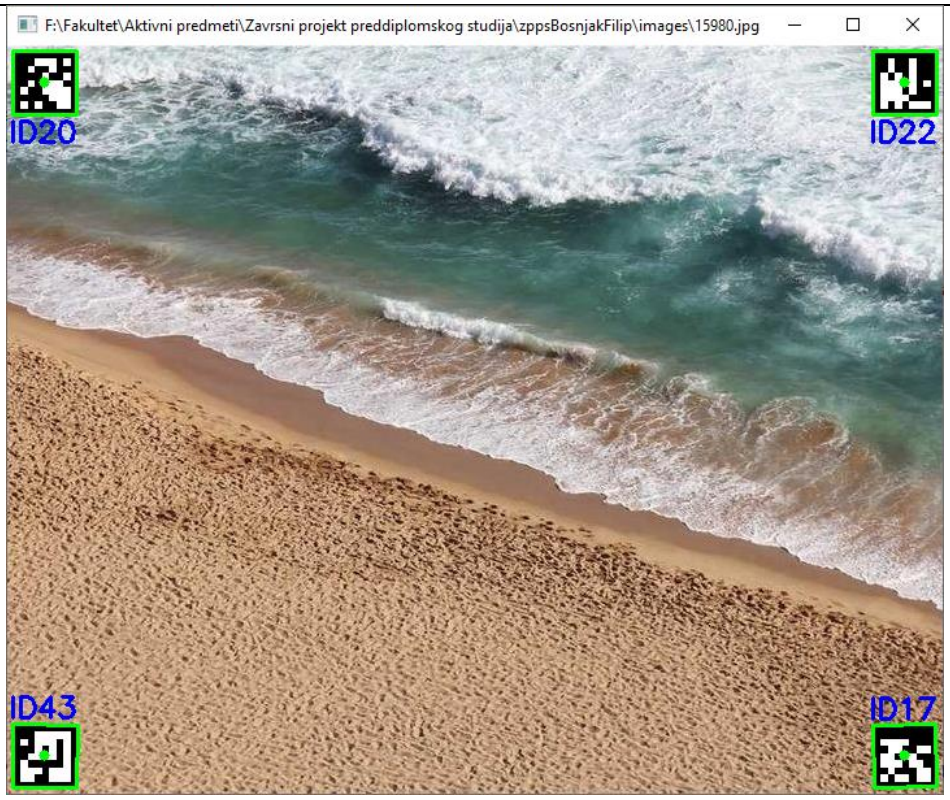
Međutim, nužno je pohraniti stvorene ArUco oznake u datoteku, inače bi se oznake prikazale na slici, ali ne bi bile sačuvane nakon zatvaranja programa. Ovaj problem rješavamo naredbom *cv.imwrite(str_image_path, array_image)*, gdje je *str_image_path* putanja slike, a *array_image* novo stvorena matrica slike s ArUco oznakama.

Naredbom *return dict_marker* funkcija *marker_generate* vraća, u ovom slučaju u glavni program, rječnik tipa ArUco oznaka, ali ne i njihove identifikatore. Ova je vrijednost važna kako bismo mogli detektirati ArUco oznake na različitim slikama i točno znali o kojom se tipu rječnika oznaka radi. Također, poznavajući originalnu rezoluciju markera, možemo izbjeći pogrešno prepoznavanje oznaka i neispravnu primjenu, poput netočne kalibracije kamere.

7.4. Detekcija ArUco oznaka



Slika 7.8 Programski kod za detekciju ArUco oznaka (detect.py)



Slika 7.9 Krajnji ishod detekcije ArUco oznaka

Potprogram za detekciju ArUco oznaka (vidjeti slike 7.8 i 7.9), nazvan detect.py, koristi biblioteku *cv2*.

```
import cv2 as cv


def marker_detect(array_image, int_image_height, array_marker_corners,
array_marker_ids):
    if len(array_marker_corners) > 0:
        array_marker_ids = array_marker_ids.flatten()
```

Najprije definiramo funkciju *marker_detect* s argumentima opisanim u potpoglavlju glavnog programa. Zatim, naredbom *if len(array_marker_corners) > 0:* provjeravamo jesmo li detektirali kutove markera. Ako jesmo, izvršavamo sljedeće korake; ako nismo, potprogram se završava i vraća sliku bez detektiranih markera.

Naredbom *array_marker_ids = array_marker_ids.flatten()* pretvaramo niz identifikatora markera u dvodimenzionalni oblik kako bismo mogli manipulirati tim podacima nad detektiranim kutovima.

```
for (markerCorner, markerID) in zip(array_marker_corners, array_marker_ids):
    corners = markerCorner.reshape((4, 2))
    (top_left, top_right, bottom_right, bottom_left) = corners
```

Linijom koda *for (markerCorner, markerID) in zip(array_marker_corners, array_marker_ids):* omogućujemo iteraciju kroz sve kutove i identifikatore u listi, pri čemu funkcija *zip* spaja elemente kutova i identifikatora u jedan element liste.

Izrazom *corners = markerCorner.reshape((4, 2))* stvaramo listu svih kutova s x i y koordinatama u njihove odgovarajuće ugniježđene liste.

Zatim, izrazom *(top_left, top_right, bottom_right, bottom_left) = corners* segmentiramo svaku ugniježđenu listu u pojedinačnu varijablu odgovarajuće pozicije detektiranih ArUco oznaka.

```
top_right = (int(top_right[0]), int(top_right[1]))
bottom_right = (int(bottom_right[0]), int(bottom_right[1]))
bottom_left = (int(bottom_left[0]), int(bottom_left[1]))
top_left = (int(top_left[0]), int(top_left[1]))
```

Skupom naredbi koje sadrže varijable *top_right*, *bottom_right*, *bottom_left* i *top_left*, te s pomoću gore navedene *for* petlje za svaku detektiranu ArUco oznaku, konačno definiramo niz (istog sadržaja kao prethodno navedeno) kojim je moguće dalje manipulirati u programu.

```
cv.line(array_image, top_left, top_right, (0, 255, 0), 2)
cv.line(array_image, top_right, bottom_right, (0, 255, 0), 2)
cv.line(array_image, bottom_right, bottom_left, (0, 255, 0), 2)
cv.line(array_image, bottom_left, top_left, (0, 255, 0), 2)
```

Linijama koda koje koriste funkciju *line* iz biblioteke *cv2* crtamo granicu kako bismo uočili otkrivene ArUco oznake. Argument *array_image* sadrži matrični zapis slike, dok su ostali argumenti dodani kako bi se granica nacrtala zelenom bojom i debljinom od 2 piksela, redom s lijeva na desno, od gore prema dolje, dok se ne ispuni cijela granica.

```
center_x = int((top_left[0] + bottom_right[0]) / 2.0)
center_y = int((top_left[1] + bottom_right[1]) / 2.0)

cv.circle(array_image, (center_x, center_y), 4, (0, 255, 0), -1)
```

Varijable *center_x* i *center_y* u svakom prolazu *for* petlje izračunavaju središte markera, uzimajući u obzir udaljenost između gornjeg lijevog i donjeg desnog kuta, te dijeleći tu udaljenost s dva. Na taj način dijagonalna udaljenost se prepolovi i zaokruži na cijeli broj.

Korištenjem naredbe *cv.circle* unesenih argumenata *(array_image, (center_x, center_y), 4, (0, 255, 0), -1)*, označavamo središte markera na slici zelenom bojom i debljinom od 4 piksela. Argument *-1* označava da je kružnica ispunjena, odnosno da nema šupljinu (kao što je vidljivo na slici 7.9).

```
            str_text = "ID" + str(markerID)
            if top_left[1] > int_image_height / 2:
                cv.putText(array_image, str_text, (top_left[0] - 3, top_left[1] - 5),
                        cv.FONT_HERSHEY_SIMPLEX, 0.75, (255, 0, 0), 2)
            else:
                cv.putText(array_image, str_text, (top_left[0] - 3, bottom_left[1] + 21),
                        cv.FONT_HERSHEY_SIMPLEX, 0.75, (255, 0, 0), 2)
```

Sljedeći dio koda sadrži varijablu *str_text*, koja sadrži tekst o identifikatoru određenog tipa rječnika ArUco oznake. S pomoću uvjetnog izraza *if top_left[1] > int_image_height / 2:* određujemo da, ako se oznaka nalazi na gornjem rubu slike, tekst identifikatora postavljamo funkcijom *putText* ispod oznake. U suprotnom *else:*, ako se oznaka nalazi na donjem rubu slike, tekst identifikatora postavljamo funkcijom *putText* iznad oznake.

```
        return array_image
```

Na kraju, funkcija *marker_detect* vraća matricu novo zapisanih vrijednosti slike. Kao što je objašnjeno u prethodnom potpoglavlju o generiranju ArUco oznaka, sliku je potrebno spremiti kako bi se sačuvale otkrivene oznake. Međutim, novu datoteku slike želimo spremiti samo prilikom generiranja oznaka, a ne prilikom njihove detekcije. Svrha ovog potprograma nije prebrisati sadržaj slike i stvoriti novu, već detektirati ArUco oznake na bilo kojoj učitanoj statičkoj slici koja ih sadrži, te ih označiti i pronaći njihove pozicije.

8. ZAKLJUČAK

Detekcija objekata u današnjem društvu neizbježan je pojam, a njezina implementacija prisutna je gotovo svugdje, od tvornica do poljoprivrednih površina. Sustav za generiranje i detekciju ArUco oznaka može se primijeniti na različite aplikacije, ovisno o njihovoj složenosti. Na primjer, za detekciju ArUco oznaka u videozapisima, a ne na statičkim slikama, potrebno je dodati i promijeniti nekoliko linija koda, dok veći dio programa ostaje nepromijenjen.

Jedna od prednosti ovog projekta je njegova fleksibilnost; nigdje u kodu nije striktno definirana varijabla nazvana ArUco marker, već samo marker, što omogućuje izmjenu koda jednostavnom promjenom biblioteka i korištenih objekata, bez potrebe za promjenom naziva varijabli, funkcija ili programa. Time se omogućuje primjena različitih fiducijalnih markera za detekciju objekata. Isto tako, gdje god je bilo moguće, zadržan je pristup koji minimalno striktno definira rješenja, ostavljajući prostor za manipulaciju podacima na različitim modelima.

Ipak, projekt ima određene nedostatke koji omogućuju daljnju razradu. Jedan od nedostataka je taj što, ako slika ne pruža dovoljan kontrast između oznake i pozadine, oznaka neće biti uočljiva jer se detekcija ArUco oznaka temelji na prepoznavanju prijelaza s pozadine na crnu granicu oznake. Također, metoda za prepoznavanje rezolucije digitalne slike i određivanje odgovarajuće oznake može se dodatno testirati. Iako je već testirana na većini standardnih formata, to ne znači da su sve mogućnosti pokrivene, posebno za nestandardne formate slika.

LITERATURA

[1] *Programiranje: Osnovni pojmovi*, arhiva-2021.loomen.carnet.hr/mod/book/view.php?id=190241&chapterid=43125, pristupljeno 8.7.2024.

[2] *Program, programiranje, programski jezici: Faze programiranja*, arhiva-2021.loomen.carnet.hr/mod/book/view.php?id=644630&chapterid=125189, pristupljeno 8.7.2024.

[3] *What Is a Programming Language?*, www.codecademy.com/resources/blog/programming-languages, pristupljeno 12.7.2024.

[4] *Toshiba 2SC5197 Transistor NPN Triple Diffused Type Power Amp Use OM0148D1*, https://richelectronics.co.uk/product/toshiba-2sc5197-transistor-npn-triple-diffused-type-power-amp-use-om0148d1, pristupljeno 6.8.2024.

[5] *VIŠE O TRANZISTORIMA*, https://soldered.com/hr/learn/vise-o-tranzistorima, pristupljeno 6.8.2024.

[6] *Beginner's Python Tutorial: Learn Python*, https://python.land/python-tutorial, pristupljeno 6.8.2024.

[7] *Welcome to Python.org*, www.python.org, pristupljeno 6.8.2024.

[8] *What Is C++? (And How to Learn It)*, www.coursera.org/articles/what-is-c-plus-plus, pristupljeno 7.8.2024.

[9] *Standard C++*, https://isocpp.org, pristupljeno 7.8.2024.

[10] *What is Java technology and why do I need it?*, www.java.com/en/download/help/whatis_java.html, pristupljeno 7.8.2024.

[11] *Java (programming language)*, https://en.wikipedia.org/wiki/Java_(programming_language), pristupljeno 7.8.2024.

[12] *Translators*, www.computerscience.gcse.guru/theory/translators, pristupljeno 7.8.2024.

[13] *What is an IDE? – Integrated Development Environment*, www.geeksforgeeks.org/what-is-ide, pristupljeno 8.8.2024.

[14] *What is PyCharm?*, www.geeksforgeeks.org/what-is-pycharm, pristupljeno 8.8.2024.

[15] *Install PyCharm*, www.jetbrains.com/help/pycharm/installation-guide.html, pristupljeno 8.8.2024.

[16] *What is computer vision?*, www.ibm.com/topics/computer-vision, pristupljeno 12.8.2024.

[17] *What is AI?*, www.ibm.com/topics/artificial-intelligence, pristupljeno 12.8.2024.

[18] *What is ML?*, www.ibm.com/topics/machine-learning, pristupljeno 12.8.2024.

[19] *What is a neural network?*, www.ibm.com/topics/neural-networks, pristupljeno 12.8.2024.

[20] *What is Computer Vision in 2024? A Beginners Guide*, https://opencv.org/blog/what-is-computer-vision, pristupljeno 12.8.2024.

[21] *YOLOv3: Real-Time Object Detection Algorithm (Guide)*, https://viso.ai/deep-learning/yolov3-overview, pristupljeno 12.8.2024.

[22] *Detection of ArUco Markers*, https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html, pristupljeno 12.8.2024.

[23] *2. Aruco Markers and Dictionaries*, www.youtube.com/watch?v=aWmzhmfahh8, pristupljeno 19.8.2024.

PRILOZI

I. Programski kod glavnog programa (main.py)

II. Programski kod za učitavanje statičke slike (interface.py)

III. Programski kod za generiranje ArUco oznaka (generate.py)

IV. Programski kod za detekciju ArUco oznaka (detect.py)

Prilog I: Programski kod glavnog programa (main.py)

```
import cv2 as cv
import interface
import generate
import detect

imagePath = interface.easy_interface()
image = cv.imread(imagePath)
(intImageHeight, intImageWidth, intImageChannels) = image.shape[:3]

markersGenerated = generate.marker_generate(imagePath, image, intImageWidth,
intImageHeight)

parameters = cv.aruco.DetectorParameters()
detector = cv.aruco.ArucoDetector(markersGenerated, parameters)
arrayMarkerCorners, arrayMarkerIds, arrayMarkerRejectedCandidates =
detector.detectMarkers(image)
markersDetected = detect.marker_detect(image, intImageHeight, arrayMarkerCorners,
arrayMarkerIds)

while True:
    cv.imshow(imagePath, markersDetected)

    pressed = cv.waitKey(0)
    if pressed == -1 or pressed == 27:
        break

cv.destroyAllWindows()
```

Prilog II: Programski kod za učitavanje statičke slike (interface.py)

```
import easygui as eg
import re


def easy_interface():
    str_exit_msg = "Izasli ste iz programa."

    str_file_ext = (".bmp$|.dib$|.jpeg$|.jpg$|.jpe$|.jp2$|.png$|.webp$|.pbm$|.pgm$|"
                    ".ppm$|.pxm$|.pnm$|.sr$|.ras$|.tiff$|.tif$|.exr$|.hdr$|.pic$")

    str_file_ext_msg = re.sub("[$]", "", str_file_ext)
    str_file_ext_msg = re.sub("[|]", " ", str_file_ext_msg)

    str_help_msg = ("Trenutna verzija programa podrzava ucitavanje datoteka formata:
\n{}\n\n"
                    "Ako ne ucitate datoteku podrzanih formata, program ce prestati raditi."
                    .format(str_file_ext_msg))

    str_msgbox_return = eg.msgbox(str_help_msg, "Pomocni prozor", "Nastavite")
    if str_msgbox_return is None:
        exit(str_exit_msg)

    str_image_path = eg.fileopenbox("Prozor za ucitavanje datoteke")

    str_invalid_input_msg = "Zato sto niste ucitali datoteku podrzanih formata, program
je prestao raditi."

    if str_image_path is None:
        exit(str_exit_msg)
    else:
        list_file_ext = re.findall(str_file_ext, str_image_path)
        if len(list_file_ext) == 0:
            exit(str_invalid_input_msg)

    return str_image_path
```

Prilog III: Programski kod za generiranje ArUco oznaka (generate.py)

```
import cv2 as cv
import numpy as np


def marker_generate(str_image_path, array_image, int_image_width,
int_image_height):
    dict_marker_dict = {
        "dict_4x4_50": cv.aruco.DICT_4X4_50,
        "dict_4x4_100": cv.aruco.DICT_4X4_100,
        "dict_4x4_250": cv.aruco.DICT_4X4_250,
        "dict_4x4_1000": cv.aruco.DICT_4X4_1000,
        "dict_5x5_50": cv.aruco.DICT_5X5_50,
        "dict_5x5_100": cv.aruco.DICT_5X5_100,
        "dict_5x5_250": cv.aruco.DICT_5X5_250,
        "dict_5x5_1000": cv.aruco.DICT_5X5_1000,
        "dict_6x6_50": cv.aruco.DICT_6X6_50,
        "dict_6x6_100": cv.aruco.DICT_6X6_100,
        "dict_6x6_250": cv.aruco.DICT_6X6_250,
        "dict_6x6_1000": cv.aruco.DICT_6X6_1000,
        "dict_7x7_50": cv.aruco.DICT_7X7_50,
        "dict_7x7_100": cv.aruco.DICT_7X7_100,
        "dict_7x7_250": cv.aruco.DICT_7X7_250,
        "dict_7x7_1000": cv.aruco.DICT_7X7_1000,
        "dict_aruco_original": cv.aruco.DICT_ARUCO_ORIGINAL
    }
    list_marker_size = [50, 100, 250, 1000, 1024]

    int_image_divisor = 10
    if int_image_width >= int_image_height:
        float_image_quotient = int_image_height / int_image_divisor
        int_image_size_min = int_image_height
    else:
        float_image_quotient = int_image_width / int_image_divisor
        int_image_size_min = int_image_width

    list_difference = []
    for i in range(len(list_marker_size)):
        list_difference.append(abs(float_image_quotient - list_marker_size[i]))

    int_marker_size = list_marker_size[list_difference.index(min(list_difference))]

    if int_image_size_min <= 720:
        str_marker_grid = "7x7"
    elif 720 < int_image_size_min <= 1080:
        str_marker_grid = "6x6"
    elif 1080 < int_image_size_min <= 1440:
        str_marker_grid = "5x5"
```

```
        elif 1080 < int_image_size_min <= 2160:
            str_marker_grid = "4x4"
        else:
            str_marker_grid = "aruco"

        if int_marker_size == 1024:
            str_marker_type = "dict_" + str_marker_grid + "_original"
        else:
            str_marker_type = "dict_" + str_marker_grid + "_" + str(int_marker_size)

        dict_marker =
    cv.aruco.getPredefinedDictionary(dict_marker_dict[str_marker_type])
        array_marker = np.zeros((int_marker_size, int_marker_size, 1), "uint8")

        int_pos_x = [4, (int_image_width - int_marker_size) - 4]
        int_pos_y = [4, (int_image_height - int_marker_size) - 4]
        for i in range(len(int_pos_x)):
            for j in range(len(int_pos_y)):
                int_marker_id = np.random.randint(0, int_marker_size - 1)
                cv.aruco.generateImageMarker(dict_marker, int_marker_id, int_marker_size,
    array_marker, 1)

                array_image[int_pos_y[j]:int_pos_y[j] + array_marker.shape[0],
                int_pos_x[i]:int_pos_x[i] + array_marker.shape[1]] = array_marker

        cv.imwrite(str_image_path, array_image)

        return dict_marker
```

Prilog IV: Programski kod za detekciju ArUco oznaka (detect.py)

```
import cv2 as cv


def marker_detect(array_image, int_image_height, array_marker_corners,
array_marker_ids):
    if len(array_marker_corners) > 0:
        array_marker_ids = array_marker_ids.flatten()

        for (markerCorner, markerID) in zip(array_marker_corners, array_marker_ids):
            corners = markerCorner.reshape((4, 2))
            (top_left, top_right, bottom_right, bottom_left) = corners

            top_right = (int(top_right[0]), int(top_right[1]))
            bottom_right = (int(bottom_right[0]), int(bottom_right[1]))
            bottom_left = (int(bottom_left[0]), int(bottom_left[1]))
            top_left = (int(top_left[0]), int(top_left[1]))

            cv.line(array_image, top_left, top_right, (0, 255, 0), 2)
            cv.line(array_image, top_right, bottom_right, (0, 255, 0), 2)
            cv.line(array_image, bottom_right, bottom_left, (0, 255, 0), 2)
            cv.line(array_image, bottom_left, top_left, (0, 255, 0), 2)

            center_x = int((top_left[0] + bottom_right[0]) / 2.0)
            center_y = int((top_left[1] + bottom_right[1]) / 2.0)

            cv.circle(array_image, (center_x, center_y), 4, (0, 255, 0), -1)

            str_text = "ID" + str(markerID)
            if top_left[1] > int_image_height / 2:
                cv.putText(array_image, str_text, (top_left[0] - 3, top_left[1] - 5),
                        cv.FONT_HERSHEY_SIMPLEX, 0.75, (255, 0, 0), 2)
            else:
                cv.putText(array_image, str_text, (top_left[0] - 3, bottom_left[1] + 21),
                        cv.FONT_HERSHEY_SIMPLEX, 0.75, (255, 0, 0), 2)

    return array_image
```