UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miha Kramberger

Iskanje prometnih znakov s programom na operacijskem sistemu Android

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Tomaž Dobravec

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Pameten telefon lahko uporabimo tudi za pomoč pri vožnji z avtomobilom. Podatke, ki jih ta zajame s pomočjo kamere in snemalnika zvoka, lahko analiziramo ter predelane posredujemo vozniku in na ta način omogočimo prijetnejšo in varnejšo vožnjo. V diplomskem delu se osredotočite na področje zaznavanja prometnih znakov za omejitev hitrosti iz posnetih fotografij. Preglejte obstoječe pristope in razvijte svojo rešitev za učinkovito zaznavanje omenjenih prometnih znakov. Algoritem vgradite v samostojen program za operacijski sistem Android. Program naj med vožnjo z avtomobilom s kamero opazuje okolico in na zaslon pametnega telefona prikazuje zadnji veljavni prometni znak za omejitev hitrosti.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Miha Kramberger sem avtor diplomskega dela z naslovom:

Iskanje prometnih znakov s programom na operacijskem sistemu Android.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Tomaža Dobravca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 2. februarja 2016 Podpis avtorja:

Zahvaljujem se mentorju, doc. dr. Tomažu Dobravcu, za nasvete in napotke pri izdelavi diplomskega dela.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
GPS	Global Positioning System	Globalni sistem za pozicijo
API	Application Programming Interface	Vmesnik za programiranje aplikacij
TADP	Tegra Android Development Pack	Tegra Android paket za razvoj
RGB	Red, Green, Blue color space	Rdeča, zelena, modra
OCR	Optical Character Recognition	Vizualno prepoznavanje teksta
XML	Extensible Markup Language	Razširljiv označevalni jezik
HSV	Hue, Saturation, Value color space	Odtenek, nasičenost, vrednost
IDE	Integrated Development Environment	Integrirano razvojno okolje
JDK	Java Development Kit	Razvojno okolje za Javo
ADT	Android Development Tools	Razvojno okolje za Android
CDT	C/C++ Development Tooling	Eclipse orodja za C/C++

Povzetek

Naslov: Iskanje prometnih znakov s programom na operacijskem sistemu Android

Cilj diplomske naloge je razvoj programa za zaznavanje prometnih znakov za hitrost na platformi Android, ki ga je z minimalnimi dodelavami mogoče razširiti tudi na druge mobilne operacijske sisteme. Program je namenjen vsem udeležencem prometa z avtomobili in ostalimi prevoznimi sredstvi, kjer je možna namestitev tabličnega računalnika ali telefona, tako da voznika ne ovira pri vožnji. Program je primeren predvsem za voznike, ki podobnih sistemov še nimajo vgrajenih v avtomobil. Diplomska naloga obsega program, ki je pretežno napisan v programskem jeziku Java za operacijski sistem Android, uporabljena orodja in njihov kratek opis ter postopki, ki so potrebni za pravilno delovanje programa. Opisana je tudi problematika zaznavanja znakov in načini, ki omogočajo zaznavanje, ter rešitve, uporabljene v obstoječih sistemih. Za zaznavanje znakov se uporablja kamera na napravi, za procesiranje dobljenih podatkov pa knjižnica OpenCV. Program ne uporablja drugih tehnologij za pomoč pri zaznavanju in ni povezan z spletom ali sistemom GPS. Namenjen je daljši uporabi v avtomobilu, zato je za avtonomijo baterije poskrbljeno s pravilno izbiro barve ozadja, ki prekriva večino zaslona, in z zajemom slik v manjši resoluciji. Med uporabo voznik postavi napravo v stojalo in jo obrne v smeri vožnje tako, da kamera ni zakrita in je rahlo obrnjena v desno smer proti znakom.

Ključne besede: program Android, zaznavanje znakov

Abstract

Title: Detecting traffic signs with a programme on the Android operating system

The objective of the diploma thesis is to develop a speed traffic signs detection application on the Android platform, which could be extended to other mobile operating systems with minimal adjustments. The programme is intended for all vehicles and other transport means in traffic, where the tablet or phone can be assembled so that does not disturb the driver at driving. The programme is appropriate for drivers who have not had similar systems installed in their cars. The diploma thesis presents a programme that is mostly prepared in the Java programming language for the Android operating system, used tools and a short description as well as procedures required for proper operations of the programme. The thesis also explains the sign detection issues and methods used to enable detection, as well as solutions used in the existing systems. Signs are detected by a camera set on the device; acquired data are processed by the OpenCV library. The programme does not use other technologies for detection and is not connected to the GPS system. The programme is intended for a long-term use in the vehicle, therefore, the battery autonomy is provided with the appropriate selection of the background colour that covers most of the screen, and with imaging in lower resolution. During the use, the driver set the device in a holder and turns it in the direction of driving so that the camera is not covered and slightly directed to the right side facing the signs.

Key words: Android programme, sign detection

Poglavje 1

Uvod

V današnjem sodobnem in visoko tehnološkem času se mobilni telefoni ne uporabljajo le za telefoniranje in pisanje tekstovnih sporočil, marveč za veliko različnih namenov. Mobilni telefoni postajajo vse bolj uporabni na različnih področjih, ki nam omogočajo, da namesto dragih namenskih naprav za to uporabimo pametni telefon ali tablični računalnik. Cilj diplomske naloge je razviti program za pametne telefone in tablične računalnike, ki namesto namenske naprave ali enote GPS le s pomočjo kamere zaznava prometne znake za hitrost in naselja, saj ta določajo tudi največjo dovoljeno hitrost. Namesto drage naprave GPS, ki kaže tudi podatke o največji dovoljeni hitrosti, te podatke pa pridobi iz predhodno naloženih zemljevidov, ki niso nujno ažurni in točni, se za prikaz največje dovoljene hitrosti lahko uporabi kamera na pametnem telefonu ali tabličnem računalniku. Tovrstne naprave morajo biti vsaj iz srednjega cenovnega razreda, saj imajo te običajno vgrajene kamere z zadostno resolucijo in kakovostjo, ter uporabljajo vsaj 2- ali 4-jedrne procesorje.

Program, izdelan v okviru diplomske naloge, deluje na osnovi zaznavanja oblik, preizkušeno pa je bilo tudi zaznavanje z barvo. Solidno uspešno zaznavanje sem dosegel s pregledovanjem že obstoječih rešitev. Pri večini rešitev, ki sem jih pregledal, so avtorji za osnovo uporabili algoritem Hough, barve so uporabljali kot pomožno sredstvo ali pa jih sploh niso uporabili, saj sliko najprej pretvorijo v sivinsko. Sklepna ugotovitev enega izmed avtorjev je, da postavitev kamere ni ključna in je hitrost algoritma za zaznavanje dovolj hitra, da znakov ne zgreši ter da deluje solidno v vseh vremenskih pogojih [1]. Moj sklep je podoben, namreč, da ni potrebno zelo hitro zajemanje slik, saj algoritem deluje dovolj hitro kljub premikanju avta, in deluje podobno dobro v vseh vremenskih pogojih, dokler je svetloba zadostna, kar pomeni, da ponoči zaznavanje ni praktično uporabno. Razlika med algoritmoma je morda le v tem, da je glede na opis avtorja moj algoritem za zaznavanje oblik malo manj uspešen.

Uspešnost zaznavanja oblik na sliki je pogojena s kakovostjo slike in njeno resolucijo, najbolj pomembno pa je, da slika ne vsebuje šuma, saj ta povzroči lažne zadetke. Čeprav iskanje enostavnih geometrijskih oblik po navadi ne predstavlja problemov, je pri prometnih znakih drugače, ker se pojavijo v rahlo popačenih pravilnih geometrijskih oblikah. To je treba upoštevati tako, da je krog lahko elipsa, kvadrat pa pravokotnik. Pri mojem programu je predvsem pomembno zaznavanje elips. Osnovne metode, ki se uporabljajo za zaznavo oblik, temeljijo na algoritmu Hough, za bolj specifične oblike pa se uporablja kaskada Haar (angl. *Haar cascade*). V pomoč je tudi zaznavanje barv, vendar moramo biti pozorni na barvni sistem.

Poglavje 2

Zaznavanje prometnih znakov

Zaznavanje prometnih znakov v realnem času je zelo koristno, saj opozori in prikaže hitrost na znaku, ki ga voznik morda spregleda. Poleg tega opazovanje znakov odvrne pozornost voznika od vožnje in spremljanja ostalega prometa, če se je potrebno na znake preveč pogosto ozirati. Program, ki to delo opravlja namesto voznika, pripomore tudi k večji varnosti v prometu [3]. Zaznavanje običajno poteka v treh korakih: predhodna obdelava podatkov, detekcija in prepoznava [3]. Moj program se nanaša na zaznavanje znakov skozi prednje steklo s pomočjo pametnega telefona ali tabličnega računalnika. Programi za ta namen sicer že obstajajo, saj so nekateri že vgrajeni v avtomobilih. Za izdelavo takšnega programa sem se odločil, da bi se na ta način bolje spoznal z okoljem Android, ki ga ne uporabljam redno in ga pred izdelavo niti nisem imel možnosti doma preizkusiti z vidika programiranja, zato prav v ta ta namen kupil tablični računalnik. Malo izkušenj sem sicer pridobil na fakulteti, vendar bolj na področju osnovnih aplikacij. Poleg želje po pridobivanju novih izkušenj sem se za tak program odločil, da bi nekomu, ki nima avtomobila z že vgrajenim podobnim sistemom, omogočil priložnost preizkusiti ta program na njegovem ali njenem telefonu, ne glede na tip in starost avta.

Največji napredek na področju zaznavanja in prepoznavanja znakov je bil dosežen v zadnjem desetletju. Kar nekaj podjetij in raziskovalnih skupin se je lotilo te problematike in dosegli so kar nekaj dobrih rezultatov. V računalništvu to področje obsega umetno inteligenco in računalniški vid [2]. Univerzalen program, ki bi obsegal vse znake na svetu ali vsaj večino, bi moral imeti zelo obsežno zbirko podatkov, zato je večina obstoječih sistemov osredotočena na eno vrsto znakov, ki se uporabljajo v posamezni državi ali na celini. Čeprav so obstoječi algoritmi zelo dobri, je uporaba v resničnem svetu še precej nezanesljiva [3]. Pri zaznavanju znakov moramo predvideti morebitne razlike med enakimi znaki, okolje v katerem se zaznava, paziti moramo na hitrost zaznavanja znakov. Med posameznimi znaki enakih pomenov se pojavljajo razlike predvsem v barvi, kar pa ni samo posledica lokacije znaka, ali je to v senci ali na soncu, ampak tudi v njegovi starosti. Starejši znaki so bledi, kontrast barve ni več tak kot je bil. Opazil sem, da so barve znakov, starejših od 20 let, rahlo drugačnega odtenka, vendar ne zato, ker so zbledeli, ampak je njihova barva drugačna že v osnovi, prav tako pa nimajo niti odsevnega sloja. To velja predvsem za znake z rdečo in rumeno barvo, znaki za hitrost imajo sicer zelo malo rdeče barve, ki je s prostim očesom ni mogoče opaziti, barvne vrednosti pa so lahko popolnoma drugačne. Enako velja tudi za rumene in ostale znake, ki pa niso tako pomembni. Večina uspešnih obstoječih sistemov zato uporablja sivinske slike in neodvisnost od barv [2].

2.1 Postavitev kamere

Postavitev kamere za zajem slike je zelo pomembna. Če jo usmerimo preveč v tla oziroma proti cesti, bi lahko zaradi tega izgubili znake, ki se pojavljajo na desni strani vozišča. V takem primeru bi ob slabši osvetlitvi kamero motile tudi luči nasproti vozečih avtomobilov. Enako velja, če bi bila kamera preveč obrnjena proti nebu, saj tako ne bi bilo zmanjšano le število morebitnih znakov, ampak bi v večini primerov, razen če uporabnik ne bi nastavitev izvedel drugače, naprave zaznavale svetlost v središču slike, nekatere dražje naprave tudi iz celotne slike. Avtomobili z že vgrajenim tovrstnim sistemom imajo po navadi več kamer, ki so nameščene na zunanji strani avtomobila na različnih lokacijah pod različnimi koti. V primeru, kjer je na voljo le ena kamera na uporabnikovi napravi, mora ta biti postavljena čim bolj učinkovito. Končno odločitev, kako je postavljen telefon ali tablični računalnik v avtomobilu, seveda sprejme voznik. Brez stojala ali držala ni mogoče učinkovito zaznavati znakov. Večina voznikov ima napravo nameščeno na stojalu ali držalu, ki pa rahlo obrnjena proti vozniku, kar se pri mojem programu od voznika tudi pričakuje. Zaradi zaznavanja oblik je pomembno, ali se znak pojavi v kameri daleč stran ali pa tik preden se peljemo mimo. Pomembno je, da so znaki v pravem trenutku obrnjeni pravokotno na kamero, razen če na znak vpliva zunanji dejavnik, vendar pa takih primerov ni mogoče predvideti in jih rešiti zgolj s postavitvijo kamere [4].

2.2 Vrste zaznavanja

Prometne znake lahko zaznavamo na veliko načinov, najbolj priljubljeno pa je zaznavanje z oblikami, prepoznavanjem barv ali kombinacija obojega. Poleg tega se lahko izvaja še prepoznavanje številk in besedila. Pri oblikah moramo ločevati predvsem okrogle in pravokotne, ponekod še trikotne in osemkotne oblike za znak stop. V mojem primeru pridejo v poštev znaki, ki določajo hitrost, ki pa so vsi okrogli, z izjemo območja za hitrost, vendar je tam znak za hitrost narisan znotraj kvadrata. V poštev pridejo tudi znaki za naselja, ki so rumeni in pravokotne oblike.

2.2.1 Zaznavanje barv

Znaki so v večini primerov obarvani tako, da jih voznik lažje opazi. Večinoma so prevlečeni z odsevno barvo, da so lažje vidni tudi ponoči. Ta odsevnost zaradi lastnosti kamer, ki so vgrajene v naprave, onemogoča zaznavanje znakov ponoči. Kamera po navadi prestavi način v črno belo, zaradi odsevnosti pa nastane le območje svetlobe oziroma bele barve. Barve znakov so običajno takšne, da izstopajo od barv okolja. Ponekod je ta barva sicer zbledela ali pa je zaradi zunanjih dejavnikov videti drugačna. Dolga izpostavljenost soncu uniči barvo [2].

Pri zaznavanju znaka z detekcijo barv mora biti znak močnih kontrastnih barv v primerjavi z okoljem, sicer je ta detekcija lahko napačna ali pa je znak nemogoče zaznati. Zaznavanje samo s pomočjo barve je težavno, ker nekateri znaki vsebujejo zelo malo določene barve, po kateri jih je možno prepoznati. To so na primer znaki za konec naselja, kjer je v popolnoma rumeni tabli z črnim napisom le par tankih diagonalnih rdečih črt, ki jih je težko zaznati, še posebej iz daljave. Tudi znaki za hitrost imajo le rob rdeče barve, ki je relativno ozek glede na celoten znak. Pri zaznavanju z barvami je torej treba zajeti barve, ki so v znakih, ostale barve pa poskušati izpustiti. Običajno se zaznavanje barv deli na izboljšanje izvorne slike in nato še prepoznavanje. Znaki istega tipa so običajno obarvani na enak način. Barve znakov se spreminjajo glede na svetlobo v okolju. Čeprav gre v osnovi za enako barvo, se jo vidi drugače. Ta učinek se običajno poskuša delno odpraviti z izravnavo histogramov, popravljanjem vrednosti gamma in izravnavo kontrasta. Glavna težava teh izboljšav je, da delujejo v barvnem prostoru RGB. Za učinkovito delovanje v barvnem prostoru RGB bi potrebovali referenco, kakšna svetloba je tista prava svetloba, v kateri so vse barve enake. Takšne reference pa ni mogoče pridobiti, ker se svetloba zaradi zunanjih dejavnikov lahko spremeni v vsakem trenutku [5].

2.2.2 Zaznavanje oblik

Ker se znaki enake vrste med seboj razlikujejo po barvah, je smiselno tudi zaznavanje z oblikami. Zaznavanje oblik poteka s pomočjo algoritmov. Obstaja že veliko knjižnic za različne programske jezike, kjer je možno prepoznavanje raznih oblik, kot so krogi, kvadrati in podobno. Za krivulje, kroge, elipse in ostale oblike, ki niso definirane z enačbami, je najbolj primeren algoritem Hough. Algoritem deluje tako, da definira obliko objekta z črtami, ki se med seboj sekajo, v primeru kroga in ostalih objektov, ki vsebujejo krivulje, je teh zelo veliko [6]. Posebna različica tega algoritma pa se lahko uporabi za detekcijo robov, na primer pri trikotnih znakih, kjer se tri črte sekajo med seboj, kot med dvema pa je šestdeset stopinj. Tri črte tako tvorijo trikotnik. Težava se pojavi, ko je na sliki več trikotnikov, saj se lahko zgodi, da se več črt seka med seboj, ker njihova dolžina ni določena in tako same tvorijo trikotnike, čeprav jih dejansko ni. Problem se omili z izvajanjem algoritma nad posameznimi deli slike namesto nad celo sliko [1]. Ti algoritmi se lahko uporabijo tudi za zaznavanje znakov, saj so znaki sestavljeni iz osnovnih geometrijskih oblik. Zaznavanje oblik je sicer bolj zahtevno, vendar je tudi bolj točno in lažje je izločiti nepravilne zadetke. Če bi uporabili samo zaznavanje oblik, ne bi bilo učinkovito, saj znaki za hitrost niso edini znaki, ki so okrogli, pa tudi znaki za naselja niso edini pravokotne oblike. Pri oblikah je treba upoštevati kot in oddaljenost od kamere, saj je lahko znak na kameri popačen in ga algoritem ne bo zaznal. Ti algoritmi običajno ne prepoznajo zelo majhnih, kar sicer ni problem, če je kamera pravilno postavljena in je zajem slik dovolj pogost. Kljub vsemu je za zaznavanje oblik

potreben kontrast barv med znakom in okoljem, sicer ga algoritem ne bo zaznal. Običajno se slika primerno obdela, preden se čez njo spusti algoritem za zaznavanje oblik. Nekateri tipi algoritmov zahtevajo, da se iz slike izluščijo samo robovi. Algoritem CannyEdge je najbolj primeren za robove, ker obdrži tudi notranje robove in oblike, kar omogoča, da se obdrži oblika znaka. Robove se lahko izloči tudi z določanjem praga, vendar tako dobimo samo zunanje obrise, kar pa je manj natančno [1].

2.2.3 Zaznavanje s prepoznavanjem objektov

Ena izmed metod za zaznavanje s prepoznavanjem objektov je uporaba kaskade Haar (angl. *Haar cascade*). Kaskada Haar deluje tako, da poskuša na dani sliki prepoznati objekte, ki so v programu že vneseni oziroma se jih je program »naučil«. Prvi programi za detekcijo obrazov so delovali s Haar kaskadami. Ta način zaznavanja je v primerjavi z zaznavanjem barv in intenzivnosti barv posameznih slikovnih pik veliko bolj učinkovit in zahteva manj procesorske moči. Ta metoda zaznavanja pa sešteje intenzivnost slikovnih pik na določenem območju slike. Teh območij je več in so kvadratna. Metoda na koncu izračuna razliko seštetih vrednosti med posameznimi območji. Pred zaznavanjem je treba imeti shranjeno bazo slik, na primer obrazov. Na podlagi teh slik se potem določijo območja, nad katerimi metoda izračunava; v primeru obraza so to oči, ki so temnejše. Zunanji okvir, preko katerega se ne izračunava, je obraz. Ta metoda je sicer hitra, ampak za učinkovito delovanje potrebuje ogromno vnaprej vnesenih podatkov, s pomočjo katerih se nauči prepoznavanja, kar pomeni, da potrebuje veliko število slik za primerjavo, sicer je neučinkovita. V resničnem svetu se največkrat uporablja za zaznavanje predmetov, ki jih je lahko označiti s kvadratom ali pravokotnikom. Za zaznavanje krogov se metoda ne uporablja pogosto, saj postane učinkovitost celo slabša kot s preverjanjem vsake slikovne pike v sliki [22].

Knjižnica OpenCV že vsebuje tak tip zaznavanja. Imenuje se Cascade Classifier. Za učinkovito uporabo naj bi bilo smiselno uporabiti več sto slik določenega objekta. Te slike so zmanjšane na zelo majhno velikost 20 x 20 slikovnih pik. Ko je klasifikator naučen, se z njim preveri določeno območje. Če je to območje zelo verjetno, da vsebuje iskani objekt, klasifikator vrne vrednost 1, drugače pa 0. Za pregled celotne slike se okno, v katerem klasifikator išče, pomika po celotni sliki. Zaznavanje iskanih objektov različnih velikosti je možno z večanjem in manjšanjem klasifikatorja, kar je bolj učinkovito kot spreminjanje velikosti slike, zato je smiselno večkratno iskanje po sliki z različnimi velikostmi klasifikatorja. Pri OpenCV je klasifikator določen z datoteko XML, kjer so zapisani izračuni na podlagi vnaprej shranjenih slik [22].

Kaskadni klasifikator je možno uporabiti tudi za zaznavanje prometnih znakov, najbolj pride prav pri zaznavanju stop znakov in takih, ki so trikotne oblike. Možno ga je uporabiti tudi za zaznavanje okroglih znakov za hitrost, pri čemer bi moral upoštevati tudi številko, ki je zapisana v znaku, ne bi pa je natančno prepoznal. Z zaznavanjem okroglih znakov bi porabil preveč procesorske moči, poleg tega pa bi še vedno moral vključiti vsaj preverjanje barv znaka. Če bi bila naloga prepoznati vse znake v prometu, potem ta način zaznavanja pride zelo prav, v tem primeru, ko gre pri programu le za zaznavanje znakov za hitrost, pa zaradi neučinkovitosti in kompleksnosti izvedbe takega preverjanja nisem uporabil.

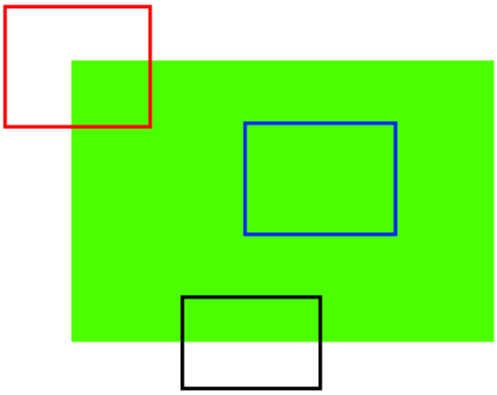
2.2.4 Prepoznavanje besedila

Ena od možnosti zaznavanja znakov je tudi zaznavanje besedila, pri čemer je optimalno, če se objekt predhodno zazna in nato analizira besedilo. Z zaznavanjem teksta bi tako lahko razločil znake med seboj in izbral prave. Če pa je zaznavanje besedila dobro implementirano, pa bi celo lahko izpisoval imena krajev in podobno. Za branje besedila obstaja nekaj knjižnic, vendar je le redko katera namenjena za Android. Učinkovite knjižnice so zahtevne za uporabo, problem pa je tudi z licencami. Tako prepoznavanje se imenuje »Optical Character Recognition« (OCR), ki pa pomeni pretvorbo na roko napisanih ali natiskanih besedil v obliko, ki jo prepozna računalnik. Najbolj običajen primer je zajem dokumenta s kamero na telefonu, ki ga potem pretvori v besedilo, ki ga je mogoče urejati. Prve različice souporabljale, podobno kot zaznavanje objektov, veliko slik raznih črk, s katerimi se je program naučil. Obstajata dva algoritma, ki se običajno uporabljata. Eden je zasnovan na matrikah in pregleda vsako slikovno piko med dano sliko in shranjenimi podatki. Ta tehnika se najpogosteje uporablja za tiskane črke. Drugi pa je zasnovan na zaznavanju določenih potez na sliki, črt, krivulj, robov in podobno. Ta se najpogosteje uporablja za črke in besedila, napisana na roko. Rezultate je možno naknadno izboljšati, tako da se vnaprej določi, kakšne črke so dovoljene, uporabno je za posamezne jezike, kjer se izločijo črke, ki niso v abecedi. Zaznavanje z OCR je sicer dokaj natančno, vendar so knjižnice za programiranje velike, zato nisem uporabil nobene posebne knjižnice. Ena takih knjižnic je Tesseract.

Sama knjižnica OpenCV ne ponuja rešitve za OCR, vendar z verzijo 3.0 vključuje možnost interakcije s knjižnico tesseract-ocr. V mojem primeru je uporabljena starejša verzija knjižnice OpenCV, zato tega ni na voljo, te možnosti pa ne bi uporabil, tudi če bi bila na voljo, saj bi bil program preveč kompleksen. Je pa možno primitivno zaznavanje črk z uporabo funkcije za zaznavanje objektov features2d, le da to uporabimo za namen zaznave črk. Tak način ni zelo učinkovit in zanesljiv.

2.2.5 Zaznavanje s primerjavo objektov

Zaznavanje s primerjavo objektov je morda najbolj učinkovita metoda. Gre za podobno metodo kot pri sestavljanki. Računalnik primerja sliko in kos slike, objekt, ki ga vzame kot del sestavljanke in mu poskuša najti svoje mesto na sliki. Pri tej vrsti zaznavanja gre za detekcijo značilk na slikah kot so robovi, koti, hitre spremembe barv in podobno. Podobna tehnika se uporablja tudi pri zaznavanju obrazov. Značilke na sliki je najbolje razložiti na primeru, kot ga prikazuje Slika 2.1 [7].



Slika 2.1: Prikaz zaznavanja značilk [7].

Modri pravokotnik ne vsebuje nobenih značilk, saj je cel obarvan z eno barvo. Lahko ga premaknemo rahlo v desno in levo ali gor in dol, še vedno bo cel obarvan enako, zato se tak primer ne šteje kot značilka. Črn pravokotnik prikazuje rob. Pravokotnik lahko premaknemo levo in desno, v tem primeru ne bo nobenih sprememb, če ga premaknemo gor ali dol, pa se vrednost barv spreminja. Črn pravokotnik je že značilka, vendar ne najboljša. Rdeč pravokotnik na zgornji sliki je najboljši primer. Prikazuje kot, lahko ga premikamo v poljubno smer, vrednost barv se bo spreminjala. Taka značilka velja za najbolj učinkovito.

Iskanje značilk poteka tako, da program poskuša najti na sliki območja, kjer se z minimalnim premikom okna, v katerem išče, pojavi čim več sprememb, tam je tudi največja možnost, da obstaja kot ali vsaj rob (Slika 2.2). Temu delu se reče zaznavanje značilk (ang. *Feature Detection*). Naslednji del je poiskati enake značilke na drugi sliki, tako da program pogleda okolico okoli zaznane značilke in si tako ustvari njen opis (ang. *Feature Description*), da jo lažje najde na drugi sliki. Ko imamo značilko in njen opis, jo lahko najdemo na drugi sliki, lahko jo tudi obračamo, če je na eni izmed slik drugače obrnjena.

Knjižnica OpenCV ima za iskanje in opis značilk več različnih algoritmov. Med njimi so pomembnejši:

- SIFT (Scale-Invariant Feature Transform),

- SURF (Speeded-Up Robust Features),

- FAST(Features From Accelerated Segment Test),

- BRIEF (Binary Robust Independent Elementary Features),

- ORB (Oriented FAST and BRIEF).

Poleg teh obstajajo še: Harris Corner Detection, MSER, BRISK, FREAK, ki pa se redkeje uporabljajo.

Slika 2.2: Primer prepoznavanja s primerjanjem [19].

2.3 Optimalna izbira zaznavanj

Najbolj zanesljivo bi bilo izbrati kar vse vrste zgoraj omenjenih zaznavanj, kar seveda nima smisla in ni niti izvedljivo v resničnem času. Zato moramo pametno izbrati, kaj bomo uporabili in česa ne. Za zaznavanje okroglih in pravokotnih znakov je smiselno izbrati zaznavanje preprostih oblik, krogov in pravokotnikov oziroma kvadratov. Tako program pokriva vse možne znake, ki jih potrebujem. Za izločanje znakov za hitrost in znakov za naselje je smiselno uporabiti zaznavanje rumene in rdeče barve. Zaznavanje objektov ne pride v poštev zaradi kompleksnosti in neučinkovitosti zaznavanja okroglih oblik. Zaznavanje besedila prav tako ni uporabno v mojem primeru, saj program ne prikazuje imen naselij, prikaz vrednosti znaka pa je rešen tako, da se prikaže sam zaznani znak. Moj program

uporablja kombinacijo zaznavanja preprostih oblik, elips in prepoznavanja točk. Ta kombinacija je še dovolj učinkovita in ni preveč kompleksna za programiranje, prav tako pa ni preveč potratna glede sredstev, ki so na voljo za omogočanje zaznavanja v resničnem času. Rešitev ni popolna, je pa zadovoljiva za uporabo.

Poglavje 3

Programska oprema in tehnologije

V tem poglavju so predstavljene vse tehnologije in programska oprema, ki so bile uporabljene pri izdelavi programa. Programski jezik Java kot glavni programski jezik, razvojno okolje Eclipse ter knjižnica Open CV, ki se uporablja za obdelavo slik in videoposnetkov, ki so že bili posneti ali se snemajo v resničnem času. Predstavljen je tudi operacijski sistem Android, za katerega je namenjen program, in knjižnica OpenCV, ki je uporabljena pri programiranju.

3.1 Android

Android je operacijski sistem za mobilne naprave, telefone in tablične računalnike in je zasnovan na Linuxovem jedru. Android je v namenjen predvsem za pametne telefone z zaslonom na dotik, čeprav se ga uporablja v vseh vrstah naprav kot so tablični računalniki, pametne ure, televizije in kot operacijski sistem v avtomobilih. Android je sicer odprtokodni sistem, a je v lasti podjetja Google. Ker je odprtokodni sistem, je velik del programov za prenos na voljo brezplačno. Velik delež podjetij ga tako uporablja kot operacijski sistem za svoje mobilne naprave, saj jim izbira operacijskega sistema ne predstavlja dodatnih stroškov. Odprtokodnost pa je povzročila tudi, da je sistem zelo priljubljen med razvijalci. Prva različica Androida je bila 1.0, zadnja različica v času pisanja diplomske naloge pa je 6.0. Za prenos programov se uporablja Google Play Market, včasih znan kot Android Market. Nekateri programi so brezplačni, drugi pa so plačljivi. Za nakup običajno zadostuje račun PayPal. Android je med vsemi operacijskimi sistemi tudi najbolj razširjen, ostala dva glavna operacijska sistema sta Blackberry in iOs podjetja Apple. Pri razvoju programov z Androidom si lahko pomagamo z emulatorjem, kot je prikazan na sliki 3.1, ali pa z napravo Android, ki jo povežemo z računalnikom. Emulator sicer lahko nadomesti nekaj funkcionalnosti prave naprave, ampak je po mojih izkušnjah v večini primerov počasnejši. Uporaben je predvsem za izdelovanje grafičnih vmesnikov, kjer tekoče delovanje ni tako nujno potrebno. Sam emulatorja nisem uporabljal, saj knjižnica OpenCv zahteva predhodno naložen manager OpenCV, ki pa se na pravi napravi brezplačno namesti prek Google Play Market, emulator pa zahteva ročno namestitev, ki mi je povzročala težave in nezdružljivost z emulatorjem [8].

Slika 3.1: Emulator operacijskega sistema Android [18].

3.2 Java

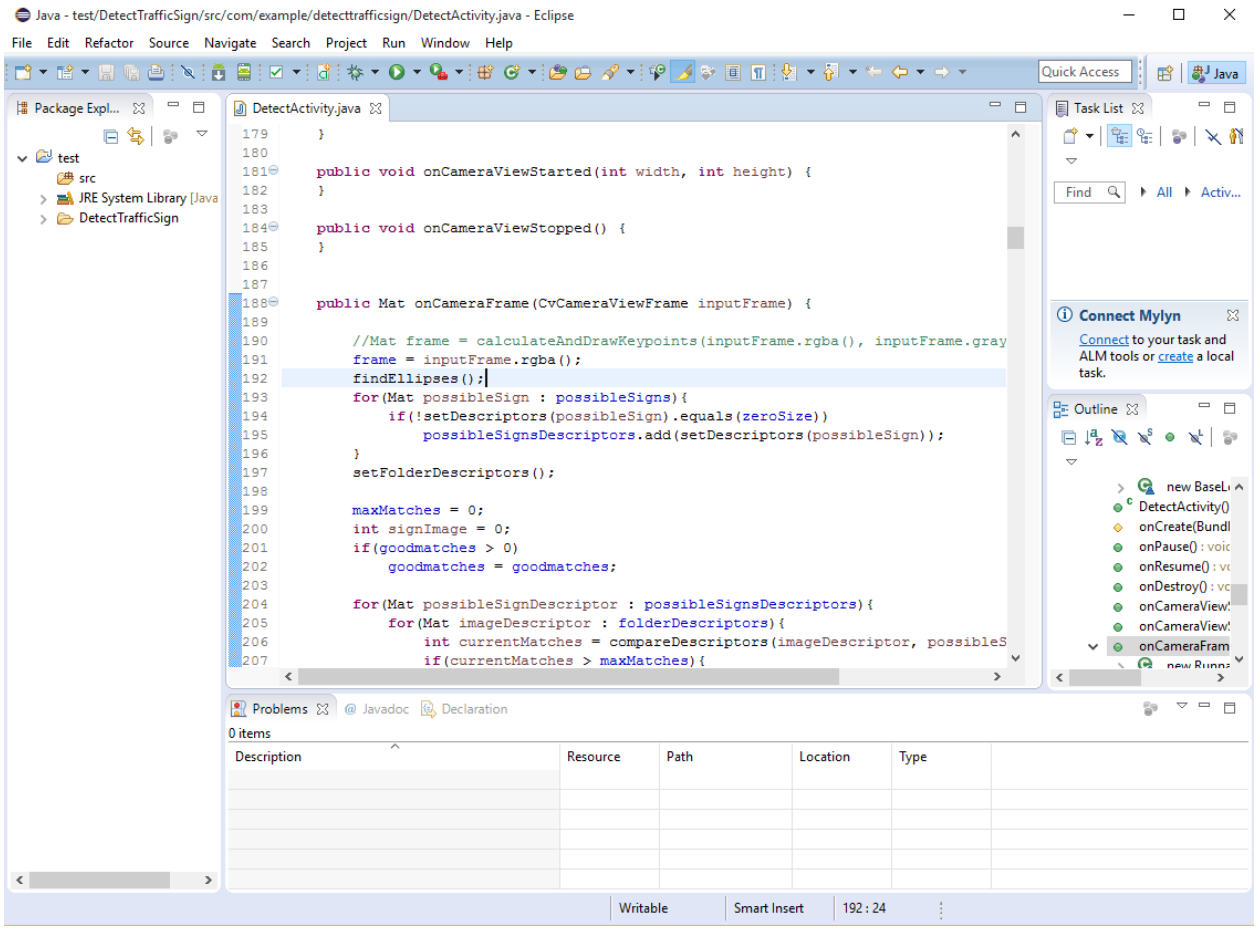
Java je objektno orientiran programski jezik. Je prenosljiv med različnimi sistemi in ga je možno poganjati na vseh sistemih, ki podpirajo Javo. Prva različica JDK 1.0 je izšla leta 1996, zadnja različica v času pisanja diplomske naloge pa je Java SE 8. Poznamo več podrazličic Jave, ki so namenjene drugim operacijskim sistemom: Java Card, ki se uporablja za pametne kartice, Java Platform Micro Edition (Java ME), ki se uporablja za sisteme z omejenimi viri, Java Platform Standard Edition (Java SE), ki se uporablja v bolj vsakdanjih okoljih, in pa Java Platform Enterprise Edition (Java EE), ki se uporablja za večje organizacije. Razlog, zakaj Javo lahko poganjamo na več sistemih, je, ker uporablja JVM. Sintaksa programiranja v Javi je podobna kot pri C++, hkrati pa je Java strogo objektno orientiran programski jezik in se brez razredov ne da narediti praktično ničesar. Programiranje v Androidu je skoraj izključno v Javi, s pomočjo NDK se sicer lahko piše tudi v C in C++, vendar je možno narediti vse ali pa vsaj večino funkcionalnosti zgolj v Javi. Za lažje programiranje Java vsebuje veliko svojih knjižnic in podatkovnih struktur, ki jih lahko

uporabimo. Z njimi je programiranje lažje. Sicer je splošno znano, da naj bi bila Java počasnejša od C++, vendar sem pri izdelovanju programa poizkusil določene elemente tudi s C++ in je hitrost približno enaka, odvisno od obremenjenosti pri izračunih za slike in video [9].

3.3 Eclipse

Program Eclipse je integrirano razvojno okolje (IDE). Napisan je v Javi in je namenjen programiranju v Javi. Uporabljamo ga lahko tudi za programiranje v drugih jezikih, vendar je za posamezne jezike potrebno namestiti vmesnike, na primer: Python, C, C++, Javascript in še veliko ostalih jezikov. Eclipse vključuje tudi Java Development Tools (JDT), to je vse, kar potrebujemo za začetek razvijanja v Javi. Prav tako je mogoče namestiti tudi pakete Eclipse CDT in Eclipse PDT za C in C++ ter PHP. V Eclipsu je mogoče razviti tudi svoje vmesnike, ki jih potem lahko namestijo tudi drugi uporabniki. Eclipse je mogoče prenesti in namestiti brezplačno in je odprtokoden program. Prva različica Eclipsa je bila 3.0 leta 2004, trenutna različica v času pisanja diplomske naloge je 4.5, napovedana pa je že različica 4.6 v letu 2016. Za razvoj v Androidu sam Eclipse ni dovolj, potrebno je namestiti za to namenjeno podrazličico, Eclipse ADT. Eclipse ADT je vmesnik podjetja Google, ki omogoča razvoj programov, namenjenih za Android. Tako se lahko razvija poleg kode v Javi še uporabniški vmesnik, dodaja dodatne knjižnice za Android, se izvaja »debug« (razhroščevanje) programa z uporabo Android SDK Tools in izvozi podpisano ali nepodpisano verzijo programa. Android SDK Tools se sicer v večini primerov ne uporablja več, nadomestil pa ga je Android Studio.

Za razvoj programa sem sprva uporabljal Android Studio, vendar sem za večino programa in dokončanje uporabil Eclipse in vmesnik za Android, saj je ta integriran in že nameščen z vsemi potrebnimi knjižnicami za uporabo, s paketom Nvidia CodeWorks for Android. Slika 3.2 prikazuje odprt program Eclipse z odprtim testnim projektom v začetni fazi izdelovanja diplomske naloge [10][11].



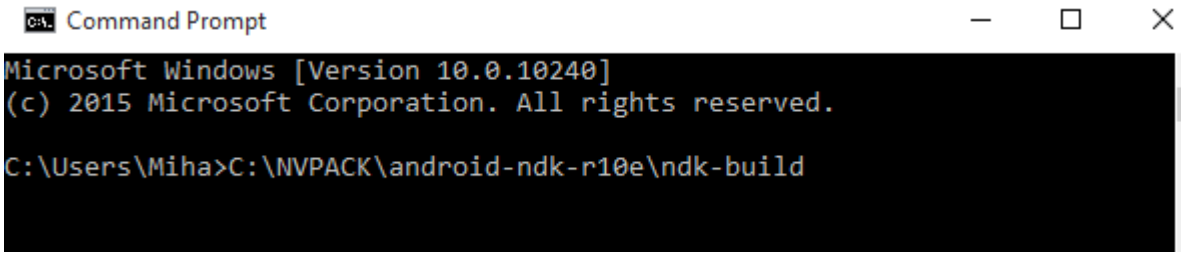
Slika 3.2: Primer okolja v Eclipsu.

3.4 Android Studio

Android Studio je integrirano razvojno okolje (IDE), namenjeno razvoju programov za Android. Na voljo je od leta 2013 za operacijske sisteme Windows, Linux in Mac OS X. Nadomestil je vmesnik za Android v Eclipsu (ADT) in je glavno razvojno okolje za programiranje programov v Androidu [12].

3.5 Android NDK

Android NDK je orodje, ki omogoča, da posamezne dele programa implementiramo v jeziku C ali C++. Običajno se uporablja tam, kjer je potrebna velika moč procesorja in programi zahtevajo zmogljivo napravo. Uporablja se pri procesiranju signalov in gonilnikov za igre in simulacije. Pri uporabi Android NDK je potrebno upoštevati, da ni nujno, da bodo zaradi tega programi delovali hitreje. Uporaba je priporočljiva le tam, kjer ni druge izbire, kot da uporabimo jezik C ali C++. Če je mogoče enako funkcionalnost doseči v Javi, se uporabi ta. Prevajanje kode poteka preko ukazne vrstice (Slika 3.3) [13].



Slika 3.3: Prevajanje C++ kode z ndk-build.

3.6 OpenCV

OpenCV je odprtokodna knjižnica, ki se najpogosteje uporablja za računalniški vid. Namenjena je kot osnova za programe, ki uporabljajo računalniški vid in uporabo v komercialnih produktih. Knjižnica vsebuje ogromno število algoritmov. Nekateri algoritmi so klasični, nekateri pa so narejeni tako, da se strojna oprema iz prejetih podatkov »uči« in se primerno odzove na prejete podatke. S pomočjo knjižnice je mogoče prepoznati obraze in prešteti, koliko jih je na sliki, razdeliti ali združiti več slik, poiskati podobnosti med slikami, poiskati objekte in jih izrezati na slikah, slediti očem, prepoznati okolje in še veliko ostalih podobnih stvari. OpenCV je na voljo za programiranje v C, C++, Python, Java in Matlab, da pa se jo uporabiti tudi v C#. Knjižnica je v celoti napisana v C++. Poizkusne alfa in beta različice so se vrstile od leta 2000 pa do 2005, ko je izšla prva stabilna različica 1.0, najnovejša različice knjižnice v času pisanja diplomske naloge pa je 3.0. Uporabiti jo je mogoče na več različnih platformah: Windows, Linux in Mac OS X, uporabiti pa jo je možno tudi na mobilnih platformah. Knjižnica je za mobilne platforme sicer rahlo prilagojena in ne ponuja popolnoma vseh funkcionalnosti, ki jih podpira polna različica [16].

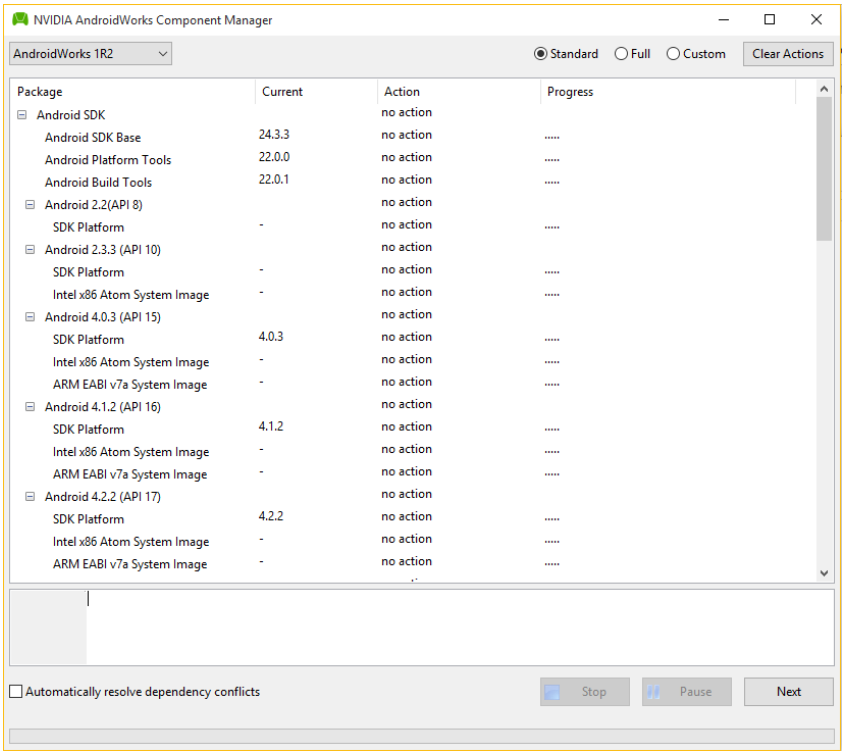
3.6.1 OpenCV Android development

Na spletni strani OpenCV je možno zaslediti podrobna navodila za namestitev okolja, potrebnega za razvijanje programov v Androidu. Velik poudarek je tudi na namestitvi okolja v Linuxu. Sam sem sicer skozi celotno delo uporabljal Windows 8.1 in Windows 10. Za uporabo mora biti nameščena Java, oziroma Java JDK in Android SDK, kjer morajo biti nameščeni vsi paketi za delo z Androidom, obvezna sta paketa Tools in pa vsaj ena izmed različic Androida (te različice lahko uporabimo za emulator Androida). Različica mora biti enaka ali nižja od različice, ki je nameščena na tabličnem računalniku ali telefonu, tako kot pri vsakem drugem programu za Android. Tu je pomembno poudariti, da knjižnica OpenCV za Android deluje na napravah, ki imajo nameščen Android 2.2 API ali novejši. Kljub temu, če razvijamo za starejšo različico od 3.0, moramo izbrati Android 3.0 API, saj se drugače koda, ki vsebuje OpenCV, ne bi prevedla. Nameščen mora biti tudi Eclipse, skupaj z vmesnikom

ADT, ki ga je mogoče namestiti kar v samem programu pod nastavitvami »Install new Software« (namesti novo programsko opremo). Ko je to nameščeno, se lahko ustvari še emulator s predhodno nastavljenimi nastavitvami, kot jih imajo pravi telefoni in tablični računalniki ali z poljubnimi nastavitvami. Ti nameščeni programi omogočajo razvoj samo v Javi, pri Androidu pa imamo še dodatno možnost delnega razvoja programa v C in C++. Zato je potrebno namestiti še Android NDK in Eclipse CDT, ki je po navadi v Eclipsu že nameščen. Kodo, ki je napisana v C in C++, je potrebno prevesti posebej in se ne prevede ob ukazu "Build" v Eclipsu. To je možno narediti na več načinov: preko programa CygWin, ki nima podpore pri OpenCV, ali preko ukazne vrstice z ukazi, ki jih moramo vnesti, da se koda prevede pa so napisani na spletni strani OpenCV, tretja možnost pa je kar v Eclipsu s pomočjo vmesnika CDT. Sam sem manjši del kode v C in C++ prevedel v ukazni vrstici, ker je postopek preprost. Sicer se lahko poganja tako kodo tudi v emulatorju, ampak pri mojem programu mi ni uspelo usposobiti C++ kode na emulatorju, zato ga skorajda nisem uporabljal [14][15].

3.7 Nvidia Codeworks

Nvidia Codeworks for Android je paket za razvoj programov v Androidu, ki vsebuje vse zgoraj omenjene pakete in programe ter še več, kar je primernega za razvoj programov. Paket vsebuje Eclipse, vmesnik CDT, Android Tools, Android NDK, knjižnico OpenCV in še mnoge druge, ki pa jih jaz nisem potreboval. Namestitveni paket nadomesti AndroidWorks in Tegra Android Development Pack (TADP). Ob namestitvi je potrebno le obkljukati, katere pakete in programe želimo imeti nameščene (Slika 3.4). V tej fazi je že možno izbrati tudi različice Androida, ki jih želimo imeti nameščene. Izbral sem Eclipse (ne najnovejšo različico), knjižnico OpenCV za Android, Android 4.4.2 API, Android NDK in dokumentacijo, shranjeno lokalno na disku. Nobenega od paketov ni treba posebej uvažati ali vnašati sistemskih spremenljivk pri Android NDK, vse se namesti popolnoma avtomatsko. Če bi hotel imeti še kakšno verzijo Androida več, je mogoče le zagnati program za izbiro paketov in obkljukati dodatne pakete ali odstraniti tiste, ki jih ne potrebujemo [16].



Slika 3.4: Paketi NVIDIA Codeworks.

Poglavje 4

Razvoj programa

V tem poglavju je predstavljen potek in razvoj programa. Predstavljen je tudi grafični vmesnik, ki je sicer zelo preprost, saj je bistvo zajeto v eni sami sliki, ki se prikazuje na zaslonu.

4.1 Zgradba programa

Program je zgrajen iz aktivnosti, kjer je glavna aktivnost ena. Program deluje sam zase, ne uporablja zunanjih naprav in se ne povezuje z internetom ali preko »bluetootha«. Prav tako ne uporablja povezave GPS, s katero bi lahko določil, kolikšna mora biti hitrost na delu ceste, kjer se peljemo. Celoten potek informacij v programu je tak, da kamera stalno deluje v video načinu, dejansko gre za zajem slike. Z videom je mogoče pridobiti od 3 do 7 informacij na sekundo, odvisno od zmogljivosti naprave. Slika se obdela, zazna znak na sliki, če je ta prisoten, ustrezno razloči, za kakšen znak gre in uporabniku na zaslonu prikaže izrezan znak iz slike, če gre za znak za hitrost ali umetno generirano sliko znaka, če gre za kakšen koli drug znak, ki ravno tako določa hitrost.

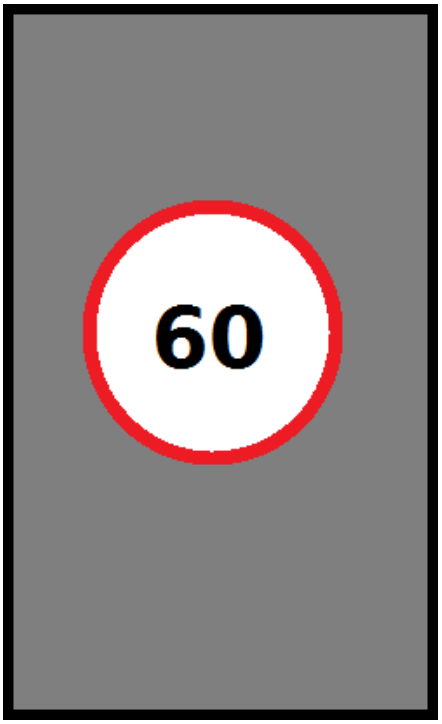
4.2 Zahteve programa

Namen je razviti program, ki v resničnem času zazna prometne znake in vozniku omogoča, da na zaslonu vedno vidi največjo hitrost, s katero mora peljati (Slika 4.1). Program je namenjen za uporabo na slovenskih cestah, pri čemer velja omeniti, da morajo biti znaki korektno in pravilno postavljeni

Zahteve so:

- zajem slike ali videa,
- zaznavanje znakov, ki določajo hitrost,
- zaznavanje znakov za naselje,
- prikaz dovoljene hitrosti na zaslonu,
- čim daljša avtonomija naprave z zagnanim programom,
- čim manjše obremenjevanje voznika med vožnjo,

- uporaba samo kamere in ne ostalih tehnologij,
- delovanje na čim več različnih napravah,
- uporaba le odprtokodnih knjižnic za izdelavo.



Slika 4.1: Predviden grafični vmesnik.

4.3 Načrt izdelave programa

Ob začetku izdelave se je bilo potrebno odločiti, na kakšen način in kako bo potekalo programiranje in testiranje. Prvi mejnik je bil spoznavanje okolja in Androida ter namestitev potrebne programske opreme. S knjižnico OpenCV so nas seznanili že na fakulteti pri predmetu Multimedijske tehnologije. Pri pripravi te naloge je treba omeniti, da je različica za Android rahlo drugačna in ni bilo mogoče uporabiti najbolj optimalnih algoritmov. Po nameščenih okoljih je bil cilj izdelati program za zaznavanje krogov, kjer je bilo tudi največ težav. Sledilo je zaznavanje z barvami, in nato prikaz želenih rezultatov na zaslonu. Po vsakemu od glavnih korakov, pa tudi vmes, je sledilo testiranje. Testiranje je potekalo doma s prepoznavanjem oblik na slikah, podobnih tistim, ki jih opazimo iz avta na cesti, ko pa je to delovalo zadovoljivo, je test potekal še z vožnjo z avtomobilom. Na podlagi rezultatov sem izvedel popravljanje in vnašanje novih metod. Tak cikel je bilo potrebno ponoviti večkrat.

4.4 Nastavitve projekta in pravice

Ob kreiranju projekta sem moral izbrati primerno verzijo Androida in pa osnovno obliko, oziroma predlogo za glavno aktivnost. Ker ni bilo potrebe po kompleksnem uporabniškem vmesniku, sem za glavno aktivnost določil, naj bo prazna. Minimalna izbrana različica Androida je bila 4.4.2. Enako različico sem izbral kot ciljno različico, pa tudi prevajalnik je deloval na tej različici zaradi tabličnega računalnika, kjer je bila nameščena. Večina uporabnikov je imela v času pisanja že nameščeno vsaj tako ali novejšo različico. Edina pravica, ki sem jo moral določiti, je bila uporaba kamere, vseh ostalih program ne potrebuje. Ker sem uporabil Nvidia Codeworks, mi ni bilo treba dodatno nameščati knjižnice OpenCV v Eclipse, zato sem jo dodal med lastnosti projekta. Ob koncu projekta sem dodal še ikono, ki je prikazana na zaslonu ob izbiri.

4.5 Nastavitev knjižnice OpenCV za kamero

Nastavitev za delovanje kamere je možna na dva pred-definirana načina, prvi je Java, drugi pa Native (C++) način, uokvirjenem v javanski razred, ki pa ju je možno izpeljati z dodatnimi lastnimi metodami in lastnostmi. Oba razreda sta izpeljana iz osnovnega razreda `CameraBridgeViewBase`.

`CameraBridgeViewBase` je abstraktni razred, ki razširja razred `SurfaceView` v Androidu. Dogodke, ki se zgodijo, posreduje enemu od dveh razredov `CvCameraViewListener` ali `CvCameraViewListener2`. Običajno aktivnost, kjer ugotavljamo dogodke, razširja enega izmed teh dveh razredov. Ta dva razreda sta odgovorna za dogodke, kot sta začetek in konec snemanja, in za zajem vsake slike videa med snemanjem. Razlika je v tem, da prva vrsta razreda prejme sliko v barvnem sistemu RGBA in jo preda kot objekt »Mat« knjižnice OpenCV, druga vrsta razreda pa prejme in preda sliko v objektu `CvCameraViewFrame`, ki ga je mogoče pretvoriti v več vrst objektov, zato je druga različica razreda bolj fleksibilna.

4.5.1 Native Camera View

Razred `NativeCameraView` izhaja iz osnovnega razreda `CameraBridgeViewBase`. Gre za Java razred, vendar je v osnovi izpeljan iz C++. Ker večino dela opravi osnovni razred, je ta razred odgovoren le za omogočanje ali onemogočanje kamere ter posredovanje slike naprej.

Prednosti in slabosti uporabe `NativeCameraView` v primerjavi z `JavaCameraView`:

- več slik na sekundo,

- zajem v barvnem sistemu RGBA namesto YUV,

- namenjen samo za armv7 arhitekturo,

- ne deluje na vseh napravah,

- ne podpira vseh funkcij, kot sta avtomatsko ostrenje ali nastavitev svetlobe.

4.5.2 Java Camera View

Tudi razred `JavaCameraView` izhaja iz osnovnega razreda `CameraBridgeViewBase`. Tako kot pri Native razredu tudi ta poskrbi le za prižiganje in ugašanje kamere. Razlika je le v posredovanju slike, ki jo mora knjižnica pretvoriti v barvni sistem RGBA32.

4.6 Detekcija okroglih objektov

Detekcija krogov je bila izvedljiva na več načinov. Ni vsak način primeren za zaznavanje v resničnem času, nekateri so hitrejši in površni, drugi pa zelo natančni, vendar počasni in neprimerni za vsaj zmerno hitro zaznavanje. Jaz sem poizkusil na dva načina in ju poskušal uporabiti z različnimi parametri. Pri prvem preizkusu sem se odločil za čim bolj enostaven pristop. Detekcijo krogov sem najprej preizkusil kar s Hough Circle Transform, kar vsebuje knjižnica OpenCV.

4.6.1 Detekcija krogov z HoughTransform

Detekcija s Hough Circle Transform je detekcija značilk na dvodimenzionalni sliki, kjer slike niso nujno lepe in čiste. V dvodimenzionalnem prostoru je krog predstavljen z enačbo, kjer se upoštevajo koordinate kroga in njegov polmer. Polmer je lahko vnaprej določen in na sliki se najdejo krogi vnaprej znanega polmera. Manjši so kriteriji, več je lažnih zadetkov.

Slika 4.2: Zaznan krog [17].

Zapis, s katerim je definiran krog: C (Xcenter, Ycenter, r). V vrstnem redu gre za x in y koordinato sredine kroga, na zgornji sliki označeno z zeleno točko, ter polmer kroga, s katerim lahko narišemo primerno velik krog, na sliki 4.2 obarvan rdeče.

Implementacija metode, oziroma programa s tem načinom je bila preprosta, za osnovo sem vzel program v dokumentaciji OpenCV, ki je sicer napisan v C++ in je rahlo hitrejša različica standardne detekcije s Hough Transform. Podoben program sem napisal v Javi, ter ga prilagodil svojim zahtevam, tako da je deloval v resničnem času s kamero in ne z že shranjeno sliko. Pred tem sem seveda preizkusil tudi s sliko in ne s kamero. Oboje se je izkazalo, da deluje pravilno. Že takoj se je izkazalo, da to ni bil zelo učinkovit pristop. Program je zaznaval le čiste oblike krogov, tudi če je bil krog le malo popačen, ga program ni zaznal. Bil je tudi zelo neučinkovit in potraten z procesorjem, dosegel je le približno 3 slike na sekundo, kar je sicer zadostno za sprotno zaznavanje, vendar je bilo to le zaznavanje krogov, brez preostalih funkcionalnosti. Program je deloval tako, da je izvirno sliko iz kamere pretvoril najprej v sivinsko sliko, nad njo izvedel Gaussovo megljenje, s tem sem zabrisal morebitne šume, ki bi se pojavili na sliki, nato pa sliko podal metodi `HoughCircles()` v knjižnici OpenCV. Kot rezultat sem dobil lokacije in velikosti krogov, če so bili ti prisotni, v obliki Mat. Iz tega rezultata sem narisal točke in krožnice na izvirni sliki. Knjižnica vsebuje tudi te metode za risanje.

Metoda `HoughCircles()` v knjižnici OpenCV je definirana tako: `HoughCircles (src_gray, circles, CV_HOUGH_GRADIENT, dp, min_dist, param_1, param_2, min_radius, max_radius).`

Posamezni parametri metode določajo, kako bo metoda zaznavala, in imajo naslednji pomen:

`src_gray` – sivinska slika (običajno pretvorjena iz izvorne slike)

`circles` – Mat objekt, kamor se shranijo lokacije in velikosti krogov

`CV_HOUGH_GRADIENT` – določitev načina zaznavanja, na izbiro je možen le ta način

`dp` – inverzno razmerje ločljivosti, vrednost tega je vedno 1

`min_dist` – minimalna razdalja med krogi, običajno zadošča število vrstic slike deljeno z 8

`param_1` – zgornja meja za detekcijo notranjih robov, privzeto je 200

`param_2` – meja za detekcijo sredine kroga

`min_radius` – najmanjši polmer kroga, ki ga metoda zazna, privzeto je na 0, za vse velikosti

`max_radius` – največji polmer kroga, ki ga metoda zazna, privzeto je na 0, za vse velikosti

Pri procesiranju slike sem celoten program napisal v metodi `onCameraFrame`, kjer pride slika kot parameter v obliki `CvCameraViewFrame`. Sliko bi tu lahko pretvoril najprej v objekt `Mat`, jo ustrezno preoblikoval in uredil in potem pretvoril v sivinsko sliko, vendar sem zaradi hitrejšega delovanja sliko raje takoj pretvoril v objekt `Mat` v sivinsko sliko. Nad sliko sem potem izvedel `HoughCircles()` iz knjižnice OpenCV, rezultate pa shranil v predhodno narejen `Mat` objekt z imenom `circleImage`. Rezultate sem izrisal tako, da sem s for zanko pregledal rezultate v tem objektu in na sivinsko sliko izrisal točko na sredini kroga in krožnico. Metoda vrne sivinsko sliko z izrisanimi krogi (Koda 4.1).

Optimalno delovanje sem poskušal doseči z večjimi resolucijami slike kot tudi z manjšimi zaradi več slik na sekundo. Poskusil sem tudi brez glajenja slike in z normalnim in ne z Gaussovim glajenjem ter dodal še detekcijo robov CannyEdge, vendar mi želenih rezultatov ni uspelo doseči.

```
@Override
public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
    Mat grayImage = inputFrame.gray();
    Mat circleImage = new Mat(grayImage.rows(), grayImage.cols(), CvType.CV_8UC1);
    Imgproc.GaussianBlur(grayImage, grayImage, new Size(5, 5), 2, 2);
    Imgproc.HoughCircles(grayImage, circleImage, Imgproc.CV_HOUGH_GRADIENT, 2.0,
             grayImage.rows()/16, 170, 150, 20, 400);

    if (circleImage.cols() > 0)
        for (int x = 0; x < circleImage.cols(); x++)
            {
            double vCircle[] = circleImage.get(0,x);

            if (vCircle == null)
                break;

            Point pt = new Point(Math.round(vCircle[0]), Math.round(vCircle[1]));
            int radius = (int)Math.round(vCircle[2]);

            Core.circle(grayImage, pt, radius, new Scalar(0,255,0), 2);
            Core.circle(grayImage, pt, 3, new Scalar(0,0,255), 2);
            }

    return grayImage;
}
```

Koda 4.1: Primer prvotne detekcije krogov, ki ni učinkovita.

4.6.1.1 Objekt Mat

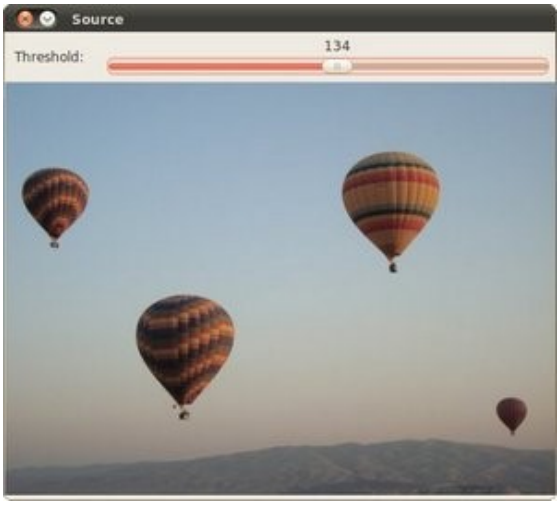
Objekt `Mat` je osnovni objekt knjižnice OpenCV, kamor se shranjuje slike, lahko pa tudi filtre in maske. Določata ga dva podatka, velikost matrike v slikovnih pikah in način, v katerem je shranjena. V prvih različicah knjižnice je bilo potrebno podati še velikost, kakršno zaseda v pomnilniku, v novejših pa sam zavzame toliko kot potrebuje. Omogoča shranjevanje slik v barvnih sistemih RGB, HSV, HLS, YCrCb in CIE L*a*b*. Ker vsebuje tabelo vseh slikovnih pik slike, je enostavno tudi spremeniti barvo in transparentnost posameznih slikovnih pik. Enostavno je tudi obrezati sliko in iz obstoječega `Mat` objekta le določiti od in do katere slikovne pike naj bo velika nova slika, kar sem uporabil pri izrezu in prikazu zaznanega znaka na zaslonu.

4.6.2 Detekcija elips

Detekcija krogov je sicer logičen pristop, vendar ni popoln. Če bi hotel z zgoraj opisanim pristopom zaznati znake, bi ti morali stati točno pred kamero in biti obrnjeni proti njej, kar pa v resničnosti ni mogoče. Znak se torej na kameri pojavi v popačeni obliki, pa čeprav zelo malo, vendar dovolj, da sami krogi niso učinkoviti in je detekcija elips bolj smiselna.

Za detekcijo elips sem uporabil podoben pristop, ki je naveden kot primer v dokumentaciji OpenCV, tam je sicer program predstavljen v C++. V Javi ni bilo mogoče uporabiti popolnoma enakega pristopa, ker se objekti, ki jih OpenCV uporablja v C++, ne skladajo s tistimi, ki se uporabljajo v Javi in imajo drugačen pomen.

Vsako sliko, ki mi jo vrne metoda, najprej pretvorim v sivinsko, v objekt `Mat` ter nad njo izvedem še zameglitev, da se odstrani morebiten šum. Nato uporabim pristop za detekcijo elips, kot je naveden v dokumentaciji. Za detekcijo se uporabljajo tako imenovani vrteči pravokotniki `RotatedRect`, ki se izrišejo okoli obrisov značilk na sliki (Slika 4.3). Za ugotavljanje ali je elipsa pa se uporabi metoda `fitEllipse()` (Slika 4.4). Izrišem jo na enak način kot kroge. Na sliki z znaki je težko zaznati s prostim očesom, da gre za elipso, saj je popačenost, ki jo razlikuje od kroga, zelo majhna, zato je primer iz dokumentacije bolj nazoren.



Slika 4.3: Slika balonov, kjer se uporabi zaznavanje s pravokotniki [20].

Slika 4.4: Rezultat in izris elips [20].

4.7 Zaznavanje točk in primerjanje slik

Zaznavanje točk sem izvedel z uporabo algoritma ORB, ki je najboljši odprtokodni algoritem za ta namen. Za primerjanje točk med slikama sem ravno tako uporabil algoritem ORB, poskusil pa sem tudi z algoritmom, ki pregleda vse možnosti, oba sta se izkazala za približno enako učinkovita pri številu ujemanj, le da pri algoritmu, ki pregleda vse možnosti, poraba procesorske moči pri večjem številu ujemanj narašča hitreje. Zaznan znak sem primerjal z dvema umetno narisanima znakoma. Za boljše primerjanje se lahko uporabi več slik. Znak (Slika 4.5), ki ga primerjam z zaznanimi znaki, ima samo številko nič, saj bi lahko morebitna druga številka zmanjšala število točk.

4.5: Znak, s katerim se primerjajo elipse.

4.8 Zaznavanje barv

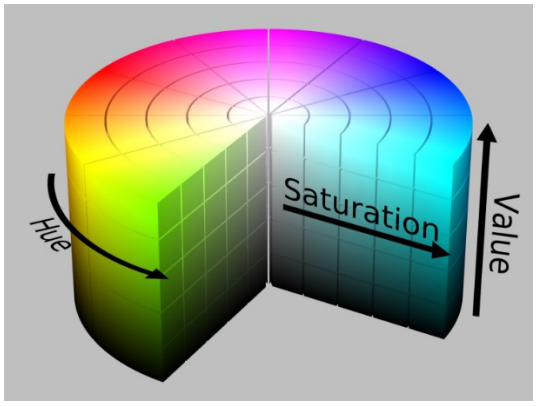
Sprva sem zaznavanje barv poskušal uvesti že pri znakih za hitrost, da bi tako ugotovil, kje se nahaja rdeč rob. Iskanja roba sem se lotil tako, da sem za vsak zaznan krog oziroma elipso od sredine v štiri smeri levo, desno, navzgor in navzdol preverjal slikovne pike, dokler nisem naletel na rdečo. Metoda je delovala zanesljivo, vendar le, če je bila svetloba zadostna. Problem metode pa je bil, da je za preverjanje vsakega kroga porabila preveč časa. Pri enem ali dveh krogih se to ni poznalo, ko pa je bilo na sliki treba preveriti 10 ali več krogov, pa je program deloval zelo počasi z eno ali dvema slikama na sekundo v najboljšem primeru, ali pa je celo zamrznil.

Kasneje sem podobno zaznavanje uporabil za rumene znake, tam ni potrebno iti do roba, saj je cel znak rumen. Pojavil se je podoben problem kot pri rdečem robu, predvsem zaradi svetlobe. Težava je bila v tem, da se na sliki, ki v barvnem prostoru RGB težko natančno določi odtenke in nasičenost barve, saj ta lahko ob drugačni svetlobi deluje kot druga barva. Rumena v slabi svetlobi postane siva ali rjava. Težavo sem rešil tako, da sem sliko najprej pretvoril v barvni sistem HSV, ki je neodvisen od svetlobe. Tam sem tako le določil okvir, v katerem so bile rumene barve.

Kljub vsemu v končni različici ni uporabljenega zaznavanja barv, ker se zaznavanje rumenih znakov ni izkazalo kot dobro za praktično uporabo z obstoječimi algoritmi. Kljub temu, da so znaki za naselje veliki, se na kameri z zaznavanjem barv velikokrat sploh ne pojavijo.

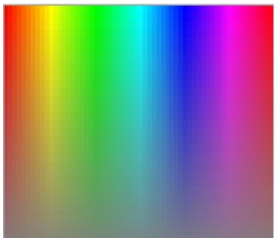
4.8.1 Barvni model HSV

Barvni model HSV je cilindrična predstavitev barvnega modela RGB (Slika 4.6). Razvit je bil v 70-ih letih prejšnjega stoletja za potrebe računalniške grafike. Danes se uporablja predvsem v programih za obdelovanje slik in pri računalniškem vidu. Kratica HSV pomeni odtenek (ang. *hue*), nasičenost (angl. *saturation*) in vrednost (angl. *value*), medtem ko pri barvnem modelu RGB pomeni vrednost za vsako od treh barv. Model HSV se dobro obnese, kadar je treba ločiti eno barvo od drugih.



Slika 4.6: Cilindrična shema barvnega modela HSV [21].

V mojem primeru je je bilo treba iskati barve v območju rdeče in rumene. Rumeno je lažje določiti, saj se nahaja samo na enem delu barvnega modela, rdeča pa na dveh (Slika 4.7). Na spodnji sliki se vidi, da se rdeča nahaja na dveh nasprotnih vrednostih modela HSV, zato je potrebno upoštevati obe, čeprav se je med testiranjem izkazalo, da je večina zadetkov z enega območja.



Slika 4.7: Dve območji rdeče barve.

Poglavje 5

Aktivnosti in razredi

V tem poglavju je predstavljen progam, ki sem ga izdelal, ter aktivnost in razredi, ki jih vsebuje. Podrobneje so predstavljene tudi ključne metode. Program je sestavljen iz ene glavne aktivnosti `DetectActivity` in dveh razredov, od katerih se eden uporablja zgolj za predstavitev zaznanega objekta, drugi za razširitev in uporabo kamere. Za samo eno aktivnost sem se odločil, ker v programu ni nobene potrebe po nastavitvah, morda bi lahko spreminjal le ozadje, vendar je to že privzeto nastavljeno na temnejšo sivo. Za uporabo nitenja sem namenil še dva razreda, vsakega za predstavitev okroglih in pravokotnih znakov, vendar nitenje v končnem programu ni uporabljeno. V program sem sprva vključil še C++ razred za zajem slike, vendar tudi ta zaradi kompleksnosti in hitrosti ni vključen.

5.1 Aktivnost DetectActivity

Aktivnost `DetectActivity` je v programu edina in zato tudi glavna aktivnost `MainActivity`, ki jo je treba določiti. Sestavljena je iz metode, ki jih mora vsebovati vsaka aktivnost, `onCreate()` in konstruktorja, ter iz metod, ki analizirajo posamezno sliko. Aktivnost implementira `CvCameraViewListener2` za interakcijo knjižnice OpenCV s kamero. Končna oblika sicer nima gumbov in ostalih kontrol za interakcijo, vendar sem si skozi razvoj pomagal z kontrolo `SeekBar` v Androidu, kjer se s potegom prsta nastavi vrednost. Pomagal sem si s tremi kontrolami, da sem lahko med samim snemanjem spreminjal parametre za odtenek, nasičenost in vrednost v barvnem sistemu HSV. Spremenljivke se inicializirajo ob klicu metode `onCreate()`, zato jih nastavim v pomožni metodi `initializeVariables()`. Kliče se ta metoda, kar je bolj pregledno, kot če bi vsako spremenljivko nastavil posebej.

Metode, ki jih implementira `CvCameraViewListener2`:

- `onPause()`,
- `onResume()`,
- `onDestroy()`,
- `onCameraFrame()`,
- `onCreateOptionsMenu()`,

- `onOptionsItemSelected()`,
- `onTouch()`.

Pomožne metode:

- `initalizeVariables()`,
- `initializeListeners()`,
- `setDescriptors()`,
- `setFolderDescriptors()`,
- `compareDescriptors()`,
- `calculateAndDrawKeypoints()`,
- `findEllipses()`,
- `loadImageFromfile()`.

5.1.1 Iskanje elips

Iskanje elips poteka v metodi `findEllipses()`, prikazana je na kodi 5.1. Metoda ne prejme nobenega argumenta in je glavna metoda programa, brez nje ni možno obdelovati nobenih znakov. Narejena je sicer tudi metoda, ki prejme kot argument objekt `Mat`. Metodi tako lahko podam zajeto sliko, ker se slika, ki jo vrne razred `SignView`, uporablja v več metodah in je spremenljivka namenjena sliki globalna, nima smisla podati slike kot argument. Metoda tako neposredno dostopa do globalne spremenljivke in ni možnosti, da bi ta spremenljivka bila prazen objekt.

```
private void findEllipses(){
    Mat thresholdOutput = new Mat();
    List<MatOfPoint> contours = new ArrayList<MatOfPoint>();
    MatOfInt4 hierarchy = new MatOfInt4();

    Imgproc.Canny(frame, thresholdOutput, 50, 180);
    Imgproc.findContours(thresholdOutput, contours, hierarchy, Imgproc.RETR_LIST, Imgproc.CHAIN_APPROX_SIMPLE);
    RotatedRect minEllipse[] = new RotatedRect[contours.size()];

    for(int i=0; i<contours.size();i++){
        MatOfPoint2f temp=new MatOfPoint2f(contours.get(i).toArray());
        if(temp.size().height > 80 && temp.size().height < 200){
            double a = Imgproc.fitEllipse(temp).size.height;
            double b = Imgproc.fitEllipse(temp).size.width;
            if(Math.abs(a - b) < 10)
                minEllipse[i] = Imgproc.fitEllipse(temp);
        }
    }
    detectedObjects.clear();

    for( int i = 0; i< contours.size(); i++ ){
        Scalar color = new Scalar(180, 255, 180);
        if(minEllipse[i] != null){
            detectedObjects.add(new DetectedObject(minEllipse[i].center));
            if((int)minEllipse[i].center.x-(minEllipse[i].size.width/2) > 0 &&
                    (int)minEllipse[i].center.y-(minEllipse[i].size.height/2) > 0 &&
                    (int)minEllipse[i].center.x+(minEllipse[i].size.width/2) < frame.cols() &&
                    (int)minEllipse[i].center.y+(minEllipse[i].size.height/2) < frame.rows()){
                Rect signRect = new Rect((int)minEllipse[i].center.x-((int)minEllipse[i].size.width/2),
                        (int)minEllipse[i].center.y-((int)minEllipse[i].size.height/2),(int)minEllipse[i].size.width,
                        (int)minEllipse[i].size.height);
                Mat cropedSign = frame.submat(signRect);
                possibleSigns.add(cropedSign);
            }
            Core.ellipse(frame, minEllipse[i], color, 2, 8);
        }
    }
}
```

Koda 5.1: Metoda `findEllipses()`.

Detekcija elips ali krogov s pomočjo HoughTransform ni možna na prvotni sliki. Če je slika sivinska, je detekcija sicer delno uspešna, vendar že ob tako velikem zmanjšanju slik na sekundo to še poslabša. Na sliki se zaradi tega problema najprej izračunajo robovi z metodo CannyEdge, v knjižnici OpenCV se do nje dostopa z `Imgproc.Canny()`. Na sliki z robovi nato metoda poišče obrise in jih shrani v seznam. Ker se elipse izračunavajo s pomočjo pravokotnikov, se rezultati shranijo v tabelo objektov `RotatedRect`. S for zanko se nato izločijo elipse, ki ne ustrezajo kriterijem, takšne, ki so prevelike ali premajhne ali pa del elipse, ni v okviru slike in je nepopolna. Takšne nepopolne elipse povzročijo izjeme, zato je bolje, da jih ni na seznamu, kot pa da bi jih kasneje uporabili za primerjanje z znakom, ki ne bi bilo uspešno zaradi premalo ujemanj. Metoda na sliko sicer elipso tudi izriše, končnemu uporabniku to ni onemogočeno, ker se slika ne prikazuje, za razumevanje delovanja pri razvoju programa pa je to nujno.

5.1.2 Opisovanje znaka

Pri elipsi, ki jo program zazna, je potrebno preveriti, ali gre za znak ali ne. Vsaka elipsa je lahko potencialno znak. Zato se v ta namen uporablja metoda `setDescriptors()` (Koda 5.2), ki na elipsi poišče točke. Metoda kot argument vzame sliko v objektu `Mat`, in jo spremeni v sivinsko. Nato se uporabi FeatureDetector, kjer nastavim, da deluje po načinu ORB (Oriented FAST and BRIEF), kar je tudi edina možnost, saj je alternativni način SURF (Speeded-Up Robust Features) licenčni. Spremenljivka `MatOfKeypoint` je namenjena hranjenju točk na sliki. FeatureDetector izračuna te točke in jih shrani v to spremenljivko. DescriptorExtractor razbere točke in jih shrani v bolj obširno uporaben objekt `Mat`.

```
private Mat setDescriptors(Mat image){
    Mat imageGray = new Mat();
    Imgproc.cvtColor(image, imageGray, Imgproc.COLOR_BGR2GRAY);
    FeatureDetector detector = FeatureDetector.create(FeatureDetector.ORB);
    MatOfKeyPoint keypoints = new MatOfKeyPoint();
    detector.detect(imageGray, keypoints);
    DescriptorExtractor extractor = DescriptorExtractor.create(DescriptorExtractor.ORB);
    Mat descriptors = new Mat(imageGray.rows(), imageGray.cols(), image.type());
    extractor.compute(imageGray, keypoints, descriptors);
    return descriptors;
}
```

Koda 5.2: Metoda `setDescriptors()`.

Dodal sem tudi metodo `setFolderDescriptors()`, ki pregleda posebej ustvarjeno mapo s slikami, kjer so ročno narisani znaki, in nad njimi uporabi metodo `setDescriptors()`, saj je tudi za te znake potreben izračun točk, da je možna primerjava z zaznanimi točkami. Mapa z imenom *minimized* s slikami se nahaja v mapi *assets*, ki je tudi namenjena shranjevanju slik in ostalih datotek za uporabo v programu za Android.

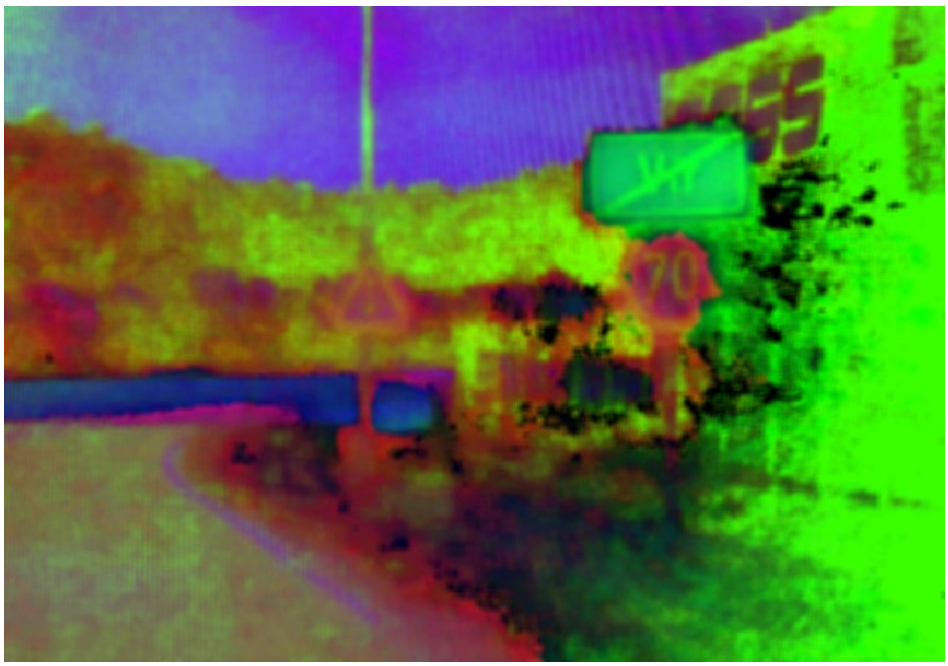
Med razvojem sem sicer uporabljal metodo `calculateAndDrawKeypoints()`, ki sicer povzame vse zgoraj naštete metode, vendar dopušča manj prilagodljivosti, zato sem jo kasneje razčlenil in dopolnil. Metoda je sicer izpeljana iz primera, ki se uporablja za zaznavanje elips, in je opisana v spletni dokumentaciji OpenCV.

5.1.3 Zaznavanje ostalih znakov

Detekcija rumenih znakov in znakov za hitrost se je izkazala za neuspešno oziroma delno uspešno. Pri znakih za hitrost je zaznavanje barv odkrilo vsaj nekaj znakov, pri tistih za naselje pa sem zaznal samo enega ali dva.

Za zaznavanje barv sem najprej pretvoril sliko v barvni model HSV, nato pa z uporabo metode inRange poiskal želene barve. V spodnjem primeru je metoda

`extractYellowAreas()`. Ta metoda sicer poišče rumena območja, vendar sem na enak način poskušal poiskati tudi rdeča območja. Primer povprečno dobre slike v barvnem sistemu HSV je viden na sliki 5.1.



Slika 5.1: Znaki v barvnem modelu HSV.

5.2 Razred DetectedObject

Razred `DetectedObject` ne vsebuje svojih metod. Vsebuje le eno lastnost točko in konstruktor, ki jo določi. Uporabljen je le za lažje razumevanje pri programiranju glavne aktivnosti.

5.3 Razred SignView

Razred `SignView` razširja razred `JavaCameraView`, zato so implementirane vse metode tega razreda. Uporabljene niso vse metode, vendar le metoda za določanje resolucije, metoda za učinke in pomožne metode za vračanje nastavljenih vrednosti.

5.4 Uporaba C++

V programu sem poskušal uporabiti tudi JNI in sicer uporabo megljenja in kasneje iskanje ter primerjanje točk po primeru v dokumentaciji. Megljenje je sicer bilo uspešno, vendar sem enako uporabil kar v Javi. Zaradi hitrosti zajemanja sem poskušal tudi primerjanje točk v

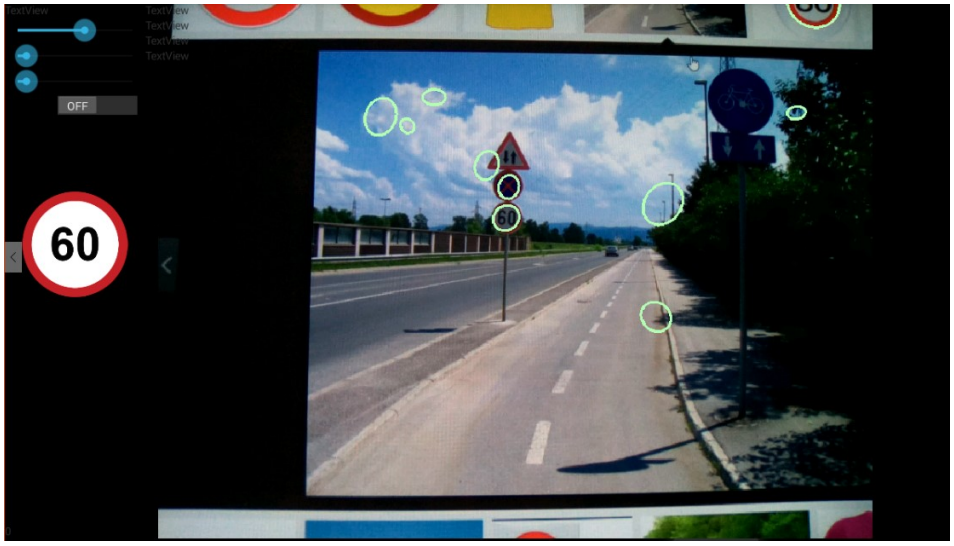
C++, če bi se to izkazalo za hitrejše, vendar na mojem tabličnem računalniku funkcija v C++ ni delovala.

5.5 Uporabniški vmesnik in testiranje

Program praktično nima uporabniškega vmesnika oziroma je preprost. Na podlagi se prikazuje rahlo obrezana slika znaka, ki ga program zazna. Prikaz zaznanega znaka je na sliki 5.2.

Slika 5.2: Prikaz znaka na zaslonu.

Čeprav končni program nima kompleksnega uporabniškega vmesnika, sem med testiranjem uporabljal drugačnega (Slika 5.3). Omogočal mi je pregled nad zaznanimi objekti in točkami, ki se ujemajo, v levem spodnjem kotu, ter sliko, s katero se primerja objekt. Omogočal pa mi je tudi nekaj nastavitev pri izbiri zaznavanja barv.



Slika 5.3: Vmesnik za testiranje in prikaz zaznanih objektov.

Na zgornji sliki je prikazan vmesnik med testiranjem, vidijo pa se tudi elipse zaznanih objektov. Med njimi je znak, ki ga iščemo, preostali pa se s pomočjo algoritmov izločijo. Znak na sliki ima s sliko na levi približno 150 zaznanih skupnih točk. Za dobro ujemanje velja znak z več kot 100 točkami. Število skupnih točk pa je odvisno tudi od vrste algoritma za zaznavanje in primerjanje točk.

Poglavje 6

Sklepne ugotovitve

V diplomskem delu sem predstavil problem zaznavanja prometnih znakov in predstavitev poteka izdelave programa za zaznavanje le teh. Z izdelavo programa sem imel nekaj težav. V preteklosti sem delal programe za operacijski sistem Blackberry 10, program, ki pa sem ga izdelal za namene diplomske naloge, pa je moj prvi večji program za Android. Nezadostno poznavanje Androida mi je torej povzročalo nekaj težav, večje težave pa mi je povzročal tudi sam tablični računalnik, na katerem sem testiral, saj je nižjega cenovnega razreda in je imel dokaj slabo kamero z veliko šuma.

Z poznavanjem knjižnice OpenCV nisem imel večjih težav, saj smo jo spoznali že na fakulteti pri predmetu Multimedijske tehnologije. Paziti sem moral le na to, da je knjižnica za Android drugačna od polne različice. Dokumentacija je zelo dobra, prav tako je kar nekaj primerov na voljo, zato sem izhajal iz njih.

Program v testnem okolju deluje brezhibno, v resničnem okolju pa malo slabše. Seveda je pomemben tudi človeški faktor, postavitve telefona v avtu, vendar se kljub temu vozilo premika zelo hitro glede na okolje, zato je možnost, da so slabše vidni ali napol zakriti znaki izpuščeni.

Hitrost delovanja in poraba energije sta sicer zadovoljivi, vendar obstaja nekaj izboljšav. S črnim zaslonom namesto sivim bi lahko prihranil še nekaj energije, vendar je to zanemarljivo. Hitrost delovanja bi lahko izboljšali z nitenjem, vendar tudi z nižjim zajemanjem slik dobimo dovolj podatkov. Projektna datoteka, izdelana z programom Eclipse je na voljo na spletu [23].

Pri izdelavi diplomske naloge sem pridobil veliko znanja o delovanju operacijskega sistema Android in razvoja programov za ta sistem. Veliko sem se naučil tudi o knjižnici OpenCV in različnih vrstah zaznavanj, ki jih je mogoče uporabiti tudi za druge namene.

Literatura

[1] Miguel Ángel García-Garrido, Miguel Ángel Sotelo, and Ernesto Martín-Gorostiza, Fast Road Sign Detection Using Hough Transform for Assisted Driving of Road Vehicles, *Department of Electronics, University of Alcalá, Alcalá de Henares, Madrid, Spain*, 2005

[2] Hasan Fleyeh , Mark Dougherty, Road and traffic sign detection and recognition, *10th EWGT Meeting and 16th Mini-EURO Conference, Poznan, Poland, 13-16 September, 2005*

[3] Sebastian Houben, A single target voting scheme for traffic sign detection, *IEEE Intelligent Vehicles Symposium (IV) Baden-Baden, Germany*, June 5-9, 2011

[4] Gareth Loy, Nick Barnes, Fast Shape-based Road Sign Detection for a Driver Assistance System, *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2004

[5] W. G. Shadeed, D.I. Abu-Al-Nadi, and M.J. Mismar, Road traffic sign detection in color images, *Electrical Engineering Department University of Jordan, Amman-Jordan,* IEEE, 2003

[6]Scott E. Umbaugh, *Digital Image Processing and Analysis, Second Edition*, Taylor & Francis Group, 2011, str 208

[7] Understanding Features. [Online]. Dosegljivo:
http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html#features-meaning
[Dostopano 05.09.2015].

[8] Android. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Android_(operating_system)
[Dostopano 05.09.2015].

[9] Java. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Java_(programming_language)
[Dostopano 05.09.2015].

[10] Eclipse. [Online]. Dosegljivo:
https://eclipse.org/org/
[Dostopano 05.09.2015].

[11] Eclipse. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Eclipse_(software)
[Dostopano 05.09.2015].

[12] Android Studio. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Android_Studio
[Dostopano 05.09.2015].

[13] Android NDK. [Online]. Dosegljivo:
http://developer.android.com/tools/sdk/ndk/index.html
[Dostopano 03.10.2015].

[14] OpenCV. [Online]. Dosegljivo:
http://opencv.org/about.html
[Dostopano 06.09.2015].

[15] OpenCV for Android. [Online]. Dosegljivo:
http://opencv.org/platforms/android.html
[Dostopano 06.09.2015].

[16] Nvidia codeworks. [Online]. Dosegljivo:
https://developer.nvidia.com/codeworks-android
[Dostopano 12.09.2015].

[17] Hough Circle Transform. [Online]. Dosegljivo:
http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html
[Dostopano 12.09.2015].

[18] Android emulator. [Online]. Dosegljivo:
http://www.makeuseof.com/tag/3-ways-run-android-apps-windows
[Dostopano 12.09.2015].

[19] Feature matching. [Online]. Dosegljivo:
http://docs.opencv.org/2.4/doc/tutorials/features2d/feature_flann_matcher/feature_flann_matcher.html
[Dostopano 14.11.2015].

[20] Rotated Rect. [Online]. Dosegljivo:
http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/bounding_rotated_ellipses/bounding_rotated_ellipses.html
[Dostopano 14.11.2015].

[21] HSV in HSL. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/HSL_and_HSV
[Dostopano 05.12.2015].

[22] Cascade Classifier. [Online]. Dosegljivo:
http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html
[Dostopano 05.12.2015].

[23] Projektna datoteka diplome. [Online]. Dosegljivo:
https://github.com/skyhawk92/DiplomaZnaki.git
[Dostopano 07.02.2015].