

Универзитет у Београду

Електротехнички факултет

ДИПЛОМСКИ РАД

Препознавање ограничења брзине са знака коришћењем формалне логике

Ментор: Студент:

проф. др. Бошко Николић Момчило Бајаловић 643/13

Београд, 2020

Садржај:

[Увод 5](#_Toc50635836)

[Опис система 6](#_Toc50635837)

[Преглед постојећих решења 6](#_Toc50635838)

[Преглед детаља решења 7](#_Toc50635839)

[Услови под којима се решење примењује 7](#_Toc50635840)

[Подешавање извршног окружења 8](#_Toc50635841)

[Имплементација решења 9](#_Toc50635842)

[OpenCV 9](#_Toc50635843)

[Припрема слике 10](#_Toc50635844)

[Обрада слике 17](#_Toc50635845)

[Препознавање цифри 31](#_Toc50635846)

[Могућности за модификацију 38](#_Toc50635847)

[Корисничко упутство 40](#_Toc50635848)

[Закључак 45](#_Toc50635849)

[Литература 46](#_Toc50635850)

**Садржај слика:**

[Сл. 1 Грешка када слика не постоји на датом путу 10](#_Toc50645718)

[Сл. 2 Mat је само матрица боја пиксела слике 10](#_Toc50645719)

[Сл. 3 Грешка када је унета слика погрешне величине 11](#_Toc50645720)

[Сл. 4 Улазна слика пре и после засићења 12](#_Toc50645721)

[Сл. 5 Слика пре и после пребацивања у црно-белу верзију 12](#_Toc50645722)

[Сл. 6 Слика пре и после примењивања blur трансформације 13](#_Toc50645723)

[Сл. 7 Матрице маске за х и у осу 13](#_Toc50645724)

[Сл. 8 Улазна црно-бела слика за пример 14](#_Toc50645725)

[Сл. 9 Примери градијената интензитета улазне слике по х оси 14](#_Toc50645726)

[Сл. 10 Примери градијената интензитета улазне слике по у оси 14](#_Toc50645727)

[Сл. 11 Формуле за снагу и правац 15](#_Toc50645728)

[Сл. 12 Пример црне тачке на белој позадини и детектованих праваца њених ивица 15](#_Toc50645729)

[Сл. 13 Пример коначне слике са извученим ивицама 16](#_Toc50645730)

[Сл. 14 Слика са извученим ивицама пре и након избељивања 16](#_Toc50645731)

[Сл. 15 Почетна слика и слика са трапезима за скенирање 17](#_Toc50645732)

[Сл. 16 Слике појединачних трапеза 17](#_Toc50645733)

[Сл. 17 Почетна слика и слика након одсецање горњег и доњег дела 18](#_Toc50645734)

[Сл. 18 Приказ свих линија које би се пролазиле за тражење празне 18](#_Toc50645735)

[Сл. 19 Визуелизација пређених линија за претрагу празне 19](#_Toc50645736)

[Сл. 20 Тачке нађене у једној вертикалној линији 19](#_Toc50645737)

[Сл. 21 Иако је у сред броја, ако фале пиксели линија ће и даље бити празна 20](#_Toc50645738)

[Сл. 22 Детектоване линије између три цифре 20](#_Toc50645739)

[Сл. 23 Приказ грешке при раздвајању цифри 20](#_Toc50645740)

[Сл. 24 Линије скениране у току horizontalScan-а 21](#_Toc50645741)

[Сл. 25 Пример детектованих тачака у првој линији horizontalScan-а 21](#_Toc50645742)

[Сл. 26 Детекција линије и означавање тачки 22](#_Toc50645743)

[Сл. 27 Иницијалне границе цифри 22](#_Toc50645744)

[Сл. 28 Исецање по границама пре и после обраде 23](#_Toc50645745)

[Сл. 29 Линије детектоване за сечење са горње и доње стране 23](#_Toc50645746)

[Сл. 30 Линије детектоване за сечење са леве и десне стране 24](#_Toc50645747)

[Сл. 31 Примери различитих седмици са наведеном шпицастом верзијом(скроз десно) 25](#_Toc50645748)

[Сл. 32 Линије скениране за налажење леве и десне стране 25](#_Toc50645749)

[Сл. 33 Почетне линије за одсецање 26](#_Toc50645750)

[Сл. 34 Почетно одсецање цифара 26](#_Toc50645751)

[Сл. 35 Крајње одсецање цифара 27](#_Toc50645752)

[Сл. 36 Пример равне и криве јединице 28](#_Toc50645753)

[Сл. 37 Разлика код писања јединице - са и без серифа 28](#_Toc50645754)

[Сл. 38 Главни делови логике код последњег исецања за 0 и 1 респективно 29](#_Toc50645755)

[Сл. 39 Нивои заобљености нуле 30](#_Toc50645756)

[Сл. 40 Области за детекцију 0 32](#_Toc50645757)

[Сл. 41 Области за детекцију 1 33](#_Toc50645758)

[Сл. 42 Области за детекцију 2 33](#_Toc50645759)

[Сл. 43 Области за детекцију 3 34](#_Toc50645760)

[Сл. 44 Области за детекцију 4 34](#_Toc50645761)

[Сл. 45 Области за детекцију 5 35](#_Toc50645762)

[Сл. 46 Области за детекцију 6 35](#_Toc50645763)

[Сл. 47 Области за детекцију 7 36](#_Toc50645764)

[Сл. 48 Области за детекцију 8 37](#_Toc50645765)

[Сл. 49 Области за детекцију 9 37](#_Toc50645766)

[Сл. 50 Улазна слика и нађене линије између њених цифара 38](#_Toc50645767)

[Сл. 51 Времена извршавања делова програма 39](#_Toc50645768)

[Сл. 52 Пример покретања програма 40](#_Toc50645769)

[Сл. 53 Пример покретања програма за једну слику из командне линије 41](#_Toc50645770)

[Сл. 54 Тестови за 90 41](#_Toc50645771)

[Сл. 55 Тестови за 70 42](#_Toc50645772)

[Сл. 56 Тестови разних бројева 42](#_Toc50645773)

[Сл. 57 Тестови за 25 43](#_Toc50645774)

[Сл. 58 Извршавање 3 слике и теста 43](#_Toc50645775)

[Сл. 59 Паљење и гашење исписа цифара 44](#_Toc50645776)

[Сл. 60 Ток програма при налажењу грешке 45](#_Toc50645777)

# **Увод**

Тренутно сви алгоритми и технике препознавања на сликама се заснивају на неком типу машинског учења. Из потребе за великим скупом примера за обучавање таквих неуралних мрежа ствара се простор за алгоритмима за препознавање слике на основу формалне логике. Ти алгоритми би, уз унапред позната правила, могли сами да одлуче да ли улазна слика одговара траженој форми.

Алгоритми за препознавање садржаја слика ће постајати све потребнији у будућности, тако да овакав релативно мали алгоритам може да послужи. Он је одличан за мање, интегрисане уређаје који немају много меморије, процесорске снаге ни посебних ВИ(AI) језгара. Чак и најслабији модерни процесори (чак и из последње деценије) неће имати проблем да покрену овај алгоритам у неком разумном временском периоду.

Предности овог начина обраде је то што нам не треба огромна неурална мрежа за обучавање са исто огромним бројем примера. И то је све само за обучавање мреже пре него што она заправо може да се користи. Модерне неуралне мреже такође често захтевају посебан хардвер ради њиховог бољег извршавања, док формална логика се може покренути на било ком општом процесору (general purpose CPU). Такође сама обрада формалне логике може да буде бржа од многобројних трансформација потребних за припрему слике да се провуче кроз неуралну мрежу. Мана је што је углавном мања толеранција на грешке у поређењу са машинским учењем.

За разлику од неуралне мреже која над великим бројем примера налази сличности и на основу њих покушава да одлучи да ли на датој слици постоји нешто што он препознаје, формална логика поставља строга правила за одлучивање. У овом раду је дата мала толеранција да алгоритам не тражи све у пиксел тачности. Добрим подешавањем ове толеранције алгоритам може да ради и са разним фонтовима бројева на знаку.

У даљем раду ће бити описано како сама логика функционише, коришћене постојеће framework-ове, места за побољшање и наравно проценат успеха.

# **Опис система**

## Преглед постојећих решења

Није никаква тајна да већ постоје системи који распознају саобраћајне знакове. Они се, логично, углавном монтирају у аутомобиле. Како да би олакшали возачу, тако и за тестирање самих алгоритама за будућу фазу аутомобилства: самовозеће аутомобиле.

Постојећа решења углавном користе већ истрениране неуралне мреже за детекцију и препознавање саобраћајних знакова. Коришћена мрежа мора бити унапред истренирана да би могла да се користи, а и онда треба да се провуче слика кроз, у зависности од саме имплементације, већи број скривених слојева мреже пре него што се добије одговор. То је све неопходно јер обучавање саме мреже и њена прецизност/тачност директно зависи од величине скупа примера за обучавање и верификацију. Зато је чак и уобичајено да се узимају супер-компјутери или серверске фарме за процесирање огромних количина улазних података за обучавање такве мреже.

Кад је већ обучена мрежа њено извршавање се може још више убрзати коришћењем посебних ВИ(AI) хардверских језгара. Ова језгра су углавном специјализована за паралелни рад, што има смисла јер више неурона може да се обрађује независно од осталих у том тренутку. Из истог овог разлога су се у почетку развијања неуралних мрежа користиле графичке карте за убрзавање.

Постоје разне технике машинског учења. За ComputerVision се углавном користи нека комбинација Fuzzy Logic-а и Feature Detection-а. Ово обезбеђује детекцију и са непотпуном, развученом и ротираном сликом. Али такође, управо из ових разлога је потребна огромна количина података за обучавање. Јер без тога исти ови алгоритми који чине машинско учење тако свестраним могу да дају много лажних позитивних детекција.

Иако је машинско учење најзаступљеније у пољу обраде слике (ComputerVision) његова опширност и тежина/време обучавања остављају места за неки лакши/специјализованији алгоритам. Због тога се овај рад фокусира на алгоритам са правилима формалне логике за обраду слике знака ограничења брзине и тачно читање самог броја из њега.

## Преглед детаља решења

Овај алгоритам користи OpenCV framework за учитавање слика. То је урађено ради коришћења функције OpenCV-а за извлачење ивица из слике. Ова основна трансформација већ постоји оптимизована у OpenCV framework-у тако да нема поенте писати је испочетка.

Рад је писан у програмском језику C++ и коришћен је основни OpenCV framework 3.4.3.

Укратко, алгоритам функционише на следећи начин:

* чита слику са задате путање
* трансформише је само у слику својих ивица
* одсеца слику испод и изнад цифре
* провлачењем вертикалних линија налази појединачне цифре
* провером одређених зона цифре их распознаје
* у конзоли исписује препознат број са слике знака

Цео пројекат се састоји из 3 логичке целине:

1. Припрема

-читање и трансформација улазне слике

1. Обрада

-издвајање појединачних цифри

1. Препознавање

-детекција бројева појединачних цифри

## Услови под којима се решење примењује

Услови за алгоритам да функционише како треба су:

1. улазна слика мора бити 300х300 пиксела величине
2. улазна слика мора бити типа који OpenCV подржава (подржава најпопуларније типове (jpg, png, bmp) као и још неке)
3. улазна слика мора бити само круга саобраћајног знака (пожељно да спољашње ивице окружујућег црвеног круга дотичу ивице слике)
4. саме цифре не смеју да буду превише скривене/оштећене

Ако постоји нешто између цифри (стикери, нацртана линија маркером...) што, ако може да се издвоји ивица, ствара линију због које не може да се провуче чиста вертикална линија између цифри. Постоји могућност да ће ометати раздвајање цифри и потенцијално извршити лоше одсецање и одатле лошу детекцију.

## Подешавање извршног окружења

Пројекат је писан у програму Eclipse 2019-12, мада би требао ад ради у било којој његовој верзији. Да би пројекат радио мора да је на систему убачен OpenCV framework. Поред њега потребан је и g++ компајлер и cmake програм (може и Cmake gui). Потребан је и gdb да би радио debugger.

Кораци за инсталацију OpenCV framework-а су следећи и односе се на Linux и MacOS системе:

1. скине се .zip фајл са OpenCV-овог сајта за одговарајућу верзију, у овом случају 3.4
2. отвори се терминал и куца следеће
3. mkdir build\_opencv
4. cd build\_opencv
5. отвори се Cmake gui
6. у поље source folder се упише путања до фасцикле где смо екстрактовали .zip фајл
7. у поље build folder се упише путања до фасцикле коју смо направили у кораку 3
8. у поље search се куца DOPENCV\_GENERATE\_PKGCONFIG и побринемо се да је штиклирано
9. притиснути configure дугме
10. кад се то заврши притиснути generate дугме
11. сад кад је све за инсталацију изгенерисано у терминал куцамо: make -j7
12. на крају: sudo make install

Последња ствар што треба да се уради је да се укључе OpenCV библиотеке у пројекат. То се ради тако што се оде десни клик на пројекат па у Properties. Одатле се иде у C/C++ Build па у Settings.

Ту под GCC C++ Compiler па Includes се у поље Include paths(-I) уписује пут до OpenCV заглавља. Он је углавном "/usr/local/include/opencv4".

Затим се под GCC C++ Linker под Libraries у поље Library search path (-L) уписује пут до OpenCV библиотека "/usr/local/lib", а у поље Libraries(-l) се наводе специфичне библиотеке које нам требају. У овом случају то су библиотеке:

"opencv\_core" -за основне типове попут Mat

"opencv\_imgproc" -за функције за манипулацију слике попут Canny, blur, cvtcolor...

"opencv\_imgcodecs" -за учитавање слика са путање

"opencv\_highgui" -за исцртавање слика

Потенцијално треба под GCC C++ Compiler у поље Command поред g++ дода још -std=c++11 да би форсирао c++11 који је неопходан да OpenCV ради.

## Имплементација решења

### OpenCV

OpenCV (Open Source Computer Vision Library) је библиотека примарно намењена рачунарском виду у реалном времену. Оригинално започето као Интелов пројекат са циљем прављења преносивог, оптимизованог кода за рачунарски вид без поновног измишљања точка у нади да ће направити заједничку инфраструктуру на коју би остали програмери могли лако да користе. Управо због тога поседује BSD лиценцу по којој сви могу да користе код у било којим условима.

OpenCV је углавном написан на програмском језику C++ али и даље садржи делове интерфејса на старијем С програмском језику. Поред тога, ради ширења публике, су урађени омотачи за неке друге популарне програмске језике (Java, MATLAB, Python, JS). API-и за ове интерфејсе се лако могу наћи у онлајн документацији.

OpenCV је тренутно једна од најважнијих библиотека за рачунарски вид. То можемо да видимо по томе што је главна библиотека за ROS (Robot Operating System) и да га је чак и NASA користила у неким пројектима (нпр. К9 Марс ровер).

OpenCV садржи мноштво оптимизованих алгоритама за малтене сваку потребу рачунарског вида. Од основних као што је детекција ивица, хистограми и основне трансформације слике (преувеличавање, ротација, развлачење...) до софистицираних техника као што је тражење слике у слици, препознавање ствари (нпр. лица) до чак и техника за коришћење у видео снимцима као што су праћење објеката у видеу.

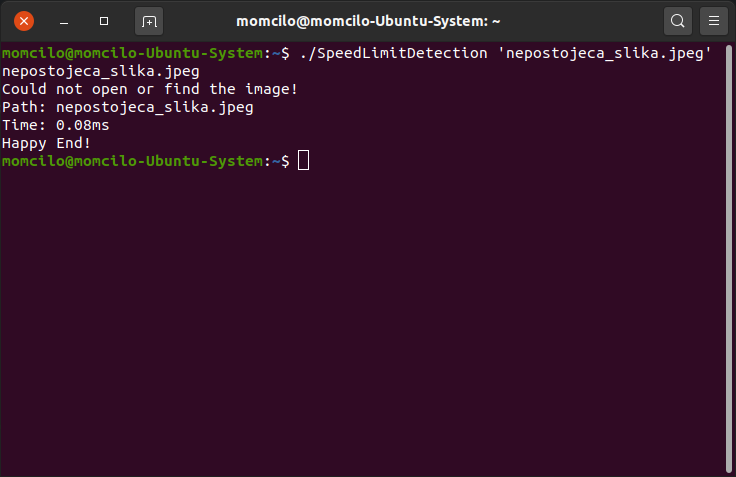
Као што смо раније рекли, већина ових алгоритама користи неки тип машинског учења због своје ефикасности. Тако да их је потребно обучити. Али OpenCV има једноставне функције и за то.

Како и обучавање и коришћење неуралне мреже је врло паралелизован процес, овај framework такође има уграђене оптимизације да искористи графичку картицу за убрзавање. То укључује Intel-ове библиотеке за њихове процесоре са интегрисаном графиком, CUDA убрзавања са Nvidia картицама и OpenGL за остале картице које га подржавају (које су у тренутку писања све остале графичке картице у задњих 20 година).

У овом раду је OpenCV коришћен за трансформацију улазне слике у само њене ивице. Ово је урађено јер нема потребе писати ту функцију поново испочетка а OpenCV је већ тад био гледан у способностима за директну детекцију. Без обучавања OpenCV не може да даје прихватљиве резултате али његове функције за припрему слике за даљу обраду су савршене.

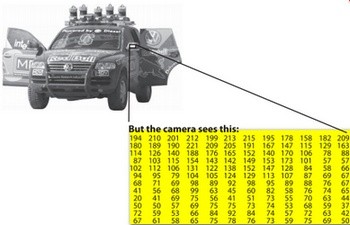
### Припрема слике

На почетку корисник уноси пут до жељене слике као аргумент. Ако слика постоји програм наставља. Ако не, исписује се одговарајућа грешка.



Сл. 1 Грешка када слика не постоји на датом путу

У овом тренутку слика се учитава у Mat променљиву OpenCV-а. Ова променљива је у суштини матрица са 3 елемента. Тј. свако поље матрице има 3 елемента. Један за сваку боју (плава, зелена, црвена тим редоследом).

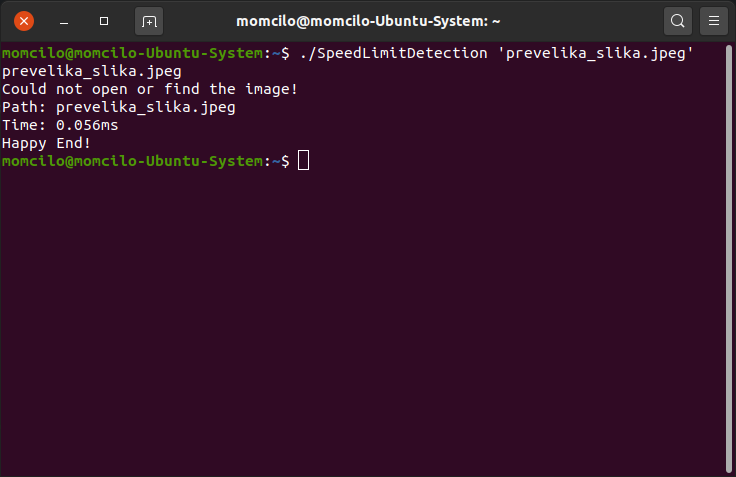


Сл. 2 Mat је само матрица боја пиксела слике

Пољима се приступа преко методе at<T>(row, col) где Т представља тип поља матрице који је уобичајено (по default-у) CV\_8UC3, a row и col ред и колону респективно. CV\_8UC3 нам говори да се ради о OpenCV класи која је у ствари троструки пар 8-битних uchar(0-255) вредности.

OpenCV приказује слике примарно са редом (као нпр. Matlab), па ако су се користиле координате (х, у) овде би се унеле at< CV\_8UC3>(у, х).

Затим програм проверава величину слике. Ако величина није 300х300, програм избацује одговарајућу грешку и прелази на следећу слику. Ако нема даљих слика за обраду, програм се завршава.



Сл. 3 Грешка када је унета слика погрешне величине

Након тога, програм повећава засићеност боја слике ради увеличавања разлика код ивица ради њихове лакше детекције. То ради функција void increaseSaturation(cv::Mat\* srcImage). Она пролази сваки пиксел засебно и ажурира их по формули где је α = вредност измећу 1.0 и 3.0, β = вредност између 0 и 100 а pcv = pixel color value тј. тренутна вредност те боје тог пиксела.

Поред ове формуле овде се још и користи saturate\_cast ради заштите од преливања. Захваљујући овој функцији нова вредност пиксела ће увек бити између 0 и 255. У нашем случају изгледа да је слика благо тамнија после засићења.

|  |  |
| --- | --- |
| ../../saturation%20before.png | ../../saturation%20after.png |

Сл. 4 Улазна слика пре и после засићења

Главна функција овог дела програма користи функције OpenCV-а за трансформацију слике у њене ивице. Али пре тога мора да пребаци нашу слику у њену црно-белу верзију. То ради лума кодингом за стари SDTV(PAL, NTSC, SECAM...). Иако се користе друге формуле за HDTV и HDR ова формула је врло распрострањена и остаће подржана бар још неко време. Формула је следећа: где је pcv = pixel color value тј. тренутна вредност те боје тог пиксела а R G и B црвена, зелена и плава, тим редоследом.

|  |  |
| --- | --- |
| ../../saturation%20after.png | ../../gray.png |

Сл. 5 Слика пре и после пребацивања у црно-белу верзију

Последња промена пред главну трансформацију је blur. Он се користи да би избацили мале оштре линије које нас тренутно не занимају (нпр. лишће окружујућег дрвета, мање флекице на самом знаку...). Ово ће оставити само велике/главне линије које нас занимају. Овде је коришћена обична функција blur која у ствари ради Box blur илити normalized box filter. Он пролази слику по мањим матрицама (у нашем случају 3х3) и замењује њихове појединачне вредности средњом вредношћу мање матрице.

Овде је могао да се користи и GaussianBlur или неки други алгоритам али за овај случај нема разлике у различитим коришћењима алгоритама за замагљивање на крајњи резултат.

|  |  |
| --- | --- |
| ../../../../gray.png | ../../../../blur.png |

Сл. 6 Слика пре и после примењивања blur трансформације

Дошли смо коначно до главне функције пред саму обраду слике. То је Canny функција OpenCV-а. Она користи Canny edge detector за извлачење ивица из слике. Canny edge detector је развио John F. Canny 1986. Детектор је такође познат и као оптимални детектор јер задовољава 3 главна критеријума:

1. Мали проценат грешки: значи да добро детектује само постојеће ивице
2. Добра локализација: раздаљина између детектованих и првих пиксела ивица мора бити што мања
3. Минимални одговор: само један одговор детектора по ивици

Он ради тако што прво узме и одради градијент интензитета као део Sobel-овог филтера (са претпоставком да смо већ одрадили неки тип blur-а или да гарантујемо да он није потребан).

Ту прво направи две маске за х и у осу:

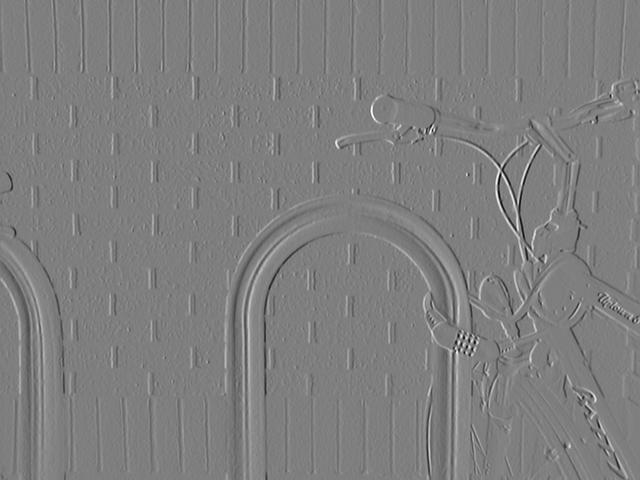
|  |  |
| --- | --- |
| ../Screen%20Shot%202020-07-30%20at%2015.04.04.png | ../Screen%20Shot%202020-07-30%20at%2015.04.13.png |

Сл. 7 Матрице маске за х и у осу

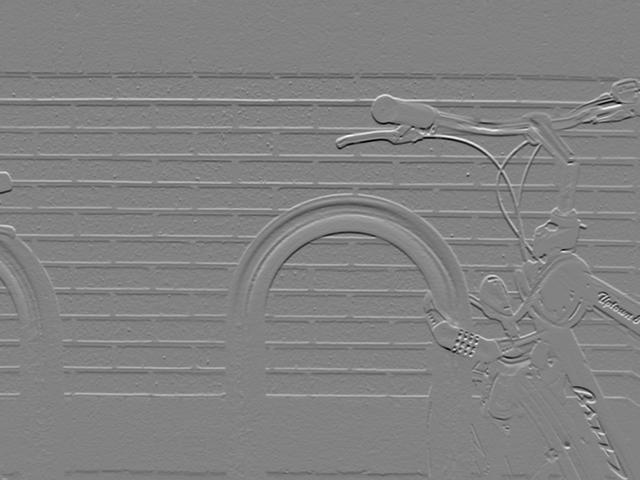
Вредности се добијају множењем маски са мањим матрицама (у нашем случају 3х3).



Сл. 8 Улазна црно-бела слика за пример



Сл. 9 Примери градијената интензитета улазне слике по х оси



Сл. 10 Примери градијената интензитета улазне слике по у оси

А затим снагу и правац:

|  |  |
| --- | --- |
| ../Screen%20Shot%202020-07-30%20at%2015.04.29.png | ../Screen%20Shot%202020-07-30%20at%2015.04.37.png |

Сл. 11 Формуле за снагу и правац

Правац се заокружује на најближи од 4 могућа угла (0, 45, 90, 135).

|  |  |
| --- | --- |
| ../Black_Circle.jpg | ../Sobel_Operator_Gradient_Angle.JPG |

Сл. 12 Пример црне тачке на белој позадини и детектованих праваца њених ивица

Следећи корак је избацивање свих пиксела који не представљају ивицу (такозвани *Non-maximum* suppression). После овога остају само танке линије, кандидати ивица.

Последњи корак ове трансформације је хистереза. Canny користи 2 границе, горњу и доњу (препорука је да се држи однос измећу граница око 2 или 3). У зависности од граница се прави коначна одлука о томе који пиксели се сматрају ивицама а који не по 3 једноставна правила:

1. ако је вредност пиксела већа од горње границе, пиксел се прихвата као део ивице
2. ако је вредност пиксела мања од доње границе он се одбацује
3. ако је вредност пиксела између две границе он се прихвата само ако је поред пиксела који чија је вредност изнад горње границе

У овом пројекту је узета вредност доње границе 90 и по препоруци да је однос горње према доњој 3. Ово је урађено да би алгоритам врло агресивно избацивао све линије за које није апсолутно сигуран да су линије.



Сл. 13 Пример коначне слике са извученим ивицама

Да би лакше видели све пикселе на овим сликама, написана је и функција void blackWhiteIfy(cv::Mat\* srcImage). Она пролази све пикселе, и кад види да неки пиксел није црн(небитно колико таман био) упише њену нову вредност као максималних 255.

|  |  |
| --- | --- |
| ../../canny.png | ../../canny%20white.png |

Сл. 14 Слика са извученим ивицама пре и након избељивања

Што је програма тиче, њему су обе слике исте. Он само проверава да ли је пиксел црн или не. Ово је да би се лакше видело на сликама кад је потребно исцртавање.

### Обрада слике

Први корак у обради је да се нађе горња и доња граница самих цифри у центру слике. То ради функција void cutOffImageAboveAndBelowTheNumbers (cv::Mat\* srcImage). Она креће од средине висине слике и проверава сваки ред до горње петине редова (нема потребе да иде преко тога јер на доброј улазној слици није могуће да дође ни до овог краја, али ако је нека грешка не губимо додатно време за непотребни пролаз редова). Такође ради убрзавања петље се не проверавају сви пиксели у датим редовима већ се иде у трапез.

|  |  |
| --- | --- |
| Initial%20crop%20before.png | Trapezi.png |

Сл. 15 Почетна слика и слика са трапезима за скенирање

|  |  |
| --- | --- |
| Trapez%20Gornji.png | Trapez%20Donji.png |

Сл. 16 Слике појединачних трапеза

Ако у било ком реду се нађе да нема ни један пиксел ивице, тај ред се чува и искаче се из петље. Исто се ради и за доњи трапез за доњу ивицу. Кад се нађу горња и доња ивица делови слике испод и изнад тих граница се одсецају тако да само средина остане за даљу обраду.

Ако се нека ивица не нађе њена вредност ће остати минимална или максимална у зависности од ивице тако да програм неће доћи до неочекиваног понашања. Али врло вероватно остатак програма неће наћи ништа корисно јер не налажење ових граница указује на неку грешку (углавном да се нешто налази на знаку што спречава налажење празног реда).

|  |  |
| --- | --- |
| ../Initial%20crop%20before.png | ../Initial%20crop%20after.png |

Сл. 17 Почетна слика и слика након одсецање горњег и доњег дела

Следећи корак је налажење линија између цифара. То ради функција std::pair<int,int> getLinesBetweenDigits(const cv::Mat\* in). Она прво пролази колоне од 44 до 53 одсто ширине (вуче мало на лево јер код двоцифрених бројева где је десетица 1 празна линија између цифри мало померена лево). Пошто иде по процентима а наша ширина је 300 значи да проверава сваку трећу колону. Ово мало убрзава извршавање и не шкоди самом алгоритму јер на овој величина размак је увек довољан. Ако нађе празну колону онда поставља њену вредност у прво поље повратног пара, а други ставља на -1 да означи да се ради о двоцифреном броју. .



Сл. 18 Приказ свих линија које би се пролазиле за тражење празне

Иначе проверава да ли постоји линија између прве и друге цифре троцифреног броја. То ради проверама колона од 29 до 39 одсто ширине. Ако не нађе ту празну линију поставља обе повратне вредности на -1 и излази из функције. У супротном, ако је нађе, поставља је на прву позицију повратне вредности и креће да тражи линију између друге и треће цифре. Њу тражи у колонама од 56 до 63 одсто ширине. Ако је нађе поставља је на другу позицију повратне вредности и завршава са функцијом. Ако је не нађе, поставља обе повратне вредности на -1 и излази из функције.

У случају троцифреног броја, због чињенице да је у њима прва цифра увек 1 а трећа цифра углавном 0, ове линије између цифара вуку мало у лево што је урачунато у избор ових процената. Колико вуку се може видети на слици 19.



Сл. 19 Визуелизација пређених линија за претрагу празне

Ако је прва позиција повратне вредности -1 у оба случаја знамо да је дошло до грешке при раздвајању цифара. Ако је друга позиција повратне вредности -1 а прва нека друга цифра, знамо да је детектован двоцифрени број а да је вредност на првој позицији број колоне који представља линију између цифара. Ако су обе вредности различите од -1, знамо да је детектован троцифрени број а да вредности на првој и другој позицији представљају бројеве колона између прве и друге, и друге и треће цифре респективно.

Све ове провере линија између цифара ради функција int findEmptyLineBetweenThresholds(const cv::Mat\* in, int thresholdLow, int thresholdHigh). Она ради тако што проверава све колоне између датих граница изражених у процентима.

Овде се за сваку колону позива функција std::vector<int> verticalScan(cv::Mat in, float ratio) која враћа низ тачки у датој колони. Она функционише исто као и std::vector<int> horizontalScan(cv::Mat in, float ratio) само што, очигледно, гледа вертикалне линије на одговарајућем делу слике уместо хоризонталне.

Функција ће детектовати само активне тачке у колони, тј. линије које пресецају ту колону и нису прекинуте у истој (имају бели пиксел у тој колони). Ако дата вертикална линија нема активних тачака у себи, тј. вектор је празан, прелази се на следећи корак провере.



Сл. 20 Тачке нађене у једној вертикалној линији

Та провера је да ли је број те нађене празне колоне мањи или већи од референтне вредности. Референтна вредност је постављена на почетку функције и односи се на то да ли се тражи раздвајање између прве и друге или друге и треће цифре. Та одлука се прави гледањем да ли су границе за скенирање веће или мање од половине ширине слике. Ако се и тај корак прође ажурира се вредност за враћање. Ако се не нађе ни једна празна колона функција враћа -1 као знак да није нашла одговарајућу линију.



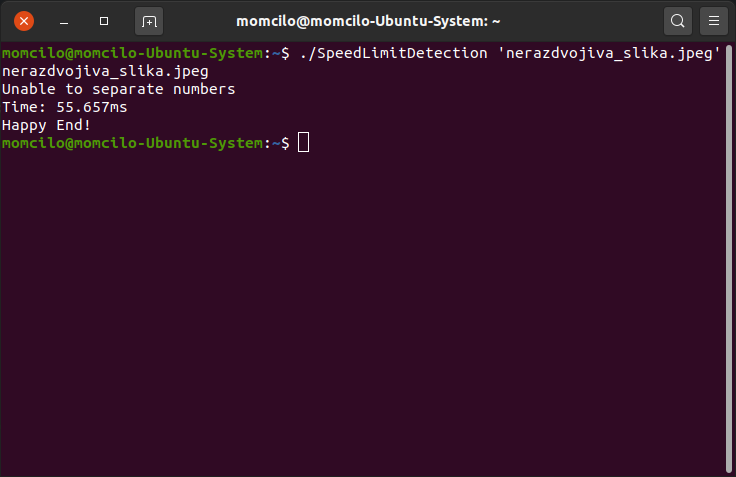
Сл. 21 Иако је у сред броја, ако фале пиксели линија ће и даље бити празна

Због ситуација, као приказано на слици изнад, алгоритам ће за границу узети линију што ближе средини слике. Ово ће нам обезбедити да детектоване линије увек буду у празнини између цифара.



Сл. 22 Детектоване линије између три цифре

Ако је детектована грешка при раздвајању цифри, она се прослеђује назад у главну функцију int getDigits(cv::Mat\* srcImage, cv::Mat\* digit1, cv::Mat\* digit2, cv::Mat\* digit3) која прелази на обраду следеће слике или ако их нема завршава програм-

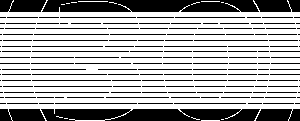


Сл. 23 Приказ грешке при раздвајању цифри

#### **Двоцифрени бројеви:**

У случају да је нађен двоцифрен број, већ знамо линију измећу те две цифре. Сад морамо да нађемо леву и десну границу тог двоцифреног броја.

Да би то урадили користимо функцију std::vector<int> horizontalScan(cv::Mat in, float ratio) на сваком проценту од 10 до 90.



Сл. 24 Линије скениране у току horizontalScan-а

На слици можемо да видимо да нису баш све линије скениране. То је због грешке у заокруживању при пребацивању из float у int.

Након скенирања, ова функција нам враћа низ тачака у том реду. При детектовању тих тачака, функција исто гледа само тачке између 10 и 90 посто. Поред тога, почетна и крајња тачка скенирања нам је дата из функције std::pair<int,int> innerCircleLeftRightEdgeDetection(const cv::Mat\* in, float ratio) која покушава да детектује круг око броја (црвени круг на ивицама знака). То ради тако што игнорише првих и последњих пар пиксела и враћа најближе детектоване пикселе тим границама.



Сл. 25 Пример детектованих тачака у првој линији horizontalScan-а

Као што можемо да видимо на примеру, спољашњи круг је игнорисан. Унутрашњи нисмо могли да игноришемо сад јер је превише далеко од ивице слике и превише близу линијама саме цифре.

Коначна одлука о томе да ли ће дати пиксел бити убачен у повратни вектор зависи од тога да ли је вредност пиксела различита од нула и да следећи пиксел није гранични (ово је из разлога да ако је следећи пиксел гранични онда је висока шанса да припадају истој линији а то нас не занима).

Затим да не би додавали суседне пикселе се врши провера да ли дати пиксел припада линији. Пиксел припада линији ако се вредности два претходна пиксела разликују од нуле.

Коначно за убацивање у вектор иде и последњи услов који каже: Да би се убацио пиксел у вектор његов претходни пиксел мора да има вредност 0 (што је тачно и ако је сам и ако је први пиксел линије) или да његов следећи пиксел има вредност 0 и да је променљива за препознавање линије тачна.



Сл. 26 Детекција линије и означавање тачки

За сваку тачку у вектору се проверава да ли је нека тачка левија од најлевије досадашње тачке (иницијализовано на број колона у слици) или деснија од најдесније досадашње тачке (иницијализовано на 0). Ако јесте ажурира се одговарајућа променљива.



Сл. 27 Иницијалне границе цифри

Сад имамо оквирно границе наших цифри, које треба даље да се сужавају. Почињемо од леве цифре чије су почетне границе за лево наша најлевља тачка коју смо управо нашли. Десна граница нам је централна линија између цифри. А горња и доња граница су нам тренутне границе слике. Све осим оног што је унутар наших граница се одсеца.

|  |  |
| --- | --- |
| ../../levo%20pre.png | ../../levo%20posle.png |

Сл. 28 Исецање по границама пре и после обраде

Даљу обраду цифре извршава void cropOutAlmostEmpty(cv::Mat\* img) функција. Њен посао је да одсече сав вишак око саме цифре. Зато се функција дели у 4 целине. Једну за сваку страну (годе, доле, лево и десно).

Прво се гледа горња страна. Она гледа све редове прве трећине слике (непотребно је гледати даље) и проверава колико има активних пиксела у тој линији. Ако има 1 или мање пиксела детектовано он ажурира нову горњу границу. Стављено је да има ту малу толеранцију ако случајно залута неки пиксел било због неке грешке или најчешће ако има у ћошку неки пиксел окружујућег круга. Петља се обавља до краја осим ако се не детектује бар 10 пиксела што говори да се сигурно већ ушло у саму цифру. Због одсецања на почетку ова петља углавном врло мало итерација изврши.

Доња страна је иста само што се гледа доња трећина слике. У оба случаја углавном се одмах искаче из петље јер је претходно одсецање скоро савршено урадило свој посао. Овде се зато углавном само одсече који празан ред ако је остао. Ово се чешће дешава ако улазна слика није скроз центрирана већ је мало ротирана.



Сл. 29 Линије детектоване за сечење са горње и доње стране

(стављен је црвени рам да би се видела горња линија)

Овде се одсецају вишци са горње и доње стране пре прелаза на следећи део. ово је урађено да би се избацило што више потенцијалних проблематичних пиксела са страна. Најчешћи проблем праве пиксели окружујућег круга. Због њихове кривине могу алгоритму да праве проблем. Па било какво одсецање може да смањи ту шансу.

Лева страна иде од леве границе до половине слике. Мора да иде оволико дубоко јер лева страна леве цифре ако захвати пикселе окружујућег круга може доста да рашири слику. Ово није био проблем за горњу и доњу страну где је, осим у случају сигурне грешке, ово немогуће.

Остатак логике је врло сличан досадашњој. Проверава се да ли нека колона има активне пикселе. Ако има повећава нам бројач. Ако бројач дође до 2 то значи да су нађене две празне колоне(углавном једна за другом). Ту се ажурира лева граница и искаче се из петље. Последња ствар што се деси пре прелажења на десну страну је да се лева граница помери за 1 колону у лево да би се видела последња линија.

Десна страна је слична левој. Креће од десне границе и иде до половине слике. Ту тражи празне колоне да би знао где да повуче линију.



Сл. 30 Линије детектоване за сечење са леве и десне стране

(стављен је црвени рам да би се видела лева линија)

Али овде имамо прву оптимизацију за специфичан број. Додата је нова променљива коју постављамо да је тачна ако детектујемо 1 или 2 пиксела. То је због цифре 7 која може у горњем десном ћошку да буде шпицаста. Ради лакше детекције ми не желимо да одсечемо тај шпиц па зато иде овај услов.

Ово може да се уради јер за десетицу двоцифреног броја десна страна је увек добро одвојена од суседне цифре(иначе не би могао да их одвоји и алгоритам би одавно изашао). А ако се користи за цифру јединица онда исто није проблем јер једина цифра која може ту да се појави је 0 или 5 а они немају тако оштре ивице са те страна. Такође њима је много лакше наћи десну ивицу па не упадају пиксели окружујућег круга у тако малим бројевима.

|  |  |  |
| --- | --- | --- |
| ../../../70%204.png | ../../../70%205.png | ../../../70%206.png |

Сл. 31 Примери различитих седмици са наведеном шпицастом верзијом(скроз десно)

Када су и лева и десна граница пронађене ради се последње одсецање и таква одсечена слика се враћа уместо улазне слике. Та слика се сад ставља у променљиву digit1 која представља прву цифру.

Сада обрађујемо десну цифру. Процедура за њу је иста као и за леву.

1. поставе се почетне вредности за леву и десну границу
2. на основу њих се исече слика за даљи рад
3. функција cropOutAlmostEmpty исече цифру до самих њених линија
4. ново исечена слика се копира у променљиву digit2 која представља другу и последњу цифру која треба да се нађе

#### **Троцифрени бројеви:**

Ако је детектован троцифрени број функција је врло слична као и за двоцифрене бројеве, с тим што имамо додатну цифру у средини. Због саме природе писања више бројева у исти круг, прва и последња цифра су сад ближе окружујућем кругу па функције за њихово исецање морају бити специјализоване.

Иначе, радимо опет std::vector<int> horizontalScan(cv::Mat in, float ratio) да би могли да нађемо крајње леве и десне ивице нашег троцифреног броја. Мала разлика је то што нам је толеранција за налажење десне границе мало блажа сад (колоне\*0.90 уместо колоне\*0.87). То је зато што нам је крајња десна нула сада још ближе ивици. Без овог проширења би одсекли део десне линије нуле па би наша десна област за ту детекцију пропала.



Сл. 32 Линије скениране за налажење леве и десне стране

Као што можемо да видимо са слике изнад, пређене су све линије од 10% до 90% висине. За разлику од двоцифрених бројева где смо имале неке линије које су биле прескочене, овде то није случај. То је зато што у оба случаја се линија за скенирање ради множењем са процентом па пребацивањем у int. Баш овде кад се заокружује број за пребацивање у int утиче сама висина слике. Пошто су нам ови централни исечци нижи за троцифрене бројеве, зато су нам скроз попуњене све линије.

Даље одсецање појединачних цифара иде скоро исто као и за двоцифрене. Нађемо њихове специфичне леве и десне стране. Тако их исечемо и проследимо функцији да одсече све остало.



Сл. 33 Почетне линије за одсецање

Пошто сад имамо и обе средње линије и крајњу леву и десну границу цифара лако се виде специфичне леве и десне стране. Крајња лева граница и средња линија између прве и друге цифре нам представљају леву и десну страну прве цифре. Обе средишње линије нам представљају леву и десну страну друге цифре. И на крају, средња линија између друге и треће цифре и крајња десна граница нам представљају леву и десну страну треће цифре.

|  |  |  |
| --- | --- | --- |
| initial%201.png | initial%203.png | initial%200.png |

Сл. 34 Почетно одсецање цифара

Након грубог одсецања цифри треба да се иде у функцију која ће одрадити коначно одсецање. За одсецање цифри код двоцифрених бројева то је била функција void cropOutAlmostEmpty(cv::Mat\* img). Овде се та иста функција користи само за одсецање средишње цифре. Лева цифра користи void cropOutAlmostEmpty1(cv::Mat\* img), а десна void cropOutAlmostEmpty0(cv::Mat\* img).

|  |  |  |
| --- | --- | --- |
| ../gotove%20cifre%201.png | ../gotove%20cifre%203.png | ../gotove%20cifre%200.png |

Сл. 35 Крајње одсецање цифара

Као што се може закључити из имена функција, оне су сличне. Тачније оне су само оптимизације за прву и последњу цифру које су увек 1 и 0. Оптимизације су биле неопходне јер да би се ставио троцифрен број у знак исте величине као и за двоцифрен сама величина броја је морала да се смањи. Ово такође значи да су крајње (прва и последња/трећа) цифра много ближе окружујућем кругу и то може да представља проблем за добро одсецање.

Оптимизације направљене у void cropOutAlmostEmpty1(cv::Mat\* img) које га разликују од основне верзије су следеће. Мало строжије одсецање са доње стране. Овде чим нађе прву празну линију, означи је за доњу границу и прелази на следећи део(такође је додата још једна линија на доњу линију да би се боље видело на одсецању).

Додато је избацивање празног дела леве стране, ако постоји. То је да би се после убрзало проверавање тих линија са додатним проверама ако не мора. Ово је само да би се сад више исекло пре провере одсецања страница. Горње одсецање је скроз исто. Сад се врши прво одсецање пред наставак на стране.

Велика разлика је да су одсецања за лево и десно овде спојена у целину која у исто време одсеца обе стране. Први део овога је пролажење свих колона слике и тражење најдужих линија. Зато смо у претходном кораку покушали да избацимо празне делове са леве стране слике.

Пошто је ово оптимизовано за тражење јединица знамо да морају да имају 2 праве линије које су приметно дуже од свих осталих линија. Зато у овом пролазу стављамо у низове нађене линије и њихове дужине. Узимамо да је линија прихватљива ако је њена дужина већа од шестине висине.

Пошто слика не мора да је скроз исправљена можемо уместо једне дугачке линије да добијемо пар мањих суседних линија. Зато, кад имамо већ нађене све линије, гледамо које од њих су суседне и остављамо само ону најдужу. Сад кад имамо све различите линије погледамо њихове дужине и извучемо две најдуже. Подешено је тако да је прва линија увек левија од друге.

|  |  |
| --- | --- |
| ../jedinica%20ravna.png | ../jedinica%20kriva.png |

Сл. 36 Пример равне и криве јединице

Са слике можемо да видимо да иако је јединица накривљена да су и даље њене најдуже линије део њене ивице. То је оно што ми тражимо. А пошто су све једна до друге узеће се само најдужи делић те линије.

Иако један може само да се напише као равна црта, то није случај на знаковима за ограничење брзине. Тако знамо да нам је следећи корак да нађемо остале линије нашег броја. Јединица може да има кукицу у горњем левом ћошку, и углавном је има. Може да има и малу хоризонталну линију на дну. У случају хоризонталне линије она исто има вертикалне линије на њеним крајевима које можемо лако да препознамо. А што се тиче кукице, и она углавном има вертикалну линију на крају коју исто можемо лако да нађемо.

|  |  |
| --- | --- |
| ../jedinica%20dole%20ravna.png | ../jedinica%20ravna.png |

Сл. 37 Разлика код писања јединице - са и без серифа

За ово пролазимо још пар пиксела(у нашем случају 20) од наших већ пронађених линија. Са леве стране тражимо линију која има преко 5 пиксела. Ако је нађемо одмах искачемо из петље. Ако не остајемо на већ нађеној главној линији. Са десне стране тражимо прву празну линију. То је зато што знамо да мора да постоји бар једна празна линија између прве и друге цифре. Овако ће или наћи неку ранију линију одмах поред наше цифре или ће доћи до оригиналне раздвојне линије.

Сад кад имамо и леву и десну страну вршимо последње одсецање и враћамо новоисечену цифру.

Функција void cropOutAlmostEmpty0(cv::Mat\* img) је као и претходна void cropOutAlmostEmpty1(cv::Mat\* img) функција, оптимизована за налажење специфичног броја. У овом случају је то 0.

|  |  |
| --- | --- |
| ../../../glavni%20delovi%200.png | ../../../glavni%20delovi%201.png |

Сл. 38 Главни делови логике код последњег исецања за 0 и 1 респективно

Одсецање горње стране је скроз исто као и код одсецања за јединицу. Већ на доњој страни налазимо разлике. Тражење и овде креће од дна слике и иде ка горе(максимално гледа доњих 30% слике). Ажурира доњу границу ако нађе две узастопне линије са једним или мање пикселом. Али има и додатни услов да искочи из петље ако детектује бар 6 пиксела у једној линији. То је из разлога што неке нуле могу да имају равно дно и овај услов нам обезбеђује да не пређемо преко тога беспотребно.

Сад кад се имају горња и дога граница радимо прво одсецање. Ово је такође овде урађено да би се одсекли вишци у горњим и доњим деловима слике који би могли да утичу на то да ли су колоне празне или не.

Слично се траже границе страна. Почињемо са налажењем свих вертикалних линија преко одређене дужине. Сада то више није шестина висине већ је обичних 5 пиксела. Затим се избацују суседне линије на исти начин као и код јединице.

За разлику од јединице која увек има две велике равне линије. Нула може да буде издужена и да их има, али може да буде и зобљена и да скоро нема равне линије. Да би направили ту разлику, можемо да прођемо дужине свих линија и да нађемо најдужу. Ако је преко одређене границе(код нас је 35 пиксела) знамо да је нула издужена. Ако не, знамо да је заобљена. У оба случаја можемо да калибришемо даљу обраду.

|  |  |  |
| --- | --- | --- |
| ../../../nula%20ravna.png | ../../../nula%20zakrivljena%20blago.png | ../../../nula%20zakrivjena.png |

Сл. 39 Нивои заобљености нуле

Пошто знамо да нула мора да има колико-толико равне четири линије то ћемо и да тражимо. Знамо да нам је низ већ сортиран слева на десно, тако да можемо само да избројимо 4 линије преко одређене дужине. Та дужина(15 пиксела за издужену нулу, 10 за заобљену) нам зависи од тога да ли је највећа линија преко дате границе, што смо проверили у претходном кораку. Кад наиђемо на прву такву линију, запамтимо је и повећамо бројач. За следеће две само повећавамо бројач. На крају запамтимо четврту и излазимо из петље.

Сада радимо последње исецање и добијамо нашу последњу цифру. По проналаску сваке цифре она се ставља у одговарајуће променљиве да би остале функције могле да им приступају. digit1 за прву, digit2 за другу и digit3 за трећу.

### Препознавање цифри

Кад су нам све цифре лепо исечене, једино што је остало је да видимо које су то заправо цифре. За ово имамо функцију void detectNumbers(const cv::Mat\* digit1, const cv::Mat\* digit2, const cv::Mat\* digit3) у коју уносимо те исечене цифре, и она нам враћа бројчану вредност сваке слике. Ако му проследимо нулу уместо неке слике, функција ће је само прескочити.

Функција коју зове за проверу цифара је void detectDigit(const cv::Mat\* digit). Одавде се у зависности од редоследа цифре која се проверава траже одговарајући бројеви. За прву цифру троцифреног броја се само проверава да ли је 0 или 1. За другу цифру двоцифреног и трећу цифру троцифреног броја се проверавају само 0 и 5. У свим осталим случајевима се проверавају сви познати бројеви (0-9). У случају налажења броја враћа се његова int вредност. Ако се не нађе ни један или више од једног одговарајућег броја враћа се релевантни код грешке. Функције за провере цифара су у формату detect\*(digit) где \* представља цифру(од 0 до 9).

Све ове функције раде на исти начин: траже да ли у одређеним областима има активних пиксела или не. Ту им припомаже функција

int countInArea(const cv::Mat\* inputDigit, cv::Mat tmp, cv::Point p1, cv::Point p2). Овде прва два аргумента представљају слику на којој се налазе области у којима тражимо пикселе и слику на којој да се исцртају те области ако желимо да их видимо.

Последња два аргумента представљају тачке које означавају два наспрамна ћошка за описивање правоугаоника који је уједно и наша област за претрагу.

Функција прво прави привремену слику исецањем само жељене области из улазне слике. Ту привремену слику можемо проследити OpenCV функцији cv::countNonZero(cropped) која враћа број нецрних пиксела. Тј. у нашем случају сви пиксели који нису црни су активни, а нас баш занима колико њих има. Тачније занима нас да ли их уопште има у датим областима а не њихов тачан број.

Такође, све ове функције на почетку имају проверу која гледа однос њихових редова и колона. Ако је тај однос преко одређене вредности(код нас 2.45) то значи да је цифра превише висока да буде било који број осим 1.

**0:**

Почећемо од прве цифре коју тражимо, нуле. Функција int detect0(const cv::Mat\* digit) нам проверава области за нулу. Она је прва и прилично једноставна. Проверава нам правоугаонике по срединама ширине и висине. Овако можемо да видимо да ли и горе и доле и лево и десно имамо линије. Тј. да ли постоје линије у најважнијим локацијама круга нуле.

Имамо још једну област у центру слике у којем не треба да буде ништа да би знали да је у центру слике рупа. Ако су сви услови испуњени функција исписује 0 на излазу.

Функција враћа одговарајући статус код за налажење дотичне цифре.

../0.png

Сл. 40 Области за детекцију 0

**1:**

Код јединице нам се једино искоришћава услов за однос висине и ширине. Овде ако је услов тачан ће одмах исписати 1 на излаз ис изаћи из функције, док ће за остале цифре само изаћи из функције. Ово нам обезбеђује налажење јединица које су скоро само линија.

Осим тога, има исто условне области као што смо имали и код нуле. Али овде их има мало више из разлога да ако их нема оволико могу да се добијају лажне детекције.

Прво проверавамо доњу леву и десну страну да видимо да ли имамо доњу хоризонталну линију. Овде ако нема доње леве линије одмах знамо да је то јединица јер је област довољно велика и тако позиционирана да би детектовала неки пиксел за сваку другу цифру.

Затим проверавамо горњу леву и десну страну. Овде треба са леве стране има кукица код јединице а са десне стране треба да је празно. Десна страна мора да буде празна да би знали да је то јединица са серифом. Ако није, онда цифра или није јединица или ће однос висине и ширине већ детектовати то и до овога неће се ни доћи.

Такође се проверавају и лева и десна страна по средини ширине. Оне за јединицу су увек празне, ако се дође до ове провере.

На крају је проверава центар цифре да би знали да није случајно детектовао лоше одсечену нулу(случај где се одсеку лева и десна ивица нуле па тај услов може да прође).

Ако је све прошло како треба, исписаће 1 на излазу и ту се завршава функција.

../1.png

Сл. 41 Области за детекцију 1

**2:**

Провера двојке наставља досадашњи шаблон за области које се налазе на срединама висине и ширине и центром. Горња и доња област проверавају горње и доње линије док центар проверава да нема шупљина у центру слике. Лева и десна страна проверавају очекивану празнину са леве стране и линију са десне. Овде нам десна страна осигурава да неће бити мешања са седмицом јер је она ту увек празна.

Додатне две области специфичне за двојку се налазе са доње десне стране. Прва је уска област која проверава да нема ништа у делу изнад хоризонталне доње линије и криве линије која формира остатак цифре. Друга је сам доњи десни ћошак, и проверава само постојање те хоризонталне линије. Ово је још једна провера коју седмица пада.

../2.png

Сл. 42 Области за детекцију 2

**3:**

И тројка остаје са основних 5 области: горе, доле, лево, десно и центар. Све области осим леве очекују да детектују линије. Ово нам одмах гарантује да не може да се збуни за 0, 1, 5 и 7. Лева страна је скраћена по висини али повећана по ширини да би могла довољно да уђе у празни део тројке и да избаци 8 и 9 из могућности за детекцију.

Остају нам доње десне две области од двојке да се ту не би направила грешка. Такође имамо и доњу леву и горњу десну област да не би дошло до забуне са 4 и 6 тим редом. Последња област у горњем левом углу је додатна провера за 7, јер неки фонтови за 7 могу да закаче десну област и да ту прођу проверу.



Сл. 43 Области за детекцију 3

**4:**

Четворка је једноставнија за детекцију јер је специфична и нема много знакова који је користе па нема много варијација фонта. Она има само 4 области: две централне, горњу леву и горњу десну.

Две централне само траже рупу у центру четворке. Мора да их буде две јер не може једна да покрије и ово мало варијација што има. Једини услов за њих је да бар једна буде празна.

Горња десна област је врло приметна због своје висине и кројена је за четворку. Она почиње из горњег десног ћошка и спушта се 60% висине слике. Овако висока област на овој позицији би могла да прође само јединица, ако је већ услов за однос висине и ширине није избацио.

Ако, ипак, успе да прође ову проверу, горња лева област ће је сигурно избацити. Горња лева област је ту да обезбеди да постоји та празнина у горњем левом ћошку. Нема потребе да се проверава цео ћошак, зато је ова област тако уска али и тако позиционирана.



Сл. 44 Области за детекцију 4

**5:**

Петица је најједноставнија за детекцију због своје специфичности да има прилаз својим централним рупама(за разлику од осмице). Због овога нам требају само 3 области за проверу: доња лева, горња десна и централна.

Доња лева је уска област која проверава прилаз ка доњој централној празнини. Горња десна је уска област која проверава прилаз ка горњој централној празнини. И на крају централна област само проверава да ли постоји линија која заобилази претходне области, тј. да ли постоји број а не празна слика.



Сл. 45 Области за детекцију 5

**6:**

Шестица је такође релативно једноставна. Главна одлика јој је горња десна кукица. Зато имамо једну уску област која проверава за њу. Имамо и велику област у горњем левом ћошку која само проверава да ли број постоји.

Једина друга цифра која може да ова два услова прође је петица. Зато су додате још две области у доњем левом ћошку које траже празнине које могу да се нађу код петице и деветке.



Сл. 46 Области за детекцију 6

На примерној слици можемо да видимо да није увек могуће изоловати окружујући круг. Његове остатке можемо видети у горњем и доњем левом ћошку. Али оно што је битно је да лева ивица слике није толико далеко од саме ивице шестице. Зато наше области могу и даље да ухвате своје пикселе и ураде свој посао.

**7:**

Седмица има доста различитих фонтова за приказ. Од најједноставнијег са једном хоризонталном и једном дијагоналном линијом до свакаквих савијања главног тела цифре. Али основне одлике су и даље ту и њих можемо да видимо.

Почевши од основних у центру и горњим ћошковима где знамо да морају да буду линије. У горњим ћошковима знамо да су крајеви горње хоризонталне и да дијагонална линија главног тела мора да пролази кроз центар.

Онда имамо леву и десну страну које су високе да би провериле да су обе стране дијагоналне линије празне. Десна страна иде и до доњег десног ћошка да провери да не постоји доња хоризонтална линија.

На крају имамо две додатне области са леве стране које проверавају празнину одмах испод хоризонталне линије. Има их две јер се не зна колико ће бити дебела хоризонтална линија и колико ће бити савијена дијагонална линија. Овако се покривају сви случајеви и услов је да бар једна од њих буде празна.



Сл. 47 Области за детекцију 7

**8:**

Због сличности између осмице и нуле би изгледало да могу да се користе исте области само да се услов промени да у средини буде попуњено а не празно. Али осмица не мора да буде хоризонтално симетрична. То је зато што се осмица састоји из два круга, један на другом, од којих је доњи углавном мало већи. То ствара шансу за лоше одсецање, где чак иако фале само ивице кругова који представљају осмицу да неке области остану празне. Осмица је једина цифра за коју се уопште не траже празнине, већ само линије.

Управо због сличности са нулом можемо да користимо те области(са мало промењеним димензијама) али ће нам бити потребно још провера. Основне области, лево, десно, горе, доле, исто као и код нуле морају а имају линије кроз себе. Мало су им другачије димензије по странама да би ухватиле неки пиксел иако дође до лошег одсецања. Разлика је у центру који за нулу баш мора да буде празан, овде треба да буде пун.

Додатне области имамо у горњем десном углу где проверавамо да нема празнина намењених 5 и 6. Исто тако имамо и области у доњем левом углу где проверавамо да нема празнина углавном намењених 5 и 9 али које такође могу да се појаве и код 2 и 7. На крају имамо велику област у доњем левом ћошку која је намењена да избаци неке специфичне 1 и 7.

Тек ако су све области испуњене може на излаз да се испише 8.



Сл. 48 Области за детекцију 8

**9:**

За деветку почињемо са стандардним областима за лево, десно, центар и горе. Оне проверавају све четири стране горњег круга цифре. Област за доле је и даље ту али је подељена на две области. Те две нове области су само лева и десна страна старе доње области. Ово је урађено да не би дошло до забуне са седмицом. Овако ће за седмицу бар једна од ових нових области бити празна.

Задржавамо уске области у горњем десном углу ради провере за 5 и 6. Углавном, додајемо две нове области у доњем левом углу за проверу приступа ка доњој централној празнини деветке. Опет, због различитих фонтова имамо две области, а услов је да бар једна буде празна.



Сл. 49 Области за детекцију 9

## Могућности за модификацију

Постоји један случај у којем би програм требао да ради, али не ради. То је у ретким случајевима кад су цифре исувише близу једна другој. Пошто алгоритам ради на принципу провлачења линија кроз слику, ако не може да нађе празну линију између цифара он излази. У овим случајевима иако постоји празна линија између цифара, док она прође трансформацију у своје ивице та линија може да се изгуби.

|  |  |
| --- | --- |
| ../6.jpeg | ../nerazdvojivo_120_6.png |

Сл. 50 Улазна слика и нађене линије између њених цифара

На слици можемо видети тај случај. На оригиналној слици изгледа као да има празна линија између 2 и 0, али након трансформације та линија дефинитивно не постоји. То је зато што и на оригиналној слици имају не-бели пиксели који заустављају празну линију да прође између али су толико бледи да се не виде лако голим оком.

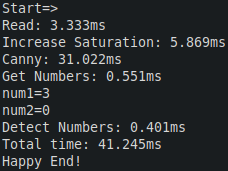
Ово би могло да се превазиђе тако што би додали и проверу јачине боје пиксела при налажењу тих линија. И, препоручљиво би било да се то гледа само у случају раздвајања цифри, из разлога да би неке друге линије могле да падну испод тог прага и да нестану делови линија који нам требају.

На први поглед то не изгледа тако јер су све линије на досадашњим сликама биле беле, али то је зато што је то урадила функција blackWhiteIfy

већ обојила све пикселе, неважно од своје јачине, у бело ради лакше обраде.

Извршавање програма обично траје ~45/~55ms од чега трансформација у ивице траје увелико најдуже (~35ms). Ако узмемо просечну брзину у обзир, ово нам даје могућност покретање 20 слика у секунди што је прихватљиво за некритични видео.

Ради убрзавања могли би да прескочимо корак са повећање засићења (са врло малим компромисом тачности), али би у сваком случају морали да скратимо извлачење линија.



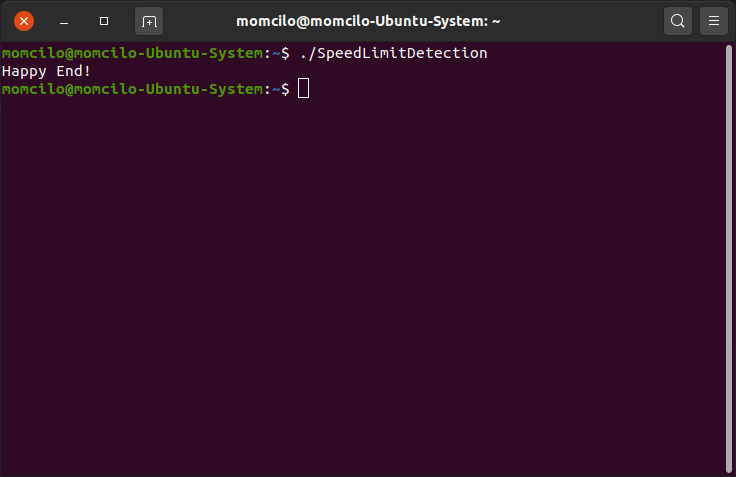
Сл. 51 Времена извршавања делова програма

OpenCV прави врло оптимизоване функције тако да ту нема нешто много додатних перформанси које могу да се извуку. Једино што нам остаје је да некако одмах на почетку смањимо слику којој извлачимо линије. Или да пишемо нову функцију за извлачење линија која неће гледати целу слику.

Такође, треба имати на уму да је потребна нека пред-функција која ће са неке слике нама извући тражену улазну за рад.

## Корисничко упутство

Коришћење је врло једноставно. Програм прима путању слике из аргумената. Било да је то из Eclipse-а или директно из командне линије. На пример, на Linux-у се само дође у фасциклу у којој се програм налази и отвори одатле из командне линије. На Ubuntu-у може и да се десним кликом из фолдера отвори Terminal који је већ у тој фасцикли.



Сл. 52 Пример покретања програма

Програм кад се заврши на излазу исписује Happy End! као ознаку тога. Програм се одмах срећно завршава ако му се не да никакав аргумент за обраду. Ако нека унета слика не може да се обради, исписује се одговарајућа грешка и програм прелази на следећи аргумент за обраду.

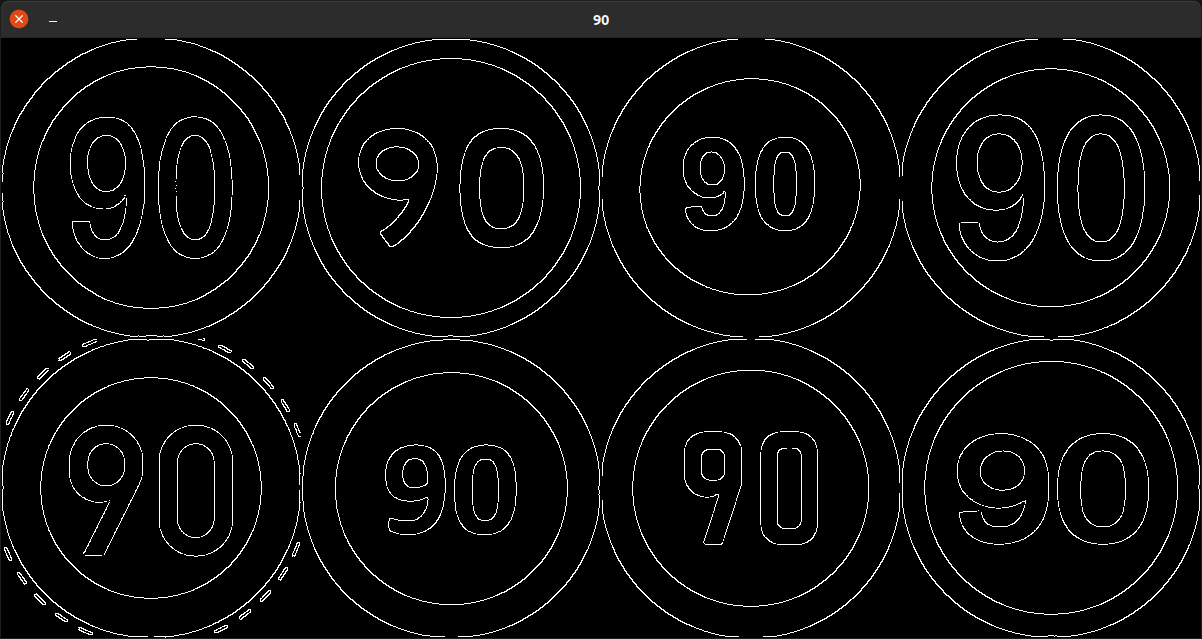
У командној линији се укуца ./SpeedLimitDetection за покретање програма. Након позивања програма се наводе путање до слика. Ако корисник не жели да ручно куца путање могуће је само превући слике на командну линију и она ће учитати њихове одговарајуће путање.



Сл. 53 Пример покретања програма за једну слику из командне линије

Додата је још једна команда за аргументе, а то је тест. Ако се као аргумент унесе -test програм ће покренути унапред изабране слике. Фасцикла са тест сликама мора бити у истој фасцикли као и програм.

Тест функција ће проћи 67 слика разних ограничења брзина и тестирати их. На стандардном излазу ће исписати број слике и шта је детектовао за њу. Поред тога ће отворити прозоре са свим тестираним сликама(са већ извученим линијама) груписаним по ограничењу.



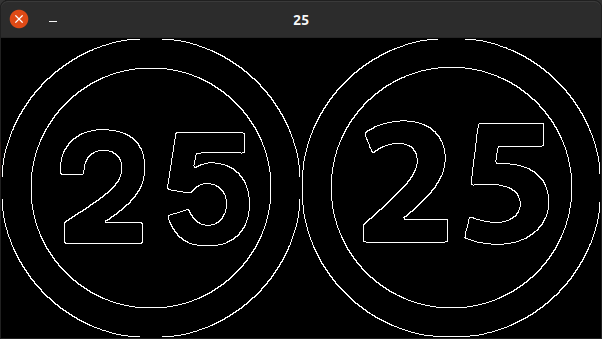
Сл. 54 Тестови за 90



Сл. 55 Тестови за 70

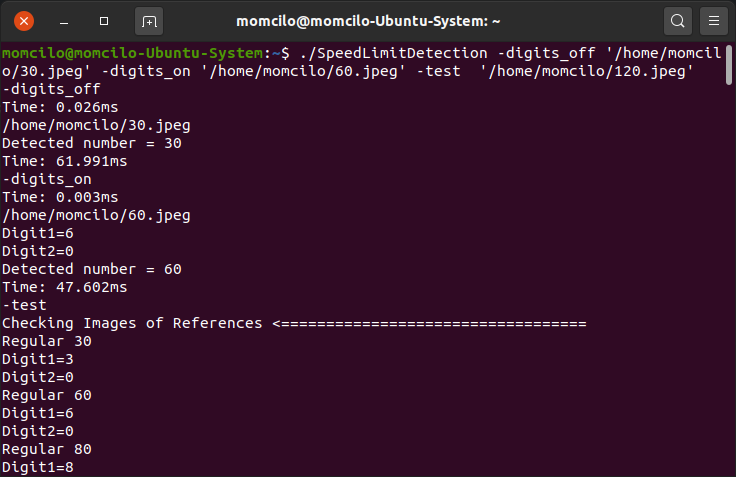


Сл. 56 Тестови разних бројева



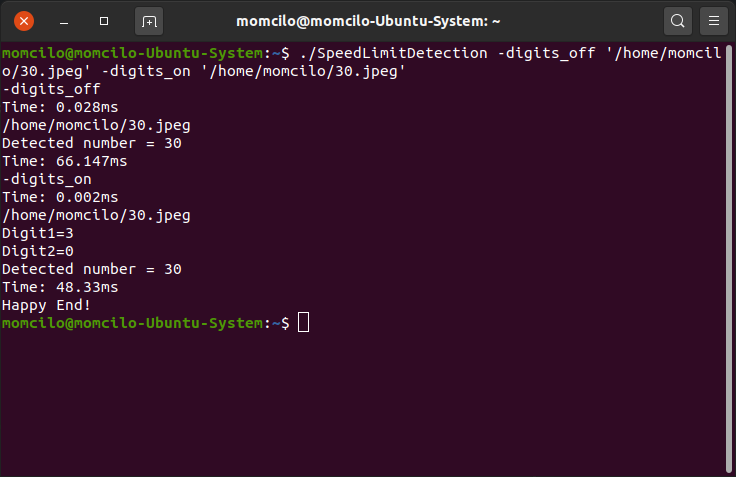
Сл. 57 Тестови за 25

Редослед улазних слика није битан. Тест представља унапред познате слике па не мора да иде на било које специфично место(почетак, крај, итд.). Редослед аргумената утиче на редослед којим ће се они извршавати. Једино где је ово битно је код аргумената -digits\_on и -digits\_off јер ће они утицати на испис слика проверених након њих.

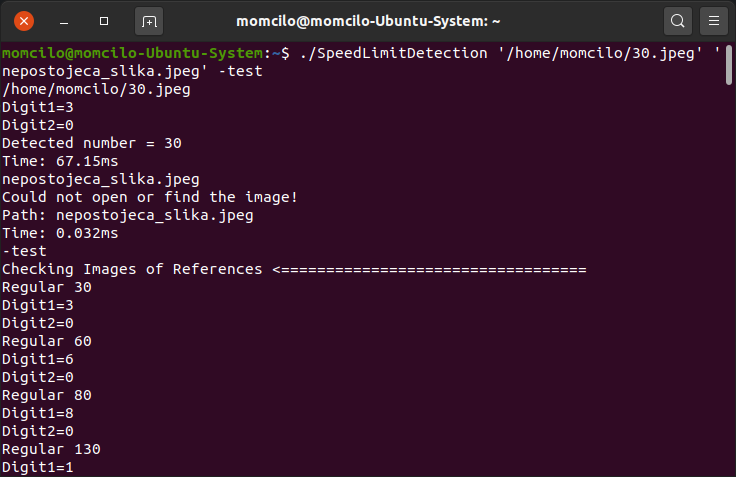


Сл. 58 Извршавање 3 слике и теста

По default-у -digits\_on је активан и све слике ће поред коначног броја исписати и шта је која цифра. С друге стране, -digits\_off ће исписивати само коначан детектован број. Могу се користити више пута у аргументима.



Сл. 59 Паљење и гашење исписа цифара



Сл. 60 Ток програма при налажењу грешке

# Закључак

Овим пројектом смо показали да је могуће вршити препознавање бројева на знаковима без икаквог учења неуралне мреже. С овим не морамо да тражимо бескрајне примере за обучавање и трошимо време на обучавање.

Формална логика нам омогућава да уз пар релативно једноставних правила направимо користан систем. Додавањем, померањем и преувеличавањем области могуће је детектовати скоро све. Ово нам повећава флексибилност ако хоћемо уместо бројева да препознајемо нешто друго.

Управо због овога, и своје модуларне природе, овај програм може да се врло лако убаци у било који систем који треба да зна ограничење брзине на путу. Од самих аутомобила, брзинских камера до обичних система за одржавање самих тих знакова(ако детектује да се нешто променило може да обавести).

Врло лако могу да се додају и други шаблони за проширивање знакова које програм може да детектује.

Једино што је потребно је неки систем да га храни са улазним сликама. Највећа мана му је што суштински зависи од функције за извлачење ивица из слике. Али и тај процес може да се олакша посебним хардвером са високом паралелизацијом. У том случају, тај хардвер не би требао чак ни да троши много енергије, нити да издаје много топлоте чинећи га одличним за уграђене компјутере.

# Литература

* https://opencv.org/
* https://docs.opencv.org/3.4/d3/dc1/tutorial\_basic\_linear\_transform.html
* https://ti.arc.nasa.gov/m/pub-archive/422h/0422%20(Pedersen).pdf
* https://en.wikipedia.org/wiki/Canny\_edge\_detector
* https://en.wikipedia.org/wiki/Sobel\_operator
* https://docs.opencv.org/3.4/da/d5c/tutorial\_canny\_detector.html
* https://en.wikipedia.org/wiki/Speed\_limits\_by\_country
* https://en.wikipedia.org/wiki/Speed\_limits\_in\_Serbia
* https://sr.wikipedia.org/sr-ec/Serif
* https://towardsdatascience.com/recognizing-traffic-signs-with-over-98-accuracy-using-deep-learning-86737aedc2ab
* https://en.wikipedia.org/wiki/Traffic-sign\_recognition
* https://www.design-reuse.com/articles/41154/traffic-sign-recognition-tsr-system.html
* https://www.mathworks.com/help/images/find-image-rotation-and-scale-using-automated-feature-matching.html
* https://aishack.in/tutorials/calibrating-undistorting-opencv-oh-yeah/
* https://en.wikipedia.org/wiki/Speeded\_up\_robust\_features
* https://dsp.stackexchange.com/questions/6194/scale-and-rotation-invariant-template-matching