УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
Трг Доситеја Обрадовића 6, 21000 Нови Сад, Република Србија
Деканат: 021 6350-413; 021 450-810; Централа: 021 485 2000
Рачуноводство: 021 458-220; Студентска служба: 021 6350-763
Телефакс: 021 458-133; e-mail: ftndean@uns.ac.rs
ИНТЕГРИСАНИ СИСТЕМ МЕНАЏМЕНТА СЕРТИФИКОВАН ОД:

PROJEKAT IZ PREDMETA PROJEKTOVANJE ELEKTRONSKIH SISTEMA

NAZIV PROJEKTA:

Automatska garaža

TEKST ZADATKA:

Potrebno je projektovati sistem pomoću računara *Raspberry Pi* koji će detektovati da je automobil došao ispred garaže a potom napraviti sliku sa koje se očitava registarska oznaka automobila. Ukoliko je registarska oznaka u bazi podataka, vrata se otvaraju, u suprotnom ostaju zatvorena. Informacije o automobilima koji ulaze u garažu se šalju na *IoT* platformu.

MENTOR PROJEKTA:

Rajs dr Vladimir

Arbanas Miloš

PROJEKAT IZRADILI:

Vladimir Marćiš, E1 83/2021

Milomir Spajić, E1 87/2021

Aleksandar Kiš, E1 91/2021

DATUM ODBRANE PROJEKTA:

14. 07. 2022.

SADRŽAJ

Uvod

U okviru projekta za predmet *Projektovanje elektronskih sistema*, realizovali smo sistem koji omogućava automatsko otvaranje i zatvaranje vrata garaže ukoliko ispred nje dodje automobil sa registarskom oznakom koja je definisana u bazi podataka.

Sistem je realizovan na računaru *Raspberry Pi 3*. Detekcija automobile vrši se IR senzorom, za pravljenje fotografije automobile koristi se *Raspberry Pi* kamera a za otvaranje i zatvaranje vrata *Micro Servo*.

Informacije o automobile koji je ušao u garažu šalju se na *IoT* platformu *WolkAbout*.

Kompletan kod u realizaciji ovog projekta pisan je u *Python* programskom jeziku.

Glavne komponente korišćene u realizaciji projekta i njihove karakteristike

Glavne komponente koje se koriste u ovom projektu su računar *Raspberry Pi 3*, *Raspberry Pi* kamera, IR senzor, servo motor *Micro Servo* i *WolkAbout* platforma.

Raspberry Pi 3

Raspberry Pi je mini računar koji je razvijen u Velikoj Britaniji za učenje računarskih nauka u školama i zemljama u razvoju. Prvi primjerci ovog računara prodati su u znatno većem broju nego što je bilo očekivano. Danas ovaj računar predstavlja jedan od najprodavanijih računara na svijetu i najčešće se koristi u robotici. Proizvodi se u fabrici *Sony* u Velsu, ali i u Kini i Japanu. Dimenzije ovog računara su 85mm x 54mm x 17mm.

Izgled *Raspberry Pi 3* računara, prikazan je na slici 1.

Slika 1. Raspberry Pi 3

Raspberry Pi 3 Model B+ je poslednji model razvijen u *Raspberry Pi 3* seriji. Osnovne karakteristike ovog modela su:

- Procesor BCM2837B0, Cortex A53 (ARMv8) 64 bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz i 5GHz IEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maksimalna brzina 300Mbps)
- Konektor sa 40 GPIO (eng. General Purpuse Input Output) priključaka
- HDMI
- 4 USB 2.0 priključka

- CSI kamera port za povezivanje Raspberry Pi kamere
- DSI displej port za povezivanje Raspberry Pi displeja osjetljivog na dodir
- 4 pole stereo izlaz i kompozitni video port
- Mikro SD port za snimanje operativnog sistema i čuvanje podataka
- 5V / 2.5A konektor za napajanje

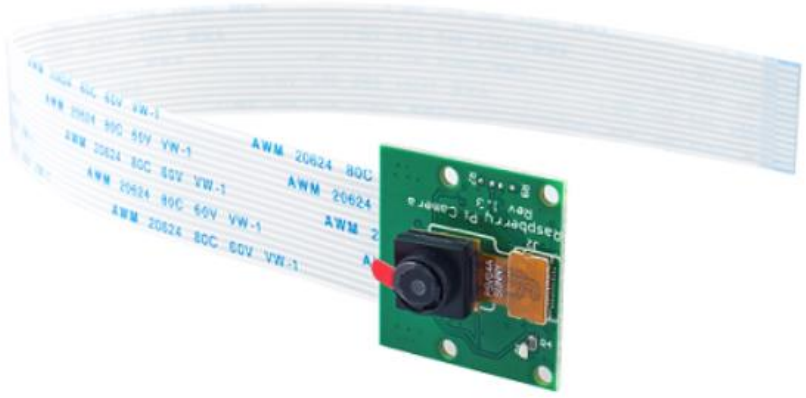
Na *Raspberry Pi* računaru, moguće je korišćenje različitih operativnih sistema kao što su:

- Raspberry Pi OS (Raspbian)
- FreeBSD
- Linux
- NetBSD
- OpenBSD
- Plan 9
- RISC OS
- Windows 10 ARM64
- Windows 10 IoT Core

Tokom realizacije našeg projekta, koristili smo *GPIO* ulaze/izlaze *Raspberry Pi* računara, CSI kamera port i wireless LAN. Operativni system koji je instaliran na računaru je *Raspberry Pi OS – Raspbian* smješten na memorijsku karticu.

Raspberry Pi kamera

Izgled *Raspberry Pi* kamere prikazan je na slici 2.



Slika 2. Raspberry Pi kamera

Ovaj sensor od 5 megapiksela sa kamera modulom OV5647 ima mogućnost snimanja 1080p video zapisa i slika koje se direktno šalju na *Raspberry Pi*. Kamera koristi CSI (eng. Camera Serial Interface) i pomoću flet kabla se povezuje na *Raspberry Pi*.

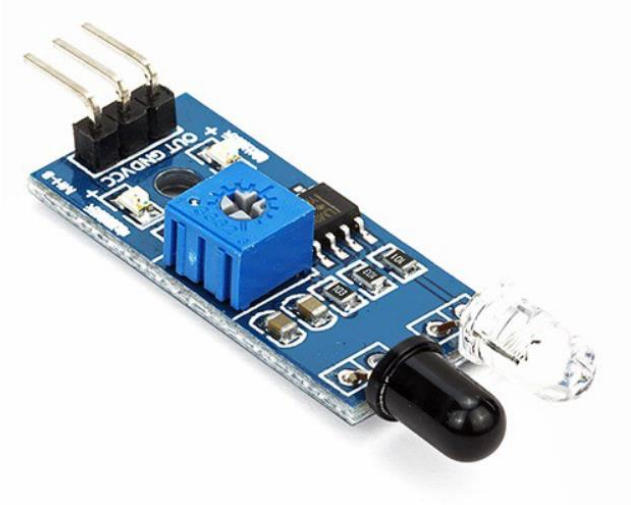
Pločica na kojoj se nalazi kamera je veoma tanka i kompaktna dimenzija 25mm x 23mm x 9mm i težine oko 3g, što je čini veoma pogodnom za razne aplikacije gdje su dimenzije važne.

Ova kamera može da pravi fotografije rezolucije 2592 x 1944 piksela, kao i video zapise 1080p30, 720p60, 640x480p90.

Za realizaciju našeg projekta, koristili smo kamera da dobijemo fotografiju automobila sa koje kasnije očitavamo registarsku oznaku.

IR senzor

IR sensor koji smo koristili pri realizaciji projekta prikazan je na slici 3.



Slika 3. IR senzor

Princip rada IR senzora je veoma jednostavan. Postoje dvije osnovne komponente i to su IR dioda koja predstavlja transmiter i foto diode koja predstavlja risiver. IR dioda emituje infracrvenu svjetlost, koja se odbija od prepreke i vraća nazad da bi je potom detektovala foto dioda i na taj način senzor zaključuje da se ispred njega nalazi prepreka. Signal sa foto diode se poredi sa referentnim naponom i preko komparatora na izlazu se dobija visok ili nizak logički nivo. Referentnim naponom na operacionom pojačavaču se određuje udaljenost prepreke koja se detektuje. Referentni napon, a samim tim i udaljenost prepreke se podešava pomoću potenciometra.

Na senzoru postoje dvije svjetleće (eng. *LED*) diode, jedna od njih signalizira da je napon napajanja senzora prisutan a druga se uključuje kada senzor detektuje prepreku. Potenciometar

se koristi za podešavanje udaljenosti na kojoj se detektuje prepreka. Senzor se napaja jednosmjernim naponom od 5V a na svom izlazu daje nizak naponski nivo kad nema detektovane prepreke, u slučaju da senzor detektuje prepreku, na izlazu će dati visok naponski nivo.

U realizaciji našeg projekta, ovaj senzor se koristi za detekciju da je automobil došao ispred vrata garaže.

Servo motor

Za otvaranje i zatvaranje vrata garaže iskoristili smo servo motor *SG90 Micro Servo*, koji je prikazan na slici 4.

Slika 4. SG90 Micro Servo

SG90 Micro Servo je servo motor relativno malih dimenzija i relativno velike izlazne snage. Osovina motora može da se rotira do 180°.

Neke od osnovnih karakteristika ovog servo motora su sledeće:

- Težina: 9g
- Dimenzije: 22.2mm x 11.8mm x 31mm
- Moment: 1.8 kg cm
- Brzina osovine: 0.1s / 60°
- Radni napon: 4.8V (~5V)
- Opseg dozvoljene radne temperature: 0°C - 50°C

Upravljanje motorom vrši se impulsno – širinskom modulacijom. Potrebno je koristiti signal frekvencije 50Hz. Faktor ispune treba biti u interval od 5% do 10%. Za faktor ispune

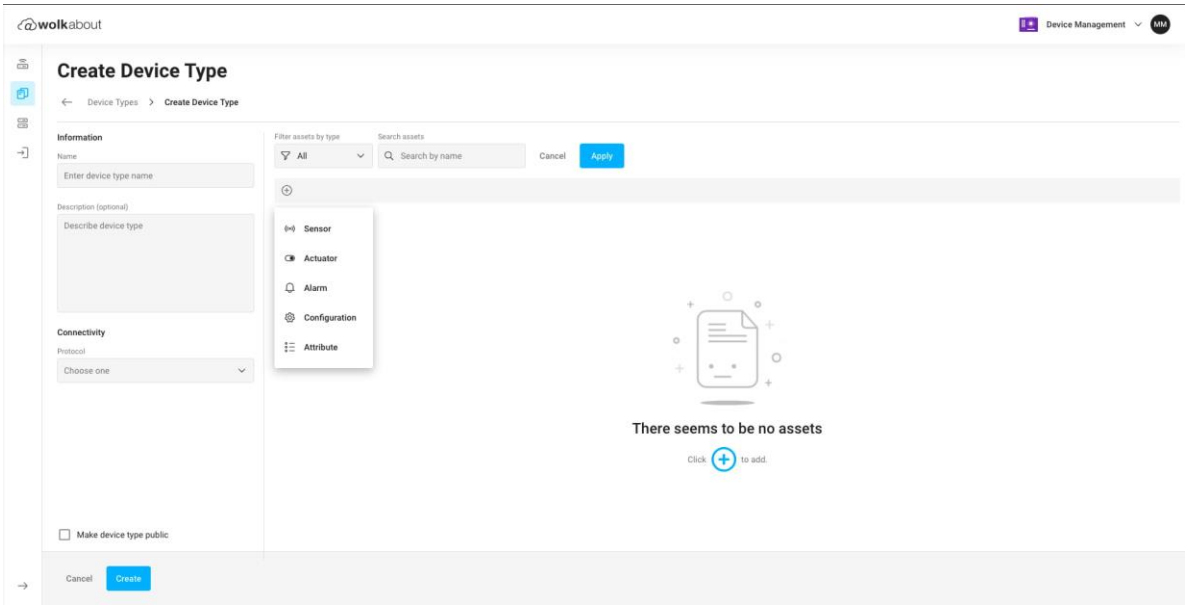
5%, osovina motora je u položaju od 0°, a za faktor ispune od 10%, osovina motora je u položaju od 180°.

U realizaciji našeg projekta, koristili smo ovaj servo motor da simuliramo otvaranje i zatvaranje vrata garaže. Iskoristili smo samo dvije krajnje pozicije, 0° za zatvorena vrata i 180° za otvorena vrata.

WolkAbout platforma – IoT platforma

Internet stvari (engleski: Internet of Things, skraćeno IoT) predstavlja međuumrežavanje fizičkih objekata, vozila, zgrada i drugih stvari sa ugrađenom elektronikom, softverom, senzorima i konektivnošću koji omogućavaju objektima da razmenjuju podatke sa proizvođačem, operaterom i/ili drugim povezanim uređajima. Wolkabout je klasičan primer jedne IoT platformi na kojoj korisnici mogu kreirati svoje uredjeje, slati podatke sa sistema i skladištiti ih na platformi, ili podatke slati prema sistemu i upravljati realnim fizičkim uređajima preko servera.

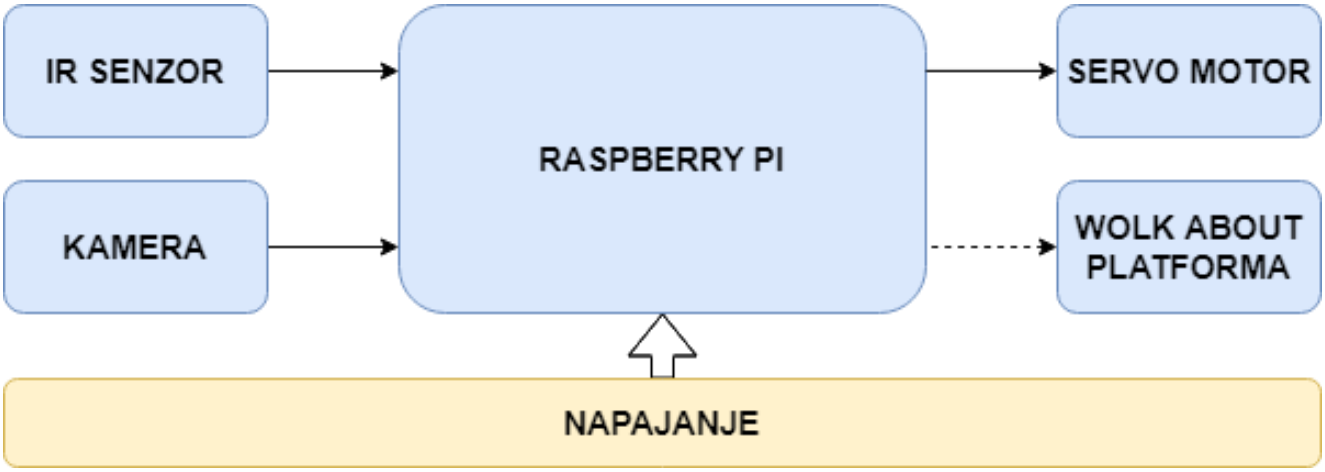
Na slici ispod se može videti sam izgled platforme, odnosno okreženje u kom se korisnik kreira svoj uređaj.



Korisnik na svoj uredjej može dodavati različite senzore, aktuatore koje njihov sistem poseduje i sa kojih žele očitavati vrednosti i skladištiti ih. Takođe platforma omogućava kreiranje grafičkih prikaza podataka koje omogućavaju bolji vizeuleni efekat na korisnika.

Blok šema sistema

Na slici 5. je prikazana blok šema sistema.



Slika 5. Blok šema sistema

Sistem se sastoji od računara *Raspberry Pi 3* koji predstavlja centralni dio i upravlja ostatkom sistema, IR senzora detekcije pokreta, kamere, servo motora i *WolkAbout IoT* platforme. *Raspberry Pi* se napaja sa eksternog napajanja preko *USB Micro B* konektora, dok se ostali moduli se napajaju direktno sa *Raspberry Pi* računara.

IR senzor se napaja sa jednosmjernim naponom od 5V direktno sa *Raspberry Pi* računara. On komunicira preko jedne linije, tako što u slučaju kada nema prepreke ispred njega, linija je na logičkoj nuli, odnosno napon je 0V. Ukoliko se ispred senzora pojavi prepreka na rastojanju koje je manje od unaprijed definisanog, on liniju povlači na logičku jedinicu, odnosno na napon od 5V.

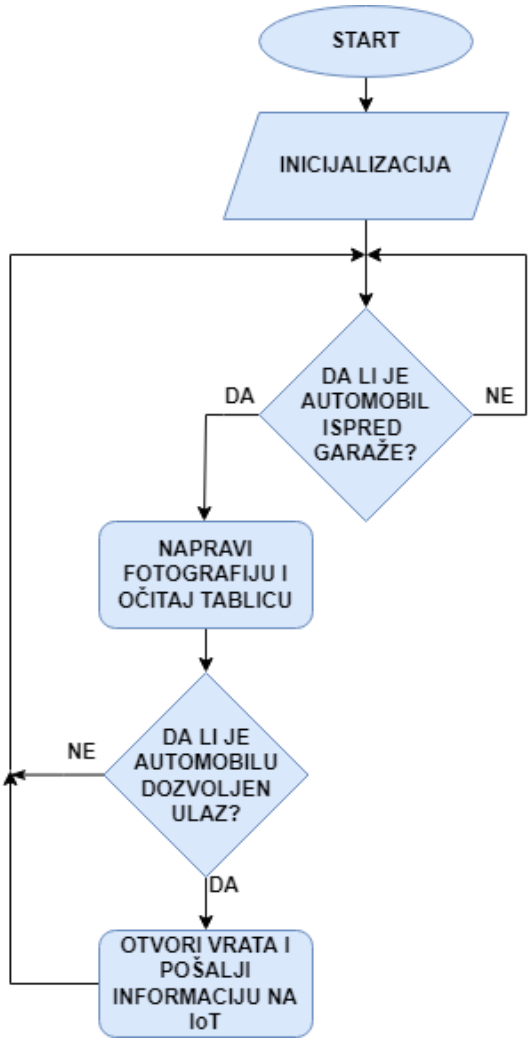
Kamera koja je korištena za realizaciju projekta je *Raspberry Pi Module* kamera. Kad senzor detektuje da je automobile došao ispred garaže, kamera se uključuje i fotografija u realnom vremenu se šalje na *Raspberry Pi*.

Servo motor se takođe napaja sa 5V i radi veoma jednostavno. Komunikacija sa njim, ostvaruje se pomoću impulsno-širinske modulacije. Potrebno je generisati signal frekvencije 50Hz. Faktor ispune treba da bude u intervalu od 5% do 10%, odnosno od 1ms do 2ms.

WolkAbout platforma je *IoT* platforma na koju se šalju podaci tako da je moguće pratiti stanje u garaži sa bilo koje lokacije. Sve što je potrebno jeste pristup internetu. Svaki automobil koji ima pristup garaži može i da uđe u istu i nakon toga se njegova registarska oznaka šalje (eng. publish) na *WolkAbout* platformu.

Algoritam rada sistema

Na slici 6. je prikazan dijagram toka za algoritam rada sistema.



Slika 6. Dijagram tok a za algoritam rada sistema

Na početku izvršavanja koda, potrebno je izvršiti inicijalizaciju svih periferija.

Nakon inicijalizacije, čeka se na dolazak automobile ispred garaže odnosno na aktivaciju IR senzora. Kad senzor detektuje prisustvo automobila ispred vrata garaže, potrebno je uključiti kameru odnosno sačuvati fotografiju sa kamere. Sa sačuvane fotografije, vrši se očitavanje odnosno prepoznavanje registarske oznake. Kada je oznaka sa registarske tablice očitana, vrši se poređenje sa oznakama koje su unaprijed definisane u bazi podataka. Ukoliko tablica nije u bazi, vrši se vraćanje na početak i čeka se novo detektovanje od strane senzora. Ukoliko je registarska oznaka u bazi, vrši se otvaranje vrata tako da automobil može ući u garažu kao i slanje oznake na *WolkAbout* platformu tako da je moguće imati pristup informacijama o svim automobilima koji se nalaze u garaži sa bilo koje lokacije.

Kod

Kod za realizaciju projekta, u potpunosti je pisan u *Python* programskom jeziku. Organizovan je tako da postoji jedan, glavni fajl koji se pokreće. U njemu se uvoze biblioteke koje su pravljene za pojedine module kao što su IR senzor, kamera, obrada slike, servo motor i *wolk about*. Nakon toga se implementira algoritam rada programa tako što se pozivaju funkcije definisane u bibliotekama.

Glavni dio koda

Prvo se vrši uvoz (eng. import) biblioteka koje su podržane od strane *Python*-a. U ovom slučaju, to je funkcija *sleep* iz biblioteke *time*.

```
# External module imports
from time import sleep
```

Nakon što su uvezene sve generičke biblioteke, vrši se uvoz biblioteka koje su kreirane isključivo za potrebe ovog projekta. To su biblioteke za IR senzor, kameru, *Wolk* platformu, servo i obradu slike. S obzirom da *WolkAbout* platforma više nije u funkciji, biblioteka je zakomentarisana jer se ne može koristiti.

```
# Internal module imports
from irsensor import *
from camera import *
# from wolk_publish import *
from servo import *
from image_processing import *
```

Potrebno je definisati funkciju koja vrši provjeru da li je automobilu sa očitanom registarskom oznako dozvoljen pristup garaži. To se vrši pomoću sledeće funkcije:

```
# Function for table check
def check_table (table):
   table1_1 = „BG“
   table1_2 = „901“
   table1_3 = „BD“
   table2_1 = „NS“
   table2_2 = „659“
   table2_3 = „RR“
   if(table1_1 in table and table1_2 in table and table1_3 in table)
      return True
   if(table2_1 in table and table2_2 in table and table2_3 in table)
      return True
   return False
```

U ovoj funkciji, definisano je šest stringova. Tri stringa, koriste se za definisanje jedne registarske oznake i nakon toga se provjerava da li string očitane tablice sadži svaki od tri definisana stringa. Ovo se radi da bi se izbjegle greške koje nastaju pri očitavanju tablice, kao što je pojava dva uzastopna razmaka (eng. *Space* karaktera), očitavanje prelaska u novi red (eng. *Enter*) i slično.

Na kraju je realizovan i glavni dio koda koji sekvencijalno u beskonačnoj petlji poziva sve definisane funkcije koje su prikazane u nastavku u redosledu koji zapravo omogućava ispravno funkcionisanje željenog algoritma rada.

```
while 1:
   print(„Wait for IR“)
   wait_for_IR()
   print(„IR detected“)
   print(„Take a picture:“)
   take_a_picture()
   print(„Picture is ready!“)
   table = process_image()
   print(„Tablica“, table)
   table_correct = check_table(table)
   if(table_correct):
      open_door()
      print(„Opened door“)
      sleep(5)
      close_door()
      # publish_wolk(table)
      # print(„Table is published“)
   else:
      print(„Incorrect table“)
```

Biblioteka za IR senzor

Prvo se vrši uvoz (eng. import) neophodnih bibloteka. U ovom slučaju to je biblioteka za manipulaciju *GPIO* priključcima *Raspberry Pi*-a i funkcija *sleep* iz biblioteke *time*.

```
# External module imports
import Rpi.GPIO as GPIO
from time import sleep
```

Nakon toga, vrši se definisanje priključka na koji se povezuje IR senzor i podešavanje priključka. Podešen je kao izlazni, sa internim *pull-down* otpornikom.

```
# Pin Definitions
IRsensorPin = 18
```

```
# Pin Setup
GPIO.setmode(GPIO.BCM) # Broadcom pin-numbering scheme
GPIO.setup(IRsensorPin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

Najvažniji dio ove biblioteke je svakako funkcija za detekciju IR senzora. Ona je definisana tako da omogućava blokiranje izvršavanja daljeg programa sve dok na priključak na koji je vezan IR senzor ne dodje rastuća ivica signala.

```
def wait_for_IR():
   GPIO.wait_for_edge(IRsensorPin, GPIO.RISING)
   sleep(1)
```

Biblioteka za kameru

Prvo se vrši uvoz (eng. import) neophodnih biblioteka. U ovom slučaju su to biblioteka za manipulaciju kamerom, odnosno funkcija *PiCamera* iz istoimene biblioteke. Iz biblioteke *time* koristi se funkcija *sleep*, a za snimanje fotografije sa kamere potrebna je biblioteka *os* koja omogućava pristup sistemskim podacima.

```
# External module imports
from picamera import PiCamera
from time import sleep
import os
```

Inicijalizacija kamere se vrši pozivom funkcije PiCamera().

```
# Camera init
camera = PiCamera()
```

Definisana je funkcija koja vrši snimanje fotografije i smještanje u projektni folder. Ova funkcija prvo provjerava da li već postoji fotografija sa istim nazivom i na istoj lokaciji i ukoliko postoji automatski se briše. Nakon toga se uključuje kamera i slika se čuva u projektnom folderu.

```
def take_a_picture():
   if(os.path.isfile('/home/pi/Desktop/PES_Projekat/tablica.jpg')):
      os.remove('/home/pi/Desktop/PES_Projekat/tablica.jpg')
   else:
      print("Picture doesn't exist")
   camera.start_preview()
   sleep(5)
   camera.capture('/home/pi/Desktop/PES_Projekat/tablica.jpg')
   camera.stop_preview()
```

Biblioteka za obradu slike

Prvo se vrši uvoz (eng. import) neophodnih bibloteka. U ovom slučaju to su biblioteke *cv2, imutils* i *pytesseract.* Biblioteka *cv2* je biblioteka koja se koristi za obradu slike pre samog izvlačenja teksta iz slike ali takođe se koriste i funkcije iz ove biblioteke koje služe za prikaz samih slika u toku obrade. Biblioteka *imutils* je biblioteka koja se koristi za promenu dimenzija slike koja se obrađuje. Biblioteka *pytesseract* je biblioteka koja se koristi za izvlačenje samog teksta sa slike. Konkretno u ovom našem slučaju za izvlačenje registarske tablice sa određene slike.

```
# External module imports
import cv2
import imutils
import pytesseract
```

Glavni deo koda za obradu slike zapravo predstavlja funkcija *process_image.* U nastavku će biti detaljno objašnjena cela ova funkcija.

Na početku funkcije, vrši se importovanje same slike kao i promena veličine te slike. Veličina slike se menja tako što se širina slike postavlja na 600 piksela. Zatim se tako promenjena slika prikazuje pomoću funkcije *imshow*. Pomoću funkcije *waitKey* se obezbeđuje vreme koliko će određena slika biti prikazana u milisekundama. Ukoliko se ovoj funkciji prosledi vrednost 0, to znači da će slika biti prikazana dokle god se "ručno" ne isključi dati prozor.

```
image = cv2.imread('tablica.jpg')
image = imutils.resize(image, width=600)
cv2.imshow("1. original image", image)
cv2.waitKey(0)
```

Sledeći korak u obradi slike jeste da se ona prebacuje u sivu sliku. To se radi uz pomoć funkcije *cvtColor.* Zatim se takva slika prikazuje u novom prozoru.

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("2. greyed image", gray_image)
cv2.waitKey(0)
```

Zatim se na sivu sliku primenjuje funkcija *bilateralFilter*. Uz pomoć ove funkcije se zapravo smanjuje šum koji postoji na sivoj slici i slika postaje više jasnija. Ovako obrađena slika se takođe prikazuje u novom prozoru.

```
gray_image = cv2.bilateralFilter(gray_image, 11, 17, 17)
cv2.imshow("3. smoothened image", gray_image)
cv2.waitKey(0)
```

Na ovakvu sliku, bez šumova, se primenjuje funkcija *Canny* koja zapravo detektuje sve moguće ivice na slici i označava te ivice. I takva slika se zatim prikazuje u novom prozoru.

```
edged = cv2.Canny(gray_image, 30, 200)
cv2.imshow("4. edged image", edged)
cv2.waitKey(0)
```

Zatim se na ovakvu sliku, na kojoj su označene sve ivice, primenjuje funkcija *findCountours.* Ova funkcija zapravo vadi sve moguće konture sa slike i uklanja sve suvišne tačke na otkrivenim konturama. Promenljiva *cnts* zapravo sadrži sve konture koje je pronašla ova funkcija. Takođe, pravi se i kopija originalne slike, jer ne želimo da menjamo sadržaj originalne slike. Uz pomoć funkcije *drawContours* se iscrtavaju pronađene konture na slici koja predstavlja kopiju originalne slike. Takva slika se zatim prikazuje u novom prozoru.

```
cnts,new=cv2.findContours(edged.copy(),cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
image1=image.copy()
cv2.drawContours(image1,cnts,-1,(0,255,0),3)
cv2.imshow("5. contours",image1)
cv2.waitKey(0)
```

Sledeće što se radi u obradi slike jesto to da se sortiraju pronađene konture. To se radi uz pomoć funkcije *sorted*. Ova funkcija sortira sve konture čija površina je veća od 30. Konture sa površinom ispod 30 se zanemaruju. Kontura koja sadrži registarski broj se čuva u promenljivoj *screenCnt.* Zatim se pravi nova kopija originalne slike i crtaju se sortirane konture na toj kopiji. Takva kopija se takođe prikazuje u novom prozoru.

```
cnts = sorted(cnts, key = cv2.contourArea, reverse = True) [:30]
screenCnt = None
image2 = image.copy()
cv2.drawContours(image2,cnts,-1,(0,255,0),3)
cv2.imshow("6. Top 30 contours",image2)
cv2.waitKey(0)
```

Naredni korak u obradi slike jeste pronalaženje konture sa četiri strane. U jednoj for petlji koja se ponavlja onoliko puta koliko je kontura pronađeno na slici, se koristi funkcija *arcLength* . Ova funkcija zapravo vraća obim određene konture. Takođe, koristi se i funckija *approxPolyDP* koja aproksimira obim sa određenom preciznošću. Zatim se proverava da li je obim konture jednak broju četiri i ukoliko jeste takva kontura se čuva u promenljivoj screenCnt jer je velika verovatnoća da je takva kontura zapravo kontura sa registarskim brojem. Zatim se vrši isecanje dela slike koja zapravo predstavlja konturu gde se nalazi registarski broj. Funkcija *boundingRect* vraća četiri broja i ta četiri broja zapravo predstavljaju koordinate četiri temena pravougaonika u kome se nalazi registarski broj. Zatim se pravi nova slika koja zapravo predstavlja isečen deo originalne slike na osnovu temena pravougaonika u kom je detektovan registarski broj. Takva slika se i čuva pod određenim nazivom.

```
i=7
for c in cnts:
        perimeter = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.018 * perimeter, True)
        if len(approx) == 4:
                screenCnt = approx

        x, y, w, h = cv2.boundingRect(c)
        new_img = image[y:y + h, x:x + w]
        cv2.imwrite('./' + str(i) + '.png', new_img)
        i += 1
        break
```

Sledeći korak u obradi slike jeste taj da se očitava slika koja je sačuvana u prethodno objašnjenom delu. Ovde je napravljena jedna pretpostavka a to je da je u pitanju slika pod nazivom *7.png* jer je pretpostavljeno da je prva slika koja ima obim veći od četiri zapravo slika koja sadrži registarski broj. Kada se učita ova slika, vrši se uvećanje te slike uz pomoć funkcije *resize* iz biblioteke *cv2.* Ovoj funkciji se prosleđuje zapravo slika kojoj se menja dimenzija, željene dimenzije nove slike i faktor skaliranja slike po x i po y osi. Ukoliko se kao željene dimenzije nove slike prosledi (0,0), veličinu nove slike određuje faktor skaliranja po x i y osi. Dobija se uvećana slika koja je uvećana tri puta po svakoj osi i koja se zatim prikazuje u novom prozoru.

```
Cropped_loc = cv2.imread('7.png')
Cropped_loc = cv2.resize(Cropped_loc, (0, 0), fx=3, fy=3)
cv2.imshow("7. cropped", Cropped_loc)
cv2.imwrite("7_cropped.png", Cropped_loc)
cv2.waitKey(0)
```

Zatim se na ovako uvećanu sliku primenjuju tri funkcije: *get_grayscale, thresholding* i *opening.* Definicije ovih funkcija su sledeće:

```
def get_grayscale(image):
     return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

def thresholding(image):
     return   cv2.threshold(image,   0,   255,   cv2.THRESH_BINARY   +
cv2.THRESH_OTSU)[1]

def opening(image):
     kernel = np.ones((5, 5), np.uint8)
     return cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
```

Funkcija *get_grayscale* vrši konverziju slike u sivu sliku.

Funkcija *thresholding* postavlja svaki piksel na slici na vrednost svog praga, odnosno svaki piksel koji se nalazi ispod svog praga postavlja na nulu a u suprotnom se postavlja na

maksimalnu vrednost. Najjednostavnije je reći da ova funkcija svetle delove sive slike prikazuje kao bele a tamnije delove predstavlja crnom bojom.

Funkcija *opening* zapravo male objekte iz prvog plana slike stapa u pozadinu te slike.

Nakon primene svake od ove tri funkcije, dobija se nova slika i svaka se prikazuje u zasebnom prozoru.

```
gray = get_grayscale(Cropped_loc)
cv2.imshow("8. grayed image", gray)
cv2.imwrite("8_grayed_image.png", gray)
cv2.waitKey(0)

thresh = thresholding(gray)
cv2.imshow("9. thresholding image", thresh)
cv2.imwrite("9_thresholding_image.png", thresh)
cv2.waitKey(0)

thresh = opening(thresh)
cv2.imshow("10. opened image", thresh)
cv2.imwrite("10_opened_image.png", thresh)
cv2.waitKey(0)
```

Na sliku koja je dobijena nakon primene funkcije *thresh* se primenjuju još dve funkcije a to su *GaussianBlur* i *medianBlur* iz biblioteke *cv2*.

Funkcija *GaussianBlur* zapravo predstavlja funkciju koja na sliku primenjuje Gausov filter odnosno funkcija koja vrši Gausovo zamućenje slike. Gausov filter predstavlja niskofrekventni filter koji uklanja visokofrekventne komponente slike.

Funkcija *medianBlur* predstavlja funkciju srednjeg zamućenja To znači da se vrednost svakog piksela zamenjuje srednjom vrednošću svih okolnih piksela. Ova funkcija najviše utiče na zamućenje ivice slova na slici.

Nakon primene svake od ove dve funkcije, dobija se nova slika koja se prikazuje u posebnom prozoru.

```
thresh = cv2.GaussianBlur(thresh, (11, 11), 0)
cv2.imshow("11. gaussian image", thresh)
cv2.imwrite("11_gaussian_image.png", thresh)
cv2.waitKey(0)

thresh = cv2.medianBlur(thresh, 9)
cv2.imshow("12. ready image", thresh)
cv2.imwrite("12_ready_image.png", thresh)
cv2.waitKey(0)
```

Na ovako obrađenu sliku se sada primenjuje *pytesseract* algoritam. Ovaj algoritam zapravo služi za izvlačenje teksta iz obrađene slike. Ovom algoritmu se prosleđuje *PSM* (eng. Page

segmentation mode) sa vrednošću 10. To znači da ovaj algoritam tretira ceo tekst sa slike kao jedan karakter. Ovaj način se pokazao kao najbolji.

```
config = r"--psm 10"# --oem 3"
plate = pytesseract.image_to_string(thresh, config = config)
```

Dakle, u promenljivoj *plate* se nalazi smešten registarski broj. Na ovu promenljivu se primenjuje još jedna funkcija a to je funkcija *filter_plate*. Ova funkcija je definisana na sledeći način:

```
def filter_plate(text):
   k=
"1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ -—";
    getVals = list(filter(lambda x:x in k, text))
    result = "".join(getVals)

    return result
```

Ova funkcija zapravo prolazi kroz tekst koji joj se prosledi i briše sve karaktere koji se nalaze u datom tekstu a ne nalaze se u stringu koji je označen slovom *k*. Ova funkcija služi za otklanjanje simbola kao što su zagrade, krugovi i polukrugovi jer postoji mogućnost da okvir oko registarskog broja bude protumačen kao navedeni simboli od strane *pytessaract* algoritma.

Dakle, poslednja funkcija koja se poziva jeste funkcija *filter_plate*. Takođe, na kraju funkcije *process_image* se zatvaraju svi prethodno otvoreni prozori sa slikama i vraća se tekst koji zapravo predstavlja registarski broj.

```
plate = filter_plate(plate)
cv2.destroyAllWindows()

return plate
```

Biblioteka za servo motor

Prvo se vrši uvoz (eng. import) neophodnih bibloteka. U ovom slučaju to je biblioteka za manipulaciju *GPIO* priključcima *Raspberry Pi*-a i funkcija *sleep* iz biblioteke *time*.

```
# External module imports
import Rpi.GPIO as GPIO
from time import sleep
```

Nakon toga, vrši se podešavanje priključka kojim se upravlja servo motorom. Potrebno je generisati *PWM* signal frekvencije 50Hz.

```
# Pin Definitions
ServoPin = 19
```

```
# Pin Setup
GPIO.setmode(GPIO.BCM) # Broadcom pin-numbering scheme
GPIO.setup(ServoPin, GPIO.OUT) # Pin set output

# PWM Setup
servo_pwm = GPIO.PWM(ServoPin, 50) # Frequency 50Hz, period 20ms
servo_pwm.start(0)
```

Glavni dio koda ove biblioteke su funkcije koje omogućavaju otvaranje i zatvaranje vrata. Za otvaranje vrata, potreban je faktor ispune od 1ms za zakretanje servoa na 0°, dok je za otvaranje vrata faktor ispune 2ms, odnosno 10%.

```
def close_door():
   servo_pwm.ChangeDutyCycle(5) # Set duty cycle 5% (1ms)
   sleep(0.1)

def open_door():
   servo_pwm.ChangeDutyCycle(10) # Set duty cycle 10% (2ms)
   sleep(0.1)
```

Biblioteka za WolkAbout

Za povezivanje na IoT platformu WolkAbout potrebno je uvesti biblioteku *wolk* iz koje se koriste ključne funkcije kako za samo povezivanje realizovanog sistema tako i za slanje podataka u ovom slučaju od sistema prema platformi. Nakon što je biblioteka uveze i napravljen uređaj na platformi potrebno je pomoću funkcije *wolk.Device(key= , password=)* inicijalizovati uređaj odnosno postaviti key i password koju nam je platforma izgenerisala pri pravljenju uređaja da bi se naš sistem povezao uspešno na željeni uređaj na platformi, zatim sve to pripremiti za slanje pomoću funkcije *wolk.WolkConnection(device)* zatim te podatke smeštamo u promenljivu *wolk_device* i pomoću funkcije *wolk_device.connection()* otpočinje uspostavljanje konekcije sa platformom.

```
device = wolk.Device(key="kilerobija", password="68QTIAMT8")

wolk_device = wolk.WolkConnection(device)

wolk_device.conncention()
```

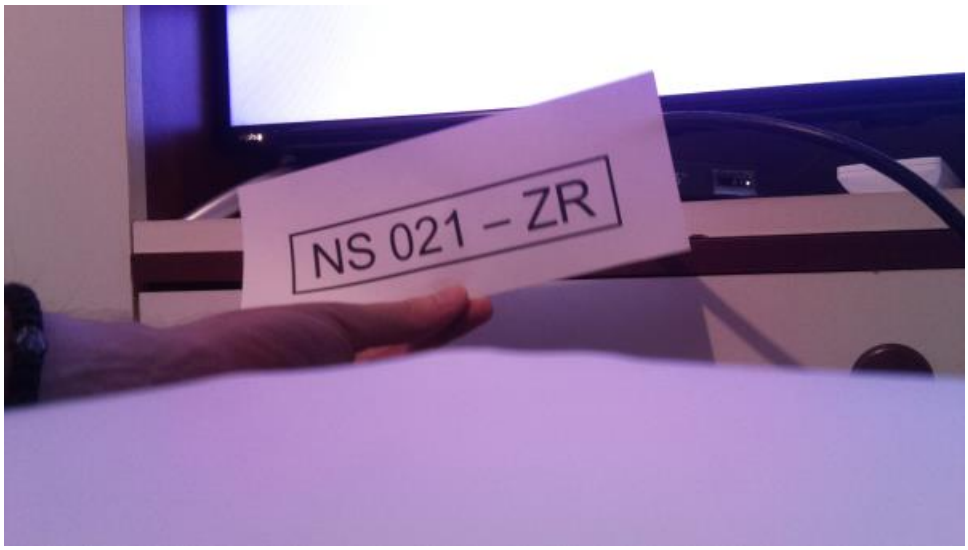
Nakon što su se sistem i platforma povezali, pomoću funkcije wolk_device.publish() sa sistema se šalju informacije prema platformi. Prvo sistem prosleđuje string odnosno tablicu koja je očitana na kameri funckiji publish_wolk() gde ona taj broj tablica smešta u *wolk_device.add_sensor_reading()* potrebno staviti referencu koja se postavlja i na platformi, jer jedan uređaj na platformi moze da ima više senzora/aktuatora pa da bi platforma tačno znala gde ga smesti pristigli podatak koristi se referentna reč koja stize zajedno sa podatkom i poklapa se sa referentnom rečju tačno jednog senzora/aktuatora uređaja na platformi. Takodje

ubačena opcija prekidanja komunikacije sistema i platforme pritiskom nekog od tastera na tastaturi.

```
def publish_wolk(tabl):
      try:
            tablica = tabl;
            wolk.device.add_sensor_reading("broj_tablica", tablica)
            print(f'Broj tablice: {tablica})
            wolk.device.publish()
      except KeyboardInterrupt:
            print("\tReceived KeyboardInterrupt. Exiting script")
            wolk.device.disconnect()
            return
```

Obrada slike – prepoznavanje registarske oznake automobila

Kao što je pomenuto u delu gde se opisuje biblioteka za obradu slike, glavna funkcija za obradu slike se naziva *process_image*. Na početku te funkcije se vrši učitavanje originalne slike odnosno slike sa koje je potrebno učitati registarski broj kao i promena veličine te slike. Originalna slika sa promenjenom veličinom je prikazana na sledećoj slici.

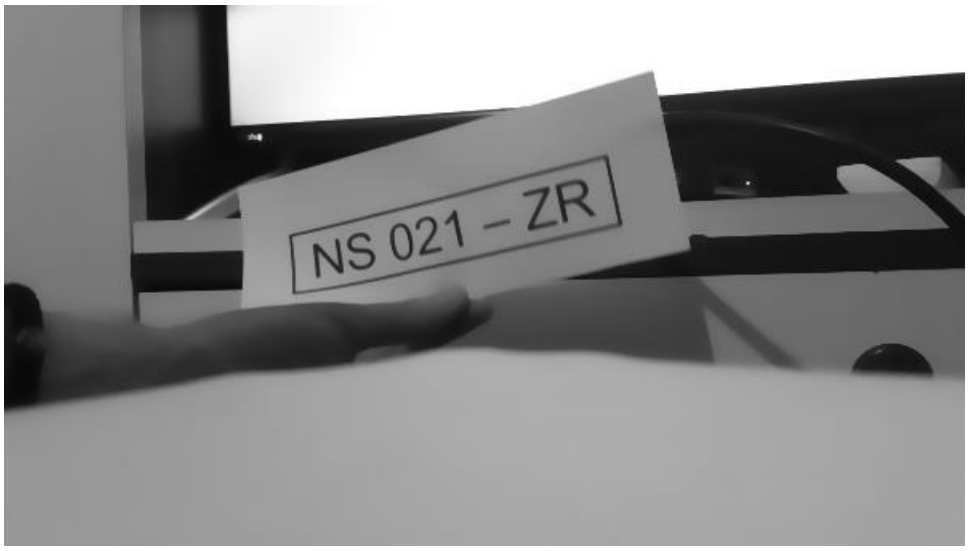


Slika 7. Originalna slika

Nakon toga, originalna slika se prebacuje u sliku sive boje. To se radi uz pomoć funkcije *cvtColor*. Na sledećoj slici je prikazana originalna slika u sivoj varijanti.

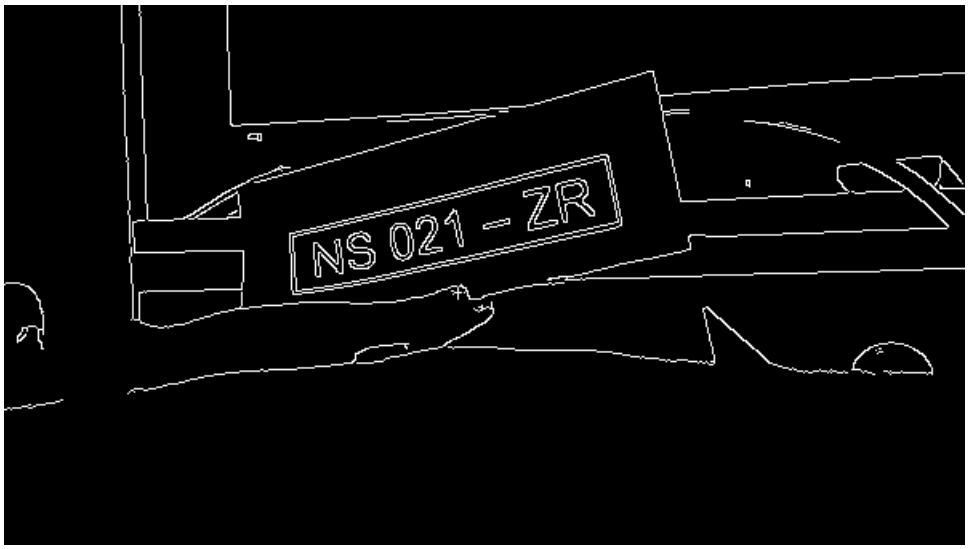
Slika 8. Originalna slika u sivoj varijanti

Sledeća funkcija koja se primenjuje jeste funkcija *bilateralFilter*. Ova funkcija smanjuje šum koji postoji i slika postaje sve vise jasnija. Na sledećoj slici će biti prikazana slika koja se dobija nakon primene ove funkcije.



Slika 9. Slika nakon primene funkcije bilateralFilter

Na ovakvu sliku se dalje primenjuje funkcija *Canny*, čiji je zadatak da označi sve moguće konture na gornjoj slici. Slika koja se dobija nakon primene ove funkcije je prikazana na sledećoj slici.



Slika 10. Slika nakon primene funkcije Canny

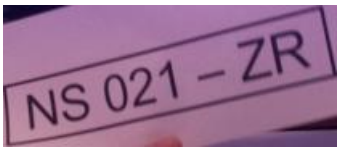
Sledeći korak predstavlja primena funkcije *findCountours*. Ova funkcija obeležava na slici sve konture i što je još važnije, otklanja suvišne detalje na datim konturama. Sledeća slika predstavlja sliku nakon primene ove funkcije.

Slika 11. Slika nakon primene funkcije findCountours

Sledeći korak predstavlja korišćenje funkcije *sorted*. Ova funkcija zapravo ostavlja samo konture čija je površina veća od 30. Slika koja je dobijena nakon primene ove funkcije je prikazana na sledećoj slici.

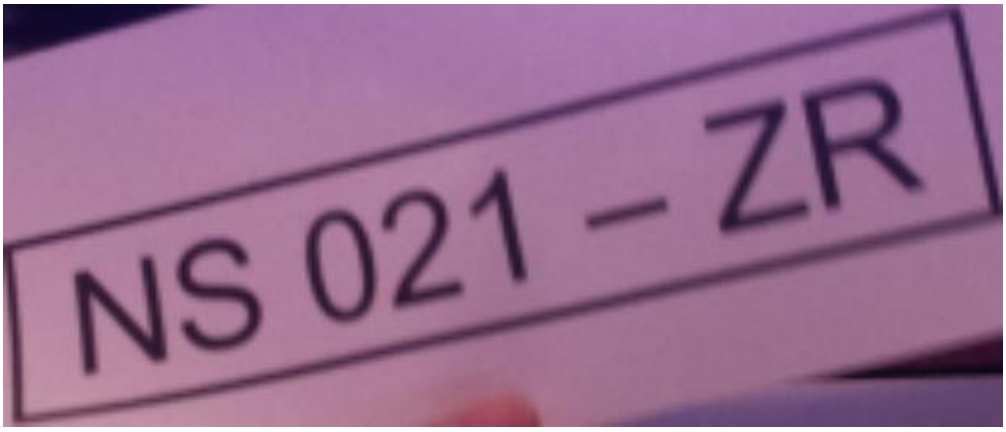
Slika 12. Slika nakon primene funkcije sorted

Naredni korak u obradi slike predstavlja pronalaženje kontura koje imaju četiri strane. Zatim se proverava da li je obim date konture jednak broju četiri. Ukoliko jeste, deo slike na kojem se nalazi takva kontura se iseca jer je velika verovatnoća da je to zapravo okvir u kome se nalazi registarski broj, Sledeća slika predstavlja isečen deo originalne slike.



Slika 13. Isečeni deo originalne slike

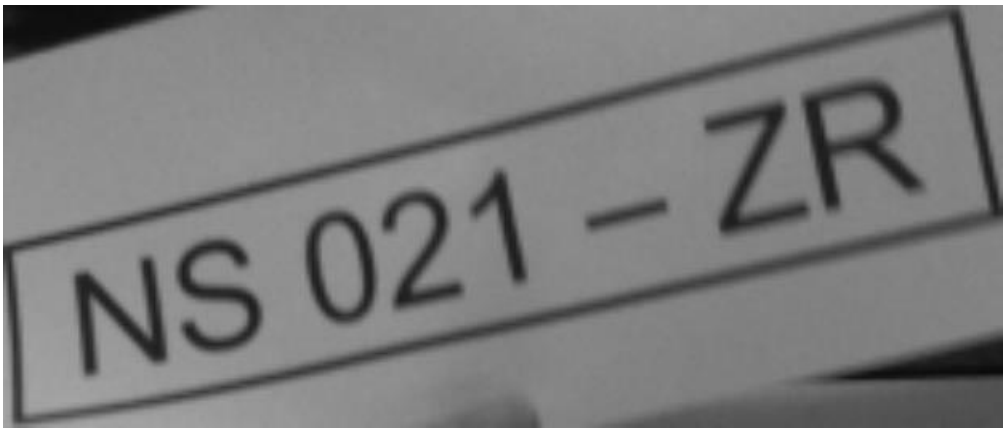
Zatim se ovakva slika uvećava tri puta po svakoj osi i dobija se sledeća slika.



Slika 14. Uvećana prethodna slika tri puta po svakoj osi

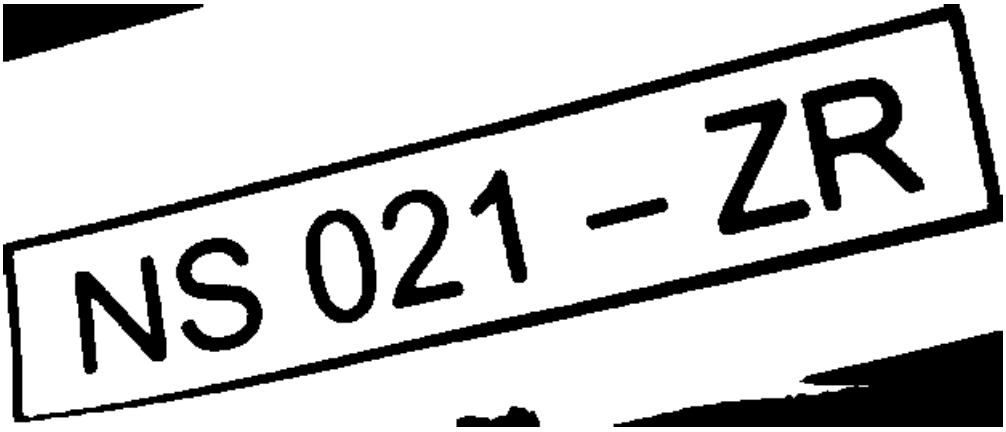
Zatim se na ovako uvećanu sliku primenjuju tri funkcije: *get_grayscale, thresholding* i *opening.*

Funkcija *get_grayscale* prebacuje sliku u sivu sliku. Ovako transformisana slika je prikazana na sledećoj slici.



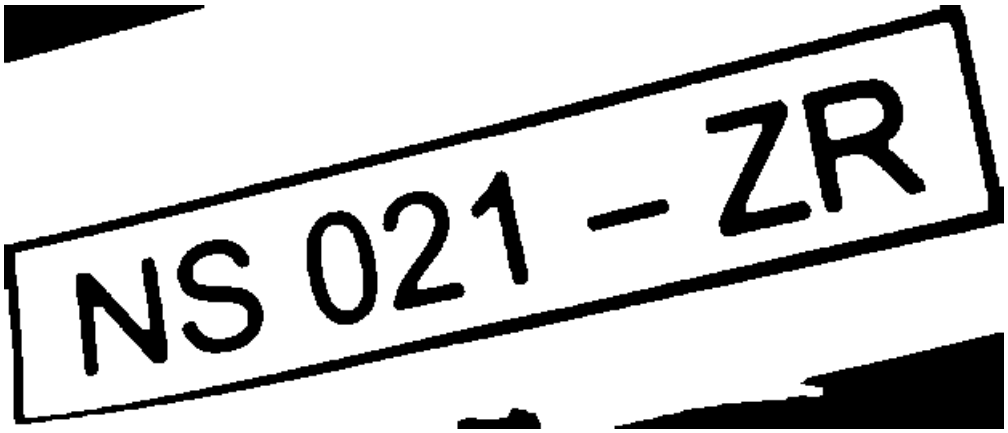
Slika 15. Slika dobijena nakon korišćenja funkcije get_greyscale

Funkcija *thresholding* postavlja svaki piksel na slici na vrednost svog praga ili na nulu. Ovim dobijamo da okvir i slova na slici budu što je moguće vise tamniji a pozadina što vise svetlija. Takva slika je prikazana na sledećoj slici.



Slika 16. Slika nakon primene funkcije thresholding

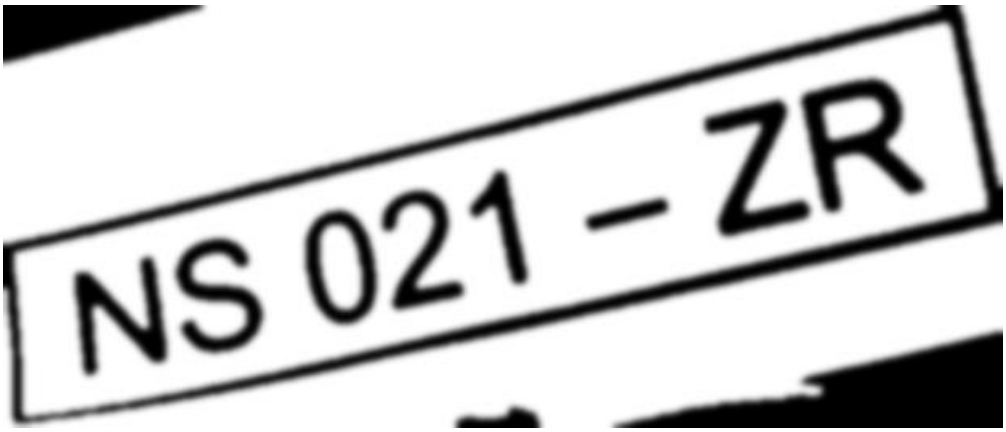
Funkcija *opening* zapravo sve detalje koje se nalaze u prvom planu stapa sa pozadinom.



Slika 17. Slika nakon primene funkcije opening

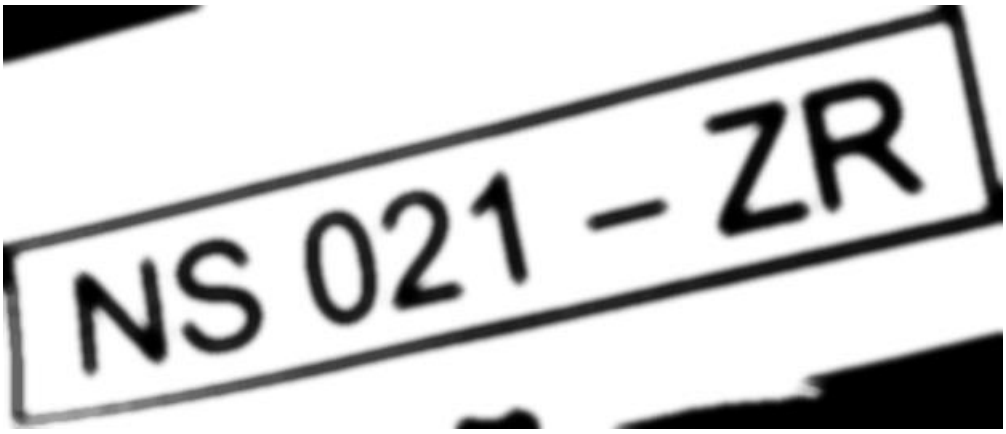
Nakon ove tri funkcije, na red dolaze još dve funkcije koje služe za filtriranje slike kako bi što lakše bilo iz nje izvući određeni tekst.

Funkcija *GaussianBlur* zapravo predstavlja funkciju koja na sliku primenjuje Gausov filter odnosno funkcija koja vrši Gausovo zamućenje slike. Nakon primene ove funkcije, dobija se slika koja je prikazana u nastavku.



Slika 18. Slika nakon primene funkcije GaussianBlur

Funkcija *medianBlur* predstavlja funkciju srednjeg zamućenja. Slika koja se dobija nakon primene ove funkcije je prikazana u nastavku.



Slika 19. Slika nakon primene funkcije medianBlur

Zamućenje koje donose prethodne dve funkcije je potrebno kako bi se što je moguće više eliminisali detalji oko slova koji nisu bitni.

Na ovakvu sliku se primenjuje *pytessaract* algoritam koji izvlači dati registarski broj sa same slike.

Zaključak

Prilikom izrade projekta urađeno je sve što je bilo propisano projektnim zadatkom. Omogućena je detekcija automobila kada se pojavi ispred vrata garaže, nakon toga prepoznavanje registarske oznake i na osnovu toga otvaranje vrata garaže ukoliko je tablica iz unaprijed definisane baze podataka.

Na kraju je detaljno testiran system odnosno sve što je bilo zadato projektnim zadatkom. Senzor ispravno detektuje prisustvo, znakovi sa registarske oznake se uspješno prepoznaju i servo motor ispravno reaguje odnosno simulira otvaranje i zatvaranje vrata garaže.

Moguće poboljšanje projekta bi bilo detekcija automobila koji napušta garažu. To bi se moglo jednostavno postići dodavanjem još jednog IR senzora koji bi se postavio sa unutrašnje strane vrata garaže. Kada ovaj senzor detektuje da automobil želi izaći iz garaže, potrebno je ponovo napraviti fotografiju njegove registarske oznake. To bi bilo moguće uraditi upotrebom još jedne kamere ili jednostavno postavljanjem postojeće kamere na servo motor tako da je moguće njeno zakretanje i snimanje i sa unutrašnje i sa spoljašnje strane vrata garaže. Na taj način bi se uz dodavanje još jednog IR senzora i još jednog jednostavnog servo motora relativno male snage mogao dobiti proizvod koji bi bio u potpunosti funkcionalan za realizaciju pametne garaže sa velikim brojem parking mjesta.

Zaključak je da je projekat uspješno realizovan i da je implementirano i testirano sve što je bilo potrebno.

Literatura

[1] https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/, jun 2022.

[2] http://wiki.sunfounder.cc/index.php?title=Raspberry_Pi_Camera_Module, jun 2022.

[3] https://circuitdigest.com/microcontroller-projects/interfacing-ir-sensor-module-with-arduino, jun 2022.

[4] Tehnička dokumentacija za SG90 Micro Servo, jun 2022.

[5] https://www.section.io/engineering-education/license-plate-detection-and-recognition-using-opencv-and-pytesseract/, jul 2022.

[6] https://tutorials-raspberrypi.com/raspberry-pi-text-recognition-ocr/, jul 2022.

[7] https://stackoverflow.com/questions/44619077/pytesseract-ocr-multiple-config-options, jul 2022.