ŠOLSKI CENTER CELJE

Srednja šola za strojništvo, mehatroniko in medije

Raziskovalna naloga

DETEKCIJA PARAMETROV OKOLICE PRI ROBOTSKEM REŠEVANJU

Avtorji:

Žan MARINŠEK, M-4. c

Florijan PLOHL, M-4. c

Tomaž REJC ZAGOŽEN, M-4. c

Marcel URANKAR, M-4. c

Mentorji:

mag. Matej VEBER, univ. dipl. inž.

mag. Andro GLAMNIK, univ. dipl. inž.

Miloš Bevc, univ. dipl. inž.

Mestna občina Celje, Mladi za Celje

Celje, februar 2019

IZJAVA*

Mentorji, Matej Veber, Andro Glamnik in Miloš Bevc v skladu z 2. in 17. členom Pravilnika raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje, zagotavljamo, da je v raziskovalni nalogi z naslovom Detekcija parametrov okolice pri robotskem reševanju, katere avtorji so Žan Marinšek, Florijan Plohl, Tomaž Rejc Zagožen in Marcel Urankar:

- besedilo v tiskani in elektronski obliki istovetno,
- pri raziskovanju uporabljeno gradivo navedeno v seznamu uporabljene literature,
- da je za objavo fotografij v nalogi pridobljeno avtorjevo dovoljenje in je hranjeno v šolskem arhivu,
- da sme Osrednja knjižnica Celje objaviti raziskovalno nalogo v polnem besedilu na knjižničnih portalih z navedbo, da je raziskovalna naloga nastala v okviru projekta Mladi za Celje,
- da je raziskovalno nalogo dovoljeno uporabiti za izobraževalne in raziskovalne namene s povzemanjem misli, idej, konceptov oziroma besedil iz naloge ob upoštevanju avtorstva in korektnem citiranju,
- da smo seznanjeni z razpisni pogoji projekta Mladi za Celje.

Celje, _____ žig šole Podpis mentorja

Podpis odgovorne osebe

*POJASNILO V skladu z 2. in 17. členom Pravilnika raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje je potrebno podpisano izjavo mentorja in odgovorne osebe šole vključiti v izvod za knjižnico, dovoljenje za objavo avtorja fotografskega gradiva, katerega ni avtor raziskovalne naloge, pa hrani šola v svojem arhivu.

DOVOLJENJE ZA OBJAVO AVTORSKE FOTOGRAFIJE V RAZISKOVALNI NALOGI

Podpisani Žan Marinšek, Florijan Plohl, Tomaž Rejc Zagožen in Marcel Urankar izjavljamo, da smo avtorji fotografskega gradiva, navedenega v priloženem seznamu in dovoljujemo v skladu z 2. členom Pravilnika raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje, da se lahko uporabi pri pripravi raziskovalne naloge pod mentorstvom Mateja Vebra, Andra Glamnika in Miloša Bevca z naslovom Detekcija parametrov okolice pri robotskem reševanju, katere avtorji so Žan Marinšek, Florijan Plohl, Tomaž Rejc Zagožen in Marcel Urankar. Dovoljujemo tudi, da sme Osrednja knjižnica Celje vključeno fotografsko gradivo v raziskovalno nalogo objaviti na knjižničnih portalih z navedbo avtorstva v skladu s standardi bibliografske obdelave.

Celje, _____ Podpis avtorjev:

Priloga: – seznam fotografskega gradiva

ZAHVALA

Zahvaljujemo se vsem, ki so nam pomagali pri izdelavi naše raziskovalne naloge. Posebej se zahvaljujemo našim mentorjem mag. Mateju Vebru, univ. dipl. inž., mag. Andru Glamniku, univ. dipl. inž., in Milošu Bevcu, univ. dipl. inž., ki so nam dajali strokovne nasvete in nas vzpodbujali. Zahvaljujemo se tudi Brigiti Renner, prof., za lektoriranje naloge in mag. Simoni Brečko, prof., za lektoriranje angleškega prevoda povzetka.

POVZETEK

V današnjem svetu se dogaja veliko naravnih katastrof, ki jih rešujejo reševalne ekipe, izpostavljene raznim tveganjem in nesrečam. V ta namen smo si zadali cilj, da s pomočjo kamere zaznavamo okolico in to v prihodnosti vgradimo v reševalni robot, ki ga izdelujemo za RoboCup Rescue RMRC tekmovanje, ki bo v letošnjem letu potekalo v Avstraliji. Del tekmovanja predstavlja tudi zaznavanje okolice in hazmat simbolov za nevarnost. Najprej smo se seznanili z vsemi izzivi, ki jih ponuja tekmovanje, nato smo se lotili načrtovanja reševalnega robota, ki bo v celoti 3D natisnjen in bo zadoščal vsem zahtevam tekmovanja. V raziskovalni nalogi smo podrobneje raziskali zaznavanje okolice s pomočjo kamere in vse težave, na katere smo naleteli. Ob tem smo pridobili veliko novih znanj, sodelovali smo v timu in pridobili neprecenljive izkušnje za našo nadaljnjo strokovno pot.

SUMMARY

In today's world, there are lot of natural disasters that are being addressed by rescue teams, exposed to various risks and accidents. Because of that, we set ourselves a goal to detect the surroundings with the help of the camera and in the future, we will install it into our rescue robot that we have been making for the RoboCup Rescue RMRC competition, which will take place in Australia this year. Part of the competition is also the perception of the environment and the hazmat danger symbols. First, we became acquainted with all the challenges that the competition offers, then we started designing a rescue robot that will be fully 3D printed and will meet all the requirements of the competition. In our research, we studied the perception of the surroundings with the help of the camera and all the problems we encountered. In addition, we gained a lot of new knowledge, we participated in the team and gained invaluable experience for our further professional career.

KAZALO VSEBINE

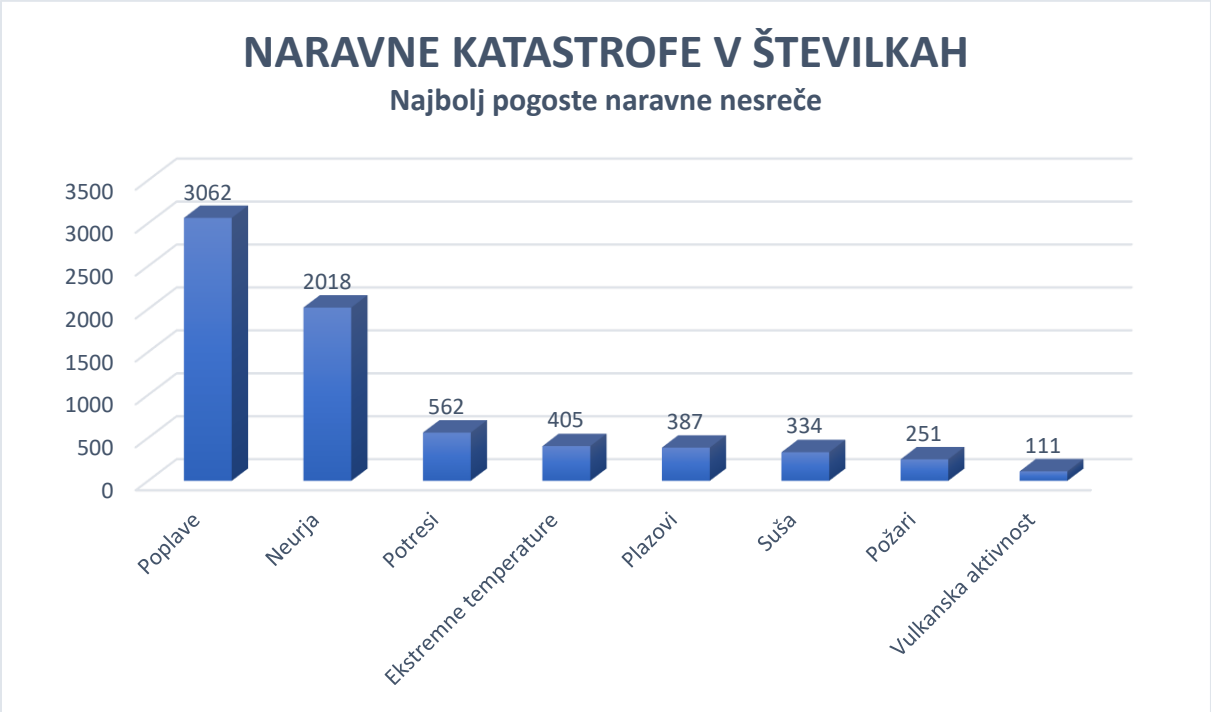
KAZALO SLIK

KAZALO TABEL

KAZALO GRAFOV

1 UVOD

Vsak dan se po svetu dogajajo nesreče, narava je zelo nepredvidljiva in povzroči veliko smrtnih žrtev. V zadnjem stoletju se je število naravnih katastrof močno povečalo. Razlogov za povečanje njihovega števila je več, od sprememb podnebja do velikega porasta števila ljudi, ki živijo v gosto naseljenih območjih sveta. Naravne nesreče, ki so posledica vremenskih razmer, so v zadnjih dveh desetletjih terjale 606 000 življenj. Število naravnih nesreč se iz leta v leto še vedno povečuje. Najpogostejše katastrofe so poplave, potresi in neurja, ki so najbolj smrtonosna. [1]



Slika 1: Najpogostejše naravne nesreče

(Vir:https://siol.net/novice/novice/katere-naravne-nesrece-so-najbolj-smrtonosne-394876)

Slovenijo kar pogosto prizadenejo potresi, ki nastanejo zaradi nenadnih premikov tektonskih plošč v globini zemeljske skorje. Zadnji močnejši so bili v letih 1998 in 2004 v Posočju. [2]

Slika 2: Potres leta 1998 v Zgornjem Posočju

(Vir:http://www.arso.gov.si/potresi/potresna%20aktivnost/potres1998.html)

Žrtev naravnih nesreč je veliko, kljub temu da imamo reševalne ekipe, ki rešujejo ljudi iz nevarnih okoliščin. Mnogokrat se zgodi, da reševalci sami postanejo žrtve zaradi nepredvidenih nevarnosti. V ta namen smo si zadali cilj, da izdelamo reševalnega robota, ki bo primeren za vožnjo po težkem terenu. Operater bo robota vodil brezžično preko kamere. V našem raziskovalnem delu bomo predstavili uporabo kamere za zaznavanje parametrov okolice, kot so zaznavanje objektov in QR-kod. Reševalni robot mora zaznati objekt in izpisati na ekranu, kaj predstavlja. Enako velja za zaznavanje QR – kode. Na tekmovanju je eden izmed izzivov zaznavanje hazmat simbolov za nevarnost, kar nameravamo vpeljati v naš sistem računalniškega vida v prihodnosti.

Slika 3: Primer reševalnega robota

(Vir: https://www.popsci.com/technology/article/2011-08/sandias-gemini-scout-rescue-robot-optimized-mining-disasters)

Slika 4: Hazmat simboli za nevarnost in QR-koda

(Osebni vir)

1.1 PREDSTAVITEV PROBLEMA

V letošnjem letu smo si zadali cilj, da bomo reševalnega robota iz preteklosti nadgradili. Pri raziskovanju smo se spopadli z veliko težavami v programskem delu. Prve težave so se pojavile pri povezovanju kamere z mikroračunalnikom, nato pri nalaganju knjižnice za kamero in programiranju ter prepoznavanju predmetov. Velik problem smo imeli tudi s pomnilnikom v krmilniku, saj nam je program, ki smo ga uporabljali za razpoznavanje okolice, zasedel veliko prostora. Vse težave smo uspeli rešiti s pomočjo raziskovanja na svetovnem spletu in lastnega znanja, ki smo ga pridobili skozi naše izobraževanje.

1.2 HIPOTEZE

Cilj naše naloge je bil, da raziščemo možnosti razpoznavanja s kamero oz. s strojnim vidom, ki ga bomo integrirali v naš reševalni robot. S pomočjo kamere bi lahko v prihodnosti dosegli avtonomnost robota, ki bi omogočal vožnjo po težkih terenih. Zato smo si zastavili nekaj hipotez, s katerimi bomo lahko potrdili ali ovrgli trditev, da je možna avtonomna vožnja s pomočjo računalniškega vida in mikroračunalnika.

Za raziskovanje smo si zastavili sledeče hipoteze:

- H1 – s kamero bomo dosegli zanesljivost in ponovljivost razpoznavanja;
- H2 – kamera bo razpoznavala ljudi;
- H3 – kamera bo razpoznala objekte;
- H4 – kamera bo dekodirala QR-kodo;
- H5 – na zaslonu se bo izpisalo, kateri objekt je bil razpoznan.

1.3 METODE RAZISKOVANJA

V postopku raziskovanja smo uporabili sledeče metode raziskovanja:

- Metodo analize, ki temelji na osnovi razčlenitve neke celote na njene osnovne sestavne dele. Na ta način smo si delo pri reševalnem robotu razdelili na tri dele: snovanje/konstruiranje, izdelava električnega vezja in programiranje. Kljub vsemu je glavni del, ki ga bomo predstavili v raziskovalni nalogi, programiranje.
- Primerjalno metodo, s katero smo si pomagali pri primerjanju različnih kamer, različnih knjižnic za programiranje itd.
- Testiranje v različnih okoliščinah reševanja, primerjava in ovrednotenje rezultatov.

2 PREDSTAVITEV POTEKA RAZISKOVALNE NALOGE

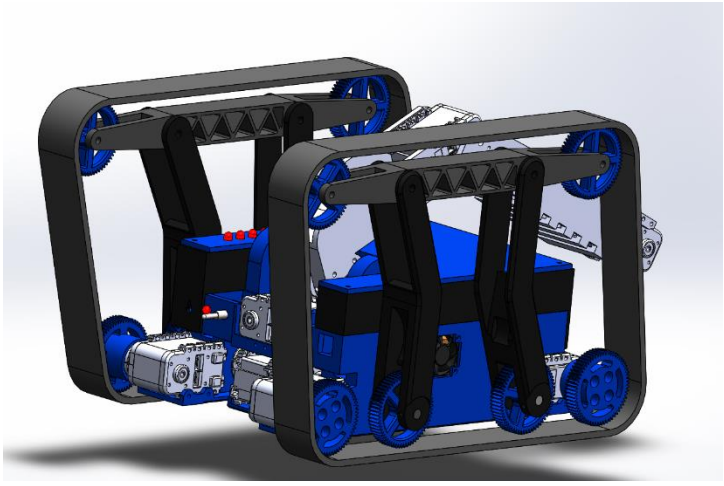
Naš glavni namen pri izdelavi raziskovalne naloge je bil, da dokažemo, da je v prihodnosti možna avtonomna vožnja robota s pomočjo kamere. Za letošnje šolsko leto smo si zadali cilj, da izdelamo reševalnega robota, ki bo lahko vozil po težkih terenih. V prihodnosti si želimo, da bi bili roboti avtonomni, kljub vsemu se nismo odločili, da bi naredili avtonomnega robota, saj se nam zaradi naše ravni znanja iz programiranja to ni zdelo mogoče. Zato smo se odločili, da bomo raziskali možnosti zaznavanja okolice in če je sploh mogoče vpeljati avtonomno vožnjo s pomočjo kamere. Kljub temu da je raziskovalna naloga namenjena predvsem programiranju (kamere), bomo tudi predstavili robota, ki ga v tem trenutku še razvijamo in izdelujemo za tekmovanje RoboCup Rescue RMRC 2019, ki bo potekalo v mesecu juliju v Sydneyju, v Avstraliji.

2.1 REŠEVALNI ROBOT

Naše delo smo si razdelili na tri dele, in sicer snovanje/konstruiranje, izdelava električnega vezja in programiranje. Vlogo programiranja sva prevzela dva člana ekipe, saj je zahtevno in je potrebno vložiti veliko dela. Veliko smo si pomagali z lanskoletno raziskovalno nalogo Hitro prototipiran terenski robot, iz katere smo črpali veliko informacij, kako poteka tekmovanje in kakšna je najboljša rešitev za vožnjo po težkih terenih. Poleg vožnje po težkih terenih mora robot uporabiti robotsko roko za dvigovanje raznih objektov, odpiranje ventilov in podobno. Zadali smo si cilj, da ima naš robot možnost hitre menjave komponent. Tako smo začeli snovati izdelek, ki bo ustrezal vsem zahtevam tekmovanja. Robot bo zelo podoben lanskoletnemu, le da bomo dodali možnost pomika obeh gosenic ločeno (v obliki paralelograma), prav tako pa tudi možnost zelo hitrega dostopa do mikroračunalnika in baterij. Da bo le-to možno, bomo namesto vijakov pri ohišju uporabili posebne ključe, ki jih je potrebno le izvleči, da pridemo do komponent. Na spodnji sliki je razvidno, kako smo si zamislili reševalnega robota, kljub temu da še ni popolnoma dokončan.

Slika 5: CAD-model reševalnega robota

(Osebni vir)



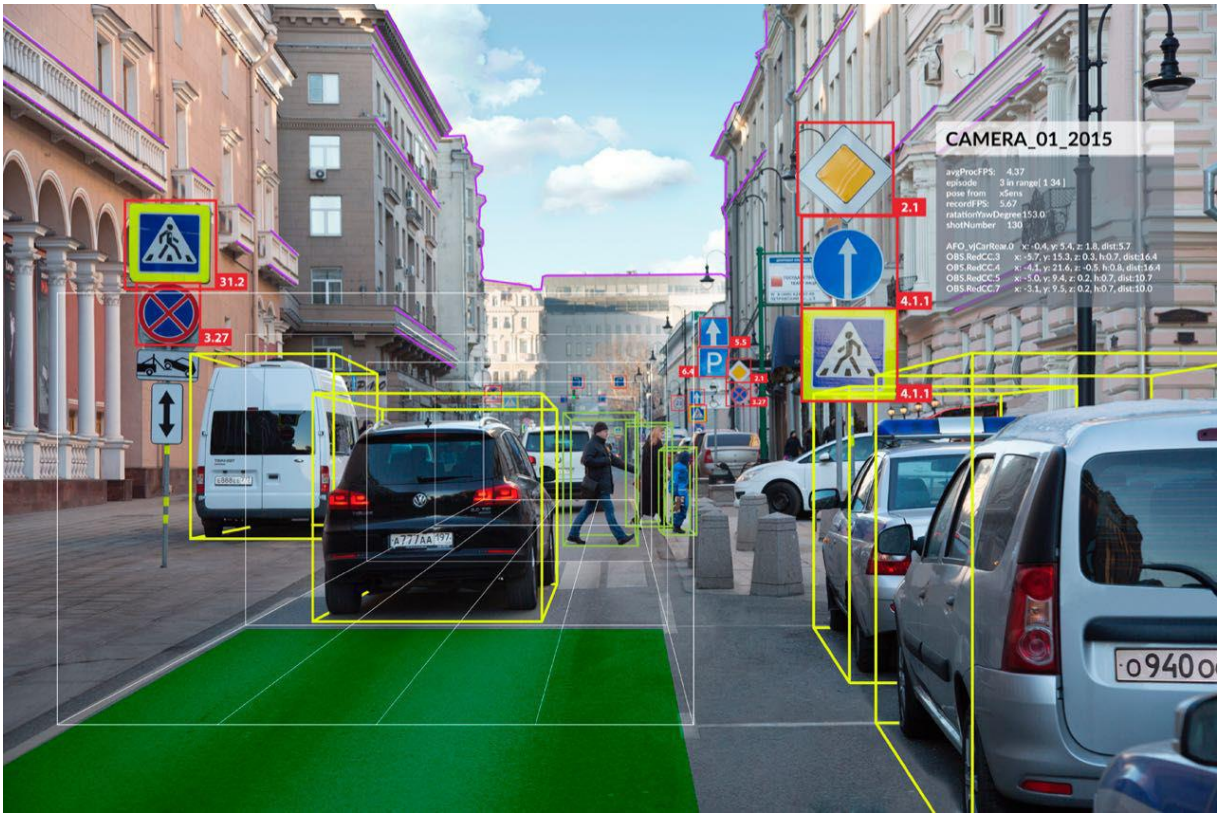
Slika 6: CAD - model reševalnega robota z gosenicami

(Osebni vir)

3 RAČUNALNIŠKI VID

Računalniški vid se ukvarja z računalniškimi sistemi, zmožnimi interpretacije in analize slikovnih podatkov. Področja uporabe računalniškega vida so: medicinska diagnostika, industrija, vojska … [3]

Največkrat srečamo računalniški vid v industriji za avtomatski nadzor in usmerjanje robotov. Uporablja se tudi za kontrolo kakovosti in sortiranje izdelkov. Dodatne aplikacije se uporabljajo tudi v prometu za enostavne merilnike hitrosti in prepoznavanja nevarnih situacij. Majhen del projektov se ukvarja z dejanskim razumevanjem slik oz. njihovih vsebin. Običajno gre le za odkrivanje predmetov na sliki. Orodja za obdelavo slik navadno izhajajo iz matematike. Inteligentne kamere so postale izjemno zmogljive in dovolj enostavne, da jih lahko inženirji vpeljejo v avtomatizirano proizvodno linijo. Z računalniškim vidom lahko v proizvodnji dosežemo večjo produktivnost, zvišanje kakovosti izdelkov in zmanjšamo stroške.

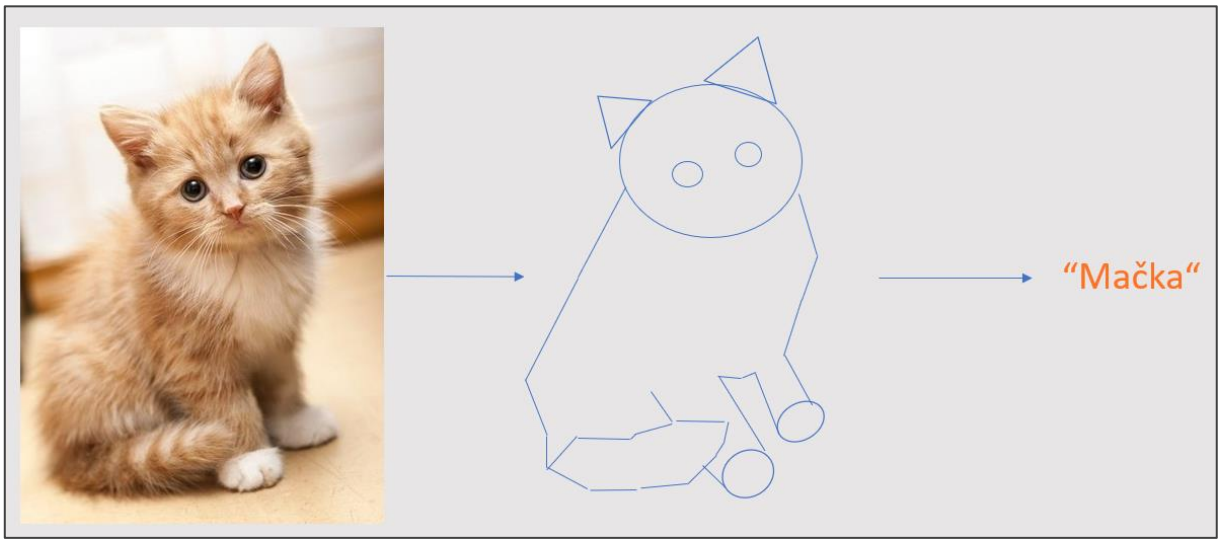


Slika 7: Računalniški vid

(Vir: http://yonah.org/channel/visionhack-computer-vision-russia/)

Pojem računalniškega vida se uporablja takrat, ko računalnik izvaja odločanja na osnovi vizualnih informacij, zapisanih v digitalni obliki. Ker gre pri razumevanju slike navadno za uporabo računalniških metod, povezanih s fotogrametrijo, lahko govorimo o umetni inteligenci. V bolj zapletenih nalogah se uporabljajo modeli. Ti vsebujejo vse potrebne informacije za identifikacijo objekta. [4]

Na spodnji sliki je prikazan primer zaznavanja mačke. Model ima sprogramirano, kje se nahajajo ušesa, oči in kakšna je oblika mačke.



Slika 8: Način delovanja računalniškega vida

(Osebni vir)

Pri zaznavanju se 3D-slika preko optičnega sistema projicira na 2D-sliko. Taka slika vsebuje milijone bitov informacij in naloga računalniškega vida je, da iz množice informacij izlušči le potrebne podatke. [5]

Slika 9: Težave pri računalniškem vidu

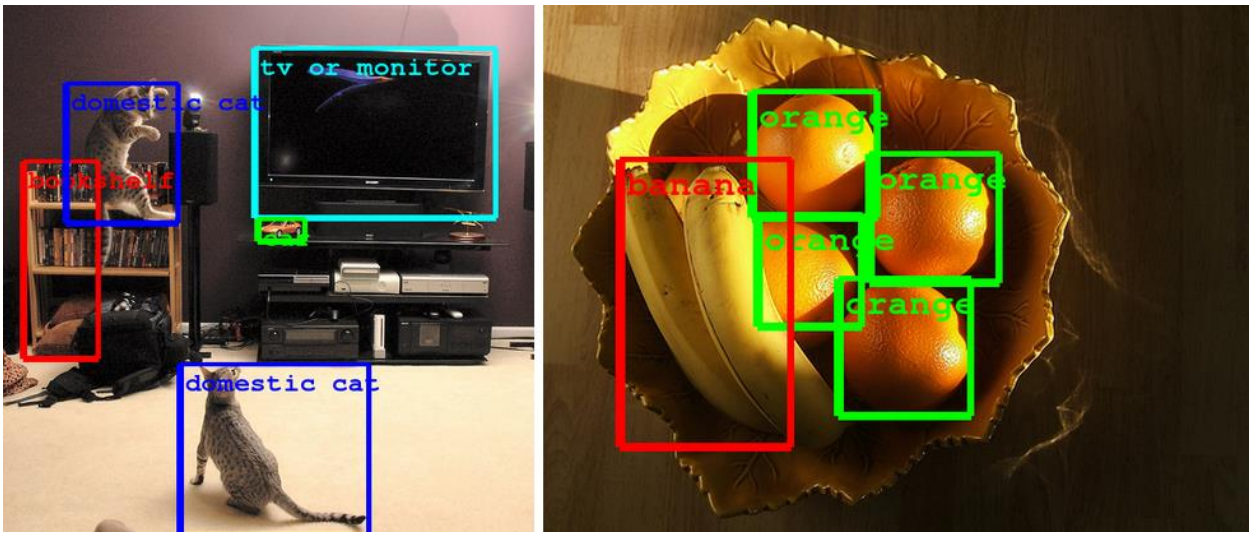
(Vir: https://www.pinterest.com/pin/474707616947833730/)

Slika 10: Težave pri računalniškem vidu

(Vir: https://unsplash.com/search/photos/cats)

Težave nastanejo v primeru, da je mačka v drugačni poziciji kot na sliki 8. Programu smo povedali, da je objekt s takšno obliko mačka. V primeru, da je mačka v drugačni poziciji, je program ne bo zaznal. Da bi omogočili zaznavanje mačk v vseh pozicijah, bi morali ustvariti več modelov.

Percepcijo pri človeku opravljajo možgani, pri računalniškem vidu pa programska oprema in tukaj nastane težava. Stopnja razvoja računalniškega vida je v primerjavi s človekom na začetni ravni. V prihodnosti bo računalniški vid postal čedalje večja in dobra zamenjava za človeško percepcijo. To bo posledica nenehnega razvoja tehnologije in algoritmov odločanja. Sistemi računalniškega vida se uporabljajo le kot preprost nadomestek človekovega vizualnega zaznavanja. [6]



Slika 11: "Preprosta" uporaba računalniškega vida

(Vir:http://answers.opencv.org/question/59391/multiple-objects-classification/)

4 UPORABNOST RAČUNALNIŠKEGA VIDA

4.1 AVTONOMNA VOŽNJA AVTOMOBILOV

Vsak dan smo bližje avtonomni prihodnosti, v kateri nas bodo prevažali avtomobili. A preden bomo popolnoma predali nadzor nad avtomobilom, bodo le-ti morali napredovati do pete stopnje avtonomnosti. Vse več proizvajalcev vozil pri predstavljanju modelov omenja stopnjo avtonomije, ki jih dosegajo. [7]

4.1.1 1. stopnja: Voznik ima popoln nadzor

Pri osnovni stopnji se uporabljajo podatki senzorjev ali kamer, toda voznik ima še vedno popolni nadzor nad vozilom in skrbi za vse elemente vožnje. Varnostni elementi vozniku olajšajo delo. [7]

Slika 12: Prva stopnja avtonomne vožnje

(Vir:https://siol.net/avtomoto/zgodbe/to-je-pet-stopenj-avtomobilske-samostojne-voznje-457861)

4.1.2 2. stopnja: Avtomobil prevzame nadzor v kritičnih situacijah

Tukaj računalnik prevzema veliko funkcij in vozniku olajša vožnjo oz. odvzema še več bremena. Celoten sistem je zmožen istočasno prilagajati hitrost in črpati podatke z več koncev. Omogoča dobro prilagajanje smeri, hitrosti in zaviranja. S pomočjo navigacije in številnih senzorjev samodejno zavira pred zavoji.

Pri drugi stopnji vozniku pomaga več sistemov, a je voznik še vedno glavni upravljalec avtomobila. [7]

Slika 13: Druga stopnja avtonomne vožnje

(Vir:https://siol.net/avtomoto/zgodbe/to-je-pet-stopenj-avtomobilske-samostojne-voznje-457861)

4.1.3 3. stopnja: Voznik mora biti še vedno prisoten in nadzoruje vožnjo

Avtomobilski proizvajalci se zelo hitro približujejo tretji stopnji avtonomije. Tretja stopnja je poimenovana tudi kot pogojna avtomatizacija – avtomobilu dovoli, da vožnjo v celoti opravlja namesto voznika, a mora biti voznik še vedno prisoten in ukrepati v kriznih situacijah.

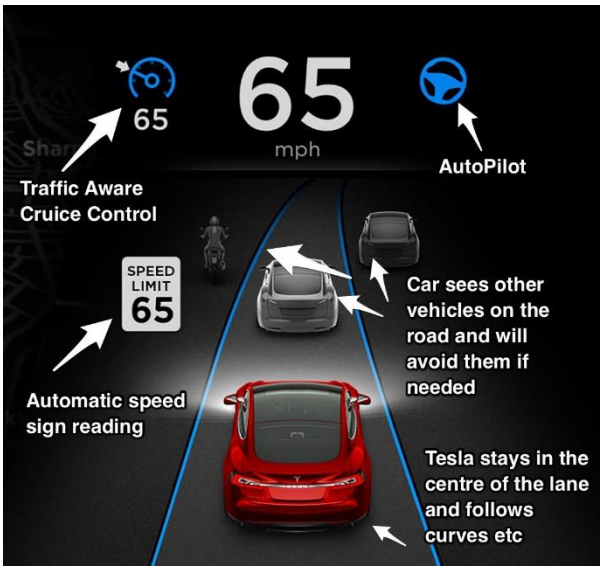
Avtomobili, ki so zmožni tretje stopnje avtonomije za samodejno vožnjo, preučujejo zemljevid, uporabljajo podatke s kamere (računalniški vid), radarja in senzorjev ter vse podatke procesirajo z zelo hitrim procesorjem.

Kljub samodejni vožnji mora biti voznik pozoren in pripravljen, da prevzame nadzor nad vozilom. [7]

Slika 14: Tretja stopnja avtonomne vožnje

(Vir:https://siol.net/avtomoto/zgodbe/to-je-pet-stopenj-avtomobilske-samostojne-voznje-457861)

Tesla je proizvajalec avtomobilov, ki se je močno približal tretji stopnji avtonomnosti. Tehnologija še ni razvita v celoti, vseeno ponuja sistem Autopilot, ki do neke mere prevzame nadzor nad vozilom. Kljub vsemu ta sistem ni povsem varen, saj še vedno ni sposoben prevzeti 100-odstotnega varnega nadzora nad avtomobilom. Kljub avtopilotu mora biti voznik še vedno prisoten pri vožnji.



Slika 15: Tesla – Autopilot

(Vir:https://fealey.co.uk/tesla/differences-between-tesla-autopilot-1-and-autopilot-2/839)

Nesreča s Teslovim avtomobilom se je zgodila v mestu Culver v zvezni državi Kalifornija, kjer je bil v prometni nesreči udeležen avtomobil, za katerega sumijo, da je bil v času trka nastavljen na vožnjo v avtonomnem načinu. Do nesreče je prišlo, ko je vozilo zavilo na pas, rezerviran za intervencijska vozila in pri hitrosti 105 km/h trčilo v zadnji del gasilskega vozila. [8]

Slika 16: Nesreča s Teslovim avtomobilom

(Vir: https://www.instagram.com/p/BeQ35kcllpR/?utm_source=ig_embed)

To dokazuje, da smo še daleč od pete stopnje avtonomne vožnje, saj še tretja stopnja ni popolnoma varna in še vedno potrebuje voznika, da ukrepa v kriznih primerih. Kljub vsemu bo v prihodnosti možna vožnja brez voznika zaradi nenehnega razvijanja algoritmov računalniškega vida.

4.1.4 4. stopnja: Popolna avtonomnost v mestih

Prvi avtomobili, ki bodo omogočali četrto stopnjo avtonomnosti, bodo po napovedih na ceste zapeljali v začetku prihodnjega desetletja. Vozili se bodo lahko po območjih, ki so zelo dobro pokrita z natančnimi podatki zemljevidov. Njihovo delovanje bo izboljšala komunikacija med avtomobili in klicnimi centri. S tem se bo močno povečala avtonomnost avtomobilov.

V četrti stopnji voznik ne bo več potreben. Takšni avtomobili bodo primerni predvsem za vožnjo po mestih, kjer je pokritost zemljevidov izjemno natančna. [7]

Slika 17: Četrta stopnja avtonomne vožnje

(Vir:https://siol.net/avtomoto/zgodbe/to-je-pet-stopenj-avtomobilske-samostojne-voznje-457861)

4.1.5 5. stopnja: Voznik postane potnik

Avtomobil pete stopnje ne potrebuje več elementov, ki jih je do zdaj potreboval voznik za upravljanje vozila. Avtomobil pete stopnje avtonomnosti se bo lahko vozil kjerkoli, ne samo v mestih, kjer so črte na cestah natančno določene in okolje v celoti nadzorovano. Avtomobil bo znal predvidevati situacije in slediti cestam, kjer infrastruktura ni popolna. V notranjosti ne bo več volana, voznik bo postal potnik. [7]

Slika 18: Peta stopnja avtonomne vožnje

(Vir:https://siol.net/avtomoto/zgodbe/to-je-pet-stopenj-avtomobilske-samostojne-voznje-457861)

4.2 ZABAVNA INDUSTRIJA

Tudi zabavna industrija uporablja računalniški vid. Na voljo je veliko programskih orodij, ki s slike oz. videa izluščijo podatke o prisotnosti ljudi v prostoru.

Primer uporabe računalniškega vida v zabavni industriji je Microsoft Kinect (med razvojem pod imenom projekt Microsoft Natal), ki je nov način igranja iger brez igralnih palic. Temelji na razpoznavi gibanja in glasu s pomočjo kamere ter ustrezno programsko opremo. Avatarje v igri krmilimo s premikanjem telesa. [6]

Slika 19: Microsoft Natal

(Vir: https://vimeopro.com/user39291740/franco-marinelli-kids-reel/video/147970036)

Za kamero se uporablja senzor Kinect, ki je igralni pripomoček za Microsoftovo konzolo Xbox 360. Njegov namen je, da igralec podaja ukaze z gibi telesa in glasom. Vse bolj je priljubljen pri razvijalcih robotike in računalniškega vida.

Slika 20: Xbox Kinect

(Vir: http://eprints.fri.uni-lj.si/1900/1/Burnar_S-1.pdf)

Ena bistvenih komponent je poleg kamere in senzorjev programska oprema, ki obdela podatke, ki jih prejme iz strojne opreme. Sistem zna zelo hitro prepoznati igralce po obrazu. [9]

4.3 INDUSTRIJA

Računalniški vid dobiva vedno večjo vlogo v industriji pri vizualnem ugotavljanju kakovosti proizvodov, ki jo je do sedaj opravljal človek. Prednost uporabe računalniškega vida v industriji je, da je pri preverjanju kakovosti natančnejši kakor človek, kar pomeni, da je veliko zanesljivejši. Sisteme, ki uporabljajo računalniški vid oz. strojni vid (izraz, ki se uporablja v industriji), najdemo tudi v robotskih aplikacijah. Strojni vid v proizvodnji poveča produktivnost, izboljša kakovost proizvodov in zniža stroške. Področja uporabe strojnega vida v industriji so: lociranje (določitev položaja), razvrščanje, ocenjevanje, pregledovanje in določanje premikanja ter vodenje in nadzor.

Najbolj razširjeno področje uporabe strojnega vida je kontrola kakovosti, ki obsega kontrolo površin in merilno tehniko. Glavna prednost strojnega vida je, da je merjenje brezkontaktno in zelo hitro. [10]

Slika 21: Strojni vid

(Vir: http://www.nojp.net/strojni_vid.html)

5 STROJNE KOMPONENTE

Med strojne komponente računalniškega vida štejemo vse naprave, ki omogočajo zajetje in obdelavo slike, kot tudi naprave, na katerih je nameščena programska oprema.

Med te komponente štejemo:

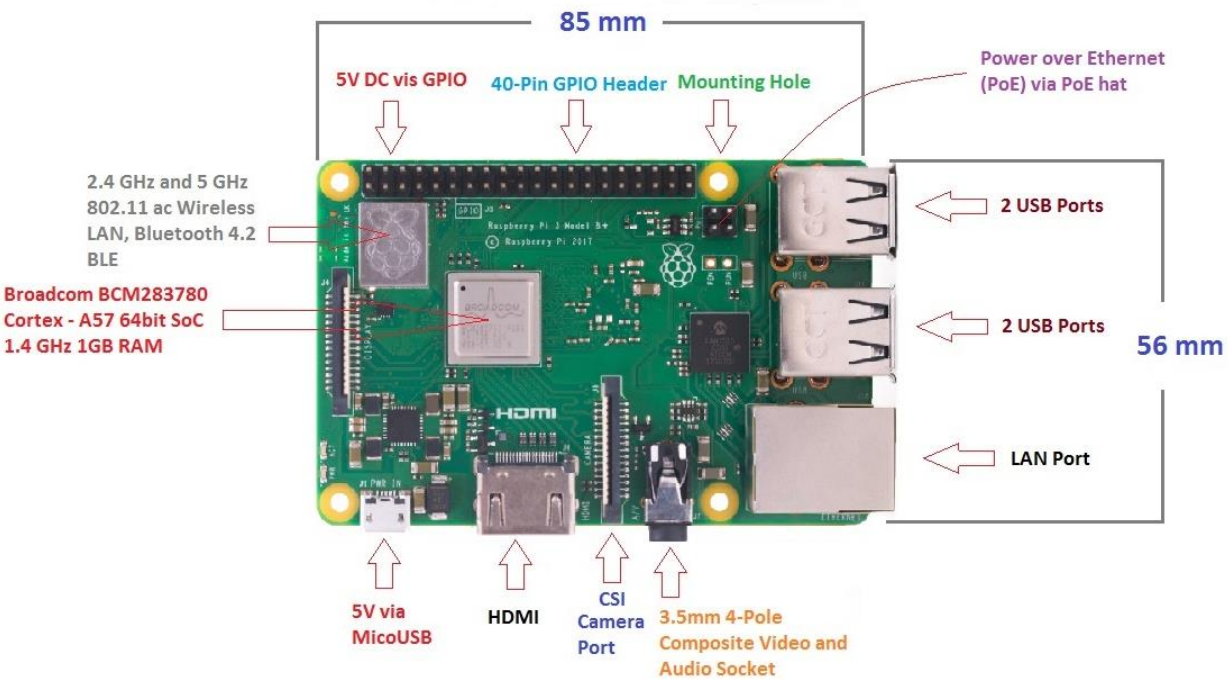
- kamere (barvne, črno-bele)
- analogno digitalni pretvorniki (naprave za digitalizacijo slike)
- računalnik oz. mikroračunalnik s programsko opremo

Pri računalniškem vidu je zelo pomembno, da imamo kakovostno kamero, s katero zajemamo sliko, saj s slabe slike ne moremo zajeti natančnih podatkov. Kvaliteta slike je odvisna od mnogih faktorjev, kot so odboj svetlobe, zmanjšana vidljivost zaradi neugodnih vremenskih razmer in premajhne svetlobe ponoči. Najboljše rezultate dosežemo v zaprtih prostorih, kjer neugodnih vremenskih razmer ni in kjer lahko poskrbimo za dobro osvetlitev. Težave nastanejo, ko želimo kamero uporabljati na prostem. Za robotsko reševanje s pomočjo računalniškega vida je potreben stabilen sistem, ki bo deloval v vseh vremenskih pogojih.

Pri kameri je zelo pomemba izbira ločljivosti in barvne globine. Kvaliteta slike je tudi zelo odvisna od izbire strojne opreme. [5]

5.1 MIKRORAČUNALNIK

Za izvršitev naše raziskovalne naloge smo si izbrali Raspberry Pi. Raspberry Pi je mikroračunalnik v velikosti kreditne kartice. Do sedaj sta bili izdani dve verziji, in sicer model A in model B. Model A nima Ethernet priključka in ima samo en USB-priključek. Za delovanje porabi 300 mA toka in je cenejši od Modela B. V naši raziskovalni nalogi smo uporabili Raspberry Pi 3 Model B+. Ta mikroračunalnik smo si izbrali predvsem zaradi večjega števila USB-priključkov, prav tako pa ima priključek za kamero, ki je bil potreben za izdelavo naše raziskovalne naloge. Raspberry Pi ima tudi wifi modul, tako da se lahko nanj povežemo na daljavo. Za delovanje uporablja operacijski sistem Raspbian (različica Debiana, Linux).



Slika 22: Raspberry Pi 3B+

(Vir:https://www.theengineeringprojects.com/2018/07/introduction-to-raspberry-pi-3-b-plus.html)

Na mikroračunalniku imamo tudi GPIO (General Purpose Input/Output) vmesnik, ki omogoča branje vhodov in krmiljenje raznih izhodov, kot so led-diode, tranzistorji …

GPIO-pini omogočajo upravljanje naprave preko protokolov I2C (vodilo za prenos podatkov), UART (strojna oprema, ki pretvarja vzporedne signale v zaporedne) in SPI (sinhrono serijsko vodilo).

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I2C)		DC Power 5v	04
05	GPIO03 (SCL1 , I2C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Slika 23: GPIO (General Purpose Input/Output) pinout

(Vir:http://www.raspberry-pi-geek.com/howto/GPIO-Pinout-Rasp-Pi-1-Model-B-Rasp-Pi-2-Model-B)

Pri priklapljanju izhodov in vhodov na mikroračunalnik moramo biti pozorni na zgornjo shemo, kjer je prikazana razporeditev GPIO-pinov. Pri uporabi je pomembno, da ne naredimo kratek stik, saj v tem primeru Raspberry Pi ne bo več deloval pravilno. Med izdelavo raziskovalne naloge se nam je zgodilo, da je programska oprema delovala pravilno, le da nismo mogli vklapljati in izklapljati izhodov kljub ukazu mikroračunalniku.

5.1.1 Napajanje

Napajanje mikroračunalnika je možno preko standardnega 5 V MicroUSB-priključka, ki dobi napajanje iz računalniškega USB-izhoda ali preko USB-vmesnika iz hišne napeljave. Za normalno delovanje je potrebno vsaj 700 mA toka.

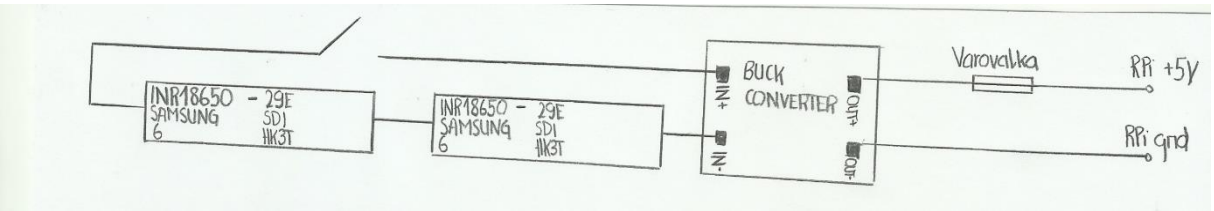
Slika 24: Napajalnik

(Osebni vir)

Slika 25: Priključitev napajalnika na mikroračunalnik

(Osebni vir)

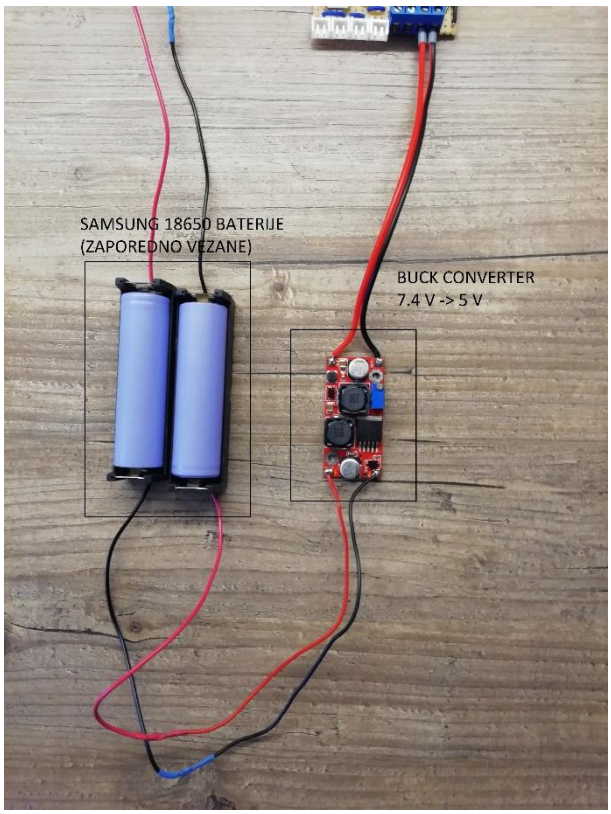
Raspberry Pi lahko napajamo tudi preko GPIO (splošno namenski vhodno-izhodni vmesnik) vhodov. Na pin 2 priključimo pozitivno napetost 5 V in na pin 6 ničli potencial (GND). V našem reševalnem robotu bomo uporabili način napajanja preko GPIO-vhodov. Na spodnjih slikah je prikazano, na kakšen način bomo priključili baterije oz. napetost na mikroračunalnik.



Slika 26: Shema napajanja

(Osebni vir)

Za napajanje uporabljamo dve Samsungovi bateriji 18650, ki ju vežemo zaporedno, da dosežemo napetost 7.4 V (ena baterija ima napetost 3.7 V). Nato s pomočjo buck converterja znižamo napajalno napetost na 5 V. Ker mora biti (teoretično) moč vhoda enaka moči izhoda, se nam poveča tok, saj znižamo napetost.

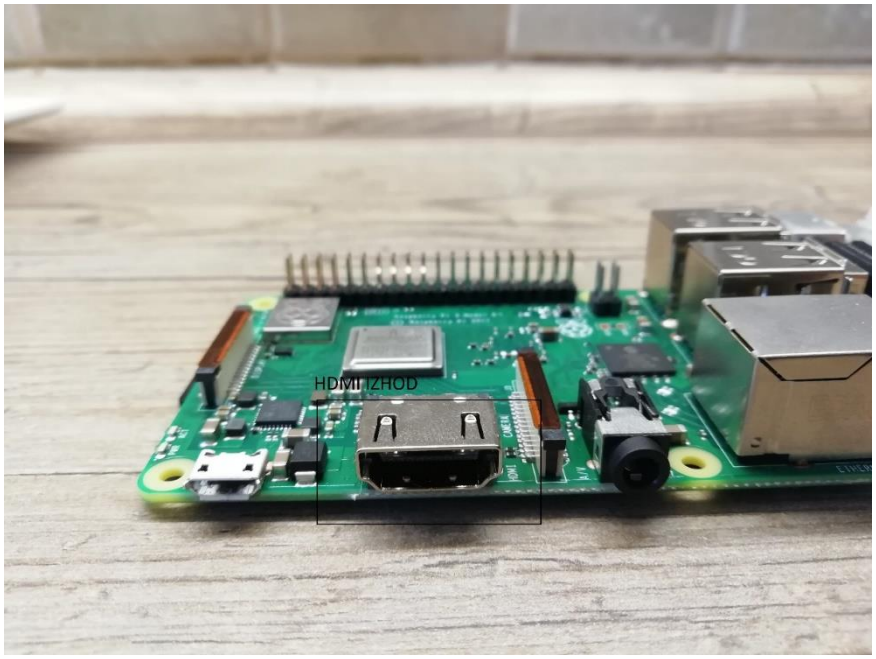


Slika 27: Realizirano napajanje

(Osebni vir)

5.1.2 Priklop zaslona na mikroračunalnik

Zaslon priklopimo s HDMI- kablom na HDMI, kompozitni video izhod na Raspberry Piju.



Slika 28: HDMI izhod

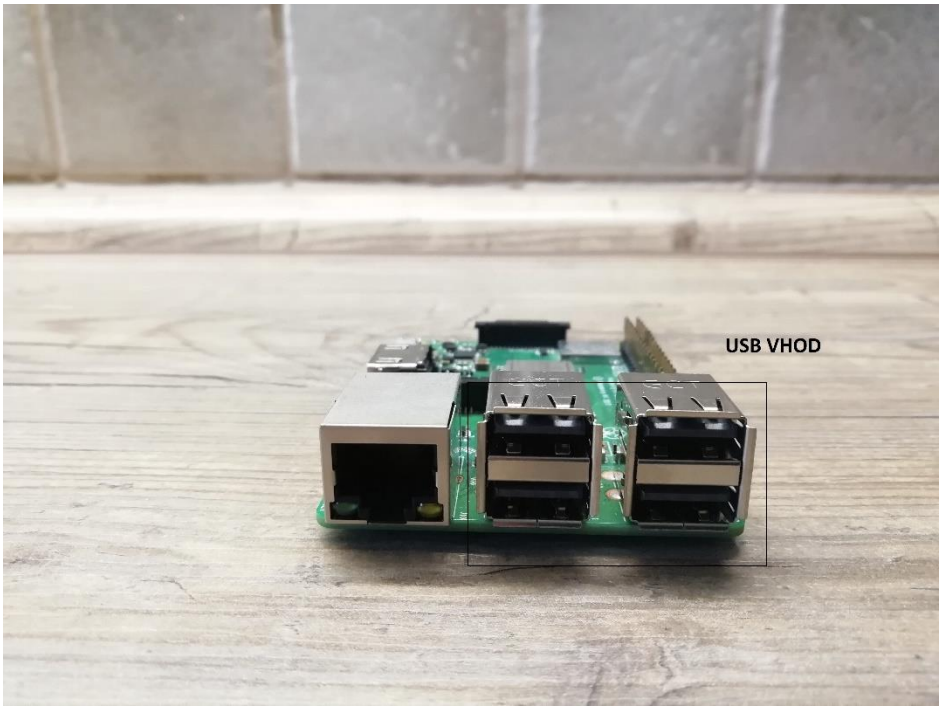
(Osebni vir)

Slika 29: Priklop HDMI-kabla

(Osebni vir)

5.1.3 Zagon mikroračunalnika

Ko priključimo napajanje in povežemo zaslon s HDMI-kablom, moramo povezati še miško in tipkovnico, da lahko uporabljamo Raspberry Pi. Miško in tipkovnico povežemo na USB-vhode na mikroračunalniku.



Slika 30: USB-vhod na mikroračunalniku

(Osebni vir)

Slika 31: Priklop miške in tipkovnice na mikroračunalnik

(Osebni vir)

Pred zagonom moramo na SD-kartico naložiti operacijski sistem. Operacijski sistem smo prenesli z uradne spletne strani. Za namestitev operacijskega sistema na SD-kartico potrebujemo računalnik z okoljem Windows.

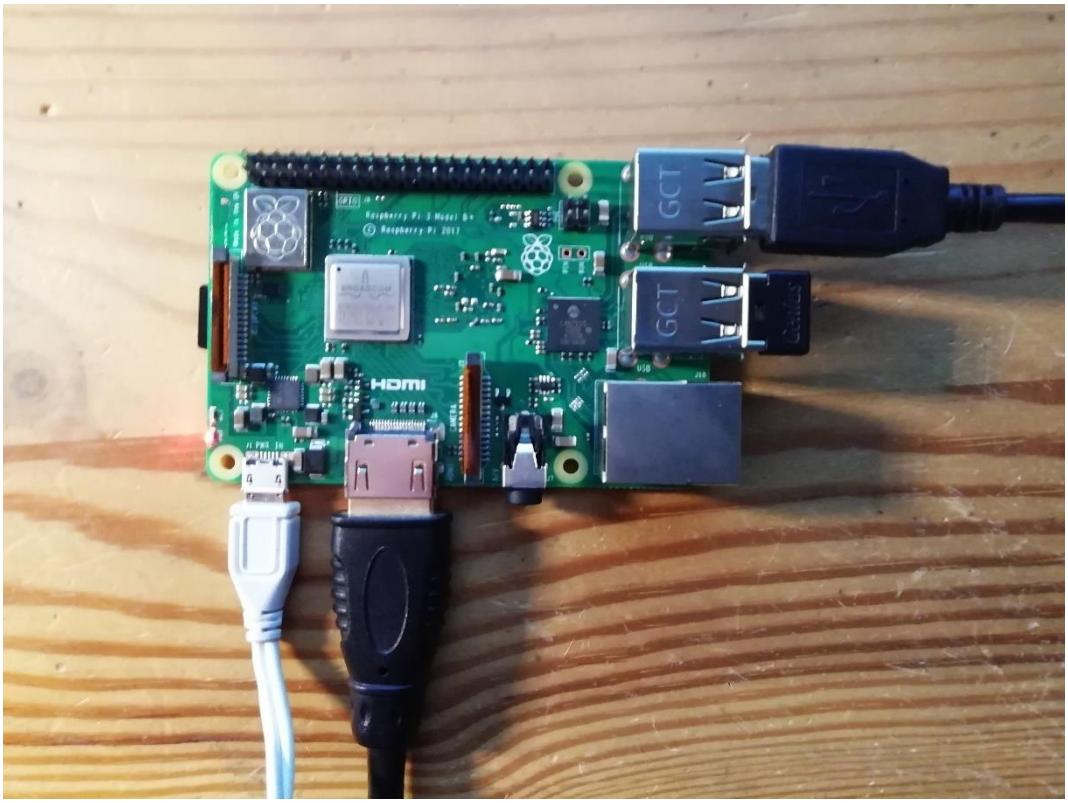
Slika 32: SD-kartica

(Osebni vir)

Slika 33: Priklop SD-kartice v mikroračunalnik

(Osebni vir)

Po prenosu operacijskega sistema na SD-kartico moramo le-to namestiti v mikroračunalnik. Nato priklopimo še miško, tipkovnico, zaslon in ko želimo zagnati Raspberry Pi, priključimo še napajanje.

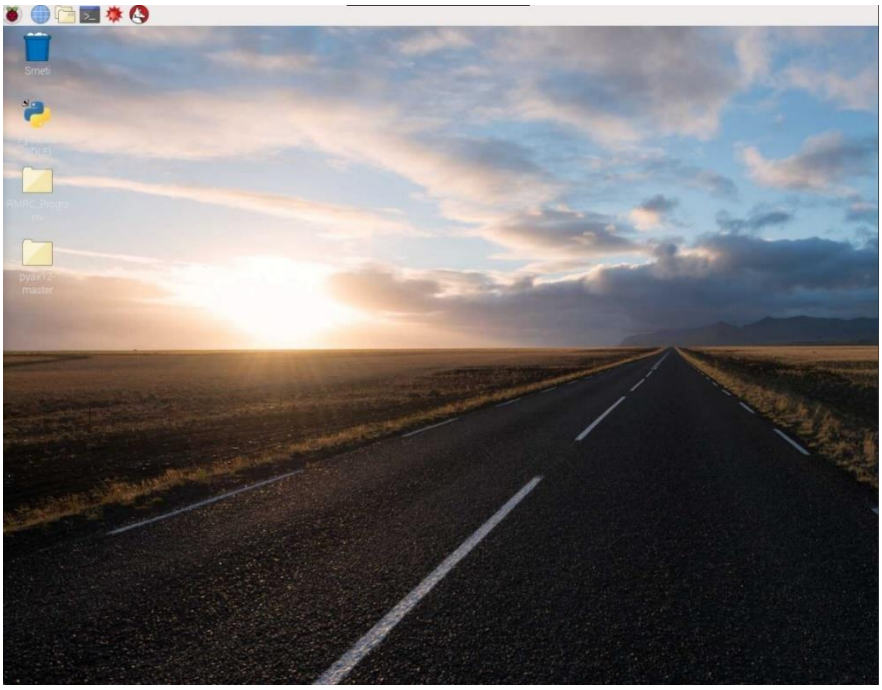


Slika 34: Priklop vseh vhodov in izhodov

(Osebni vir)

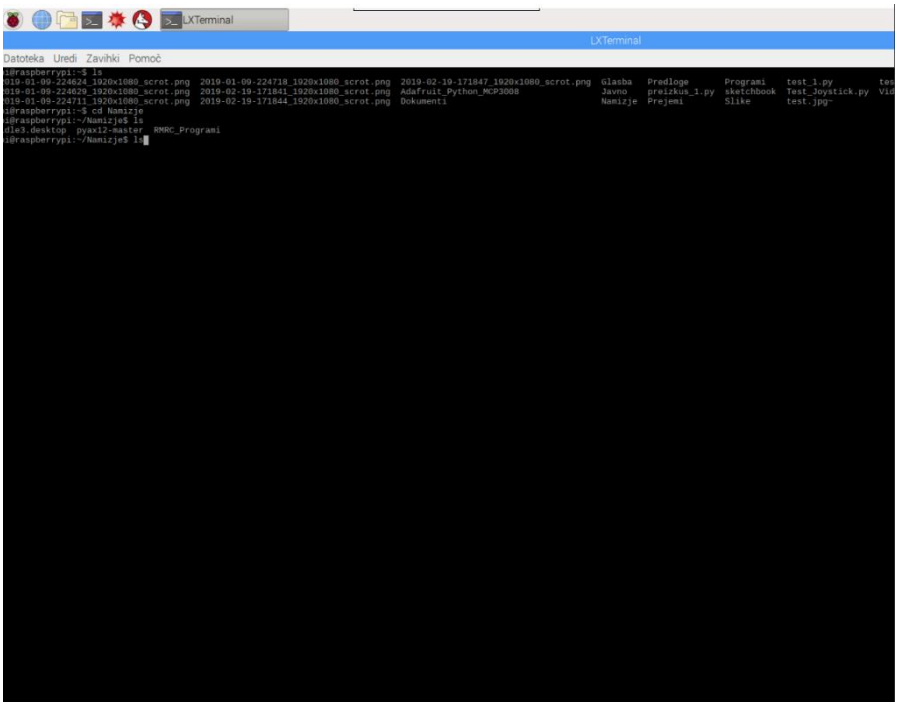
Ko priključimo napajanje, se mikroračunalnik vklopi. Po konfiguraciji preidemo na grafični način upravljanja Raspberry Pija. S tem lahko začnemo uporabljati sistem, kot bi uporabljali navadni računalnik. Lahko brskamo po spletu, programiramo …

Raspberry Pi omogoča način upravljanja preko terminala. V tem načinu lahko prav tako programiramo in delamo enako kot v grafičnem načinu, le da je težje za uporabo.



Slika 35: Grafični način

(Osebni vir)



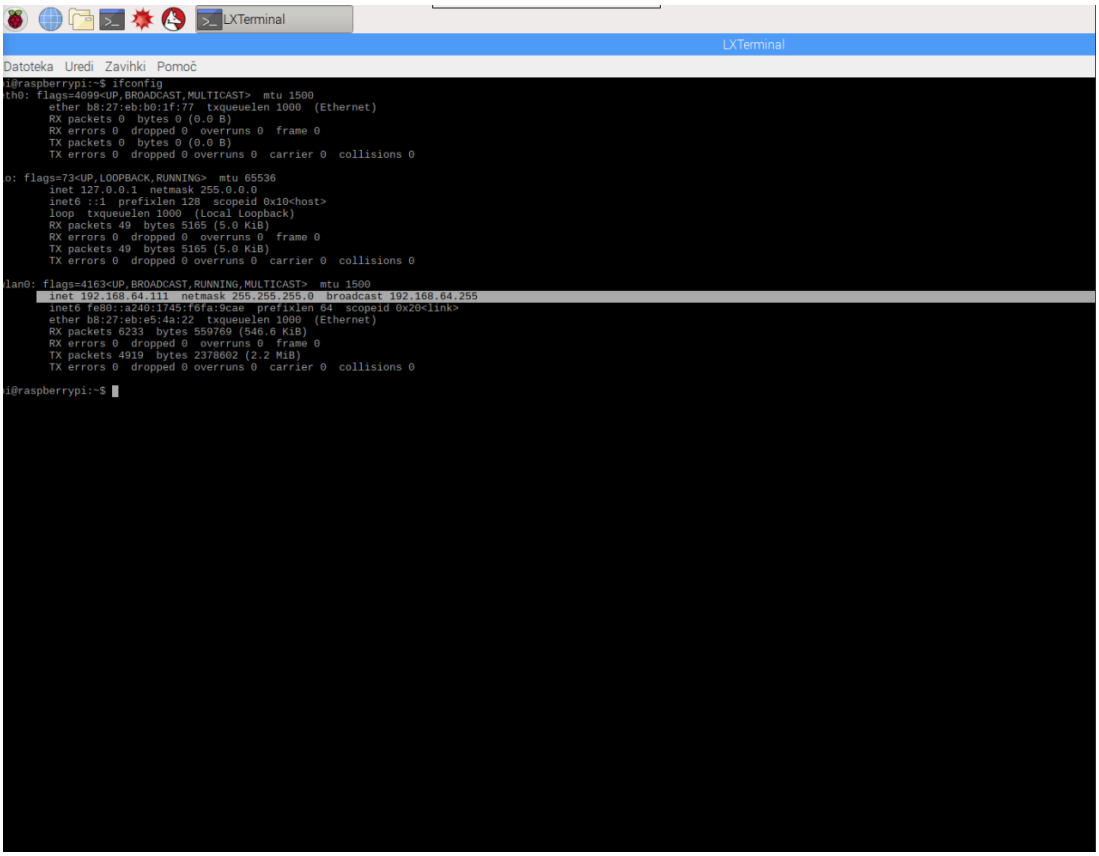
Slika 36: Terminal

(Osebni vir)

5.1.4 Oddaljen dostop

Oddaljen dostop je zelo uporaben, saj nam ni potrebno priključevati vse potrebne periferije (miško, tipkovnico, zaslon). Oddaljen dostop lahko omogočimo le z napajanjem in omrežjem.

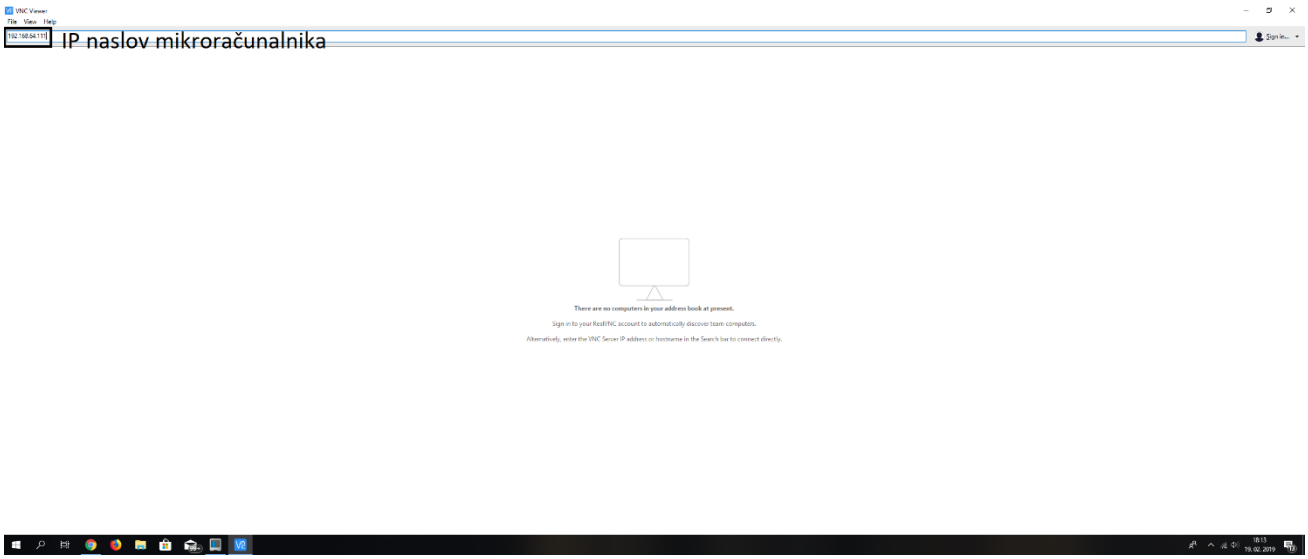
Najprej moramo prenesti program VNC Viewer na naš računalnik (namizni ali prenosni). VNC omogoča grafično deljenje namizja preko računalnika. Nato vpišemo v terminal mikroračunalnika »ifconfig« in prikaže se nam IP-naslov Raspberry Pija, kot je prikazano na spodnji sliki.



Slika 37: Iskanje IP-naslova

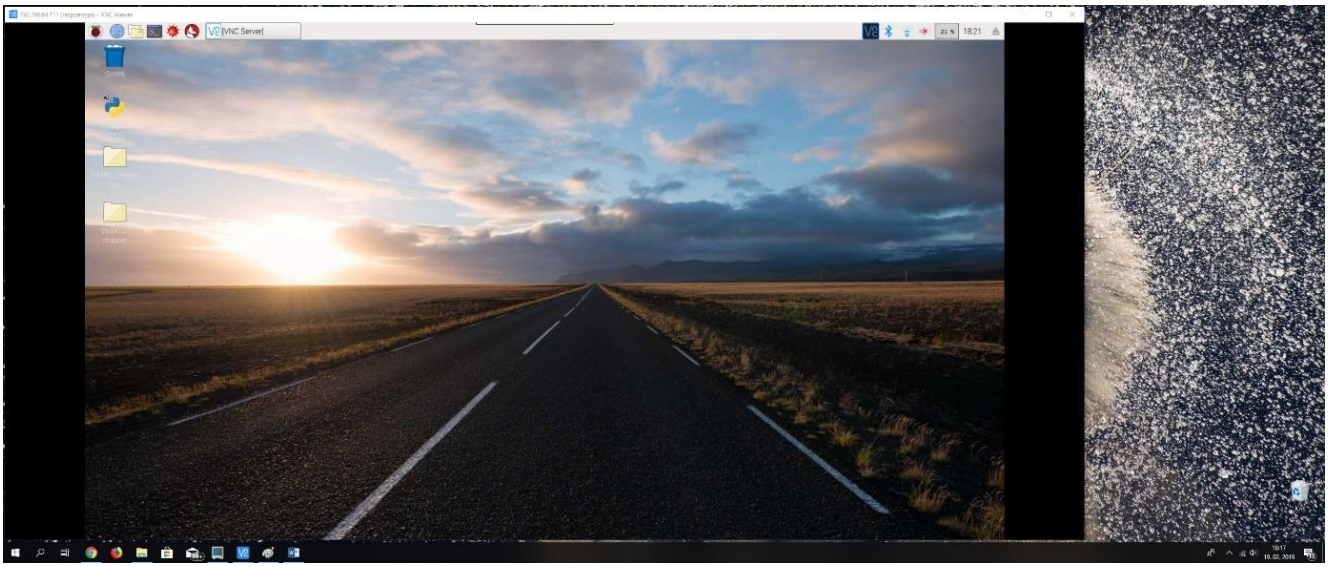
(Osebni vir)

IP-naslov mikroračunalnika vpišemo v VNC viewer, kar nam omogoči oddaljen dostop do namizja mikroračunalnika.



Slika 38: VNC viewer

(Osebni vir)



Slika 39: Oddaljen dostop do mikroračunalnika

(Osebni vir)

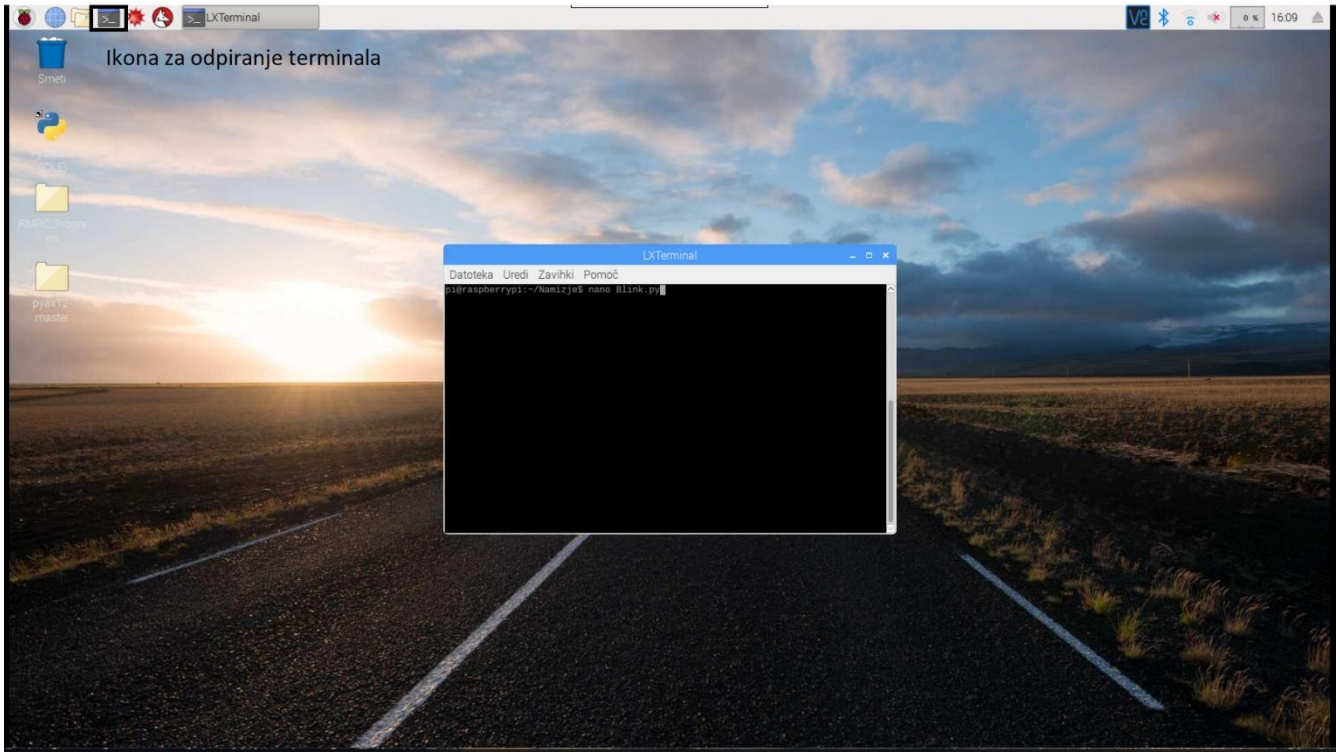
5.1.5 Programiranje mikroračunalnika

Mikroračunalnik se programira v določenem programskem okolju, ki prevede programski jezik v strojni jezik, ki ga naprava, ki jo programiramo, razume. Python je osnovni jezik, ki ga podpira mikroračunalnik Raspberry Pi, zato se v nadaljevanju raziskovalna naloga osredotoča predvsem nanj. Python je programski jezik, ki je preprost za začetnika, a hkrati zelo zmogljiv. Je berljiv in omogoča zelo hitro učenje. Uporabljamo lahko dve verziji Pythona, in sicer Python 2 in Python 3. Različica Python 3 je bila objavljena že leta 2008, ampak je bila z moduli (knjižicami), ki nam olajšajo programiranje, slabo podprta. Iz tega razloga je veliko uporabnikov še vedno zagovornikov različice Python 2, ki je zelo podprta. Kljub vsemu smo za izdelavo naše raziskovalne naloge uporabili najnovejšo različico programskega okolja Python. [11]

Mikroračunalnik omogoča, da lahko Python programiramo v grafičnem načinu ali v terminalu. Na spodnjih slikah je prikazan primer programa v grafičnem načinu upravljanja mikroračunalnika in v terminalu. Program ukaže krmilniku, naj LED-diodo vklaplja in izklaplja v časovnem intervalu 1 sekunde.

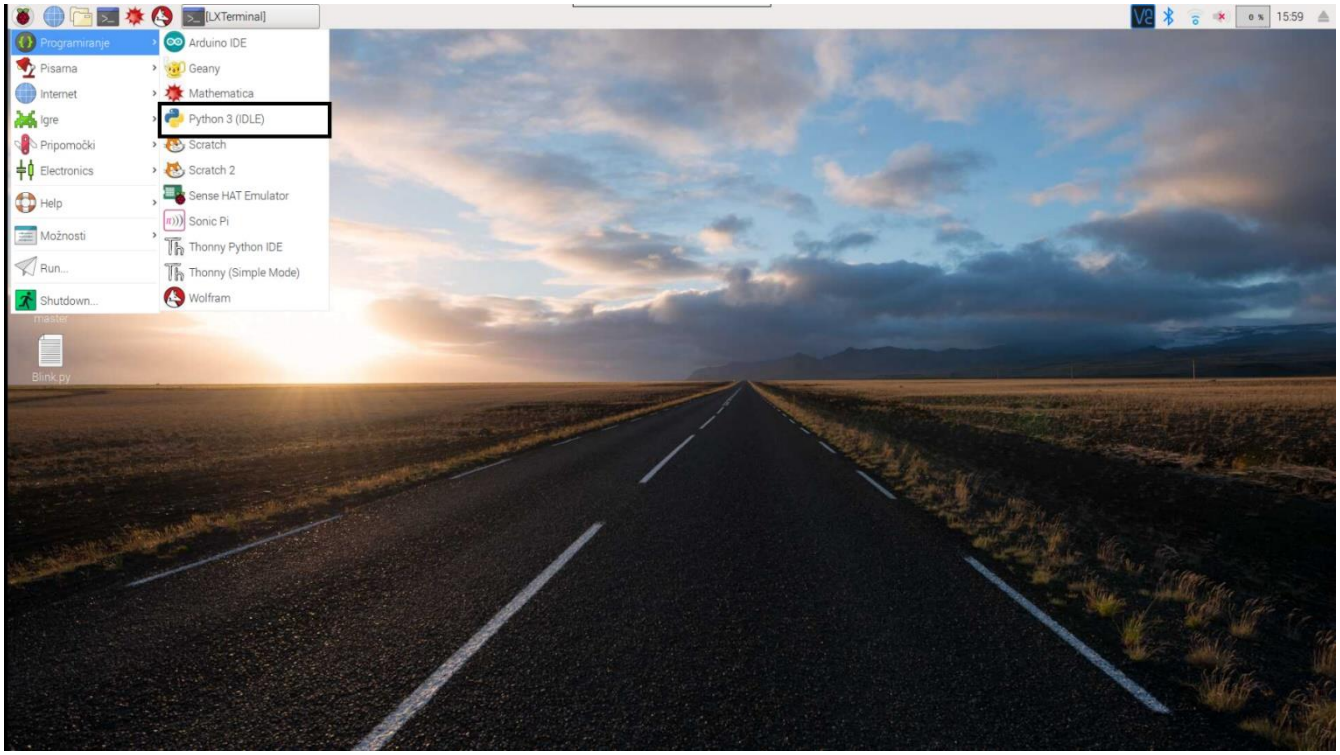
Slika 40: Logotip programskega okolja Python

(Vir: https://www.fbk.eu/en/event/euroscipy-2018/python-logo/)



Slika 41: Dostopanje do programskega okolja Python preko terminala

(Osebni vir)



Slika 42: Kako dostopati do programskega okolja Python v grafičnem načinu

(Osebni vir)

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BCM)

#Pin 21 dolocimo kot izhod
GPIO.setup(21, GPIO.OUT, initial = 0)

try:
    while(1 == 1): #Neskoncna zanka
        GPIO.output(21, 0)
        sleep(1)
        GPIO.output(21, 1)
        sleep(1)

except KeyboardInterrupt:
    GPIO.cleanup()
```

Slika 43: Programiranje v grafičnem načinu upravljanja z mikroračunalnikom

(Osebni vir)

```
 GNU nano 2.7.4

import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BCM)

#Pin 21 dolocimo kot izhod
GPIO.setup(21, GPIO.OUT, initial = 0)

try:
    while(1 == 1): #Neskoncna zanka
        GPIO.output(21, 0)
        sleep(1)
        GPIO.output(21, 1)
        sleep(1)

except KeyboardInterrupt:
    GPIO.cleanup()
```

Slika 44: Programiranje preko terminala

(Osebni vir)

5.2 KAMERA

5.2.1 Priključitev kamere

Izbrali smo kamero, ki je namenjena mikroračunalniku Raspberry Pi, in sicer RPI WWCAM2.

Slika 45: RPI WWCAM2

(Vir: https://m.reichelt.com/de/en/raspberry-pi-camera-5mp-wide-angle-1080p-rpi-wwcam2-p219885.html?fbclid=IwAR1_JfNoDUGa5P7H_FdBO_6igwXFzxRZLp5pXIK48i_n0ny-MyGNw19-f6s)

HD-Wide-Angle-Camera for Raspberry Pi with an viewing angle of 200°

Specifications

- Chip: OV5647
- Resolution: 5 megapixels
- Photo-Resolution: 2952 x 1944
- Viewing Angle: 200°
- Focal Length: 0.87 mm
- Sensor: CMOS
- Aperture: 2.0
- Cable Length: 16 cm
- Dimensions: 25 x 24 x 16,5 mm

Scope of Delivery

5MP Wide-Angle Camera, Ribbon Cable

The camera can be connected via the CSI interface of your Raspberry Pi.

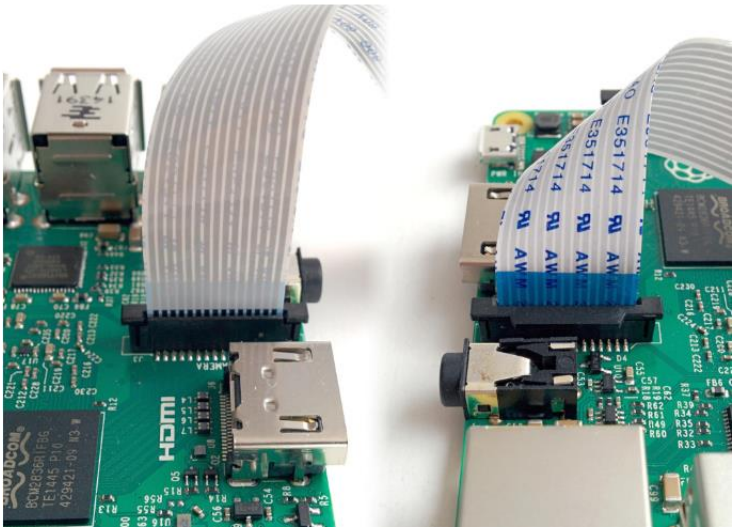
Slika 46: Specifikacije kamere RPI WWCAM2

(Vir: https://m.reichelt.com/de/en/raspberry-pi-camera-5mp-wide-angle-1080p-rpi-wwcam2-p219885.html?fbclid=IwAR1_JfNoDUGa5P7H_FdBO_6igwXFzxRZLp5pXIK48i_n0ny-MyGNw19-f6s)

Da lahko uporabljamo kamero, jo moramo pravilno priključiti na Raspberry Pi.

Slika 47: Priključek za kamero

(Osebni vir)

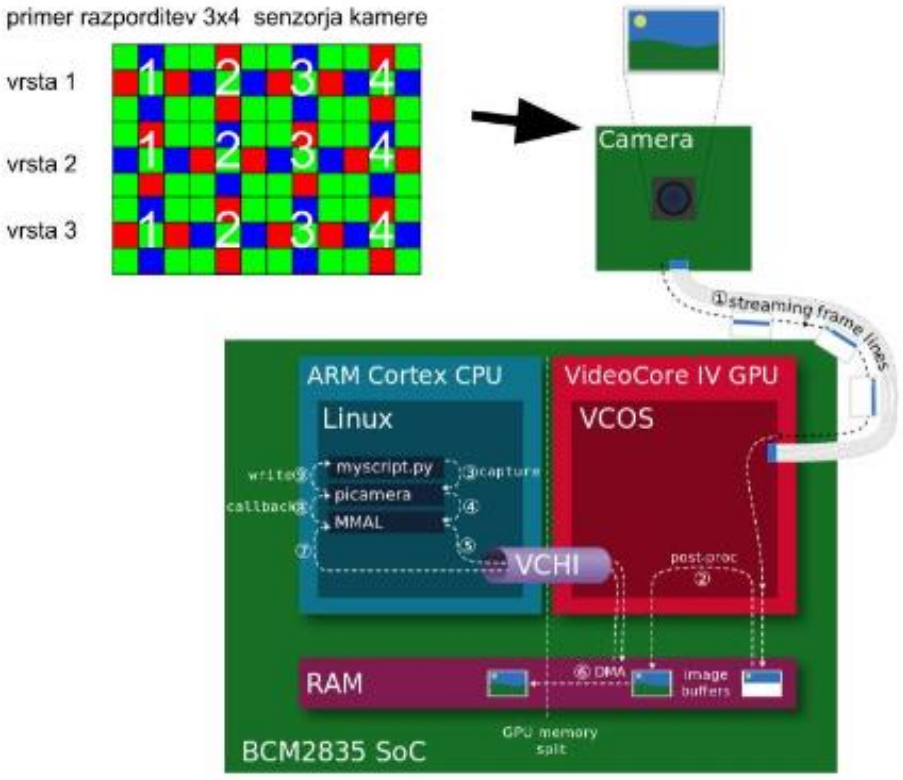


Slika 48: Priključitev kamere na Raspberry Pi

(Vir: https://picamera.readthedocs.io/en/release-1.13/quickstart.html)

5.2.2 Delovanje kamere

Kamera med svojim delovanjem konstantno zajema slike in jih pošilja v GPU (Graphics Processing Unit), tudi če program ne dela ničesar s kamero. Po zagonu je pomembno, da program počaka sekundo, da lahko kamera začne pošiljati podatke. To tudi omogoči, da se senzor prilagodi svetlobi. Na ta način dobimo jasne slike. Senzor si lahko predstavljamo kot matrico, sestavljeno iz števcev (senzorskih enot), ki štejejo količino rdeče, zelene in modre svetlobe.



Slika 49: Princip delovanja kamere

(Vir: https://picamera.readthedocs.io/en/release-1.13/fov.html)

Vsak piksel na sliki predstavlja svojo senzorsko enoto na senzorju kamere. Senzorske enote, postavljene v vrste, sestavljajo rdeči, modri in zeleni števci. Vrste se preberejo in pošljejo v GPU (Graphics Processing Unit), kjer se obdelajo za uporabo v programu. [12]

6 PROGRAMIRANJE RAČUNALNIŠKEGA VIDA

6.1 DEŠIFRIRANJE KODE QR

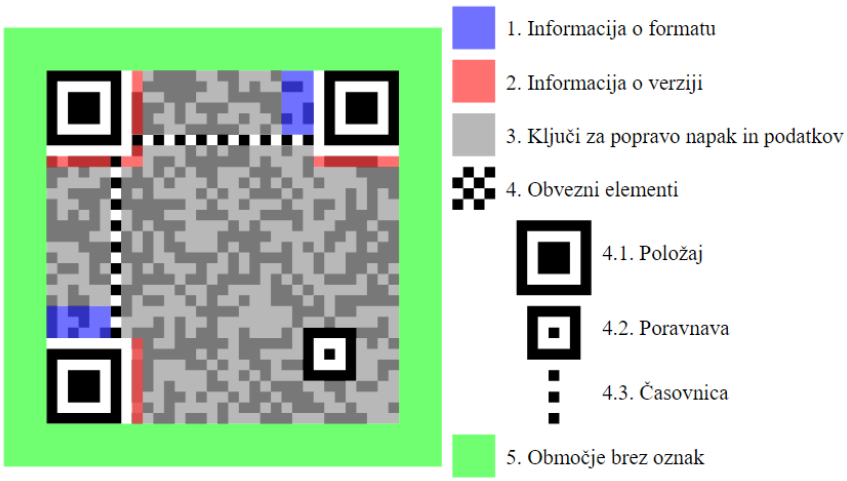
Koda QR (angleško QR code) je dvodimenzionalna koda, ki se je razvila za potrebe avtomobilske industrije na Japonskem in se je zaradi njenega hitrega ter zanesljivega dekodiranja razširila izven industrije. Beseda QR je kratica za angleški besedi "Quick Response", kar v slovenščini pomeni "hiter odziv". V praksi se uporablja v oglaševalske namene in najpogosteje vsebuje besedilo ali naslov URL, ki se po dekodiranju kode izpiše ali preusmeri na spletno stran.

Za učinkovito shranjevanje podatkov koda QR za šifriranje uporablja štiri standardne načine šifriranja: numerično, alfanumerično, bajtno/binarno in kandži (kitajske pismenke uporabljene v japonskem sistemu pisanja).

Koda QR je sestavljena iz črnih kvadratov, ki so razvrščeni na beli podlagi, in je lahko dekodirana s slikovnimi napravami, kot je kamera, ter obdelana z uporabo Reed – Solomon, dokler kode ni mogoče primerno interpretirati. Zahtevani podatki so nato pridobljeni iz vzorcev, ki so prisotni v horizontalnih in vertikalnih komponentah kode. [13]

Koda QR vsebuje naslednje podatke, ki so razvidni na sliki 50:

1. informacija o formatu
2. informacija o verziji
3. ključi za popravo napak in podatkov
4. obvezni elementi:
    a) položaj
    b) poravnava
    c) časovnica



Slika 50: Podatki, ki jih vsebuje koda QR.

(Vir: https://upload.wikimedia.org/wikipedia/commons/5/52/QR_code_structure_SLO.svg)

6.1.1 OpenCV, imutils in pyzbar

Za dekodiranje kode QR smo v našem primeru uporabili programski jezik Python 3, knjižnico OpenCV (angleško Open Source Computer Vision Library), kar v slovenskem prevodu pomeni odprtokodna knjižnica za računalniški vid in knjižnico imutils, ki vsebuje vrsto različnih funkcij za upravljanje s sliko, ter knjižnico pyzbar, ki se uporablja za branje enodimenzijskih črtnih kod in kod QR. Ker navedene knjižnice niso že vnaprej nameščene v operacijski sistem Raspbian, ki ga uporablja Raspberry Pi za delovanje, jih moramo namestiti ročno.

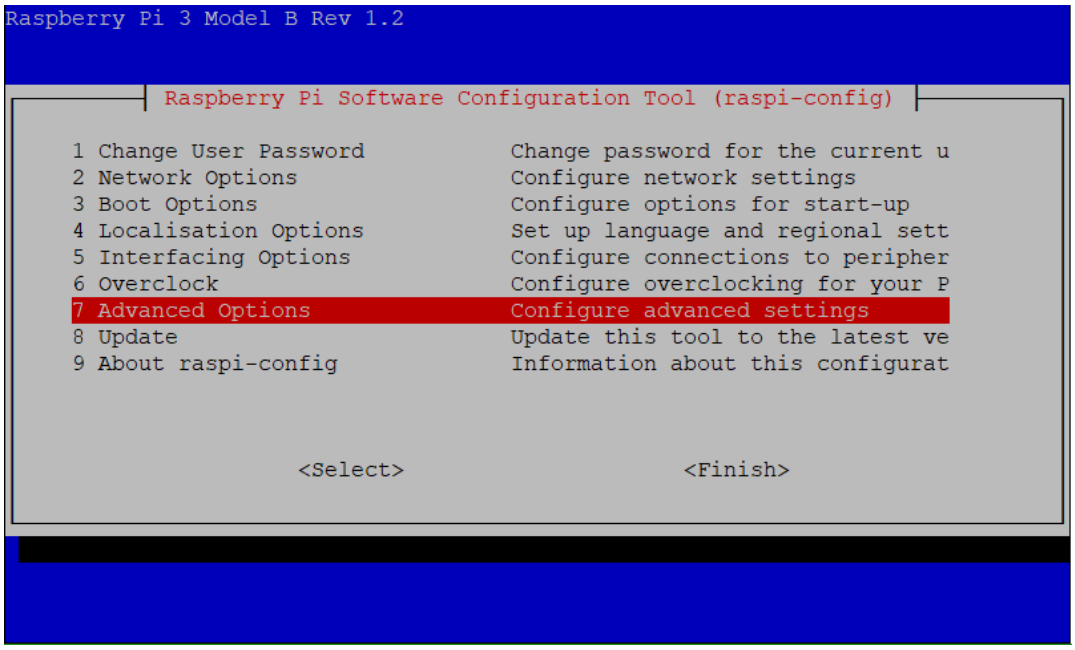
6.1.2 Namestitev OpenCV

Preden začnemo z namestitvijo te knjižnice, moramo vedeti, da njena namestitev zasede veliko prostora na SD-kartici, zato je priporočeno, da se uporabi kartica, katere velikosti je vsaj 16 GB. Namestitev poteka preko terminala, zato vse ukaze, ki imajo na začetku znak “$”, vtipkamo v terminal (brez znaka “$”).

Najprej moramo razširiti t. i. datotečni sistem, da bo vseboval ves razpoložljivi prostor mikro SD-kartice:

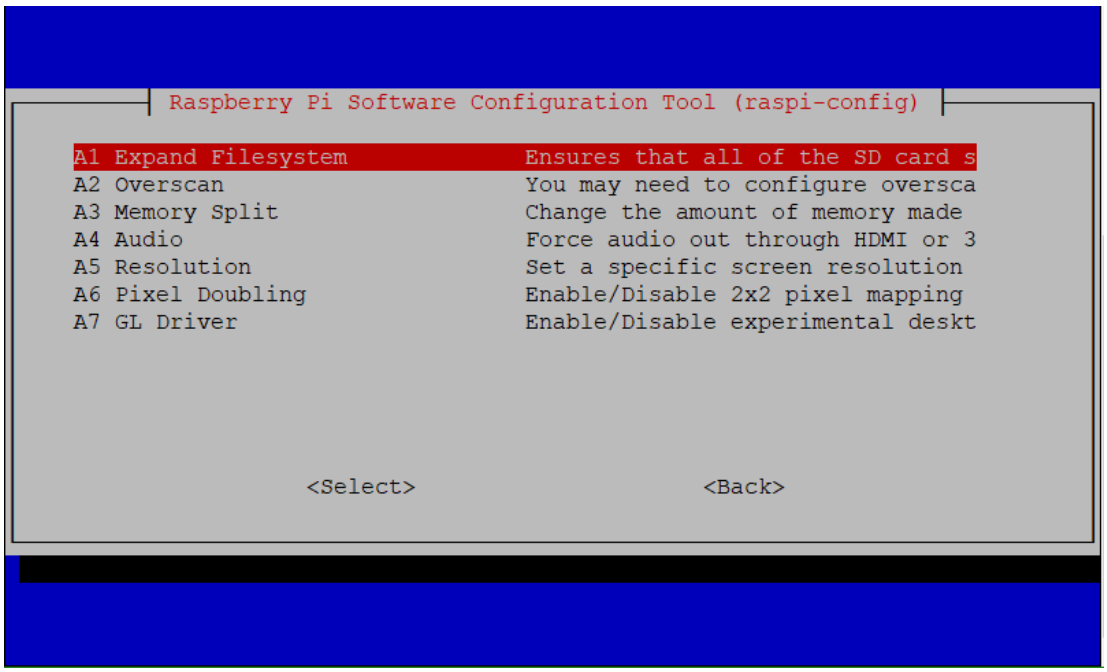
$ sudo raspi-config

Nato izberemo “Advanced Options”:



Slika 51: Izbira Advanced Options

(Osebni vir)



Slika 52: Izbira Expand Filesystem

(Osebni vir)

Zatem izberemo “Expand filesystem” in izberemo prvo možnost “A1. Expand File System” ter zaključimo z izbiro tipke “Finish”.

Razširitev datotečnega sistema bo nastopila šele po ponovnem zagonu, zato v terminal napišemo:

```
$ sudo reboot
```

Pred namestitvijo moramo posodobiti sistem:

```
$ sudo apt-get update && sudo apt-get upgrade
```

In namestiti potrebne programske pakete:

```
$ sudo apt-get install build-essential cmake unzip pkg-config
$ sudo apt-get install libjpeg-dev libpng-dev libtiff-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
$ sudo apt-get install libxvidcore-dev libx264-dev
$ sudo apt-get install libgtk-3-dev
$ sudo apt-get install libcanberra-gtk*
$ sudo apt-get install libatlas-base-dev gfortran
$ sudo apt-get install python3-dev
```

Naslednji korak je prenos opencv.zip in opencv_contrib.zip:

```
$ cd ~
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/4.0.0.zip
$ wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.0.0.zip
```

Prenesene datoteke razširimo:

```
$ unzip opencv.zip
$ unzip opencv_contrib.zip
```

In jih preimenujemo za lažji dostop:

```
$ mv opencv-4.0.0 opencv
$ mv opencv_contrib-4.0.0 opencv_contrib
```

Ker je programski paket NumPy predpogoj za namestitev OpenCV knjižnice, ga namestimo:

```
$ pip3 install numpy
```

Zatem odpremo mapo “opencv”, nato v njej naredimo mapo, poimenovano “build”, ki jo odpremo:

```
$ cd ~/opencv
$ mkdir build
$ cd build
```

Zdaj zaženemo CMake, da konfiguriramo gradnjo OpenCV 4:

```
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
    -D ENABLE_NEON=ON \
```

```
-D ENABLE_VFPV3=ON \
-D BUILD_TESTS=OFF \
-D OPENCV_ENABLE_NONFREE=ON \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D BUILD_EXAMPLES=OFF ..
```

Preden začnemo kompilirati, je priporočeno, da povečamo izmenljivi prostor (angleško swap space). To nam omogoča, da kompiliramo OpenCV z vsemi štirimi jedri Raspberry Pija, ne da bi prišlo do napake zaradi izčrpanosti spomina.

Najprej odpremo datoteko /etc/dphys-swapfile:

```
$ sudo nano /etc/dphys-swapfile
```

In nato spremenimo "CONF_SWAPSIZE" spremenljivko iz 100 na 2048:

```
# set size to absolute value, leaving empty (default) then uses computed value
#  you most likely don't want this, unless you have an special disk situation
CONF_SWAPSIZE=2048
```

Na koncu moramo ponovno zagnati storitev swap:

```
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo /etc/init.d/dphys-swapfile start
```

Zdaj smo pripravili vse in lahko začnemo kompilirati OpenCV 4:

```
$ make -j4
```

Nato namestimo OpenCV:

```
$ sudo make install
$ sudo ldconfig
```

Ko smo končali z namestitvijo, ne smemo pozabiti spremeniti vrednost spremenljivke "CONF_SWAPSIZE" nazaj na 100 in nato ponovno zagnati storitev swap.

Na koncu moramo spremeniti ime datoteke cv2.cpython-35m-arm-linux-gnueabihf.so v cv2.so, ki se lahko nahaja /usr/local/lib/python3.5/dist-packages/ ali /usr/local/lib/python3.5/site-packages/. V našem primeru se je nahajala v /usr/local/lib/python3.5/dist-packages/:

$cd/usr/local/lib/python3.5/dist-packages/
$sudo mv /usr/local/lib/python3.5/dist-packages/cv2.cpython-35m-arm-linux-gnueabihf.so cv2.so [14]

6.1.3 Namestitev imutils in pyzbar

Namestitev knjižnic imutils in pyzbar je veliko krajša v primerjavi z nameščanjem knjižnice OpenCV:

$ pip3 install imutils
$ pip3 install pyzbar

6.1.4 Delovanje programa za branje kod QR v realnem času

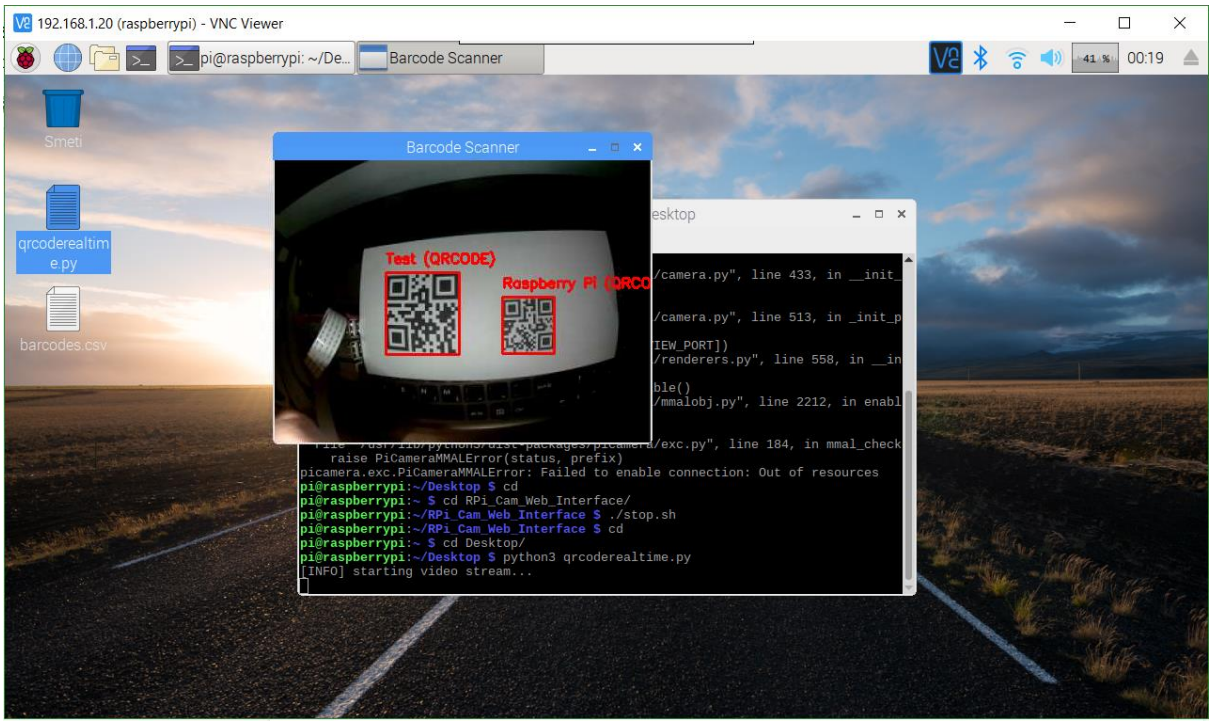
Na sliki 53 je prikazana koda programa za branje kod QR v realnem času.

```
# uvoz potrebnih knjižnic
from imutils.video import VideoStream
from pyzbar import pyzbar
import argparse
import datetime
import imutils
import time
import cv2

# konstrukcija razčlenjevalnika argumentov in razčlenitev argumentov
ap = argparse.ArgumentParser()
ap.add_argument("-o", "--output", type=str, default="barcodes.csv",
        help="path to output CSV file containing barcodes")
args = vars(ap.parse_args())

# initalizacija video prenosa
print("[INFO] starting video stream...")
# vs = VideoStream(src=0).start()
vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)

# odpiranje CSV dokumenta v katerega se bodo shranile najdene kode QR
csv = open(args["output"], "w")
found = set()

# glavna zanka
while True:
        # spremenitev velikosti videa tako, da je maksimalna širina videa
        # 400 slikovnih pik
        frame = vs.read()
        frame = imutils.resize(frame, width=400)

        # iskanje kod v vsaki sliki videa in dešifriranje vseh kod
        barcodes = pyzbar.decode(frame)

        # loop over the detected barcodes
        for barcode in barcodes:
                # risanje kvadratov okoli zaznanih kod QR na sliki
                (x, y, w, h) = barcode.rect
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)

                barcodeData = barcode.data.decode("utf-8")
                barcodeType = barcode.type

                # pisanje besedila, ki ga predstavlja posamezna koda QR
                text = "{} ({})".format(barcodeData, barcodeType)
                cv2.putText(frame, text, (x, y - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

                # če besedila kode še ni v CSV datoteki, zapišemo
                # časovni žig + kodo na disk
                if barcodeData not in found:
                        csv.write("{},{}\n".format(datetime.datetime.now(),
                                barcodeData))
                        csv.flush()
                        found.add(barcodeData)
        # prikaz izhodne slike
        cv2.imshow("Barcode Scanner", frame)
        key = cv2.waitKey(1) & 0xFF

        # če pritisnemo tipko "q" se zanka prekine
        if key == ord("q"):
                break

# zaprtje CSV datoteke
print("[INFO] cleaning up...")
csv.close()
cv2.destroyAllWindows()
vs.stop()
```

Slika 53: Program za branje kod QR v realnem času

(Osebni vir)

Na sliki 54 je prikazano delovanje programa, prikazanega na sliki 53.



Slika 54: Branje kod QR v realnem času

(Osebni vir)

6.1.5 Delovanje programa za branje kod QR z računalnikom

Ker Raspberry Pi nima dovolj procesorske moči, sta branje kode in prikaz slike v živo upočasnjena. Zato smo se odločili, da bomo kodo QR brali na način, da bo Raspberry Pi čim bolj razbremenjen, zato je potek dekodiranja kode QR sledeč:

1. Raspberry Pi s kamero zajame sliko in jo s programskim paketom RPi-Cam-Web-Interface pošlje na osebni računalnik.
2. Računalnik prejeto sliko prikaže v spletnem brskalniku.
3. Oseba, ki upravlja robota, na računalniku zažene program za branje kode QR.
4. Program najprej na zaslonu izriše odštevalnik, da vemo, kdaj računalnik zajame sliko prikazanega zaslona, ki jo nato shrani in dekodira vse kode, ki so prikazane na zaslonu. Nato prikaže okno, ki vsebuje prej zajeto sliko z vsemi dekodiranimi kodami.

Ker bomo naslednje programe zagnali na računalniku, moramo nanj namestiti OpenCV, imutils, pyzbar in PyAutoGUI, ki nam omogočajo zajem zaslona. Knjižnice Tkinter, ki nam omogoča ustvariti grafični uporabniški vmesnik, ni treba namestiti, saj je vključena z vsemi standardnimi Python distribucijami. Vse ostale knjižnice namestimo preko terminala z ukazi:

$ pip3 install opencv-python
$ pip3 install imutils
$ pip3 install PyAutoGUI

Ko namestimo potrebne knjižnice, lahko zaženemo program, ki je prikazan na sliki 55. Ta program najprej zažene program, ki nam prikaže časovni števec. Nato počaka 3 sekunde in zažene program za zajem zaslona ter na koncu odpre zajeto sliko zaslona in program za dešifriranje kode QR, ki na koncu izriše okno z dešifriranimi kodami.

```
import sc #uvoz knjižnice za zajemanje zaslona
import os #uvoz knjižnice za zagon programa, ki dešifrira kodo QR
import time #uvoz knjižnice za uporabo časa
from counter import counter #uvoz knjižnice za prikaz časovnega števca

counter() #prikaz časovnega števca

time.sleep(3) #čakanje

sc.capture()  #zajem zaslona

os.system("qrsc.py --image qr.png") #zagon programa, ki dešifrira kodo QR in
                                    #nalaganje zajete slike zaslona
```

Slika 55: Program, ki kliče ostale programe in knjižnice v zaporedju.

(Osebni vir)

```
import tkinter as tk #uvoz knjižnice, ki nam omogoča ustvariti
                     #grafični uporabniški vmesnik
import time #uvoz knjižnice za uporabo časa

root = tk.Tk()
time_str = tk.StringVar()
def count_down(): #odštevalnik

    for t in range(3, -1, -1):

        time_str.set(t)
        root.update()
        time.sleep(1)

def counter(): #izris okna z odštevalnikom

    label_font = ('helvetica', 40) #izbira pisave
    tk.Label(root, textvariable=time_str, font=label_font, bg='white',
             fg='blue', relief='raised', bd=3).pack(fill='x', padx=5, pady=5)

    count_down() #izvedba odštevanja
    root.destroy() #zapiranje grafičnega uporabniškega vmesnika
    root.mainloop()
```

Slika 56: Program, ki izriše grafični uporabniški vmesnik z odštevalnikom.

(Osebni vir)

```
# uvoz potrebnih knjižnic
import numpy as np
import pyautogui
import imutils
import cv2

def capture():
    image = pyautogui.screenshot() # zajem slike in shranitev slike v spomin

    image = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR) #pretvorba slike PIL/Pillow
                                                             #v NumPy "array", katerega
                                                             #podpira knjižnica OpenCV
    img = image [89:1030, 122:1798]
    cv2.imwrite("qr.png", img) # zapis slike na disk
    cv2.waitKey(0)
```

Slika 57: Program za zajemanje slike zaslona

(Osebni vir)

```
# uvoz potrebnih knjižnic
from pyzbar import pyzbar
import argparse
import cv2


# konstrukcija razčlenjevalnika argumentov in razčlenitev argumentov
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
        help="path to input image")
args = vars(ap.parse_args())

# uvoz slike
image = cv2.imread(args["image"])
image = cv2.resize(image, (960, 540))
# iskanje kod QR na sliki ter dešifriranje vseh kod
barcodes = pyzbar.decode(image)

# glavna zanka
for barcode in barcodes:
        # risanje kvadratov okoli zaznanih kod QR na sliki
        (x, y, w, h) = barcode.rect
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)

        barcodeData = barcode.data.decode("utf-8")
        barcodeType = barcode.type

        # pisanje besedila, ki ga predstavlja posamezna koda QR
        text = "{} ({})".format(barcodeData, barcodeType)
        cv2.putText(image, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
                0.5, (0, 0, 255), 2)

        # izpis tipa kode in podatkov v terminal
        print("[INFO] Found {} barcode: {}".format(barcodeType, barcodeData))

 # prikaz slike z dešifriranimi kodami
cv2.imshow("Image", image)
cv2.waitKey(0)
```

Slika 58: Program za branje kod QR

(Osebni vir)

Slika 59: Branje kod QR z računalnikom

(Osebni vir)

6.1.6 Rezultati branja kod QR

Branje kod QR samo z uporabo Raspberry Pija je mogoče in uporabno, ker prenos slike poteka brez motenj in je dovolj zanesljiv. Problem se pojavi, če kamero želimo hkrati uporabljati še za kakšno drugo nalogo. Rešitev za to je branje kod QR z računalnikom, saj ta hkrati razbremeni Raspberry Pi in omogoča uporabo kamere še za druge namene. Ta način je zanesljivejši kot dešifriranje samo z uporabo Raspberry Pija. Edina pomanjkljivost je, da zaznavanje traja dlje. Torej, če bi v praksi imeli na voljo samo en Raspberry Pi, bi bila najboljša rešitev, da bi uporabili metodo, ki vključuje dešifriranje z računalnikom. Če bi imeli na voljo več kot en Raspberry Pi oziroma bi bil njegov namen samo branje kod QR, bi bilo bolje, da uporabimo metodo, pri kateri se uporablja samo Raspberry Pi.

Izvedli smo meritve v različnih stopnjah osvetljenosti, načinih branja in oddaljenosti od QR-kode. Rezultati, ki so podani v tabeli 1, so bili izmerjeni z Raspberry Pijem in rezultati, podani v tabeli 2, so bili izmerjeni z računalnikom. Za merjenje osvetljenosti smo uporabili pametni telefon.

Tabela 1: Meritve pri branju QR-kode z Raspberry Pijem

Osvetljenost [lx]	Razdalja [cm]	Ali je bila koda prebrana?
15	10	Da
15	20	Da
15	30	Ne
15	40	Ne
15	50	Ne
15	60	Ne
100	10	Da
100	20	Da
100	30	Da
100	40	Ne
100	50	Ne
100	60	Ne
300	10	Da
300	20	Da
300	30	Da
300	40	Ne
300	50	Ne
300	60	Ne

Tabela 2: Meritve pri branju QR-kode z računalnikom

Osvetljenost [lx]	Razdalja [cm]	Ali je bila koda prebrana?
15	10	Da
15	20	Da
15	30	Da
15	40	Da
15	50	Da
15	60	Ne
100	10	Da
100	20	Da
100	30	Da
100	40	Da
100	50	Da
100	60	Da
300	10	Da
300	20	Da

300	30	Da
300	40	Da
300	50	Da
300	60	Da

Iz podatkov v tabelah je razvidno, da je branje QR-kode zanesljivejše, če uporabimo način, pri katerem računalnik dešifrira QR-kodo. Zanesljivost je najboljša na kratki razdalji med QR-kodo in kamero ter ob veliki stopnji osvetljenosti. Branje QR-kode z Raspberry Pijem na razdalji, daljši od 30 cm, je z uporabljeno kamero nemogoče zaradi nizke kakovosti slike. Dešifriranje z računalnikom nam omogoča, da beremo QR-kodo tudi na razdaljah, daljših od 50 cm, če je QR-koda dovolj osvetljena. Razlog za to je, da je prejeta slika nekoliko približana in so s tem podrobnosti na sliki vidnejše.

6.2 ZAZNAVANJE OBJEKTOV

Kot smo že omenili v poglavju, v katerem smo govorili o računalniškem vidu in kako le-ta deluje, se za zaznavanje objektov uporabljajo naučeni modeli. Te modele naučimo tako, da računalniku damo različne slike objektov, ki jih želimo zaznavati. Najbolje je, da mu podamo čim več slik teh objektov, saj bo računalnik lahko preučil več slik in značilnosti objektov na njih. Posledično bo tudi večja verjetnost, da bo zaznal objekte zanesljivo in hitro.

V našem primeru smo uporabili že vnaprej naučen model, da bi prihranili čas, saj učenje takšnega modela traja več ur. Ker uporabljamo Raspberry Pi, smo uporabili model tipa MobileNet. MobileNet je arhitektura, ki je primerna za mobilne aplikacije in aplikacije, ki temeljijo na vgrajenem vidu, pri katerem je pomanjkanje računalniške moči. [16]

6.2.1 Delovanje programa za zaznavanje ljudi in objektov

Za uporabo programa, navedenega na sliki 60, je potrebno imeti nameščene knjižnice OpenCV, imutils in NumPy, ki smo jih že namestili.

```
# uvoz potrebnih knjižnic
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import time
import cv2

# konstrukcija razčlenjevalnika argumentov in razčlenitev argumentov
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
        help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
        help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
        help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# initalizacija seznama oznak vrst, katere je bil MobileNet naučen zaznavati
# generiranje barve pravokotnika, ki se bo izrisal ob zaznavanju objektov
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
        "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
        "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
        "sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
```

Slika 60: Program za zaznavanje ljudi in objektov v realnem času 1. del

(Osebni vir)

```
# nalaganje MobileNet modela iz mikro SD kartice
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# initalizacija video prenosa
# initalizacija števca sličic na sekundo
print("[INFO] starting video stream...")
# vs = VideoStream(src=0).start()
vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)
fps = FPS().start()

# galvna zanka
while True:
	# sprememba velikosti videa tako, da je maksimalna širina videa
	# 400 slikovnih pik
	frame = vs.read()
	frame = imutils.resize(frame, width=400)

	# pridobitev dimenzij okvirjev
	(h, w) = frame.shape[:2]
	blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
		0.007843, (300, 300), 127.5)
	net.setInput(blob)
	detections = net.forward()


	for i in np.arange(0, detections.shape[2]):
		# pridobitev verjetnosti, ki je povezana z predvidevanjem
		confidence = detections[0, 0, i, 2]

		# filtiranje šibkih detekcij s tem, da zagotovimo, da je verjetnost
		# večja od minimalne nastavljene verjetnosti
		if confidence > args["confidence"]:
			# pridobitev oznak vrst iz detekcij in izračun koordinat
			# okvirja objekta
			idx = int(detections[0, 0, i, 1])
			box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
			(startX, startY, endX, endY) = box.astype("int")

			# izris oznake, pravokotnika in izpis verjetnosti
			label = "{}: {:.2f}%".format(CLASSES[idx],
				confidence * 100)
			cv2.rectangle(frame, (startX, startY), (endX, endY),
				COLORS[idx], 2)
			y = startY - 15 if startY - 15 > 15 else startY + 15
			cv2.putText(frame, label, (startX, y),
				cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)

	# prikaz slike izhoda
	cv2.imshow("Frame", frame)
	key = cv2.waitKey(1) & 0xFF

	# če pritisnemo tipko "q" se zanka prekine
	if key == ord("q"):
		break

	# osvežitev števca sličic
	fps.update()

# ustavitev časovnika in prikaz informacij o številu sličic na sekundo
fps.stop()
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

cv2.destroyAllWindows()
vs.stop()
```

Slika 61: Program za zaznavanje ljudi in objektov v realnem času 2. del

(Osebni vir)

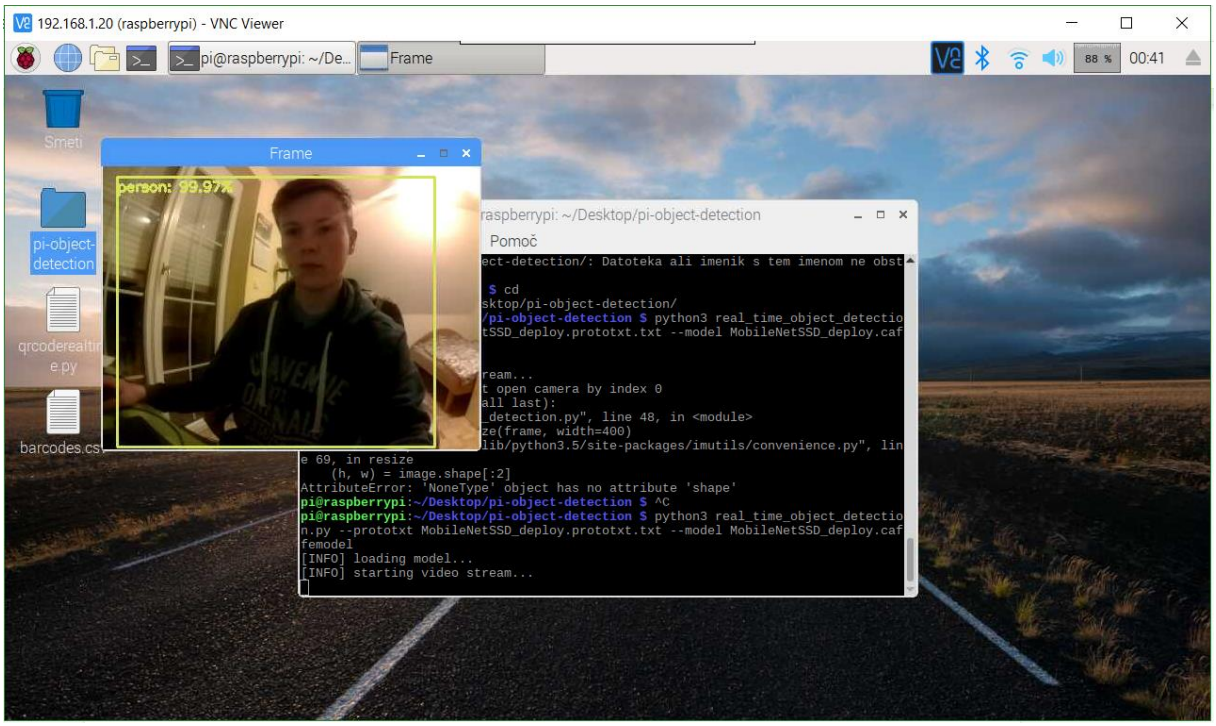
Ker moramo poleg zagona programa naložiti tudi MobileNet, to naredimo z naslednjim ukazom:

$ python3 real_time_object_detection.py\

--prototxMobileNetSSD_deploy.prototxt.txt\

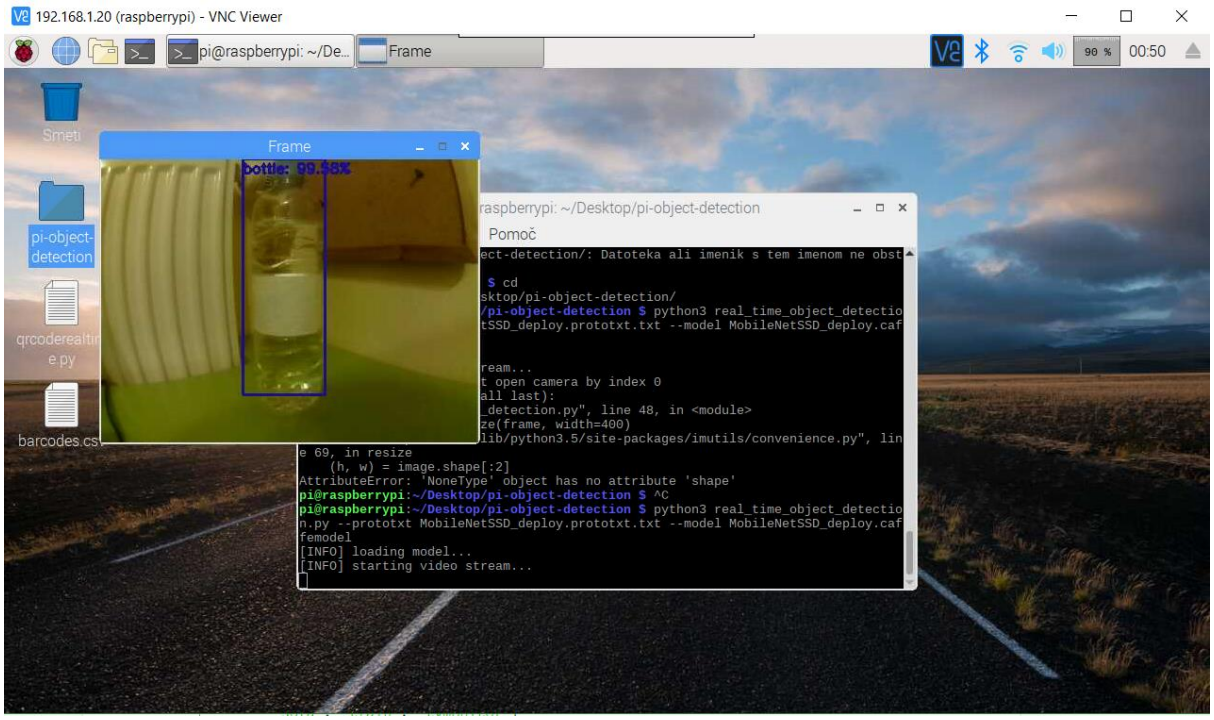
--model MobileNetSSD_deploy.caffemodel

Kljub številu sličic na sekundo in slabi resoluciji slike živega prenosa je Raspberry Piju uspelo prepoznati osebo, plastenko vode, avtomobil in rastlino v lončku.



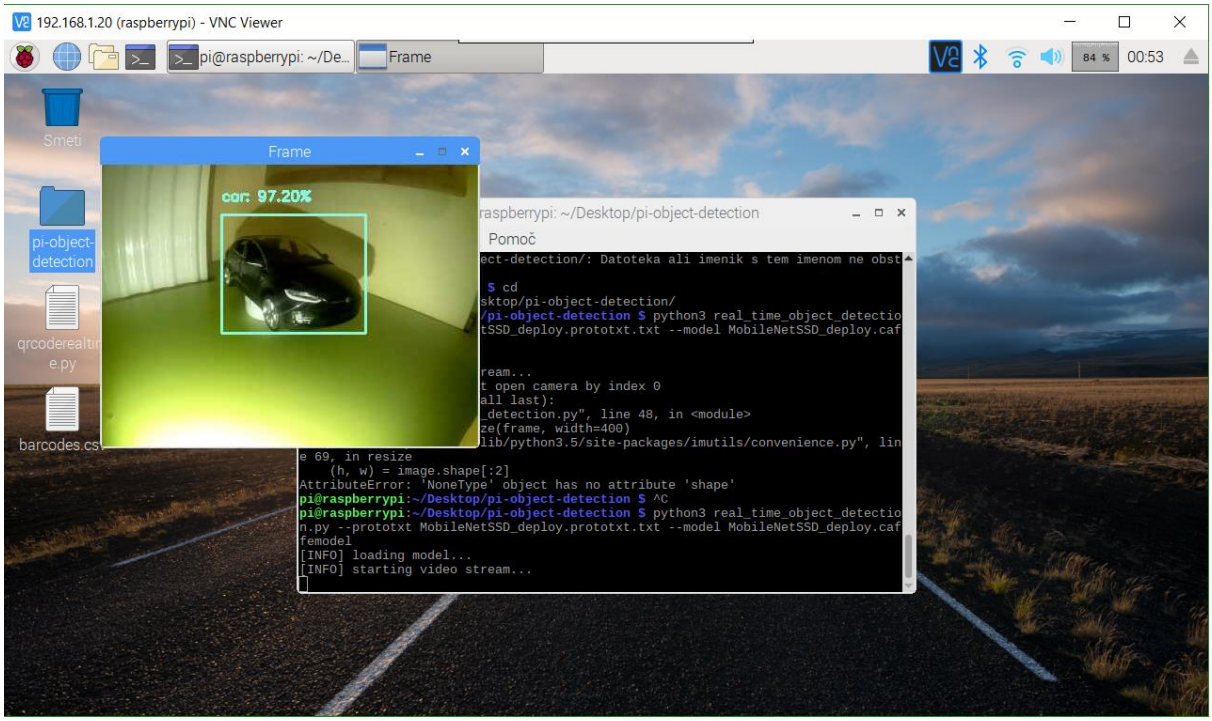
Slika 62: Zaznavanje ljudi v realnem času

(Osebni vir)



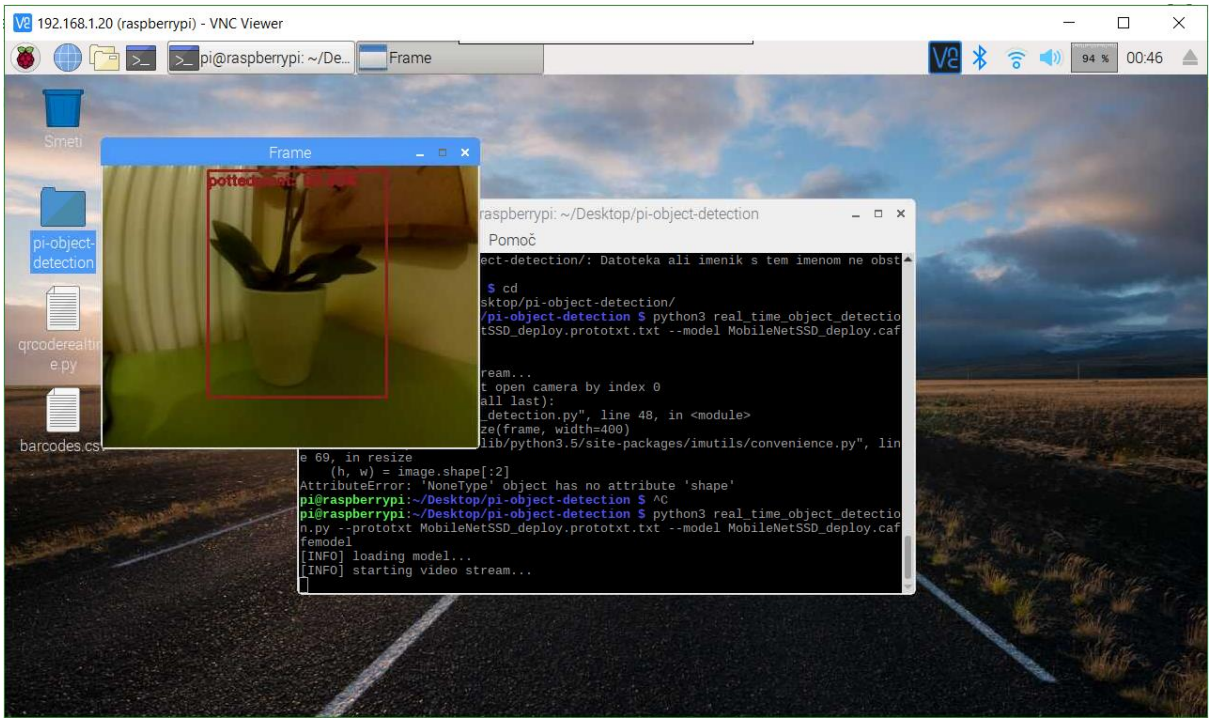
Slika 63: Zaznavanje plastenke vode v realnem času

(Osebni vir)



Slika 64: Zaznavanje modela avtomobila v realnem času

(Osebni vir)



Slika 65: Zaznavanje rastline v lončku v realnem času

(Osebni vir)

6.2.2 Delovanje programa za zaznavanje ljudi in objektov z računalnikom

Kot smo že omenili, je kakovost prenosa slike v živo zelo slaba in zaznavanje objekta traja nekaj časa, ker se mora kamera stabilizirati. Zaradi tega smo se odločili, da bomo brali na enak način kot kodo QR. Edina razlika je v tem, da bo računalnik, namesto da bi zagnal program za branje kode QR, zagnal program, ki zaznava ljudi in objekte ter naloži MobileNet model.

```
import sc #uvoz knjižnice za zajemanje zaslona
import os #uvoz knjižnice za zagon programa, ki zaznava ljudi in objekte
import time #uvoz knjižnice za uporabo časa
from counter import counter #uvoz knjižnice za prikaz časovnega števca

counter() #prikaz časovnega števca

time.sleep(3) #čakanje

sc.capture()  #zajem zaslona

os.system("object_detection.py --image screenshot.png \
        --prototxt MobileNetSSD_deploy.prototxt.txt \
        --model MobileNetSSD_deploy.caffemodel") #zagon programa, ki zaznava ljudi in objekte
                                #nalaganje zajete slike zaslona
                                #nalaganje MobileNet modela
```

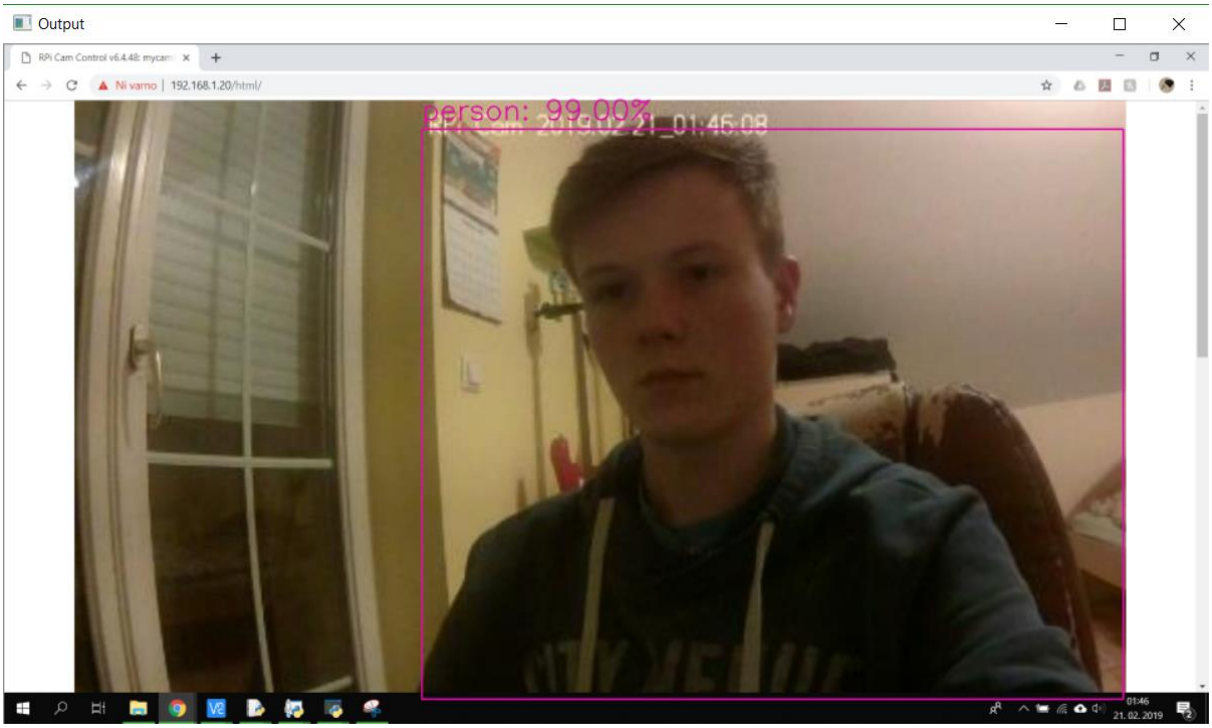
Slika 66: Program, ki kliče ostale programe za zaznavanje in knjižnice v zaporedju.

(Osebni vir)

```
# uvoz potrebnih knjižnic
import numpy as np
import argparse
import cv2

# konstrukcija razčlenjevalnika argumentov in razčlenitev argumentov
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
        help="path to input image")
ap.add_argument("-p", "--prototxt", required=True,
        help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
        help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.2,
        help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# initalizacija seznama oznak vrst, katere je bil MobileNet naučen zaznavati
# generiranje barve pravokotnika, ki se bo izrisal ob zaznavanju objektov
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
        "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
        "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
        "sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# nalaganje MobileNet modela iz mikro SD kartice
print("[INFO] nalaganje modela...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

print("[INFO] nalaganje slike...")
image = cv2.imread(args["image"])
(h, w) = image.shape[:2]
blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 0.007843,
        (300, 300), 127.5)

# pridobivanje detekcij in predvidevanj
print("[INFO] računanje zaznavanja objektov...")
net.setInput(blob)
detections = net.forward()

# glavna zanka
for i in np.arange(0, detections.shape[2]):
        # pridobivanje verjetnosti, ki je povezana z predvidevanjem
        confidence = detections[0, 0, i, 2]

        # filtriranje šibkih detekcij s tem, da zagotovimo, da je verjetnost
        # večja od minimalne nastavljene verjetnosti
        if confidence > args["confidence"]:
                # pridobitev oznak vrst iz detekcij in izračun koordinat
                # okvirja objekta
                idx = int(detections[0, 0, i, 1])
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")

                # izris oznake, pravokotnika in izpis verjetnosti
                label = "{}: {:.2f}%".format(CLASSES[idx], confidence * 100)
                print("[INFO] {}".format(label))
                cv2.rectangle(image, (startX, startY), (endX, endY),
                        COLORS[idx], 2)
                y = startY - 15 if startY - 15 > 15 else startY + 15
                cv2.putText(image, label, (startX, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 1.5, COLORS[idx], 2)

# prikaz slike izhoda
image = cv2.resize(image, (960, 540))
cv2.imshow("Output", image)
cv2.waitKey(0)
```

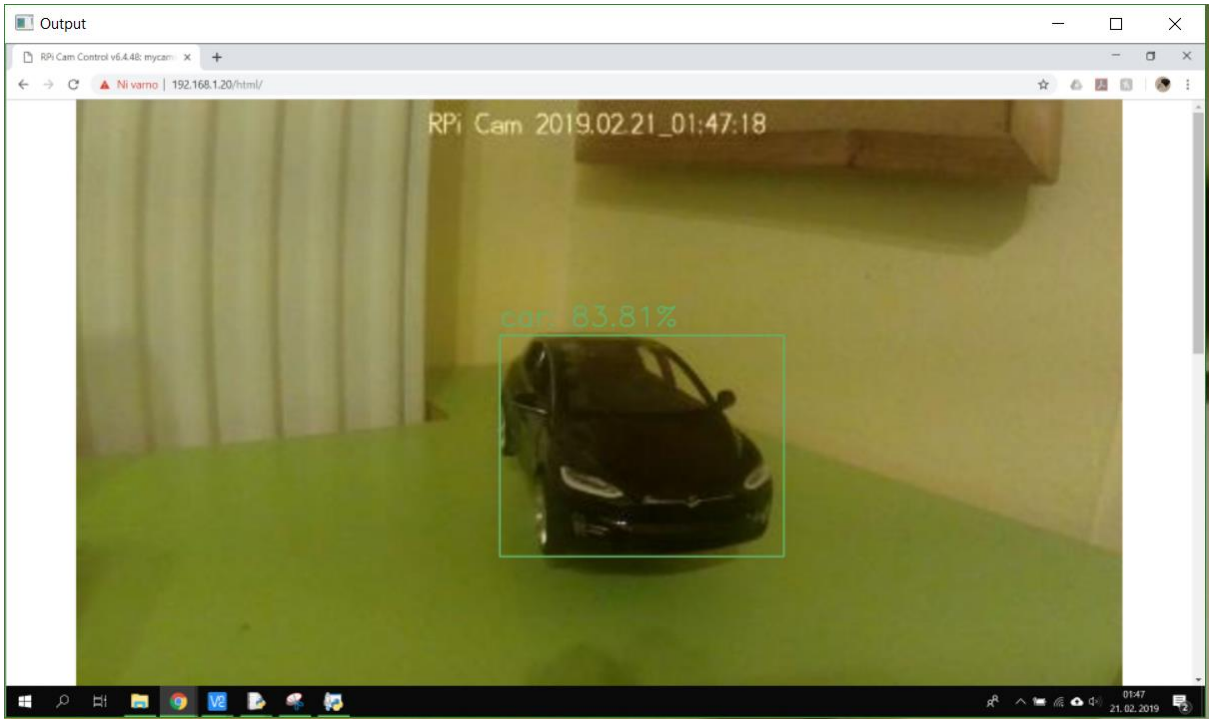
Slika 67: Program za zaznavanje ljudi in objektov z računalnikom

(Osebni vir)



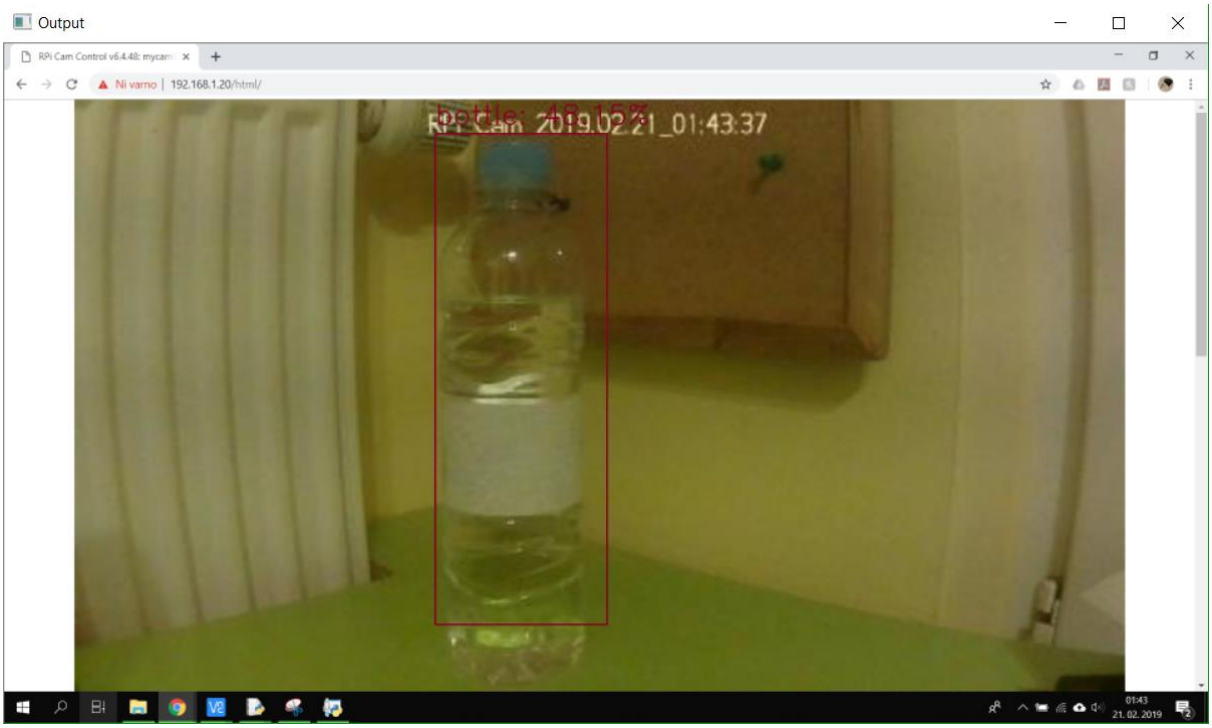
Slika 68: Primer zaznavanja osebe

(Osebni vir)



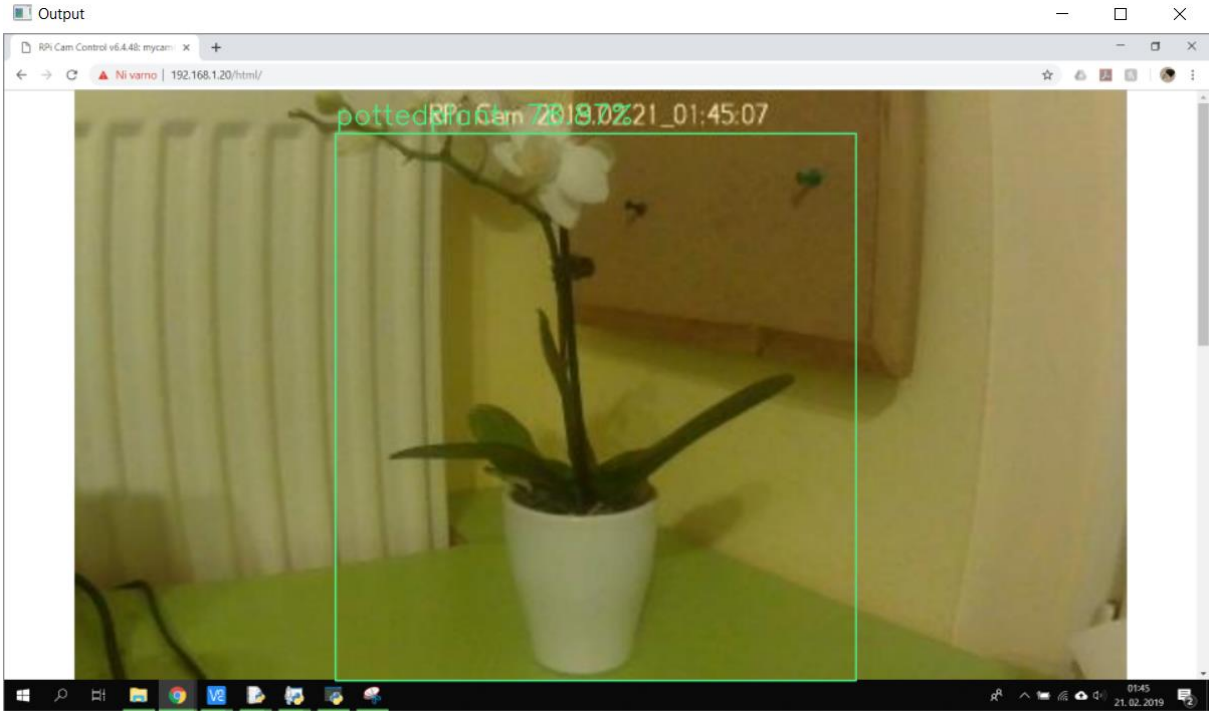
Slika 69: Primer zaznavanja avtomobila

(Osebni vir)



Slika 70: Primer zaznavanja plastenke

(Osebni vir)



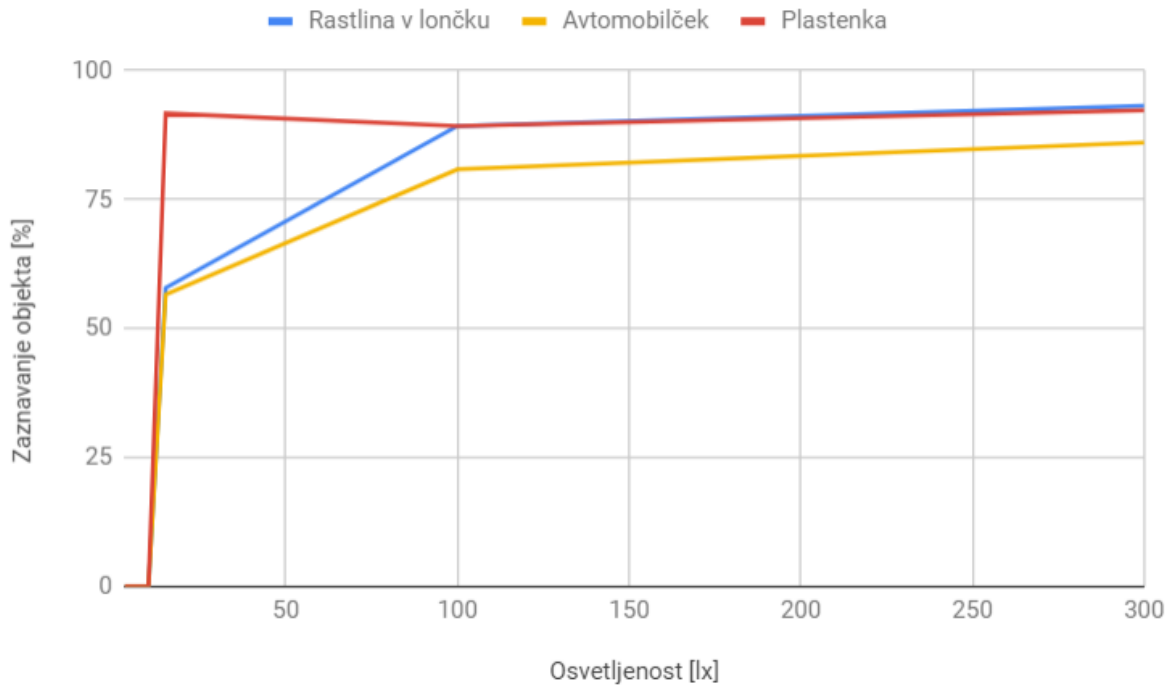
Slika 71: Primer zaznavanja rastline

(Osebni vir)

6.2.3 Rezultati zaznavanja ljudi in objektov

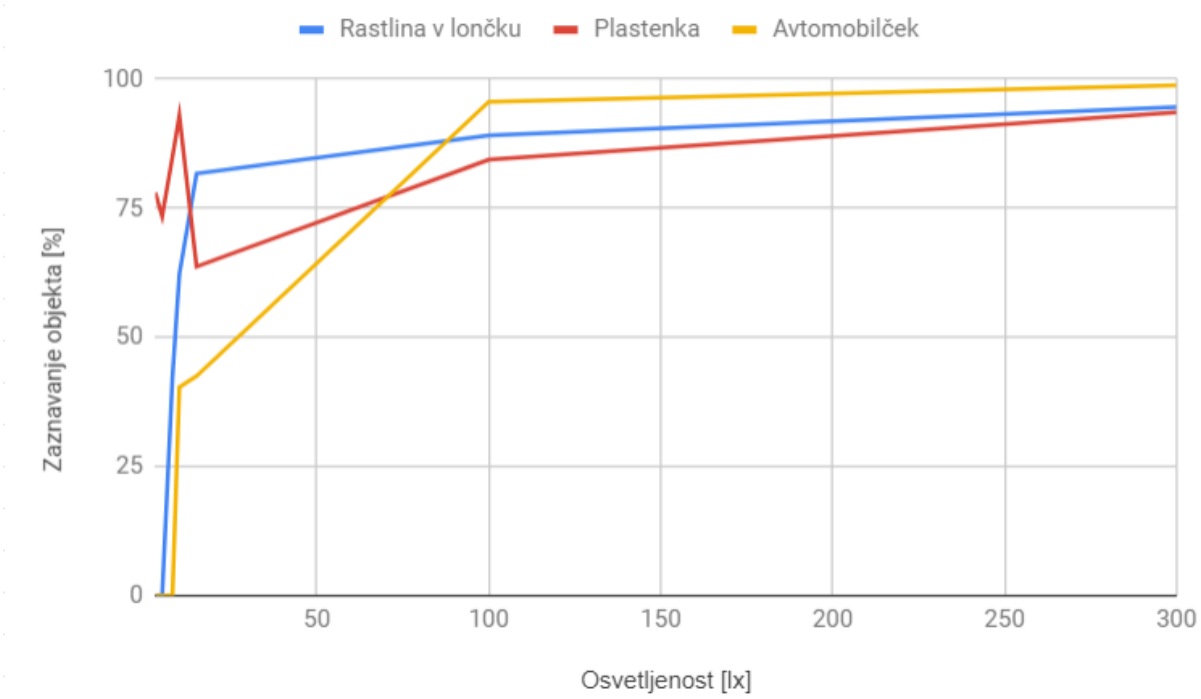
Zaznavanje ljudi in objektov z Raspberry Pijem je uporabno samo, če Raspberry Pi uporabimo kot napravo, ki pošilja sliko kamere na računalnik. Glavni razlog za to je zelo nizka hitrost sličic živega prenosa, saj se hitrost med zaznavanjem spusti pod 1 sličico na sekundo, kar je v robotu neuporabno. Drugi razlog za to je, da zaznavanje v povprečju uporabi ≈90 % procesorja, kar lahko ovira ostale procese, ki se istočasno izvajajo. V skrajnem primeru lahko pride tudi do sesutja sistema. Za zaznavanje ljudi in objektov je torej veliko bolj uporabno, da to izvedemo preko računalnika, saj je zaznavanje na ta način hitrejše in zanesljivejše, ker ni potrebno čakati, da se ob premiku kamere slika stabilizira. V praksi bi torej za zaznavanje v živo potrebovali boljši računalnik, kot je Raspberry Pi.

Izvedli smo tudi meritve, pri katerih smo merili, koliko odstotno ujemanje je pri zaznavanju objekta za določeno osvetljenost. Graf na sliki prikazuje zaznavanje z računalnikom in zaznavanje z Raspberry Pijem. Objekti so bili pri vseh meritvah postavljeni 40 cm stran od kamere. Meritve smo izvajali na osvetljenosti 3 lx, 5 lx, 8 lx, 10 lx, 15 lx, 100 lx in 300 lx. Osvetljenost smo izmerili s pametnim telefonom.



Graf 1: Meritev zaznavanja z Raspberry Pijem

(Osebni vir)



Graf 2: Meritev zaznavanja z računalnikom

(Osebni vir)

Kot je razvidno z grafov je zaznavanje z večjo osvetljenostjo zanesljivejše. Metoda, pri kateri objekte zaznava računalnik, je zanesljivejša, sploh pri manjših stopnjah osvetljenosti, ker lahko preko programskega paketa RPi Cam Web Interface nastavljamo svetlost prejete slike. Iz dobljenih podatkov je razvidno tudi, da pri manjših stopnjah osvetljenosti pride pri določenih objektih do večje natančnosti. Razlog za to je najverjetneje temna okolica, ki bi drugače lahko delovala kot motnja pri zaznavanju.

Zaznavanje objektov smo izvedli tudi na različnih oddaljenostih od objektov in pod določeno osvetljenostjo. Rezultati so prikazani v tabeli 3 in tabeli 4.

Tabela 3: Meritve zaznavanja objektov z Raspberry Pijem

Osvetljenost [lx]	Razdalja [cm]	Zaznavanje: Rastlina v lončku	Zaznavanje: Avtomobilček	Zaznavanje: Plastenka
15	10	26,96%	88,29%	88,57%
15	20	76,27%	62,73%	86,56%
15	30	79,49%	58,21%	54,12%
15	40	57,88%	56,58%	91,70%
15	50	0%	0%	27,21%
15	60	0%	0%	0,00%
100	10	24,25%	80,79%	83,57%
100	20	67,15%	98,59%	99%
100	30	45,97%	91,17%	78,63%
100	40	89,28%	80,86%	89,21%
100	50	35,89%	0%	77,63%
100	60	26,26%	0%	0%
300	10	25,53%	99,79%	97,57%
300	20	29,91%	99,32%	94,06%
300	30	49,21%	92,21%	57,63%
300	40	93,13%	86,04%	92,32%
300	50	54,83%	0%	81,17%
300	60	35,98%	0%	51,38%

Tabela 4: Meritve zaznavanja objektov z računalnikom

Osvetljenost [lx]	Razdalja [cm]	Zaznavanje: Rastlina v lončku	Zaznavanje: Avtomobilček	Zaznavanje: Plastenka
15	10	27,09%	62,77%	89,54%
15	20	77,23%	68,75%	87,69%
15	30	79,95%	62,77%	58,72%
15	40	81,69%	42,49%	63,69%
15	50	44%	0%	28,24%
15	60	0%	0%	0%
100	10	28,05%	99,67%	92,42%
100	20	78,62%	99,98%	98,22%
100	30	55,68%	98,56%	95,26%
100	40	89,09%	95,62%	84,40%
100	50	37,65%	0%	77,53%
100	60	27,11%	0%	0%
300	10	27,38%	99,89%	92,81%
300	20	82,72%	99,99%	92,34%
300	30	56,47%	98,84%	83,04%

300	40	94,56%	98,79%	93,52%
300	50	56,63%	0%	82,59%
300	60	39,43%	0%	60,22%

Izmerjeni rezultati nam povedo, da je pri zaznavanju pomembno, da je objekt primerno oddaljen od kamere in primerno osvetljen. Če je kamera preblizu objekta, ga program ne prepozna, ker ne vidi njegove oblike, sploh če so objekti večji. Če je kamera preveč oddaljena od objekta, ga ne prepozna, ker se ne da avtomatsko izostriti, kar pomeni, da slika ni dovolj ostra, da bi lahko program natančno zaznal značilnosti objekta. Če je objekt premalo osvetljen, ga program ne zazna, ker kamera ne vidi njegovih podrobnosti.

7 PREDSTAVITEV REZULTATOV

Kljub temu da se nam je zdela naloga na začetku zelo zapletena, smo uspeli raziskovalno nalogo pripeljati do konca. Pri našem raziskovanju smo naleteli na veliko težav, a smo jih uspeli razrešiti s pomočjo spleta in lastnega znanja, ki smo ga pridobili med izobraževanjem. Prvič smo se srečali s programskim okoljem Python, kar nam je predstavljalo izziv, vendar smo s tem pridobili veliko novih izkušenj in novega znanja. Po testiranju računalniškega vida smo prišli do raznih ugotovitev, s katerimi lahko potrdimo oz. zavržemo naše hipoteze.

Tabela 5: Potrditev hipotez

H1 – s kamero bomo dosegli zanesljivost in ponovljivost razpoznavanja;	✗
H2 – kamera bo razpoznavala ljudi;	✓
H3 – kamera bo razpoznala objekte;	✓
H4 – kamera bo dekodirala QR-kodo;	✓
H5 – na zaslonu se bo izpisalo, kateri objekt je bil razpoznan;	✓

H1 – s kamero bomo dosegli zanesljivost in ponovljivost razpoznavanja. Vsekakor je kamera zanesljiva v primeru, da za naše zaznavanje v realnem času uporabimo le dekodiranje QR-kod. Težava nastane, če želimo hkrati zaznavati QR-kodo in objekte. V tem primeru se uporabi veliko procesorske moči, kar lahko ovira druge procese, ki se izvajajo. Edina rešitev v tem primeru bi bila, da izberemo močnejši mikroračunalnik kot Raspberry Pi, ki nam bo zagotovil zaznavanje objektov in dekodiranje QR-kod v realnem času. Z Raspberry Pijem ne bi bila mogoča avtonomna vožnja, zato lahko to hipotezo zavržemo.

H2 – kamera bo razpoznavala ljudi. Največ težav se je pojavilo pri programiranju z OpenCV, ki nam omogoča zaznavanje ljudi. Težave so se pojavile pri prenosu knjižnic na mikroračunalnik. Knjižnica je zavzela veliko prostora, a nam je kljub vsem težavam na koncu uspelo razpoznati ljudi, zato lahko to hipotezo potrdimo.

H3 – kamera bo razpoznala objekte. Pri našem preizkušanju smo uporabili knjižnico, v kateri so že naloženi osnovni modeli, ki nam omogočajo prepoznavo objektov. Pri našem preizkusu programa smo zaznavali model avtomobila, plastenko in rastlino. Vse objekte je program uspešno prepoznal, zato lahko to hipotezo potrdimo.

H4 – kamera bo dekodirala QR-kodo. To je bila naša prva naloga, ki smo si jo zadali, saj je v primerjavi s programom za razpoznavanje ljudi veliko lažja in je porabila najmanj našega časa. Na koncu nam je uspelo vse QR-kode uspešno prebrati, zato lahko to hipotezo potrdimo.

H5 – na zaslonu se bo izpisalo, kateri objekt je bil razpoznan. Program na osnovi modelov, ki mu jih podamo, zazna razne objekte in okoli njih nariše pravokotnik ter izpiše na zaslon njihovo ime. S tem lahko potrdimo tudi to hipotezo.

8 MOŽNOST NADALJNJEGA RAZISKOVANJA

Kljub temu da smo zaključili z našo raziskovalno nalogo, upamo, da smo vzpodbudili zanimanje za nadaljnje raziskovanje, ne le na področju računalniškega vida, ampak tudi na področju reševalnega robota. Prostora za izboljšave je veliko, predvsem na področju raznih senzorjev. V prihodnosti bi lahko preizkusili natančnost CO_2- in temperaturnega senzorja, ki sta ključnega pomena pri reševanju. Zelo zanimiva bi bila povezava med reševalnim robotom in dronom za preiskovanje terena. S tem bi omogočili veliko stopnjo avtonomnosti. Menimo, da je prihodnost računalniškega vida v kombinaciji z umetno inteligenco.

9 ZAKLJUČEK

Raziskovalna naloga nam je na začetku predstavljala velik izziv, saj smo vedeli, da bomo porabili veliko časa za učenje novega programskega jezika. Programski jezik smo se začeli učiti od začetka kljub predhodnemu znanju programiranja Atmel mikrokrmilnika. Nov programski jezik je razširil naše znanje o programiranju, ki nam bo v prihodnosti koristilo. Prav tako smo ugotovili, da je računalniški vid zelo uporaben v raznih sistemih. Vendar smo prišli do ugotovitve, da z mikroračunalnikom Raspberry Pi ne bi mogli doseči avtonomne vožnje zaradi premajhne procesorske moči. V prihodnje bomo vgradili sistem računalniškega vida v reševalni robot, ki ga izdelujemo, in izpopolnili zaznavanje s kamero. Računalniški vid je zelo uporaben način zaznavanje okolice, kljub vsemu ni najbolj zanesljiv ob različnih svetlostih in razdaljah. Menimo, da bo zaradi konstantnega razvijanja tehnologije, strojni vid napredoval in mogoče se bodo v prihodnosti resnično pojavili avtonomni roboti, ki bodo zaznavali okolico z računalniškim vidom.

10 VIRI IN LITERATURA

[1] Naravne nesreče (Svetovni splet). (citirano 21. 02. 2019). Dostopno na naslovu: https://siol.net/novice/novice/katere-naravne-nesrece-so-najbolj-smrtonosne-394876

[2] Potresi (Svetovni splet). (citirano 21. 02. 2019). Dostopno na naslovu: http://www.arso.gov.si/potresi/potresna%20aktivnost/potres1998.html

[3] Računalniški vid (Svetovni splet). (citirano 21. 02. 2019). Dostopno na naslovu: https://sl.wikipedia.org/wiki/Ra%C4%8Dunalni%C5%A1ki_vid

[4] Strojni vid (Svetovni splet). (citirano 26. 02. 2019). Dostopno na naslovu: https://sl.wikipedia.org/wiki/Strojni_vid

[5] GOLI, B. Računalniški vid v sistemih varovanja. Diplomsko delo, 2010, str. 8, str. 11, str. 12

[6] LEPEJ, P. Uporaba računalniškega vida za nadzor in upravljanje mobilnih platform. Diplomsko delo, 2009, str. 2, str. 16

[7] To je pet stopenj avtomobilske samostojne vožnje (Svetovni splet). (citirano 21. 02. 2019). Dostopno na naslovu: https://siol.net/avtomoto/zgodbe/to-je-pet-stopenj-avtomobilske-samostojne-voznje-457861

[8] Instagram (Svetovni splet). (citirano 21. 02. 2019). Dostopno na naslovu: https://www.instagram.com/p/BeQ35kcllpR/?utm_source=ig_embed

[9] BRUNAR, S. Upravljanje računalnika s senzorjem Kinect. Diplomsko delo, 2012, str. 3

[10] WEICHARDT, J., CIMPERMAN, Ž. in ŽUNTER, Ž. Strojni vid. Raziskovalna naloga, 2012, str. 13

[11] FABČIČ, S. Priklop perifernih naprav na Raspberry Pi. Diplomsko delo, 2014, str. 7

[12] Picamera (Svetovni splet). (citirano 26. 02. 2019). Dostopno na naslovu: https://picamera.readthedocs.io/en/release-1.13/fov.html

[13] QR-code (Svetovni splet). (citirano 17. 02. 2019). Dostopno na naslovu: https://en.wikipedia.org/wiki/QR_code

[14] Install OpenCV on Raspberry Pi (Svetovni splet). (citirano 19. 02. 2019). Dostopno na naslovu: https://www.pyimagesearch.com/2018/09/26/install-opencv-4-on-your-raspberry-pi/

[15] OpenCV QR code scanner (Svetovni splet). (citirano 19. 02. 2019). Dostopno na naslovu: https://www.pyimagesearch.com/2018/05/21/an-opencv-barcode-and-qr-code-scanner-with-zbar/

[16] What is MobileNet (Svetovni splet). (citirano 19. 02. 2019). Dostopno na naslovu: https://www.quora.com/What-is-MobileNet

[17] Object detection Raspberry Pi (Svetovni splet). (citirano 19. 02. 2019). Dostopno na naslovu: https://www.pyimagesearch.com/2017/10/16/raspberry-pi-deep-learning-object-detection-with-opencv/

[18] Deep learning opencv object detection (Svetovni splet). (citirano 19. 02. 2019). Dostopno na naslovu: https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/