SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5059

Sustav za automatsko optičko očitanje mjernog uređaja realiziran ugradbenim računalom

Marin Bralić

Zagreb, lipanj 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 8. ožujka 2017.

ZAVRŠNI ZADATAK br. 5059

Pristupnik: **Marin Bralić (0036481753)**
Studij: Računarstvo
Modul: Računalno inženjerstvo

Zadatak: **Sustav za automatsko optičko očitanje mjernog uređaja realiziran ugradbenim računalom**

Opis zadatka:

U okviru završnog rada potrebno je istražiti mogućnosti izvedbe ugradbenog računalnog sustava za automatsko optičko očitanje numeričkih ili alfanumeričkih prikaznika. Mjerni uređaji poput brojila koja se koriste u kućanstvu za mjerenje utroška energije (struje, plina, tople vode) ili utroška vode imaju mehaničke ili elektroničke prikaznike izmjerenih veličina. Potrebno je razviti sustav temeljen na kameri za dvodimenzionalno očitanje prikaznika mjernog uređaja i na ugradbenom računalu, za prihvat slike i njenu obradu, te automatsko prepoznavanje znamenki očitanja mjernog uređaja. Sustav mora biti prilagodljiv na različite tipove mjernih uređaja, a mora omogućavati pohranu izmjerenih veličina u datoteku na ugradbenom računalu, odnosno vizualizaciju izmjerenih podataka putem grafičkog i/ili web-sučelja. Sustav je potrebno razviti korištenjem ugradbenog računala Raspberry Pi pod operacijskim sustavom Linux, a za obradu slike koristiti odgovarajuće biblioteke otvorenog koda poput OpenCV.

Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 9. lipnja 2017.

Mentor:
Prof. dr. sc. Davor Petrinović

Djelovođa:
Prof. dr. sc. Danko Basch

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Mario Kovač

SADRŽAJ

1. Uvod **5**

2. Raspberry Pi **6**

3. Princip rada **8**

3.1 Predobrada slike **10**

3.2 Pronalaženje znamenki **13**

3.3 Očitavanje znamenki **20**

4. Upute za instalaciju **24**

5. Zaključak **26**

6. Literatura **28**

7. Sažetak **29**

8. Dodatak: programski kod **30**

1. Uvod

Sustav za automatsko očitanje pokaznika baziranih na 7 segmentnom display-u, namijenjen za očitavanje vrijednosti mjernih uređaja poput brojila koja se koriste u kućanstvu za mjerenje utroška energije (struje, plina, vode), a može se upotrijebiti i za druge primjene. Sustav je izveden ugradbenim računalom Raspberry pi i sadrži web kameru spojenu putem USB porta, ali može se ugraditi i bilo koja druga kamera i CSI port umjesto USB porta. Kamera se postavlja na par centimetara od 7 segmentnog pokaznika tako da je cijeli pokaznik u kadru, zatim sustav periodički okida i sprema sliku, vrši obradu i očitava numeričke vrijednosti te pohranjuje rezultat u datoteku dok istovremeno može i ispisivati na terminal operacijskog sustava. Ugradbeno računalo Raspberry pi koristi Raspbian operacijski sustav baziran na Debian-u iz familije UNIX sustava, ali može se instalirati i raditi na bilo kojem drugom operacijskom sustavu kojeg podržava računalo Raspberry pi. Programska podrška obavlja glavni dio posla (obradu, analizu i očitavanje numeričkih vrijednosti sa slike), napisana u programskom jeziku Python, zbog čega sustav i je fleksibilan s obzirom na arhitekturu samog računala i operacijski sustav. Obrada slike obavlja se pomoću OpenCV biblioteke otvorenog koda, napisana je u jezicima C/C++, a sadrži mnoštvo funkcija za obradu slike u realnom vremenu te se koristi u području računalnog vida.

2. Raspberry Pi

Raspberry pi je ugradbeno računalo sa ARM arhitekturom, veličine kreditne kartice (slika 1.0). Neki od operacijskih sustava koje podržava su:
Ubuntu MATE, Kali Linux, Windows 10, RISC OS, Debian FreeBSD, NetBSD i drugi. Računalo dolazi u nekoliko verzija od kojih će u ovom slučaju
bit korištena 3. generacija "Raspberry Pi 3 B", a može se koristiti I bilo koja druga verzija. Hardware je baziran na SoC (system on chip), cijeli računalni sustav procesor, memorija, grafički procesor i ostale komponente su u jednom čipu: Broadcom BCM2837, dok se operacijski sustav i podatci nalaze u vanjskoj memoriji na micro SD kartici. Hardware uređaja je djelomično open source jer ne postoji potpuna dokumentacija cijelog sustava. Sustav za automatsko očitanje koristit će standardnu web kameru preko USB porta. Za UNIX operacijske sustave postoji univerzalni upravljački program (driver) za kameru koji se može pokrenut preko terminala, detaljnije informacije u poglavlju ‘Upute za instalaciju’.

Slika 1.0

Osnovni podatci računala:

Procesor: 1.2 GHz 64/32-bitni četverojezgreni ARM Cortex-A53 procesor
Grafički procesor: Broadcom VideoCore IV 3D grafička jezgra 400MHz
Memorija: 1 GB LPDDR2 RAM 900MHz
Bežićno povezivanje: 802.11n Wireless LAN, Bluetooth 4.1
Konektori: 4 USB porta, 40 GPIO pinova, full HDMI port, Ethernet port, 3.5mm audio port, CSI konektor za kameru, DSI konektor za ekran, micro SD slot
Potrošnja energije: 1.5W prosječno, do 6.7W maksimalno.

3. Princip rada

Programska podrška sastoji se od jedne 'bash' skripte 'run.sh' (slika 3.0.1) i samog python programa. Unutar skripte periodično se okida slika u beskonačnoj petlji sa kamere i sprema u isti direktorij. Nakon toga poziva se program za očitavanje: ocitavanje.py sa dva argumenta. Prvi argument je ime slike koju je kamera napravila kako bi ju program mogao učitati i nastaviti daljnju obradu (tmpimg.jpg), a drugi je cijeli broj koji se koristi samo kod testiranja kako bi svaka slika međukoraka imala svoj index. Program ocitavanje.py sam ispisuje očitane vrijednosti na terminalu. Na kraju petlje dodana je još i 'sleep' naredba kako bi se brzina okidanja i očitavanja mogla kontrolirati i prilikom testiranja usporiti.

```
#! /bin/bash

rm *.png

time=2
name="tmpimg.jpg"
while [ 1 ]
do
        fswebcam $name
        python ocitavanje.py $name 0
        echo "\n"
        sleep $time
done
```

Slika 3.0.1

Ove dvije datoteke 'run.sh' i 'ocitavanje.py' možemo staviti u bilo koji direktorij u datotečnom sustavu koji ima postavljene dozvole za pisanje te nakon toga program se pokreće unosom naredbe: './run.sh' u terminal nakon čega počinje periodično očitavanje pokaznika.

Program ‘ocitavanje.py’ prolazi kroz 3 glavna koraka:

1. Predobrada slike: Na okinutoj slici potrebno je napraviti nekoliko izmjena kako bi se sljedeći korak mogao uspješno provesti.

2. Pronalaženje znamenki: U ovom koraku se pronalazi lokacija i veličina pravokutnika unutar kojega se nalaze znamenke.

3. Očitavanje znamenki: Sa svake znamenke se posebno analiziraju segmenti i na kraju se od svih znamenki kreira jedan cijeli broj.

3.1 Predobrada slike

Program započinje glavnom funkcijom (slika 3.1.1) unutar koje se pokušava očitati pokaznik na sljedeći način. Kao parameter program dobiva ime slike koju učitava u varijablu i odmah pretvara u crno bijelu:

```
img=cv2.imread(sys.argv[1], cv2.CV_LOAD_IMAGE_GRAYSCALE)
```

Nakon učitavanja, unutar 'for' petlje se pokušava očitati vrijednost sa različitim parametrima koji se nalaze unutar prethodno definiranog polja 'parametri'. Svaki član polja sadrži dvije vrijednosti koje predstavljaju jedan raspon vrijednosti. Kasnije kada se slika bude pretvarala u binarnu (isključivo crna i bijela boja bez sivih tonova) sve vrijednosti koje se nalaze unutar granice bit će bijele boje, a sve ostalo izvan će biti crne boje. Možemo vidjeti da polje u prvom dijelu sadrži niske granice (od 0 do 60), a u drugom dijelu visoke granice (od 220 do 255). To je potrebno jer ne znamo kakav je pokaznik na slici, da li je svjetleći sa diodama, ili je sivi sa tekućim kristalom. Zbog toga će segmenti na sivom pokazniku biti blizu crne boje i potrebno je dobro vanjsko osvjetljenje pokaznika, dok segmenti sa led diodama mogu biti raznih boja ali emitiraju svijetlost pa će na slici biti kao vrlo svijetle površine blizu bijele boje.

Unutar petlje uzima se jedan par parametara i predaje funkciji 'ocitajBroj'. Funkcija još prima kao argument i ime slike, index parametra i samu sliku, od kojih ime i index parametara nisu potrebni, ali se koriste za ispise međurezultata prilikom testiranja. Nakon toga funkcija će pokušati očitati pokaznik i vratiti vrijednost u varijablu 'broj' ako uspije ispravno pročitat te odmah iskače iz 'for' petlje I ispisuje sadržaj varijable 'broj' tj očitanu vrijednost sa pokaznika. Ukoliko se vrijednost ne uspije pročitati funkcija vraća prazan string te se uzima sljedeći parametar i sa sljedećim pokušava očitati. Zbog toga polje 'parametri' sadrži nekoliko parametara koje možemo svrstati u dvje grupe: parametri za svjetleći pokaznik i parametri za sivi pokaznik.
Ovi pokušaji očitavanje se temelje na tome što ako je riječ o sivom pokazniku, segmenti neće uvijek biti crne niti tamno sive boje nego ponekad svijetliji a ponekad tamniji. Isto vrijedi i za svjetleći pokaznik. Ovime sustav donekle postaje otporan na ovakve smetnje uzrokovane vanjskim osvjetljenjem i jačinom pokaznika.

```
#################################
##########   Start   ############
#################################
img=cv2.imread(sys.argv[1], cv2.CV_LOAD_IMAGE_GRAYSCALE)

parametri=[(250,255), (245,255), (240,255), (235,255), (230,255), (225,255), (220,255),
           (0,10), (0,15), (0,20), (0,25), (0,30), (0,35), (0,40), (0,45), (0,50), (0,55), (0,60)]

broj=""
for i in range(0, len(parametri)):
        broj=ocitajBroj(sys.argv[1], i, img, parametri[i][0], parametri[i][1])
        if broj!="": break


print broj
```

Slika 3.1.1

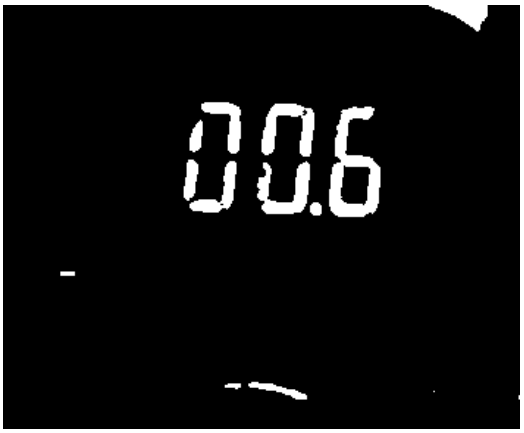
Nakon ulaska u funkciju ‘ocitajBroj’ (slika 3.1.6) funkcija kao argument sadrži prethodno dobivenu crno-bijelu sliku (slika 3.1.2). Sljedeći korak obrade je uklanjanje visokih frekvencija na slici. To se može postići jednom funkcijom iz openCV biblioteke: ‘medianBlur’. Nakon toga slika će izgledati nešto mutnije: 3.1.3.

Da bi se znamenke na slici još bolje isticale, pojačat ćemo contrast slike funkcijom ‘equalizeHisz’, nakon toga slika 3.1.4 prikazuje izgled nakon provođenja ove funkcije. Kod sivih pokaznika ovime će znamenke biti još tamnije dok će okolina biti svijetlija, a kod svjetlećih obrnuto. Sada preostaje napraviti posljednji i završni korak u pred obradi kako bi slika postala spremna za pronalaženje znamenki. U posljednjem koraku sliku ćemo pretvoriti u binarnu, dakle uklanjaju se sve nijanse sivih boja I sve će biti ili isključivo crno ili isključivo bijelo. Koja površina će postati koje boje određuje se parametrima koje je funkcija dobila kao argument. Parametri su prethodno u glavnoj funkciji bili objašnjeni. U ovoj funkciji su to argumenti ‘lp’ i ‘hp’ koji određuju interval unutar kojeg će sve nijanse postati bijele, a sve izvan toga će postati crno. Jedan primjer dobivene slike prikazuje slika 3.1.5. Nakon toga može krenuti pronalaženje pozicija znamenki. Ostatak funkcije ‘ocitajBroj’ bit će objašnjena u sljedećem koraku obrade.

Slika 3.1.2

Slika 3.1.3

Slika 3.1.4



Slika 3.1.5

```
def ocitajBroj(name, i, img, lp, hp):
        img2=cv2.medianBlur(img, 5)
        img2=cv2.equalizeHist(img2)
        img2=cv2.inRange(img2, lp, hp)
```

Slika 3.1.6

3.2 Pronalaženje znamenki

Prije nastavka prethodne funkcije, potrebno je definirati dvije pomoćne funkcije koje će se koristiti dalje unutar algoritma. To su funkcije: 'sizeOk' i 'prolaziKrozSredinu'. Obje funkcije vraćaju vrijednost 'true' ili 'false'.

Funkcija 'sizeOk' će služiti za odbacivanje svih objekata na slici koji ne zadovoljavaju određenu visinu, širinu i površinu, odnosno sve objekte koji sigurno ne mogu biti znamenka će se time zanemariti. Argumenti koje funkcija prima su visina i širina pravokutnika unutar kojega se možda nalazi znamenka i visina i širina slike. U slučajevima da je pravokutnik viši od 50% ili manji od 7% visine slike, smatra se da u pravokutniku nije znamenka te će se on odbaciti. Dalje ukoliko je širina pravokutnika veća od 30% širine slike ili manja od 1% širine slika također se odbacuje, kao I ako je površina veća od 10% ili manja od 0.1% površine slike.

Funkcija 'prolaziKrozSredinu' će reći da li se pravokutnik na slici nalazi unutar dviju granica, gornje i donje, ako prelazi te granice, odbacuje se. Gornja granica postavljena je na 20% visine slike počevši odozgo, a donja granica je postavljena na 80% visine slike također počevši odozgo. Drugim riječima svaki pravokutnik koji sadrži prvih 20% i zadnjih 20% redova pixela će se odbaciti, a samo oni sadržani unutar tih granica će se prihvatiti. Obje funkcije i njihov kod prikazani su na slici 3.2.1.

```
####################################
#     provjeravanje velicine       #
####################################
def sizeOk(dW, dH, imW, imH):
        maxH=imH*0.5
        minH=imH*0.07
        maxW=imW*0.3
        minW=imW*0.01

        if dW>maxW or dW<minW or dH>maxH or dH<minH:
                return False

        P=dW*dH
        refPH=imW*imH*0.1
        refPL=imW*imH*0.001

        if P>refPH or P<refPL:
                return False

        return True
```

```
#######################################
# odbaci sve sto ne prolazi sredinom #
#######################################
def prolaziKrozSredinu(y, h, height):
        hGornja=height*0.20
        hDonja=height*0.80

        if y<hGornja or (y+h)>hDonja:
                return False
        else:
                return True
```

Slika 3.2.1

Sada možemo dalje krenuti u pronalaženje znamenki. Sljedeći korak će biti na binarnoj slici koju imamo uokviritu u pravokutnike sve bijele površine, to znači da će svaka bijela površina bez obzira na njen oblik biti pohranjena u jedno polje u obliku 4 atributa: pozicija gornjeg lijevog vrha, visina i širina pravokutnika. Biblioteka openCV ima gotovu funkciju za pronalaženje i određivanje parametara svake bijele površine:

```
cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Slika 3.2.2 prikazuje upotrebu funkcije 'findContours' i ostatak algoritma za pronalaženje znamenki.

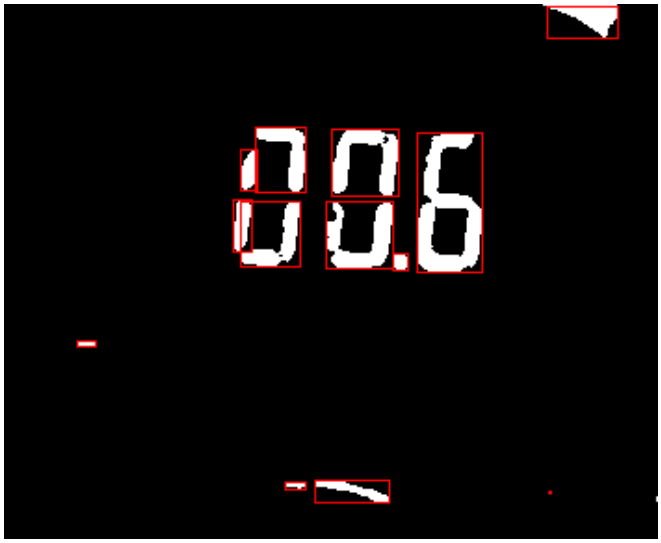
```
height, width=img2.shape
cnts, hierarchy=cv2.findContours(img2.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)


#PROLAZ-------1 izbacivanje premalih i prevelikih
digits=[]
for c in cnts:
        (x,y,w,h)=cv2.boundingRect(c)

        if sizeOk(w, h, width, height):
                digits.append(c)


#PROLAZ-------2 izbacivanje izvan okvira
digits2=[]
for c in digits:
        (x,y,w,h)=cv2.boundingRect(c)

        if prolaziKrozSredinu(y, h, height):
                digits2.append(c)


#PROLAZ-------3 poravnavanje
digits3=[]
Ymax=1000
Wmax=0
Hmax=0
for c in digits2:
        (x,y,w,h)=cv2.boundingRect(c)
        Ymax=y
        if w>Wmax: Wmax=w
        if h>Hmax: Hmax=h
```

Slika 3.2.2

Na početku za potrebe prethodno definirane dvije funkcije 'sizeOk' i 'prolaziKrozSredinu' potrebno je i odrediti dimenzije slike. Kao što slika prikazuje to se obavlja odmah na početku te nakon toga poziva funkcija 'findContours' koja će sve bijele površine 'spremiti' u varijablu 'cnts' u obliku polja.

Sada je cilj filtrirati sve bijele površine i odrediti na kojima su znamenke, a koje su smeće. To će se postići kroz 4 prolaza, a svodi se na prebacivanje iz jednog polja u drugo i pri tome odbacujući nepotrebne bijele površine.

Prije nego počnemo na slici 3.2.3 možemo vidjeti sliku koju bismo imali nakon poziva funkcije ‘findContours’ kada bismo te pravokutnike nacrtali.



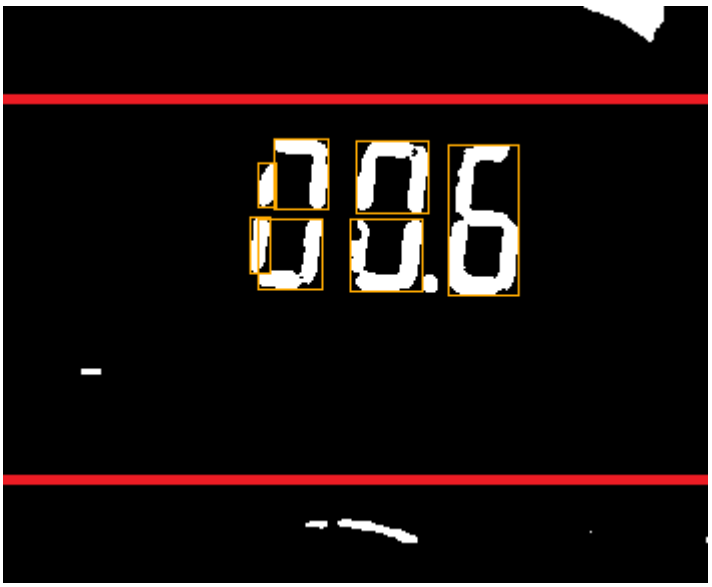
Slika 3.2.3

Sada slijedi prvi prolaz. U prvom prolazu ćemo odbaciti sve bijele površine čije dimenzije nam ne odgovaraju, to su sve premale i prevelike bijele mrlje na slici koje ne predstavljaju znamenku. Odbacit ćemo ih pomoću prethodno definirane funkcije ‘sizeOk’. To ćemo izvesti tako da definiramo novo prazno polje ‘digits’ i u ‘for’ petlji prođemo kroz sve bijele površine koje imamo u varijabli ‘cnts’. Jednostavnim ‘if’ ispitivanjem i funkcijom ‘sizeOk’. Za sve bijele površine koje zadovoljavaju dimenzije funkcije ‘sizeOk’ će vratiti ‘True’ i tako popuniti definirano prazno polje ‘digits’.

Slike 3.2.4

Ovime je završen prvi prolaz, a na slici 3.2.4 je prikazan rezultat.
Kao što se može vidjeti, donje bijele površine (smetnje) više nisu uokvirene, jer ne predstavljaju znamenku, a izbačene su jer njihove dimenzije ne odgovaraju kriterijima funkcije 'sizeOk'.

Slijedi drugi prolaz. U ovome ćemo prolazu na isti način kao u prethodnome izbaciti sve površine koje se nalaze skroz na vrhu slike ili skroz na dnu slike bez obzira na njihove dimenzije. Ovo ćemo napraviti pomoću također prethodno definirane funkcije 'prolaziKrozSredinu', a filtrirane bijele površine koje prihvaćamo i koje se nalaze negdje u sredini slike ćemo sada prebaciti u novo polje 'digits2'. Slika 3.2.5 prikazuje otprilike granice unutar kojih se prihvaćaju bijele površine, a sve izvan toga se odbacuju. Budući da na slici koju trenutno imamo 3.2.4 nemamo u tom području označenih (uokvirenih) površina, nemamo što izbaciti pa će sljedeći međurezultat biti ista slika, a polje 'digits2' će sadržavati sve isto što i polje 'digits'.
Razlog za ovaj korak je taj što će se često dogoditi da imamo najviše smetnji uz gornji i donji rub slike jer se pokaznici uglavnom protežu u širinu slike, a izvan toga gore i dolje imamo detalje koji ne predstavljaju pokaznik.



Slika 3.2.5

Nakon toga slijedi treći prolaz. Slika 3.2.6 prikazuje ostatak algoritma, treći i četvrti prolaz. Pretpostavljamo da su glavne smetnje uklonjene i da se negdje u sredini slike nalaze znamenke, u trećem koraku ćemo sve pravokutnike poravnati u odnosu na najviši i najširi jer su znamenge sve istih dimenzija u nizu.

Također može se desiti da neka znamenka nije potpuno cijela uokvirena, ovim prolazom ćemo takve pravokutnike detektirati i proširiti preko cijele znamenke.
Algoritam počinje ponovo definiranjem novog polja ‘digits3’ u koje ćemo ubaciti prihvatljive pravokutnike. Definirat ćemo i varijable Ymin, Wmax i Hmax.

```
#PROLAZ-------3 poravnavanje
digits3=[]
Ymin=1000
Wmax=0
Hmax=0
for c in digits2:
        (x,y,w,h)=cv2.boundingRect(c)
        if y<Ymin: Ymin=y
        if w>Wmax: Wmax=w
        if h>Hmax: Hmax=h

for c in digits2:
        (x,y,w,h)=cv2.boundingRect(c)
        if abs(Ymin-y)<height*0.05:
                x=x+w-Wmax
                h=Hmax+abs(Ymin-y)
                w=Wmax
                y=Ymin

                value, dp=citaj_znamenku(img2, x, y, w, h, 0)
                if value!=-1 and h>(height*0.2) and w>(width*0.05):
                        obj=Znamenka(x, w, y, h, value, dp)
                        digits3.append(obj)

#PROLAZ-------4 ocitavanje
digits3=sorted(digits3, key=lambda zn: zn.x)
Broj="";
for i in range(0, len(digits3)):
        if len(digits3)>(i+1) and (digits3[i].x+digits3[i].w)>digits3[i+1].x:
                continue

        if digits3[i].value!=-1: Broj+=str(digits3[i].value)
        else: Broj+="?"

        if digits3[i].dp==1: Broj+="."

return Broj
```

Slika 3.2.6

U prvoj ‘for’ petlji pronaći ćemo najširi i najduži pravokutnik te sve ostale poravnati u odnosu na njih, također ćemo pronaći Y koordinatu najvišeg i sve poravnati sa njime. Nakon toga sve znamenke koje nisu u potpunosti obuhvaćene pravokutnikom kao što se vidi na slici 3.2.5 (obje nule su polovično uokvirene), bit će potpuno obuhvaćene jer znamenka sa najvećom visinom i širinom je znamenka 6 i ona je potpuno uokvirena. Ako znamo da su sve znamenke jednake, možemo preostale dvije nule jednostavno uokviriti sa pravokutnikom od šestice, to ćemo napraviti upravo poravnavanjem sa njezinom visinom i širinom.
Nakon prolaza prve ‘for’ petlje znamo dimenzije najveće znamenke, u ovom slučaju šestice. Sada u drugoj petlji samo moramo ostale prevokutnike poravnati sa jednom y koordinatom, onom najvišom, jer ako uzmemo neku nižu, nemamo garanciju da će sve ostale koje su možda više biti potpuno obuhvaćene.

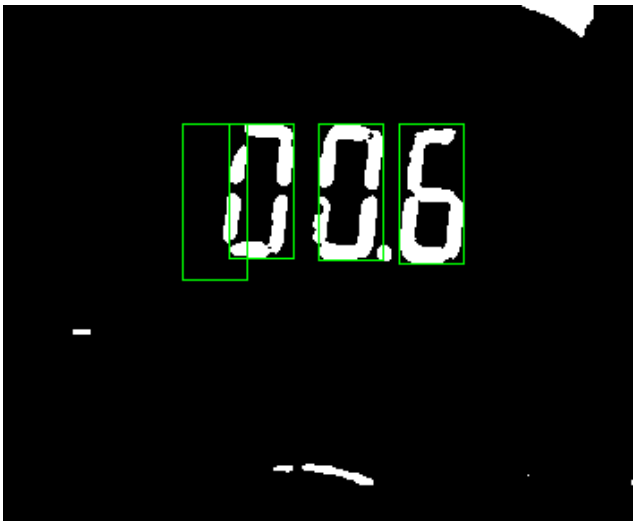
Da bismo uklonili pravokutnike koji se nalaze daleko ispod razine najviše znamenke, jednostavnim 'if' ispitivanjem ćemo ih preskočiti i oni neće biti spremljeni u novo polje 'digits3'. Svi čija y koordinata odstupa vise od 5% visine slike od najviše y koordinate će biti preskočeni. Oni pravokutnici koji zadovoljavaju prethodni kriterij, postavit ćemo im nove dimenzije, visinu i širinu da bi svi bili jednaki. Rezultat dobiven nakon ovoga prikazuje slika 3.2.8.

Odmah nakon toga očitavamo vrijednost funkcijom 'citaj_znamenku' koja će u sljedećem poglavlju biti detaljnije objašnjena. Funkcija vraća dva podatka, vrijednost znamenke 0-9 i 'True' ili 'False' za segment dp(točku). Ukoliko se znamenka ne može pročitati funkcija će vratiti vrijednost -1. Ako dobijemo vrijednost znamenke u interval [0, 9], a dimenzije su otprilike zadovoljavajuće, smatramo da se u tom pravokutniku nalazi ispravna znamenka i spremamo ju sa dimenzijama pravokutnika u polje 'digits3'. U ovo polje ćemo na malo drugačiji način spremati znamenke. U obliku objekta koji će sadržavati i veličine pravokutnika pored vrijednosti znamenke i točke. Ovo je potrebno raditi kako bismo kasnije mogli sortirati znamenke po redu kako su prikazane i na pokazniku (po x koordinati pripadajućeg pravokutnika). Zbog toga moramo definirati razred 'Znamenka' sa pripadnim varijablama: vrijednost znamenke, točka, pozicija pravokutnika, visina širina kao što je prikazano na slici 3.2.7. Sada preostaje još samo četvrti korak u kojem ćemo očitati konačnu vrijednost broja prikazanog na pokazniku.

```
class Znamenka:
        x=0
        w=0
        y=0
        h=0
        value=-1
        dp=0

        def __init__(self, x, w, y, h, value, dp):
                self.value=value
                self.dp=dp
                self.x=x
                self.w=w
                self.y=y
                self.h=h
```

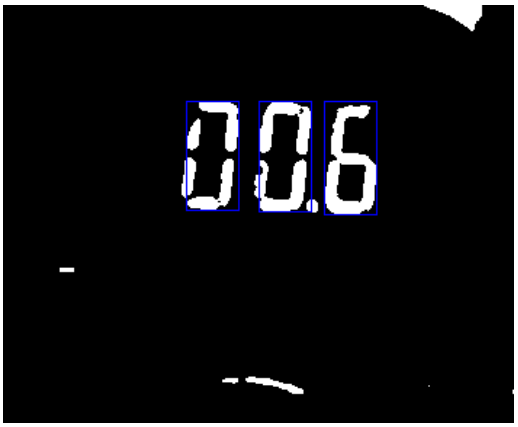
Slika 3.2.7

Četvrti korak sastoji se od sortiranja kako je prethodno bilo navedeno. Sortiranje se odvija automatski jednom naredbon ‘sorted’ po koordinati x. Nakon toga u polju ‘digits3’ imamo znamenke poredane točno kako su prikazane na pokazniku. Potrebno je samo u jednoj ‘for’ petlji proći i sve znamenke ubaciti u jednu varijablu ‘Broj’ tipa ‘string’. Ukoliko je uz znamenku i točka, nakon dodavanja znamenke dodajemo i točku.



Slika 3.2.8

Prije nego ubacimo sve znamenke u varijablu tipa ‘string’ postoji još jedna stvar koju je potrebno napraviti. Kao što se vidi na slici 3.2.8 postoji i mogućnost za preklapanje znamenki. Ovo također možemo riješiti jednim ‘if’ ispitivanjem: ako sljedeća znamenka ulazi u polje sadašnje, sadašnju preskoči i nemoj ju dodati u varijablu tipa ‘string’. Tako će prvi lijevi pravokutnik biti preskočen jer ulazi u polje sljedećega koji sadrži znamenku 0. Nakon toga u varijabli ‘Broj’ ostaju znamenke kao da je očitana slika 3.2.9. Time je algoritam očitavanja završen i funkcija ‘ocitajBroj’ sa slike 3.1.6 završava i vraća broj koji je očitan (varijabli ‘Broj’ tipa ‘string’).



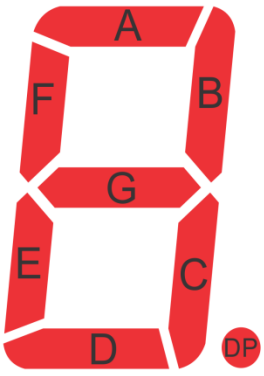
Slika 3.2.9

3.3 Očitavanje brojčane vrijednosti

Očitavanje znamenke vrši funkcija: ocitaj_znamenku. Funkcija kao parametre prima koordinate(gornji lijevi vrh) pravokutnika, dimenzije pravokutnika i samu sliku koja je rezultat prethodnih obrada, a vraća kao rezultat dvije varijable. Prva varijabla sadrži očitanu znamenku (0-9) ili -1 ukoliko se znamenka ne može očitati. Druga varijabla sadrži binarnu vrijednost koja određuje da li je upaljen 'dp' segment (decimalna točka nakon znamenke).

Deklaracija funkcije prikazana je na slici 3.3.4, vidimo da se odmah unutar funkcije definiraju varijable, za svaki segment jedna koja će sadržavati binarno stanje tog segmenta, te se sve inicijaliziraju na 0, a ako se naknadno uspostavi da određeni segmenti svijetle, određena varijabla će biti postavljena 1.

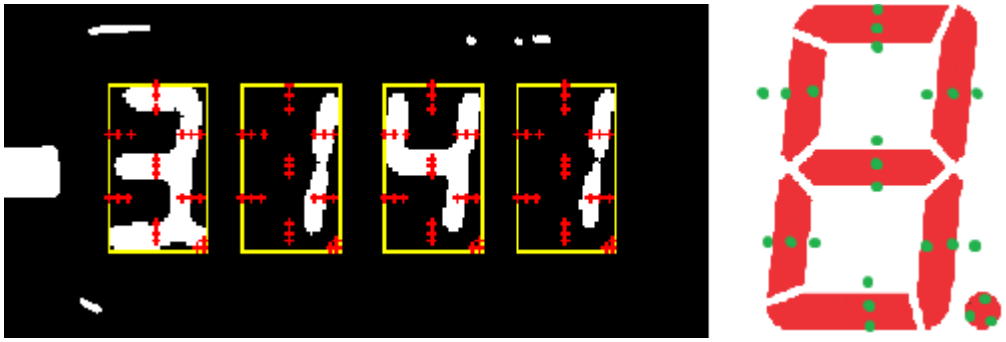
Nakon toga potrebno je odrediti relativne lokacije točaka koje će se uzimati za očitavanje pojedinih segmenata. Slika 3.3.1 prikazuje 7 segmentni pokaznik i njegove segmente, te na osnovu toga treba procijenit na koji način će se odrediti i očitavati stanje segmenata. U idealnom slučaju bismo mogli negdje u sredini svakog segmenta postaviti lokaciju točke i očitati vrijednost, te točke, na taj način popuniti tablicu stanja pojedinih segmenata (za svaki segment 1 ili 0 ovisno o tome da li je upaljen ili ugašen) i očitati iz toga brojčanu vrijednost. Međutim slike dobivene prethodnim obradama najčešće neće biti ni blizu idealnom izgledu i položaju segmenata pa njihova lokacija unutar dobivenog pravokutnika može odstupati, također i širina segmenata može odstupati pa zbog toga ako bismo uzimali jednu točku za svaki segment mogli bismo promašiti lokaciju segmenta i time očitati krivu vrijednost. Slika 3.3.2 prikazuje otprilike izgled segmenata u realnosti.



Slika 3.3.1

Slika 3.3.2

Zbog navedene opasnosti od pogreške algoritam će uzimati 3 točke za svaki segment kako bi se smanjila vjerojatnost pogreške. Točke će biti u nizu za vodoravne segmente okomite, a za okomite segmente vodoravne. Ako bi barem jedna točka bila bijele boje, a ostale crne, zaključuje se da je segment upaljen. Slika 3.3.3 prikazuje pozicije točaka za očitavanje segmenata.



Slika3.3.3

Slijedi programski kod za određivanje lokacija točaka za očitavanje: slika 3.3.4. Definiramo za svaki segment jednu varijablu koja će predstavljati taj segment sa samo dva stanja: 0 ili 1.

```
################################################
#   ocitavanje segmenata za jednu znamenku   #
################################################
def citaj_znamenku(image, x, y, w, h, draw):
        a=0
        b=0
        c=0
        d=0
        e=0
        f=0
        g=0
        dp=0
```

Slika 3.3.4

Na slici 3.3.4 u deklaraciji funkcije je još jedan argument ‘draw’ koji se koristi kao zastavica(stanje 0 ili 1) za crtanje samog pravokutnika i točaka. Za sam rad programa ovo se ne koristi i može se izostaviti jer nije potrebno crtati pravokutnik i točke, ali može pomoći kod razvoja i testiranja algoritma. Kod normalnog rada argument ‘draw’ uvijek ima vrijednost 0 te se pravokutnik i točke ne iscrtavaju na slici.

Sljedeći korak je određivanje lokacija samih točaka preko kojih će se postavljati vrijednosti definiranih varijabli segmenata: a,b, c…g, dp. Točke se određuju relativno u odnosu na visinu i širinu zadanog pravokutnika kao što je prikazano na slici 3.3.5.

```
ax=x+int(w*0.5)
ay=y+int(h*0.02)
ay2=y+int(h*0.07)
ay3=y+int(h*0.15)
bx=x+int(w*0.75)
bx2=x+int(w*0.85)
bx3=x+int(w*0.95)
by=y+int(h*0.3)
cx=x+int(w*0.75)
cx2=x+int(w*0.85)
cx3=x+int(w*0.95)
cy=y+int(h*0.7)
dx=x+int(w*0.5)
dy=y+int(h*0.85)
dy2=y+int(h*0.90)
dy3=y+int(h*0.95)
ex=x+int(w*0.02)
ex2=x+int(w*0.1)
ex3=x+int(w*0.2)
ey=y+int(h*0.7)
fx=x+int(w*0.02)
fx2=x+int(w*0.1)
fx3=x+int(w*0.25)
fy=y+int(h*0.3)
gx=x+int(w*0.5)
gy=y+int(h*0.45)
gy2=y+int(h*0.5)
gy3=y+int(h*0.55)
dpx=x+int(w*0.92)
dpy=y+int(h*0.98)
dpx2=x+int(w*0.98)
dpy2=y+int(h*0.94)
dpx3=x+int(w*0.98)
dpy3=y+int(h*0.98)
```

Slika 3.3.5

Na primjer za segment G koji se nalazi na sredini, prolazit će 3 točke okomito poredane (jer je segment vodoravan). Zbog toga će sve tri imati istu x koordinatu, a iznos će biti gx=x+w*0.5, dakle na poziciju pravokutnika radimo još pomak za njegovu širinu pomnoženu za konstantom 0.5 čime će koordinata x biti negdje u sredini pravokutnika. Funkcija 'int' će samo zaokružiti vrijednost i pretvoriti ponovo u cijeli broj jer radimo sa pixelima. Koordinata y će biti u 3 varijable jer imamo i 3 točke okomito poredane. Formula za računanje y koordinata je ista, samo ovaj put uzimamo pomak u odnosu na visinu pravokutnika, time će gy biti na 45% visine, gy2 na 50%, točno na polovici pravokutnika, a gy3 niže na 55% visine pravokutnika. Poziciju svih točaka prikazuje slika 3.3.3 desno. Nakon ovoga sve su točke postavljene, sada samo još preostaje očitati svaki pixel sa tih lokacija i na osnovu toga ažurirati varijable segmenata 'a' do 'dp'. To se radi jednostavnim ispitivanjem vrijednosti pixela na prethodno definiranim pozicijama: slika 3.3.6.

```
if image[ay,ax]==255 or image[ay2,ax]==255 or image[ay3,ax]==255: a=1
if image[by,bx]==255 or image[by,bx2]==255 or image[by,bx3]==255: b=1
if image[cy,cx]==255 or image[cy,cx2]==255 or image[cy,cx3]==255: c=1
if image[dy,dx]==255 or image[dy2,dx]==255 or image[dy3,dx]==255: d=1
if image[ey,ex]==255 or image[ey,ex2]==255 or image[ey,ex3]==255: e=1
if image[fy,fx]==255 or image[fy,fx2]==255 or image[fy,fx3]==255: f=1
if image[gy,gx]==255 or image[gy2,gx]==255 or image[gy3,gx]==255: g=1
if image[dpy,dpx]==255 or image[dpy2,dpx2]==255 or image[dpy3,dpx3]==255: dp=1
```

Slika 3.3.6

Nakon postavljanja varijabli segmenata možemo očitati koji je broj prikazan u zadanom pravokutniku jednostavnim ispitivanjem kombinacija koje predstavljaju određeni broj, slika 3.3.7. Ova funkcija za očitavanje znamenke koja je definirana na slici 3.3.4 vraća dvije vrijednosti što možemo vidjeti na slici 3.3.7: prva vrijednost je cijeli broj od 0-9 uključivo, a druga vrijednost je varijabla dp koja označava da li je i

točka prikazana, 1 ako je i 0 ako nije.

U slučaju da se pročita bilo koja druga kombinacija segmenata koja ne odgovara niti jednom broju, funkcija vraća vrijednost (-1,-1) što znači da se znamenka ne može pročitati. Kasnije tokom programa za ovakve slučaje će se odbacivati i sve ostale znamenke koje su možda ispravno pročitane iz razloga što bi taj broj u konačnici kao završni rezultat bio pogrešan pa nema smisla ispisivati dio broja.

```
if(a==1 and b==1 and c==1 and d==1 and e==1 and f==1 and g==0): return 0, dp
elif(a==0 and b==1 and c==1 and d==0 and e==0 and f==0 and g==0): return 1, dp
elif(a==1 and b==1 and c==0 and d==1 and e==1 and f==0 and g==1): return 2, dp
elif(a==1 and b==1 and c==1 and d==1 and e==0 and f==0 and g==1): return 3, dp
elif(a==0 and b==1 and c==1 and d==0 and e==0 and f==1 and g==1): return 4, dp
elif(a==1 and b==0 and c==1 and d==1 and e==0 and f==1 and g==1): return 5, dp
elif(a==1 and b==0 and c==1 and d==1 and e==1 and f==1 and g==1): return 6, dp
elif(a==1 and b==1 and c==1 and d==0 and e==0 and f==0 and g==0): return 7, dp
elif(a==1 and b==1 and c==1 and d==0 and e==0 and f==1 and g==0): return 7, dp
elif(a==1 and b==1 and c==1 and d==1 and e==1 and f==1 and g==1): return 8, dp
elif(a==1 and b==1 and c==1 and d==1 and e==0 and f==1 and g==1): return 9, dp
else: return -1, -1
```

Slika 3.3.7

4. Upute za instalaciju

Za funkcioniranje sustava potrebna je jedna kamera i bilo kakvo računalo. Za ovaj primjer koristi se GEMBRID CAM90U web kamera rezolucije 5MP, iako se za obradu koriste fotografije znatno niže rezolucije ispod 1MP. Za UNIX operacijske sustave postoji "fswebcam" upravljački program koji se može pokrenut iz terminal, a instalacija se može izvršit unosom naredbe u terminal:

sudo apt-get install fswebcam

Kada process završi ukoliko je kamera spojena na računalo može se provjeriti funkcionalnost unosom naredbe:

fswebcam ime.jpg

Nakon toga u direktoriju u kojem je terminal pozicioniran trebala bi se pojaviti slika "ime.jpg" pri čemu nastavak ".jpg" ne označava format slike nego spada pod ime jer kamera uvijek pohranjuje sliku u jpg format osim ako se eksplicitno ne navede drugi. Više informacija o načinima rada upravljačkog programa na "man" stranici unosom: man fswebcam.

Primjer okidanja slike:

```
marin@Samsung:~/Desktop$ fswebcam ime.jpg
--- Opening /dev/video0...
Trying source module v4l2...
/dev/video0 opened.
No input was specified, using the first.
Adjusting resolution from 384x288 to 352x288.
--- Capturing frame...
Captured frame in 0.00 seconds.
--- Processing captured image...
Writing JPEG image to 'ime.jpg'.
marin@Samsung:~/Desktop$
```

Sljedeće je potrebno instalirat phyton interpreter koji je po defaultu instaliran na većini UNIX operacijskih sustava. Provjeru možemo napravit unošenjem naredbe "python --version" nakon čega bi na terminal trebala biti ispisana verzija interpretera:

```
marin@Samsung:~/Desktop$ python --version
Python 2.7.12
```

Ukoliko nije može se instalirati sljedećim postupkom:

wget https://www.python.org/ftp/python/2.7.12/Python-

2.7.12.tgz

tar xzf Python-2.7.12.tgz

cd Python-2.7.12

sudo ./configure

sudo make altinstall

Preostalo je još instalirat OpenCV biblioteku koja sadrži potrebne funkcije za obradu slike:

sudo apt-get install python-opencv

Ukoliko je sve uspješno završeno možemo provjeriti da li biblioteka ispravno radi, to možemo provjeriti jednostavnim python programom koji će ispisati broj verzije biblioteke. Da bi to napravili potrebno je kreirati datoteku "ocv_verzija.py" unutar koje treba upisati text:

```
import cv2
print  cv2.__version__
```

Nakon što spremimo datoteku, u terminalu je potrebno izvršiti program upisom:

python ocv_verzija.py

Rezultat izvršavanja bi trebao izgledati kao na slici:

```
marin@Samsung:~/Desktop$ python ocv_verzija.py
2.4.9.1
marin@Samsung:~/Desktop$
```

Sada kada je sve potrebno za rad sustava instalirano i ispravno radi možemo staviti i pokrenuti sam program. Program se sastoji od jedne bash skripte (nastavak .sh) i samog programskog koda (nastavak .py). Ove dvije datoteke se mogu smjestiti bilo gdje unutar datotečnog sustava pod uvjetom da su obje u istom direktoriju i da je dozvoljeno pisanje unutar istog direktorija kako bi program mogao pohranjivati slike. Sustav se pokreće iz terminal pokretanjem skripte:

./run.sh

Te nakon toga periodično očitava vrijednost pokaznika i rezultat ispisuje na terminal.

Zaključak

Automatsko očitavanje sedam segmentnog pokaznika nije jako složen algoritam ukoliko već imamo gotove biblioteke za obradu slike, u ovom slučaju OpenCV. OpenCV biblioteka je također i podržana u C++ programskom jeziku koji bi možda bio prigodniji za profesionalnu upotrebu kao i izvedba na drugačijem hardware (bez korisničkih operacijskih sustava) što bi omogućilo puno manju potrošnju resursa. Ovaj sustav je napisan u programskom jeziku Python te algoritam nije vrlo kompliciran za shvatiti, vrlo je fleksibilan i lako se može instalirati na bilo koje računalo. Kamera za potrebe očitavanja ne mora biti visoke kvalitete, već je i bolje ukoliko su slike manje rezolucije zbog brže obrade. Ovaj sustav koristi jeftiniju web kameru koja okida slike rezolucije približno 300x200 pixela što je sasvim dovoljno za kvalitetno očitavanje. Kvaliteta rada sustava dosta ovisi o uvjetima (pozadinskom osvijetljenju), a kakvi su uvjeti prigodniji ovisi o tipu pokaznika. Postoje pokaznici koji emitiraju svijetlost (izvedeni LED diodama). Za takve pokaznike idealni uvjeti su tamnija okolina te im nije potrebno mnogo vanjske svjetlosti i manje su osjetljivi na smetnje i intenzitet vanjske svjetlosti. Također boja ovakvih pokaznika uopće ne igra važnu ulogu tj. nije potrebno voditi računa o tome prilikom implementacije sustava. Drugi tip pokaznika su oni izvedeni LCD tehnologijom (tekućim kristalima). Ovaj tip pokaznika ne emitira svjetlost te su njihovi segmenti vrlo tamni (u idealnom slučaju crne boje). Zbog toga sustav za očitavanje ovakvih pokaznika treba imati dobro osvijetljenje kako bi contrast bio bolji tj kako bi se ti tamni segmenti bolje isticali. Očitavanje ovakvih pokaznika je nešto teže jer ima dosta smetnji koje je teško otkloniti pa ovo zahtijeva strože uvjete rada zbog čega je ponekad kvaliteta očitavanja lošija nego za prethodni slučaj sa LED pokaznikom. Pošto se za izvedbu ovog sustava koristi dosta jače računalo nego što je potrebno, moguća nadogradnja bila bi izvedba i web server što bi omogućilo povezivanje sustava na mreži i tima flaksibilniji pristup informacijama očitavanja. U komercijalnoj upotrebi možemo naći sustave bazirane na manjim računalima koji se napajaju preko baterije. Ovakvi sustavi su puno efikasniji i računalo se pali samo prilikom okidanja slike i obrade u određenim

vremenskim intervalima dok je inače u načinu rada sa zanemarivom potrošnjem električne energije.

LITERATURA

https://en.wikipedia.org/wiki/Raspberry_Pi

https://www.raspberrypi.org/

https://opencv-python-tutroals.readthedocs.io/en/lates

http://www.pyimagesearch.com/2017/02/13/recognizing-digits-with-opencv-and-pytho

Sustav za automatsko optičko očitanje mjernog uređaja realiziran ugradbenim računalom

Sažetak

Sustav za automatsko očitavanje izveden je ugradbenim računalom Raspberry Pi 3 (sa UNIX operacijskim sustavom) i standardnom web kamerom spojenom na USB port računala. Programska potpora sastoji se od 'bash' skripte i glavnog programa za očitavanje: Python skripte. Program prolazi kroz nekoliko koraka: okidanje slike i pokretanje programa za očitavanje ('bash' skripta), te predobrada, lociranje znamenki i očitavanje znamenki (Python skripta). Sustav periodično okida slike i očitava brojeve te ih ispisuje na terminal. Frekvencija okidanja slike je podesiva. Sustav funkcionira samo za sedam segmentne pokaznika te zahtjeva instaliranu potporu na računalu: Python interpreter i OpenCV biblioteka koja sadrži funkcije za obradu slike.

System for automatic optical reading of the measuring device realized by the embedded computer

Abstract

System for automatically sensing is carried out by embedded computer Raspberry Pi 3 (with UNIX operating system) and standard web cam connected to computer USB port. Software consists of 'bash' script and main sensing program: Python script. Program goes through several steps: picture triggering and launching sensing program ('bash' script), pretreatment, locating digits and sensing digits (Python script). System periodically triggers pictures and senses numbers which are then printed on the terminal. The picture triggering frequency is adjustable. The System functions only for seven segment pointing devices and requires installed computer support : Python interpreter and OpenCv library which contains functions for picture editing.

Programski kod cijelog sustava

Datoteka "run.sh":

```
#! /bin/bash
rm *.png
time=2
name="tmpimg.jpg"

while [ 1 ]
do
        fswebcam $name
        python ocitavanje.py $name 0
        echo "\n"
        sleep $time
done
```

Datoteka “ocitavanje.py”:

```
import cv2
import sys
import numpy as np

####################################
#              provjeravanje velicine              #
####################################
def sizeOk(dW, dH, imW, imH):
        maxH=imH*0.5
        minH=imH*0.07
        maxW=imW*0.3
        minW=imW*0.01

        if dW>maxW or dW<minW or dH>maxH or dH<minH:
                return False

        P=dW*dH
        refPH=imW*imH*0.1
        refPL=imW*imH*0.001

        if P>refPH or P<refPL:
                return False

        return True


#########################################
#       odbaci sve sto ne prolazi sredinom          #
#########################################
def prolaziKrozSredinu(y, h, height):
        hGornja=height*0.20
        hDonja=height*0.80

        if y<hGornja or (y+h)>hDonja:
                return False
        else:
                return True
```

```
##############################################
#                ocitavanje segmenata za jednu znamenku          #
##############################################
def citaj_znamenku(image, x, y, w, h, draw):
        a=0
        b=0
        c=0
        d=0
        e=0
        f=0
        g=0
        dp=0

        ax=x+int(w*0.5)
        ay=y+int(h*0.02)
        ay2=y+int(h*0.07)
        ay3=y+int(h*0.15)
        bx=x+int(w*0.75)
        bx2=x+int(w*0.85)
        bx3=x+int(w*0.95)
        by=y+int(h*0.3)
        cx=x+int(w*0.75)
        cx2=x+int(w*0.85)
        cx3=x+int(w*0.95)
        cy=y+int(h*0.7)
        dx=x+int(w*0.5)
        dy=y+int(h*0.85)
        dy2=y+int(h*0.90)
        dy3=y+int(h*0.95)
        ex=x+int(w*0.02)
        ex2=x+int(w*0.1)
        ex3=x+int(w*0.2)
        ey=y+int(h*0.7)
        fx=x+int(w*0.02)
        fx2=x+int(w*0.1)
        fx3=x+int(w*0.25)
        fy=y+int(h*0.3)
        gx=x+int(w*0.5)
        gy=y+int(h*0.45)
        gy2=y+int(h*0.5)
        gy3=y+int(h*0.55)
        dpx=x+int(w*0.92)
        dpy=y+int(h*0.98)
        dpx2=x+int(w*0.98)
        dpy2=y+int(h*0.94)
        dpx3=x+int(w*0.98)
        dpy3=y+int(h*0.98)
```

```
        if image[ay,ax]==255 or image[ay2,ax]==255 or    image[ay3,ax]==255:
a=1
        if image[by,bx]==255 or image[by,bx2]==255 or image[by,bx3]==255:
b=1
        if image[cy,cx]==255 or image[cy,cx2]==255 or image[cy,cx3]==255: c=1
        if image[dy,dx]==255 or image[dy2,dx]==255 or image[dy3,dx]==255:
d=1
        if image[ey,ex]==255 or image[ey,ex2]==255 or image[ey,ex3]==255:
e=1
        if image[fy,fx]==255 or image[fy,fx2]==255 or image[fy,fx3]==255: f=1
        if image[gy,gx]==255 or image[gy2,gx]==255 or image[gy3,gx]==255:
g=1
        if image[dpy,dpx]==255 or image[dpy2,dpx2]==255 or
image[dpy3,dpx3]==255: dp=1


        if draw!=0:
                p=1
                im=cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
                color=(0, 0, 255)
                cv2.rectangle(im,(x,y),(x+w,y+h),(0,255,255),0)
                cv2.circle(im, (ax, ay), p,color,-1)
                cv2.circle(im, (ax, ay2), p,color,-1)
                cv2.circle(im, (ax, ay3), p,color,-1)
                cv2.circle(im, (bx, by), p,color,-1)
                cv2.circle(im, (bx2, by), p,color,-1)
                cv2.circle(im, (bx3, by), p,color,-1)
                cv2.circle(im, (cx, cy), p,color,-1)
                cv2.circle(im, (cx2, cy), p,color,-1)
                cv2.circle(im, (cx3, cy), p,color,-1)
                cv2.circle(im, (dx, dy), p,color,-1)
                cv2.circle(im, (dx, dy2), p,color,-1)
                cv2.circle(im, (dx, dy3), p,color,-1)
                cv2.circle(im, (ex2, ey), p,color,-1)
                cv2.circle(im, (ex3, ey), p,color,-1)
                cv2.circle(im, (ex, ey), p,color,-1)
                cv2.circle(im, (fx, fy), p,color,-1)
                cv2.circle(im, (fx2, fy), p,color,-1)
                cv2.circle(im, (fx3, fy), p,color,-1)
                cv2.circle(im, (gx, gy), p,color,-1)
                cv2.circle(im, (gx, gy2), p,color,-1)
                cv2.circle(im, (gx, gy3), p,color,-1)
                cv2.circle(im, (dpx, dpy), p,color,-1)
                cv2.circle(im, (dpx2, dpy2), p,color,-1)
                cv2.circle(im, (dpx3, dpy3), p,color,-1)

                cv2.imwrite('seg'+str(draw)+'.png', im)
                print  str(a)+str(b)+str(c)+str(d)+str(e)+str(f)+str(g)
```

```
        if(a==1 and b==1 and c==1 and d==1 and e==1 and f==1 and g==0):
return 0, dp
        elif(a==0 and b==1 and c==1 and d==0 and e==0 and f==0 and g==0):
return 1, dp
        elif(a==1 and b==1 and c==0 and d==1 and e==1 and f==0 and g==1):
return 2, dp
        elif(a==1 and b==1 and c==1 and d==1 and e==0 and f==0 and g==1):
return 3, dp
        elif(a==0 and b==1 and c==1 and d==0 and e==0 and f==1 and g==1):
return 4, dp
        elif(a==1 and b==0 and c==1 and d==1 and e==0 and f==1 and g==1):
return 5, dp
        elif(a==1 and b==0 and c==1 and d==1 and e==1 and f==1 and g==1):
return 6, dp
        elif(a==1 and b==1 and c==1 and d==0 and e==0 and f==0 and g==0):
return 7, dp
        elif(a==1 and b==1 and c==1 and d==0 and e==0 and f==1 and g==0):
return 7, dp
        elif(a==1 and b==1 and c==1 and d==1 and e==1 and f==1 and g==1):
return 8, dp
        elif(a==1 and b==1 and c==1 and d==1 and e==0 and f==1 and g==1):
return 9, dp
        else: return -1, -1


class Znamenka:
        x=0
        w=0
        y=0
        h=0
        value=-1
        dp=0

        def __init__(self, x, w, y, h, value, dp):
                self.value=value
                self.dp=dp
                self.x=x
                self.w=w
                self.y=y
                self.h=h


def ocitajBroj(name, i, img, lp, hp):
        cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_01.png', img)
        img2=cv2.medianBlur(img, 5)
        cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_02.png', img2)
        img2=cv2.equalizeHist(img2)
        cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_03.png', img2)
        img2=cv2.inRange(img2, lp, hp)
```

```
        cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_04.png', img2)

        height, width=img2.shape
        cnts, hierarchy=cv2.findContours(img2.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

        img3=cv2.cvtColor(img2, cv2.COLOR_GRAY2RGB)
        img4=cv2.cvtColor(img2, cv2.COLOR_GRAY2RGB)
        img5=cv2.cvtColor(img2, cv2.COLOR_GRAY2RGB)
        img6=cv2.cvtColor(img2, cv2.COLOR_GRAY2RGB)
        img7=cv2.cvtColor(img2, cv2.COLOR_GRAY2RGB)
        img8=cv2.cvtColor(img2, cv2.COLOR_GRAY2RGB)


        #PROLAZ-------1 izbacivanje premalih i prevelikih
        digits=[]
        for c in cnts:
                (x,y,w,h)=cv2.boundingRect(c)
                cv2.rectangle(img3,(x,y),(x+w,y+h),(0,0,255),0)#

                if sizeOk(w, h, width, height):
                        digits.append(c)
                        cv2.rectangle(img4,(x,y),(x+w,y+h),(0,170,255),0)#

        #cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_1.png', img3)
        #cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_2.png', img4)

        #PROLAZ-------2 izbacivanje izvan okvira
        digits2=[]
        for c in digits:
                (x,y,w,h)=cv2.boundingRect(c)
                cv2.rectangle(img5,(x,y),(x+w,y+h),(0,255,255),0)

                if prolaziKrozSredinu(y, h, height):
                        digits2.append(c)

        #cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_3.png', img5)


        #PROLAZ-------3 poravnavanje
        digits3=[]
        Ymin=1000
        Wmax=0
        Hmax=0
        for c in digits2:
                (x,y,w,h)=cv2.boundingRect(c)
                if y<Ymin: Ymin=y
                if w>Wmax: Wmax=w
                if h>Hmax: Hmax=h
```

```
        for c in digits2:
                (x,y,w,h)=cv2.boundingRect(c)
                if abs(Ymin-y)<height*0.05:
                        x=x+w-Wmax
                        h=Hmax+abs(Ymin-y)
                        w=Wmax
                        y=Ymin
                        cv2.rectangle(img6,(x,y),(x+w,y+h),(0,255,0),0)

                        value, dp=citaj_znamenku(img2, x, y, w, h, 0)
                        if value!=-1 and h>(height*0.2) and w>(width*0.05):
                                cv2.rectangle(img7,(x,y),(x+w,y+h),(255,255,0),0)
                                obj=Znamenka(x, w, y, h, value, dp)
                                digits3.append(obj)


        #cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_4.png', img6)
        #cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_5.png', img7)


        #PROLAZ-------4 ocitavanje
        digits3=sorted(digits3, key=lambda zn: zn.x)
        Broj="";
        for i in range(0, len(digits3)):
                if len(digits3)>(i+1) and (digits3[i].x+digits3[i].w)>digits3[i+1].x:
                        continue


        cv2.rectangle(img8,(digits3[i].x,digits3[i].y),(digits3[i].x+digits3[i].w,digi
ts3[i].y+digits3[i].h),(255,0,0),0)

                if digits3[i].value!=-1: Broj+=str(digits3[i].value)
                else: Broj+="?"

                if digits3[i].dp==1: Broj+="."


        if Broj!="":
                cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_0.png', img2)
                cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_1.png', img3)
                cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_2.png', img4)
                cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_3.png', img5)
                cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_4.png', img6)
                cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_5.png', img7)
                cv2.imwrite('R'+name[5:9]+'_'+str(i)+'_6.png', img8)

        return Broj
```

```
##################################
###########      Start      #########
##################################
img=cv2.imread(sys.argv[1], cv2.CV_LOAD_IMAGE_GRAYSCALE)

parametri=[(250,255), (245,255), (240,255), (235,255), (230,255), (225,255),
(220,255),
          (0,10), (0,15), (0,20), (0,25), (0,30), (0,35), (0,40), (0,45), (0,50), (0,55),
(0,60)]

broj=""
for i in range(0, len(parametri)):
        broj=ocitajBroj(sys.argv[1], i, img, parametri[i][0], parametri[i][1])
        if broj!="": break


print broj
```