SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

SLIJEĐENJE OBJEKTA MOBILNIM ROBOTOM UPRAVLJANIM POMOĆU MOBILNOG TELEFONA

Diplomski rad

Antonio Lončar

Osijek, 2015.

ETFOS
ELEKTROTEHNIČKI FAKULTET OSIJEK
Sveučilište Josipa Jurja Strossmayera u Osijeku

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek,

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Antonio Lončar
Studij, smjer:	Računarstvo, Procesno računarstvo
Mat. br. studenta, godina upisa:	D-595R, 2013
Mentor:	izv.prof.dr.sc. Robert Cupec
Sumentor:	
Predsjednik Povjerenstva:	doc.dr.sc. Emmanuel Karlo Nyarko
Član Povjerenstva:	doc.dr.sc. Damir Filko
Naslov diplomskog rada:	Slijeđenje objekta mobilnim robotom upravljanim pomoću mobilnoga telefona
Primarna znanstvena grana rada:	procesno računarstvo
Sekundarna znanstvena grana (ili polje) rada:	umjetna inteligencija
Zadatak diplomskog rada:	Izraditi mobilnu platformu upravljanu pomoću mobilnog telefona i mikrokontrolera koja će slijediti zadani objekt držeći ga u vidnom polju kamere mobilnog telefona.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: Postignuti rezultati u odnosu na složenost zadatka: Jasnoća pismenog izražavanja: Razina samostalnosti:

Potpis sumentora:	Potpis mentora:

Dostaviti:

1. Studentska služba

U Osijeku, godine Potpis predsjednika Odbora:

ELEKTROTEHNIČKI FAKULTET OSIJEK
Sveučilište Josipa Jurja Strossmayera u Osijeku

IZJAVA O ORIGINALNOSTI RADA

Osijek,

Ime i prezime studenta:	Antonio Lončar
Studij :	Računarstvo
Mat. br. studenta, godina upisa:	D-595R, 2013

Ovom izjavom izjavljujem da je rad pod nazivom**:**
Slijeđenje objekta mobilnim robotom upravljanim pomoću mobilnoga telefona

izrađen pod vodstvom mentoraizv.prof.dr.sc. Roberta Cupeca

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.

Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Zadatke koje ljudi rješavaju svakodnevno i bez razmišljanja, za robota mogu predstavljati veliki problem. Malo dijete staro mjesec dana prepoznaje majčin glas, a već sa četiri mjeseca ima dovoljno razvijeno osjetilo vida da može pratiti predmet koji se pomiče. Ideja diplomskoga rada je napraviti robotska kolica malih dimenzija koja detektiraju objekt na slici te ga prate. Za upravljanje robotskim kolicima koristi se Arduino Uno platforma koja ima ugrađeni Atmega328 mikrokontroler. Međutim, Arduino Uno nema procesorsku moć koja može samostalno obrađivati sliku s kamere niti ima kameru. Prijenosno računalo ima veliku moć obrade podataka i kameru, ali njegova veličina predstavlja problem. Stoga se u ovome radu za obradu slike koristi pametni telefon s Android operacijskim sustavom i OpenCV4Android biblioteka. Mogli smo primjetiti da je proteklih godina razvoj mobilnih telefona doživio strahovit tehnološki napredak. Pametni telefon sa svojim malim dimenzijama, performansama i ugrađenom kamerom je idealan uređaj za ovaj zadatak. Pametni telefon je *master* uređaj, koji prima ulazne podatke s kamere, obrađuje ih te na temelju njih šalje poruke preko bluetooth-a Arduinu. Arduino je *slave* uređaj koji prima poruke bluetooth-om s pametnoga telefona, obrađuje poruke i upravlja istosmjernim motorima na robotskim kolicima. Ovisno o dobivenim porukama robotska kolica se pomiču naprijed, nazad, lijevo ili desno i prate odabrani objekt.

Diplomski rad je podijeljen u pet poglavlja, prvo poglavlje opisuje kako i koje boje ljudi vide te prostore boja u računalnim sustavima. Drugo poglavlje opisuje metodu detekcije objekta na slici. U četvrtome poglavlju je navedeno sklopovlje koje se koristi i krajnji izgled robotskih kolica. Peto poglavlje opisuje klase i metode koje se koriste u izradi programa, algoritme i njihovu implementaciju na Arduino i pametni telefon te konačni izgled aplikacije za pametni telefon.

2. BOJA U SUSTAVIMA RAČUNALNOG VIDA

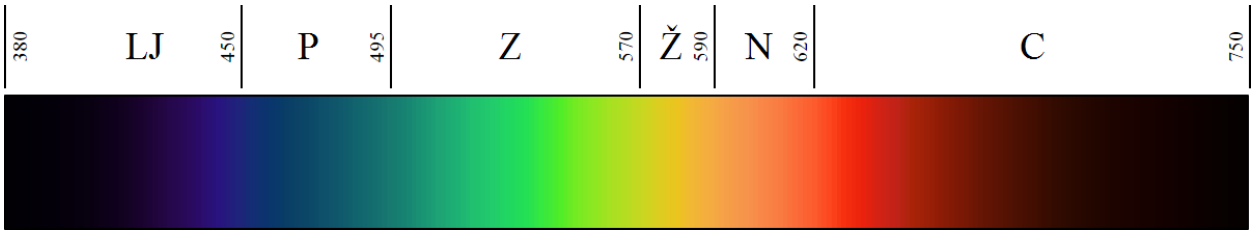
Svjetlost je elektromagnetsko zračenje koje je vidljivo ljudskom oku. Elektromagnetsko zračenje možemo zamisliti kao roj čestica koje zovemo fotoni. Svaki foton nosi određenu količinu energije. Elektromagnetska zračenja se razlikuju po frekvenciji. Svjetlost s više energije ima manju frekvenciju i veću valnu duljinu dok svjetlost s manje energije ima veću frekvenciju i manju valnu duljinu.

Ljudsko oko može percipirati elektromagnetsko zračenje u rasponu od 380 do 750 nanometara valne duljine. U tome malome rasponu valne duljine smještene su sve boje koje neki objekt može poprimiti. Najkraću valnu duljinu ljudsko oko percipira kao ljubičastu boju, dok najdulju valnu duljinu kao crvenu boju.

2.1. Boja

Boja je osjetilni doživljaj kada svjetlost određene valne duljine pobudi receptore u mrežnici oka. Mrežnica ili retina je funkcionalno najvažniji dio oka. Uloga mrežnice je primiti svjetlosne podražaje koje pretvara u akcijske potencijale vođene vidnim živcem prema mozgu. Za ispunjavanje toga zadatka na mrežnici se nalazi nekoliko slojeva živčanih stanica međusobno povezanih sinapsama. Jedine živčane stanice koje su fotosenzitivne su fotoreceptorske stanice. Fotoreceptorske stanice čine većinom dva tipa stanica, čunjići i štapići. [**1**] Ljudi imaju jedan tip štapića. Oni su osjetljivi na svjetlost te omogućuju vid u slabim svjetlosnim uvjetima. Čunjića ima tri vrste, za crvenu, plavu i zelenu boju. U normalnim svjetlosnim uvjetima čunjići nam omogućuju vidjeti boje, dok u slabim svjetlosim uvjetima ljudi ne raspoznaju boje. Treći daleko rjeđi tip fotosenzitivnih stanica čine fotosenzitivne ganglijske stanice koje su važne za refleksne reakcije na jarko danje svjetlo.

U vidljivome spektru zračenja nalaze se ljubičasta, plava, zelena, žuta, narančasta, crvena boja te njihove nijanse, što možemo vidjeti na slici 2.1. Boja objekta kojega mi vidimo nastaje apsorbcijom odnosno refleksijom određenog svjetlosnog spektra bijele svjetlosti. Bijela svjetlost je sastavljena od svih boja vidljivoga spektra. Zelene biljke iz bijele svjetlosti upijaju crvenu i plavu svjetlost, a reflektiraju zelenu. Dakle boja ovisi o ferkvenciji reflektirajućega zračenja.



Sl. 2.1 Vidljivi spektar boja. (Slika preuzeta iz [**2**])

Tradicionalno, boje u umjetnosti su podjeljene na osnovne i složene. Osnovne boje se ne mogu dobiti miješanjem drugih boja i u njih spadaju tri boje: crvena, plava i žuta. [**3**] Složene boje se dobivaju miješanjem osnovnih boja. Druga podjela boja je na tople, hladne i neutralne, zato što se u prirodi mogu zamijetiti uz određena toplinska stanja.

Kako bi dobili nama željenu boju moramo izabrati jednu ili pomiješati dvije ili više različitih boja. Postoji dva načina miješanja boja. To je aditivno miješanje koje se temelji na zbrajanju svjetloća pojedinih boja i suptraktivno miješanje pod koje podrazumjeva stvaranje percepcije boje oduzimanjem jednoga dijela spektra. Modeli boja umjetnika i znanstvenika su potpuno različiti. Za razliku od umjetnosti, u znanosti postoje drugačiji pristupi određivanja osnovnih boja zbog praktičnih i tehnoloških razloga. Aditivne osnovne boje su crvena, plava i zelena, dok su suptraktivne osnovne boje cijan, žuta i purpurna. Zbrajanjem svih aditivnih osnovnih boja dobijemo bijelu boju, dok zbrajanjem suptraktivnih osnovnih boja dobijemo crnu boju.

Pošto mrežnica ima osjetila za crvenu, zelenu i plavu boju, u televizorima i računalnim monitorima koristi se aditivni RGB prostor. RGB je engleska kratica za *red* (crevno), *green* (zeleno) i *blue* (plavo). Ovo je aditivni model boja gdje se boja opisuje kroz tri vrijednosti.

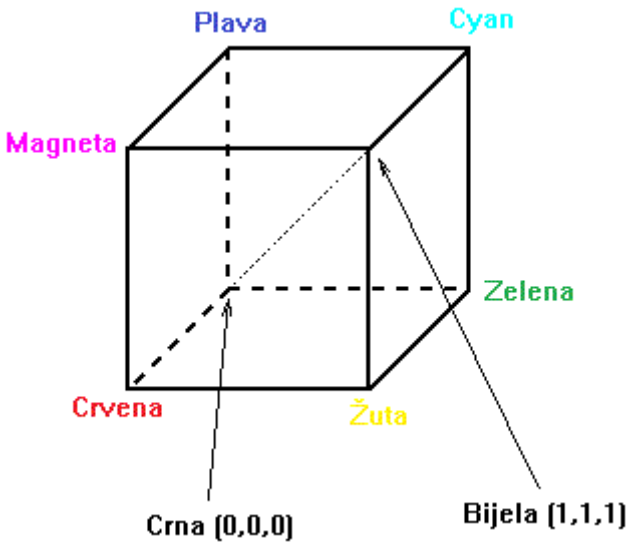
Sl. 2.2 Slika kuće i planina u RGB prostoru boja. (Slika preuzeta iz [**4**])

2.1.1. RGB prostor boja

Prostor boja je definiran u Cartezijevom koordinatnome sustavu. Područje u kojemu su definirane boje može se predočiti jediničnom kockom. U daljenjm tekstu će se spominjati kanali slike koji predstavljaju sliku načinjenu samo od jedne primarne boje. Na slici 2.2. možemo vidjeti sliku u RGB prostoru boja, a na slici 2.3. istu tu sliku rastavljenu na tri slike načinjene samo od crvene, zelene ili plave boje. Kombiniranjem te tri slike dobije se slika 2.2.

Sl. 2.3 RGB slika rastavljena na kanale. (Slika preuzeta iz [**4**])

Na slici 2.4. možemo vidjeti jediničnu kocku koja opisuje prostor boja RGB modela. Aditivne osnovne boje nalaze na osima Cartezijevog koordinatnoga sustava, dok se cijan, magenta i žuta nalaze u kutovima. U ishodištu koordinatnoga sustava je crna boja dok je na suprotnoj strani bijela boja.

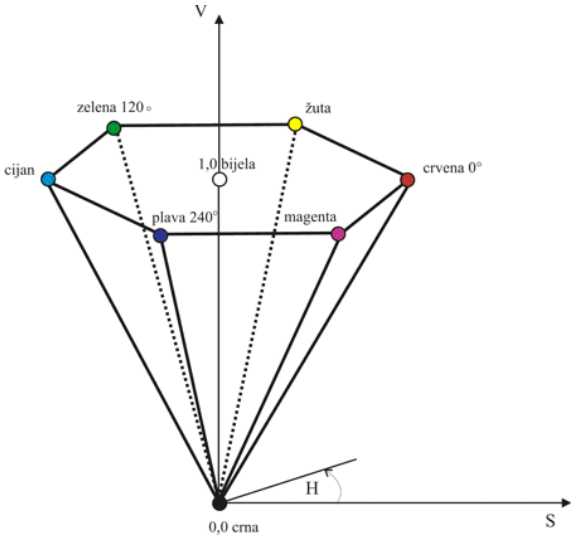


Sl. 2.4 Područje u kojemu su definirane boje kod RGB modela. (Slika preuzeta sa [**5**])

RGB prostor boja svoju primjenu pronalazi kod uređaja koji emitiraju svjetlost, kao što su monitori, televizori, zasloni mobitela. Zbog svoje jednostavnosti pronašao je široku upotrebu, no problem ovoga modela je odrediti koliko svake osnovne boje sadrži određena boja. Veza između količine plave, crvene i zelene boje te boje koja se dobije kao krajnji rezultat nije intuitivan.

2.1.2. HSV prostor boja

Kako bi stvorio intuitivniji model i bliži tradicionalnome modelu boja, pionir računalne grafike Alvy Ray Smith 1970-tih opisao je HSV model boja. Ovaj prostor boja opisuje boje najsličnije ljudskoj percepciji. HSV je engleska kratica za hue (nijansa), saturation (zasićenost) i value (svjetlina). Na slici 2.5. možemo vidjeti da je HSV model definiran u cilindričnome koordinatnome sustavu. Boje su prikazane u podskupu prostora omeđenom šesterostranom piramidom. Vrh piramide je u ishodištu i odgovara crnoj boji [**6**].Vertikalna os određuje svjetlinu boje (*value*), kut zakretanja oko vertikalne osi određuje nijansu boje (*hue*), a radijalna udaljenost od osi *value* određuje razinu zasićenosti boje (*saturation*).



Sl. 2.5 HSV prostor boja. (Slika preuzeta sa [**5**])

Prednost HSV prostora nad RGB prostorom boje je to što svjetlost i sjena ne mijenjaju puno informaciju o nijansi boje predmeta, već mijenjaju informacije o intenzitetu svjetline i zasićenju. Ta prednost čini HSV prostor boja dobrim odabirom za detekciju i praćenje objekta bazirano na njegovoj boji.

Transformacija RGB prostora u HSV prostor je sljedeća. Pretpostavimo da svaki slikovni element (*eng.pixel*) u RGB prostoru ima vrijednost kanala (R, G, B) od 0 do 255. Svaku vrijednost slikovnog elementa podjelimo sa 255 kako bismo dobili vrijednosti (R', G', B') koje se nalaze unutar intervala [0, 1]. Nakon toga odredimo koji od R', G' i B' kanala ima maksimalnu, a koji minimalnu vrijednost, te odredimo njihovu razliku.

$$\mathrm{M} = \max(\mathrm{R}', \mathrm{G}, \mathrm{B}') \tag{2-1}$$

$$\mathrm{m} = \min(\mathrm{R}', \mathrm{G}', \mathrm{B}') \tag{2-2}$$

$$\Delta = \mathrm{M} - \mathrm{m} \tag{2-3}$$

Kad smo dobili $M,\ m\ i\ \Delta$ možemo izračunati H, S, V vrijednosti kanala za taj slikovni element.

$$\mathrm{H} = \begin{cases} 60° \times \dfrac{\mathrm{G}' - \mathrm{B}'}{\Delta} \bmod 6, & \text{ako je } \mathrm{M} = \mathrm{R} \\ 60° \times \dfrac{\mathrm{B}' - \mathrm{R}'}{\Delta} + 2, & \text{ako je } \mathrm{M} = \mathrm{G} \\ 60° \times \dfrac{\mathrm{R}' - \mathrm{G}'}{\Delta} + 4, & \text{ako je } \mathrm{M} = \mathrm{B} \end{cases} \tag{2-4}$$

$$\mathrm{S} = \begin{cases} 0, & \text{ako je } \mathrm{M} = 0 \\ \dfrac{\Delta}{\mathrm{M}}, & \text{ako je } \mathrm{M} \neq 0 \end{cases} \tag{2-5}$$

$$\mathrm{V} = \mathrm{M} \tag{2-6}$$

3. DETEKCIJA OBJEKTA NA SLICI

Detekcija i segmentacija objekta je jedna od najvažnijih i najizazovnijih zadataka u robotskome vidu. Pronalazi primjenu u mnogim aplikacijama kao što su pretraživanje slika, razumjevanje okoline, praćenje objekta itd. Ovovremenski algoritmi za detekciju objekata su brzi i robusni. Zbog svoje brzine mogu biti implementirani u sustave stvarnoga vremena, dok zbog svoje robusnosti mogu zadovoljavajuće detektirati objekt na kompleksnoj i raznolikoj pozadini. Međutim još uvijek ne postoji algoritam koji savršeno detektira objekte. Zbog sličnosti objekta i pozadine ili zbog osvjetljenja objekta često dolazi do pogrešaka.

Najjednostavnija detekcija objekta je ona bazirana na njegovoj boji. Zbog svoje jednostavnosti implementacije i relativno dobrih rezultata korištena je u izradi ovog diplomskoga rada. Kako bi ova metoda detekcije bila učinkovita mora postojati razlika boje objekta i boje pozadine. U sljedećem poglavlju je opisano kako detektirati objekt na temelju boje.

3.1. Detekcija objekta na temelju boje

Opisivanjem predmeta koji se nalaze na stolu bez spominjanja njihove boje uskratili bi slušatelju važne informacije. Boju kao vrlo bitnu informaciju koju možemo vidjeti na nekome objektu iskoristit ćemo kako bi detektirali objekt na slici. Pretpostavimo da se objekt nalazi na slici gdje je pozadina bitno različite boje od boje objekta. Tom pretpostavkom otklanjamo poteškoće koje bi se mogle stvoriti kad bi objekt stavili na pozadinu koja je iste boje kao i on.

Slika koju kamera sprema na računalo je u RGB prostoru boja. Svaki kanal crvene, zelene i plave boje možemo promatrati odvojeno kao jednu matricu s informacijama o tome kanalu. Pri promjeni osvjetljenja objekta dolazi do velikih oscilacija vrijednosti kanala u RGB prostoru boja pa se zbog toga za detekciju objekta koristi HSV prostor boja. Osvjetljenje ne mijenja znatno informaciju o boji koja se nalaze u *hue* kanalu, osim ako nije ekstremno osvjetljenje i ekstremno zatamnjenje. Ta dva ekstrema mijenjaju boju objekta u bijelu odnosno crnu. Sljedeći postupak određivanja objekta može se provoditi na više kanala slike, ali u ovome slučaju provodi se samo na *hue* kanalu. Jedan piksel hue kanala za 8-bitnu sliku ima vrijednost od 0 do 180. [**7**] Razlog tome je maksimalna vrijednosti varijable za 8-bitne brojeve koja je 255. Puni kut zakretanja oko vertikalne osi (360°) ne može biti zapisan u 8-bitnu varijablu. Zbog toga se kut zakretanja oko vertikalne osi dijeli sa dva te se taj broj upisuje kao vrijednost elementa *hue* matrice.

Unutar toga raspona za hue kanal nalaze se sve boje koje objekt može imati. Pomoću „metode praga“ obrade slike, hue kanal pretvorimo u binarnu sliku koja prestavlja masku objekta.

Metoda praga je najosnovnija metoda za segmentaciju slike. Piksel $\mathrm{m}[\mathrm{x},\mathrm{y}]$ zamjenjuje se sa crnim pikselom ako se vrijednost piksela ne nalazi unutar intervala $[\mathrm{T}_{\min},\mathrm{T}_{\max}]$ ili se zamjenjuju s bijelim pikselom ako se vrijednost piksela nalazi unutar intervala $[\mathrm{T}_{\min},\mathrm{T}_{\max}]$. Rezultat tresholdinga je binarna slika. Binarna slika ima samo jedan kanal. Pikseli u binarnoj slici mogu poprimiti vrijednosti jedan ili nula, bijelo ili crno te je iste veličine kao i orginalna slika. Takva birnarna slika zove se još i maska objekta.

$$\mathrm{m}[\mathrm{x},\mathrm{y}] = \begin{cases} 1, & \text{ako je } \mathrm{m}[\mathrm{x},\mathrm{y}] \geq \mathrm{T}_{\min} \text{ i } \mathrm{m}[\mathrm{x},\mathrm{y}] \leq \mathrm{T}_{\max} \\ 0, & \text{za sve ostalo} \end{cases} \tag{3-1}$$

Na dobivenoj maski objekta odredimo najveću zatvorenu konturu koja opisuje objekt. Kontura je niz piksela $m_1, m_2, m_3, \dots, m_{\mathrm{n}}$ takav da svaki element pripada N^8susjedstvu prethodnoga elementa, ako je pri tome $m_{\mathrm{n}} = m_1$, radi se o zatvorenoj konturi [**8**]. Koristeći Greenovu formulu nad zatvorenom površinom uniformne gustoće možemo odrediti momente objekta na slici. Pomoću dobivenih momenata može se odrediti točka centroida objekta na slici koju omeđuje pronađena najveća kontura. Formula za računanje momenta glasi:

$$\mathrm{m}_{p,q} = \iint \mathrm{x}^p \mathrm{y}^q \mathrm{f}(\mathrm{x},\mathrm{y}) \mathrm{dxdy} \tag{3-3}$$

gdje je :

$$\mathrm{f}(\mathrm{x},\mathrm{y}) = \begin{cases} 1, & \text{točka pripada objektu} \\ 0, & \text{točka pripada pozadini} \end{cases} \tag{3-2}$$

Zbroj p i q je indikator stupnja momenta. Za računanje centroida potrebni su moment nultoga i prvoga stupnja:

Moment nultoga stupnja, odnosno površina omeđena konturom $((p,q) = (0,0))$:

$$\mathrm{m}_{0,0} = \iint \mathrm{f}(\mathrm{x},\mathrm{y}) \mathrm{dxdy} \tag{3-4}$$

Momenti prvoga stupnja ($(p,q) = (1,0)$) i ($(p,q) = (0,1)$)

$$m_{1,0} = \iint xf(x,y)dxdy \tag{3-5}$$

$$m_{0,1} = \iint yf(x,y)dxdy \tag{3-6}$$

Računanjem omjera momenta prvog i momenta nultoga stupnja dobijemo informaciju o x i y koordinatama centroida objekta na slici [**9**].

$$x_c = \frac{m_{1,0}}{m_{0,0}} \tag{3-7}$$

$$y_c = \frac{m_{0,1}}{m_{0,0}} \tag{3-8}$$

Obradom niza slika i određivanjem centroida objekta na svakoj od njih dobivamo informacije koje nam omogućuju praćenje objekta.

4. SKLOPOVSKA REALIZACIJA RJEŠENJA PROBLEMA

Za realizaciju sklopovskoga rješenja korišteno je jeftino i svima pristupačno sklopovlje naručeno preko interneta. Jedan od ciljeva ovoga diplomskoga rada je bio pokazati da je moguće napraviti jeftina programirljiva robotska kolica malih dimenzija. U ovome poglavlju su opisani dijelovi koji su korišteni za izradu sklopovlja te shema sklopovlja.

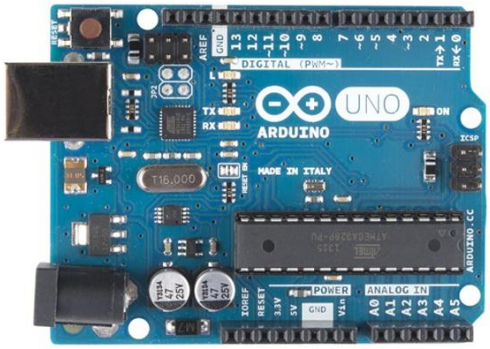
4.1. Sklopovlje

Sklopovlje je podijeljeno na dvije komponente, tj. na master i slave uređaj. Master uređaj je pametni telefon za koji se pretpostavlja da je već tvornički cijeli sastavljen, dok je slave uređaj Arduino uno platforma. Slave uređaj prima podatke s pametnoga telefona, parsira ih i upravlja istosmjernim (DC) motorima. Ovo poglavlje će se bazirati na dijelovima slave uređaja.

Slanje i primanje podataka s pametnoga telefona se odvija preko bluetooth komunikacije koju omogućuje HC-05 bluetooth (serial port protokol) modul. HC-05 bluetooth modul je jeftin, može raditi kao master ili slave uređaj te zadovoljavajuće obavlja zadatak primanja podataka s pametnoga telefona.

Sl. 4.1 HC-05 bluetooth modul. (Slika preuzeta sa [**10**])

Nakon što HC-05 primi podatke , šalje ih ATmega328 mikrokontroleru koji se nalazi na Arduino uno platformi. Arduino uno je open-source lako programljiva platforma, malih dimenzija koja se na internetu može naći za malu cijenu. Programira se u C++ programskim jezikom.



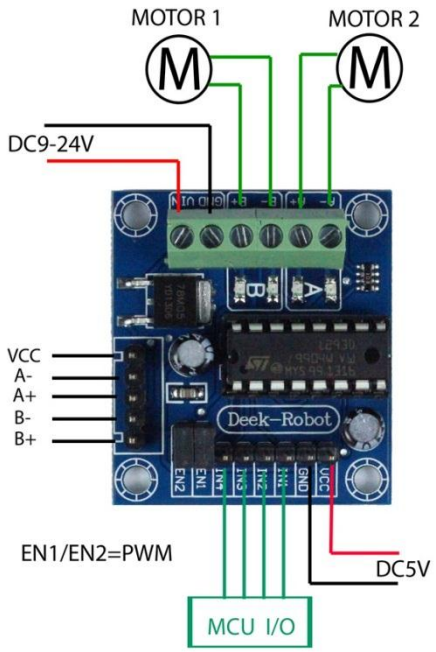
Sl. 4.2 Arduino uno platforma. (Slika preuzeta sa [**11**])

Kako bi ispitivanje i upravljanje robotskim kolicima bilo što jednostavnije koristi se poseban oblik poruke koju šalje pametni telefon. Prvi znak definira kretanje robotskih kolica, drugi znak je broj između 0 i 255 koji definira brzinu okretanja motora i treći znak je broj koji definira vrijeme trajanja okretanja motora. Treći znak ne mora nužno biti poslan, ali će se u tome slučaju motor vrtiti sve dok ne primi naredbu stop. Znakovi u poruci su odvojeni dvotočkom. Primjer poruke je „R:255:1000:“ , robotska kolica se rotiraju u desno, punom snagom 1000 milisekundi ili drugi tip poruke „R:255:“ , robotska kolica se rotiraju u desno punom snagom sve dok ne prime naredbu „S:“ .

Tab. 4.1 Oblik poruke koju prima Arduino

R:	(0 – 255):	(0 - ∞):	Rotiranje u desno
L:	(0 – 255):	(0 - ∞):	Rotiranje u lijevo
F:	(0 – 255):	(0 - ∞):	Kretanje naprijed
B:	(0 – 255):	(0 - ∞):	Kretanje unazad
S:			Naredba stop

Arduino uno nakon što parsira primljenu poruku šalje signal na H-most (eng. *H-bridge*). H-most je sučelje između Arduina i istosmjernih (DC) motora. Ima dva ulaza za napajanje, jedno zajedničko uzemljenje s Arduinom, četiri izlaza na koje se spajaju istosmjerni motori i šest ulaza kojima se oni upravljaju. Jedno se napajanje koristi za napajanje istosmjernih motora, može biti između 9*V* i 25*V*, ovisno o zahtjevim istosmjernih motora. Drugo se napajanje koristi za napajanje L293D čipa i ono je 5*V*. H-most podržava ulaznu jačinu struje do1.2*A* po motoru. Ulazni pinovi na H-mostu za svaki istosmjerni motor su: enable1-2 pinovi i input1-4 pinovi. Enable1-2 pinovi određuje brzinu okretanja motora i spojen je na izlazni *PWM* (eng. *Pulse With Modulation*) pin Arduina, dok input1-4 pinovi određuju u kojemu se smjeru vrti istosmjerni motor.



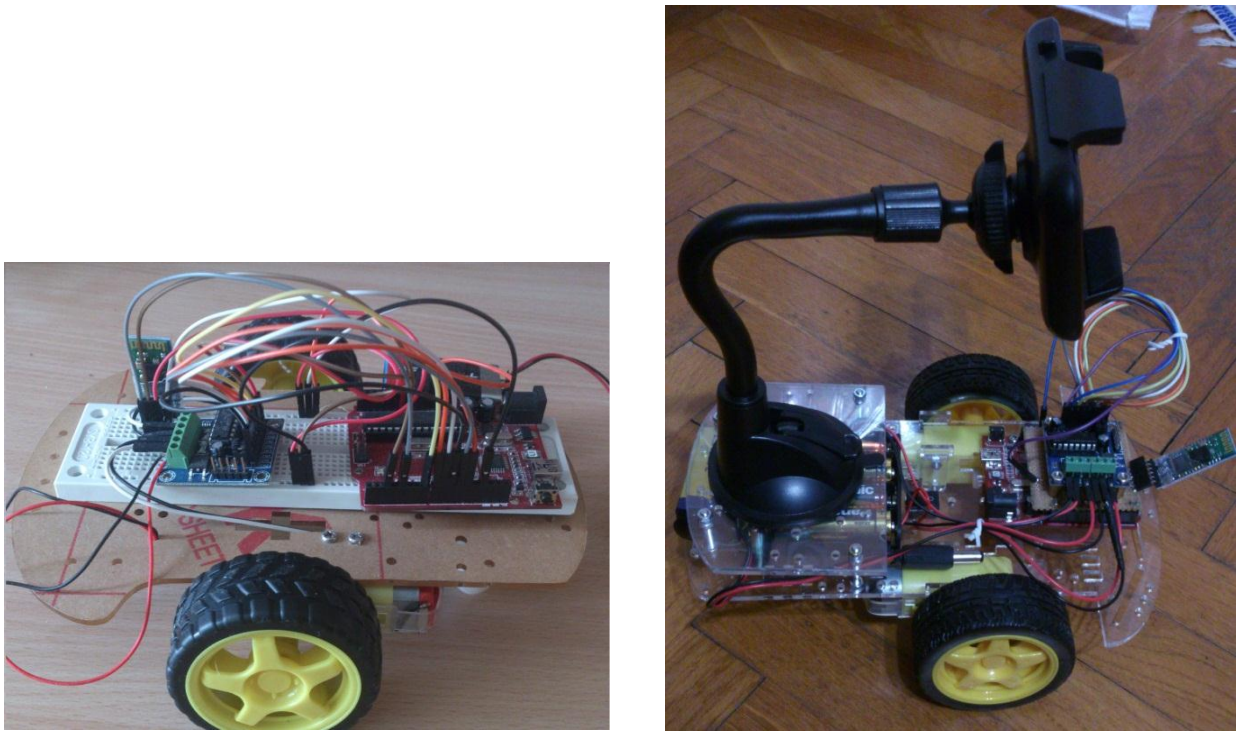
Sl. 4.3 L293D H-most modul. (Slika preuzeta sa [**12**])

U tablici 4.2. možemo vidjeti u kojemu će se smjeru vrtiti istosmjernih motor ako input1 i input2 pin postavimo u određeno logičko stanje.

Tab. 4.2 Smjer vrtnje istosmjernih (DC) motora

Input1	**Input2**	**Smijer:**
L	L	Miruje
L	H	U smjeru kazaljke na satu
H	L	U smjeru suprotno od kazaljke na satu
H	H	Miruje

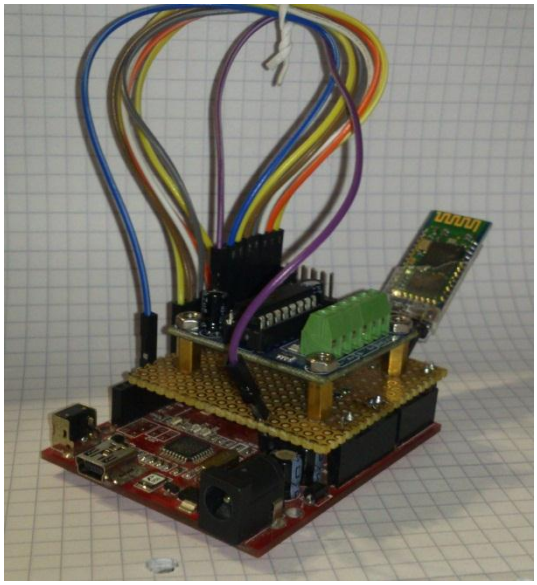
Prvi prototip sklopovskoga rješenja može se vidjeti na slici 4.4. pod a). Kako bi se oslobodilo mjesta na vozilu za držač mobitela napravljena je pločica koja objedinjuje Arduino, H-most i bluetooth modul. Novi izgled prototipa može se vidjeti na slici 4.4. pod b).



a) b)

Sl. 4.4 Na slici a) je prvi prototip, a na slici b) konačni prototip sklopovskoga rješenja

Izgled pločice i Arduina izbliza može se vidjeti na slici 4.5.

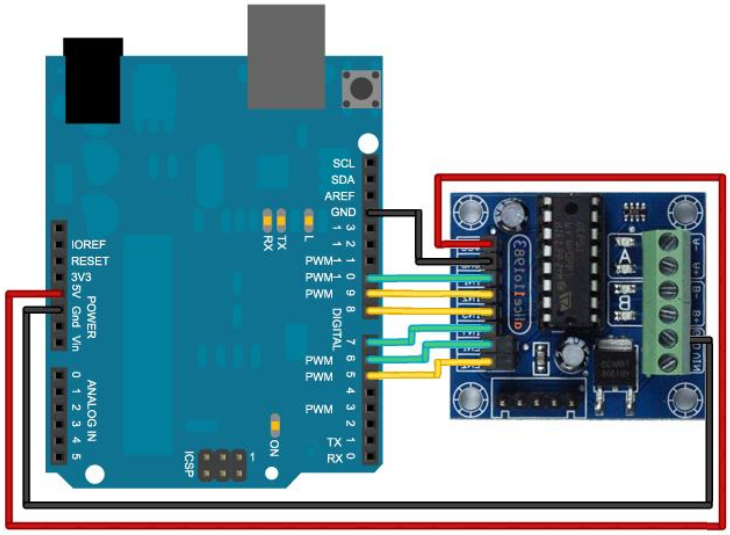


Sl. 4.5 Izgled pločice i Arduina

4.2. Shema rješenja

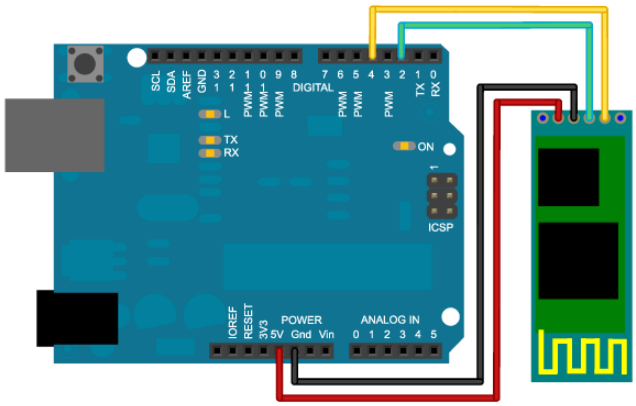
Kao što je već navedeno, sklopovlje se sastoji od dva modula koji se spajaju na Arduino. Ti moduli su: L293D H-most koji se nalazi na industrijski predefiniranoj pločici, te HC-05 bluetooth modul.

L293D H-most ima: četiri izlazna pina OUTPUT1-4 na koje se spajaju istosmjerni (DC) motori, te Vs i Gnd pinove na koje se spaja napajanje istosmjernih motora. Istosmjerni motori, L293D čip i Arduino moraju imati zajedničko uzemljenje. Istosmjerni (DC) motori se upravljaju pinovima INPUT1-2 za jedan i INPUT3-4 za drugi motor, brzina se upravlja preko ulaznih pinova ENABLE1-2. Shema spajanja H-mosta i Arduina je prikazana na slici 4.6.



Sl. 4.6 Shema spajanja H-mosta i Arduina.

HC-05 bluetooth modul ima Tx i Rx pinove, Vcc i Gnd te još dva pina koja nisu korištena u ovome projektu. Tx pin je spojen na Arduino pin 4. koji je na Arduinu inicijaliziran kao Rx pin, dok je Rx pin spojen na Arduino pin 2, koji je inicijaliziran kao Tx pin. Inicijalizacija Rx i Tx pinova na Arduinu omogućila je SoftwareSerial.h biblioteka. O tome zašto se ona koristi napisano je u poglavlju o programskome rješenju. Na slici 4.7. se nalazi spajanje HC-05 bluetooth modula i Arduina.



Sl. 4.7 Shema spajanja Arduina i HC-05 bluetooth modula

5. REALIZACIJA PROGRAMSKOGA RJEŠENJA PROBLEMA

U ovome poglavlju je opisana logika programskoga rješenja. Programsko rješenje je napisano za Arduino razvojnu pločicu i mobitel s Android operacijskim sustavom. Arduino obrađuje poruke koje prima bluetooth vezom i upravlja istosmjernim motorima. Mobitel obrađuje slike s kamere i nakon što ih obradi, šalje poruke bluetooth vezom na Arduino.

5.1. Programsko rješenje za Arduinu

Srce Arduina Uno je ATmega328 mikrokontroler. On izvršava zadatke ovisno o kodu koji se nalazi u njegovoj memoriji. Sučelje za programiranje može biti službeni Arduino IDE ili bilo koje drugo sučelje za C/C++ programske jezike koje podržava programiranje mikrokontrolera. Arduino izvorni kod čine dvije glavne funkcije, to su: *setup()* i *loop()* funkcije. *Setup* funkcija služi za inicializaciju varijabli i izvodi se na početku, samo jednom, dok se *loop* funkcija započinje izvoditi nakon *setup* funkcije i stalno se ponavlja. Služi za obradu podataka s ulaza i slanje podataka na izlaze.

U setup petlji programskoga rješenja otvara se bluetooth veza s pametnim telefonom na 9600 bauda. Za uspostavljanje veze koristi se *SoftwareSerial* objekt čija se klasa nalazi u *SoftwareSerial.h* biblioteci. *SoftwareSerial* objekt omogućava stvaranje serijske veze (engl. se*rialconnection*) na pinovima 2 i 4. Također *SoftwareSerial* biblioteka omogućuje stvaranje više pinova koji bi služili kao serijska komunikacija te brzinu komunikacije do 115200 bps.

```
#include <SoftwareSerial.h>
SoftwareSerial myBluetooth(2, 4);

void setup()
{
 myBluetooth.begin(9600);
}
```

Nakon setup petlje i uspostave bluetooth veze izvodi se loop petlja. U njoj se nalazi cijela logika primanja poruke, njezino parsiranje i upravljanje robotskim kolicima. Na početku se provjerava je li novootvorena veza slobodna. Ukoliko je, čita se znak po znak koji dolazi bluetooth vezom i sprema se u niz *c*. Svaki novospremljeni znak se uspoređuje s dvotočkom ':'. Dvotočka označava kraj naredbe u poruci koja sadrži tri naredbe. Ukoliko je znak jednak dvotočki provjerava se koja je to dvotočka po redu te se ovisno o tome izvršava naredba. Naredba prije prve dvotočke definira smjer kretanja robotskih kolica. Druga naredba pokreće robotska kolica u definiranome

smjeru brzinom koja je zapisana između prve i druge dvotočke. Treća naredba, čija se vrijednost nalazi između druge i treće dvotočke, definira koliko će se dugo robotska kolica kretati u zadanome smjeru. Ako vrijeme nije definirano, kolica će se kretati sve dok Arduino ne primi naredbu stop 'S:'. Kod *loop* funckije se nalazu u prilogu P.5.1., a njezin pseudokod je sljedeći:

```
Spremaj ulazne podatke u niz sve dok je bluetooth veza slobodna
  Ako je zadnji znak ':'
    Obriši znak ':' i zamjeni ga sa znakom za kraj stringa '\0'

    Ako je niz znakova prvi dio poruke
      Spremi niz znakova u varijablu za smjer.

      Ako je smjer jednak znaku 'S'
        Zaustavi auto.

    Ako je niz znakova drugi dio poruke
      Spremi niz znakova u varijablu za brzinu
      Pokreni auto u zadanome smjeru zadane brzine

    Ako je niz znakova treći dio poruke
      Spremi niz znakova u varijablu za kontrolu

      Ako je znak za kontrolu jednak znaku 'S'
        Zaustavi auto
      U suprotnome
        Zaustavi auto nakon vremena koje je zapisano
        u kontrolnoj varijabli
```

Arduino bluetooth vezom prima znakove tipa *char*, što bi značilo da broj neće biti primljen kao *int* već kao niz *charova.* Zbog toga je napravljena pomoćna funkcija *stringToInt()* koja za ulazni parametar prima niz znakova *a* i njihovu dužine *length* te vraća *int* varijablu koja je zapisana u nizu znakova *a*.

```
int stringToInt(char *a, int length)
{
   int number = 0;
   for (int i = 0; a[i] != '\0'; i++)
   {
    number += (a[i] - 48)*pow(10.0, (length - 1) - i);
   }

   return (number);
}
```

Upravljanje robotskim kolicima je maksimalno pojednostavljeno pozivanjem funkcija *forward(), backward(), rotateLeft(), turnleft(), rotateRight(), turnRight() i stop()* koje pruža objekt iz bibliteke*CarController.h.CarControler* biblioteka je sučelje između programskoga

koda i motora kojima upravlja. Konstuktor *CarController* objekta za ulaz prima 6 varijabli koje definiraju pinove na H-mostu za dva istosmjerna motora. Svaki motor ima tri pina, odnosno tri varijable: enable, input1 i input2. Sljedeće linije koda prikazuju uključivanje biblioteke u kod i instanciranje objekta.

```
#include <CarController.h>
CarController myCar(7,8,9, 11,12,10);
```

Sve funkcije za pomicanje kolica, u koje se ne uvrštava funkcija *stop()* za zaustavljanje, imaju jednu ulaznu varijablu koja određuje brzinu vrtnje motora. Ona može biti pozitivan broj od 0 do 255 te definira veličinu PWM signala.

```
myCar.forward(255);
myCar.backward(255);
myCar.rotateLeft(255);
...
myCar.turnRight(255);
myCar.stop();
```

Sve funkcije pomicanja su objedinjene u dvije funckije, *moveOn()* i *moveOnWithDelay()*. Te dvije funkcije pojednostavljaju pisanje koda u Arduinu i doprinose njegovoj preglednosti. Ulazni parametri ovih funkcija su smjer vrtnje motora i njihova brzina za *moveOn()* funkciju te dodatni parametar za funkciju *moveOnWithDelay()* koji definira koliko će se dugo vrtiti motori. Smjer vrtnje motora je niz znakova ili samo jedan znak koji predstavlja prvo veliko slovo u engleskoj riječi za smjer pomicanja. Ulazne varijable koje definiraju smjer vrtnje istosmjernog motora mogu se vidjeti u tablici 5.1.

Tab. 5.1 Ulazne varijable koje definiraju smjer vrtnje motora

Hrvatki / Engleski naziv za smjera pomicanja	**Ulazna varijabla**
Naprijed / Forward	F
Nazad / Backward	B
Rotiraj desno / Rotate Right	RR
Rotiraj lijevo / Rotate Left	RL
Okreni desno / Turn Right	TR
Okreni lijevo / Turn Left	TL

Primjer korištenja te dvije funkcije: prva funkcija pokreće kolica prema naprijed na neodređeno vrijeme, dok druga funkcija pokreće kolica prema naprijed samo 1 sekundu.

```
myCar.moveOn(„F“);
myCar.moveOnWithDelay(„F“, 1000);
```

Budući da se motori na robotskim kolicima nalaze jedan nasuprot drugoga, za kretanje naprijed ili nazad motori se moraju okretati zrcalno jedan nasuprot drugome. Rotiranje robotskih kolica u lijevo ili u desno zahtjeva okretanje istosmjernih motora u istome smjeru. Kako bi upravljanje istosmjernim motorima bilo jednostavnije napravljena je klasa *DCMotorControler*. Objekt *DCMotorControler* klase upravlja svakim istosmjernim motorom pojedinačno. Konstruktor *DCMotorControler* objekta prima tri varijable, enable, input1 i input2. Enable varijabla definira brzinu okretanja motora te zahtjeva da signal prima s PWM Arduino pina. Smjer vrtnje istosmjernih motora ovisi o smjeru struje kroz istosmjerni motor koji je određen varijablama input1 i input2. Prema tablici 4.2. možemo vidjeti u kojem će se smjeru vrtiti istosmjerni motor ako postavimo input1 i input2 varijable u određeno logičko stanje.

5.2. Programsko rješenje pokretano na Androidu

Programsko rješenje računalnog vida je napravljeno u obliku Android aplikacije. Android aplikacija je podjeljena na dva *Activity*-a: *MainActivity* i *CameraActivity*. Izgled *Activity*-a definira *XML* kod, dok njegovu funkcionalnost definira *java* kod.

5.2.1. Main Activity

MainActivity prikazuje listu detektiranih bluetooth uređaja koji se nalaze u neposrednoj blizini mobitela te rukuje njima i po potrebi stvara bluetooth veze s Arduinom. Izgled *MainActivity*-a je definiran XML kodom danim u prilogu P.5.2.

U *Main Activity*-u se nalazi *ListView* koji služi za izlistanje bluetooth uređaja te *Button* koji inicira stvaranje bluetooth veze. Dodavanje i brisanje stavki s *ListView*-a izvršava *BluetoothCustomAdapter* klasa. To je klasa koja nasljeđuje *ArrayAdapter* klasu i sadrži listu objekata tipa *BluetoothDevice.*

```
public class BluetoothCustomAdapter extends ArrayAdapter<BluetoothDevice>
```

BluetoothDevice klasa sadrži sve potrebne informacije o jednom bluetooth uređaju, kao što su: ime uređaja, njegova MAC adresa, trenutni status, tip uređaja itd.

Funkcija zadužena za izmjene *ListView*-a je funkcija *getView*. Ona se poziva za svaki objekt iz liste i vraća novi *View* koji se prikazuje na *ListView*-u. Izgled svakoga od *View*-a u *ListView*-u je također definiran XML kodom koji je dan u prilogu P.5.3.

getView() je funkcija klase *ArrayAdapter*, ali pošto od njenog prvobitnog koda nemamo nikakve koristi, potrebno ju je prepisati (eng. *Override*) i izmjeniti njenu funkcionalnost. Prepisana funkcija napuhuje (eng. *inflate*) novi View, inicijalizira i postavlja *View*-e unutar njega te postavlja slušač klikova (eng. *click listener*) na *RadioButton*. Dodirom *View*-a sprema se redni broj *RadioButton*-a koji je potreban za dohvaćanje *BluetoothDevice* objekta iz liste i uspostavu veze. Kod funkcije *getView* je dan u prilogu P.5.4.

BroadcastReciever je Android komponenta koja dopušta da se pretplatite na sistemske ili aplikacijine događaje. Svaki put kad se taj događaj dogodi poziva se pretplaćeni *BroadcastReciever* objekt i poziva se njegova funkcija *onRecieve().* Za otkrivanje bluetooth uređaj u blizini mobitela potrebno je pretplatiti *BroadcastReciever* objekt na *BluetoothDevice.ACTION_FOUND* događaj pomoću funkcije *registerReciever*.

```
IntentFilter event = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReciever, event);
```

Svaki put kad mobitel otkrije bluetooth uređaj, *onRecieve()* funkcija ažurira *ListView* s novim *Viewima* pomoću funkcije *notifyDataSetChange* objekta *BluetoothCustomAdapter*. Kod inicijalizacije *BroadcastReciever* objekta je dan u prilogu P.5.5. Uspostava veze s označenim bluetooth uređajem na listi inicira se pritiskanjem gumba na kojemu piše „Connect to“. Kako bi to bilo moguće na gumb je postavljen *onClickListener* koji prilikom dodira poziva *onClick()* funkciju. U prepisanoj (eng. *Override*) *onClick()* funkciji se provjerava je li označen neki od uređaja s liste. Ako je, stvara se instanca *BluetoothConnection* objekta koji je zadužen za upravljanje vezom između mobitela i odabranog bluetooth uređaja, te se otvara *CameraActivity*. Kod funkcije *onClick* dan je u prilogu P.5.6.

Klasa *BluetoothConnection* otvara vezu funkcijom *openSocket()*, zatvara vezu funkcijom *closeSocket()* te šalje podatke funkcijom *write()*. Funkcija primanja podataka s Arduina nije bila potrebna za ovaj projekt te nije implementirana. Ona se može postići tako da *BluetoothConnection* klasa nasljedi klasu *Thread* te da se prepiše funckija *run()* koja se poziva svaki put kad se nit (eng. *Thread*) pokreće s funkcijom *run()*. Važno je napomenuti da korištenje bluetooth uređaja na mobitelu ne bi bilo moguće da se u *Android Manifestu* ne dodijeli dozvola za njegovo korištenje.

```
<uses-permissionandroid:name="android.permission.BLUETOOTH"/>
<uses-permissionandroid:name="android.permission.BLUETOOTH_ADMIN"/>
```

openSocket() funkcija otvara nesigurnu (eng. *insecure*) vezu s bluetooth uređajem, koja za razliku od sigurne (eng. *secure*) ne zahtjeva pin. Sigurna veza nije bila potrebna za ovaj projekt te samim time nije implemenitrana. Postoji više načina za otvaranja nesigurne veze, jedan od njih je sljedeći. Treba napomenuti da postoji vjerojatnost da će se nesigurna veza u budućim inačicama Android sustava izbaciti iz uporabe i onemogućiti mogućnost njezinog otvaranja. Zatvaranje bluetooth veze se svodi na zatvaranje otvorenog soketa. Kod za funkcije *openSocket* i *closeSocket* je dan u prilogu P.5.7 i P.5.8. Za slanje *stringa* podataka, potrebno ga je prvo prebaciti u niz bajtova funkcijom *getBytes()* te se nakon toga *string* može uspješno poslati pomoću *OutputStream* objekta funkcijom *write()*. Sve funkcije su *boolean* povratnog tipa kako bi se lakše primjetila pogreška u vezi. Prilikom neuspješnog slanja poruke ili neuspješnoga otvaranja soketa funkcije će vratiti *false*. Kod za funkciju *write* je dan u prilogu P.5.9.

5.2.2. Camera Activity

CameraActivity prikazuje i obrađuje slike s kamere. Sve funkcije za obradu slike koje su korištene u *CameraActivity*-u su dio OpenCV4Android biblioteke. Detaljan opis dodavanja i postavljanja OpenCV4Android biblioteke je dan u prilogu P.5.10. Dodirom na objekt neke boje raspoznaje se njegova boja i započinje se s praćenjem toga objekta. Na slikama se određuje centroid odabranoga objekta i šalju se poruke Arduinu za upravljanje vozilom. Izgled *CameraActivity* definiran je XML kodom koji je dan u prilogu P.5.11. U XML kodu *CameraActivity*-a možemo vidjeti *View CameraClass* koji je modificirani *JavaCameraView*, dodana je funkcija za postavljanje rezolucije kamere. U Android Manifestu potrebno je dati dozvolu aplikaciji za korištenje kamere.

```
<uses-permissionandroid:name="android.permission.CAMERA"/>
```

Unutar java koda *CameraActivity*-a nalazi se gornja klasa koja implementira dva sučelja (eng. *interface*). To su *CvCameraViewListener2* i *OnTouchListener*.

```
public class CameraActivity extends Activity implements CvCameraViewListener2,
OnTouchListener {}
```

CvCameraViewListener2 određuje da klasa mora implementirati metode za kontrolu kamere. To su metode *onCameraViewStarted*, koja se pozove jednom kad se upali kamera, *onCameraViewStopped*, pozove se kad se kamera zaustavi te *onCameraFrame* koja se poziva svaki put kad kamera zabilježi okvir slike (eng. *frame*). *OnTouchListener* određuje da klasa mora implemetirati metodu *onTouch* koja se pozove svaki put kad se dodirne ekran mobitela.

Prilikom pokretanja *Activity*-a i svaki put kad se *Activity* vrati iz pauze potrebno je asinhrono učitati i inicijalizirati OpenCV biblioteku. To se izvodi u funkciji *onResume*, pozivanjem funkcije *initAsync* pomagačke (eng. *helper*) klase *OpenCVLoader*. Za sada *mCarController* objekt zanemarite.

```
@Override
public void onResume()
{
super.onResume();
   OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_3, this, mLoaderCallback);
   if(mCarController != null)
mCarController.openConnectionIfClosed();
}
```

Uz ulaznu varijablu koja određuje verziju OpenCV-a, predaje se i objekt *BaseLoaderCallback* klase koji, nakon što se završi inicijalizacija, poziva funkciju *onManagerConnected*. Nakon inicijalizacije spremno je korištenje OpenCV biblioteke, ukoliko je prije toga na mobitel instaliran OpenCV Manager, zato se unutar *onManagerConnection* funkcije inicijaliziraju varijable koje koriste OpenCV bibloteku. *mOpenCvCameraView* je objekt *CameraClass* klase koja je zadužena za upravljanje istoimenim *View*-om iz XML koda. Kod inicijalizacije *BaseLoaderCallback* objekta dan je u prilogu P.5.12.

Kad se aplikacija pokrene, poziva se i funkcija *onCreate* u kojoj se inicializiraju sve potrebne varijable koje ne koriste OpenCV bibloteku. Jedna od njih je već spomenuti objekt klase *CarController* koji je zadužen za uspostavu veze između mobitela i Arduina te upravljanje istosmjernim motorima. Kao i pomagačka klasa *OpenCVLoader*, *CarControler* objekt u *onResume* funkciji uspostavlja veze, dok u *onPause* i *onDestroy* prekida. Također u *onPause* i *onDestroy* funkcijama, *CameraClass* objekt prekida proces dohvaćanja slika s kamere. Kod prepisane *onPause* funkcije dan je u prilogu P.5.13.

Prilikom dodira ekrana poziva se *onTouch* funkcija, koju smo već spomenuli. Ako prije toga dodira nije određena boja objekta, odrede se koordinate mjesta dodira u odnosu na koordinatni sustav ekrana i srednja vrijednost boje oko njih. Srednju vrijednost boje određuje objekt *ColorBlobDetector* klase funkcijom *setHsvColor*za HSV prostor boja unutar kvadrata 8x8 piksela okolo mjesta dodira. Nakon što se odredi vrijednost srednje boje za HSV kanale, okvir slike se procesira i pretražuje se najveća kontura te boje, funkcijom *findMaxContour*. Ako postoji kontura te boje, *ColorBlobDetector* objekt inicijalizira područje interesa (eng. *region of interes*) funkcijom *findInitialRoi,* a *CarController* objekt postavlja inicijalnu veličinu konture. Inicijalna veličina konture objekta nam je potrebna da vozilo zna procjeniti treba li ići naprijed ili nazad, udaljavati se od objekta ili približavati. Ako je već određena boja objekta, ponovnim dodirom se zaustavi vozilo i omogući novi odabir objekta. Kod *onTouch* funkcije dan je u prilogu P.5.14., međutim pseudokod je sljedeći:

```
Kad se dodirne ekran i ako nije odabrana boja
  Odredi koordinate dodira
  Oduzmi od njih 4 piksela i napravi kvadrat veličine 8x8 piksela
    (nove koordinate su koordinate lijevog gore kuta)
  Iz cijele slike izdvoji sliku koju opisuje kvadrat 8x8 piksela
  Zbroji sve vrijednosti piksela male slike svakog kanala posebno i podjeli ih sa 64
  Srednjoj vrijednosti svakog kanala oduzmi broj kako bi napravio donju granicu i
  dodaj broj kako bi napravio gornju granicu.
  Pronađi konturu na slici
  Ako je kontura pronađena
    Inicijaliziraj područja interesa i veličinu konture
```

```
Kad se dodirne i ako je već odabrana boja
  Zaustavi vozilo
  Omogući ponovni odabir boje
```

Nakon što se odredi boja objekta, pozivanjem funkcije *onCameraFrame* svaki se okvir slike procesira. Njihovim se procesiranjem određuju koordinate konture u prostoru od interesa.

Ako je odabran objekt i određena njegova boja, pronalazi se kontura na prostoru od interesa. Ukoliko ona postoji, postavlja se novo područje interesa funkcijom *updateRoi*. Ažuriraju se granice za okretanje vozila u lijevo i desno funkcjiom *updateBound* te se ovisno o veličini i poziciji objekta pomiče vozilo, funkcijom *moveCar*. Ako kontura nije pronađena, vozilo se zaustavi i ponovno počinje pretraga cijele slike za objektom odabrane boje. Kod *onCameraFrame* funkcije dan je u prilogu P.5.15., dok je pseudokod sljedeći:

```
Ako je boja označena i postoji kontura
  Pronađi novu konturu na prostoru od interesa
  Ako postoji kontura
    Ažuriraj područje interesa i granice zakretanja vozilo
    Pomakni vozilo
  Ako ne postoji kontura
    Zaustavi vozilo
Ako je boja označena i ne postoji kontura
  Pronađi novu konturu na cijeloj slici
```

Kao što možemo zaključiti iz predhodnog teksta, objekt klase *ColorBlobDetector* obavlja posao pronalaženja objekta. ColorBlobDetector klasa ima četiri bitne funkcije, a to su: *setHsvColor*, *findMaxContour*, *initRoi* i *updateRoi*.

setHsvColor funkcija za zadane parametre prima cijelu sliku tipa *Mat* i kvadrat 8x8 piksela tipa *Rect*. Iz cijele slike uzima dio koji je opisan kvadratom 8x8 piksela i pretvara ga iz RGB u HSV prostor boja funkcijom *Imgproc.cvtColor*. Nakon toga računa srednju vrijednost svih piksela unutar kvadrata. Kod *setHsvColor* funkcije dan je u prilogu P.5.16., a pseudokod je sljedeći:

```
Iz cijele slike odvojio dio koji opisuje kvadrat područja interesa i spremi ga u novu
varijablu
Novu varijabu u koju je spremljen dodirnuti dio slike pretvori u HSV prostor boja
Zbroji sve piskele svakog kanala
Podjeli vrijednosti zbroja piksela svih kanala s umnoškom visine i širine kvadrata
Od srednje vrijednosti za H,S i V kanal dodaj broj n kako bi dobio gornju granicu,
odnosno oduzmi broj n kako bi dobio donju granicu.
```

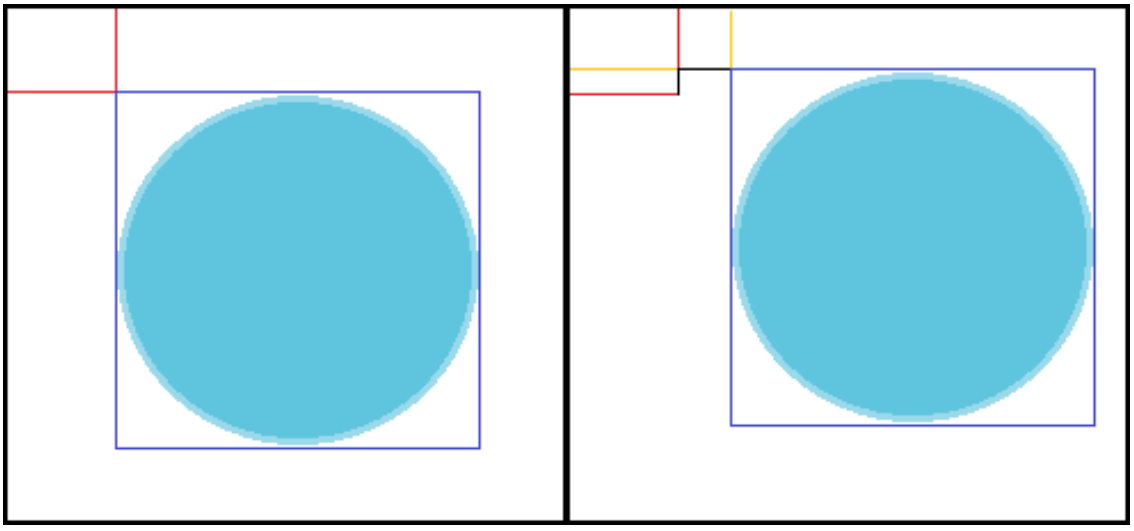
Funkcijom *calculateUpperAndLowerBound* računaju se gornje i donje granice za svaki kanal HSV prostora boja. Gornja i donja granica za *hue* kanal se dobije tako da se od srednje vrijednosti oduzme odnosno zbroji broj 25. Dok se granice za *saturation* i *value* kanal dobije tako da se od srednje vrijednosti oduzme odnosno zbroji broj 50. Broj koji određuje gornju i donju granicu može se postaviti funkcijom *setColorRadius*. Kod funckije calculateUpperAndLowerBound dan je u prilogu P.5.17.

findMaxContour za ulazni parametar prima *Mat* varijablu koja predstavlja sliku nad kojom se vrši operacija pronalaženja konture. Prvo se slika dva puta umanji kako bi je zamutili i smanjili broj podataka koji obrađujemo, tj. veličinu matrice. Slika se pretvori u HSV prostor boja te se s gornjim i donjim granicama odredi binarna slika, odnosno maska objekta. Operacijama kao što je *dilate* pokušava se umanjiti šum na binarnoj slici kako bi se izbjegle nepotrebne konture. Funkcija *Imgproc.findContours* pronalazi konture na binarnoj slici i sprema ih u listu. Iz liste se pronalazi najveća kontura i računa se njezina površina pomoću Greenove formule objašnjene u poglavlju 3.1. Funkcija koja implementira tu formulu je *Imgproc.contourArea*. Veličina površine se sprema u varijablu tipa *double* i šalje kao povratni tip. Kod funkcije findMaxContour dan je u prilogu P.5.18. Obradom podataka iz inicijaliziranoga područja interesa, a ne cijelog okvira slike postižu se određena poboljšanja algoritma. Broj okvira po sekundi (eng. *frame per second*) se povećao s prosječnih 7 za rezoluciju slike od 1280 x 720 piksela na 19, dok kod rezolucije slike od 720 x 480 piksela ide i do 28 okvira po sekundi. U daljnjem tekstu su opisane funkcije koje omogućuju to poboljšanje.

findInitialRoi funkcija služi za inicializaciju prostora od interesa kad se prvi put dodirne ekran i odredi željeni objekt. Ako postoje konture u listi, uzme se ona najveća i uveća se četiri puta funkcijom *Core.multiply*. Četiri puta se uveća zato što je prilikom traženja kontura bila umanjena četiri puta. Momenti se odrede funkcijom *Imgproc.moments* i iz njih se izluče koordinate centra konture, odnosno centroid. Aproksimacijom se odredi kvadrat koji opisuje konturu i pomoću njega se dobije širina i visina konture. *globalRect* je globalna varijabla tipa *Rect* koja opisuje područje interesa. Sadrži koordinate lijevog gornjeg vrha kvadrata u odnosu na koordiatni sustav okvira slike. Prilikom inicijalizacije *globalRect* varijabla se postavlja tako da ima pomak (eng. offset) u lijevo, desno i gore,dole. Potrebno je uzeti malo više prostora oko objekta kako bi on svojim pomicanjem ostao u prostoru od interesa. Podsjetimo se, obrađujemo samo područja interesa, a ne cijeli okvir slike. Kod *initRoi* funkcije dan je u prilogu P.5.19.

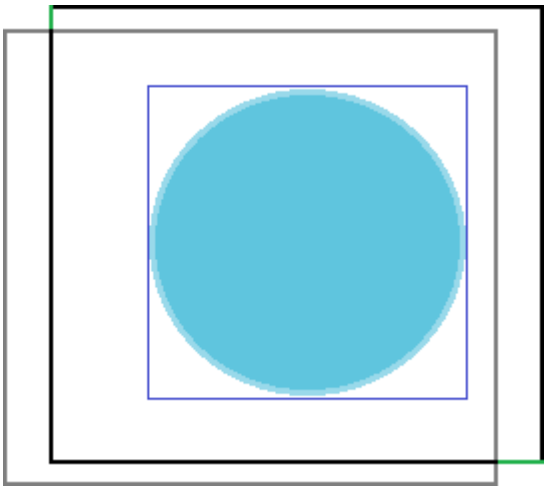
Nakon inicijalizacije globalne varijable *globalRect* započinje obrada prostora od interesa, unutar *onCameraFrame* funkcije. Na početku je obrada ista kao i kod inicijalizacije prostora od interesa, međutim izmijenjen je dio u kojemu se mijenjaju koordinate pozicije prostora od interesa.

Varijabla *rect* ima informacije o koordinatama kvadrata koji opisuje konturu u odnosu na područja interesa u kojemu se kontura traži. Ako su koordinate novoga kvadrata koji opisuje konturu veće od pomaka to znači da se kontura pomaknula u desno odnosno dole, odnosno ako su koordinate manje od pomaka to znači da se kontura pomaknula u lijevo odnosno gore. Na lijevoj slici 5.1. možemo vidjeti inicijalizirano područje interesa, kvadrat koji opisuje konturu (plava boja) i zadani pomak (crvena boja). Na desnoj slici 5.1. je pokazan primjer kad se kontura pomakne desno i gore. Crnom bojom je označena razlika u pomaku, te se na slici 5.2. može vidjeti novonastalo područje interesa.



Sl. 5.1 Pomicanje kvadrata koji opisuje konturu

Na slici 5.2. novo područje interesa je označen kvadratom crne boje, dok je staro područje interesa označen kvadratom sive boje. Svijetlo zelenom bojom označeno je za koliko se novi kvadrat pomaknio gore i desno.



Sl. 5.2 Novo područje interesa

Nakon određenih novih koordinata pomiče se područje interesa i računaju se nove koordinate centroida. Kod *updateRoi* funkcije dan je u prilogu P.5.20.

Već navedene funkcije za upravljanje vozilom se nalaze u klasi *CarController*. Klasa CarController osim funkcija za uspostavu veze s Arduinom, koje su opisane u poglavlju *MainActivity*, ima i dvije funkcije koje služe za upravljanje vozilom. To su funkcije *moveCar* i *updateBound*. *moveCar* funkcija prima dva parametra, veličinu konture i njezinu poziciju. Ovisno o veličini konture vozilo se pomiče naprijed i nazad. Ako pogledamo neki predmet i on je jako malen, možemo pretpostaviti da se nalazi daleko od nas, dok u suprotnome, ako je predmet velik, možemo pretpostaviti da se on nalazi blizu nas. Ovisno o poziciji predmeta vozilo se okreće lijevo i desno. Ukoliko se veličina konture nalazi između gornje i donje granice vozilo miruje, isto tako vrijedi i za okretanje u lijevo i desno. Gornja i donja granica za kretanje naprijed i nazad su konstantne i iznose ±20% od inicijalne veličine konture, dok se gornja i donja granica za pomicanje u lijevo i desno neprestano ažurira funkcijom *updateBounds*. Kod *moveCar* funkcije dan je u prilogu P.5.21., a njezin pseudokod je sljedeći:

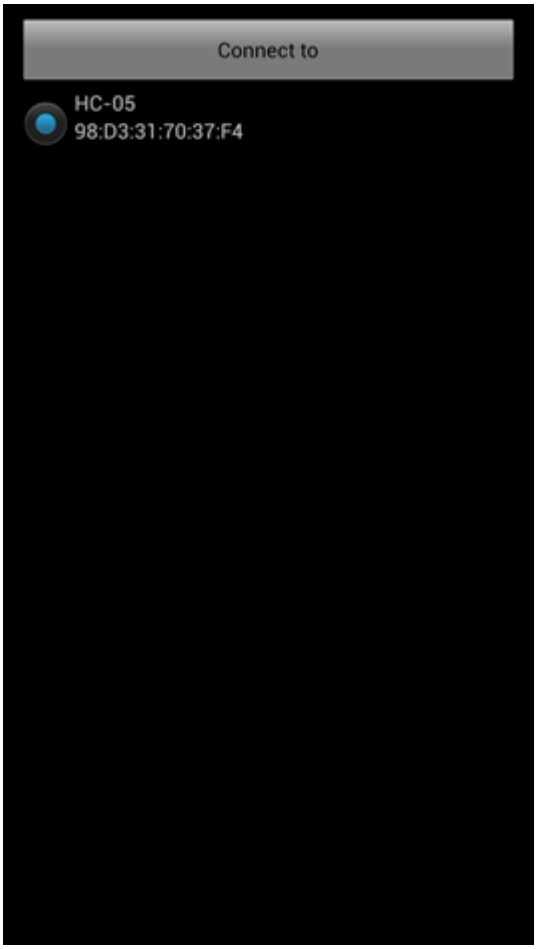
```
Ako je x koordinata veća od donje granice i ako je manja od gornje granice onda
  Ako je veličina konture manja od gornje granice i ako je veća od donje granice onda
    Zaustavi vozilo i miruj
  Ako je veličina konture veća od gornje granice
    Vozilo pomiči unazad
  Ako je veličina konture manja od donje granice
    Vozilo pomiči u naprijed
Ako je x koordinata manja od donje granice
  Ako je veličina konture veća od gornje granice ili manja od donje granice
    Rotiraj vozilo u desno
  U suprotnome
```

```
    Zakreni vozilo u desno
Ako je x koordinata veća od gornje granice
  Ako je veličina konture veća od gornje granice ili manja od donje granice
    Rotiraj vozilo u lijevo
  U suprotnome
    Zakreni vozilo u lijevo
```

Ažuriranje granica za zakretanje lijevo i desno ovisi o veličini objekta, to ažuriranje provodi funkcija *updateBound*. Ukoliko je objekt daleko od vozila granica se smanjuje i vozilo postaje osjetljiviji na pomicanje objekta. Međutim, ako je objekt blizu vozila, granica se poveća i vozilo prestane reagirati na pomicanje objekta dok se dovoljno ne udalji od njega. Granice ne mogu biti manje od postavljene konstantne vrijednosti, dok maksimalnu vrijednost granica određuje širina okvira slike. Kod funkcije *updateBound* dan je u prilogu P.5.22.

5.2.3. Izgled grafičkog sučelja android aplikacije

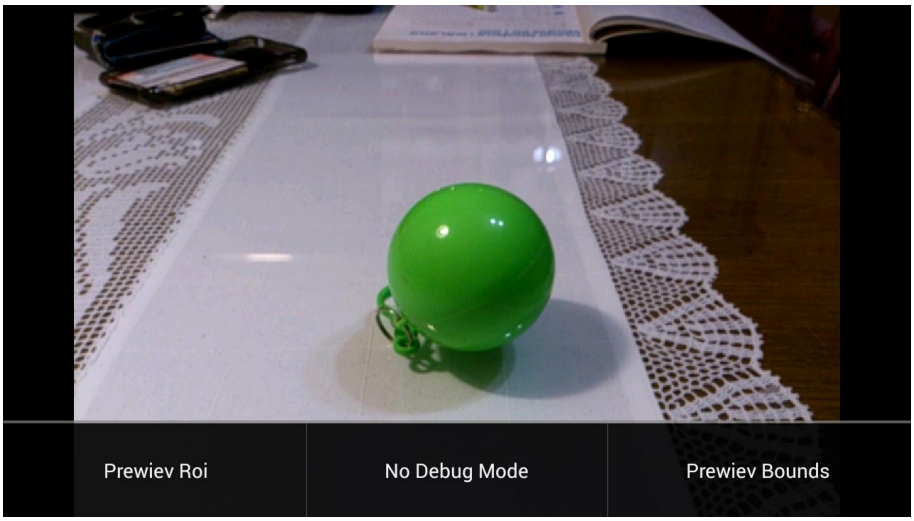
Kao što je već objašnjeno, izgled android Activitiy-a definiran je XML kodom. XML (eng. *EXtensible Markup Language*) je opisni jezik koji je napravljen kako bi bio jednostavno čitljiv računalima i ljudima. Primjer sličnoga jezika je HTML. U MainActivity-u definiran je jedan *Button View* i jedan *ListView* što u konačnici izgleda kao na slici 5.3.



Sl. 5.3 Izgled MainActivity-a

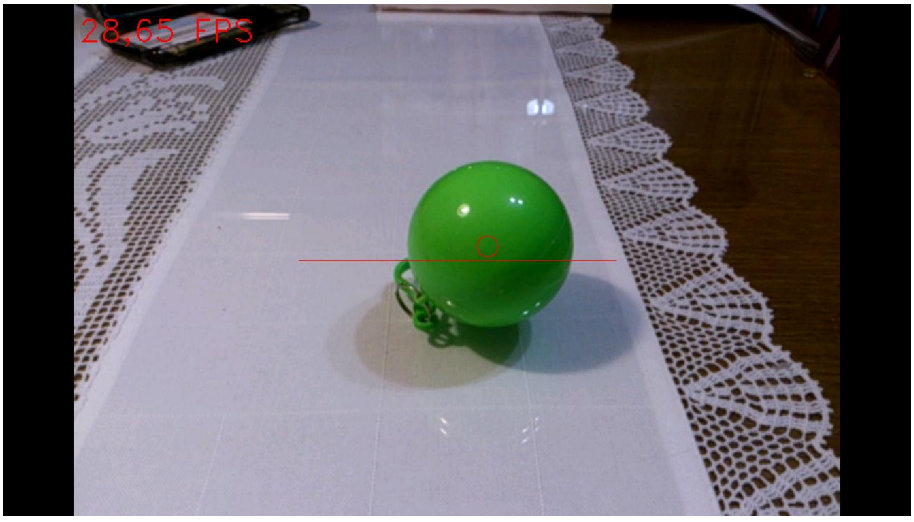
U *CameraActivity*-u preko cijeloga ekrana nalazi se rastegnuta *CameraClass* klasa koja nasljeđuje *JavaCameraView*. To je view koji omogućuje prikazivanje slika s kamere, dopušta promjenu rezolucije te ima informacije o veličini ekrana i slike. S funkcijom *setOnTouchListener* omogućuje se *onTouch* funkcija. *onTouch* funkcija se poziva kada se dodirne prikazana slika kamere na ekranu mobitela. Dok se s funkcijom *setCvCameraViewListener* omogućuju funkcije kao što je *onCameraFrame*, unutar koje se obrađuju slike s kamere.

Zbog lakšeg nadgledanja rada aplikacije u *CameraActivity*-u su isprogramirana dva *moda* za otklanjanje pogrešaka (eng. *debug mod*). Prvi *mod* prikazuje koliko kamera dohvaća okvira po sekundi, točku centroida objekta te gornju i donju granicu za zakretanje vozila. Drugi mod prikazuje područje interesa. Na slici 5.4. vidimo odabir modova za otklanjanje pogrešaka.

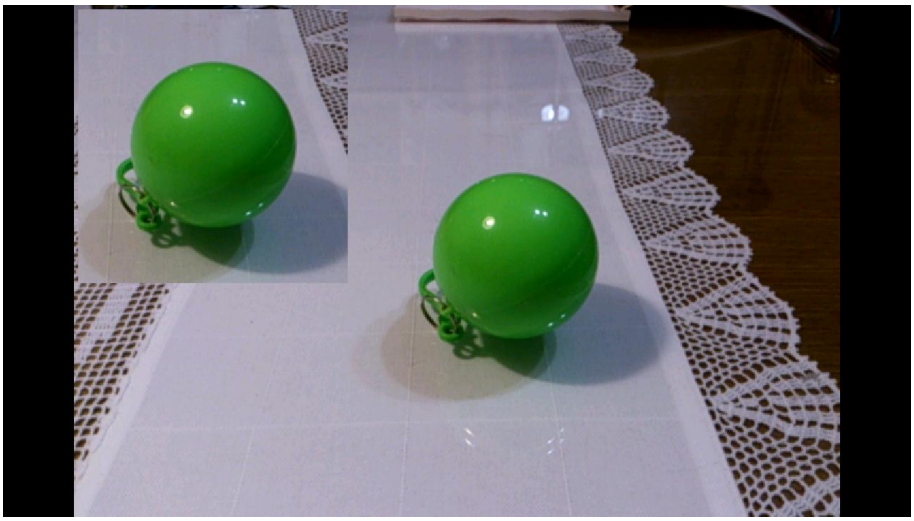


Sl. 5.4 Odabir debug modova.

Na slici 5.5. je *mod* koji prikazuje granice zakretanja i ostale navedene stavke, dok je na slici 5.6. *mod* koji prikazuje područje interesa.



Sl. 5.5 *Mod* koji prikazuje granice zakretanja, broj okvira po sekundi i centroid objekta.



Sl. 5.6 *Mod* koji prikazuje područje interesa

6. ZAKLJUČAK

U ovome diplomskome radu pokazano je da za postizanje elementarne funkcionalnosti slijeđenja objekta mobilnim robotom pomoću računalnog vida nisu potrebna robotska kolica velikih dimenzija na kojima je montiran laptop koji obrađuje podatke i upravlja njima, već se razmatrana funkcionalnost može postići i kombiniranjem mobitela s Android operacijskim sustavom i Arduino Uno platforme. U sklopu diplomskoga rada napravljeno je sklopovsko i programsko rješenje za robotska kolica. Arduino Uno platforma služi kao uređaj koji prima naredbe i pomiče robotska kolica, slave uređaj. Pametni telefon prikuplja informacije okoline pomoću ugrađene kamere, obrađuje ih i šalje poruke Arduinu, master uređaju. Programsko rješenje za Arduino Uno pisano je u objektno orijentiranom C++ programskom jeziku. Kako bi kodiranje bilo lakše i preglednije napisana je biblioteka za Arduino, *CarController*. *CarController* klasa pomaže pri upravljanju s robotskim kolicima, odnosno upravljanju istosmjernih (DC) motora. Programsko rješenje pametnog telefona pisano je u Java programskome jeziku za Android operacijski sustav. Sve funkcije za obrađivanje slike s kamere omogućila je biblioteka OpenCV4Android, koja je zapravo OpenCV biblioteka namjenjena za pisanje Android aplikacija. Algoritam praćenja objekta temeljen je na boji objekta. Pretvaranjem slike iz RGB prostora boja u HSV prostor boja te primjenom metode praga na *hue, saturation* i *value* kanale dobivamo binarnu sliku, odnosno masku objekta. Određivanjem najveće konture na binarnoj slici unutar koje se nalazi odabrani objekt, možemo odrediti centroid objekta te na osnovi njega znati gdje se objekt nalazi na slici. Kada znamo gdje se objekt nalazi na slici lako je slati podatke na Arduino i upravljati robotskim kolicima. Ovo je najjednostavniji algoritam praćenja objekta. U sljedećim verzijama ovoga rada mogao bi se promjeniti algoritam tako da prati predmet ovisno o njegovim SIFT značajkama, praćenje bi se poboljšalo i implementiranjem Kalmanovog filtera te programiranjem cijeloga algoritma u C++-u, odnosno u JNI okruženju.

LITERATURA

[1] Mrežnica (oko), Wikipedia. [Online]. https://hr.wikipedia.org/wiki/Mre%C5%BEnica_(oko)

[2] Visible spectrum, Wikipedia. [Online]. https://en.wikipedia.org/wiki/Visible_spectrum

[3] Boja, Wikipedia. [Online]. https://hr.wikipedia.org/wiki/Boja

[4] A comprehensive guide to parallel video decoding, Emeric's GSoC blog. [Online]. https://emericdev.wordpress.com/2011/08/26/a-comprehensive-guide-to-parallel-video-decoding/

[5] Boja u računarskoj grafici, FESB. [Online]. http://lab405.fesb.hr/igraf/Frames/fP5_1.htm

[6] Begušić D. Boja u računarskoj grafici. [Online]. http://lab405.fesb.hr/igraf/Frames/fP5_1.htm

[7] OpenCV. Miscellaneous Transformations, cvtColor. [Online]. http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor

[8] Robert Cupec. (2011, Listopad) Temeljne operacije filtriranja slike.

[9] Johannes Kilian, Simple Image Analysis By Moments Version 0.2, 2001.

[10] Serial Port Bluetooth Module (Master/Slave) : HC-05, ITEAD Wiki. [Online]. http://wiki.iteadstudio.com/Serial_Port_Bluetooth_Module_(Master/Slave)_:_HC-05

[11] Getting Started with Arduino on Windows, Arduino. [Online]. https://www.arduino.cc/en/Guide/Windows

[12] Deek-Robot. [Online]. http://www.deek-robot.com/productShow.asp?id=17

[13] LCL. (2013, Lipanj) lcl-optika. [Online]. http://lcl-optika.hr/kako-funkcionira-ljudsko-oko/

SAŽETAK

Zadatak diplomskoga rada je napraviti robotska kolica koja autonomno prate odabrani objekt primjenom računalnog vida. Sklopovlje robotskih kolica je realizirano pomoću Arduino uno platforme i pametnoga telefona, dok je komunikacija između njih omogućena HC-05 bluetooth modulom. Algoritam za praćenje objekta na pametnom telefonu pisan je u Java programskom jeziku pomoću biblioteke Opencv4Android. Dok je kod za Arduino uno pisan u C++ programskom jeziku. Algoritam za praćenje objekta na pametnome telefonu temelji se na praćenju boje zadanog objekta, dodirom na ekran mobitela pohrani se informacija o boji objekta te se obradom svake sljedeće slike koju kamera snimi određuje kontura odabranog objekta i njegov centroid. Ovisno o položaju objekta na slici pametni telefon šalje poruke Arduinu. U porukama koje prima Arduino uno nalaze se podaci o smjeru kretanja robotskih kolica, brzini kretanja i koliko dugo će se kretati. Na Arduinu je isprogramiran program koji služi za parsiranje primljenih poruke te pomicanje robotskih kolica.

Ključne riječi: mobilni robot, Arduino uno, pametni telefon, praćenje objekta bazirano na boji, opencv4android.

ABSTRACT

The task of this thesis is to build a robotic car which follows an object autonomously using computer vision. The hardware of the robotic car is realised with an Arduino uno platform and a smartphone which comunicate via HC-05 bluetooth modul. The program for object tracking using a smartphone is written in Java programming language with library Opencv4Android while the code for Arduino is written in C++ programming language. The object tracking algorithm is color based. The information about the color of the object is stored with a touch on the mobile phone screen. The program processes every next image that the camera records finds the contour and compute the centroid of the object. The smartphone sends a messages to Arduino depending on the location of the tracked object. The messages that Arduino recieves contain information about the direction of motion of the robotic car, speed of motion and how long the motion should be performed. There is a program on Arduino which serves for parsing the recieved messages and for moving the robotic car.

Keywords: mobile robot, Arduino uno, smartphone, color based tracking, opencv4android.

ŽIVOTOPIS

Antonio Lončar, rođen 18.01.1992. godine u Đakovu, sin Snježane i Žarka. Pohađao osnovnu školu „I.G.Kovačića“ u Đakovu te nakon njenog završetka upisao prirodoslovno-matematičku gimnaziju „A.G.Matoš“ u Đakovu. Tijelom školskih dana imao je afinitete prema matematici, fizici i informatici više nego prema društvenim predmetima te se zbog toga nakon srednje škole odlučio upisati na Elektrotehnički fakultet Osijek, preddiplomski studij smjera računarstvo. Po završetku preddiplomskoga studija upisuje diplomski studij na istome fakultetu, smjera procesno računarstvo.

Potpis: _____

PRILOZI

Arduino

P.5.1. Kod za parsiranje poruka

```
byte speedin = 100;
int ndelay = 500, partOfInput = 0, j = 0;
char c[5], direct, nspeed[4], dtime[5] ;

void loop() {
while(myBluetooth.available()){
c[j++] = (char) myBluetooth.read();
if(c[j-1] == ':')
    {
      c[j-1] = '\0';

      if(partOfInput == 0)
      {
        partOfInput++;
        direct = c[0];

        if(direct == 'S' ||
         !(direct == 'F' || direct == 'B' || direct == 'R' || direct == 'L')
          )
        {
          myCar.stop();
          partOfInput = 0;
        }
      }
      else if(partOfInput == 1)
      {
        partOfInput++;
        strcpy(nspeed, c);
        speedin = stringToInt(nspeed, j-1);

        myCar.moveOn(direct, speedin);
      }
      else if(partOfInput == 2)
      {
         if(c[0] == 'S' )
         {
          myCar.stop();
          myBluetooth.write("stop");
         }
         else
         {
           strcpy(dtime, c);
           ndelay = stringToInt(dtime, j-1);

           //myCar.moveOnWithDelay(direct, speedin, ndelay);
           delay(ndelay);
           myCar.stop();
         }

         partOfInput = 0;
      }
```

```
      j = 0;
      }
    }
  }
```

MainActivity

P.5.2. XML kod MainActivity-a

```
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_margin="10dp">
<Button
android:text="Connect to"
android:id="@+id/buttonId"
android:layout_width="match_parent"
android:layout_height="wrap_content"/>
<ListView
android:id="@+id/deviceList"
android:layout_width="match_parent"
android:layout_height="wrap_content"/>
</LinearLayout>
```

P.5.3. XML kod View-a u ListView-u

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
<RadioButton
        android:id="@+id/rButton"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"/>
<LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
      <TextView
              android:text="DEVICE NAME"
              android:id="@+id/device_name"
              android:layout_height="wrap_content"
              android:layout_width="match_parent"/>
      <TextView
              android:text="DEVICE ADRESS"
              android:id="@+id/device_adress"
              android:layout_height="match_parent"
              android:layout_width="match_parent"/>
      </LinearLayout>
```

```
</LinearLayout>
```

P.5.4. Kod getView funkcije

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
	if(convertView == null)
	{
		isPopulated = true;
		convertView = View.inflate(super.getContext(), resourceId, null);
	}

	RadioButton rbutton = (RadioButton) convertView.findViewById(R.id.rButton);
	TextView deviceName = (TextView) convertView.findViewById(R.id.device_name);
	TextView deviceAddres = (TextView)
convertView.findViewById(R.id.device_adress);

	device = super.getItem(position);
	button.setChecked(position == selectedPosition);
	rbutton.setTag(position);

	rbutton.setOnClickListener(new OnClickListener() {
		@Override
		publicvoid onClick(View v) {
			selectedPosition = (Integer) v.getTag();
			notifyDataSetChanged();
		}
	});

	deviceName.setText(device.getName());
	deviceAddres.setText(device.getAddress());
	return convertView;

};
```

P.5.5. Inicijalizacija BroadcastReciever objekta

```
BroadcastReceiver mReciever = new BroadcastReceiver(){

	@Override
	publicvoid onReceive(Context context, Intent intent) {

	String intenetAction = intent.getAction();
	if(BluetoothDevice.ACTION_FOUND.equals(intenetAction))
		{
		BluetoothDevice br_device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
		try{
			if(!listNext.contains(br_device))
			{
			listNext.add(br_device);
			adapter.notifyDataSetChanged();
			}
		}
```

```
            catch(Exception e)
            {
                  Log.i("adapter", e.getMessage());
            }
            }
      }
};
```

P.5.6. Prepisana onClick funkcija gumba

```
button.setOnClickListener(new OnClickListener() {
@Override
publicvoid onClick(View v) {
      int rbPosition = adapter.getPositionOfSelectedRadioButton();
      if(rbPosition == -1)
      {
            Toast.makeText(getApplicationContext(), "Device isn't selected. Can't
find device!", Toast.LENGTH_SHORT).show();
      }
      else
      {
            device = listNext.get(rbPosition);
            Toast.makeText(getApplicationContext(), "Connecting to " +
device.getName() + "! Please wait...", Toast.LENGTH_SHORT).show();
            cancelIfDiscovering();

            mBTConnection = BluetoothConnection.getInstance(device);
            if(mBTConnection != null)
            {
            Intent intent = new Intent(MainActivity.this, CameraActivity.class);
            MainActivity.this.startActivityForResult(intent,
      StaticVariables.SECOND_ACTIVITY);
            }
            else
            {
                  Toast.makeText(getApplicationContext(), "Try again. Instance is
null", Toast.LENGTH_SHORT).show();
            }
      }
}
});
```

P.5.7. Funkcija openSocket

```
public boolean openSocket()
{
device = BluetoothAdapter.getDefaultAdapter().getRemoteDevice(adress);

try {
      Method m = device.getClass().getMethod("createRfcommSocket", new
Class[]{int.class});
      socket = (BluetoothSocket) m.invoke(device, 1);
```

```
}
catch (Exception e) {
	Log.i("Error", e.getMessage());
return false;
}
OutputStream tmpos = null; //temporary variable for stream
InputStream tmpis = null;

try {
	socket.connect();
	tmpos = socket.getOutputStream();
	tmpis = socket.getInputStream();
} catch (IOException e) {
	Log.i("Error", e.getMessage());
	return false;
}
is = tmpis; //global variable for stream
os = tmpos;
return true;

}
```

P.5.8. Funkcija closeSocket

```
public boolean closeSocket()
{
	try {
		socket.close();
	} catch (IOException e) {
		Log.i("Error", e.getMessage());
	return false;
	}

return true;

}
```

P.5.9. Funkcija write

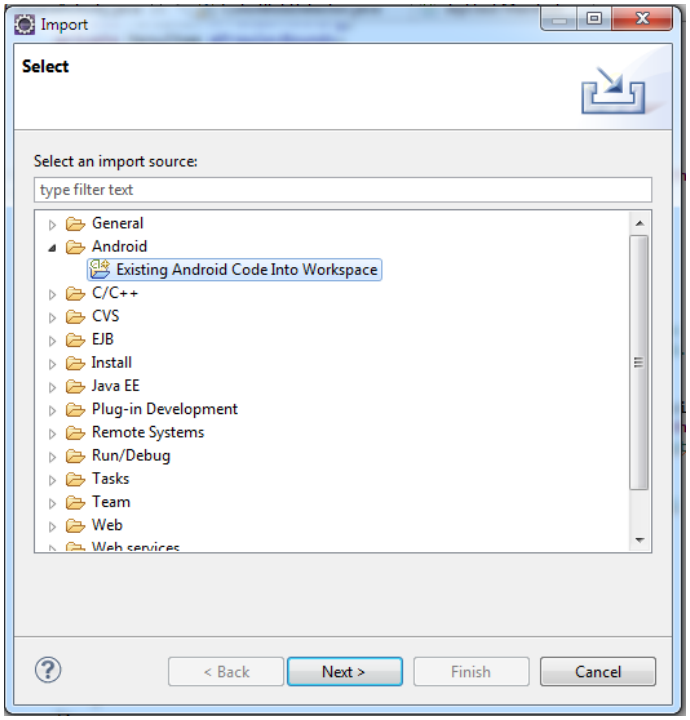
```
public boolean write(String msg){
	byte[] b = msg.getBytes();
	try {
		os.write(b);
		os.flush();
	} catch (IOException e) {
		Log.i("Error", e.getMessage());
		return false;
	}
return true;

}
```

CameraActivty

P.5.10. Postavljanje OpenCV4Android biblioteke

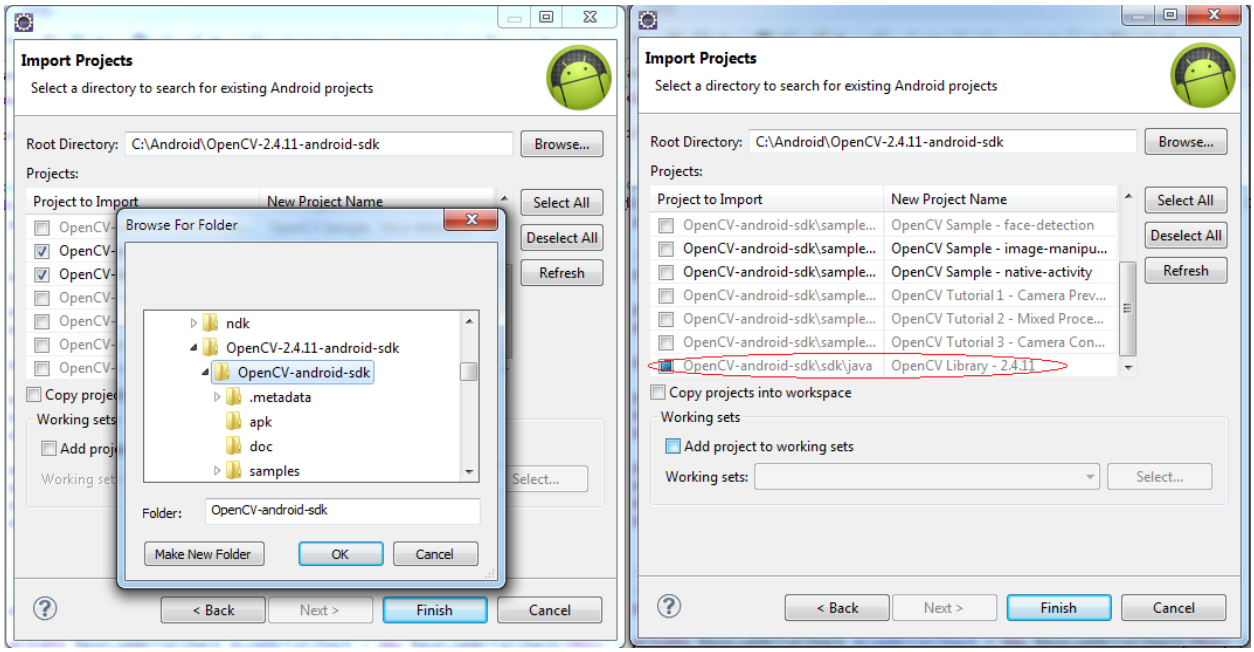
OpenCV je C/C++ biblioteka napravljena za izradu aplikacija koje su uglavnom usmjerene na obradu slike u stvarnome vremenu, odnosno računalni vid. OpenCV4Android biblioteka je java omotač (eng. *javawrapper*) OpenCV biblioteke napravljen da se može koristiti na Android operacijskim sustavima. Ako pretpostavimo da je biblioteka skinuta na računalo sa službene stranice, sljedeći postupak omogućuje njenu upotrebu u Eclipse IDE-u.

Prvo je potrebno uvesti (eng. *import*) već potojići projekt (File -> Import...)



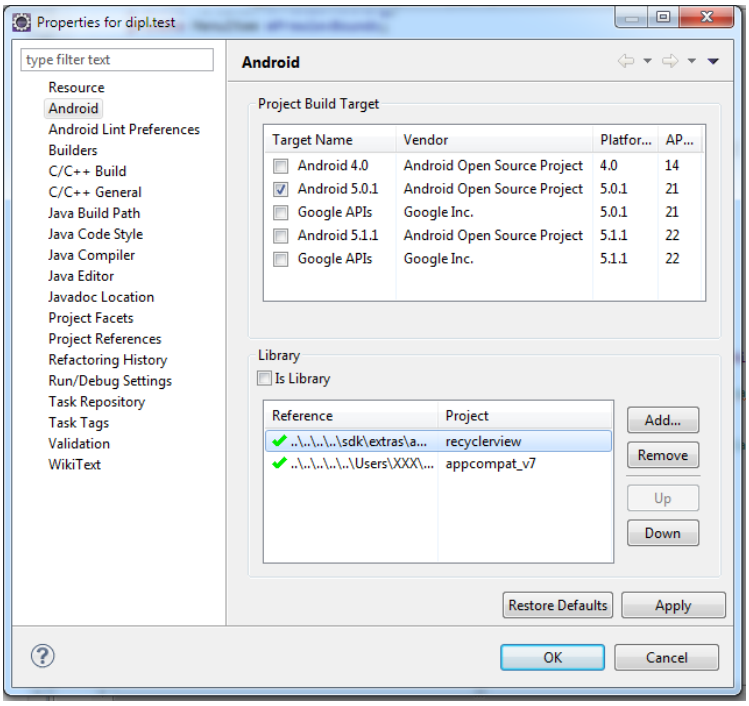
Sl. 1 Uvoz već postojećeg projekta

Odabirom gumba *Next* otvara se prozor u kojemu je potrebno odabrati korijenski direktorij (eng. *root directory*) u kojemu se nalazi OpenCV4Android biblioteka. Nakon toga se odabere željeni projekt koji se uvozi, a u našem slučaju je to OpenCV Library- 2.4.11.



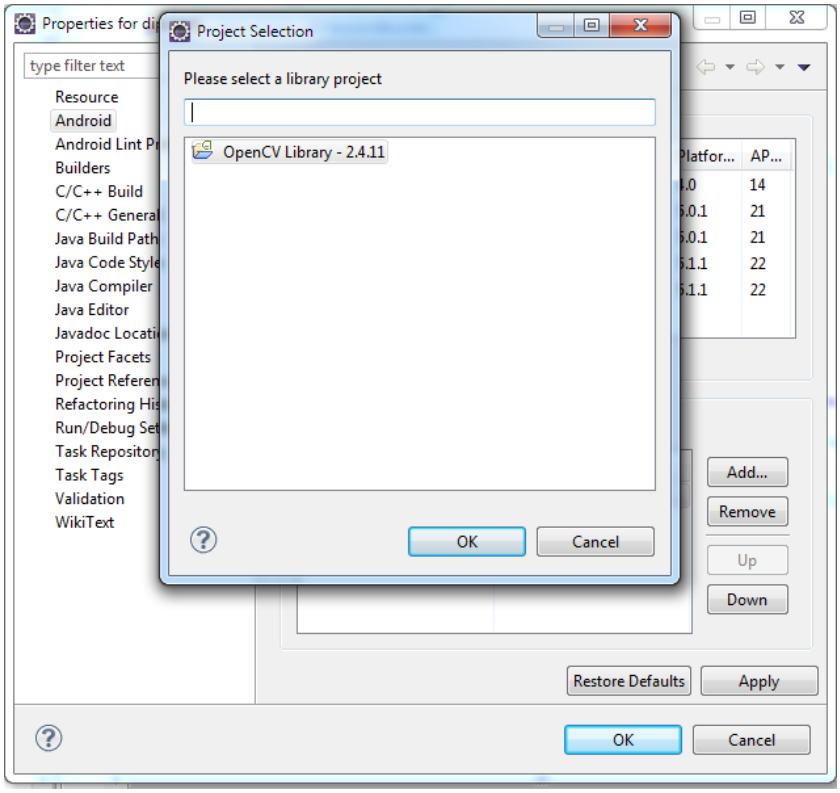
Sl. 2 Uvoz OpenCV bibloteke

Sada je još potrebno dodati uvezenu bibloteku u naš projekt gdje je želimo koristiti. Kliknemo desni klik na projekt i odemo na karticu *Properties* te u novootvorenome prozoru odaberemo karticu *Android.*



Sl. 3 Kartica Android.

U kartici *Android* pritisnemo gumb *Add...* i dodamo projekt koji se zove kao i uvezena bibloteka: OpenCv Library – 2.4.11. Nakon što pritisnemo gumb *Ok* i spremimo sve postavke možemo početi s korištenjem biblioteke.



Sl. 4 Odabir biblioteke.

P.5.11. XML kod CameraActivity-a

```
<RelativeLayoutxmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent">
<hr.etfos.diplomski.ImageProcessing.CameraClass
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:id="@+id/camera"/>
</RelativeLayout>
```

P.5.12. Inicijalizacija BaseLoaderCallback objekta

```
private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
@Override
public void onManagerConnected(int status) {
switch (status) {
case LoaderCallbackInterface.SUCCESS:
   {
mOpenCvCameraView.enableView();
mOpenCvCameraView.setCvCameraViewListener(CameraActivity.this);
mOpenCvCameraView.setOnTouchListener(CameraActivity.this);
mOpenCvCameraView.setResolution(480, 720);

      mDetector = new ColorBlobDetector();
meter = new FpsMeter();
meter.init();
      }break;
default:
      {
super.onManagerConnected(status);
      } break;
   }
 }
};
```

P.5.13. Prepisana onPause funkcija

```
@Override
public void onPause()
    {
       super.onPause();

if (mOpenCvCameraView != null)
mOpenCvCameraView.disableView();

mIsColorSelected = false;
mCarController.stopCar();
mCarController.closeConnection();
  }
```

P.5.14. Prepisana onTouch funkcija

```
@Override
public boolean onTouch(View v, MotionEvent event) {
	if(!mIsColorSelected)
	{
	int cols = mRgba.cols();
	int rows = mRgba.rows();
	int xpos = (int) (event.getX() * cols) / mOpenCvCameraView.getWidth();
	int ypos = (int) (event.getY() * rows) / mOpenCvCameraView.getHeight();
	if ((xpos < 0) || (ypos < 0) || (xpos > cols) || (ypos > rows)) return false;

mCarController.setCentreOfWidth(mRgba.cols()/2);
	Rect roiRect = new Rect();
	roiRect.x = (xpos>4) ? xpos-4 : 0;
	roiRect.y = (ypos>4) ? ypos-4 : 0;
	roiRect.width = (xpos+4 < cols) ? xpos + 4 - roiRect.x : cols - roiRect.x;
	roiRect.height = (ypos+4 < rows) ? ypos + 4 - roiRect.y : rows - roiRect.y;

	mDetector.setHsvColor(mRgba, roiRect);
	mInitialContureArea = mDetector.findMaxContour(mRgba);
	if(mInitialContureArea != -1)
	{
		mCarController.setInitialContourArea(mInitialContureArea);
		mDetector.initRoi(mRgba);
		mIsColorSelected = true;
		contourLost = false;
	}
	}
	else
	{
		mIsColorSelected = false;
		mCarController.stopCar();
	}

return false;
}
```

P.5.15. Funkcija onCameraFrame

```
public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
mRgba = inputFrame.rgba();

if(mIsColorSelected&& !contourLost)
{
	double mContureArea = mDetector.findMaxContour(mDetector.getRoi());
	if(mContureArea != -1 &&mCarController.isSocketReady())
	{
		mDetector.updateRoi(mRgba);
		mCarController.updateBound(mDetector.getRect());
		mCarController.moveCar(mDetector.getCentroid(), mContureArea);
	}
	else{
```

```
                mCarController.stopCar();
                contourLost = true;
        }
}
else if(contourLost&&mIsColorSelected&&mCarController.isSocketReady())
{
        double area = mDetector.findMaxContour(mRgba);
        if(area != -1)
        {
                mDetector.findInitialRoi(mRgba);
                contourLost = false;
        }
}
return mRgba;
}
```

P.5.16. Funkcija setHsvColor

```
public void setHsvColor(Mat image, Rect roi) {
        Mat touchedRgb = image.submat(roi);
        Mat touchedHsv = new Mat();
        Scalar mBlobColorHsv = new Scalar(255);
        Imgproc.cvtColor(touchedRgb, touchedHsv, Imgproc.COLOR_RGB2HSV_FULL);
        mBlobColorHsv = Core.sumElems(touchedHsv);
        int pointCount = roi.width*roi.height;
        for (int i = 0; i < mBlobColorHsv.val.length; i++)
                mBlobColorHsv.val[i] /= pointCount;

        calculateUpperAndLowerBound(mBlobColorHsv);
        Imgproc.resize(touchedRgb, mRoi, ROI_SIZE);

        touchedHsv.release();
        touchedRgb.release();
}
```

P.5.17. Funkcija calculateUpperAndLowerBound

```
private void calculateUpperAndLowerBound(Scalar hsvColor) {
double minH = (hsvColor.val[0] >= mColorRadius.val[0]) ? hsvColor.val[0] -
mColorRadius.val[0] : 0;
double maxH = (hsvColor.val[0] + mColorRadius.val[0] <= 255) ? hsvColor.val[0] +
mColorRadius.val[0] : 255;

mLowerBound.val[0] = minH;
mUpperBound.val[0] = maxH;
mLowerBound.val[1] = hsvColor.val[1] - mColorRadius.val[1];
mUpperBound.val[1] = hsvColor.val[1] + mColorRadius.val[1];
mLowerBound.val[2] = hsvColor.val[2] - mColorRadius.val[2];
mUpperBound.val[2] = hsvColor.val[2] + mColorRadius.val[2];
mLowerBound.val[3] = 0;
mUpperBound.val[3] = 255;
```

```
}
```

P.5.18. Funkcija findMaxContour

```
public double findMaxContour(Mat rgbaImage)
    {
      mMaxContourArea = -1;

        Imgproc.pyrDown(rgbaImage, mPyrDownMat);
        Imgproc.pyrDown(mPyrDownMat, mPyrDownMat);
        Imgproc.cvtColor(mPyrDownMat, mHsvMat, Imgproc.COLOR_RGB2HSV_FULL);

        Core.inRange(mHsvMat, mLowerBound, mUpperBound, mMask);
        Imgproc.dilate(mMask, mDilatedMask, new Mat());

contours.clear();
        Imgproc.findContours(mDilatedMask, contours, mHierarchy,
Imgproc.RETR_EXTERNAL, Imgproc.CHAIN_APPROX_SIMPLE);

      for(int i = 0; i <contours.size(); i++)
      {
            double area = Imgproc.contourArea(contours.get(i));

            if(area >mMaxContourArea)
            {
                  mMaxContourArea = area;
                  maxContourPosition = i;
            }
      }

return mMaxContourArea;
    }
```

P.5.19. Funkcija initRoi

```
public void initRoi(Mat rgbaImage)
{
if(contours.size() > 0)
      {
            mBiggestContour = contours.get(maxContourPosition);
            Core.multiply(mBiggestContour, new Scalar(4,4), mBiggestContour);

            Moments m = Imgproc.moments(mBiggestContour);
            centroid.x =  (m.get_m10() / m.get_m00());
            centroid.y = (m.get_m01() / m.get_m00());

            MatOfPoint2f contour2f = new MatOfPoint2f(mBiggestContour.toArray());
            double approxDistance = Imgproc.arcLength(contour2f, true)*0.01f;
            Imgproc.approxPolyDP(contour2f, approxCurve, approxDistance, true);

            globalRect = Imgproc.boundingRect(new
MatOfPoint(approxCurve.toArray()));

            int newy = globalRect.y - offset< 0 ? 0 : globalRect.y - offset;
```

```
            int newx = globalRect.x - offset< 0 ? 0 : globalRect.x;
            int newheight = globalRect.y + globalRect.height + offset>
rgbaImage.rows() ? globalRect.height : globalRect.height + offset;
            int newwidth = globalRect.x + globalRect.width + offset>
rgbaImage.cols() ? globalRect.width : globalRect.width + offset;

            globalRect.x = newx;
            globalRect.y = newy;
            globalRect.width = newwidth;
            globalRect.height = newheight;
            mRoi = rgbaImage.submat(globalRect);

            contour2f.release();
      }
}
```

P.5.20. Funkcija updateRoi

```
public void updateRoi(Mat rgbaImage)
{
mBiggestContour = contours.get(maxContourPosition);
Core.multiply(mBiggestContour, new Scalar(4,4), mBiggestContour);

MatOfPoint2f contour2f = new MatOfPoint2f(mBiggestContour.toArray());
double approxDistance = Imgproc.arcLength(contour2f, true)*0.01f;
Imgproc.approxPolyDP(contour2f, approxCurve, approxDistance, true);
rect = Imgproc.boundingRect(new MatOfPoint(approxCurve.toArray()));

if(rect.x>offset + 10)
      globalRect.x = globalRect.x + (rect.x - offset) < rgbaImage.cols() ?
globalRect.x + (rect.x - offset) : globalRect.x;
else if(rect.x<offset - 10)
      globalRect.x = globalRect.x - (offset - rect.x) > 0 ? globalRect.x - (offset -
rect.x) : globalRect.x;

if(rect.y>offset + 10)
      globalRect.y = globalRect.y + (rect.y - offset) < rgbaImage.rows() ?
globalRect.y + (rect.y - offset) : globalRect.y;
else if(rect.y<offset - 10)
      globalRect.y = globalRect.y - (offset - rect.y) > 0 ? globalRect.y - (offset -
rect.y) : globalRect.y;

globalRect.width = globalRect.x + (2*offset + rect.width) < rgbaImage.cols() ?
2*offset + rect.width : (rgbaImage.cols() - globalRect.x);
globalRect.height = globalRect.y + (2*offset + rect.height) < rgbaImage.rows() ?
2*offset + rect.height : (rgbaImage.rows() - globalRect.y);

mRoi = rgbaImage.submat(globalRect);

Moments m = Imgproc.moments(mBiggestContour);
centroid.x = globalRect.x + (m.get_m10() / m.get_m00());
centroid.y = globalRect.y + (m.get_m01() / m.get_m00());

contour2f.release();
}
```

P.5.21. Funkcija moveCar

```
public void moveCar(Point position, double mContureArea)
    {
      if(position.x >leftBound&& position.x <rightBound)
      {
            if( mContureArea <upperAreaB&& mContureArea >lowerAreaB&&mDataSend2)
           {
      mConnection = mBTConnection.write("S:");
      mDataSend2 = false;
      idle = true;
            }
            elseif( mContureArea >upperAreaB&& !mDataSend2)
           {
      mConnection = mBTConnection.write("S:B:"+speed+":");
      mDataSend2 = true;
      idle = false;

            }
else if( mContureArea <lowerAreaB&& !mDataSend2)
            {
      mConnection = mBTConnection.write("S:F:"+speed+":");
      mDataSend2 = true;
      idle = false;
            }
      }
      else if(position.x >rightBound)
      {
            if(mDataSend2)
            {
                  mConnection = mBTConnection.write("S:");
            }
            if(!idle)
                  mConnection = mBTConnection.write("TL:"+speedlr+":10:");
            else
                  mConnection = mBTConnection.write("RL:"+speedlr+":10:");
                  mDataSend2 = false;
      }
      else if(position.x <leftBound)
      {
            if(mDataSend2)
            {
                  mConnection = mBTConnection.write("S:");
            }

            if(!idle)
                  mConnection = mBTConnection.write("TR:"+speedlr+":10:");
            else
                  mConnection = mBTConnection.write("RR:"+speedlr+":10:");

                  mDataSend2 = false;
      }

      if(!mConnection)
        {
      activity.setResult(StaticVariables.CONNECTION_LOST);
      activity.finish();
        }
```

```
  }
```

P.5.22. Funkcija updateBound

```
public void updateBound(Rect rect)
{
	centreBound = (rect.width) >halfOfCols ? halfOfCols : (rect.width);
	centreBound = centreBound< 75 ? 75 : centreBound;

	rightBound = halfOfCols + centreBound ;
	leftBound = halfOfCols - centreBound;

}
```

ELEKTRONIČKA VERZIJA DIPLOMSKOG RADA