PAMETNI SISTEM ZA NADGLEDANJE PARKINGA

Seminarski rad sa kursa Upravljački algoritmi u realnom vremenu

STUDENTI: Nedeljko Tešanović
Katarina Ranisavljević
Danijela Poljašević

MENTOR: Boris Jakovljević

Novi Sad
Fakultet Tehničkih nauka
Školska 2019/2020.

SADRŽAJ

Specifikacija problema

Problem automatizacije parkinga, tj. konceptualizacija pametnog sistema za nadgledanje parkinga. Sistem je dužan da u realnom vremenu obezbedi prikazivanje, popunjavanje i pražnjenje parkinga, kao i da evidentira parkirana vozila.

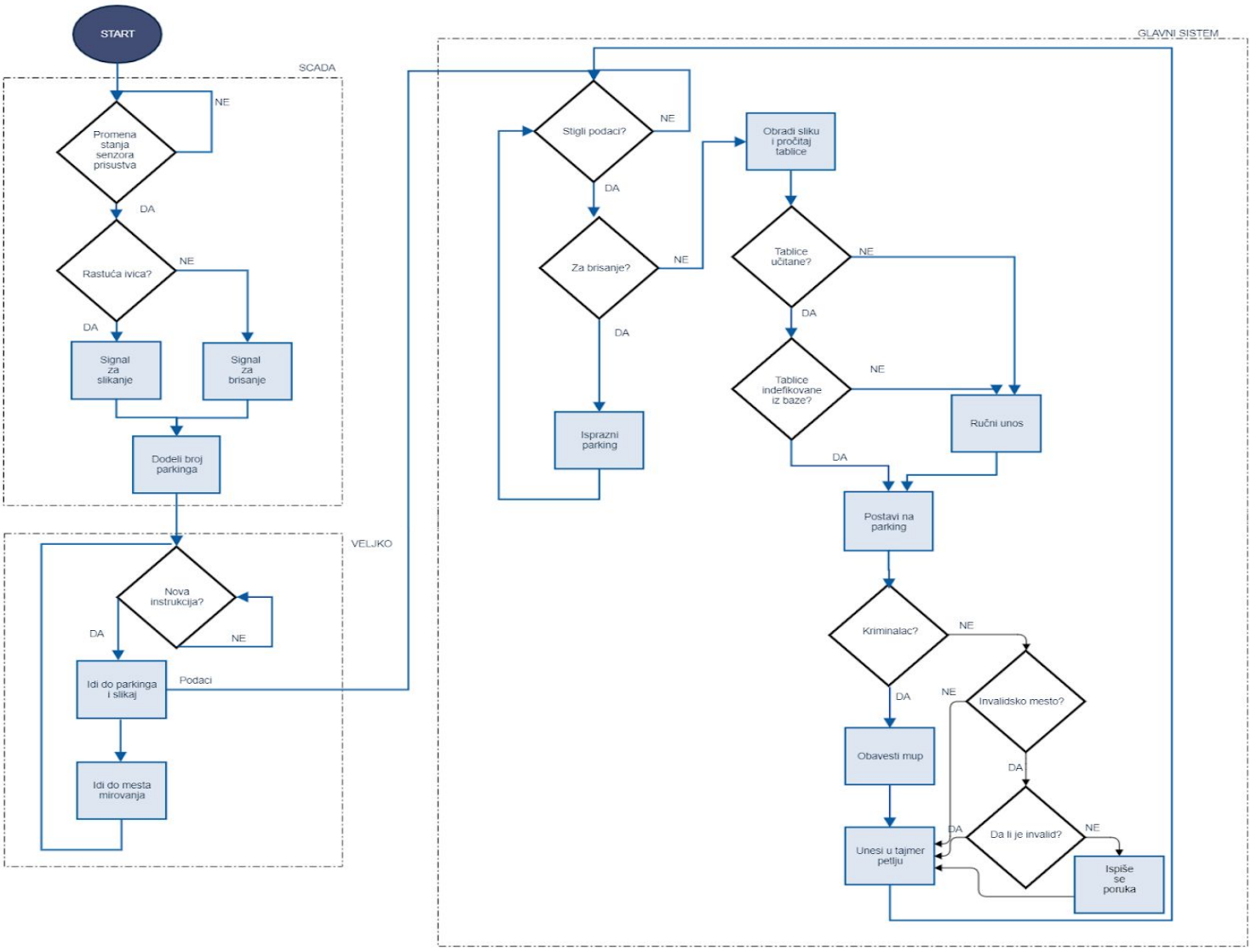
Idejno rešenje

Ideja projekta je da se automatizuje parking servis. Robot sa kamerom će zameniti radnika koji vrši evidenciju parkiranih vozila, sa predefinisanom rutom kretanja kroz parking zonu. Kada robot sa senzora prisustva, koji se nalazi na svakom parking mestu, dobije signal da se novi automobil parkirao, on će otići na to parking mesto i slikati tablice vozila. Tablica sa slike će zatim biti pročitana pomoću neuronske mreže i evidentirana u bazi podataka.
Ideja je da senzori parkinga budu realizovani uz pomoć SCADA sistema, kretanje robota uz pomoć MicroPython-a, a ceo proces obrade slike i provere baze podataka u National Instruments LabVIEW-u.

Opis sistema parkinga

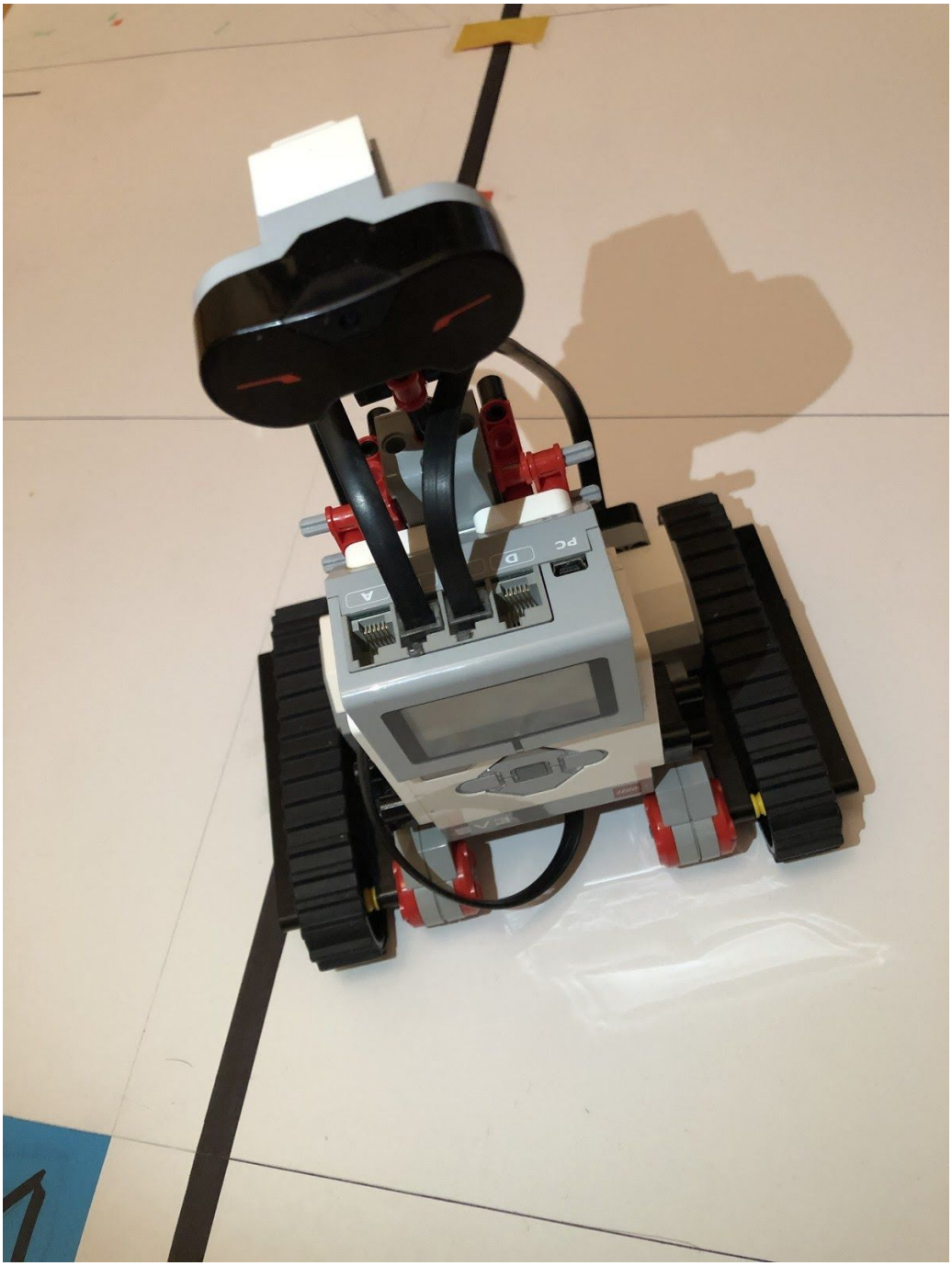
Kontrolna mašina PSNP-a ima sledeće delove:

- ➔ Komunikacijski kanal sa Python shell-om
- ➔ Komunikacijski kanal sa vanjskim SCADA sistemom i njegovim bazama podataka
- ➔ Komunikacijski kanal sa Ministarstvom Unutrašnjih Poslova Republike Srbije
- ➔ Glavnu LabVIEW aplikaciju

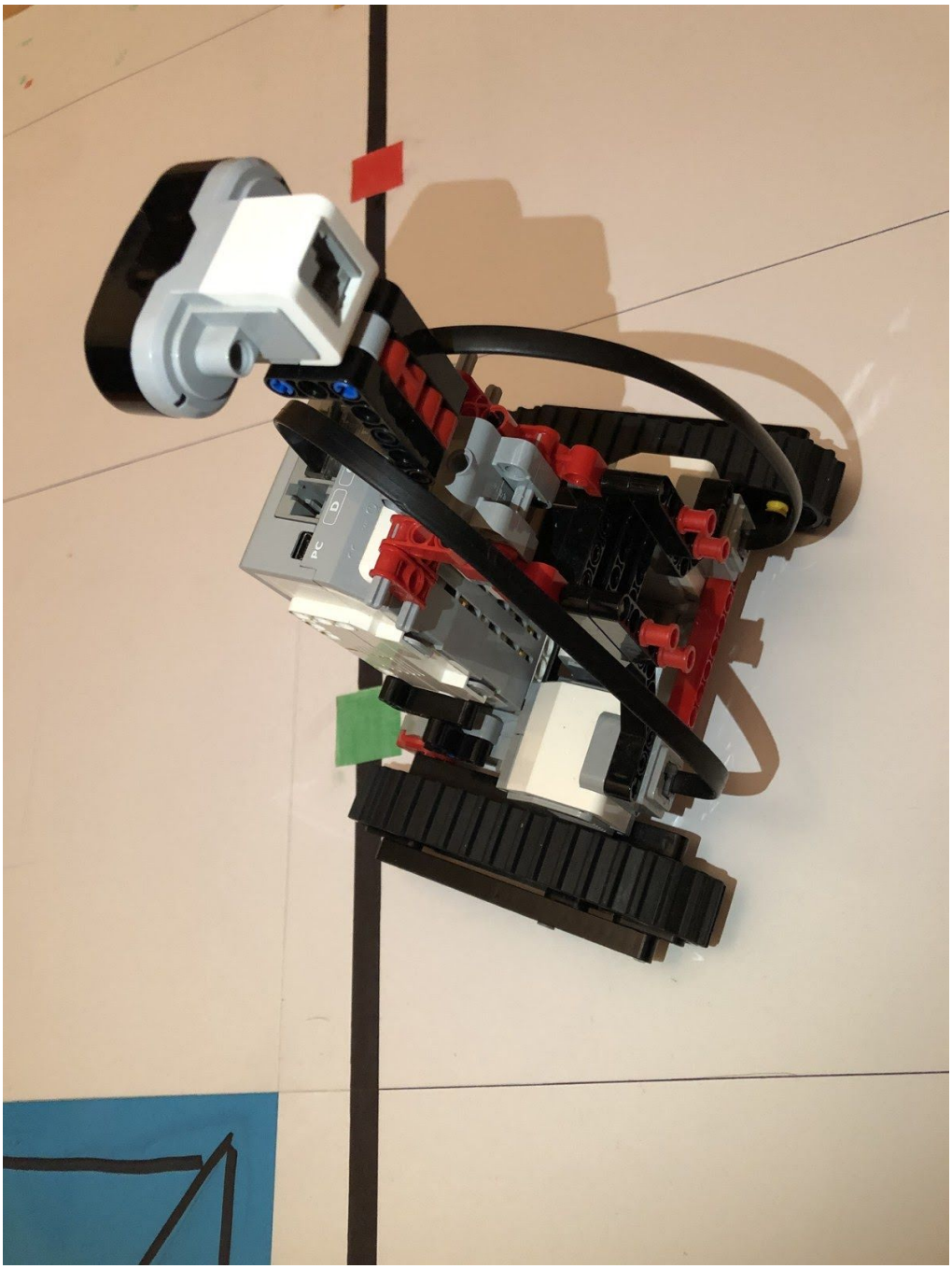


Block diagram

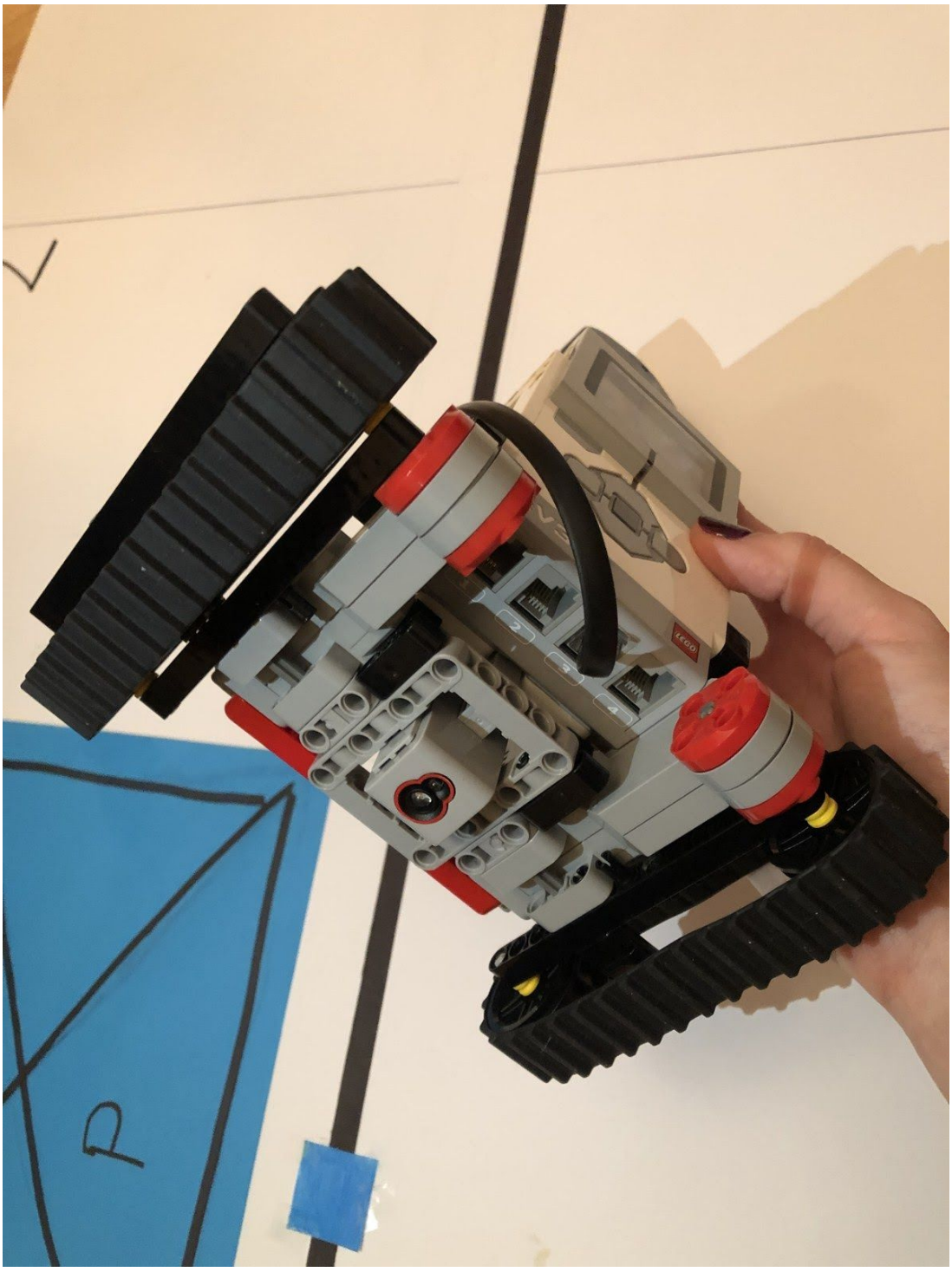
Izgled robota



Front view



Back view



Bottom view - Color sensor

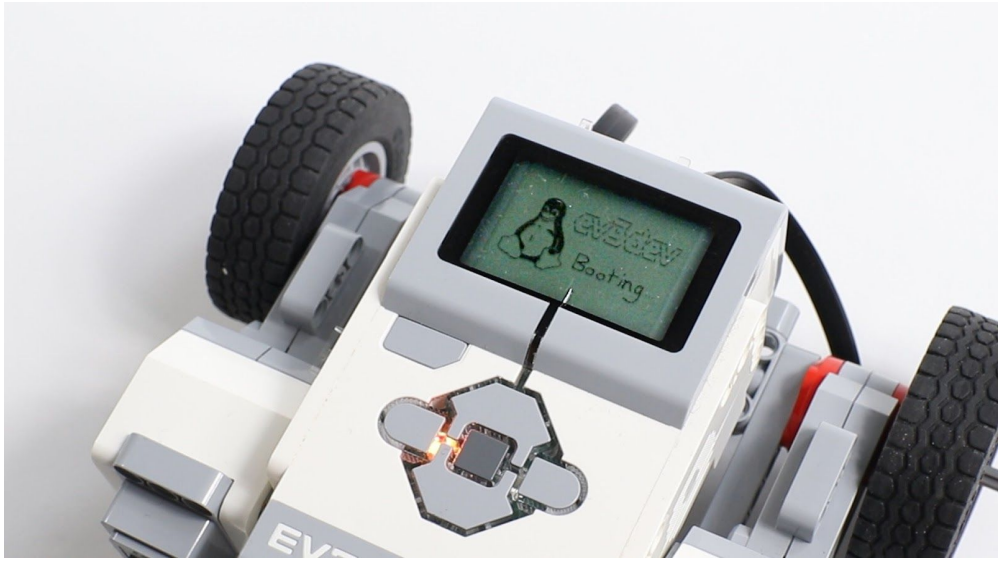
Parking design

Šta je Lego Mindstorms?

Lego Mindstorms je linija programabilnih robota, koji se baziraju na Lego kockama. Svaka verzija sistema uključuje inteligentnu *ciglu*, tj. kompjuter koji kontroliše sistem, set modularnih sensora i motora, kao i Lego delove iz linije Technic, za pravljenje mehaničkih sistema. Originalni Lego Mindstorms Robotics Invention System je rođen iz kolaboracije MIT-ja i Lego grupe.
Trenutno, postoje 4 generacije Lego robota, i svaka može da se programira iz zvaničnog Lego softvera, mada su se njihovim razvijanjem pojavile i biblioteke za njihovo programiranje u raznim jezicima: Pyhton, LabView, Matlab (Simulink), ruby, i mnogi drugi.

Lego Mindstorms set

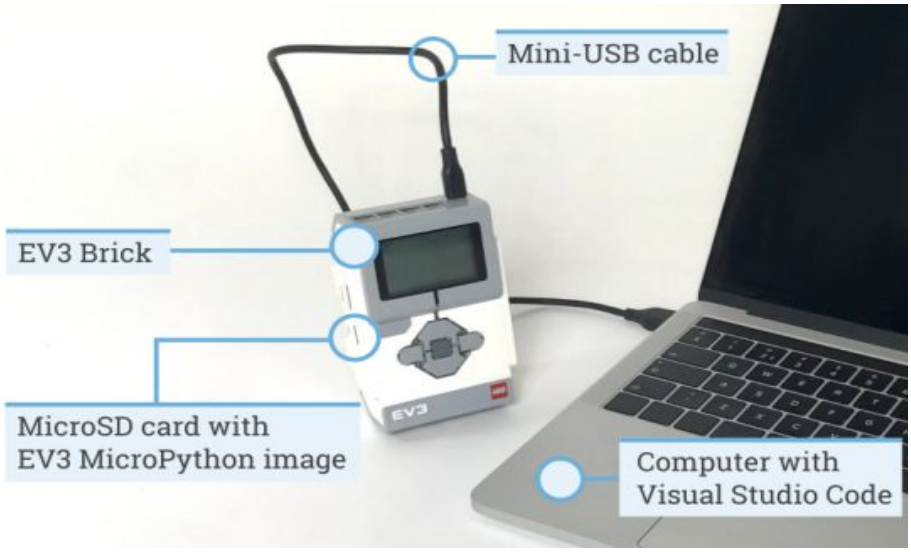
MicroPython



Kao što je već rečeno, za programiranje Lego robota su se vremenom pojavile biblioteke u nekim dosta popularnim jezicima. Za ovaj projekat, odabrano je proširenje jezika Python, i u narednom poglavlju biće detaljno objašnjeno kako je robot podešen za njega. Kompletna dokumentacija može se naći *ovde*.

Šta je potrebno da se sistem prebaci na robota?

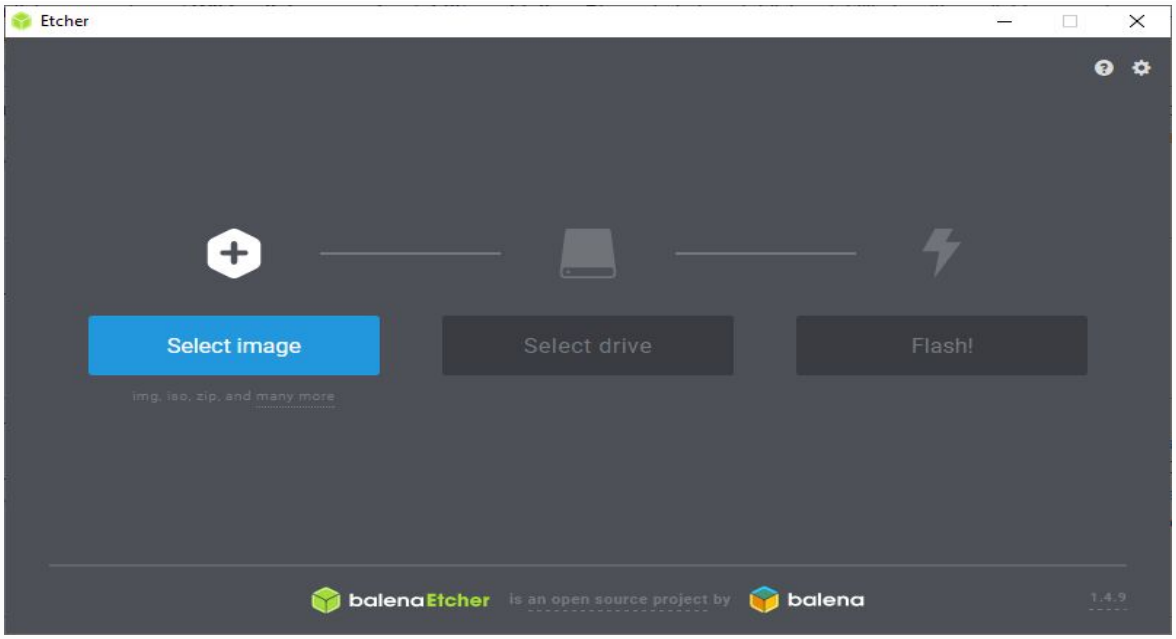
1. SD kartica (16GB)
2. Čitac SD kartice
3. Softver koji omogućava korišćenje .iso, .img i zip-ovane fajlove
4. Visual Studio Code



Pokretanje MicroPython-a i Python-a na Lego Mindstorms EV3

Koraci:
1.**Skidanje EV3 MicroPython slike sa LEGO education sajta**, koji ne sme da se raspakuje.

2. **Prebacivanje slike na Micro SD karticu**. Za pisanje slika možete korisiti Balena Etcher softver. Softver će možda zatražiti administratorsku lozinku, jer joj je potreban nizak nivo pristupa kartici.

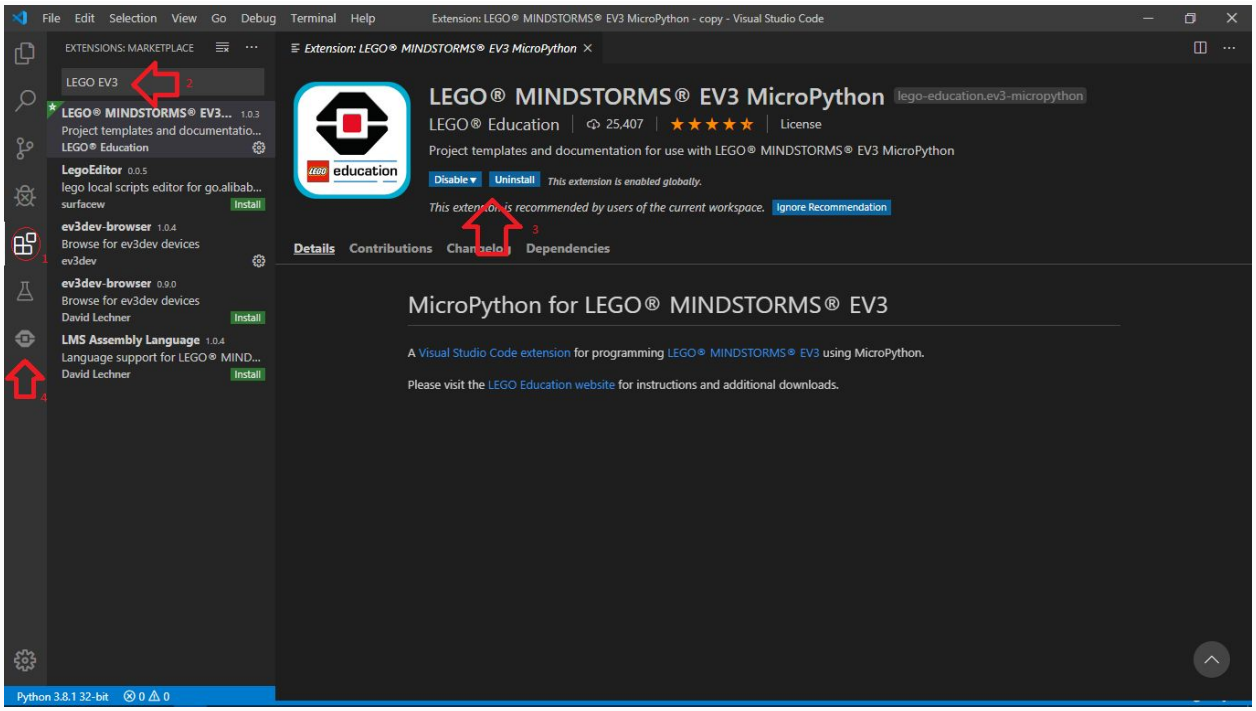


Balena Etcher interface

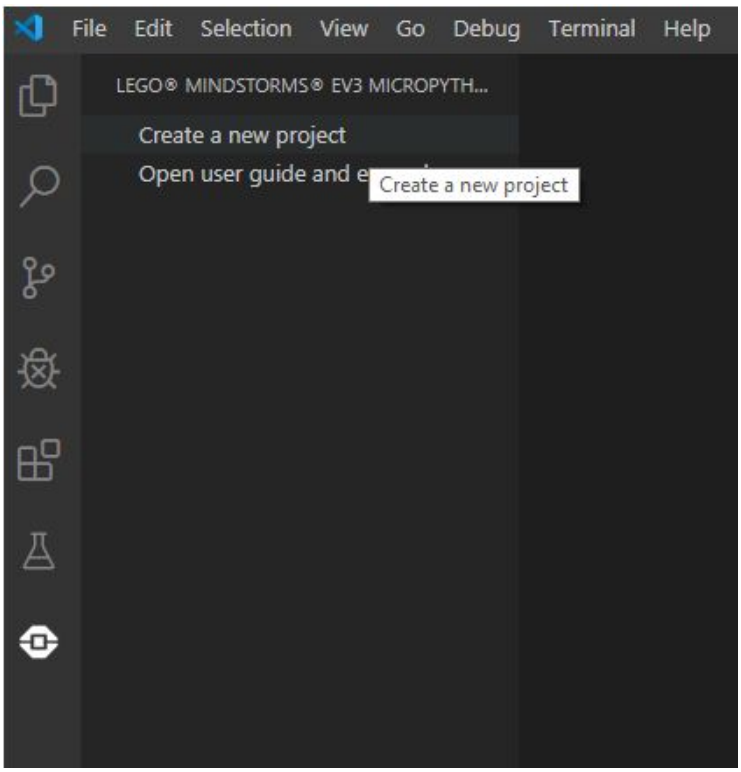
3. **Nakon toga, ubacite SD karticu u robota, i zatim ga uključite.** Na ekranu će se pojaviti dosta opadajućeg teksta, i to može potrajati nekoliko minuta, za to vreme možete preći na sledeći korak.

4. Za vreme izvršavanja trećeg koraka potrebno je **skinuti i instalirati besplatni Visual Studio Code editor.**

5. **U ekstenzijama instalirajte, i potom aktivirajte, LEGO Education EV3 proširenje.** Pri pokretanju Visual Studio Code-a, sa leve strane će se pojaviti ikonica.



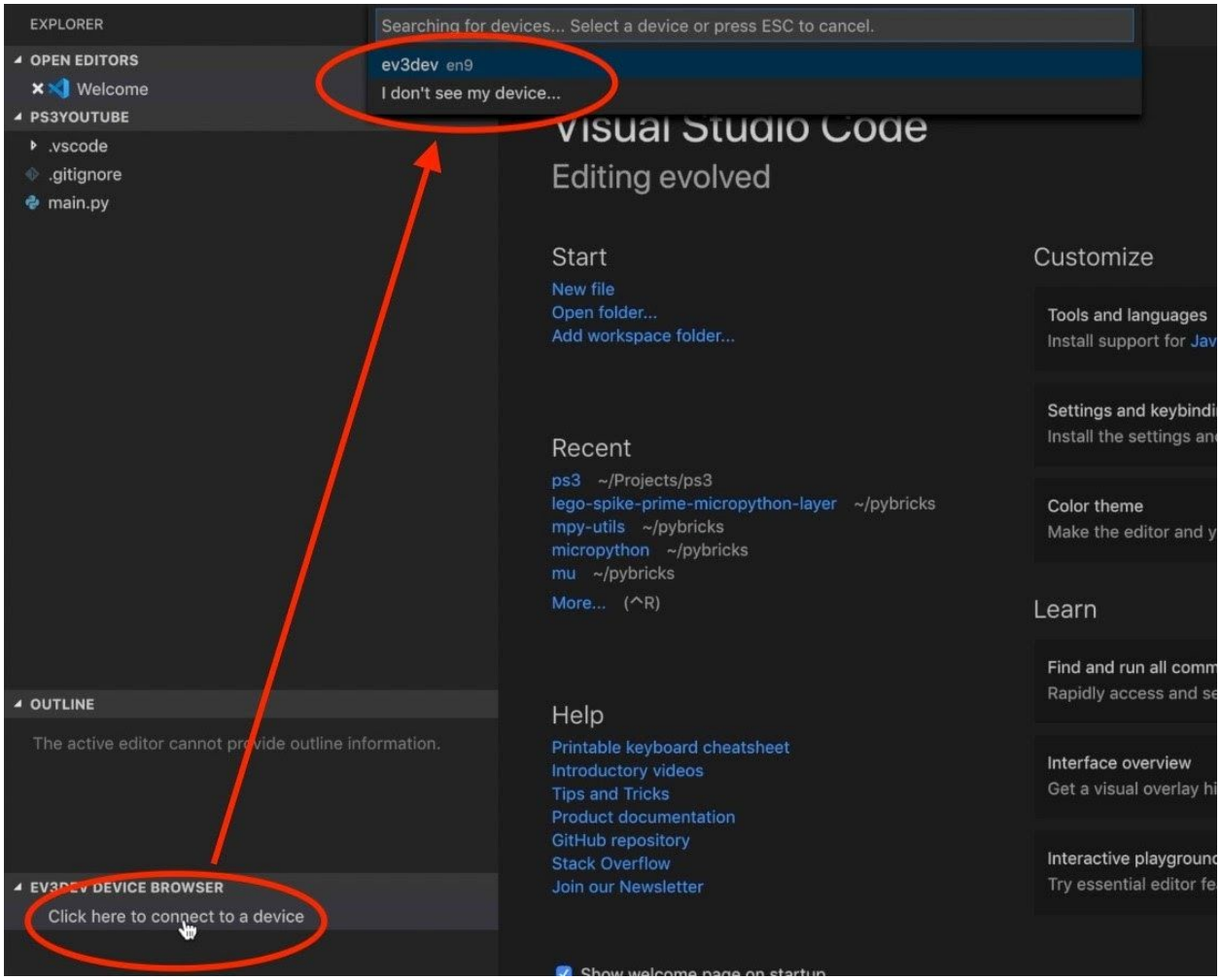
6. **Povežite EV3 ciglu sa računarom preko USB-a ili bluetooth-a.** Sada je najverovatnije završio sa dizanjem sistema.



Kada se napravi projekat projekat, ekstenzija automatski učitava sve pakete koji su dostupni, i koji se koriste za upravljanje robotom.

main.py fajl, koji je takođe automatski generisan, jeste ono što se pokreće na robotu. Možete ubacivati pomoćne fajlove,samo ih je potrebno importovati u *main* programu.

7. **Povezivanje sa robotom.** Nakon ovih 6 koraka, ono što je preostalo jeste povezivanje sa robotom. U explorer meniju sa leve strane, dole će se nalaziti deo “ev3dev device browser”. Klikom na to će se pojaviti opcija “Click here to connect…”. Odaberite vašeg EV3 robota.



Kada napišete svoj program, klikom na F5 vaš program će biti prebačen na robota i pokrenut.

Detaljniji opis instalacije možete naći na *zvaničnoj stranici*.

Šta je sledeće? Sad, kad ste uspešno konfigurisali robota, možemo početi sa delom projekta koji se tiče kretanje robota.

Praćenje linije

Robot da bi se kretao po parkingu, potrebno je da prati predefinisanu putanju, tj. liniju ispod sebe. Ovo je neophodno iz više razloga, jedan jeste da ne bi skrenuo sa puta usled nesavršenosti realizacije rotiranja, a drugi je da bi se mogao isti kod koristiti i za slobodan oblik parkinga. Ovo će biti realizovano sa jednim senzorom za boje, u režimu merenja refleksije. Bolji način jeste realizacija sa dva senzora, ali se ovako može realizovati sa samo jednim Mindstorms setom.

Način rada je sledeći: kada se robot nalazi polovično na liniji (bitan je kontrast slobodne površine i linije), izmerena refleksija je, u teoriji, 50%. Najbolje je da ovo testirate i vidite koja je vaša vrednost, jer će broj varirati od vrste podloge. U našem slučaju ta vrednost je 47, i ona predstavlja referencu (*set_point*) koju robot prati. U toku kretanja, može doći do malog poremećaja, sa time će se promeniti i upravljanje koje će robota vratiti bliže referentnoj vrednosti.

Ceo kod je dostupan *ovde*. Kroz neke delove proćićemo u sledećem tekstu.

Za početak, potrebno je Lego cigli definisati njene ulaze (senzore) i izlaze (motore) koje će biti upotrebljene, da bi na osnovu toga slali signale. To je najbolje uraditi na samom početku koda.

```
#defining all sensors and motors that will be used
right_motor = Motor(Port.C)
left_motor = Motor(Port.B)
color_sensor = ColorSensor(Port.S3)
```

Dostupni portovi za izlaz su A, B, C i D, na njih se mogu zakačiti samo kontrolni delovi, koji će uticati na kretanje robota. Senzori, tj. ulazi, imaju svoje portove: S1, S2, S3 i S4. U kodu, pristupa im se preko klase Port, iz paketa `pybricks.parameters`

Sva imena promenljivih, kao i odabrani portovi su proizvoljni, i možete ih sami birati. Kao što se iz koda vidi, ovde su odabrani portovi za dva točka B i C, i S3 za senzor boja.

Sledeće što treba uraditi jeste inicijalizacija šasije koju robot koristi za kretanje:

```
#Initialization of a driving base
robot = DriveBase(right_motor, left_motor, 35, 144)
```

Ovde inicijalizujemo objekat klase DriveBase, koji očekuje redom sledeće parametre: port desnog motora, port levog motora, prečnik točka, razmak točkova na osovini. U zavisnosti od izgleda/dimenzija osovine, robot će sam da aktivira točkove za kretanje (sinhronizovano), i sam će praviti logiku za kretanje. Dimenzije koje se prosleđuju su u milimetrima.

Iz klase DriveBase dostupne su sledeće metode:

`drive`(*brzina, ugao*) - Početak vožnje zadatom brzinom, sa zadatim uglom, računanje se vrši u odnosu na centar osovine. Brizina se zadaja u mm/s, a ugao deg/s.

`drive_time`(*brzina, ugao, vreme*) - Robot vozi određeno vreme zadatom brzinom, sa zadatim uglom. Vreme je zadato u milisekundama.

`stop`(*stop_type=Stop.COAST*) - Način zaustavljanja. Ceo spisak mogućih parametara možete videti u dokumentaciji. Za potrebe projekta, kao što ćete kasnije videti, korišćeno je `Stop.BRAKE`.

Pre nego što pređemo na sam kod, ukratko ćemo objasniti funkcije iz *lib.py*, fajla sa potrebnim pomoćnim funkcijama. Najbolje je da pomoćne funkcije odvojite u zaseban fajl, koji ćete importovati u glavnom programu, radi lakšeg razumevanja i urednosti. Neke funkcije postoje već u drugim paketima, ali smo radi jednostavnije realizacije i usled nemogućnosti da robot pristupi internetu i nabavi potrebne pakete napravili svoje. Potrebno je napomenuti da ***robot ima svoju kopiju koda*** koju pokreće na svom operativnom sistemu, i nije dovoljno imati paket na uređaju na kom pišete kod.

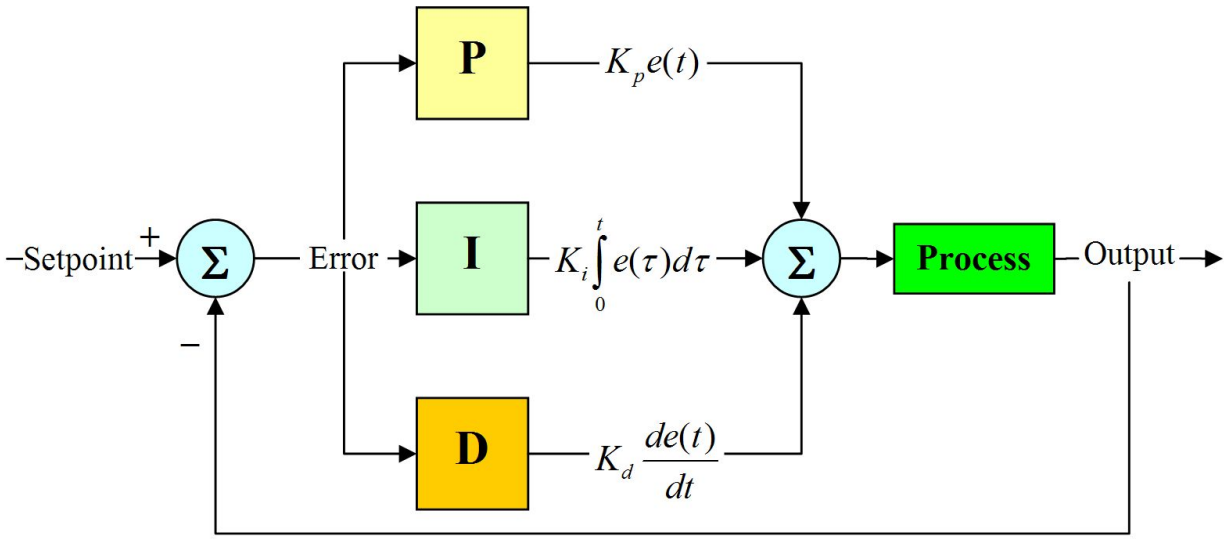
```
#Calculating the sum of all elements of the given array
def sum(elements: list):
    sum = 0
    for element in elements:
        sum += element
    return sum
```

Ovo parče koda računa sumu elemenata prosleđene liste. Ovo će biti potrebno kada budemo računali ukupno odstupanje od zadate putanje. Za to će takođe biti potrebna i `diff` funkcija, koja računa razliku između dva susedna elementa liste.

```
#Calculates differences between adjacent elements of the given array
def diff(arr: list):
    retArr = []
    for i in range(len(arr) - 1):
        retArr.append(arr[i + 1] - arr[i])
    return retArr
```

Funkcija koja sledi zahteva površno znanje iz teorije sistema automatskog upravljanja, ali to nije glavni fokus projekta, pa će biti objašnjena samo osnovna ideja.

Dok robot prati liniju dolazi do malog odstupanja od nje, i zbog toga je potrebna regulacija da svaki put odredi novu vrednost upravljanja. U našem slučaju, upravljanje je ugao pod kojim robot treba da se kreće napred. Pošto model sistema u ovom slučaju nije poznat, ne možemo realizovati pravi PID regulator, ali se možemo poslužiti osnovnim konceptom. Integral greške, u našem slučaju, biće vrednost greške u nekoliko prethodnih iteracija. Pošto ne želimo da nam sve istorijske vrednosti utiču na upravljanje, samo prethodne 4 vrednosti biće uzete u obzir. Diferencijalno dejstvo regulatora utiče na grešku u budućnosti, što ćemo mi realizovati preko funkcije `diff` koju smo napisali. Na taj način, posmatraćemo kojom brzinom se greška menjala, tj. koliko monotono, u prošlim iteracijama. Proporcionalno dejstvo ćemo samo da množimo sa prosleđenim koeficijentom. Iako ovo nije pravi PID regulator, poslužiće za realizaciju kada ne poznajemo model sistema.



PID regulation diagram

```
def PID(values: list, kp: float, ki: float, kd: float):
    integral = sum(values[-4:])
    derivative = sum(diff(values[-2:]))
    return kp*values[-1] + ki*integral + kd*derivative
```

Sada se vraćamo na main.py. Ovde se nalaze dve glavne petlje programa. Pošto je najčešća akcija robota praćenje linije, to ćemo napraviti kao zasebnu funkciju, kojoj ćemo prosleđivati parametre koji se razlikuju u zavisnosti od završnog cilja kretanja.

Senzor za boje, koji smo na početku inicijalizovali, može da radi u nekoliko različitih režima rada. Mi ćemo koristiti mogućnosti merenja refleksije i prepoznavanja boja. Prepoznavanje boja će biti korišćeno kao signal zaustavljanja. Kao što je već rečeno, logika praćenja linije je usmerena na kretanje po ivici linije. Kada robot počne da odstupa od linije, vrednost refleksije će se menjati, i sa time ćemo menjati rotaciju. Ukoliko je skrenuo na slobodnu površinu parkinga (što je u našem slučaju belo), refleksija će se povećati u odnosu na referentnu vrednost, a PID će robotu dati uravljanje u suprotnu stranu, da vi vratio robota na liniju i približio se referenci.

Promenljiva `driving_side` nam govori sa koje strane linije se robot nalazi, pošto će ovo zavisiti od smera njegovor kretanja. Kada se krece ka severu nalaziće se sa leve strane, a sa desne kada se kreće ka jugu (u odnosu na parking).

Kada robot dostigne zadatu lokaciju, označenu bojom, naglo se zaustavlja, a povratna vrednost je koliko je na kraju robot bio rotiran u odnosu na nulti položaj - 0 stepeni rotacije. Ovo nam je neophodno zbog kasnijeg rotiranja, da se ne bi rotirao više ili manje, nego tačno zahtevani ugao.

```
def drive_straight(driving_side: int, color: Color):
    error_history = [] # needed for PID
    steering_history = [] #needed for deviation
    set_point = 47 # number got in experiment

    # drive straight until you reach the given point
    while color_sensor.color() != color :

        #appending the error so we can get the next rotation angle
        error = set_point - color_sensor.reflection()
        error_history.append( error )
        steering = lib.PID(error_history, 0.1, 0, 0.1) # only PD
        steering_history.append(steering)
        robot.drive(100, driving_side*steering)

        wait(2) #Wait so we don’t change the steering every 0.xx ms

    robot.stop(stop_type = Stop.BRAKE)
    return lib.sum( lib.diff(steering_history))
```

Kada smo objasnili sve potrebne funkcije, možemo preći na glavnu petlju našeg programa. Da ne bi morali često da pokrećemo isti program, ideja je da se ista logika ponavlja sve dok se na robotu ne pritisne neko od dugmića- prinudni stop signal. Funkcija `any` vraća True, ukoliko je neka vrednost iz parametara jednaka True.

```
while not any( brick.buttons() ):
```

U realnoj situaciji, robot će sa induktivnih senzora na parking mestu dobijati informaciju o pristiglom automobilu, ali za potrebe projekta, senzor ce biti emuliran sa korisničkog unosa. Zbog štednje energije, robot se neće kretati dok ne dobije informaciju da je parking mesto zauzeto.

```
    print("Enter parking number: ")
    n = int( input() ) # waiting for a signal that there is a new car
```

Kada dobijemo informaciju, moramo da odredimo logiku kretanja. Parking mesta su označena brojevima 1-6, kao što je prikazano na slici u početku dokumenta. Za svako mesto boja zaustavljanja se menja, kao i smer rotacije. Kako je parking simetričan, ukoliko nam numerisanje mesta počinje sa jedne, a završava sa druge, lako može implementirati logika. Ovo ćemo dobiti od funkcije `getLogic` iz fajla lib.py (Napomena, u lib.py moraju biti importovani svi paketi koji su u upotrrebi).

lib.py:

```
color = [Color.GREEN, Color.RED, Color.BLUE]
def getLogic(parking_number: int):
    rotation = 1 if parking_number < 4 else -1 # rotation direction
    retColor = color[parking_number%3] # stopping signal
    return (rotation, retColor)
```

Nastavak main funkcije:

```
    rotation_side, color = lib.getLogic(n)

    offset = drive_straight(1, color)
    #depending on the side, offset will affect the rotation differently
    rotation_angle = (90 + offset) if n > 3 else (90 - offset)
    robot.drive_time(0, rotation_side*rotation_angle , 1000)
```

Pošto smo dobili informacije o rotiranju možemo izvršiti kretanje do određenog parking mesta. Kada dođemo do određene dubine parkinga - označeno određenom bojom, potrebno je da se rotiramo ka automobilu, i sačekamo obradu tablica. Ovaj deo će u kasnije biti posebno obrađen. Kao što je već objašnjeno, robot se neće okrenuti tačno 90 stepeni, jer se u toku svog kretanja potencijalno malo zaokrenuo. U tom slučaju, rotaciju računamo u odnosu na to odstupanje.

Demonstrativno: ako je poslednje upravljanje bilo u desno, robot je za 2 stepena okrenut u desno, i umesto 90, okrenuće se samo 88 stepeni u desno, tj. 92 u levo.

Sada, čekamo povratnu informaciju sistema verifikacije da li je parking plaćen. Takođe, mi ćemo emulirati korisničkim unosom signalizaciju. Robot će u zavisnosti od toga da odreaguje nekim zvučnim signalom. Sistem verifikacije će po registracijama tablica poslati kaznu na adresu na koju je automobil registrovan.

```
print("Verify the payment ")
paid = int(input())

if not paid:
   brick.sound.file(SoundFile.UH_OH)
else:
   brick.sound.file(SoundFile.CHEERING)
```

Nakon toga, robot se vraća na staru poziciju, i čeka sledeću akciju. Tu je kraj naše petlje, i ceo ovaj proces se ponovo pokreće. Pri povratku, kretaće se sa druge strane linije, pa fukciji šaljemo -1 kao oznaku smera.

```
# turn again to face the resting spot
robot.drive_time(0, rotation_side*90, 1000)

rideback = drive_straight(-1, Color.YELLOW)
robot.drive_time(0, 90, 2000) #90 degrees for 2 seconds = 180
```

Pokretanje

Kada završite ceo kod, i prebacite ga na robota, korak koji preostaje jeste pokretanje. Pošto koristimo korisnički unos, ne možemo pokrenuti sa F5, nego moramo koristiti SSH sesiju. Desnim klikom na ime robota koji je povezan, pojaviće vam se opcija za otvaranje SSH terminala.

Kao što smo napomenuli, robot na sebi ima svoj operativni sistem (Linux), i na njemu svoju **kopiju** koda. Zato kod koji sadrži korisnički unos, moramo pokrenuti program direktno sa robota, koristeći *Secure Shell - SSH,* mrežni prokol koji omogućava uspostavljanje sigurnog komunikacijskog kanala između dva računara. Na taj način, sve što upišemo na našem (SSH) terminalu, direktno se izvršava na robotu.

Pokretanje se vrši pomoću komande **brickrun**, naglašavanjem direktorijuma i fajla koji pokrećete. U ovom slučaju, ukoliko ništa niste sami menjali, to će biti sledeća komanda:
robot@ev3dev:~$ brickrun --directory=”/home/robot/*ime_projekta*”
“home/robot/*ime_projekta*/main.py” **-r**

Jako je važno navesti dodatni parametar “**-r**”, koji će omogućiti korisnički unos preko tastature SSH terminala. Bez ovoga to vam neće biti omogućeno.

U narednom delu, biće objašnjeno skeniranje slike, i provera dugovanja.

National Instruments LabVIEW

Aplikacija za Pametni Sistem Nadgledanja Parkinga (PSNP) je realizovana na National Instruments LabVIEW platformi (verzija 2019 SP1).

Standardne biblioteke koje dolaze sa studentskom verzijom instalacije NI LabVIEW-a nisu sadržale sve potrebne alate za izradu ovog sistema i njegove logike, te su korišćeni dodatni moduli i paketi:

Python Integration Toolkit - Standard
Vision Development Module
Vision Development Module - Runtime
NI-IMAQ
NI-IMAQ I/O
NI-IMAQdx

LabVIEW aplikacija ima sledeće ulaze sa SCADA-e :

- ➔ **Putanja do slike tablice**
  Javlja sistemu odakle da ucita sliku tablice spremnu za obradu.
- ➔ **Pozicija na parkingu**
  Javlja sistemu o kom parking mestu je reč.
- ➔ **Signal za upis u PSNP**
  Na rastućoj ivici ovog signala se vrši unos i obrada podataka dobijenih sa spoljašnjih sistema. Koristi se da bi se novoparkirano vozilo postavilo na adekvatnu poziciju korisničkog interfejsa.
- ➔ **Signal za brisanje sa PSNP-a**
  Ukoliko je ovaj ulaz logička jedinica, umesto unosa, mesto se prazni. Koristi se pri odlasku vozila sa parkinga.

Opis rada

Prilikom pokretanja programa, uspostavljaju se veze sa bazama podataka tako što se podese putanje do samih baza podataka. Te putanje se ne mogu menjati tokom rada sistema, te je potrebno prvo sistem zaustaviti na taster STOP u gornjem desnom uglu, izvršiti izmenu, i zatim ponovo pokrenuti program. Na isti način se podešava putanja do upotrebljenih python skripti.

Baze podataka sa kojim se uspostavljaju veze su:

- ➔ Registrovanih vozila, koja sadrži podatke o svim registrovanim vozilima (registracijski broj, ime i prezime vlasnika, kao i tip vozila)
- ➔ MUP-a RS, koja sadrži registracijske brojeve vozila traženih od strane zakona
- ➔ Samog parkinga, ažurirana od strane Telekoma, koja sadrži listu registracijskih brojeva vozila koja su platila parking, kao i trajanje plaćenog parkinga
- ➔ Registrovanih invalida - ove osobe su hendikepirane i imaju rezervisano mesto na parkingu

U momentu kada SCADA pošalje signal PSNP-u, proverava se da li je to rastuća ivica (što znači da se neko upravo parkirao, i to nam daje signal za upis u parking mesto), ili padajuća ivica (što znači da je neko upravo napustio svoje parking mesto i to nam daje signal za brisanje). O kom parking mestu je reč nam takođe javlja SCADA.

Ukoliko je reč o upisu, dobijena slika vozila sa tablicom (koju je dobavio PSR "Veljko" M1A) se obrađuje, tj. priprema za Neuronsku Mrežu (NM) koja će da pročita tekst sa nje. Nakon što se tekst pročita sa slike, registracijski broj se propusti kroz mehanizam pretrage baze Registrovanih vozila (pošto sva vozila moraju da se registruju pre dozvoljene upotrebe, smatra se da će ova baza sadržati većinu, ako ne i sva vozila). Nakon uspešnog pronalaska, izvlače se podaci o vozaču i vozilu i postavljaju na odgovarajuće mesto u korisničkom interfejsu. Status validnosti tiketa - tj. da li je vreme za taj tiket isteklo ili ne, prikazano je crvenim, tj. zelenim LED-om (crveno = vreme je isteklo, zeleno = tiket je jos uvek važeći). Ukoliko vozač obnovi svoj tiket, to se broji kao novi unos - a smatra se da će obaveštenja o isticanju tiketa obaviti Telekom Srbije.

Nakon što je ustanovljeno da imamo registracijski broj vozila, on se propušta kroz bazu kriminalnih zapisa MUP-a Republike Srbije, gde se proverava da li je vozilo traženo od službenih organa zbog zakonskih prekršaja. Ukoliko se utvrdi da vozilo jeste traženo - kontaktira se dispečer policije, koji nakon toga obaveštava najbližu policijsku jedinicu na terenu o situaciji. U ovom slučaju se savetuje ljudskim radnicima da, koliko je moguće diskretno, zadrže traženog vozača na mestu parkinga dok službena lica ne dođu na scenu - ali ne po cenu ugrožavanja sopstvenog života. Takođe se savetuje da se, ukoliko vozač pokuša da pobegne, spreči odlazak traženog vozila tako što se aktiviraju blokade na izlazu sa tog parking mesta. Dalji protokoli nisu odgovornost PSNP sistema i neće biti objašnjavani u ovoj dokumentaciji.

Takođe, ukoliko je reč o parking mestu rezervisanom za hendikepirane osobe, proverava se da li to vozilo pripada registrovanom invalidu. Ukoliko ne pripada, obaveštava se kontroler da ispita situaciju.

Prilikom rada PSNP sistema, neminovno je da u jednom trenutku dođe do greške sistema. U tom slučaju, sistem se neće terminisati, nego će se javiti poruka o grešci, praćena zvučnim upozorenjem koje je namenjeno da kontroleru skrene pažnju na problem. Poruka o grešci sadrži kod greške, kao i izvor - pored komandi za ignorisanje (i nastavljanje sa radom) ili zaustavljanje rada sistema. Zaustavljanje rada sistema se takođe može obaviti pritiskom na dugme *STOP* u gornjem desnom uglu interfejsa. Ukoliko je sistem iz nekog razloga zaustavljen, podaci o parking mestima neće biti ugroženi (osim u slučaju kvara hardvera), te kontroler može bezbedno da radi na otklanjanju kvara dok sistem ne radi - ali u to vreme novi unosi ne mogu biti primljeni od strane SCADA sistema, te to treba imati na umu prilikom korišćenja PSNP-a.

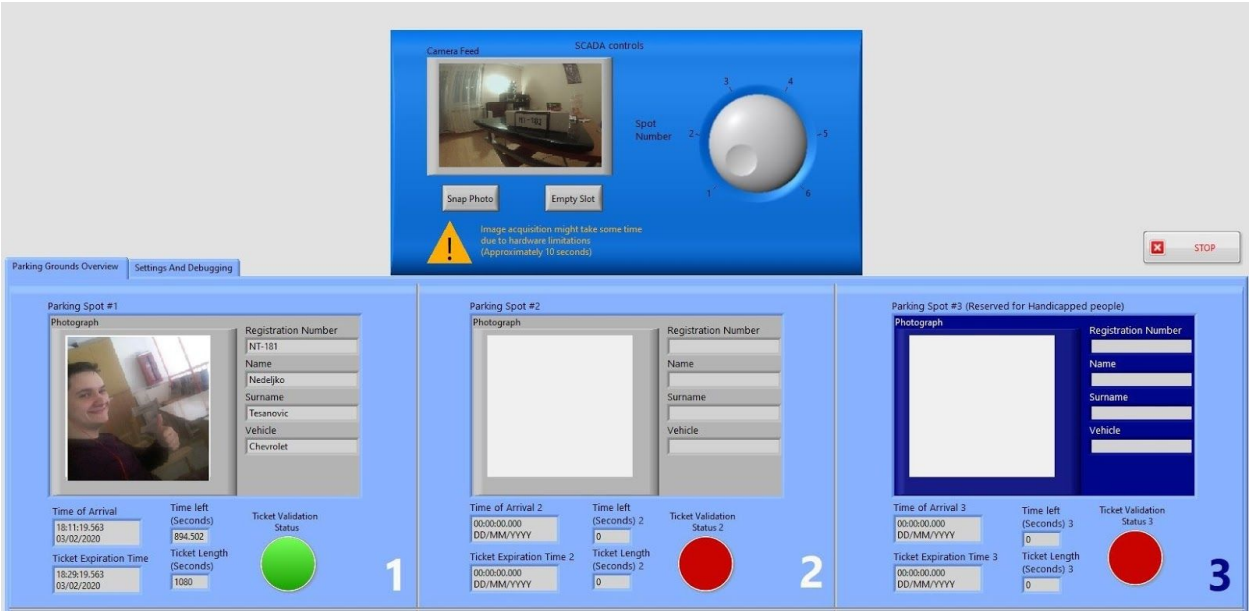
Vision Assistant - Obrada slike i Neuronska Mreža

Za realizaciju čitanja teksta sa registracijske tablice vozila je upotrebljen alat Vision Assistant iz Vision and Motion modula. U njemu je izvršena obrada slike, kao i čitanje putem neuronske mreže, u jednom mahu, tj. kao ulaz prima sliku, a kao izlaz daje tekst - sve za cenu jednog Express VI-ja.

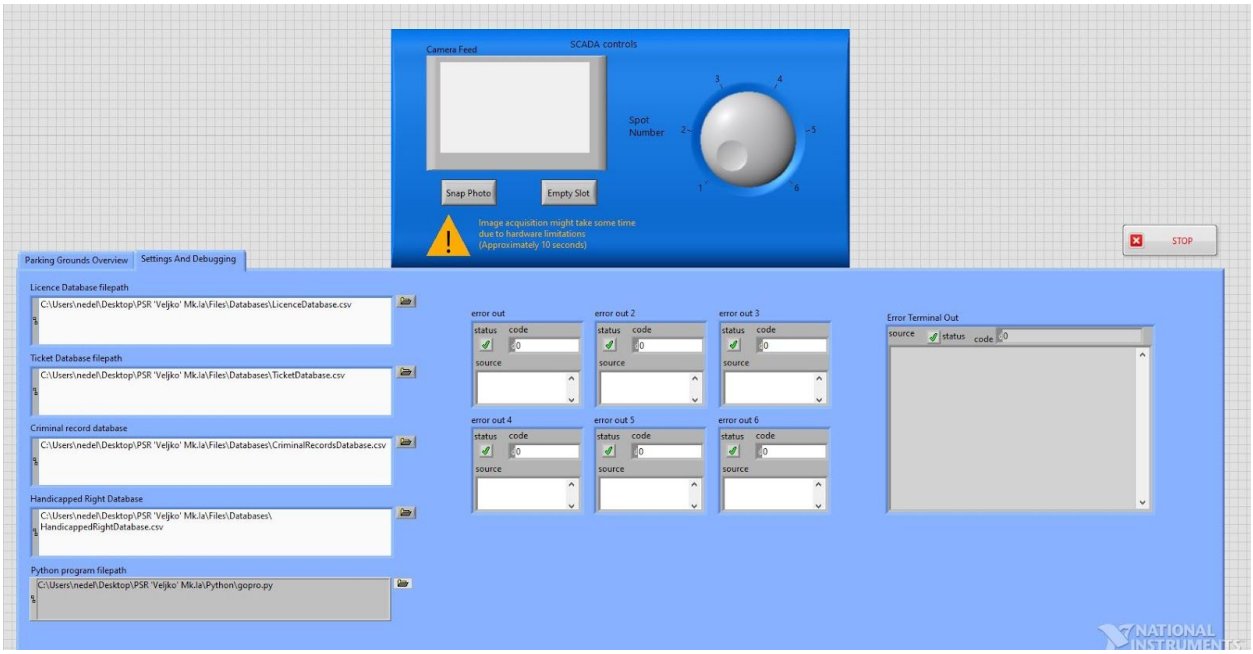
Pošto su vozila i tablice približno istih dimenzija, teren parkinga se ne menja, a PSR "Veljko" M1A svoj zadatak obavlja repetativno i efikasno bez većih devijacija - možemo pouzdano da tvrdimo da će se tablice uvek pojaviti u određenom regionu slike. Nazovimo taj region *region interesovanja* - ROI. Pozicija ROI u slici je izabrana kao medijan stotina slika. Pošto nas ne zanima ništa van ROI-a, sliku sečemo i posmatramo sam ROI, koji sada sadrži samo sliku tablice. Novodobijena slika je dekolorizovana, a zatim se njen kontrast povećava da bi crni tekst bio upadljiviji u odnosu na belu pozadinu tablice - to se radi da bi se maksimizovale šanse uspešnog čitanja teksta sa neuronskom mrežom.

Neuronska mreža koristi OCR/OCV (Optical Character Recognition/Verification) biblioteku ugrađenu u Vision And Motion modul. Mreža je istrenirana tako što je kroz nju propušteno na stotine slika, te su detektovani karakteri adekvatno sačuvani. Sam proces prepoznavanja karaktera se svodi na posmatranje pravougaonika koji je istih dimenzija kao očekivani karakter širom slike, i poređenje sa slikama poznatih karaktera (OCR element). Zarad jednostavnosti, pošto je u pitanju demonstracioni primer rada sistema, mreža je istrenirana za 20 karaktera:
Slova A, E, H, K, N, P, R, T, U i, X,
Kao i cifre 0, 1, 2, 3, 4, 5, 6, 7, 8 i 9
Pošto su tablice uvek istog formata (dva slova, crta, pa tri cifre - taj format se očekuje pri čitanju (OCV element - u slučaju “sličnih” karaktera, kao B i 8, očekivani format pomaže pri čitanju).
Nakon čitanja, potrebno je ukloniti sve razmake u tekstu, ukoliko ih ima.
Ekstremni slučajevi, gde su u pitanju specijalni karakteri ili nekonvencionalan izgled tablice, kao i slučaju zakazivanja neuronske mreže su pokriveni ručnim unosom tablice u sistem.

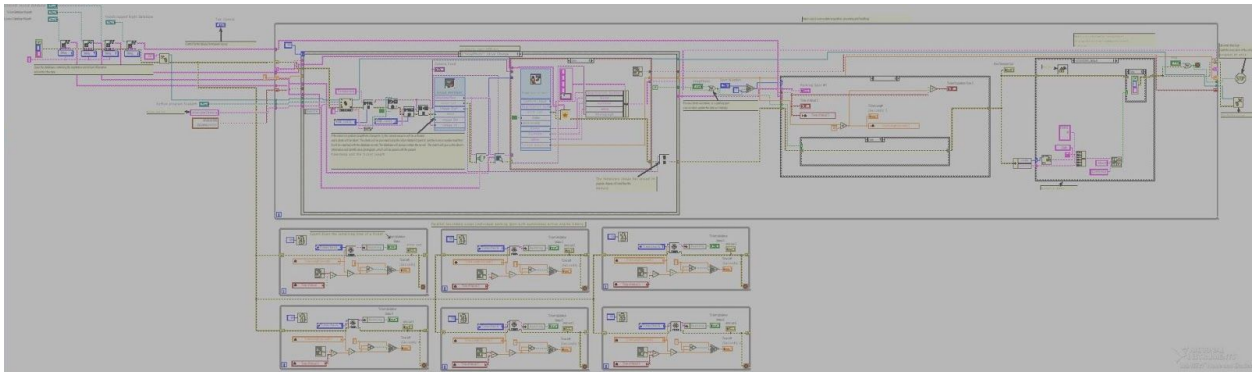
Korisničke kontrole (Front Panel)



Ovo je korisnički front panel sa interfejsom parking mesta i simuliranim SCADA kontrolama očekivanih signala. Posebno mesto rezervisano za hendikepirane osobe je ofarbano u tamno plavu boju radi identifikacije. Mesta su numerisana u odnosu na maketu parkinga, i lako se mogu rearanžirati, dodavati/uklanjati radi uklapanja sa bilo kakvim dizajnom parkinga. Interfejs se sastoji od dve stranice, kojima se može pristupati preko klika na listić gornje ivice gornjeg levog ugla. Ispod je data slika stranice podešavanja, koja služi da se podese parametri baza podataka, kao i da prikažu greške.

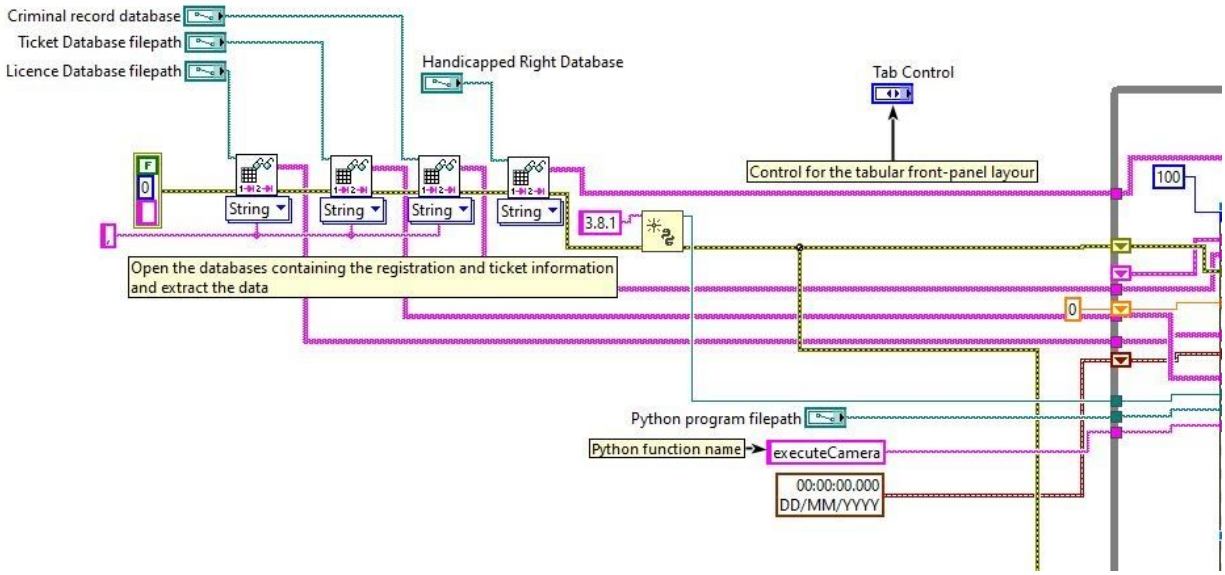


Blok dijagram

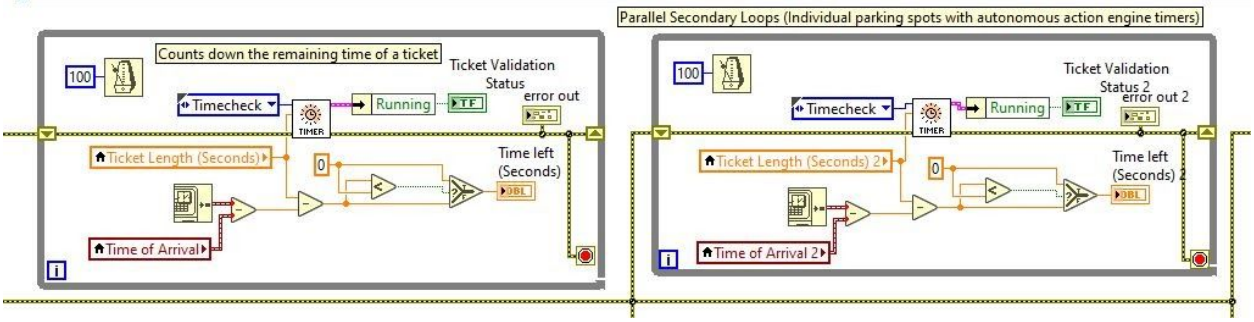


Ovo je blok dijagram čitave LabVIEW aplikacije. Sastoji se iz tri ključna dela:

➔ Inicijalizacijski blok - gde se učitavaju podešeni parametri baza i pajtona

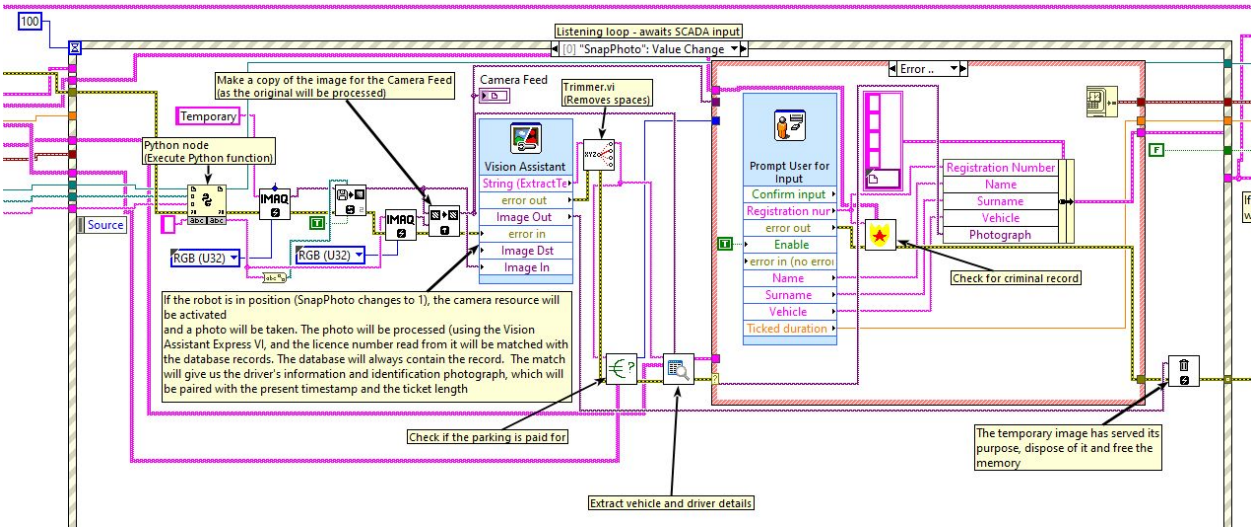


➔ Paralelne petlje individualnih parking mesta - svaka petlja koristi Timer.

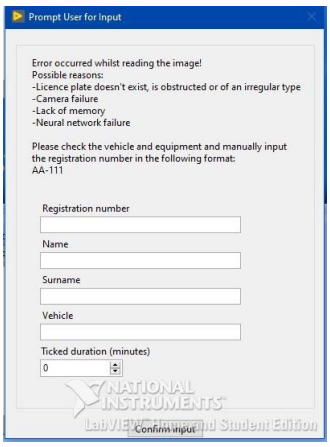


Timer - VI realizovan od strane prof dr. Borisa Jakovljevića, i ima svoj Timer Action Engine. U slučaju zaustavljanja programa, i ponovnog pokretanja, tajmeri će nastaviti sa radom kao da se ništa nije desilo - to je posledica realizacije tajmera sa Timestamp logikom.

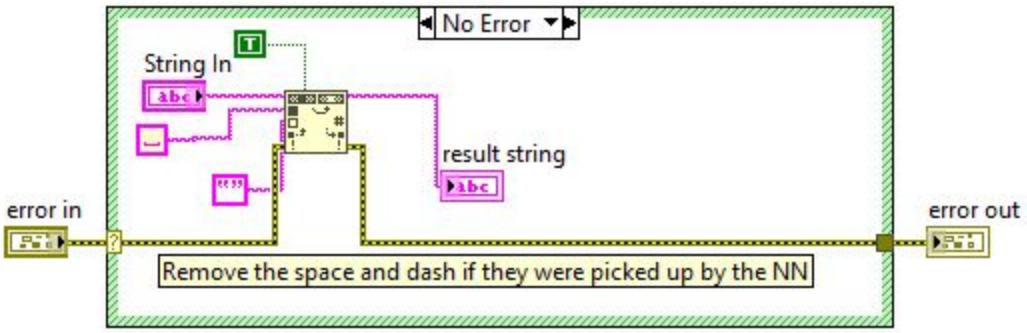
➔ Glavna petlja - Pri pojavi rastuće ivice signala senzora prisustva, izvršava se



executeCamera python funkcija koja pristupa kameri na robotu i prikuplja sliku. Zbog hardverskih limitacija (spora konekcija) ovaj proces može potrajati desetak sekundi. Slika se čuva na HDD-u kontrolne mašine radi evidencije, a zatim se putanja do nje same prosleđuje IMAQ funkcijama koje je učitavaju u privremenu LabVIEW memoriju u formi Image žice. Ova žica, koja sada sadrži informaciju u vidu slike, se dovodi na Vision Assistant express VI koji sadrži neuronsku mrežu za obradu te slike. Izlaz iz njega je broj registarske tablice u vidu stringa, koji se zatim koristi da bi se dobila informacija o plaćenom vremenu parkiranja, informacije o vozaču i vozilu, te i da li vozač ima kriminalnu istoriju (u poslednjem slučaju se aktivira i zvučni signal). U slučaju zakazivanja sistema, traži se ručni unos svih podataka, i smatra se da su oni uvek tačni (za sliku se u tom slučaju postavlja neobrađena slika dobijena od kamere). U nastavku su date slike za prozor ručnog unosa, kao i za obaveštavanje o grešci i kriminalcu.

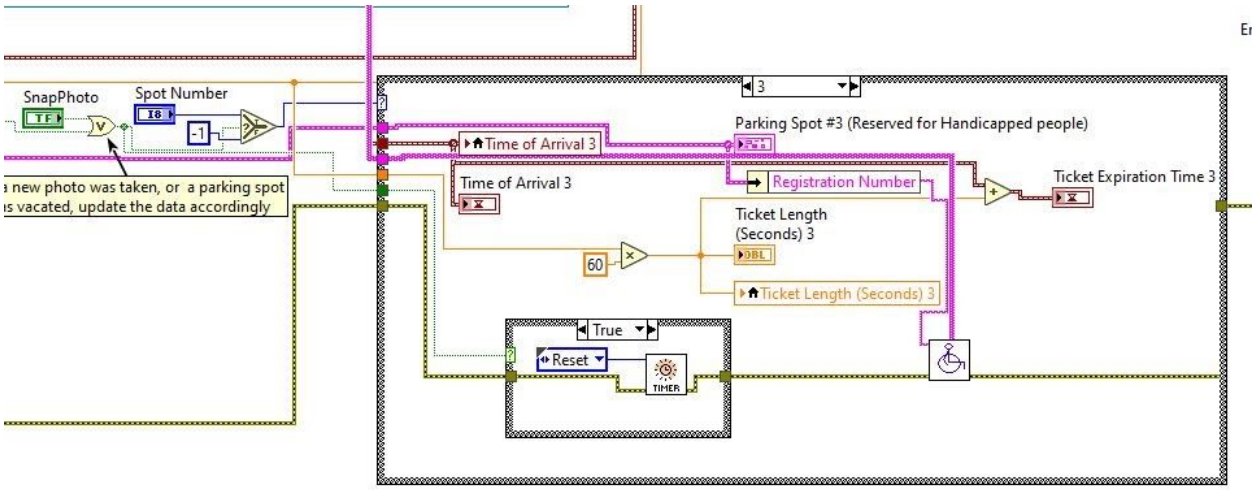


➔ Trimmer.vi uklanja sve razmake unutar stringa.

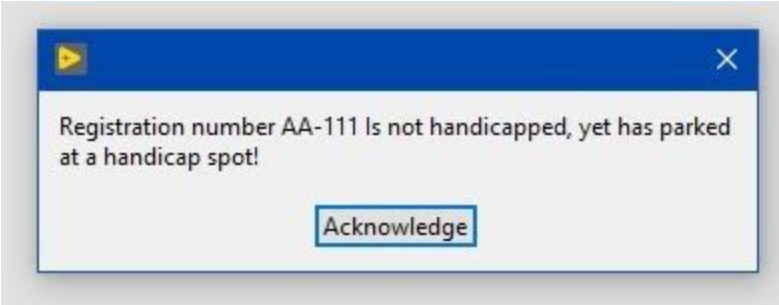


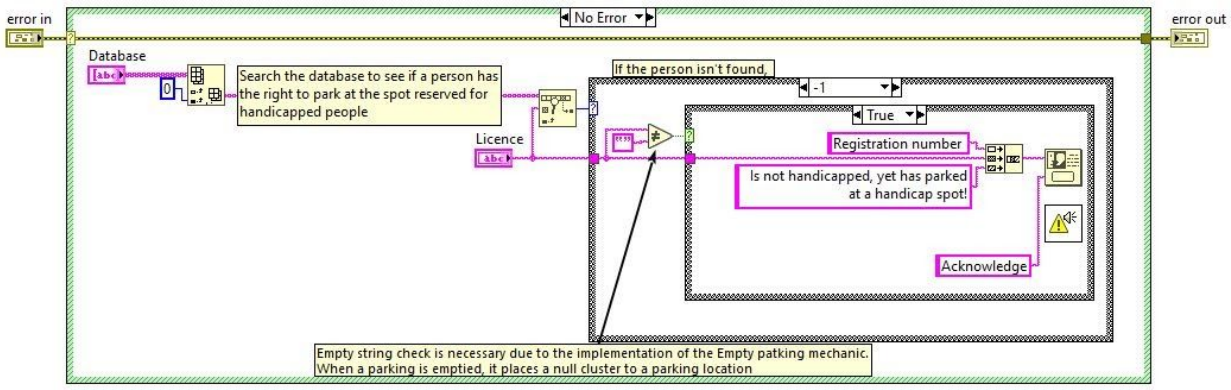
Interna logika

Proveravanje za kriminalni zapis se vrši u CheckForCriminalRecord.vi

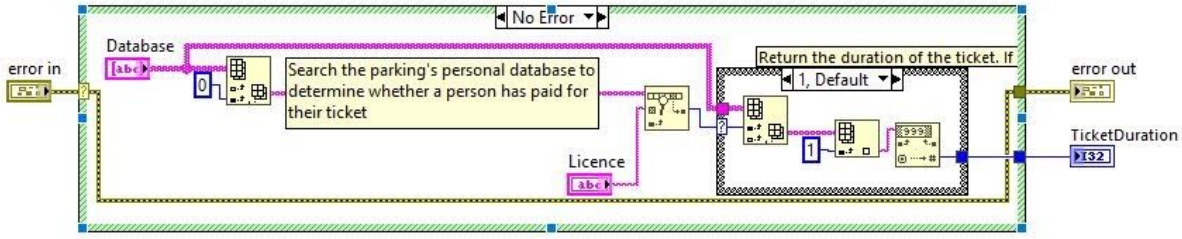


Proveravanje za status invalida se vrši u CheckForHandicap.vi - ova provera se vrši samo ukoliko je parking mesto rezervisano za invalide. U slučaju parkiranja osobe koja nije invalid na mesto invalida, ispisuje se poruka koja kontrolera obaveštava o događaju

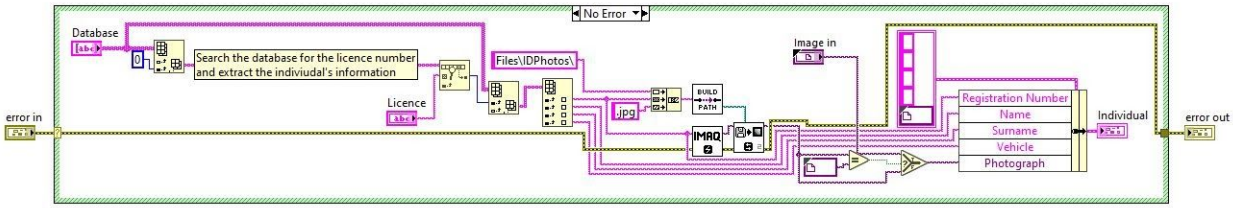




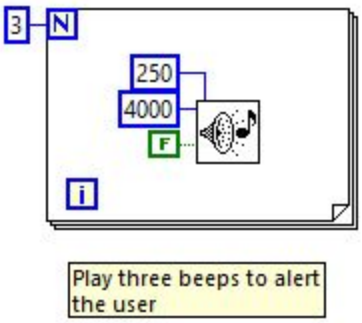
Proveravanje statusa uplate tiketa se vrši u SearchForTicket.vi

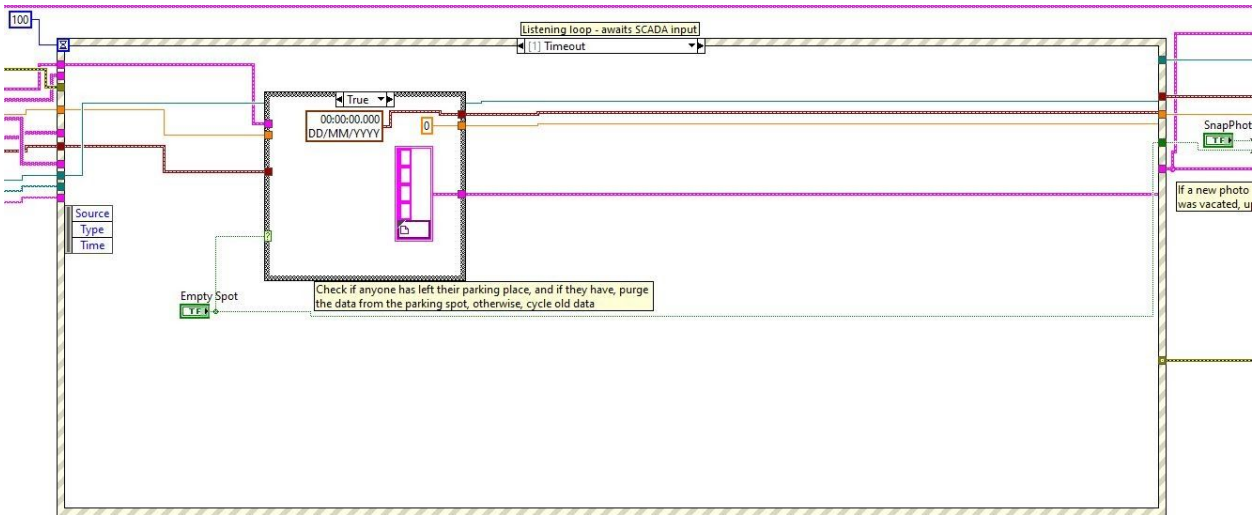


Traženje podataka vozača i vozila se vrši u SearchDatabase.vi

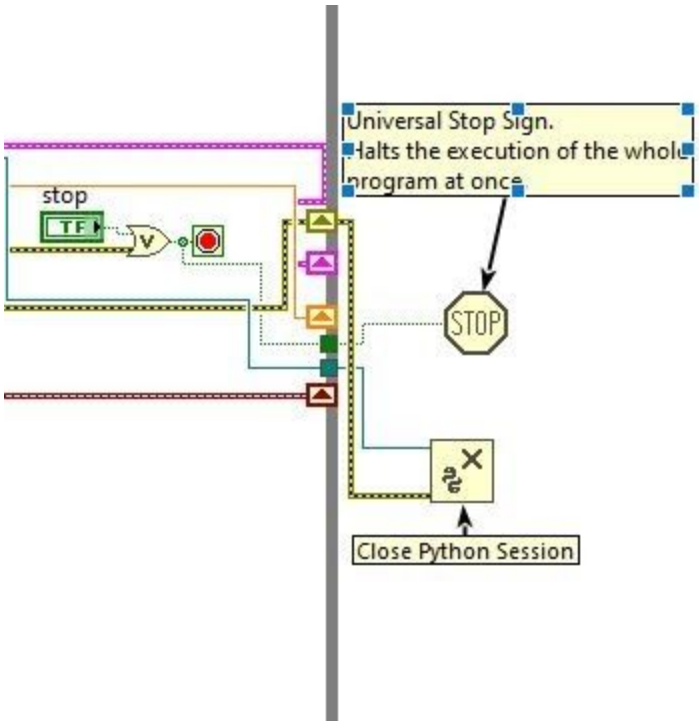


Zvučni signal je realizovan u Alert.jpg i sastoji se od tri uzastopna piska. Pražnjenje parkinga se obavlja tako što se deteketuje opadajuća ivica signala senzora prisustva i u parking upiše prazan član i resetuje tajmer za to parking mesto



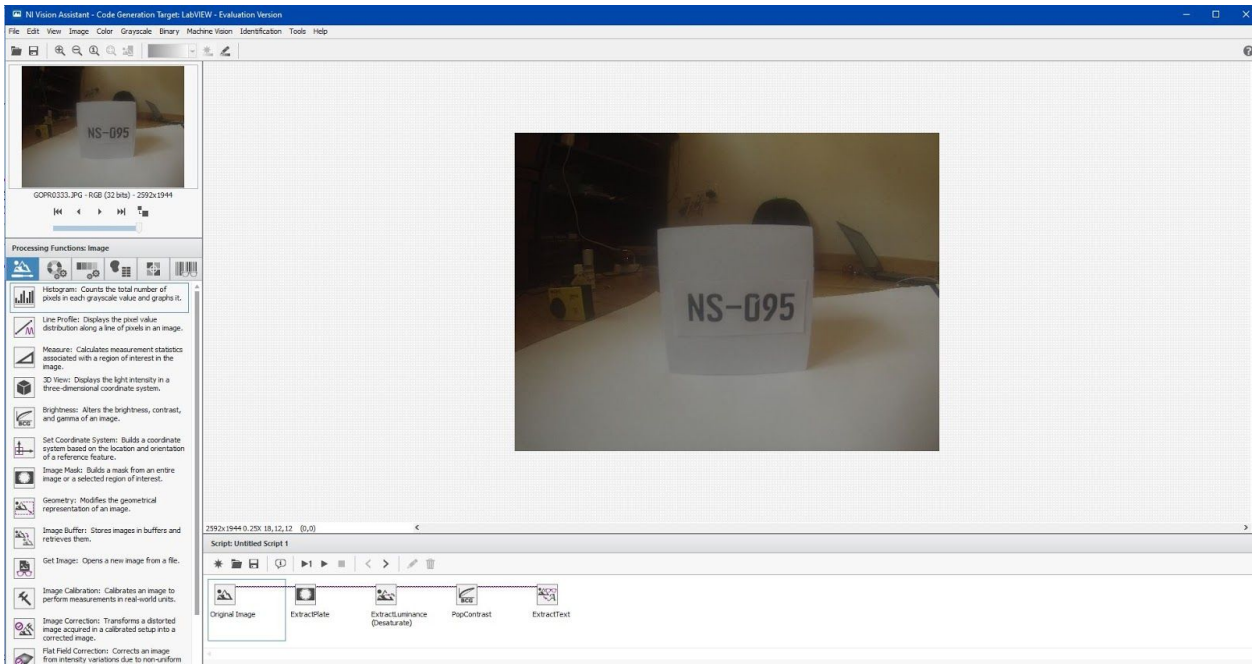


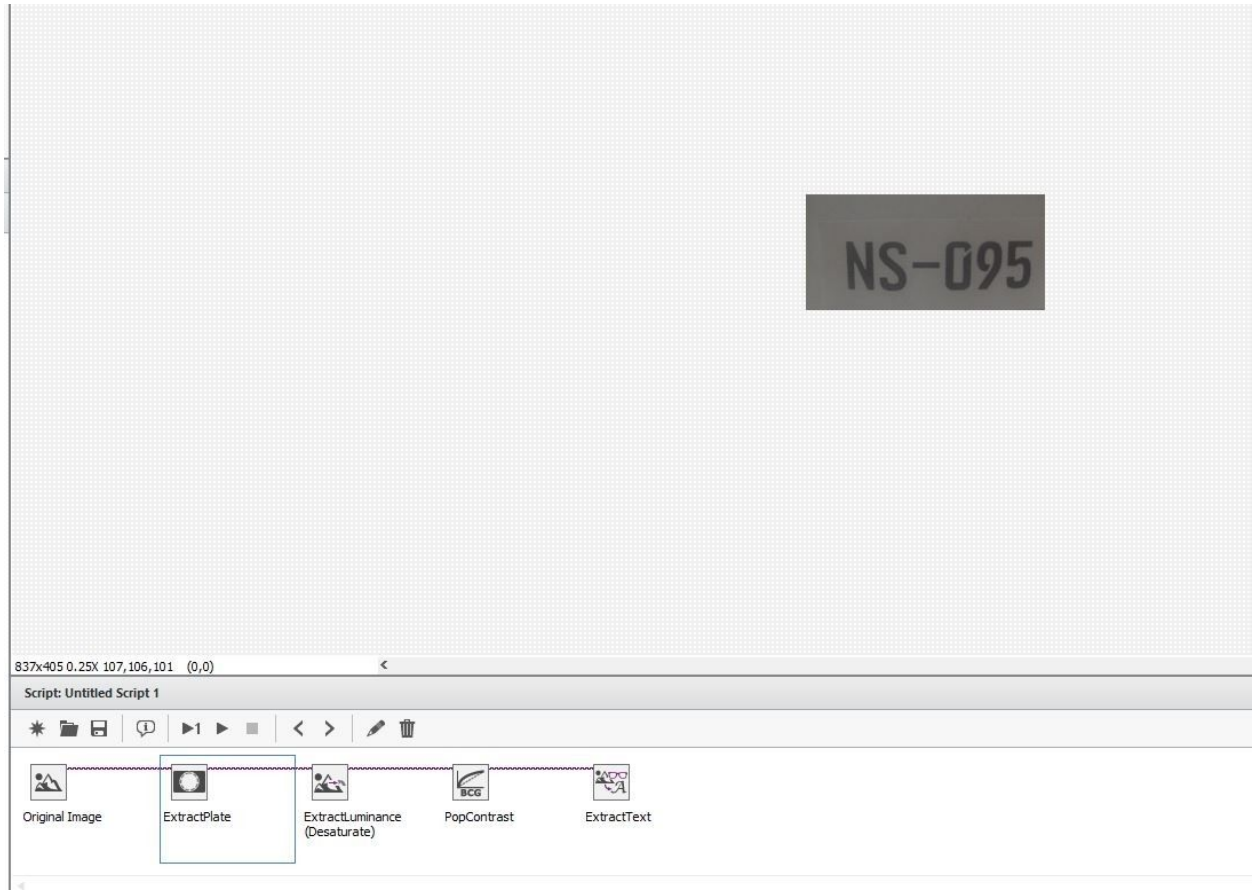
Zaustavljanje programa se obavlja zajedničkim stop signalom koji sve petlje zaustavlja istovremeno, nezavisno od njihovih pozicija i hijerarhija.

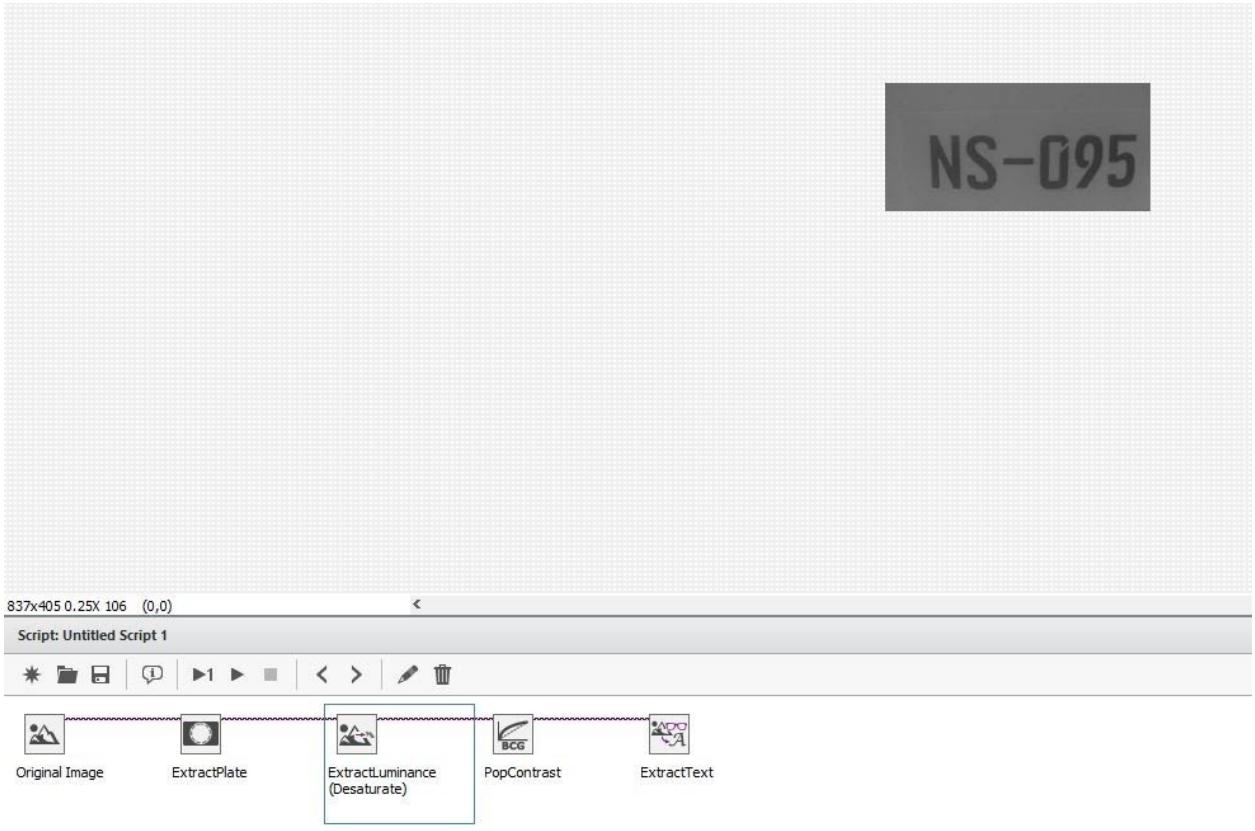


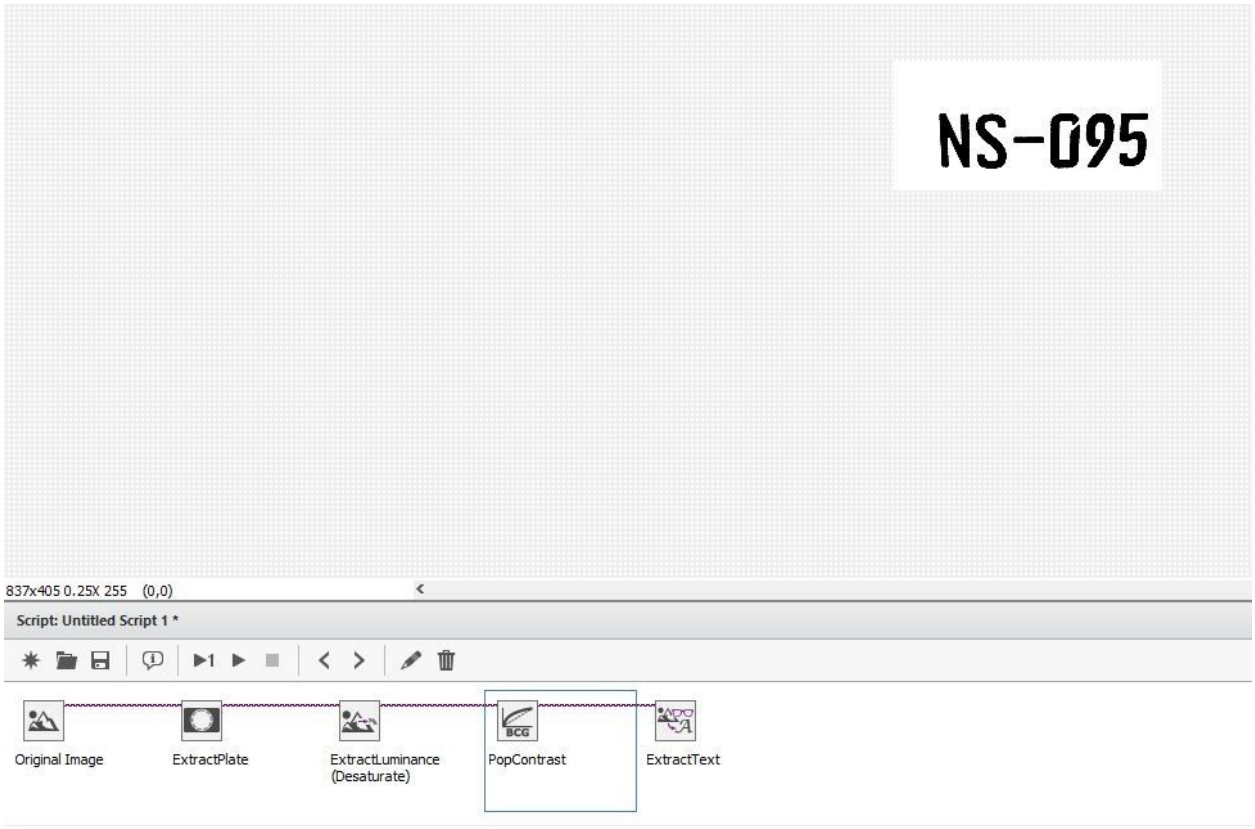
Vision Assistant

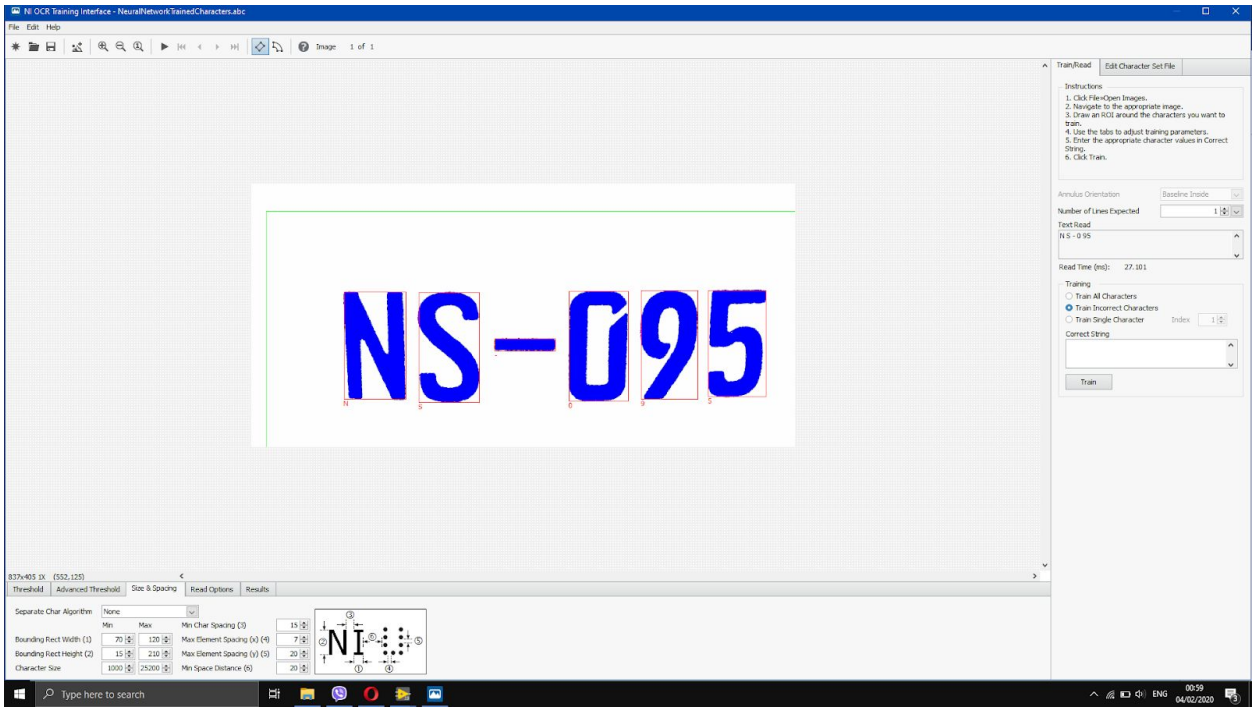
Rad sistema za prepoznavanje tablica je u prethodnom tekstu bio opisan, a za detaljniji način rada možete posetiti zvaničnu stranicu. U narednim slikama možete videti kako to praktično izgleda.






NS-095
837x405 0.25X 106 (0,0)
Script: Untitled Script 1
Original Image
ExtractPlate
ExtractLuminance (Desaturate)
PopContrast
ExtractText


NS-095
837x405 0.25X 255 (0,0)
Script: Untitled Script 1 *
Original Image
ExtractPlate
ExtractLuminance (Desaturate)
PopContrast
ExtractText


NI QCR Training Interface - NeuralNetworkTrainedCharacters.abc
File Edit Help
Image 1 of 1
NS-095
Train/Read
Edit Character Set File
Instructions
1. Click File>Open Images.
2. Navigate to the appropriate image.
3. Draw an ROI around the characters you want to train.
4. Use the tabs to adjust training parameters.
5. Enter the appropriate character values in Correct String.
6. Click Train.
Annulus Orientation
Baseline Inside
Number of Lines Expected
Text Read
N S - 0 95
Read Time (ms): 27.101
Training
Train All Characters
Train Incorrect Characters
Train Single Character
Index
Correct String
Train
837x405 IX (552,125)
Threshold
Advanced Threshold
Size & Spacing
Read Options
Results
Separate Char Algorithm
None
Min
Max
Bounding Rect Width (1)
Bounding Rect Height (2)
Character Size
Min Char Spacing (3)
Max Element Spacing (x) (4)
Max Element Spacing (y) (5)
Min Space Distance (6)
Type here to search
ENG
00:59
04/02/2020

gopro.py modul

U prethodnom tekstu, navedeno je korišćenje python funkcije za dobijanje slike. Pošto direktno korišćenje kamere nije bilo moguće zbog softverskih ograničenja, slike se preuzimaju pomoću python skripte. Kompletan kod je dotupan na linku sa kodom robota (pod nazivom gopro.py).

Kada je kamerica putem WIFI-ja povezana sa računarom, diže se njen *hotspot*, i preko njega može da se pristupi svim njenim fajlovima putem HTTP protokola.

Funkcija koju poziva LabView jeste `executeCamera()`, i njena povratna vrednost je putanja do fajla koja je neophodna python node-u da nastavi obradu slike. Ova funkcija poziva druge, pomoćne funkcije, koje će sada biti ukratko objašnjene.

Pre svega, potrebno je kamerici dati signal da slika. To je posao funkcije `takePicture()`. Sve što ona radi, jeste pozivanje određenog *GET* zahteva. Posle svakog poziva, neophodno je zaustaviti program na nekoliko sekundi, da bi se obradio zahtev i dobio odgovor.

```
def takePicture():
    #changes the mode of the camera to photo mode
    r = requests.get('http://10.5.5.9/camera/CM?t=goprohero&p=%01')
    time.sleep(2)

    #activates the shutter, thus taking a picture
    p = requests.get('http://10.5.5.9/bacpac/SH?t=goprohero&p=%01')
    time.sleep(2)

    #checks whether either of the http requests failed,
    #code 200 denotes successful completion
    if p.status_code != 200 or r.status_code != 200:
        print("ERROR, http request failed!")
```

Kada smo napravili fotografiju, moramo da saznamo pod kojim imenom je ona memorisana, da bi smo mogli da je preuzmemo. Da bi to uradili, prvo sa liste svih fotografija moramo da dobijemo ime poslednje. Olakšavajuća činjenica jeste da kamerica čuva fotografije po vrednostima, i poslednja slika će uvek imati najveću vrednost. Povratna vrednost je ime poslednje fotografije.

Ovaj deo je implementiran u funkciji `getLatestPictureName()`.

```
def getLatestPictureName():
    r = requests.get('http://10.5.5.9:8080/gp/gpMediaList')
    data = r.json()

    index_of_latest = 0
    value_of_latest = 0

    for index,x in enumerate(data['media'][0]['fs']):
        if x['n'][:4] == "GOPR" :
            if int(x['n'][4:8]) > value_of_latest:
                value_of_latest = int(x['n'][4:8])
                index_of_latest = index

    picture = data['media'][0]['fs'][index_of_latest]['n']
    return picture
```

Sve što je preostalo, jeste preuzimanje slike pod imenom koje smo pronašli.

```
def downloadPicture(picture_name):
    url = 'http://10.5.5.9:8080/videos/DCIM/100GOPRO/' + picture_name
    response = requests.get(url, stream=True)
    with open(r'C:/path_example/your_directory', 'wb') as out_file:
        shutil.copyfileobj(response.raw, out_file)
    del response
```

Fotografija će biti sačuvana u vašem direktorijumu (čiju ste putanju naveli gore). Ove tri funkcije se redom pozivaju u `executeCamera()`, i omogućavaju automatizovan sistem slikanja.

Nesavršenost dizajna

Ovaj proces je složen, samim tim je veoma sklon poremećajima i greškama. Potencijalne greške do kojih može doći su brojne: kamera može da zakaže, slika može da se neadekvatno obradi, neuronska mreža može da pogrešno pročita znakove, vozilo može da bude neregistrovano te ne bude pronađeno u bazi podataka, robot može da usled loše procene kretanja mnogo skrene sa puta... Čak i prilikom nominalnog rada sistema gde nije došlo do hardversko-softverske greške (vozilo je bez tablice, tablica je prekrivena, robot se loše pozicionirao...), uveden je mehanizam ručnog unosa pri pojavi otkazivanja bilo kog dela sistema. Prilikom ručnog unosa, ljudski radnik na parkingu će da obavi izviđanje/otklon kvara i da kontroleru u kontrolnoj kuli javi podatke koje će kontroler da unese u sistem, i nastaviće se kao da nije došlo do greške.

Problemi na koje smo naišli

Kao i u svakom projektu, suočili smo se sa raznim izazovima. Prethodno rešenje projekta jeste i način na koji smo mi neke od tih problema rešili, mada uvek postoji mnogo načina.
Početni plan je bio da LabView i PSR ‘Veljko’ M1A imaju direktnu komunikaciju. Međutim, saznali smo da robot, kako ima svoju kopiju svih programa, nikako ne može da funkcioniše bez posrednika. Jedan od načina jeste bio da robot pristupa serveru, i tako komunicira sa spoljašnjosti, ali kako nema pristup internetu, ni ta ideja se nije mogla realizovati. Ono što je usvojeno jeste emuliranje senzora sa korisničkim unosom, za šta smo morali posebno istraživati način SSH komunikacije i pokretanja. Rotacija i kretanje su se malo promenili usled dodatnog tereta kamere, i realna vrednost uglova je dobijena eksperimentalno. Kod skeniranja tablica, morali smo posebno prilagođavati izgled okruženja, da ne bi uzrokovali smetnje za neuronsku mrežu. Bio je neophodno odrediti prosečnu poziciju tablice, što u realnoj situaciji ne bi bilo moguće. Ukoliko se robot zbog nekog poremecaja u kretanju ne rotira tacno 90 stepeni, ugao kametice nece biti normalan na tablice, i neuronska mreza potencijalno nece moci da prepozna tablice. Kamera je bila problem za sebe. Bila nam je potrebna kamerica sa WIFI aplikacijom, da kablovi ne bi sputavali robota. Kako Windows 10 operativni sistem ne može zbog svog protokola da vidi tu kamericu, slikanje smo realizovali preko hotspot-a koji kamerica pravi pri konektovanju. Posebna python skipta je preko http protokola preuzimala sliku i čuvala je u naznačeni folder. Ovo je samo deo problema, koji su oduzeli najviše vremena, i najviše doprineli odstupanju od prvobitne ideje realizovanja.

Zaključak

Zadatak projekta je bio automatizovati rad parking servisa, uz pomoć LEGO Mindstorms EV3 robota.

Ovo je realizovano do maksimalne mogućnosti dostupnih sredstava. Robot uspešno dobija informaciju na koje parking mesto da se pozicionira, uspešno na aplikaciji preuzimamo sliku, i iz nje vadimo registracijski broj. Većinu potencijalnih greški smo uspeli da pokrijemo, da sistem ne bi došao u stanje otkaza.

Deo koji nije automatizovan, a poboljšao bi projekat, jeste komunikacija robot-SCADA, za šta je bilo potrebno mnogo više vremena, resursa i materijala. To je dovelo do ograničenja realizacije - robot može da proverava samo jedno parking mesto u jednom prolazu parkinga, što je ograničila upravo potreba za korisničkim unosom koji emulira komunikaciju sa senzorom. Takođe, robot mora da bude usmeren normalno na tablicu, i smatra se da je automobil savršeno parkiran, da bi se obezbedilo sigurno vađenje teksta sa tablica. Ukoliko to nije slučaj, obezbeđen je ručni unos.

Za vođenje projekta korišćena je platforma TRELLO-Gold (link).

Reference

1. https://vk.com/doc348852382_457508879?hash=2b5439ab62d6ab917b&dl=30bd0f4ade23474e92
2. https://forums.ni.com/t5/Example-Programs/Programmatically-Train-IMAQ-OCR-Characters-using-LabVIEW/ta-p/3510439?profile.language=en
3. https://www.sciencedirect.com/science/article/pii/S166564231470598X
4. https://www.wikipedia.org/
5. https://stackoverflow.com/
6. https://antonsmindstorms.com/2019/06/15/how-to-run-python-on-an-ev3-brick/
7. https://le-www-live-s.legocdn.com/sc/media/files/ev3-micropython/ev3micropythonv100-71d3f28c59a1e766e92a59ff8500818e.pdf
8. https://www.ni.com/en-rs.html