Operacijski sustavi za ugrađena računala

Rasinec, Karlo

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:402108

Rights / Prava: Attribution 3.0 Unported/Imenovanje 3.0

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Rasinec

Operacijski sustavi za ugrađena računala

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Rasinec

Matični broj: 44466/15–R

Studij: Baze podataka i baze znanja

Operacijski sustavi za ugrađena računala

DIPLOMSKI RAD

Mentor:

Izv.prof.dr. sc. Ivan Magdalenić

Varaždin, kolovoz 2022.

Karlo Rasinec

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Karlo Rasinec

_____

Sažetak

Sama obrada teme sadrži nekoliko glavnih cjelina: pronalazak relevantnih operacijskih sustava za različita često korištena ugrađena računala, usporedba pronađenih operacijskih sustava i izrada reprezentativne aplikacije za neko ugrađeno računalo. Sam pronalazak operacijskih sustava se svodi na korištene web izvora, i raznih knjiga sa svrhom pronalazaka različitih operacijskih sustava, od operacijskih sustava opće primjene, do vrlo specifičnih. Usporedba operacijskih sustava se odnosi na detaljnu analizu operacijskih sustava, te uočavanja i dokumentiranja razlika između operacijskih sustava, kao i općenito njihovih glavnih značajki. Zadnja cjelina jest izrada same aplikacije za ugrađeno računalo, specifično radi se o izradi aplikacije za Raspberry Pi ugrađeno računalo. Sama aplikacija će biti izrađena u python programskom jeziku i raditi će se o prepoznavanju registracije. Općenito isprobavanje sustava za prepoznavanje registracije, te pronalazak optimalnih postavki za potrebe rada aplikacije.

Ključne riječi: ugrađeno računalo; ugrađeni sustav; operacijski sustav; Jezgra operacijskog sustava; Prepoznavanje registracije

Sadržaj

Sadržaj iii
1. Uvod 1
2. Ugrađeni sustavi 2
3. Operacijski sustavi 4
3.1. Operacijski sustavi opće namjene 6
3.1.1. Raspbian operacijski sustav 6
3.1.2. Fedora operacijski sustav 7
3.1.3. Usporedba Fedora i Raspbian operacijskih sustava 8
3.2. Operacijski sustavi za mreže senzora 9
3.2.1. TinyOS 9
3.2.2. Tizen operacijski sustav 11
3.2.3. Usporedba TinyOS i Tizen operacijskih sustava 12
3.3. Operacijski sustavi za prikaz multimedijskih sadržaja 13
3.3.1. Rockbox operacijski sustav 13
3.3.2. Slax operacijski sustav 14
3.3.3. Usporedba Rockbox i Slax operacijskih sustava 15
3.4. Operacijski sustavi za ugrađena računala u automobilima 16
3.4.1. Windows embedded automative operacijski sustav 16
3.4.2. QNX Neutrino RTOS 17
3.4.3. Usporedba Windows embedded automative i QNX Neutrino RTOS operacijskih sustava 18
4. Aplikacija za prepoznavanje registracijske oznake 19
4.1. Opis aplikacije 19
4.2. Funkcionalnosti 20
4.2.1. Baza podataka 21
4.2.2. Slikanje registracije 22

4.2.3. Digitalna obrada slike 23
4.2.4. Prepoznavanje teksta s OCR alatom 29
4.2.5. Usporedba teksta 31
4.3. Statistika prepoznavanja registracijske oznake 33
5. Zaključak 35
Popis literature 37
Popis slika 39
Popis tablica 40

1. Uvod

Ovaj rad je napisan na temu: operacijski sustavi za ugrađena računala. Razlog odabira teme je vrlo jednostavan, a to je želja za izradom aplikacije za prepoznavanje registracije, a to se uobičajeno radi na ugrađenom računalu (u ovom radu to je raspberry pi ugrađeno računalo). Ugrađena računala su prisutna u raznim uređajima i često se ih susreće svaki dan. Ugrađena računala su uobičajeno dio većeg sustava koji se naziva ugrađeni sustav, koji zatim mogu biti povezani sa drugim ugrađenim sustavima. Često uz samo ugrađeno računalo se koriste i dodatni uređaji kao što su čitači i senzori.

S tim da je ključni dio teme sami operacijski sustavi, u radu će biti prikazani razni operacijski sustavi, s time da su svi upareni kako bi se mogla napraviti nekakva usporedba između njih. Za ovaj rad odabrani su operacijski sustavi opće namjene, vezani uz mreže senzora, za prikaz multimedijskog sadržaja i za infotainment u automobilima.

Ugrađena računala se nalaze posvuda i imaju jako raznolike uloge, ovisno o sustavu u kojem se nalaze. Neki primjeri gdje se koristi ugrađeno računalo su: automobili, TV-i. svirači uređaji, mikrovalne, frižideri, itd. Iako se koriste na raznim mjestima, nije uvijek potrebno da imaju operacijski sustav, već nekad imaj softver koji upravlja sustavom i izvršava neku jednostavnu funkcionalnost.

Što se tiče već spomenute aplikacije za prepoznavanje registracijskih oznaka, ona je napravljena u python jeziku, na već spomenutom raspberry pi ugrađenom računalu, te se uz samo računalo koristi još i kamera kako bi se mogle slikati registracije. Aplikacija slika auto, prepoznaje registraciju i ako je dozvoljena registracija, otvara rampu (ispisuje tekst, a u stvarnom sustavu šalje signal uređaju koji upravlja podizanjem i spuštanjem rampe).

Tema je široka i mogle se ići u različitim smjerovima, jedan od kojih je prikazan u ovom radu.

2. Ugrađeni sustavi

Ugrađena računala su ključni dio ugrađenih sustava, te će u ovom poglavlju biti kratko opisani ugrađeni sustavi, njihova definicija, podjela i komponente.

Tipična definicija ugrađenih sustava jest da je to sustav koji je kombinacija ugrađenog računala (kompjuterski hardver i softver) i možda nekih dodatnih dijelova koji mogu biti mehanički (rampa) ili elektronski (kamera). Uz ovo definiciju ugrađeni sustavi se mogu opisati i po nekim svojim osobinama [1], [2]:

- Generalno su više ograničeni u pogledu hardvera i softvera od na primjer osobnih računala (ova osobina je istinita za većinu ugrađenih sustava).
- Većinom obavljaju jednu specifičnu funkciju (na primjer dizanje rampe), iako postoje sustavi koji obavljaju više funkcionalnosti, s tim da nije sigurno dali takvi primjeri spadaju u ugrađene sustave.
- Moraju biti visoke kvalitete i pouzdani. Ovo je istina samo za ugrađene sustave koji imaju kritičnu ulogu kao što su ugrađeni sustavi koji se koriste tijekom operacije, ili sustavi koji upravljaju motorom automobila.

Uz to što često obavljaju samo jednu specifičnu funkcionalnost, ugrađeni sustavi su često dio nekog većeg sustava, na primjer dio sustava koji se naziva automobil, koji sadrži mnogo ugrađenih sustava unutar sebe.[1]

U nastavku će biti ukratko nabrojeni tipovi ugrađenih računala po nekim kriterijima.

Ugrađeni sustavi prema izvedbenim i funkcionalnim zahtjevima [3]:

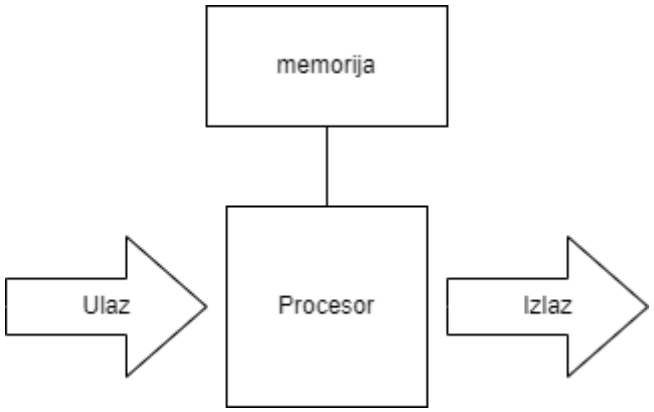
- Ugrađeni sustavi u stvarnom vremenu moraju pružiti rezultat u zadanom vremenskom okviru (na primjer sustav za upravljanje prometa).
- Samostalni ugrađeni sustavi mogu raditi samostalno, te ne ovise o nikakvom poslužitelju, već primaju ulaz (digitalni ili analogni), te pružaju izlaz (eng. Output). Na primjer MP3 player.
- Umreženi ugrađeni sustavi su povezani preko mreže sa nekim poslužiteljem kako bi mogli pružiti izlaz na uređaj s kojim su spojeni (na primjer bankomat).
- Mobilni ugrađeni sustavi su često jednostavni za korištenje, te naravno prenosivi (na primjer digitalna kamera).

Ugrađeni sustavi prema performansama i mikrokontroleru [3]:

- Ugrađeni sustavi malih razmjera imaju 8-bitni ili 16-bitni mirkokontroler. Za napajanje je dovoljna baterija, te imaju vrlo ograničenu memoriju i procesorsku moć, pa su često dio nekog većeg sustava.
- Ugrađeni sustavi srednjeg razmjera imaju 16-bitni ili 32-bitni mikrokontroler, integracija hardvera i softvera je kompleksnija, te sam i time imaju više memorije i veću moć procesiranja.
- Kompleksni ugrađeni sustavi imaju 32-bitni ili 64-bitni mikrokontroler, te služe za izvršavanje velikih i kompleksnih funkcionalnosti, te imaju veliku složenost hardvera i softvera.

Općenito svi ugrađeni sustavi imaju procesor i nekakav softver na sebi, moraju imati stalnu memoriju i memoriju s izravnim pristupom. Također svi ugrađeni sustavi imaju neki oblik ulaza i izlaza (na primjer ulaz može biti kamera, a izlaz LED zaslon). Senzori, čitači, kamere, gumbi i komunikacijski signali su najčešći ulazi u ugrađeni sustav. Komunikacijski signali, zasloni, te promjene u stvarnom svijetu su najčešći izlazi iz ugrađenih sustava. [1]

Slika ispod je osnovni prikaz ugrađenog sustava.



Slika 1: Generički ugrađeni sustav (napravljeno prema slici iz [1])

Ovo je generali prikaz ugrađenog sustava, odnosno sadrži sve osnovne komponente koje ima svaki ugrađeni sustav, što ulaz i izlaz sadrže je različito ovisno o kojem ugrađenom sustavu se radi. Osim hardvera ugrađeni sustavi se razlikuju i prema softveru koji često mora biti izrađen za specifično ugrađeno računalo, te postoji dosta velika vjerojatnost da takav softver ne bi radio na drugom ugrađenom računalu. [1]

Općenito ugrađeni sustavi su sustavi koji obavljaju neku funkcionalnost tako da primaju podatke pomoću ulaza (slika preko kamere, očitavanja senzora, …), obrađuju ih preko softvera, te daju nekakve podatke na izlazu (komunikacijski signal, prikaz na zaslonu, …).

3. Operacijski sustavi

U ovom poglavlju će biti opisane osnove operacijskih sustava, koja je njihova uloga u ugrađenim sustavima, koji su najčešći modeli operacijskih sustava za ugrađene sustave, te će nakon toga biti opisani i uspoređeni operacijski sustavi raznih namjena.

Jedna od definicija operacijskih sustava jest da je riječ o softverskim komponentama koje upravljaju računalnim hardverom. Operacijski sustavi imaju jezgru (eng. Kernel), te imaju pokretački program (eng. Bootloader) koji je potreban kako bi se pokrenula jezgra kod paljenja računala. [4]

Iako ugrađeni sustavi često koriste neki operacijski sustav, on nije obavezni dio samog sustava. Operacijski sustav može biti korišten samo na procesorima koje podržava, tako da odabir operacijskog sustava ovisi o tome koje se ugrađeno računalo koristi, odnosno koji procesor. Operacijski sustav u ugrađenim računalima ima dvije glavne svrhe: predstavlja apstrakcijski sloj za softvere koji su iznad operacijskog sustava (tako da bi taj softver manje ovisio o hardveru), čime se postiže jednostavniji razvoj aplikacija. I druga svrha jest upravljanje hardverskim i softverskim resursima kako bi sustav ispravno radio.[2]

Svi operacijski sustavi imaju jezgru, a to je program koji upravlja resursima računalnog sustava, odnosno alocira resurse između različitih procesa na kontrolirani način. Pri čemu procesi mogu biti pokrenuti od strane različitih korisnika, pa se radi i o alociranju resursa između korisnika. Različiti kod ima različitu razinu pristupa, pri čemu kod jezgre ima pristup hardverskim komponentama, odnosno pristup hardveru kontrolira jezgra. [4]

U nastavku će biti navedene i ukratko opisane funkcionalnosti jezgre [4], [2]:

- Upravljanje procesima: računalni sustav u sebi ima pohranjene neke programe, kada se ti programi pokrenu, oni postaju procesi. Naravno u računalnom sustavu moguće je izvođenje više procesa odjednom, pri čemu to može biti ostvareno tako da svaki proces dobi dio procesorskog vremena (pa se čini kao da se izvršavaju istovremeno) ili ako procesor ima više jezgra, mogu se izvršavati dva ili više procesa istovremeno, svaki na jednoj jezgri. To je ujedno i uloga sustava upravljanja procesima, kada dodijeliti kojem procesu pristup resursima procesora.
- Upravljanje memorijom: memorijski prostor računala dijele svi procesi, tako da je potrebno upravljati pružanjem pristupa i alociranjem memorijskog prostora, a time se bavi upravljanje memorijom.

- Datotečni sustav: trajni podaci se pohranjuju u datoteke koje su organizirane po direktorijima. Datotečni sustav pruža funkcionalnosti i strukture podataka potrebne za upravljanje datotekama i direktorijima koji se nalaze unutar neke particije diska.
- Upravljanje uređajima: računalni sustav ima više različitih perifernih uređaja (I/O uređaji, mrežna sučelja, ...). Upravljački programi uređaja (eng. Drivers) su dijelovi softvera unutar jezgre koji su odgovorni za upravljanje uređajima.

Svi operacijski sustavi imaju jezgru, ali uz nju često imaju i druge sistemske softverske komponente (kao na primjer upravljački programi uređaja i srednji softver (eng. Middleware)). Naravno nemaju svi operacijski sustavi te komponente. Većina se ugrađenih operacijskih sustava bazira na jednom od sljedeća tri modela: monolitski, slojeviti i klijent-server model (mikrojezgra). Ti modeli se razlikuju po dizajnu jezgre kao i drugim sistemskim softverima. [2]

U monolitskom operacijskom sustavu, upravljački programi uređaja i srednji softver su dio jezgre. Radi se o jednoj izvršnoj datoteci koja sadrži sve komponente jezgre. Ovakve operacijske sustave je često teže skalirati, modificirati i debagirati u odnosu na druge operacijske sustave, a razlog tome je što su veliki, integrirani, te imaju veliku razinu međuovisnosti. Zato postoji algoritam koji poboljšava i olakšava rad s takvim operacijskim sustavom. Kao što je već prije bilo spomenuto sve funkcionalnosti su integrirane u jednu izvršnu datoteku koja se sastoji od više modula, od kojih svaki sadrži kod vezan u neku funkcionalnost operacijskog sustava. Primjer takvog operacijskog sustava jest linux-ov ugrađeni operacijski sustav. [2]

U slojevitom operacijskom sustavu, sustav je podijeljen na hijerarhijske slojeve, gdje su gornji slojevi ovisni o funkcionalnostima nižih slojeva. Kao i kod monolitskih operacijskih sustava, slojeviti operacijski sustavi su sadržani u jednoj velikoj datoteci, ali su jednostavniji za razvijati. Negativna strana je to da API-ji svakog sloja kreiraju dodatne troškove u obliku veće veličine datoteke i sporijih performansi. Primjer ovakvog operacijskog sustava je VRTX operacijski sustav. [2]

Operacijski sustav tipa klijent-server sadrži samo upravljanje procesima i upravljanje memorijom unutar svoje jezgre (a neki sadrže samo upravljanje procesima). Ostale funkcionalnosti su izvađene iz jezgre, te je način upravljanja procesima nešto drugačiji nego kod drugih modela operacijskih sustava. Ovakav model je tipično više skalabilan, lakši za debugiranje, a dodatne komponente koje su potrebne mogu biti dinamički dodane. Nedostaci su to što može biti sporiji od drugih oblika operacijskih sustava, te nastaju dodani troškovi kod mijenjanja između jezgre, drugih komponenti operacijskog sustava i komponenti koje nisu dio operacijskog sustava. Ovo je najzastupljeniji model, te je QNX primjer ovakvog operacijskog sustava. [2]

Sada slijede potpoglavlja sa primjerima operacijskih sustava za ugrađena računala.

3.1. Operacijski sustavi opće namjene

U ovom poglavlju će biti opisana i nakon toga uspoređena dva operacijska sustava opće namjene koji se mogu koristiti u ugrađenim računalima, preciznije dva koja će biti opisana ispod se najčešće koriste za raspberry pi ugrađeno računalo. Oba operacijska sustava su bazirana na linux distribucijama.

3.1.1. Raspbian operacijski sustav

Raspbian je operacijski sustav baziran na Debian operacijskom sustavu. Rasbian je optimiziran za raspberry pi ugrađeno računalo, odnosno optimiziran da efektivno koristi i upravlja hardverom tog ugrađenog računala. Raspbian sadrži prilagodbe koje su dizajnirane kako bi olakšale korištenje raspberry pi ugrađenog računala, te također dolazi s mnogo softverskih paketa koji se mogu odmah koristiti. [5], [6]

Raspbian dijeli većinu značajki s Debian operacijskim sustavom, uključujući i Debian-ov repozitorij softverskih paketa (takvih paketa ima mnogo, više od 35000). [6]

Debian je GNU/Linux i GNU/kFreeBSD distribucija. Radi se o kompletnom operacijskom sustavu, uključujući softvere i sisteme za instalaciju i upravljanje, koji su bazirani na Linux (monolitska jezgra) ili FreeBSD (monolitska jezgra sa dinamičnim učitavanjem modula) jezgri i otvorenim softverima. [7]

Većina koda Raspbian operacijskog sustava je otvorena, što znači da se može slobodno mijenjati i modificirati. Raspbian koristi linux jezgru što znači da se radi o monolitskom modelu. [6]

Raspbian se satoji od sljedećih ključnih komponenti: raspberry pi pokretački program, linux jezgra, demoni (eng. Daemons), ljuska, pomoćni programi ljuske, X.org grafički poslužitelj i radna površina. [6]

Raspberry pi pokretački program služi za inicijalizaciju hardvera u poznato stanje, a zatim učitava linux jerzgru. Postoje pokretački programi prve i druge razine. Pokretački programi prve razine se nalaze u ROM memoriji ugrađenog računala i ne mogu se mijenjati, dok se pokretački programi druge razine nalaze na SD kartici i moguće ih je mijenjati. [6]

Linux jezgra upravlja svim operacijama raspberry pi ugrađenog računala (na primjer prikazivanje teksta na ekranu). Da bi se mogao koristiti neki hardverski uređaj, jezgra mora znati koji je i kako ga koristiti, srećom veliki broj uređaja je podržan u linux jezgri. Jednom

kada je jezgra učitana ona pokreće program init koji dovršava inicijalizaciju ugrađenog računala, te nakon toga učitava sve demone u pozadinu, i nakon toga učitava grafičko sučelje. [6]

Demoni su dijelovi softvera koji pružaju operacijskom sustavu različite značajke. Primjer demona je Apache web server. [6]

Nakon pokretanja demona, init pokreće ljusku koja predstavlja sučelje ugrađenog računala koje omogućuje praćenje i kontrolu ugrađenog računala. Preko ljuske se mogu pokretati skripte, te ljuska predstavlja bitnu komponentu samog operacijskog sustava. [6]

Pomoćni programi ljuske su zapravo naredbe koje se mogu koristiti unutar ljuske za na primjer kopiranje datoteka, instalaciju softvera i slično. [6]

Uloga X.org-a jest da pruža zajedničku platformu za izgradnju grafičkog korisničkog sučelja. Odgovoran je na primjer za pomicanje miša (pokazivača) po ekranu. [6]

Radna površina omogućuje oblik interakcije različit od tipkovnice, odnosno interakcija preko ikona.[6]

Općenito Raspbian operacijski sustav je napravljen za raspberry pi ugrađeno računalo, pa se često koristi kao operacijski sustav tog ugrađenog računala. Jednostavno je za koristiti (pogotovo ako korisnik ima iskustva s linux operacijskim sustavima) i dolazi sa mnogo korisnih softverskih paketa koji su unaprijed instalirani, te se praktički odmah može početi s razvojem aplikacije.

3.1.2. Fedora operacijski sustav

Fedora operacijski sustav je baziran na linuxu, a kreiran je od srane Red hat organizacije, te predstavlja njihov besplatni operacijski sustav koji je napravljen kako bi pratio brze promjene u svijetu softvera baziranog na otvorenom kodu, dok istovremeno predstavlja dobro podržanu distribuciju linuxa. Fedora sadrži više od 15000 softverskih paketa u svojem repozitoriju, a neke mogućnosti koje ti paketi nude su: spajanje na LAN i Internet, kreiranje dokumenata i objava istih na Internetu, rad s multimedijskim sadržajem, igranje računalnih igrica, itd. Fedora operacijski sustav se često ažurira kako bi podržavao najnovije hardverske komponente i druge uređaje. Cijela poanta Fedora operacijskog sustava jest da bude ažuran i da pruža programerima pristup najnovijem softverskim značajke i popravcima, odnosno nastoji biti u koraku sa svim izmjenama vezanima uz linux. [8]

U nastavku će biti ukratko opisane neke značajke Fedora operacijskog sustava.

Kao što je već bilo spomenuto Fedora podržava najnovije linux tehnologije, te ima dobro dokumentirani proces instalacije. [8]

Koristi RPM upravitelj paketima koji omogućava da korisnici koji nisu pretjerano tehnološki sposobni, mogu relativno lagano instalirati linux softvere. Također su upakirani dodatni alati za još jednostavniji pristup softverskim paketima (na primjer PackageKit). [8]

Podržava Unix sistem V mehanizam za pokretanje i zaustavljanje servisa, ovakav sustav javlja korisniku u obliku poruka, koji servisi su se uspješno pokrenuli, a koji nisu. [8]

Naravno kao i Raspbian, Fedora ima podršku za radnu površinu, pri čemu generalno nudi dvije različite opcije: GNOME i KDE. [8]

GUI alati za administraciju koji olakšavaju podešavanje servisa (na primjer vezanih uz korisnika ili datotečni sustav ili sigurnost ili umrežavanje), te također nudi automatsko kreiranje konfiguracijskih datoteka. [8]

Sav softver koji dolazi u sklopu sa operacijskim sustavom je detaljno testiran, te se softveri koje Fedora koristi često ažuriraju, te sam operacijski sustav nudi jednostavan način selekcije, preuzimanja i instalacije ažuriranih softverskih paketa (koristi yam objekt (eng. Facility) [8]

S obzirom da se radi o linux distribuciji, Fedora koristi linux jezgru koja je bazirana na monolitskom modelu. Fedora operacijski sustav je popularan zato što nudi najnovije linux tehnologije i softverske pakete, te se često ažurira kako bi se kontinuirano pratile promjene u svijetu softvera baziranog na otvorenom kodu.

3.1.3. Usporedba Fedora i Raspbian operacijskih sustava

U ovom potpoglavlju će biti uspoređena dva operacijska sustava opće namjene: Fedora i Raspbian. Oba operacijska sustava se mogu koristiti kao operacijski sustavi za raspberry pi ugrađeno računalo, te će se uspoređivati s tim na umu.

Oba dva sustava su linux distribucije, odnosno oba imaju linux jezgru koja je monolitska. Rasbian je napravljen za raspberry pi ugrađeno računalo, dok se Fedora operacijski sustav može koristiti u mnogo različitih računala, te ne podržava sve verzije raspberry pi ugrađenog računala.

S obzirom da je Rasbian baziran na Debian operacijskom sustavu, osim onog što je već iznad napomenuto, ova usporedba će se svesti na usporedbu između Debiana i Fedore.

Prva i osnovna razlika između njih je to što je Debian napravljen za korisnike koji žele stabilnu distribuciju sa najboljom softverskom i hardverskom podrškom, dok je Fedora napravljene za korisnike koji žele koristiti najnovije linix softvere. [9]

Što se tiče softverskih paketa, oba operacijska sustava imaju veliki repozitorij softverskih paketa, ali Debian i dalje ima znatno više softverskih paketa. Iako Fedora ima podršku za stari hardver, generalno je Debian bolji u tome dijelu. Fedora generalno troši manje hardverskih resursa. Debian ima bolju plaćenu podršku kada nastane problem, te je generalno stabilnija distribucija od Fedore (to je zato što se Fedora često mijenja), dok Fedora ima aktivniju zajednicu korisnika. Oba dva operacijska sustava su dobro dokumentirana. [9]

Općenito usporedba ovih operacijskih sustava se može svesti na to što tražimo od njih. Rasbian je napravljen za raspberry pi, radi čega ga je vrlo lagano postaviti, te podržava sve verzije ugrađenog računala, s druge strane Fedora nije specifično namijenjena za to ugrađeno računalo, ali nudi najnovije linux tehnologije. Rasbian ima veći repozitorij softverskih paketa, te je generalno lakši za koristiti, odnosno za korištenje Fedore se očekuje neka razina predznanja o korištenju linuxa. Oba sustava podržavaju radnu površinu i mogu se koristiti za izradu aplikacija. Ako treba jednostavan operacijski sustav kojeg je lagano postaviti i koristiti s raspberry pi ugrađenim računalom, Rasbian je vjerojatno bolji izbor. S druge strane ako su za razvoj potrebne najnovije linux tehnologije i ako postoji dovoljno predznanje o linuxu, Fedora je vjerojatno bolji izbor.

3.2. Operacijski sustavi za mreže senzora

U ovom potpoglavlju će biti opisana dva operacijska sustava koji se mogu koristiti za upravljanje mrežom senzora, jednom od ta dva to je i primarna uloga, dok se drugi može koristiti i u druge namjene. Prvi je TinyOS koji je dizajniran specifično za bežične senzore, a drugi je Tizen koji je specijaliziran za pametne uređaje, te njihovu međusobnu komunikaciju.

3.2.1. TinyOS

TinyOS je lagani (eng. Lightweight) operacijski sustav specifično dizajniran za bežične senzore s niskom potrošnjom (eng. Low-power). To znači da je napravljen s fokusom na rad sa vrlo malo snage. Napravljen je da može raditi na mikrokontrolerima, te ima vrlo agresivan sustav za uštedu energije. Radi se o fleksibilnom, aplikacijski-specifičnom operacijskom sustavu koji se koriste za mreže senzora, odnosno to mu je primarna namjena. TinyOS nije operacijski sustav u tradicionalnom smislu, već se radi o programskom okviru za ugrađene

sustave, koji omogućava izgradnju aplikacijski-specifičnih operacijskih sustava u svaku aplikaciju. [10], [11]

U nastavku će biti opisana četiri zahtjeva koji su motivirali dizajn TinyOS-a [10]:

- Limitirani resursi: bežični senzori imaju jako limitirane fizičke resurse, jer se radi o malim uređajima, koji su jeftini i koji moraju koristiti malo energije. Oni imaju procesor sa snagom procesiranja od 1-MIPS, te nekoliko desetaka kilobajta memoriju.
- Reaktivna konkurentnost: u tipičnoj aplikaciji za mrežu senzora, svaki čvor(senzor) mjeri nešto u svojoj okolini, obavlja lokalno procesiranje, prenosi podatke, usmjerava podatke drugih čvorova, te sudjeluje u izvršavanju distribuiranih zadataka vezanih uz procesiranje.
- Fleksibilnost: potreban je fleksibilan operacijski sustav koji je specifičan za svaku aplikaciju, kako bi se smanjila potrebna količina memorije i smanjila potrošnja energije. Također treba biti i neovisan o granici između hardvera i softvera.
- Niska potrošnja: S obzirom na zahtjeve vezane uz veličinu i cijenu senzora, nužan je operacijski sustav koji ima nisku potrošnju energije, odnosno resursa.

Primarna uloga TinyOSa je olakšati izgradnju aplikacija za mreže senzora. S tim na umu TinyOS pruža tri stvari kako bi se olakšao razvoj takvih aplikacija [11]:

- Komponenti model koji definira kako pisati malene, ponovno iskoristive dijelove koda, te kako ih sastaviti u veće abstrakcije. Ovakvi dijelovi koda eksplicitno definiraju svoje ovisnosti.
- Model istovremenog izvođenja definira kako komponente isprepliću svoje izračune, te kako međusobno komuniciraju prekidni i bez prekidni kod. Ovakav model mora podržavati komponente koje djeluju u isto vrijeme, a moraju koristiti malu količinu RAM-a.
- Pruža aplikacijsko programsko sučelje, servise, biblioteke komponenti i ukupnu strukturu komponenti koja olakšava pisanje novih aplikacija i servisa. TinyOS ima set API-a za opće funkcionalnosti (slanje paketa, čitanje senzora, …).

Općenito TinyOs je aplikacijski-specifičan operacijski sustav čija je primarna namjena da bude korišten za izradu aplikacija koje služe za upravljanje mrežom senzora. Radi se o vrlo laganom operacijskom sustavu, koji mora takav biti radi ograničenja hardverskih resursa.

3.2.2. Tizen operacijski sustav

Tizen je otvorenog koda, te je među-arhitektonski operacijski sustav baziran na linuxu, primarno podržan od strane Samsunga, ali i od strane drugih mobilnih proizvođača i proizvođača pametnih uređaja. Tizen je napravljen da može povezati sve pametne uređaje: pametne televizore, pametne telefone, nosive pametne uređaje (na primjer pametan sat), IoT uređaje, itd. Tizen nudi efektivno povezivanje i komunikaciju između različitih uređaja. [12], [13]

Korisnici Tizen operacijskog sustava mogu slobodno mijenjati softver svojeg uređaja jer je Tizen otvorenog koda, te je jedna od glavnih značajki Tizen operacijskog sustava to što je kompatibilan s više mobilnih platformi, odnosno aplikacije napravljene u Tizen operacijskom sustavu, mogu relativno jednostavno biti promijenjene da rade na drugim operacijskim sustavima (na primjer Android). Tizen također dozvoljava razvoj raznolikih aplikacija koje mogu biti pokrenute na različitim uređajima. Tizen je napravljen kako bi korisnici prilagodili njegove značajke svojim potrebama, odnosno potrebama tržišta. [12]

S obzirom da Tizen radi sa različitim uređajima, za svaki od tih postoji i profil: TV, mobilni, IoT i nosivi uređaji. Svi profili su građeni na zajedničkoj, dijeljenoj infrastrukturi koja se zove Tizen common, radi toga se mogu jednostavnije dodati profili za nove vrste uređaja. S obzirom da jedan korisnik može imati više pametnih uređaja, Tizen omogućava spajanje tih uređaja. [13]

Tizen podržava razne IoT uređaje i razvoj na njima, a jedan od tih je i raspberry pi ugrađeno računalo. Tizen dozvoljava izgradnju vlastite IoT slike (eng. Image) kombinacijom API blokova, što dozvoljava korisniku da uzme samo ono što treba, naravno da postoji neki dio koji predstavlja jezgru, te je obavezan za sve slike. [13]

Tizen se može koristiti za upravljanje senzorima i primanje podataka od senzora, a za to koristi svoj senzor API koji uključuje [14]:

- Slušatelj (eng. Listener) senzora: pomoću njega se mogu detektirati i nadzirati dostupni senzori, slušatelj reagira na registrirane događaje senzora, te podatke dostavlja aplikaciji u prethodno definiranim intervalima. Naravno moguće je imati više slušatelja odjednom.
- Ručka (eng. Handle) senzora: omogućuje pristup hardverskim podacima senzora (na primjer ime, tip, raspon mjerenja, učestalost mjerenja, …)
- Tipovi senzora: Tizen podržava samo određene senzore, s tim da podržanih senzora ima mnogo, a neki od njih su: brzinomjer, senzor vlažnosti, senzor osvjetljenja, senzor pritiska, itd.

- URI za senzore: opći oblik URI-a jest `http://<vendor> /sensor/ <category>/ <sensor-type>/<sensor-name>` pri čemu `<vendor>` mora bit tizen.org, `<category>` može biti `general` ili `healthinfo`, `<sensor-type>` je tip senzora (na primjer `light`), i na kraju `<sensor-name>` element, koji nije obavezan (Ako ga nema tada se odnosi na sve senzore određenog tipa), a predstavlja ime senzora.

Općenito Tizen je operacijski sustav koji se može koristiti na bilo kojem pametnom uređaja, te je napravljen da bude fleksibilan i otvoren, kako bi ga programeri koji ga koriste mogli prilagodi svojim potrebama i potrebama svojih korisnika.

3.2.3. Usporedba TinyOS i Tizen operacijskih sustava

U ovom potpoglavlju će biti uspoređena dva operacijska sustava koji mogu upravljati mrežom senzora, a ti operacijski sustavi su TinyOS i Tizen.

Prva i vrlo velika razlika je to što TinyOS nije operacijski sustav u tradicionalnom smislu, već služi za kreiranje operacijskih sustava specifičnih za pojedine aplikacije koje ga koriste, s druge strane Tizen je operacijski sustav baziran na linuxu.

Oba dva operacijska sustava se mogu koristit na uređajima sa limitiranim hardverom, ali TinyOS je napravljen specifično za uređaje koji moraju imati malu potrošnju, a sam i time slabe procesore i memoriju.

Primarna svrha TinyOS je da upravlja mrežom bežičnih senzora, najčešće u svrhu prikupljanja podataka i izvlačenja statistike iz tih podataka, s druge strane Tizen bi vjerojatno uz samo prikupljanje mogao koristiti te podatke za donošenje odluka i upravljanje pametnim uređajima, na primjer kada korisnik uđe u kuću i prođe pokraj senzora za pokret, uređaj koji prati taj senzor uspostavlja komunikaciju s pametnim TV-om u svrhu pokretanja TV-a na korisnikov omiljeni kanal. Naravno ako bi koristili raspberry pi i skupinu bežičnih senzora, Tizen bi također mogao obavljati sličnu svrhu kao i TinyOS, samo što bi uz mrežu senzora postojala i centralna jedinica za prikupljanje i obradu podataka. TinyOS može podijeliti neki zahtjevni zadatak među senzorima u mreži, pa možda i nije potrebna centrala jedinica za obradu, već samo vjerojatno za pohranu podataka, ako i to.

S obzirom na to da TinyOS kreira poseban operacijski sustav za svaku aplikaciju (iako postoje moduli koji se mogu ponovno koristiti između aplikacija), vrlo je vjerojatno kompleksniji za naučiti i koristiti. TinyOS koristi nesC (network embedded systems C) programski jezik koji je ekstenzija c jezika bazirana na komponentama koje su međusobno spojene. Tizen je baziran na linuxu, te izgleda da se kod piše u c jeziku, pa također

vjerojatno nije najjednostavniji za naučiti, ali na prvi pogled kod izgleda relativno jednostavno.

Ova usporedba se svodi na to što se zapravo želi postići. Ako se želi ostvariti mreža bežičnih senzora koji imaju vrlo nisku potrošnju i mogu raditi dugo vremena bez potrebe za nekakvim popravcima ili zamjenama baterija, tada je TinyOS vjerojatno bolji izbor, a ako treba kreirati nekakvo pametno okruženje koje se sastoji od senzora i pametnih uređaja, tada je Tizen vjerojatno bolji izbor.

3.3. Operacijski sustavi za prikaz multimedijskih sadržaja

U ovom potpoglavlju će biti opisana dva operacijska sustava koji se mogu koristiti za prikaz multimedijskog sadržaja, te će nakon toga biti uspoređeni. Ta dva operacijska sustava su Rockbox i Slax, pri čemu Rockblox primarno služi za prikaz multimedijskih sadržaja, a Slax je praktički mini linux distribucija.

3.3.1. Rockbox operacijski sustav

Rockbox je firmware/operacijski sustav koji služi za prikaz multimedijskog sadržaja na raznim uređajima koji su namijenjeni za tu svrhu. Osim što je operacijski sustav, Rockbox također ima i aplikaciju za prikaz multimedijskog sadržaja, koja ima minimalnu konekciju sa operacijskim sustavom, kako bi ju bilo lakše prebaciti na druge arhitekture. Sam operacijski sustav je ovisan o arhitekturi uređaja, te je podijeljen na dva dijela, kod vezan uz logički upravljački program i bit twiddling niske razine (ovisan o arhitekturi). Trebalo bi u budućnosti napraviti API za nisku razinu, kako bi prebacivanje na nove arhitekture bilo jednostavnije. [15]

Rockbox podržava višedretvenost te ima svoj vlastiti kooperativni višedretveni planer (eng. Scheduler). Ovaj dio postoji samo na operacijskom sustavu, aplikacija ima samo jednu dretvu. Unutar operacijskog sustava postoji više funkcija koje su podijeljene u iduće dretve [15]:

- ATA dretva (uređaji koji imaju tvrdi disk): za rukovanje rotacijom tvrdog diska.
- Dretva pozadinskog osvjetljenja: paljenje i gašenje pozadinskog osvjetljenja.
- MMC dretva (uređaji koji podržavaju flash): dozvoljava promjenu kartice dok je uređaj upaljen.
- Dretva reproduciranja: upravlja niskom razinom snimanja/reprodukcije.
- Dretva napajanja: odgovorna za upravljanje napajanjem i punjenje uređaja.
- Dretva pomicanja: odgovora za pomicanje sadržaja gore i dolje po ekranu.

- USB dretva: procesiranje događaja umetanja i vađenja USB uređaja.

Naravno postoji komunikacija među dretvama kako bi cijeli sustav mogao funkcionirati.

Rockbox jezgra nije bazirana na nekoj od postojećih, već je napisana od nule. S obzirom da većina digitalnih audio svirača koje Rockbox podržava nema MMU (Memory Menagement Unit), sama Rockbox jezgra nema nikakvu metodu za dinamičko alociranje metode, već se sva memorija mora alocirati statički. Najveći razlog tome je što se time ušteđuje na resursima, a to je bitno jer Rockbox često radi na platformama koje su memorijski ograničene. Rockbox jezgra koristi dretveni model i to kooperativni oblik. To znači da nakon što je procesor dodijeljen dretvi, ona sama odlučuje kada će prestati koristi procesor i omogućiti planeru da dodijeli procesor nekoj drugoj dretvi. Ovakav pristup se smatra zastarjelim i trebalo bi ga promijeniti. [16]

U nastavku će biti nabrojene neke značajke koje Rockbox nudi [16]:

- Označavanje knjižnih oznaka (eng. Bookmarking): Najviše se koristi kod audio knjiga, kako bi korisnik mogao nastaviti slušati tamo gdje je stao
- Mogućnost određivanja koje informacije se ispisuju na ekranu
- Sustav povratne informacije glasom: za korisnike koji imaju slabi vid
- Reproduciranje filmova, te mnoge druge značajke

Rockbox je specijaliziran operacijski sustav koji je namijenjen za uređaje koji služe za reproduciranje multimedijskog sadržaja, te mu je to glavna uloga i ne pokušava biti ništa druga, iako je na primjer moguće igrati igricu Doom na Rockbox operacijskom sustavu.

3.3.2. Slax operacijski sustav

Slax je lagani operacijski sustav, koji je linux distribucija bazirana na Slackwareu i Debianu. Razlog zašto je dizajniran da bude lagani jest taj da se često postavlja na USB uređaje, iako se može postaviti i na tvrde diskove i SD kartice. Sam operacijski sustav nije besplatan, već se mora kupiti (radi se o jednokratnoj kupovini). S obzirom da se može postaviti na USB uređaje, Slax je poprimio naziv džepni operacijski sustav. Iako je lagani operacijski sustav Slax nudi podršku za reproduciranja multimedijskih sadržaja, ali nisu podržani svi formati. Uz to Slax nudi povezivanje na Internet, te instalaciju raznolikih modula koje je vrlo jednostavno nadodati na postojeći operacijski sustav. Slax ima i podršku za radnu površinu, koja iako izgleda relativno jednostavno, služi svojoj svrsi.[17]

S obzirom da je linux jezgra već nekoliko puta bila opisana, te je i Debian bio opisan u jednom djelu, ovdje se neće ulaziti dublje u način rada operacijskog sustava.

Sve u svemu Slax je još jedna distribucija linuxa kojoj je glavna svrha biti lagana i lagano prenosiva, a istovremeno imati sve bitne funkcionalnosti velikih operacijskih sustava, uz mogućnost nadogradnje pomoću modula.

Slax se može koristiti na raspberry pi ugrađenom računalu, ali se može instalirati i na uređaje poput MP3 svirača, te je to i razlog zašto je dio ovog poglavlja.

3.3.3. Usporedba Rockbox i Slax operacijskih sustava

U ovom potpoglavlju će biti uspoređeni operacijski sustavi Rockbox i Slax, i to primarno na njihov mogućnosti za prikaz multimedijskog sadržaja. Rockbox je napravljen za tu funkciju, dok je Slax džepna distribucija linuxa.

Rockbox ima jezgru koja je bila kreirana iz nule, te se radi o dretvenoj jezgri kooperativnog tipa (koja je malo zastarjela), dok Slax koristi linux jezgru. Rockbox je besplatan za koristiti, a Slux se plaća, što može biti ograničavajući faktor.

Oba dva operacijska sustava mogu reproducirati multimedijski sadržaj, ali čini se da Rockbox podržava više različitih formata od Slaxa, što je i za očekivati, jer mu je to primarna uloga. S obzirom na informacije na webu, čini se da Rockbox ne može pristupiti Internetu kako bi se mogli preuzeti novi multimedijski sadržaji, već izgleda da ono što se postavi na uređaj na kojem je Rockbox (možda preko USB-a ili MicroSD karitce) je jedino što je dostupno. Sad dali je to problem ovisi za što se koristi, korisnik može staviti sve pjesme na uređaj i biti zadovoljan s time, ili staviti sve filmove koje misli gledati tijekom nekog puta. Naravno to znači da se preko Rockboxa ne može pristupati djeliteljima multimedijskog sadržaja koji se nalaze na Internetu. S druge strane Slax podržava bežično spajanje na Internet, i sam i time ima pristup multimedijskom sadržaju koji je tamo dostupan. Ujedno je to vjerojatno i ključna razlika radi koje bi se odabrao jedan ili drugi operacijski sustav. Ako je potreban operacijski sustav koji podržava razne formate multimedijskih sadržaja, te ako nije ključan pristup Internetu, Rockbox je vjerojatno bolji odabir. A ako nije bitno da budu podržani razni formati multimedijskog sadržaja, a bitno je da je moguć pristup Internetu kako bi se mogli konzumirati multimedijski sadržaji koji su tamo dostupni, tada je Slax vjerojatno bolji odabir.

3.4. Operacijski sustavi za ugrađena računala u automobilima

U ovom potpoglavlju će biti opisana dva operacijska sustava koji se koriste u ugrađenim sustavima u automobilima. Prvi je Windows embedded automative, a drugi je QNX Neutrino RTOS. Oba sustava služe za infotainment, te će biti opisani i nakon toga uspoređeni.

3.4.1. Windows embedded automative operacijski sustav

Windows embedded automative (u nastavku WEA) je operacijski sustav baziran na Microsoftovom ugrađenom operacijskom sustavu, te je primarno dizajniran za razvoj infotainment (kombinacija informativnih sadržaja i zabave) sustava. Radi se o platformi za izgradnju rješenja koja omogućuju komunikaciju, zabavu i usluge bazirane na lokaciji. Sam operacijski sustav dolazi uz set integriranih, testiranih i fleksibilnih srednjih softvera, uz sve druge komponente koje inače nudi Microsoftov ugrađeni operacijski sustav (Windows Embedded Compact, u nastavku WEC). Sam WEA operacijski sustav se može koristiti u raznim automobilima, odnosno u automobilima raznih marki i modela.[18]

S obzirom da je WEA baziran na WEC, on koristi značajke tog operacijskog sustava [19]:

- Mali otisak (eng. Small footprint): ovisno o izboru komponenata, na primjer unutar 300KB može biti sadržano 700 komponenta.
- Deterministična operacijski sustav strogo u realnom vremenu: determinističko vrijeme odgovora.
- Mogućnost povezivanje na Internet.
- Podrška za dvojezgrene procesore.
- Unificirana jezgra, odnosno jezgre u jednoj datoteci.
- Podrška za razne jezike.
- Microsoftov .NET kompaktni okvir
- …

WEA na već postojeće značajke koje dolaze u sklopu sa WEC operacijskim sustavom, dodaje značaje specifične za automobile kao što je na primjer Auto System Tools (AST). [18]

Još će ukratko samo biti spomenuti srednji softveri i usluge koje dolaze zajedno sa operacijskim sustavom. Neke od tih usluga su telefonija i podatkovna komunikacija, bluetooth mogućnosti, korištenje telefona bez ruku (eng. Hands-free phone), zabava, upravljanje uređajima, itd. [18]

Kao što je i na početku bilo spomenuto radi se o operacijskom sustavu za infotaiment, koji nudi mnogo mogućnosti za ostvarivanje svojeg cilja, od spajanja preko bluetootha do mogućnosti pristupa Internetu, pregled multimedijskih sadržaja, itd. Što se tiče jezgre operacijskog sustava, čini se da WEA operacijski sustav koristi hibridnu jezgru, kombinacija monolitske i mikrojezgre.

3.4.2. QNX Neutrino RTOS

QNX Neutrinos RTOS (u nastavku QNX) je operacijski sustav koji radi u realnom vremenu, te osigurava dobru kombinaciju performansi, sigurnosti i pouzdanosti u kritičnim sustavima (jedan od kojih je automobilski). [20]

Sama jezgra operacijskog sustava koristi mikrojezgrenu arhitekturu, koja dozvoljava da sve aplikacije, sustavi datoteka, upravljački programi uređaja budu pokrenuti izvan jezgre u zasebnom memorijski zaštićenom adresnom prostoru. Time se osigurava da sustav ne bude ugrožen ako u nekim komponentama dođe do greške. Ako neka aplikacija padne, ona se automatski ponovno pokreće, bez potrebe za ponovnim pokretanjem cijelog sustava. [20]

QNX također nudi dinamičko dodavanje i uklanjanje komponenti, bez da to utječe na druge dijelove sustava, što povećava fleksibilnost i prilagodljivost korisniku. [20]

Što se tiče značajki koje nudi, odnosno koje dolaze s operacijskim sustavom, one su vrlo slične kao i kod WEA operacijskog sustava. Neke od njih su: Okvir za multimedijske sadržaje, bluetooth, GPS, prepoznavanje glasa, pristup Internetu, itd. Uz to QNX podržava višejezgrene procesore i 3D grafiku. [20]

I ovaj operacijski sustav se može koristiti za infotaiment u ugrađenom sustavu automobila, te se čini da nudi slične mogućnosti kao i WEA operacijski sustav, s time da stavlja poseban fokus na pouzdanost i toleranciju na greške.

3.4.3. Usporedba Windows embedded automative i QNX Neutrino RTOS operacijskih sustava

U ovom poglavlju će biti uspoređeni dva operacijska sustava koji se mogu koristit u ugrađenim sustavima automobila i to u svrhu infotaimenta. Prvi je WEA koji je napravljen od strane Microsofta, a drugi je QNX koji je napravljen od strane QNX softver sistema, a pruža ga BlackBerry organizacija.

Oba dva operacijska sustava rade u realnom vremenu, te se mogu koristiti za infotainment, ali se QNX može koristiti i u druge svrhe, te u drugačijim ugrađenim sustavima (na primjer medicina) kod kojih je bitno da sustav radi pouzdano i bez grešaka.

Operacijski sustavi se razlikuju i po jezgrama, pri čemu WEA ima hibridnu jezgru, a QNX čistu mikrojezgru, s kojom se nastoji smanjiti mogućnost da dođe do pada sustava.

Što se tiče samih mogućnosti, čini se da oba operacijska sustava nude manje-više iste mogućnosti, pri čemu jedna od razlika bi bila to što WEA podržava dvojezgrene procesore, a QNX višejezgrene, odnosno sa više od dvije jezgre.

Kroz pretraživanje Interneta dolazi se do zaključka da je QNX pouzdaniji sustav koji se rjeđe kvari, te definitivno stavlja veliki fokus na taj dio.

Oba sustava se plaćaju, pri čemu se čini da Microsoft očekuje rad samo sa proizvođačima automobila, te prilagođava svoj operacijski sustav pojedinim proizvođačima, odnosno nadograđuje svoj sustav kako bi zadovoljio potrebe svojih klijenta. S druge strane QNX se može preuzeti (čak i besplatno na 30 dana kako bi se mogao isprobati) odmah sa stranice, bez potrebe za kontaktom sa organizacijom, te se čini da bi netko sam mogao isprobati operacijski sustav, te ako ima potreban hardver i znanje, možda ga i postaviti u automobil koji ga podržava.

Sve u svemu oba sustava se čine solidna i dobra u tome što rade, te će se izbor vjerojatno svesti na to s kakvom tehnologijom se želi raditi i koliko je bitna pouzdanost sustava. Tvrtke i drugi korisnici koji su upoznati sa Microsoftovim tehnologijama će vjerojatno odabrati WEA radi lakše implementacije sustava za infotainment, a ako je potreban operacijski sustav koji je napravljen da bude pouzdan i ima veću toleranciju na greške, tada će odabrati QNX.

4. Aplikacija za prepoznavanje registracijske oznake

U ovom poglavlju će biti opisana sama reprezentativna aplikacija, njen detaljan opis, funkcionalnosti, te sama statistika koliko se pouzdano prepoznaje registracijska oznaka ovisno o udaljenosti.

Koristiti će se dosta referenci na literaturu vezanu uz samu digitalnu obradu slika radi nedostatka iskustva u tom području.

Sama aplikacija je izrađena u python jeziku (primarno se koristi OpenCV knjižnica), te se u ovom slučaju pokreće na raspberry pi ugrađenom računalu koje na sebi ima raspberry pi os (nekad zvan raspbian) za operacijski sustav. Razvojno okruženje korišteno za izradu aplikacije jest Geany koji dolazi u paketu sa operacijskim sustavom, te se može koristi za razvoj aplikacija u raznim programskim jezicima, ali na žalost nema puno značajki koje su uobičajene za razvojna okruženja (nema automatsko dovršavanje koda).

Kao oprema se koristilo već spomenuto ugrađeno računalo, kamera (5 mega piksela) namijenjena za to računalo, te tronožac koji inače služi kao postolje digitalnim kamerama, ali u ovom slučaju je postolje za ugrađeno računalo.

4.1. Opis aplikacije

Kako je već bilo spomenuto sama aplikacija služi za prepoznavanje registracijske oznake, detaljnije ideja je da prepoznaje oznake u svrhu podizanja i spuštanja rampe kako bi se nekome dozvolio pristup parkiralištu. To znači da je cilj aplikacije prepoznati samu registracijsku oznaku, pogledati u bazu podataka dali takva registracija postoji, i ako da javiti to porukom na ekran. Na Internetu postoji nekolicina implementacija ovakve aplikacije (jedna od kojih je prikazana u ovom radu) a većinom se svode na prepoznavanje pravokutnika koji sadrži registraciju, sad dali je to napravljeno tako da se na slici istaknu svi rubovi i nakon toga se traži pravokutni oblik, ili se istaknu svijetle površine, ta nakon toga se pokušava naći pravokutnik temeljem omjera visine i širine registracije, ili neke druge implementacije. Nabrojene implementacije su istaknute radi toga što su obje bile isprobane, prva sa rubovima se čini puno manje pouzdana od druge (vjerojatno radi toga što se mora gledati svako geometrijsko tijelo koje ima četiri kuta, a registracije u hrvatskoj imaju okvir koji ponekad nije niti prepoznat kao da ima četiri kuta), te je zato druga korištena u ovom radu.

Općenito aplikacije za prepoznavanje registracije sa slike su često korištene u različite svrhe: prepoznavanje registracije u svrhe naplaćivanja kazna zbog prebrze vožnje, općenito

praćenje registracija kako bi se znalo gdje se netko vozio, automatsko naplaćivanje na temelju registracija za autoceste, pristup nekom parkiralištu, itd.

Neki od razloga zašto bi se koristila aplikacija za prepoznavanje registracijske oznake u svrhu pristupa parkiralištu su: nema potrebe za karticama koje dozvoljavaju pristup (mogu se koristiti kao sekundarna opcija u slučaju prestanka rada kamere, ili ne mogućnosti prepoznavanja registracije), nema potrebe za mobilnom aplikacijom koja dozvoljava pristup i slično. Poanta je da se ne oslanja na ljude, odnosno netko bi mogao zaboraviti ponijeti karticu, dok registraciju ne mogu zaboraviti jer je na automobilu.

Alternativa koja je slična ovom sustavu dozvoljavanja pristupa jest sustav koji radi sa RFID naljepnicama. Naljepnica se zalijepi negdje na ili unutar automobila, te se na ugrađeno računalo spoji RFID čitač koji prepoznaje kada netko sa naljepnicom dođe dovoljno blizu (problem je što većina RFID čitača nema veliki doseg, većinom ispod jedan metar) .

Aplikacija je jednostavna, ali izrada iste ne toliko, iz dva glavna razloga: nedostatak znanja o digitalnoj obradi slika, te subjektivnost slika, pod tim se misli na to da sama obrada dosta ovisi o kutu, osvjetljenju, pozadini, i sličnom.

4.2. Funkcionalnosti

Sama aplikacija se može podijeliti na pet glavni funkcionalnost: dohvat registracija iz baze, samo slikanje registracije, digitalna obrada slike, prepoznavanje teksta na slici i usporedba teksta. U nastavku će svaka funkcionalnost biti ukratko objašnjena.

Dohvat iz baze je jednostavan, te služi kako bi se dobila lista registracija koja se poslije koristi za usporedbu sa registracijom koja je prepoznata iz slike.

Slikanje registracije se svodi na par linija koda, bitno je da kamera ima nekoliko sekundi vremena da fokusira i nakon toga se može slikati.

Sama digitalna obrada slike se svodi na to da se pokušava izolirati sama registracija i nakon toga pripremiti istu za prepoznavanje teksta.

Prepoznavanje teksta se radi sa OCR alatom tesseract OCR. Prije samog prepoznavanja poželjno je definirati postavke.

I naposljetku uspoređivanje teksta kojeg OCR alat prepoznaje sa registracijskim oznakama iz baze.

4.2.1. Baza podataka

Sama baza podataka za ovu aplikaciju je jako jednostavna, sastoji se od samo jedne tablice koja ima samo jedno svojstvo, i to je sama registracija. Moglo se dodati na primjer ime i prezime osobe čija je registracija, ali s obzirom da se ne sakuplja nikakva statistiku dolazaka i odlazaka, nije bilo potrebe. Eventualno bi se ime i prezime moglo koristi za ispis poruke na ekran, ali s obzirom da to uobičajeno ne bi bilo vidljivo korisniku (osim ako bi bio spojen LED ekran na ugrađeno računalo), također nije potrebno. Razlog za korištenje baze umjesto tekstualne datoteke jest taj da ako bi se ikad nadograđivala aplikacija, bilo bi jednostavnije isto napraviti preko baze (na primjer dodati nekoliko dodatnih svojstva u postojeću tablicu, ili kreirati novu tablicu za bilježenje dolazaka i odlazaka).

S obzirom da se radi o jednoj tablici nema smisla raditi EVA model, pa će u nastavku biti opisana tablica i njeno polje. Sama tablica se zove registracije, a naziv polja je registracija i to polje je tekstualnog oblika (u tablici je samo jedna registracija, registracija vlastitog automobila, kako ne bi bile uvedene tablice od drugih ljudi, u svrhu čuvanja informacije registracijske oznake)

U nastavku će biti prikazan kod iz aplikacije vezan uz bazu podataka:

```
konekcija = psycopg2.connect("dbname=reg user=karlo password=041996")
kursor = konekcija.cursor()
kursor.execute("SELECT * FROM registracije")
red = kursor.fetchone()
lista=[]
while red is not None:
      for unos in red:
            lista.append(unos)
      red = kursor.fetchone()
kursor.close()
if konekcija is not None:
      konekcija.close()
```

Preko prve linije se ostvaruje spajanje na bazu (koristi se `psycopg2` adapter za PostgreSQL baze podataka), argumenti su: ime baze podataka na koju se treba spojiti, naziv korisnika i lozinka korisnika. Nakon toga se kreira kursor preko kojeg se izvršava upit za dohvat svih registracija, te nakon toga se u varijablu sprema prvi red rezultata (preko kursora se može dohvatiti jedan, svi ili određeni broj redova odgovora na upit). Zatim se kreira prazna lista u koju će biti smještene sve registracijske oznake. Iduća linija je `while` petlja koja se vrti sve dok postoje redovi u rezultatu, odnosno sve dok se ne dođe do praznog reda. Unutar te petlje je `for each` petlja u kojoj se sve vrijednosti registracije koje se nalaze u redu

spremaju u listu. Nakon `for each` petlje se u varijablu za red sprema idući red. Nakon petlji se zatvara kursor, i ako već nije zatvorena, zatvara se i konekcija.

Sam kod je vrlo jednostavan, te se svodi na dohvaćanje svih zapisa iz tablice registracije. Sama lista je globalna varijabla te se može koristiti u svim metodama, a koristi se u metodi za usporedbu teksta.

Razlozi zašto se odmah na početku spremaju svi zapisi u listu, a ne zove se upit po potrebi i za specifične registracije su sljedeći: postoji samo jedan zapis u bazi (u realnoj situaciji ne bi ih bilo više od sto, tako da bi ih i dalje bilo malo), s obzirom na to da OCR (optičko prepoznavanje slova, eng. Optical Character Recognition) rijetko vrati identičnu registraciju, upit kojem se predaje registracijska oznaka u svrhu pronalaženja iste u bazi bi rijetko kada vraćao rezultat (OCR često prepoznaje neka dodatna slova, pa upit ne bi vraćao rezultat).

Nakon dohvata iz baze kreće glavana petlja programa, odnosno konstantno slikanje i analiza, svakih nekoliko sekundi.

4.2.2. Slikanje registracije

Samo slikanje se svodi na nekoliko linija koda, uz to u ovom pod potpoglavlju će biti opisana i glavna petlja aplikacije. Kod ispod sadrži linije vezane uz kameru, te glavnu petlju aplikacije:

```
brojac = 1
camera = PiCamera()
try:
      while True:
            camera.start_preview()
            sleep(3)
      camera.capture("/home/karlo/Desktop/Aplikacija/Slika"+str(brojac)+".j
pg")
            rezultat = analiza()
            brojac +=1
            if(rezultat == True):
                  print("Tablica uspješno prepoznata; Otvaram rampu")
            else:
                  print("Tablica nije prepoznata")
except KeyboardInterrupt:
      print("Aplikacija se gasi")
```

Prva linija koda služi za postavljanje početne vrijednosti brojača, a sam brojač služi za to da se svaka slika sprema u zasebnu datoteku, odnosno da se slike međusobno ne prepisuju. Druga linija je kreiranje `PiCamera` instance koja se koristi za rad s kamerom.

Cijela petlja je unutar `try` bloka, `except` blok služi za lovljenje prekida preko tipkovnice, u slučaju predkida, aplikacija se gasi. Sama glavna petlja je beskonačna te odmah u prvoj liniji petlje se pokreće pretpregled kamere čime se kamera pali, te počinje gledati svoju okolinu. Nakon te linije je naredba za spavanje, koja služi kako bi kamera imala dovoljno vremena da fokusira sliku. Odmah nakon te linije se izvršava slikanje, te se slika sprema na određenu lokaciju. Nakon što je slika spremljena, pokreće se metoda za analizu koja vraća `boolean` vrijednost (ovo metoda će biti opisana u iduća dva potpoglavlja), nakon njenog izvršavanja, brojač se povećava za jedan, tako da će iduća slika biti spremljena u zasebnu datoteku, odnosno neće prepisati prijašnju sliku. I na posljetku ovisno o tome dali je rezultat metode analiza istina ili laž, ispisuje se poruku o uspjehu (otvaranje rampe) ili neuspjehu (tablica nije prepoznata).

Glavna petlja se svodi na slikanje, analizu slike, te ispis rezultata te analize, ovo se radi u beskonačnoj petlji, odnosno stalno se slikaju slike, neovisno o tome da li je u slici prisutan automobil. Ovo bi se moglo bolje napraviti tako da se koristi neki senzor koji bi prepoznao kada je automobil došao na naznačeno mjesto, te bi se tek tada pokretala kamera. To bi se trebalo napraviti isto u beskonačnoj petlji, ili bi trebao postojati neki rukovatelj događajima koji bi se pokrenuo kada senzor prepozna vozilo.

Iduće na redu je potpoglavlje vezano uz digitalnu obradu slike koja je bila poslikana i spremljena prije samog poziva metode.

4.2.3. Digitalna obrada slike

U ovom potpoglavlju će biti opisana većina koda u metodi za analizu fotografije, primarno sve vezano uz digitalnu obradu slike. Svrha digitalne obrade slike je da se pronađu konture koje bi mogle sadržavati samu registracijsku oznaku. Ovdje se zapravo radi o algoritmu za prepoznavanje registracijske oznake. Kao što je već prije bilo spomenuto sam algoritam je preuzet, uz neke promjene postavka i logike kako bi bolje radio za ovo aplikaciju, tako da većina koda koji slijedi je preuzeta s izvora [21]

Promjene napravljene nad algoritmom će biti posebno napomenute, te razlog iza tih promjena. Prvo će biti opisan sam kod i objašnjenje naredbi, te nakon toga će biti sumiran sam postupak digitalne obrade slike. Sav kod prikazan ispod je iz `analiza()` metode.

Prvih nekoliko linija koda nisu vezane uz sam algoritam, već su ono što algoritam očekuje kao ulaz:

```
putanja = r'/home/karlo/Desktop/Aplikacija/Slika'+str(brojac)+'.jpg'
slika = cv2.imread(putanja,0)
```

```
slikaRez = cv2.resize(slika,(1920,1080))
```

Prva linija je zapravo postavljanje putanje do slike u varijablu, u putanji se koristi varijabla `brojac` kako bi se dohvatila najnovija slika. Druga linija je učitavanje slike u varijablu, ovdje se već počinje koristiti OpenCV biblioteku koja dok se učitava (eng. Import) ima naziv cv2. Za argumente, čitanje prima putanju do slike, te je drugi argument zastava koja definira način na koji slika treba biti učitana (u ovom slučaju vrijednost je nula, a to znači učitavanje slike u sivim tonovima, eng. Grayscale). Većinom kada se vrši digitalna obrada slike, radi se nad slikom bez boje, jer zauzima manje prostora, te su operacije nad takvom slikom brže. U zadnjoj liniji se koristi operacija za promjenu veličine slike, u ovom slučaju veličina se postavlja preko rezolucije. Inače se često koristi manja rezolucija, ali tada OCR alat ima veće probleme kod prepoznavanja teksta. Ukratko kod iznad služi za učitavanje slike bez boje, te postavljanje njezine veličine.

Idući isječak koda je vezan uz morfološku transformaciju pod nazivom crni šešir (eng. Blackhat).

Operacija crni šešir se definira kao razlika između unesene slike i zatvaranja unesene slike. Zatvaranje je operacija koja prvo izvodi operaciju proširivana, nakon toga operaciju erozija. Operacija proširivanja se u ovom slučaju koristi kako bi se popunile rupe u slici (većinom se povećavaju bijele površine na slici), ali se ovom operacijom povećavaju i mala bijela područja koja nisu poželjna. Zato se koristi operacija erozije koja radi obrnuto od proširivanja, te se ovom operacijom nastoji eliminirati nepoželjna mala bijela područja. [22]

```
kernelRegistracija = cv2.getStructuringElement(cv2.MORPH_RECT,(13,5))
blackhat = cv2.morphologyEx(slikaRez,cv2.MORPH_BLACKHAT,kernelRegistracija)
```

U prvoj liniji definiramo zrno (eng. Kernel) koja se zatim koristi u morfološkoj transformaciji (druga linija), sama metoda prima sliku, zastavu koja definira vrstu transformacije, te zrno. Ovdje definirano zrno je veličine 13x5 piksela, te je korištena ta veličina radi toga što navodno odgovara obliku registracijske oznake.

U ovom slučaju se operacija crni šešir koristi kako bi se istaknuli tamni objekti na svijetloj pozadini, odnosno kako bi se istaknula crna slova registracije.

Sljedeći isječak koda je vezan uz morfološku transformaciju zatvaranja i dovođenje praga (eng. Thresholding).

Operacija zatvaranja je već prije bila opisana, radi se o proširivanju nakon kojeg slijedi erozija. Dovođenje praga je osnovni oblik segmentiranja slike (segmentirane slike je jednostavnije analizirati), radi se o podjeli slike na dva dijela: pozadinu i prvi plan (eng.

Foreground). Odnosno određuje se koji pikseli zadovoljavaju zadani prag, te ti postaju dio prvog plana (bijeli), a ostali pozadine (crni). [23], [24]

```
kernelKocka = cv2.getStructuringElement(cv2.MORPH_RECT,(3,3))
svijetlaPodrucja = cv2.morphologyEx(slikaRez,cv2.MORPH_CLOSE, kernelKocka)
svijetlaPodrucja = cv2.threshold(svijetlaPodrucja,0,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
```

Prvo se kreira zrno veličine 3x3 piksela, to je tipična veličina zrna kod procesiranja slika. Iduća naredba je morfološka transformacija zatvaranja koja se izvodi na ulaznoj slici (slika u sivim tonovima), te je cilj istaknuti sva svijetla područja na slici (jedno takvo područje bi trebala biti i registracija, s obzirom da je bijele boje). Nakon toga se izvodi dovođenje praga, prva vrijednost je slika iz prijašnje linije, zatim je zadani prag (u ovom slučaju 0, crno, ali s obzirom da se koristi Otsu binarizacija ova vrijednost nije bitna), sljedeće je vrijednost koju će poprimiti svi pikseli koji imaju veći intenzitet piksela od zadanog praga (ovdje 255, odnosno bijelo) i zadnji argument je vrsta tip dovođenja praga (u ovom slučaju, to su dva tipa koji se izvode jedan za drugim). Gore je dan opis kako binarni tip određuje vrijednost pikselima, a drugi je u ovom slučaju Otsu binarizacija (postavljena kao dodatna zastava nakon `cv2.THRESH_BINARY` zastave) koja automatski određuje vrijednost praga, pa vrijednost koja je morala biti zadana (0) će biti zamijenjena sa automatski određenom vrijednošću. To je vrlo praktično jer se ne mora unaprijed znati kako je slika osvijetljena, odnosno u teoriji, ne mora postojati konzistentno osvjetljenje. Razlog zašto se dohvaća drugi element polja jest taj da je to element koji sadrži sliku (prvi sadrži automatski izračunatu vrijednost praga).

Sama svrha koda je da istakne sva svjetla područja na slici, s ciljem da jedno od tih područja bude i registracijska oznaka, jer je i ona bijele boje.

Sljedeći isječak koda je vezan uz gradijent po x-osi, te se za to koristi `sobel` operator.

`Sobel` operator se korsti za isticanje rubova, odnosno rezultat je slika sa na kojoj su istaknuti rubovi [22]

```
gradijentPoX = cv2.Sobel(blackhat, ddepth=cv2.CV_32F, dx=1, dy=0, ksize=-1)
gradijentPoX = np.absolute(gradijentPoX)
(minVri, maxVri) = (np.min(gradijentPoX), np.max(gradijentPoX))
gradijentPoX = 255 * ((gradijentPoX-minVri)/(maxVri-minVri))
gradijentPoX = gradijentPoX.astype("uint8")
```

U prvoj liniji se poziva sobel metoda kojoj se predaje slika na kojoj je bila izvršena operacija crni šešir, nakon toga je dubina slike (u ovom slučaju se postavlja vrijednost koja definira da svaki piksel može poprimit vrijednost od 0 do 1.0, odnosno radi se o float vrijednosti, više

neće biti u rasponu od 0 do 255), sljedeće se definira da se radi o gradijentu po x osi (vrijednost 1), za y-os se postavlja 0, te zadnjim argumentom se postavlja veličina zrna (ovdje je vrijednost -1, koja nije dozvoljena, pa se koristi zadana (eng. Default) vrijednost). Ostale linije služe kako bi se slika vratila u 8-bitni zapis, odnosno u raspon od 0 do 255.

Svrha ovog djela koda je isticanje rubova, kako bi bilo lakše naći konture koje sadrže registracijsku oznaku, te je sljedeći korak grupiranje područja u svrhu kreiranja konture koja sadrži sve znakove registracijske oznake.

Sljedeći isječak koda služi za prethodno opisao grupiranje, te se koristi Gaussovo zamućivanje, te nakon toga operacija zatvaranja i dovođenje praga.

Gaussovo zamućivanje služi za zamućivanje slike u svrhu smanjenja buke ili smanjenja utjecaja artefakata kamere, te također može služiti za reduciranje rezolucije slike. Za zamućivanje se koristi Gaussovo zrno. [22], [24]

```
zagladenaSlika = cv2.GaussianBlur(gradijentPoX,(5,5),0)
zagladenaSlika = cv2.morphologyEx ( zagladenaSlika, cv2.MORPH_CLOSE, kernelRegistracija)
zagladenaSlika = cv2.threshold(zagladenaSlika,0,255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
```

Prva linija je samo zamućivanje te je za prvi argument dana slika iz prijašnjeg koraka, zatim je zadano zrno veličine 5x5 piksela, i naposljetku je standardna devijacija, koja je postavljena na 0. Nakon toga dolazi već poznata operacija zatvaranja i nakon toga binarizacija slike kako bi se dobila slika sa samo crnim i bijelim pikselima.

Svrha koda je zamutiti sliku tako da se rubovi iz prijašnjeg koraka spoje u mrlju koja će predstavljati konturu koja sadrži registracijsku oznaku. Nakon zamućivanja se izvodi zatvaranje kako bi se popunile rupe u mrlji, te nakon toga se izvodi binarizacija kako bi se dobila slika sa samo crnim i bijelim pikselima.

Idući isječak koda koristi operacije erozije i proširivanja kako bi se uklonila nepotrebna područja

```
erozijaSlika = cv2.erode(zagladenaSlika,None,iterations=2)
prosirenjeSlika = cv2.dilate(erozijaSlika,None,iterations=2)
```

Radi se o dvije vrlo jednostavne linije koda, prva izvršava eroziju nad slikom, pri čemu se izvršavaju dvije iteracija, te druga radi isto, samo što se radi o operaciji proširivanja.

Svrha koda je da se erozijom eliminiraju mala bijela područja ili točke, tako da ostanu samo veća područja, a jedno takvo područje bi treba biti i registracija. Naravno erozija smanjuje i ta veća područja, pa je potrebno sa proširivanjem povećati ta područja.

U sljedećem dijelu koda koristi se operacija koja radi s bitovima, kako bi se iskoristila maska svijetlih područja, koja je bila prije napravljena. Također se izvršavaju još dvije konačne operacije, prije nego što se uzimaju konture iz slika.

Operacije koje rade s bitovima se kao što i samo ime sugerira izvode na razini bitova. Radi se o jednostavnim operacijama koje se mogu brzo izračunati, pa su dobre za rad s slikama. Moguće operacije su: AND, OR, XOR i NOT. One rade na isti načina kao i kod skupova, odnosno na primjer AND operacija između dvije slike će za rezultat imati sliku na kojoj se nalaze samo one površine koje se nalaze u obje slike. [22]

```
bitovniANDSlika = cv2.bitwise_and (prosirenjeSlika, prosirenjeSlika, mask = svijetlaPodrucja)

konacnaSlikaJedan = cv2.dilate( bitovniANDSlika, None, iterations=10)
konacnaSlikaDva=   cv2.dilate(   bitovniANDSlika,   None,   iterations=23)

konacnaSlikaJedan = clear_border(konacnaSlikaJedan)
konacnaSlikaDva = clear_border(konacnaSlikaDva)
```

Prva linija je sama operacija nad bitovima, radi se o AND operaciji. Odmah se može primijetiti da su prva dva argumenta ista, odnosno AND operacija nad te dvije slike, vraća tu istu sliku. Zato se koristi maska, na kojoj se nalaze sva svjetla područja, to znači da kada se maska primjeni na sliku koja je rezultat AND operacije, ostati će samo područja koja se nalaze na istim mjestima kao i svjetla područja na maski, odnosno s obzirom da je područje slike na kojem je registracija definitivno jedno od svijetlih područja, ovime se zapravo eliminiraju područja koja su nastala tijekom digitalne obrade slike, a nisu bila svijetla područja.

Druge dvije linije su operacije proširivanja i ovdje počinje odvajanje od preuzetog algoritma. S obzirom da je algoritam bio korišten nad drugačijim registracijama od onih koje aplikacija treba prepoznati, rezultat digitalne obrade je drugačiji. Slova na registracijama koje je algoritam prepoznavao su bila više skupljena, odnosno gušća, pa je područje već do ovog trenutka bilo dovoljno vidljivo, ali s obzirom da su slova na registracijama koje aplikacija treba prepoznati veća, i imaju veće razmake, samo područje ne izgleda kao bijeli pravokutnik, već skoro kao slova. U algoritmu se ovdje koriste naredba proširivanja i erozije, dok u se u implementaciji za aplikaciju koristi samo proširivanje. To proširivanje također ima značajno veći broj iteracija, kako bi se dobilo pravokutno bijelo područje oko registracije. Još jedna

razlika je što ovdje postoje dvije konačne slike iz kojih će se vaditi konture, jer ovisno o udaljenosti od kamere, uvjetima osvjetljenja i slično, bolje rezultate daje jedna ili druga konačna slika. Nakon proširivanja se još koristi operacija za uklanjanje rubova. Ta operacija uklanja sva područja koja diraju rub slike, s obzirom da bi registracija trebala biti blizu sredine slike, ovime se zapravo eliminiraju nepotrebna područja.

Svrha koda je eliminacija nepotrebnih površina korištenjem maske, zatim proširivanje preostalih područja, i na posljetku eliminacija svih područja koja diraju rub slike.

Od sada na dalje će biti prikazan kod vezan samo uz jednu od konačnih slika, jer je kod isti za obje konačne slike.

Idući isječak koda je zapravo pronalaženje i izvlačenje kontura iz slike.

```
kontureSlikaJedan = cv2.findContours (konacnaSlikaJedan.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
kontureSlikaJedan = imutils.grab_contours(kontureSlikaJedan)
kontureSlikaJedan = sorted(kontureSlikaJedan, key = cv2.contourArea, reverse = True)[:5]
```

U prvoj liniji se pronalaze konture, koristi se kopija slike jer sama metoda mijenja sliku, drugi argument definira da se uzimaju samo područja koja su na vrhu hijerarhije, odnosno ako postoje konture unutar kontura, uzima se samo najveća kontura, a djeca te konture se ignoriraju. Zadnji argument je zastava za vrstu aproksimacije, u ovom slučaju jednostavna, što znači da uklanjaju sve redundantne točke i time se štedi memorija (umjesto da se spremaju sve rubne točke, ovdje se spremaju samo četiri). U drugoj liniji se dohvaćaju sve konture, a u trećoj se poredaju po veličini površine konture, te se postavlja obrnuti poredak i uzima se prvih pet kontura (obrnuti poredak kako bi dohvatili pet najvećih kontura).

Svrha koda je da se dohvate konture koje bi mogle sadržavati registracijsku oznaku, a da se istovremeno ne dohvate sve konture, već ih se filtrira po veličini (kontura koja sadrži registracijsku oznaku bi trebala biti među konturama s najvećom površinom).

Ovo je bio zadnji dio koda vezan uz digitalnu obradu slike, pa će sad cijeli algoritam biti sažeto opisan.

Algoritam koji aplikacija koristi očekuje sliku u sivim tonovima nad kojom se vrši operacija crni šešir kako bi se istaknula crna slova registracije, zatim se kreira maska svijetlih površina pomoću operacije zatvaranja. Nakon toga se izvodi gradijent po x osi kako bi se istaknuli rubovi registracije i slova registracije. Nakon toga se zamućuje slika kako bi se ti rubovi povezali i kako bi nastalo područje koje obuhvaća cijelu registraciju (u slučaju ove aplikacije to nije ovdje još postignuto, već se kasnije koristi operacija proširivanja kako bi se to

postiglo). Iduće je erozija i proširivanje slike u svrhu smanjenja buke, odnosno uklanjanje nepotrebnih površina. Zatim se izvodi AND operacija nad bitovima, te je cilj eliminirati sva svjetla područja koja nisu u maski svijetlih područja. Zatim se izvodi operacija proširivanja kako bi se dobila kontura u kojoj se nalazi registracijska oznaka, te se uklanjaju sve površine koje diraju rub. Na kraju se iz slike vade konture iz kojih će se u idućem koraku uz pomoć OCR alata pokušati iščitati registracijska oznaka.

Ispod slika prikazuje konačni rezultat digitalne obrade slike, prije nego što se vade konture.

Slika 2: Rezultat digitalne obrade slike (izvor: slika izvađena iz vlastite aplikacije)

Bijelo pravokutno područje blizu sredine slike jest sama registracijska oznaka, te kako se može primijetiti to je jedna od najvećih bijelih površina na slici. Najčešći uzrok neuspješnog prepoznavanja registracijske oznake je taj da je bijelo područje podijeljeno na dva, odnosno nije spojeno, te se tada prepoznaje kao dvije različite konture.

4.2.4. Prepoznavanje teksta s OCR alatom

U ovom poglavlju će biti opisan kod vezan uz prepoznavanje teksta iz slike, koje konfiguracije se koriste, koji znakovi su dozvoljeni, u kojim slikama se prepoznaje tekst, i slično. Kao što je već bilo spomenuto, koristiti će se tesseract OCR, specifičnije `pytesseract` biblioteka.

Isječak koda prikazan ispod je vezan uz konfiguracije i dozvoljene znakove.

```
podrucjeInteresa = None
```

```
dozvoljeniZnakovi="ABCDEFGHIJKLMNOPRSTUVZČĆŽĐŠ0123456789"
konfiguracijaP = "-c tessedit_char_whitelist=" + dozvoljeniZnakovi + " --psm 7 "
konfiguracijaD = "-c tessedit_char_whitelist=" + dozvoljeniZnakovi + " --psm 11 "
```

Prva linija je samo dodjeljivanje vrijednost varijabli, linija nakon toga je obični `string` koji sadržava sve dozvoljene znakove. Zadnje dvije linije su zapravo dvije konfiguracije koje su korištene za prepoznavanje znakova (u različitim slučajevima bolje radi jedna od konfiguracija). U prvom dijelu konfiguracije se definira bijela lista znakova, odnosno koji znakovi su dozvoljeni, a u drugom dijelu se definira način segmentacije stranice (PSM, eng. Page Segmentation Mode). PSM 7 znači da se slika gleda kao da je jedna linija teksta, a PSM 11 traži tekst gdje god može, odnosno pokušava pronaći sav tekst neovisno o tome gdje se nalazi na slici (dali je u istoj liniji ili je negdje u kutu). S te dvije konfiguracije je bilo najviše uspjeha, pa se koriste iz tog razloga.

U originalnom algoritmu sada bi se gledao odnos visine i širine konture, te ako odgovora unaprijed zadanom odnosu, tada bi se izvršavalo prepoznavanje teksta. S obzirom na moguće kutove slika, ova metoda je rijetko kada uspjela, pa je zato izbačena, te se prepoznaje tekst iz svake konture, sve dok se ne nađe registracija koja odgovara registraciji u bazi, ili se prođu sve konture (za svaku konačnu sliku ih je pet, znači deset ukupno)

Idući isječak koda je zapravo korištenje OCR alata za prepoznavanje teksta iz slike.

```
    for kontura in kontureSlikaJedan:
        (x,y,w,h) = cv2.boundingRect(kontura)
        registracijskaOznaka = slikaRez[y:y + h, x:x + w]
        sirinaOz= int(registracijskaOznaka.shape[1]*4)
        visinaOz= int(registracijskaOznaka.shape[0]*4)
        dimenzijaOz= (sirinaOz,visinaOz)
        podrucjeInteresa = cv2.threshold(registracijskaOznaka,0,255,
            cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
        registracijskaOznaka = cv2.resize (registracijskaOznaka, dimenzijaOz, interpolation = cv2.INTER_AREA)
        sirinaPI=int(podrucjeInteresa.shape[1]*4)
        visinaPI = int(podrucjeInteresa.shape[0]*4)
        dimenzijaPI = (sirinaPI,visinaPI)
        podrucjeInteresa = cv2.resize (podrucjeInteresa ,dimenzijaPI, interpolation=cv2.INTER_AREA)
        tekst = py.image_to_string(podrucjeInteresa, config = konfiguracijaP).strip()
        tekstDva = py.image_to_string(registracijskaOznaka, config = konfiguracijaP).strip()
```

```
            tekstTri   =   py.image_to_string(podrucjeInteresa,   config   =
konfiguracijaD).strip()
            tekstCetri  =  py.image_to_string(registracijskaOznaka,  config  =
konfiguracijaD).strip()
            if(len(tekst)>5):
                  usp = usporedbaTeksta(tekst)
                  if(usp !=None):
                        print("Pronađena tablica jest: "+ usp)
                        print("Tablica pronađena u tekstu: " + tekst)
                        return True
```

U prvoj liniji se dohvaćaju četiri točke unutar kojih je kontura, zatim se iz slike s crno bijelim tonovima uzima samo dio slike na kojem se nalazi kontura, te se u varijable sprema širina i visina konture pomnožena sa četiri (kako bi se bolje vidjela slova). Nakon toga se izvršava binarizacija slike kako bi se dobila crno bijela slika i to se zove područje interesa, tu se u originalnom algoritmu samo iz tog područja očitava registracijska oznaka, ali ovdje se očitava i iz slike same konture, prije binarizacije (jer ponekad OCR bolje prepoznaje znakove na toj slici). Nakon binarizacije se mijenja veličina slike s crno bijelim tonovima (povećava se četiri puta), te se isto radi i za binariziranu sliku. Nakon toga se izvršava prepoznavanje teksta i to sa metodom `image_to_string` koja prima sliku i prije definirane konfiguracije (postoje dvije konfiguracije i dvije slike, tako da se prepoznavanje teksta izvršava četiri puta). Nakon prepoznavanja teksta provjerava se dali je tekst veći od pet znakova (registracijska oznaka ima minimalno sedam znakova, a tekst može proći usporedbu ako se podudara u svim znakovima osim jednog, pa zato minimalna duljina teksta treba bit šest, da bi se slao na usporedbu). Ako tekst ima više od pet znakova, šalje se na usporedbu, metoda za usporedbu vraća registraciju ako je usporedba bila uspješna, a inače vraća `none`. Ako je bilo uspješno, ispisuje se nađena tablica, tekst iz kojeg je prepoznata, te se vraća istina. To isto se radi i za ostale potencijalne tekstove, ali ovdje je prikazano samo za jedan, jer su svi ostali isti.

4.2.5. Usporedba teksta

Usporedba teksta se nalazi u zasebnoj metodi, te je vrlo jednostavna, kod ispod je kod iz same metode.

```
print(tekst)
      for unos in lista:
            prvi = unos[:2]
            drugi= unos[2:-2]
            treci = unos[-2:]
            if((prvi in tekst) and (drugi in tekst) and (treci in tekst)):
```

```
                return unos
        else:
                listaPreklapanje = list(set(tekst)&set(unos))
                preklapanje = len(listaPreklapanje)
                if(len(unos)==7):
                        if(preklapanje>5):
                                return unos
                elif(len(unos)==8):
                        if(preklapanje>6):
                                return unos
    return None
```

Prvo se ispsuje tekst koji će se uspoređivati s registracijskim oznakama iz baze (to se radi kako bi se vidjelo dali je bila uspješno prepoznata registracija koja možda nije u bazi). Slijedi vrlo jednostavna `for each` petlja u kojoj se rastavlja registracija na tri dijela: prva dva slova, brojke u sredini i zadnja dva slova. Nakon toga se provjerava dali tekst sadrži sva tri djela registracije, ako sadrži tada se vraća registracijska oznaka, a inače se izvodi alternativno uspoređivanje teksta. Provjerava se koliko istih znakova imaju tekst i registracijska oznaka, te ako se preklapaju u svim znakovima osim jednog, tada se vraća registracijska oznaka, a u suprotnom se vraća `none`.

4.3. Statistika prepoznavanja registracijske oznake

U ovom poglavlju će biti prikazana statistika prepoznavanja registracijske oznake, pri čemu se sakupljanje statistike izvodilo na istoj lokaciji, u isto vrijeme, nad nekoliko automobila i na različitim udaljenostima od kamere. Samo prepoznavanje bi trebalo raditi u bilo koje vrijeme dana, ali bi za to bila potrebna konzistentna osvijetljenost koja nije mogla biti postignuta u ovom slučaju, pa se radi toga sva statistika skupljala u isto vrijeme u danu.

Tablica ispod sadrži prikupljene statistiku.

Udaljenost (m)	Crveni automobil	Srebreni automobil sa oštećenim grbom	Srebreni automobil	Crni automobil s niskom registracijom	Uspješnost prepoznavanja (%)
1	10	2	8	0	50
1.5	10	7	10	10	92.5
2	10	10	10	10	100
2.5	10	0	10	10	75
3	3	0	10	10	57.5
3.5	0	0	1	2	7.5

Tablica 1: Statistika prepoznavanja registracijske oznake (izvor: vlastiti rad)

U prvom stupcu je udaljenost od kamere, zatim slijede stupci za automobile, te je zadnji stupac uspješnost prepoznavanja. Prepoznavanje je najgore za srebreni auto sa oštećenim grbom, a razlog je to da radi odsutnosti grba, proširivanjem se nije dobila jedna kontura, već je registracija bila podijeljena na dvije konture, pa prepoznavanje često nije uspjelo. Za auto s niskom registracijom, jedan metar udaljenosti je bilo preblizu da bi se registracija uopće vidjela, pa je zato bilo nula prepoznavanja. Prepoznavanje najbolje radi na udaljenosti od jedan i pol do dva i pol metra (za dva i pol metra statistiku narušava auto s oštećenim grbom).

S obzirom na nepouzdanost prepoznavanja registracijske oznake iz oblika ili omjera stranica, izrađena aplikacija radi na način da se pokušava istaknuti konturu koja sadrži registraciju, a istovremeno se uklanja što više nepotrebnih kontura, tako da bi kontura koja sadrži registraciju trebala biti u pet najvećih kontura, iz kojih se zatim prepoznaje tekst, te se provjerava dali odgovara nekoj od registracija u bazi. Aplikacija radi najbolje na udaljenosti

od jedan i pol do dva i pol metra, što je dobro, jer je to razumna udaljenost auta od rampe. Aplikacija bi se mogla poboljšati sa boljom kamerom, uvođenjem strojnog učenja, i slično.

5. Zaključak

U samom radu su opisana neka osnovna teorija vezana uz ugrađene sustave, ugrađena računala i operacijske sustave, te se nakon toga se prolazi na glavni dio rada, a to su sami operacijski sustavi.

U ovom radu pokriveno je osam operacijskih sustava, pri čemu su podijeljeni u parove po funkcionalnostima kako bi ih se moglo usporediti.

U prvom paru su bili operacijski sustavi opće namjene: Rasbian i Fedora. Oba operacijska sustava su distribucije linuxa, te svaki ima svoje prednosti i nedostatke. Na kraju se došlo do zaključka da je Rasbian jednostavniji za koristiti i nudi više softverskih paketa, a Fedora je teža za koristit, a nudi najnovije linux tehnologije.

U drugom paru su bili operacijski sustavi koji se mogu koristiti za mreže senzora: TinyOS i Tizen. TinyOS je namijenjen za bežične mreže senzora, a Tizen je namijenjen za upravljanje pametnim uređajima i uspostavljanje konekcije između istih. Zaključak je bio da je TinyOS bolji za mreže senzora ako je cilj samo sakupljati i agregirati podatke, a Tizen je bolje koristiti kada nešto radimo s tim podacima, na primjer da se na temelju podataka pokreću neke radnje pametnih uređaja.

U trećem paru su bili operacijski sustavi za prikaz multimedijskog sadržaja: Rockbox i Slax. Rockbox je namijenjen za prikaz multimedijskog sadržaja, dok je Slax mini linux distribucija koja podržava prikaz multimedijskog sadržaja. Zaključak je bio da Rockbox podržava više formata multimedijskog sadržaja, ali ne pruža pristup Internetu, a sam i time multimedijskom sadržaju koji se tamo nalazi, a s druge strane Slax podržava manje formata, ali ima pristup Internetu.

U četvrtom paru su bili operacijski sustavi za infotainment sustave u automobilima: Windows embedded automative i QNX Neutrinos RTOS. Oba dva su operacijska sustava u realnom vremenu, te nude slične mogućnosti, pa je zaključak bio da je QNX pouzdaniji i otporniji na greške, a da WEC nudi pristup Microsoftovim tehnologijama za razvoj aplikacija i sam i time sustava.

Nakon toga još ostaje sama reprezentativna aplikacija koja služi za prepoznavanje registracijske oznake. Njezin opis se može svesti na to da se iz baze uzimaju sve registracije i stavljaju u listu. Nakon toga počinje glavna petlja aplikacije u kojoj se kontinuirano slika i analizira slika, te u slučaju uspjeha, diže i rampa. Sama analiza se svodi na digitalnu obradu slike uz pomoć raznih operacija i transformacija, izdvajanje kontura iz obrađene slike,

prepoznavanje teksta iz kontura i na posljetku uspoređivanje prepoznatog teksta sa registracijama iz baze.

Iako je sam algoritam za digitalnu obradu bio preuzet, sama testiranje i prilagodba algoritma potrebama aplikacije je potrajalo. Originalan algoritam je pokušavao izolirati konturu s registracijom na temelju odnosa visine i širine konture koja ju sadrži, ali zbog kuta kamere u aplikaciji za ovaj rad, to nije dobro radilo, pa se promijenilo na to da se gledaju sve konture nakon što je njihov broj smanjen na pet.

Neki generalni zaključak rada je da su ugrađena računala krucijalni dio raznih sustava koje ljudi susreću u svakodnevnom životu, iako ih možda ne primijete svi jer se ne zapitaju kako neki sustav zapravo radi, odnosno što se nalazi ispod površine.

Najzahtjevniji dijelovi izrade rada su bili sama aplikacija, te pronalaženje izvora za neke operacijske sustave. Sve u svemu rad na temi je bio zabavan i poučan, iako nije bilo moguće pokriti cijeli opseg teme, jer su ugrađena računala prisutna u velikom broju različitih sustava, te obavljaju svakojake različite funkcionalnosti, a sam i time za njih postoji veliki broj operacijskih sustava, koji su prilagođeni potrebama ugrađenog računala i samim sustavima.

Popis literature

[1] M. Barr i A. Massa, Programming embedded systems: with C and GNU development tools. O'Reilly Media, Inc., 2006.

[2] T. Noergaard, Embedded systems architecture: a comprehensive guide for engineers and programmers. Newnes, 2012.

[3] Classification of Embedded Systems, GeeksforGeeks, 18.08.2020. https://www.geeksforgeeks.org/classification-of-embedded-systems/ pristupano 20.08.2022.

[4] A. Holt and C.-Y. Huang, Embedded operating systems. Springer, 2014.

[5] A. Kurniawan, Raspbian OS Programming with the Raspberry Pi. Springer, 2019.

[6] W. Harrington, Learning raspbian. Packt Publishing Ltd, 2015.

[7] R. Hertzog i R. Mas, The Debian Administrator's Handbook, Debian Wheezy from Discovery to Mastery. Lulu. com, 2014.

[8] C. Negus i E. Foster-Johnson, Fedora Bible: Featuring Fedora Linux 12, vol. 631. John Wiley & Sons, 2010.

[9] B. Gunasekaran, Debian vs Fedora: Similarities & Differences!, Embedded Inventor, 17.09.2020. https://embeddedinventor.com/debian-vs-fedora-similarities-differences/ pristupano 20.08.2022.

[10] P. Levis i ostali, TinyOS: An operating system for sensor networks, in Ambient intelligence, Springer, 2005, pp. 115–148.

[11] P. Levis i D. Gay, TinyOS programming. Cambridge University Press, 2009.

[12] G. Vashisht i R. Vashisht, A study on the Tizen Operating System, International Journal of Computer Trends and Technology, vol. 12, no. 1, pp. 14–15, 2014.

[13] Tizen docs (2022.a), Introduction to Tizen https://docs.tizen.org/platform/what-is-tizen/overview/ pristupano 20.08.2022.

[14] Tizen docs (2022.b) Device sensors https://docs.tizen.org/application/native/guides/location-sensors/device-sensors/ pristupano 20.08.2022.

[15] Rockbox wiki, Rockbox Architecture, 02.04.2021. https://www.rockbox.org/wiki/RockboxArchitecture pristupano 20.08.2022.

[16] B. Childs, Rockbox, Linux Journal, vol. 2010, no. 199, p. 5, 2010.

[17] Dedoimedo ,SLAX Linux - Your pocket operating system - Review, 13.02.2009. https://www.dedoimedo.com/computers/slax.html pristupano 20.08.2022.

[18] Microsoft, A Technical Companion to Windows Embedded Automotive 7 01.07.2010.

[19] S. Phung, D. Jones, i T. Joubert, Professional Windows Embedded Compact 7. John Wiley & Sons, 2011.

[20] N. Struck, Realtime operating systems in automotive infotainment, ATZelektronik worldwide, vol. 2, no. 1, pp. 23–25, 2007.

[21] A. Rosebrock, OpenCV: Automatic License/Number Plate Recognition (ANPR) with Python, PyImageSearch, 21.09.2020. https://pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/ pristupano 20.08.2022.

[22] A. F. Villán, Mastering OpenCV 4 with Python: a practical guide covering topics from image processing, augmented reality to deep learning with OpenCV 4 and Python 3.7. Packt Publishing Ltd, 2019.

[23] A. Pajankar, Python 3 Image Processing Learn Image Processing with Python 3, NumPy, Matplotlib, and Scikit-image. BPB Publications, 2019.

[24] G. Bradski i A. Kaehler, Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media, Inc., 2008.

Popis slika

Slika 1: Generički ugrađeni sustav (napravljeno prema slici iz [1])....3

Slika 2: Rezultat digitalne obrade slike 29

Popis tablica

Tablica 1: Statistika prepoznavanja registracijske oznake (izvor: vlastiti rad).....33