Mogućnosti i ograničenja grupe e-puck mobilnih robota

Rajaković, Marko

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Marko Rajaković

Zagreb, 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Dr. sc. Petar Ćurković

Student:

Marko Rajaković

Zagreb, 2016.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem svima koji su na bilo koji način doprinijeli mom obrazovanju i pisanju ovog rada.

Marko Rajaković

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **Marko Rajaković** Mat. br.: 0035168531

Naslov rada na hrvatskom jeziku: **Mogućnosti i ograničenja grupe *e-puck* mobilnih robota**

Naslov rada na engleskom jeziku: **Possibilities and limitations of a group of *e-puck* mobile robots**

Opis zadatka:

E-puck mobilni robot standardna je platforma za istraživanje ponašanja grupe mobilnih robota. Omogućuje vizualnu, zvučnu te server-klijent komunikaciju unutar grupe. Dostupan je i niz standardnih senzora koji omogućavaju djelovanje robota u složenim, unaprijed nepoznatim okolinama.

U okviru ovog zadatka potrebno je:

- Upoznati se s programskim okružjem e-puck robota.
- Upoznati se s građom e-puck robota.
- Uspostaviti komunikaciju između više robota.
- Osigurati koordinirani rad više robota temeljen na međusobnoj komunikaciji.
- Implementirati algoritam zaobilaženja prepreke.
- Testirati vizijski sustav mobilnog robota, implementirati slijeđenje zadanog objekta.

Navedene cjeline potrebno je implementirati na dostupnoj e-puck robotskoj platformi.

Zadatak zadan:
29. rujna 2016.

Rok predaje rada:
1. prosinca 2016.

Predviđeni datum obrane:
7., 8. i 9. prosinca 2016.

Zadatak zadao:

Doc. dr.sc. Petar Ćurković

Predsjednik Povjerenstva:

Prof. dr. sc. Franjo Cajner

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

D_c	prijeđeni put središta robota
D_l	prijeđeni put lijevog kotača robota
D_r	prijeđeni put desnog kotača robota
l	udaljenost između kotača
N	ukupni broj koraka
ΔN	razlika broja koraka u vremenu
r	radijus kotača
v	translacijska brzina
v_l	translacijska brzina lijevog kotača
v_r	translacijska brzina desnog kotača
x	pozicija robota po osi x
y	pozicija robota po osi y
θ	orijentacija robota
ω	kutna brzina

SAŽETAK

Cilj ovog diplomskog rada bio je ispitivanje funkcionalnosti *e-puck* mobilnog robota i njegova praktična primjena kroz implementaciju niza zadataka: kretanje, izbjegavanje prepreke, komunikacija, slijeđenje.

Obzirom da postoji relativno malo dokumentacije o ovim robotima, kako na internetu, tako i priloženo uz robota, cilj rada bio je i omogućiti budućim korisnicima brz početak rada, olakšati postavljanje programske okoline, dokumentirati i na primjerima pokazati kako koristiti aktuatore robota i očitati stanja senzora, te dati osvrt na mogućnsti i ograničenja e-puck platforme u samostalnom i grupnom radu.

U prvom dijelu rada opisano je postavljanje programske okoline i postupak stavljanja programskog koda na *e-puck* robot. Drugi dio rada detaljno objašnjava korištenje pojedinih senzora robota i njihovo programiranje. Praktični dio rada sastoji se od projektiranja i izrade tiskane pločice koja služi kao senzor za detekciju tla. Za eksperimentalnu verifikaciju izrađeno je i implementirano više algoritama kojima se testiraju zadane funkcionalnosti robota.

UVOD

Mobilni roboti su roboti koji imaju sustave za pokretanje, određivanje relativnog položaja u prostoru i prepoznavanje okoline; ukratko – roboti koji imaju sposobnost kretanja u prostoru. Nekad su bili smatrani podskupinom industrijskih robota, no danas se njihovi radni zadaci uvelike razlikuju. Sredinom osamdesetih godina prošlog stoljeća mobilna robotika postala je samostalna znanstvena disciplina, koja se, za razliku od industrijske robotike, temelji na iskustvima bioloških istraživanja građe i ponašanja živih organizama.

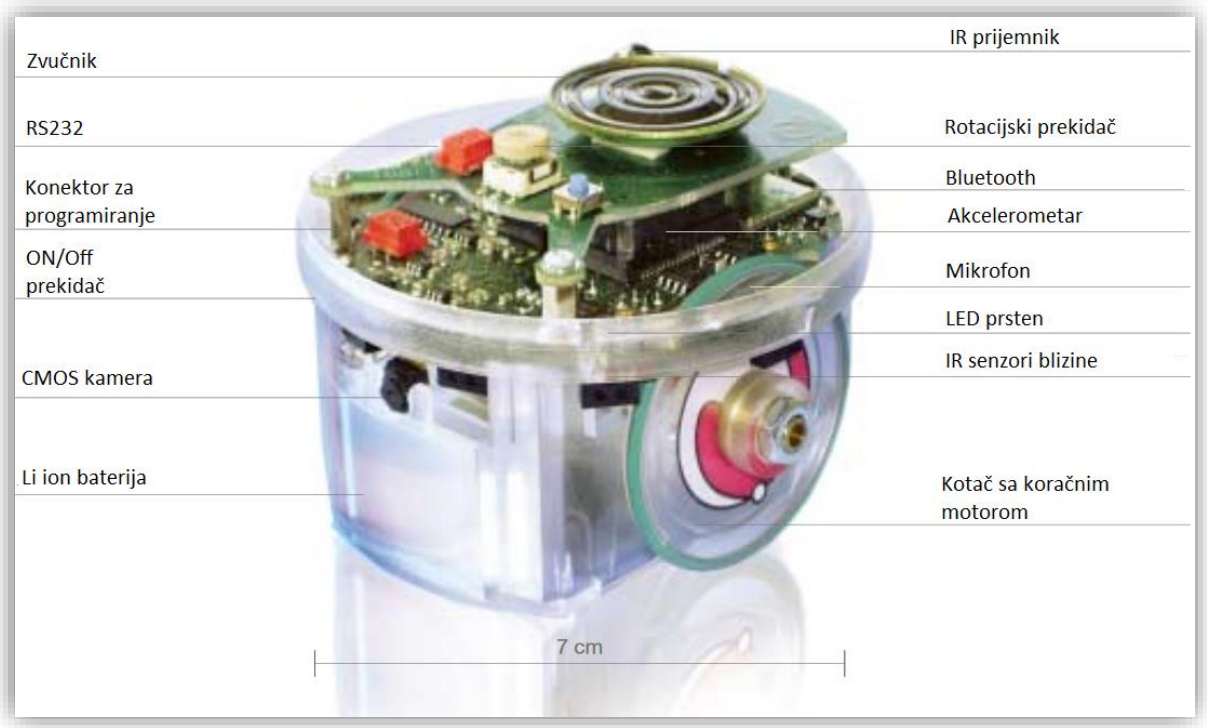
Mnogo je različitih vrsta mobilnih robota, a razlikuju se po građi sustava za pokretanje, navođenje i upravljanje. Tako postoje roboti pokretani kotačima, roboti pokretani nogama, podvodni roboti, itd. Ono što je zajedničko svim mobilnim robotima jesu 3 sustava koja moraju imati: sustav za pokretanje, sustav za navođenje i upravljački sustav.

Mobilni roboti počeli su se masovno razvijati sredinom devedesetih godina prošlog stoljeća. Masovnu proizvodnju manjih i jeftinijih robota (100€ ili više) u edukacijske svrhe danas proizvode tvrtke *K-Team, Parallax, Trossen Robotics* i *Lynxmotion*. Obzirom na razvoj web trgovine i pristupačnu cijenu, ovi su roboti danas dostupni svakom domu. Rasprostranjenost i korištenje mobilnih i industrijskih robota najveće je u razvijenim zemljama, posebice Japanu, SAD-u i Zapadnoj Europi. U zemljama s manje razvijenom industrijom vladaju predrasude o robotima te bojazni da će im isti oduzeti posao. Slično je stanje i u Hrvatskoj, gdje industrija nije modernizirana, a budžet izdvojen za kupovinu robota u edukacijske svrhe je malen. Obzirom da Fakultet strojarstva i brodogradnje posjeduje pet *e-puck* mobilnih robota (Slika 1), a dokumentacija za njihovo korištenje je gotovo nepostojeća, svrha ovog rada jest opisati funkcionalnosti i demonstrirati mogućnosti njihovog programiranja.

Tehničke specifikacije e-puck robota [1]

- Promjer: 70mm
- Visina: 50mm
- Masa: 200g
- Maksimalna brzina: 12.8cm/s
- Autonomija: oko 3h
- Mikrokontroler: dsPIC30F6014A
- RAM memorija: 8KB
- Flash memorija: 144KB
- Senzori blizine: 8 komada (TCRT 1000)
- Kamera: 640x480 (PixelPlus PO3030)
- Dva koračna motora
- Osam LED svjetala po prstenu

- Jedno LED svjetlo na prednjoj strani
- Jedno LED svjetlo na kućištu
- 3D akcelerometar
- Zvučnik
- Tri mikrofona
- Rotacijski prekidač s 16 pozicija



Slika 1. *E-puck* robot [2]

1. INSTALACIJA I PODEŠAVANJE RAZVOJNE OKOLINE

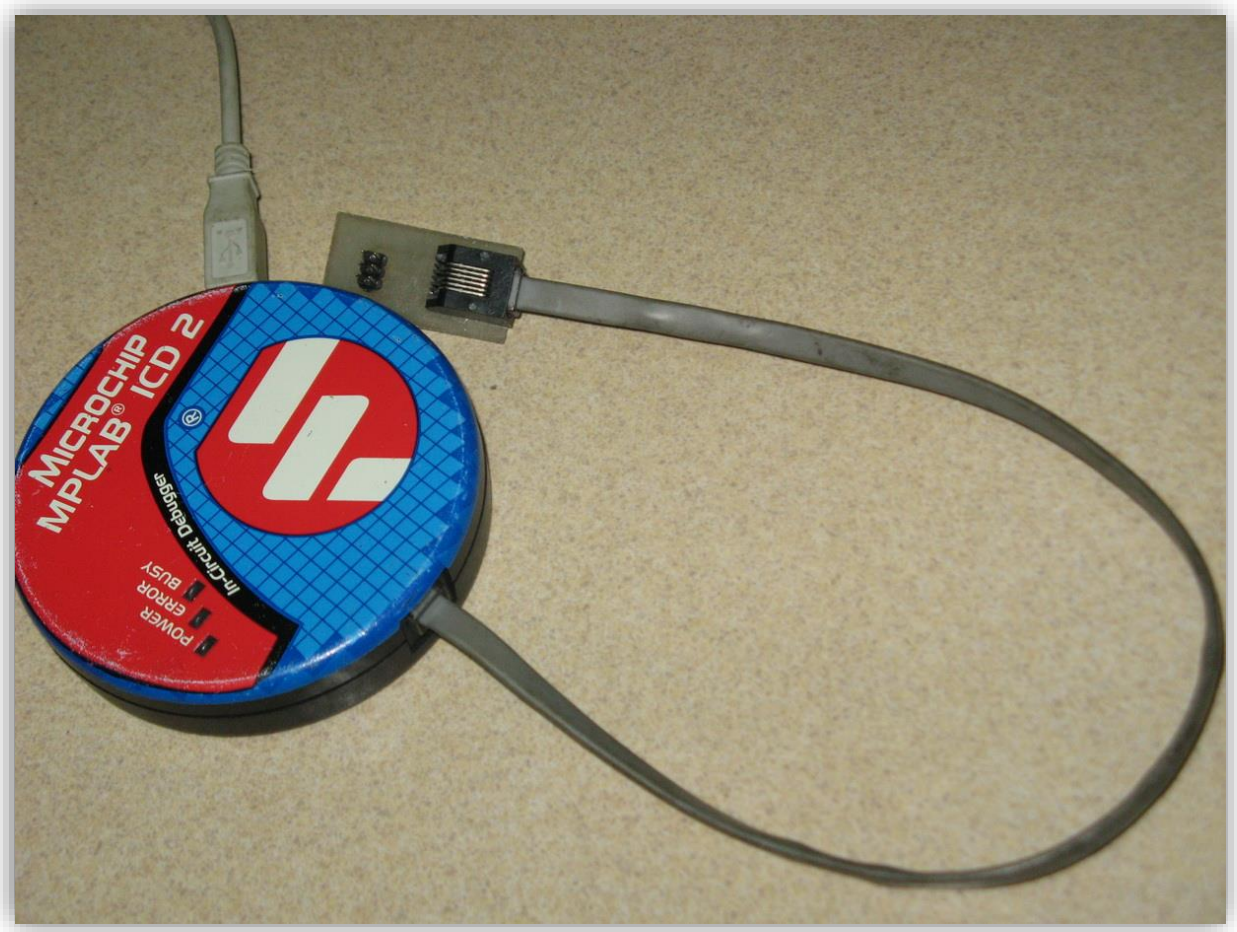
Kako bi rad na *e-puck* robotu mogao započeti, potreban je *e-puck* s odgovarajućim *bootloaderom* te treba podesiti razvojnu okolinu.

Ukoliko *e-puck* nema *bootloader* ili mu je nemoguće pristupiti zbog programske pogreške, potrebno je na njega staviti *bootloader*.

1.1. Bootloader

Iako su u sklopu ovog projekta korišteni *e-puck* roboti s ispravnim *bootloaderom* na sebi, u sljedećih nekoliko koraka biti će opisan postupak instaliranja *bootloadera* na *e-puck* robot.

Potrebno je imati *MPLAB ICD 2* (Slika 2) programator za *microchip* mikrokontrolere (*part number: DV164005*).



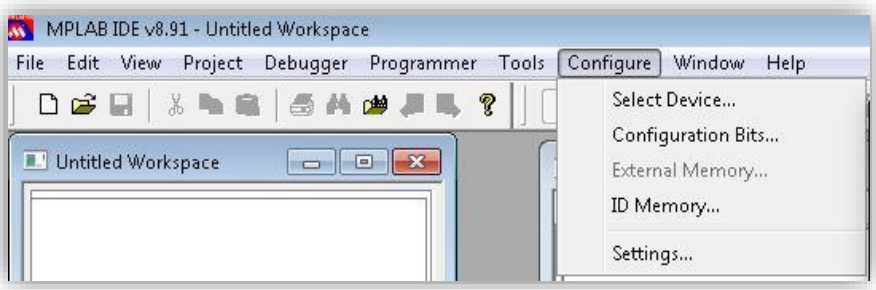
Slika 2. MPLAB ICD 2 programator [8]

Za spajanje *e-puck* robota i programatora, potreban je poseban kabel (Slika 3).

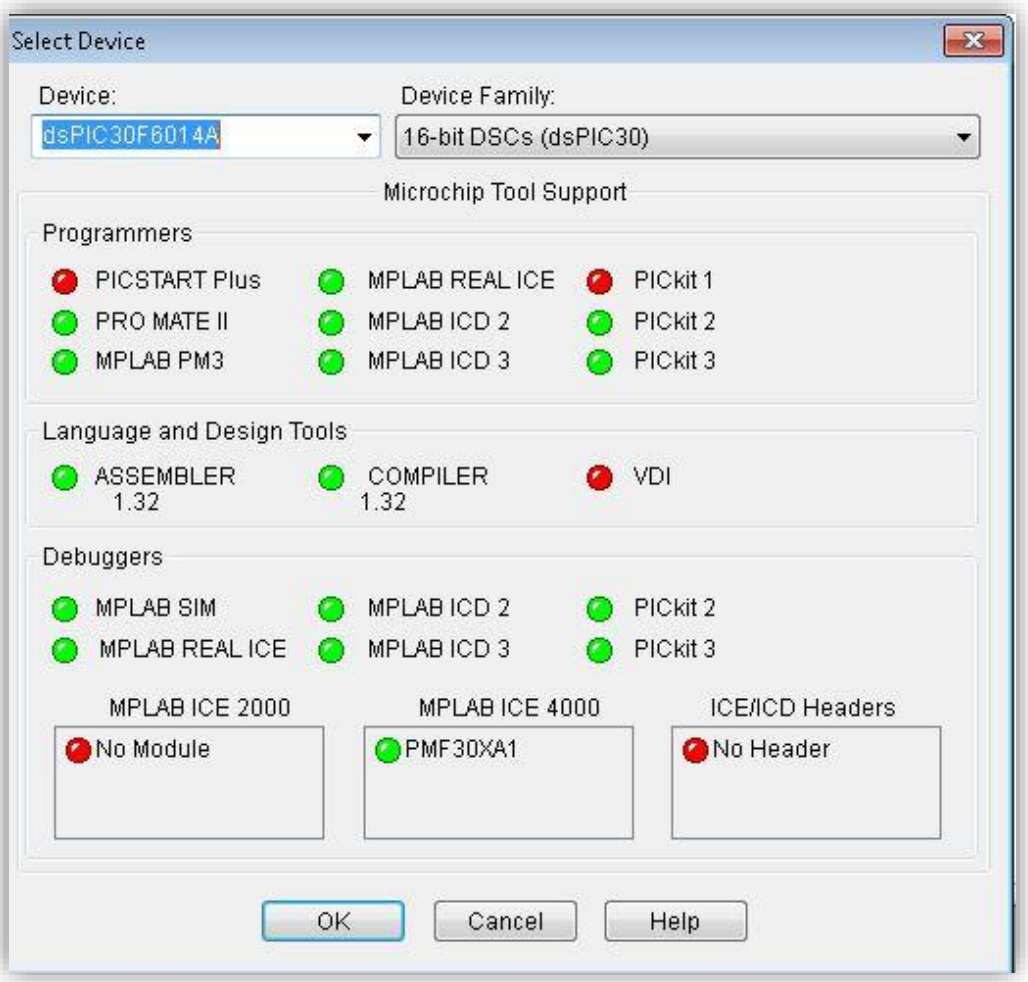
Slika 3. Kabel za programiranje [9]

Na računalo je potrebno instalirati *MPLAB IDE* (MPLAB_IDE_8_92.zip). Verzija korištena u ovom radu jest *MPLAB IDE v8.92*. Korištena je na *Windows 7 x86* operacijskom sustavu; na *Windows 10 x64* nije bilo moguće instalirati ovu verziju.

Nakon instalacije potrebno je pokrenuti *MPLAB*, odabrati *Configure –> Select Device* (Slika 4) i podesiti opcije za *e-puck* mikrokontroler (Slika 5).



Slika 4. MPLAB Select Device



Slika 5. dsPIC30F6014A

Za sljedeći korak potrebna je .hex datoteka *bootloadera* koju je moguće pronaći na www.e-puck.org/ ili skinuti direktno sa sljedećeg dropbox linka: bootloader.zip.

Kada se *bootloader* nalazi lokalno na računalu i opcije za mikrokontroler su korektno postavljene, potrebno je povezati programator s računalom i *e-puck* robotom te na *File –> Import*, odabrati .hex datoteku *bootloadera*.

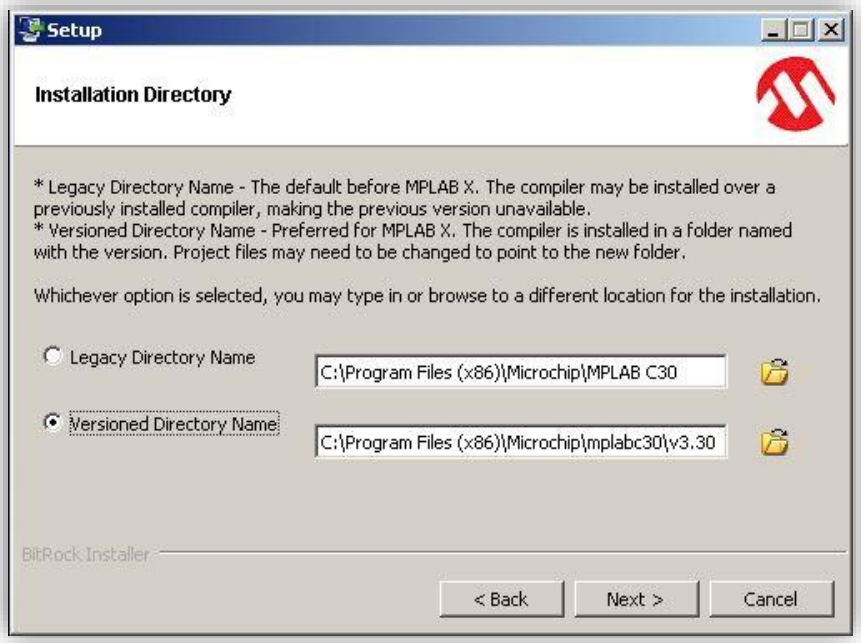
Sljedeći korak jest postavljanje *ICD2* za programator: *Programmer –> Select Programmer –> MPLAB ICD 2*, nakon čega treba napraviti konekciju: *Programmer –> Connect* i za kraj izvršiti samo programiranje: *Programmer –> Program*. Programiranje može trajati do dvije minute, a kada završi, na ekranu će biti ispisano *"...Programming succeeded <datum i vrijeme>"*.

1.2. Razvojna okolina

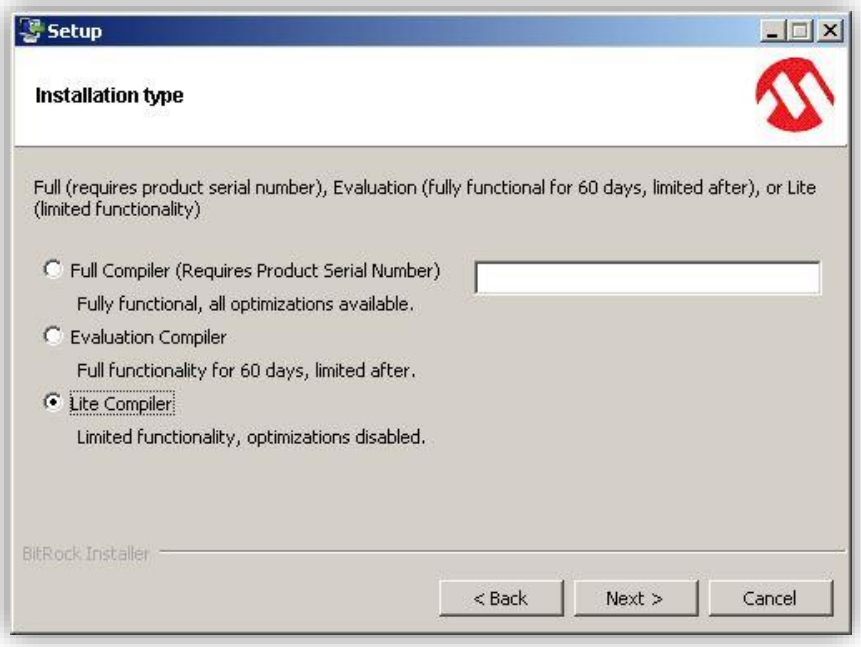
Kako *MPLAB IDE v8.92* ne radi na novijim operativnim sustavima, za programiranje se može koristiti *microchip*-ov noviji program *MPLAB X ID*E, koji će biti detaljnije objašnjen u nastavku.

MPLAB X IDE moguće je pronaći na stranici www.microchip.com, a verzija korištena u ovom radu jest *MPLAB X IDE v3.25*. Uz instalirani *MPLAB*, potrebno je instalirati *C30GNUC* kompajler: mplabc30_v3_30c_windows.exe.

Prilikom instalacije kompajlera, potrebno je odabrati opcije *Versioned Directory Name* (Slika 6) i *Lite Compiler* (jedina besplatna verzija kompajlera) (Slika 7).



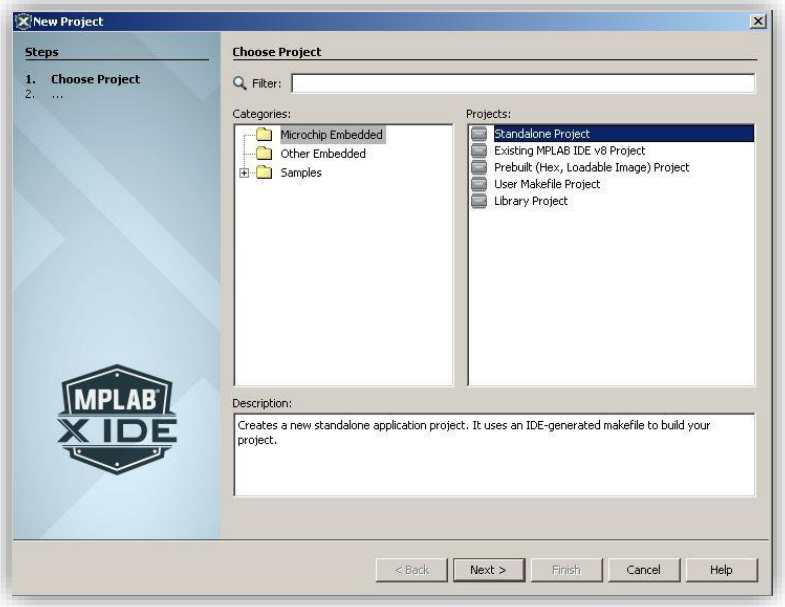
Slika 6. Instalacija kompajlera



Slika 7. Verzija kompajlera

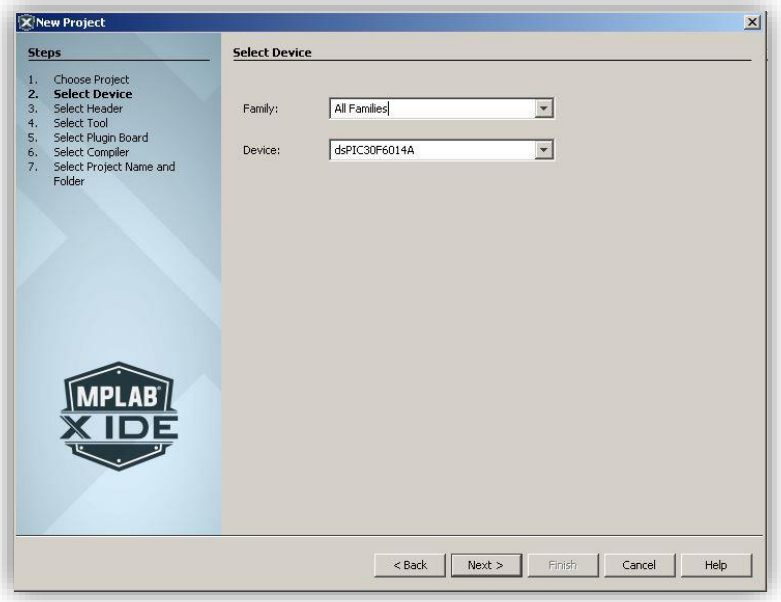
1.3. Izrada čistog projekta

Nakon instalacije slijedi kreiranje novog projekta. Potrebno je otvoriti *MPLAB X IDE* i napraviti projekt: *File –> New Project -> Standalone Project* (Slika 8).



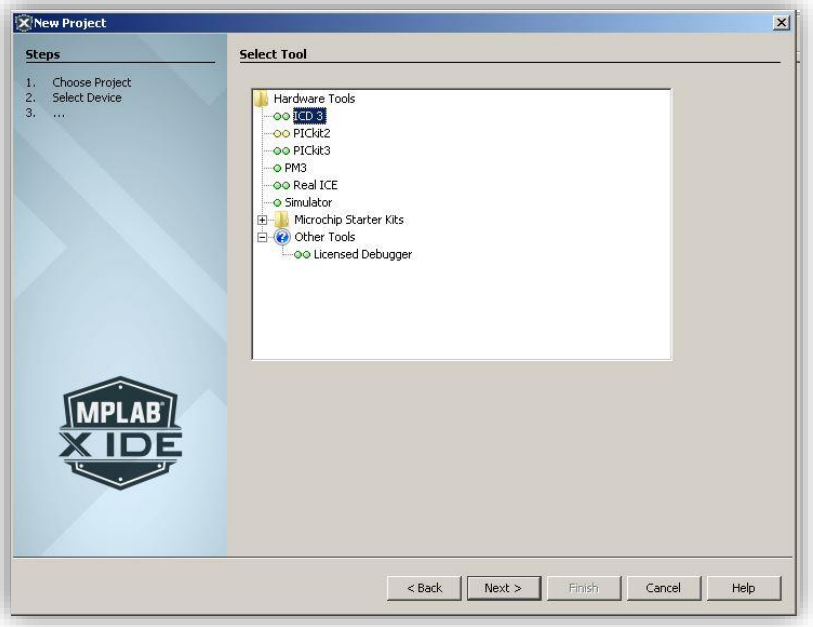
Slika 8. MPLAB Standalone Project

Sljedeći korak jest odabir odgovarajućeg mikrokontrolera: *dsPIC30F6014A* (Slika 9).



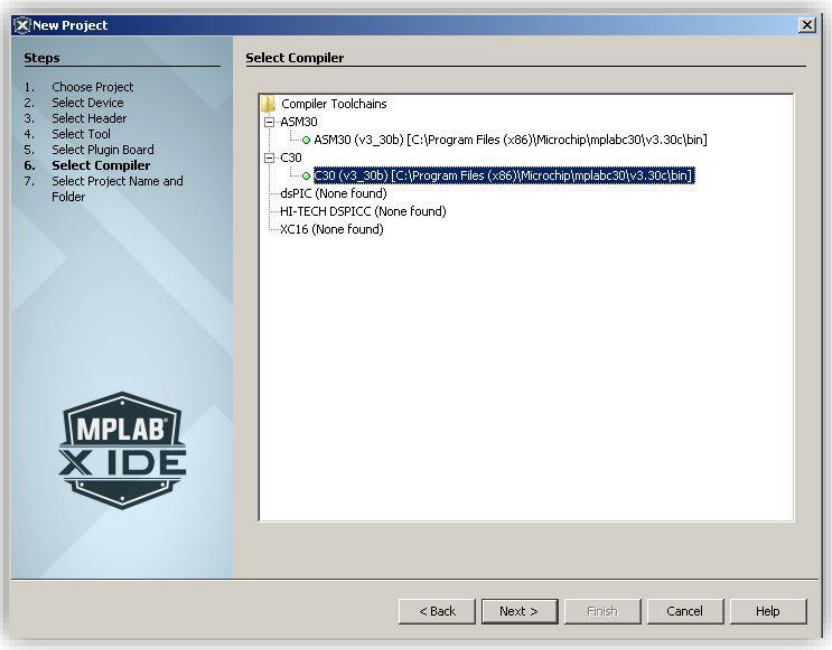
Slika 9. MPLAB odabir mikrokontrolera

Slijedi odabir programatora - ovaj korak nije bitan jer *e-puck* na sebi ima *bootloader* i bit će programiran preko *bluetooth*a (Slika 10).



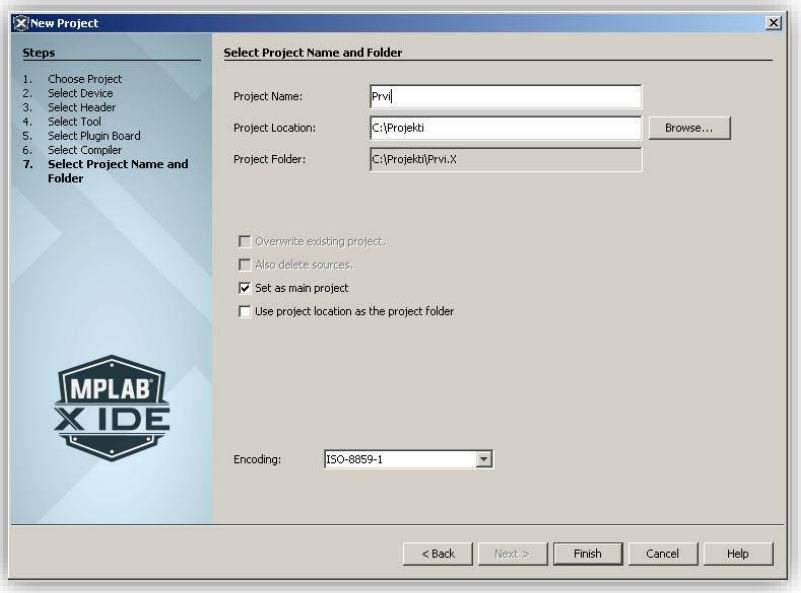
Slika 10. MPLAB odabir programatora

Nakon toga slijedi odabir kompajlera. Potrebno je odabrati prethodno instaliran kompajler *C30* (Slika 11).



Slika 11. MPLAB Odabir kompajlera

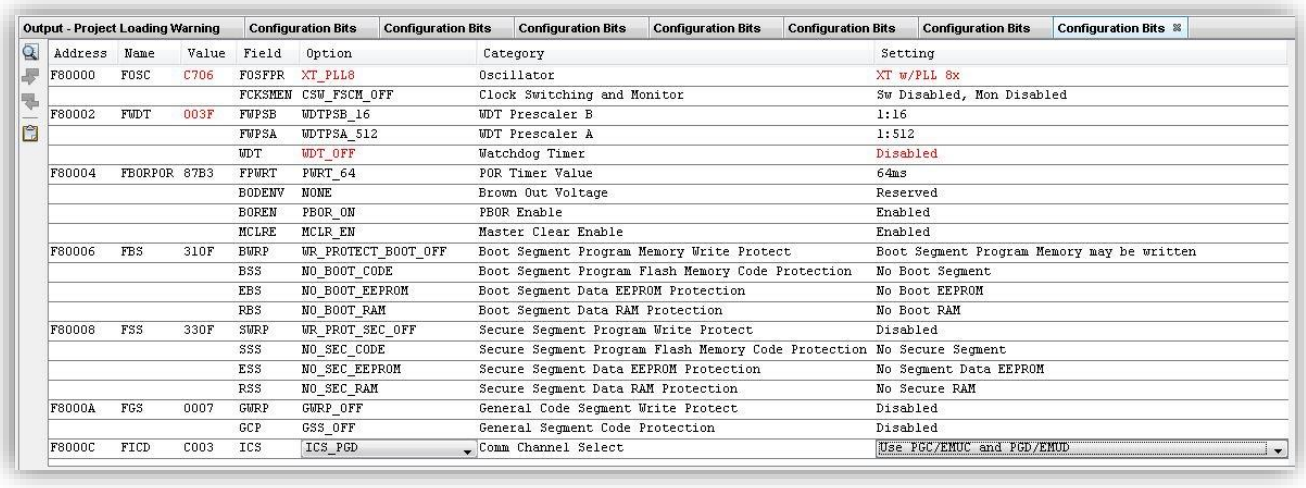
Završni korak izrade projekta jest odabir imena i mjesta pohranjivanja (Slika 12).



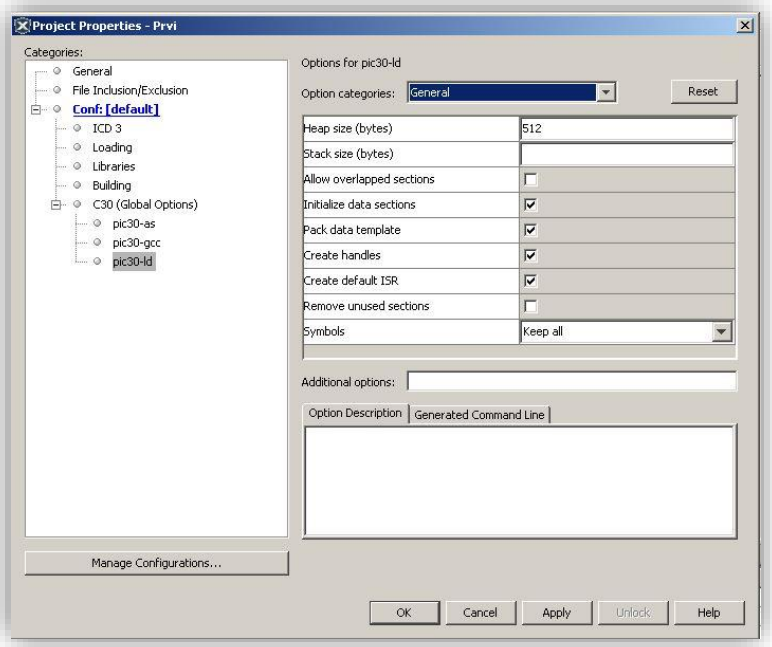
Slika 12. MPLAB odabir imena projekta

Dodatno je potrebno podesiti parametre unutar projekta:

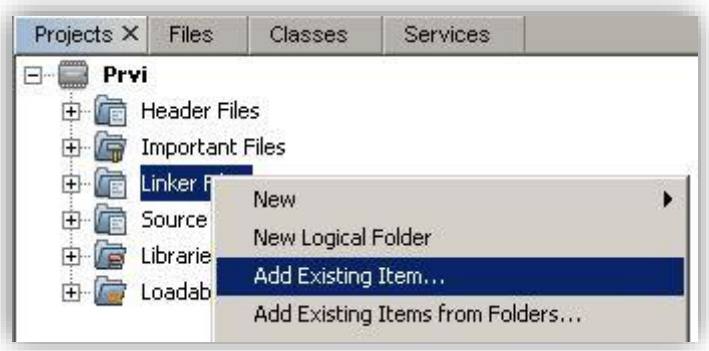
- Parametri mikrokontrolera: *Run -> Set Configuration Bits*. Potrebno je promijeniti opciju *"Oscilator* u *XT w/PLL 8x"* i W*atchdog Timer* u *"Disabled"* (Slika 13).
- Kompajler *heap size*: *Run –> Set Project Configuration -> Customize* potrebno je podesiti opciju *Heap size* na *512 bajta* (Slika 14).
- *Linker* datoteka*:* pod opcijama projekta, desnim klikom na *Linker Files* dobiva se izbornika na kojem je potrebno odabrati opciju *Add Existing Ite* (Slika 15) te u slijedećem prozoru navesti put do datoteke *p30F6014A.gld* (Slika 16).



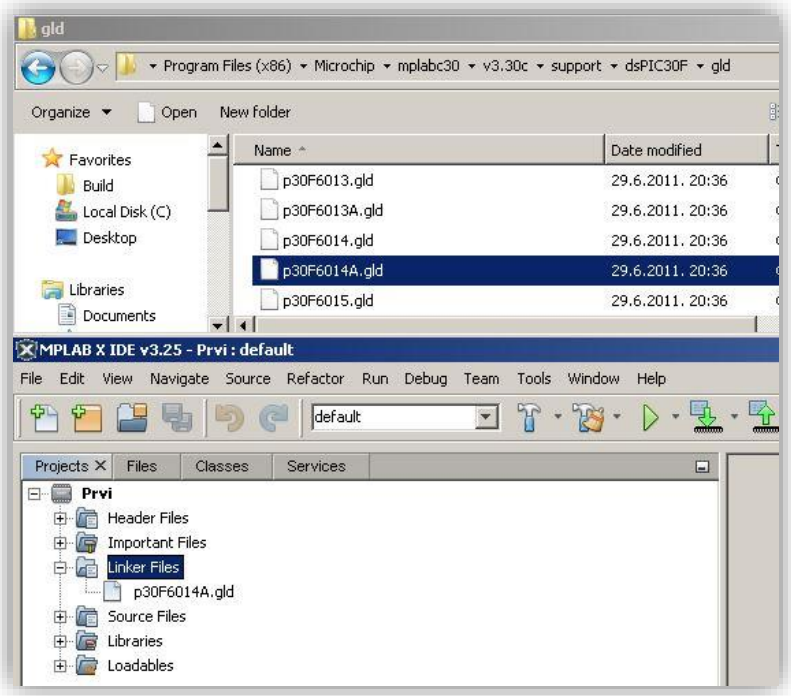
Slika 13. *Configuration Bits*



Slika 14. *Heap size*



Slika 15. Dodavanje *Linker* datoteke



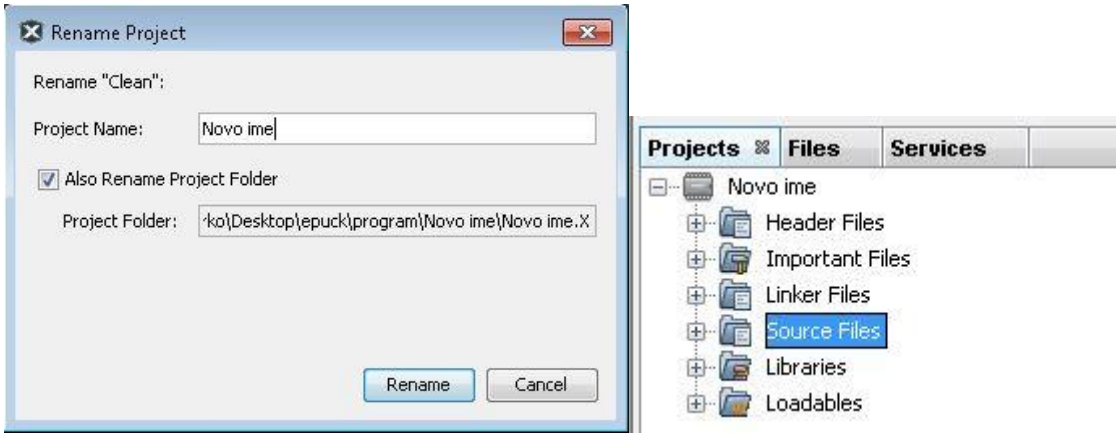
Slika 16. Dodana *Linker* datoteka

Završni korak jest kreiranje datoteke za pisanje vlastitog C programa za *e-puck*: desnim klikom na *Source Files* dobiva se padajući izbornik; potrebno je odabrati opciju *New –> C Main File -> <ime nove datoteke>* i pisanje programa može početi.

1.4. Brza izrada projekta

Obzirom da je upravo opisani postupak kreiranja novog projekta mukotrpan, napravljen je projekt kojeg treba samo kopirati i preimenovati, kako bi se preskočilo postavljanje svih dodatnih opcija. Potrebno je skinuti i otpakirati datoteku epuck.zip. Struktura mapa mora ostati ista; u mapi *program* treba napraviti kopiju mape *Clean* te ju preimenovati u željeno ime.

Potom je potrebno otvoriti *MPLAB X IDE* i kroz izbornik *File –> Open Project* otvoriti projekt "*Clean*" koji se nalazi unutar upravo kopirane mape. Za preimenovanje samog projekta, potrebno je odabrati opciju *Rename* koja se pojavljuje unutar padajućeg izbornika nakon desnog klika na ime projekta (Slika 17).



Slika 17. Preimenovanje projekta i struktura preimenovanog projekta

Unutar mape *Source Files*, već postoji datoteka *main.c* s osnovnim naredbama za početak programiranja.

Kada je program napisan, potrebno ga je kompajlirati: *Run –> Build Project -> <ime projekta>*. Ako sve prođe kako treba, ispisati će se *BUILD SUCCESSFUL* i put do novokreirane .hex datoteke. Primjer ispisa:

Loading code from C:/Users/Marko/Desktop/epuck/program/Novo ime/Novo ime.X/dist/default/production/Novo_ime.X.production.hex

Novokreirana .hex datoteka (*Novo_ime.X.production.hex* iz primjera) biti će potrebna za prebacivanje programa na *e-puck*.

1.5. Prijenos programa na *e-puck* robot

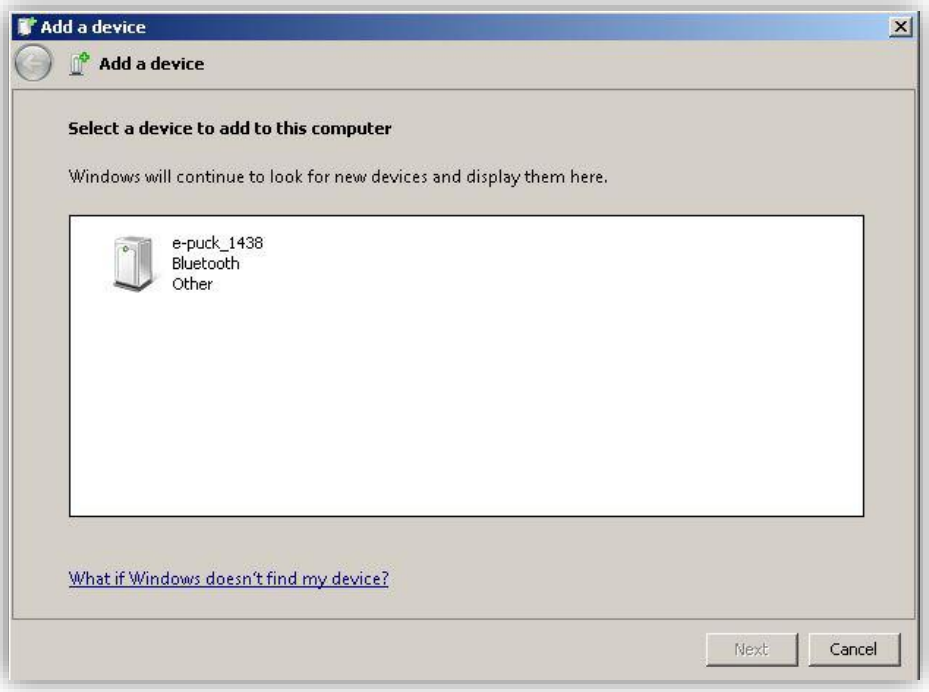
Zbog neposjedovanja odgovarajućeg kabela za spajanje *e-puck* robota i računala, u sklopu ovog projekta, roboti su programirani isključivo putem *bluetootha*. U poglavlju koje slijedi biti će opisan postupak programiranja robota putem *bluetootha.*

Za početak, potrebno je u *Windowsima* otvoriti *Device Manager* i pogledati postoji li sekcija *"Ports (COM & LPT)"*. Ako sekcija ne postoji, to znači da trenutno na računalu nije otvoren niti jedan port (Slika 18).

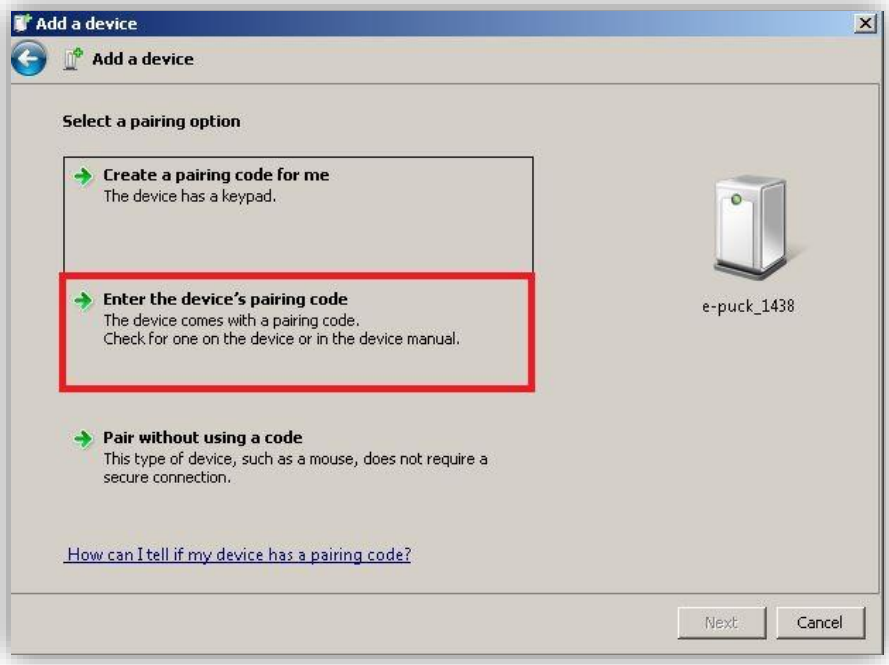
Slika 18. *Device manager* – nema sekcije *Ports (COM & LPT)*

Potrebno je upaliti *e-puck* (ima mali prekidač ispod prstena), a na računalu se pozicionirati u: *Control Panel/Hardware and Sound* i odabrati opciju *Add a Bluetooth device.*

U novom prozoru odabrati *e-puck_1438* te opciju *Next*. U ovom slučaju, brojka 1438 označava *bluetooth* kod za spajanje s odgovarajućim *e-puck* robotom. Slikama (Slika 19, Slika 20 i Slika 21) je objašnjen postupak spajanja robota s računalom.



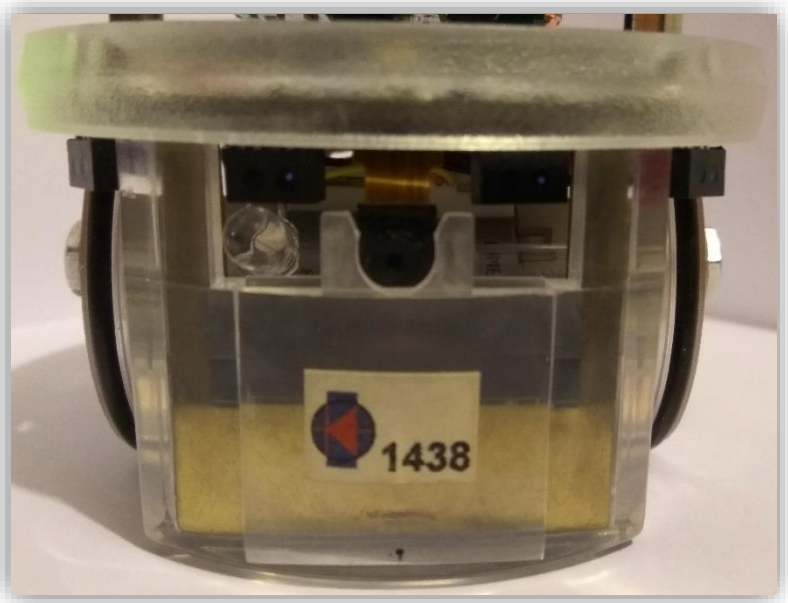
Slika 19. Windowsi su detektirali novi *e-puck* uređaj



Slika 20. Povezivanje *e-puck* robota s računalom

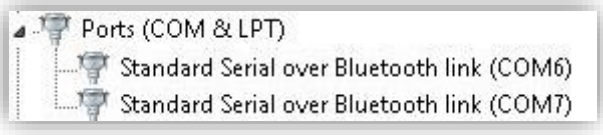
Slika 21. Upisivanje odgovarajućeg pina

Osim u imenu, *bluetooth* kod svakog robota piše na njegovoj prednjoj strani (Slika 22).



Slika 22. Bluetooth kod za spajanje s računalom

Nakon uspješnog spajanja robota s računalom, u *Device Manageru* će se otvoriti sekcija *"Ports (COM & LPT)"* s dva nova porta, kao što je prikazano na slici (Slika 23).



Slika 23. Novootvoreni portovi za *e-puck* robote

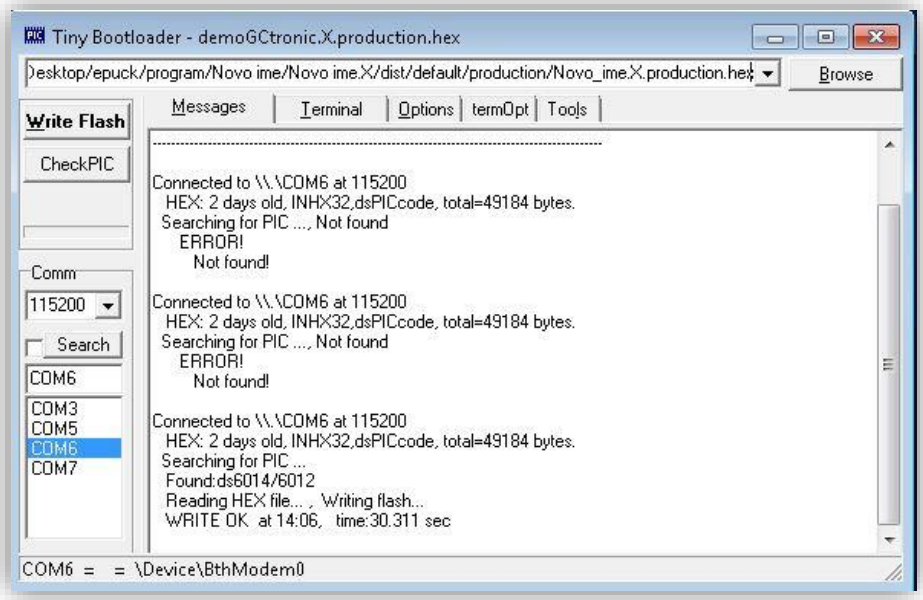
Za sljedeći korak potreban je program *Tiny Bootloader*, a preporučeno je korištenje na *Windows 7* operacijskom sustavu (u sklopu ovog projekta, pokazalo se da na *Windows 10* operacijskom sustavu ovaj program ne radi najbolje). Program je moguće skinuti na linku: TinyBld_1_10_6_pc_beta.zip.

Nakon instalacije programa, potrebno ga je pokrenuti, te odabrati odgovarajuću .hex datoteku (stvorenu putem *MPLAB X IDE* programa u prethodnom koraku) klikom na opciju *Browse*.

Potom je potrebno odabrati komunikacijski port sa slike (Slika 23) – u pravilu je odgovarajući port onaj s manjim brojem, u konkretnom slučaju, to bi značio port *COM6.* Slijedi zapisivanje .hex datoteke na *e-puck*: odabrati opciju *Write Flash* i čekati dok se na robotu ne upali narančasto LED svjetlo, a potom stisnuti plavu tipku na robotu (Slika 24).

Slika 24. Narančasti LED indikator i reset tipka

Kada programiranje završi, u programu će pisati *WRITE OK* (Slika 25).



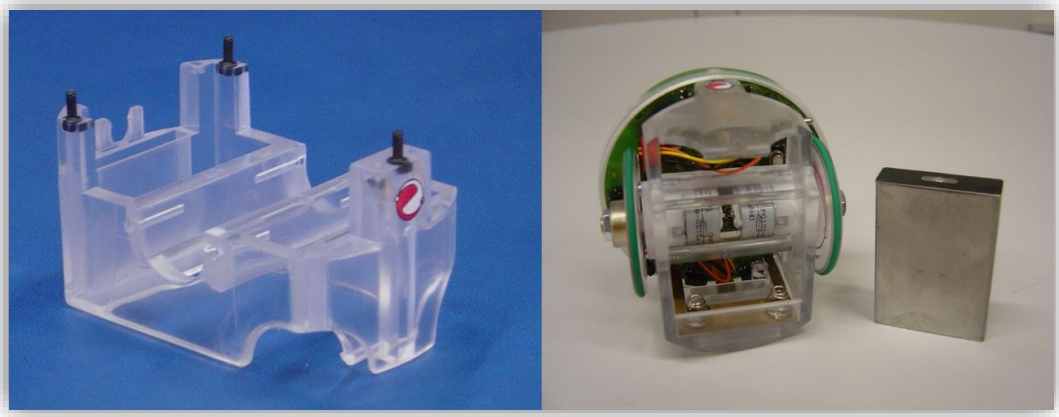
Slika 25. Završeno programiranje

Napomena: često se dogodi da programiranje ne uspije od prve te treba ponoviti postupak nekoliko puta.

Također, prilikom programiranja, dobra je praksa ostaviti jedan položaj na selektoru gdje *e-puck* ne radi ništa, čime je olakšano stavljanje novog programa na njega. Ova mjera je potrebna zato što kada je *e-puck* zauzet izvršavanjem programa, teško je uhvatiti trenutak u kojem je slobodan za zapisivanje nove .hex datoteke na njega te će biti potrebno ponoviti postupak veliki broj puta. Primjer takvih programa (s položajem selektora gdje robot ne radi ništa) moguće je pronaći u dodatku u isječcima koda koji su priloženi ovom diplomskom radu.

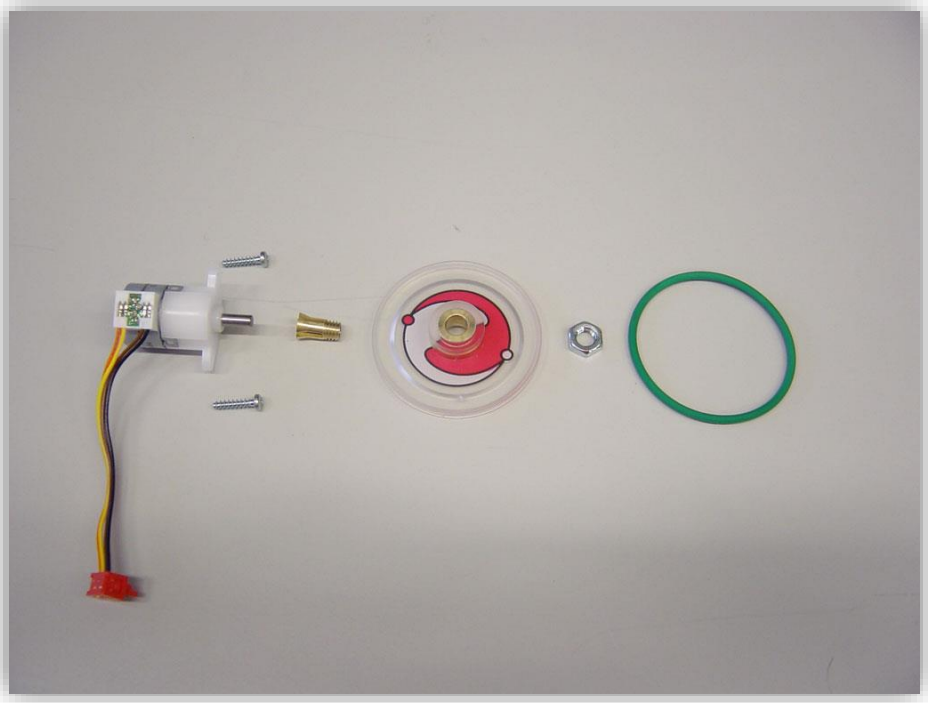
2. MEHANIKA *E-PUCK* ROBOTA

Konstrukcija *e-puck* mobilnog robota sastoji se od jednog dijela, promjera 70mm koji služi kao kućište ostalim komponentama (Slika 26).

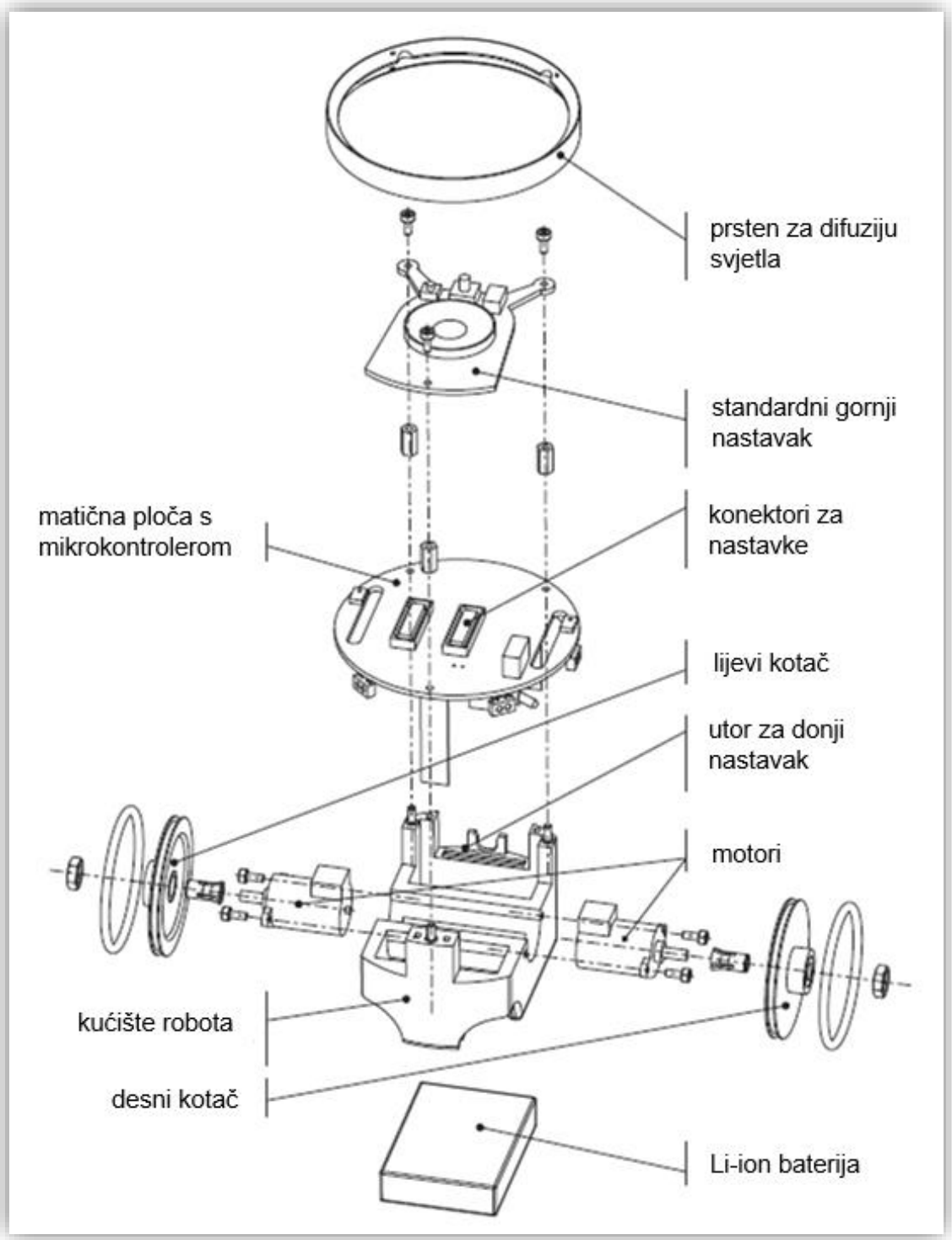


Slika 26. Kućište sa i bez komponenti [3]

Koračni motori robota imaju 20 koraka po okretaju i reduktor 50:1. Maksimalna brzina motora je 1000 koraka/s, što s reduktorom daje 1 okretaj/s. Na vratilo motora dolazi kotač dimenzije 41mm, a međuosovinski razmak između kotača je 53mm [3].



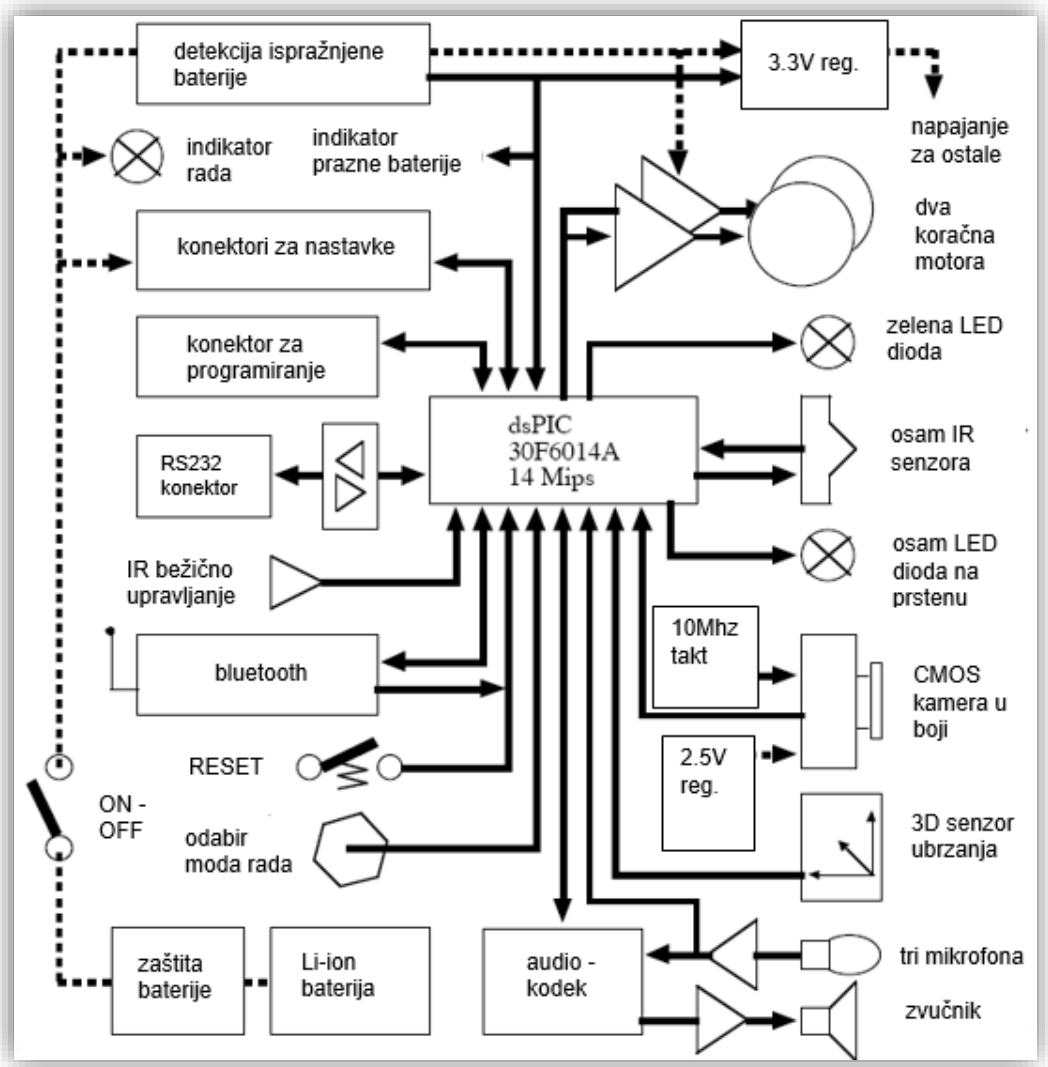
Slika 27. Koračni otor s kotačem [3]



Slika 28. Struktura *e-puck* robota [4]

3. ELEKTRONIKA I SENZORI *E*-PUCK ROBOTA

Elektronika *e-puck* robota bazirana je na 16-bitnom DSP *(digital signal processor*) mikrokontroleru. Proizvođač mikrokontrolera je *Microchip*, model *dsPIC310F3014A*. Na slici koja slijedi (Slika 29) prikazana je elektronička struktura robota.



Slika 29. Elektronička struktura *e-puck* robota [4]

3.1. Senzor ubrzanja

Senzor ubrzanja može mjeriti ubrzanje *e-puck* robota kao 3D vektor, čime omogućuje detekciju sudara, pada ili stanja mirovanja. Model senzora ubrzanja je *MMA7260*, a podešen je u modu rada 2g, čime ostvaruje maksimalnu osjetljivost od $19{,}62 m/s^2$ [5].

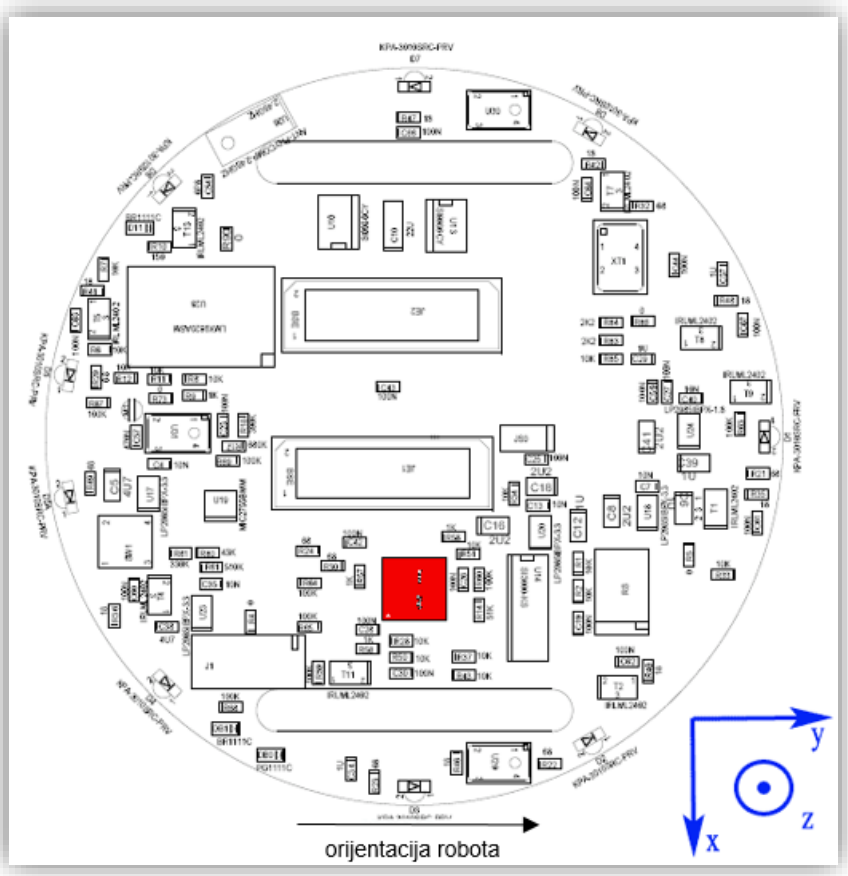
Očitanje senzora ubrzanja za svaku će os dati očitanje između 0 i 4095, što je vrijednost 12-bitnog ADC (*analog-to-digital converter*) pretvornika. Preračunavanje vrijednosti senzora u konkretnu jedinicu (m/s^2) treba odraditi ručno:

$$\frac{V_{adc}}{V} = \frac{adc}{N} \quad (1)$$

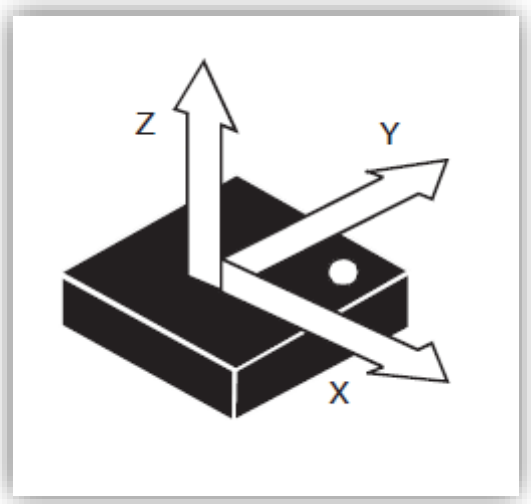
$$V_{adc} = \frac{adc}{N} * V \quad (2)$$

- $Vadc$ – traženi napon senzora
- V – referentni napon mikrokontrolera ($3300mV$)
- adc – očitanje ADC pretvornika (vrijednost između 0 i 4095)
- N – rezolucija ADC pretvornika – (4095)

Za vrijednosti $Vadc$ veće od $1650mV$, ubrzanje je pozitivno, a za vrijednosti manje od $1650mV$, ubrzanje je negativno. Promjena vrijednosti $Vadc$ za $\sim 600mV$ je promjena ubrzanja od $1g$, odnosno $9{,}81 m/s^2$ [6].



Slika 30. Položaj i orijentacija senzora ubrzanja u odnosu na robot [7]



Slika 31. Orijentacija koordinatnog sustava senzora ubrzanja u odnosu na sam senzor

3.1.1. Kod za čitanje vrijednosti senzora ubrzanja

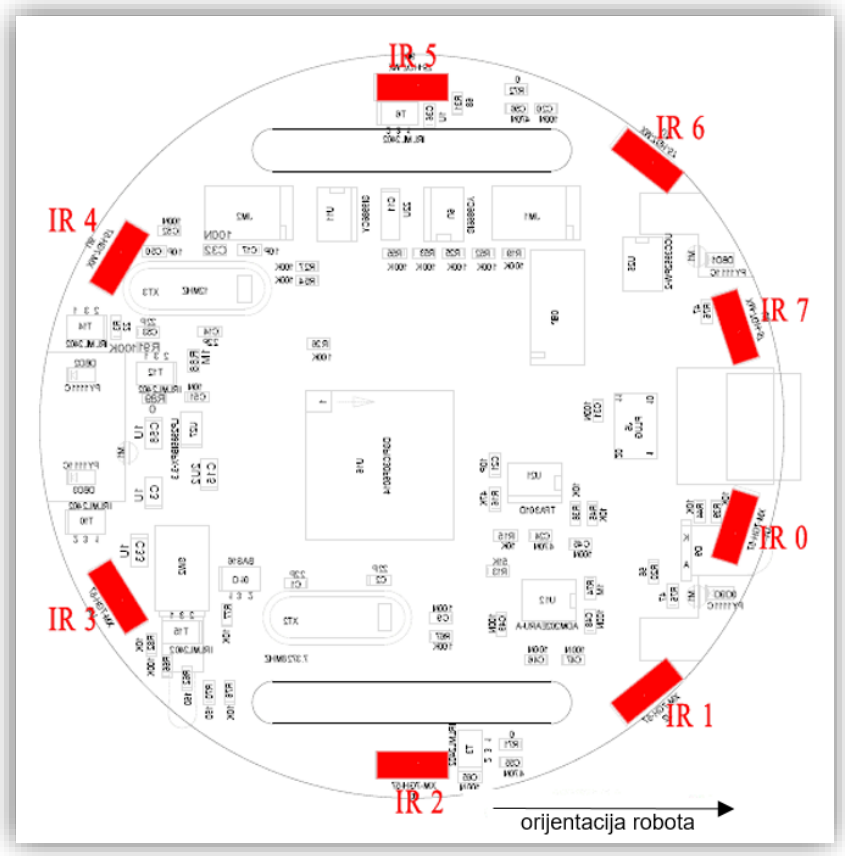
```
#include "../library/motor_led/e_epuck_ports.h"
#include "../library/motor_led/e_init_port.h"
#include "p30F6014A.h"
#include "../library/a_d/advance_ad_scan/e_acc.h"
#include "../library/a_d/advance_ad_scan/e_ad_conv.h"

main()
{
	e_init_port();
	int selector = getselector();
	switch (selector) {
		case 0: {
			e_init_ad_scan(ALL_ADC);

			int os_x, os_y, os_z;
			os_x = e_get_acc(0);
			os_y = e_get_acc(1);
			os_z = e_get_acc(2);
			break;
		}
		case 15: {
			unsigned long i = 0;
			for (i = 0; i < 2000000; i++) {
				asm("nop");
			}
			break;
		}
	}
	while (1);
}
```

Dio koda koji je označen u svjetlo sivoj boji nije bitan za očitanje senzora ubrzanja, ali je ponekad nužan za ponovnu mogućnost programiranja *e-puck* robota preko *bluetootha*.

Vrijednosti očitanja ADC pretvornika za pojedinu os nalaze se u varijablama *os_x*, *os_y*, *os_z*. Kako bi se vrijednosti pretvorile u napon, potrebno je koristiti izraz (2), pozitivan smjer je označen na slikama (Slika 30) i (Slika 31).

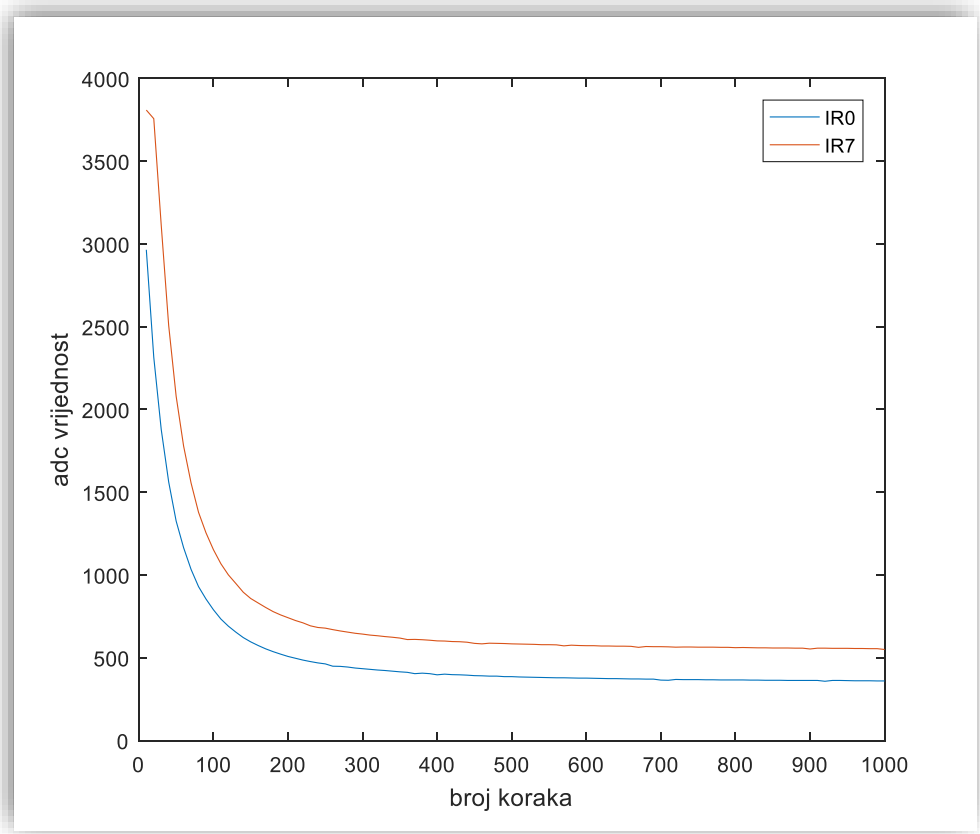
3.2. Infracrveni senzori blizine (*infrared sensors*)

E-puck ima osam infracrvenih senzora blizine (IR senzora) koji su postavljeni oko tijela robota (Slika 32). Osim za mjerenje udaljenosti od predmeta, ovi senzori mogu se koristiti i za mjerenje intenziteta svjetla u okolini. Očitanje IR senzora jednako je očitanju senzora ubrzanja: vrijednost ADC pretvornika, između 0 i 4095.

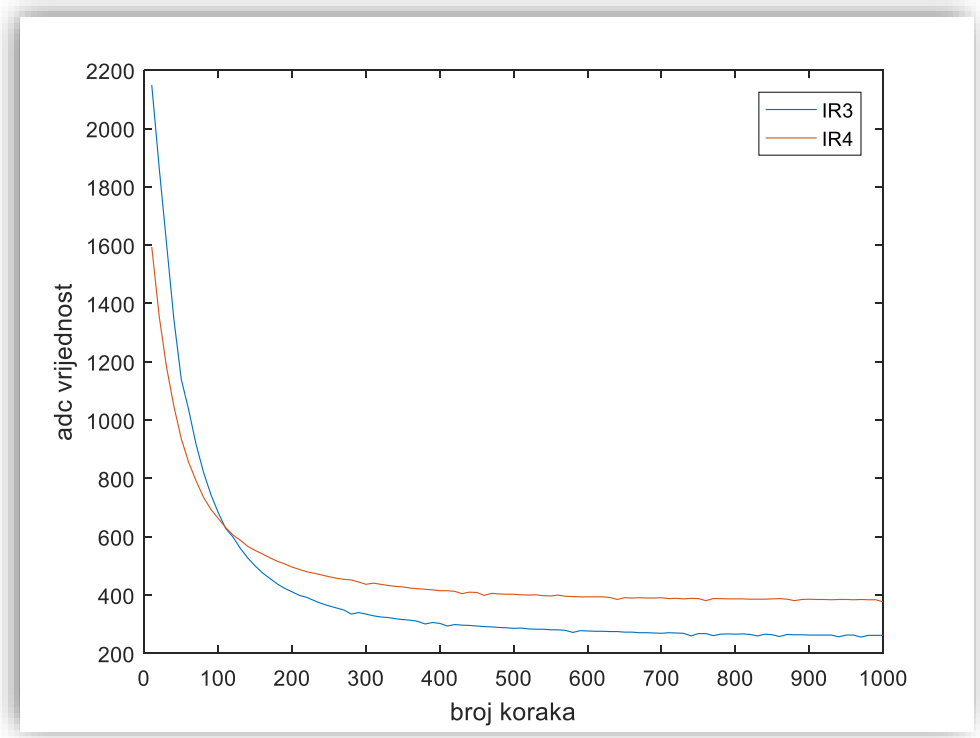


Slika 32. Položaji IR senzora [3]

Slika 33 prikazuje vrijednost očitanja ADC pretvornika u odnosu na udaljenost robota od zida. Na apscisi koordinatnog sustava je broj koraka koračnog motora *e-puck* robota ($1000 koraka = 12{,}8 cm$).



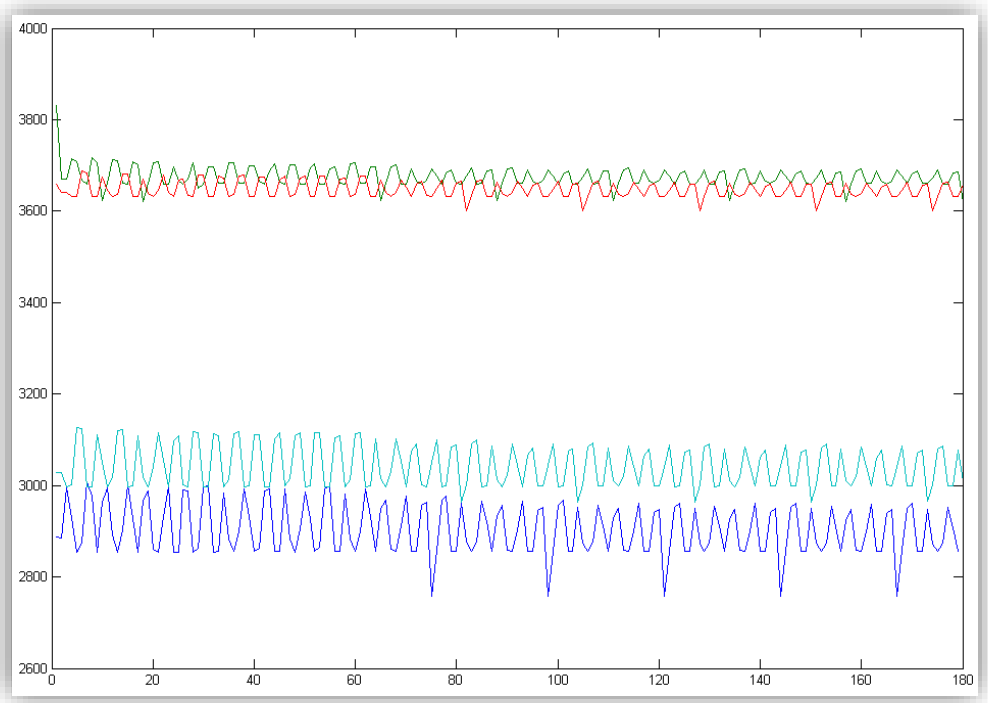
Slika 33. Očitanje senzora IR0 i IR7 kada se robot udaljava od zida



Slika 34. Očitanje senzora IR3 i IR4 kada se robot udaljava od zida

Na slici (Slika 32) se može vidjeti da su senzori IR0 i IR7 te IR3 i IR4 u paru. Mjerenje točne udaljenosti od prepreke je otežano jer svaki senzor daje različite vrijednosti za jednaku udaljenost, kao što se može vidjeti na slikama (Slika 33) i (Slika 34). Za točne vrijednosti potrebno je napraviti samostalno umjeravanje svakog senzora.

Slika 35 prikazuje šum IR0, IR1, IR6 i IR7 senzora uzrokovan koračnim motorima robota (1000 koraka/s). Šum se pojačava pražnjenjem baterije [3].



Slika 35. Šum na senzorima [3]

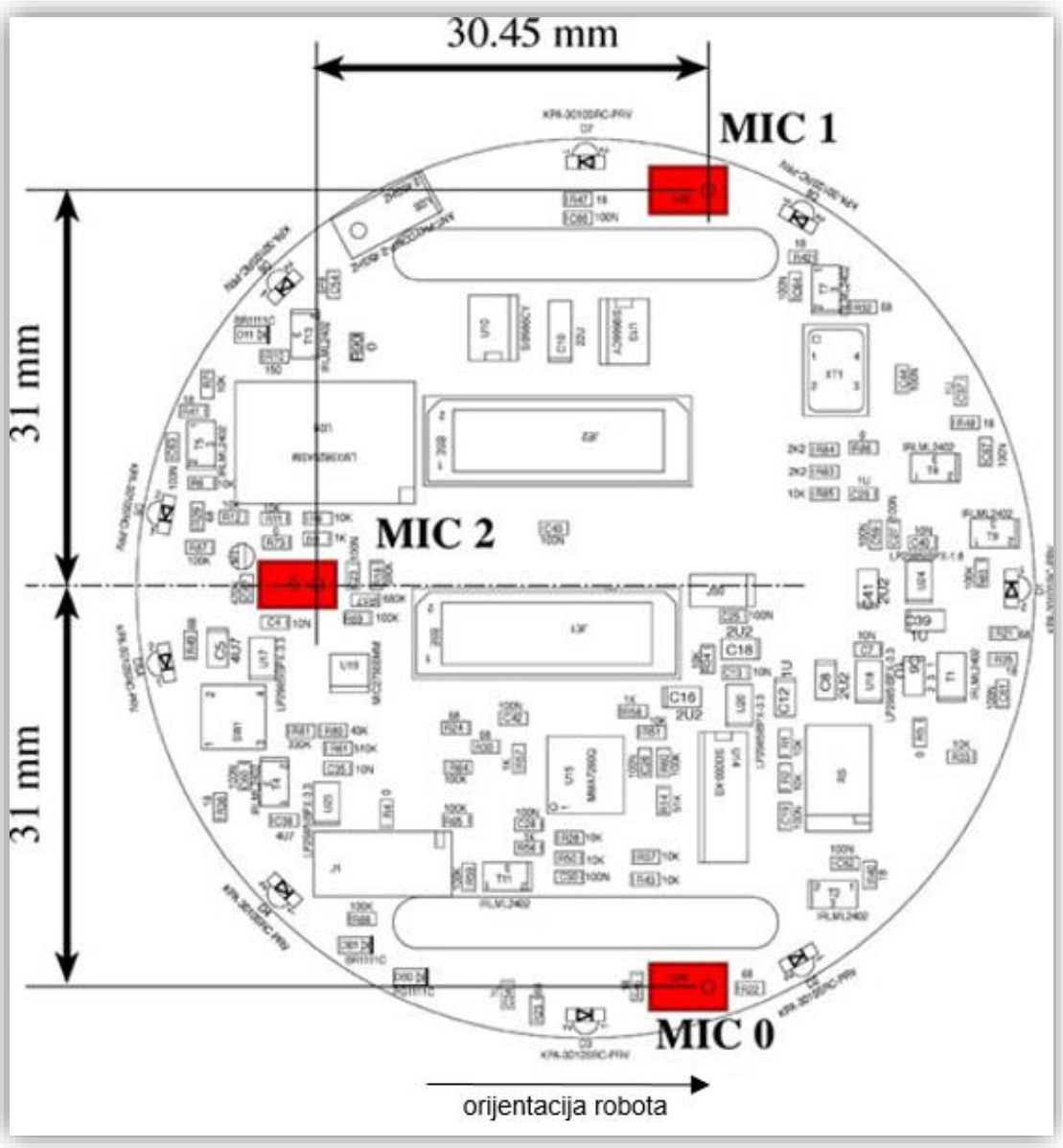
3.2.1. Kod za čitanje vrijednosti senzora blizine

```
#include "../library/motor_led/e_epuck_ports.h"
#include "../library/motor_led/e_init_port.h"
#include "../library/utility/utility.h"
#include "p30F6014A.h"
#include "../library/a_d/advance_ad_scan/e_ad_conv.h"
#include "../library/a_d/advance_ad_scan/e_prox.h"

main()
{
	e_init_port();
	int selector = getselector();
	switch (selector) {
	case 0: {
		e_init_ad_scan(ALL_ADC);

		int IR_0, IR_1, IR_2, IR_3, IR_4, IR_5, IR_6, IR_7;
		IR_0 = e_get_prox(0);
		IR_1 = e_get_prox(1);
		IR_2 = e_get_prox(2);
		IR_3 = e_get_prox(3);
		IR_4 = e_get_prox(4);
		IR_5 = e_get_prox(5);
		IR_6 = e_get_prox(6);
		IR_7 = e_get_prox(7);
		break;
	}
	case 15: {
		unsigned long i = 0;
		for (i = 0; i < 2000000; i++) {
			asm("nop");
		}
		break;
	}
	}
	while (1);
}
```

Čitanje vrijednosti senzora blizine istaknuto je tamnom bojom. Ime senzora IR_0 odgovara imenu senzora IR0 sa slike (Slika 32), a vrijednosti se kreću od *0* do *4095*.

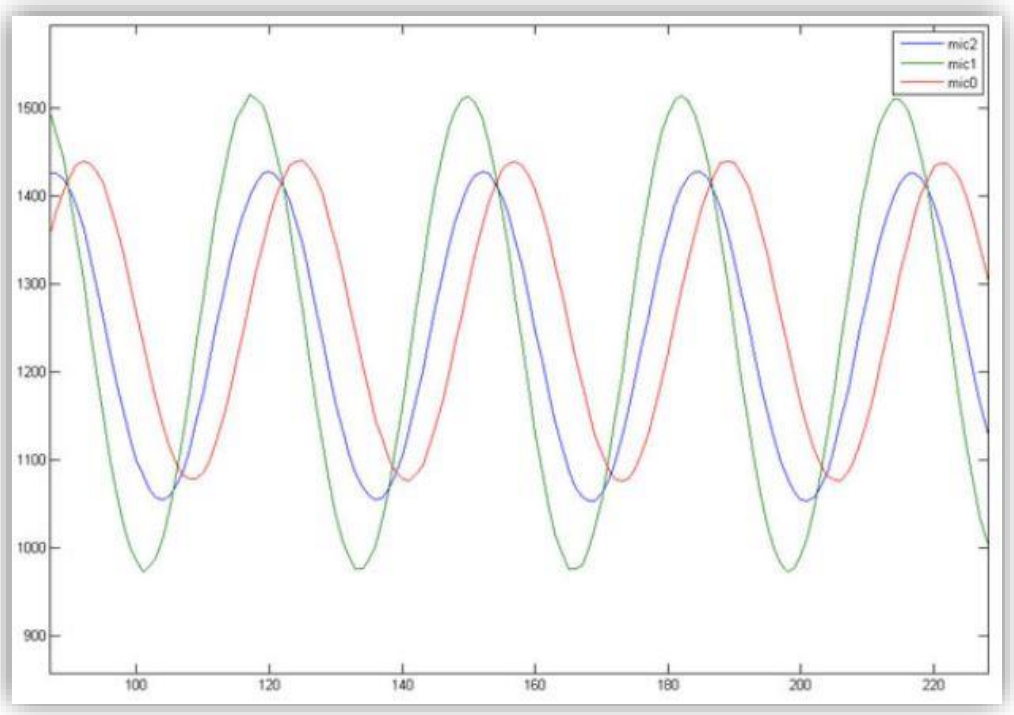
3.3. Mikrofoni

E-puck robot ima tri mikrofona (Slika 36) koja su spojena na ADC pretvornik. Maksimalna brzina pretvorbe ADC pretvornika je $100kHZ$, što za jedan mikrofon daje maksimalnu brzinu uzorkovanja od $100kHz$. Za tri mikrofona ta brzina se ravnomjerno dijeli, čime iznosi $33kHz$ (uzevši u obzir da ADC pretvorbu ne koristimo niti za jedan drugi senzor u tom trenutku). Dobivene vrijednosti kreću se od *0* do *4095*, gdje *4095* označava maksimalnu razinu zvuka, a *0* označava da se nije detektirao nikakav zvuk.

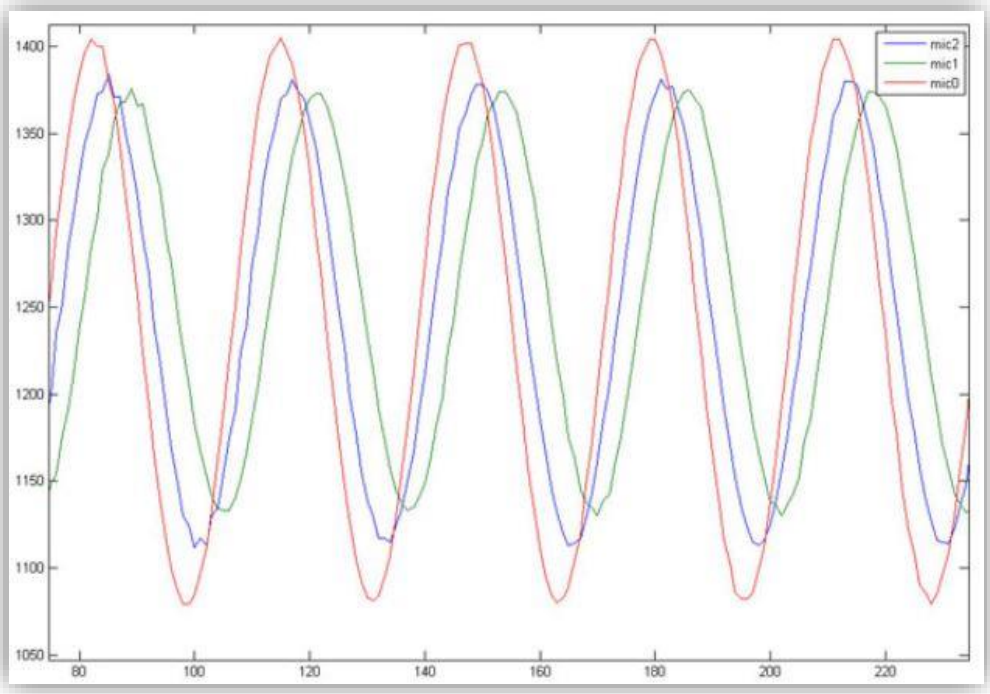


Slika 36. Položaj mikrofona na robotu [3]

Slike koje slijede prikazuju uzorkovanje zvuka od 1kHz koje dolazi s lijeve (Slika 37) i desne (Slika 38) strane *e-puck* robota.



Slika 37. Zvuk od 1kHz koji dolazi s lijeve strane *e-puck* robota [3]



Slika 38. Zvuk od 1kHz koji dolazi s desne strane *e-puck* robota [3]

3.3.1. Kod za čitanje vrijednosti mikrofona

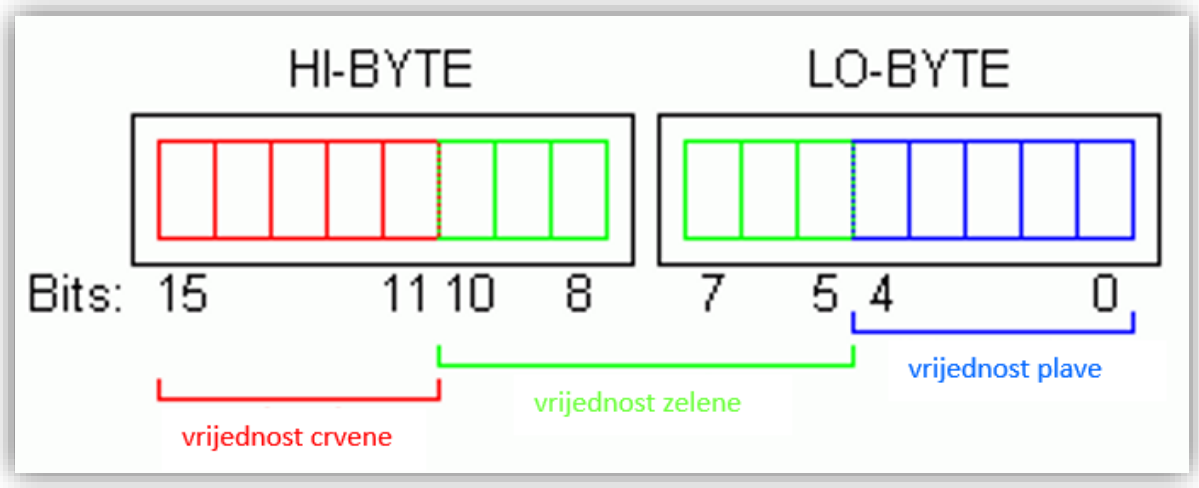
```
#include "../library/motor_led/e_epuck_ports.h"
#include "../library/motor_led/e_init_port.h"
#include "p30F6014A.h"
#include "../library/a_d/advance_ad_scan/e_ad_conv.h"
#include "../library/a_d/advance_ad_scan/e_micro.h"

main()
{
	e_init_port();
	int selector = getselector();
	switch (selector) {

	case 0: {
		e_init_ad_scan(ALL_ADC);
		int mic_0, mic_1, mic_2;
		mic_0 = e_get_micro_volume(0);
		mic_1 = e_get_micro_volume(1);
		mic_2 = e_get_micro_volume(2);
		break;
	}

	case 15: {
		unsigned long i = 0;
		for (i = 0; i < 1000000; i++) {
			asm("nop");
		}
		break;
	}
	}
	while (1);
	return (0);
}
```

Tamnom bojom istaknuto je očitanje vrijednosti svakog od 3 mikrofona.

3.4. Kamera

E-puck ima *CMOS* kameru u boji (model kamere *PixelPlus PO3030* [5]) s rezolucijom 640x480 piksela. Ako je slika zapisana prema *RGB565* standardu (Slika 39), jedan piksel sastoji se od dva bajta, što daje veličinu slike od 600kB $\left(\frac{640*480*2}{1024} = 600kB\right)$.

Obzirom da mikrokontroler *e-puck* robota ima samo 8kB radne memorije, slika od 40x40 piksela u boji uzima gotovo pola sveukupne memorije ($3{,}125kB$).



Slika 39. RGB565 [7]

Prema *RGB565* standardu, vrijednosti svih boja sadržane su unutar dva bajta (Slika 39). Prvi bajt (*low-byte*) sadrži pet bitova plave boje i prvu polovicu bitova zelene boje (tri bita), a drugi bajt (*high*-byte) sadrži drugu polovicu bitova zelene boje (tri bita) i pet bitova crvene boje.

Obzirom da su crvena i plava boja zapisane u samo pet bitova, postoje samo 32 razine tih boja (2^5, što daje raspon od 0 do 31). Zelena boja je zapisana u šest bitova, čime ona ima 64 razine boje (2^6, što daje raspon od 0 do 63).

3.4.1. Kod za čitanje vrijednosti kamere

```
#include "p30F6014A.h"
#include "stdio.h"
#include <../library/motor_led/advance_one_timer/e_agenda.h>
#include <../library/camera/fast_2_timer/e_poxxxx.h>
#include <../library/motor_led/e_epuck_ports.h>
#include <../library/uart/e_uart_char.h>

unsigned char buffer[400];

int main(void) {
	e_init_port();
	e_init_uart1();
	e_start_agendas_processing();
	int selector = getselector();

	switch (selector) {

	case 0: {

		int cam_height = 20, cam_width = 20, color_mode = GREY_SCALE_MODE;
		int subsampling_height = 8, subsampling_width = 8;
		e_poxxxx_init_cam();
		e_poxxxx_config_cam((ARRAY_WIDTH - cam_width*subsampling_width) / 2,
(ARRAY_HEIGHT - cam_height*subsampling_height) / 2, cam_width*subsampling_width,
cam_height*subsampling_height, subsampling_width, subsampling_height, color_mode);
		e_poxxxx_write_cam_registers();

		while (1) {
			e_poxxxx_launch_capture(&buffer[0]);
			while (!e_poxxxx_is_img_ready());
			char temp[50];
			sprintf(temp, "%d\r\n", buffer[0]);
			e_send_uart1_char(temp, strlen(temp));
			while (e_uart1_sending());
		}

		break;
	}

	case 15: {
		unsigned long i = 0;
		for (i = 0; i < 1000000; i++) {
			asm("nop");
		}
		break;
	}
	}
	while (1);
}
```

U ovom kodu kamera je podešena da uzme sliku veličine 160x160 piksela u tonovima sive boje. Kako je slika te dimenzije prevelika za radnu memoriju robota (potrebno je *25kB* radne memorije, a robot ima samo *8kB*) napravljen je *subsampling* od 8x po visini i širini, što daje konačnu dimenziju slike od 20x20 piksela. Nakon što je podešena dimenzija slike, postavke se

zapisuju u kameru i izvršava se beskonačna petlja gdje se uzrokuje slika i ispisuje prvi piksel te slike.

Dodatno objašnjenje koda očitanja kamere

`e_poxxxx_init_cam()` – funkcija za inicijalizaciju kamere (potrebno izvršiti samo jednom).
`e_poxxxx_config_cam(a, b, c, d, e, f, g)` – funkcija za podešavanje broja piksela, položaj tih piksela i mod boje kamere:

- `a:` Mjesto prvog uzorkovanog piksela po širini kamere. U programu sam koristio formulu `(ARRAY_WIDTH - cam_width * subsampling_width) / 2,` što je u konkretnom slučaju $\frac{640-20*8}{2} = 240$. Obzirom da je uzorkovanje po širini 160 piksela, to je sredina (uzorkuje se raspon od 240 do 400 piksela).
- `b:` Mjesto prvog uzorkovanog piksela po visini kamere. U programu sam koristio formulu `(ARRAY_HEIGHT - cam_height*subsampling_height) / 2` što je u konkretnom slučaju $\frac{480-20*80}{2} = 160$. Obzirom da po visini također uzorkujemo 160 piksela, to je sredina (uzorkuje se raspon od 160 do 320 piksela).
- `c:` Broj piksela koje želimo uzrokovati po širini kamere (u ovom slučaju, to je 160 piksela).
- `d:` Broj piksela koje želimo uzrokovati po visini kamere (u ovom slučaju, to je 160 piksela).
- `e:` *subsampling* po širini (raspoložive razine su 1, 2, 4, 8, 16)
- `f:` *subsampling* po visini (raspoložive razine su 1, 2, 4, 8, 16)
- `g:` mod boja – dostupna su dva moda: *GREY_SCALE_MODE*, tonovi sive boje gdje je jedan piksel veličine jednog bajta te *RGB_565_MODE* gdje je jedan piksel veličine dva bajta. U ovom programu, korišten je *GREY_SCALE_MODE*.

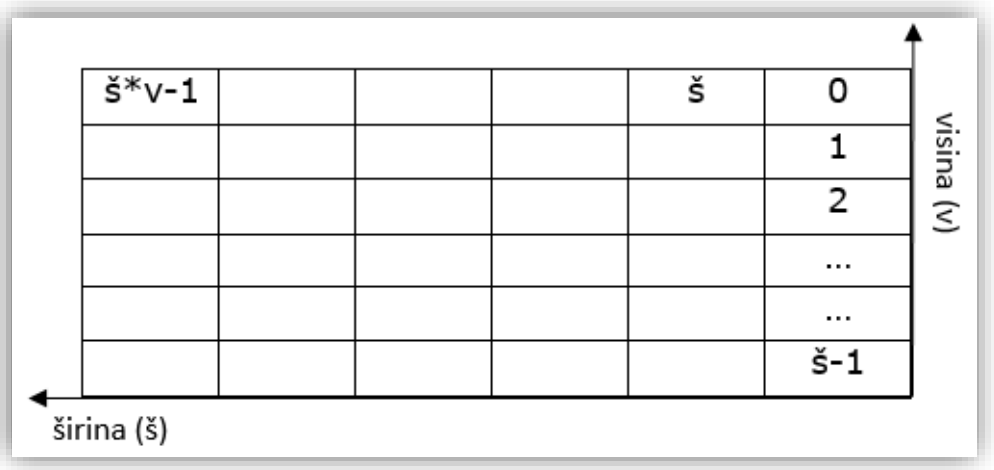
`e_poxxxx_write_cam_registers()` – funkcija koja zapisuje postavke iz prethodne funkcije (*`e_poxxxx_config_cam`*) u kameru.

`e_poxxxx_launch_capture(&buffer[0])` – nakon što je kamera inicijalizirana i podešena s prethodne dvije funkcije, ova funkcija započne uzorkovanje slike i sprema piksele u polje *buffer* koje mora biti dovoljno veliko da cijela slika stane u njega (ovu funkciju potrebno je izvršiti kad god se želi uzorkovati nova slika).

`while (!e_poxxxx_is_img_ready())` – čekanje da prethodna funkcija uzorkuje sliku i zapiše ju u *buffer*.

Ove funkcije napisane su u odnosu na koordinatni sustav kamere, a kamera je u robotu mehanički zarotirana, što rezultira time da uzorkovana širina odgovara stvarnoj visini slike, a uzorkovana visina odgovara stvarnoj širini slike.

Vezu između stvarne orijentacije slike i položaja piksela u polju *buffer* može se vidjeti na slijedećoj slici (Slika 40).



Slika 40. Veza između buffer-a i stvarnog položaja piksela

Tumačenje slike u boji je nešto kompliciranije, obzirom da je svaki piksel smješten unutar dva bajta. Zato sam napravio funkciju koja prima dvije vrijednosti iz *buffera* (dva bajta jednog piksela) te vraća vrijednosti crvene, zelene i plave boje. Za razumijevanje funkcije potrebno je osnovno znanje operacija s bitovima (*Bitwise Operations*).

3.4.1.1. Operacije s bitovima

Bitwise operator I (&, *AND*):

Tablica 1. Logička stanja operatora I, & [10]

Logička tablica operatora & (*AND*)		
Ulazi		Izlaz
A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Pod pretpostavkom da je ulaz *A=60*, a *B=13*, rezultat *A & B = 12*. U binarnom obliku:

$$\boldsymbol{A = 60 = 0011\ 1100},$$
$$\boldsymbol{B = 13 = 0000\ 1101},$$
$$\boldsymbol{A}\ \&\ B = 0011\ 1100\ \&\ 0000\ 1101$$
$$\boldsymbol{A}\ \&\ \boldsymbol{B = 0000\ 1100}$$

Bitwise operator ILI (|, OR):

Tablica 2. Logička stanja operatora ILI, | [10]

Logička tablica operatora | (*OR*)		
Ulazi		Izlaz
A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Pod pretpostavkom da je ulaz *A=60*, a *B=13*, rezultat *A | B = 61*. U binarnom obliku:

$$A = 60 = 0011\ 1100,$$
$$B = 13 = 0000\ 1101,$$
$$A\ |\ B = 0011\ 1100\ |\ 0000\ 1101,$$
$$A\ |\ B = 0011\ 1101$$

Bitwise operator XILI (^, XOR):

Tablica 3. Logička stanja operatora XILI, ^ [10]

Logička tablica operatora ^ (*XOR*)		
Ulazi		Izlaz
A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Pod pretpostavkom da je ulaz *A=60*, a *B=13*, *r*ezultat *A | B = 49*. U binarnom obliku:

$$A = 60 = 0011\ 1100,$$
$$B = 13 = 0000\ 1101,$$
$$A\ \hat{}\ B = 0011\ 1100\ \hat{}\ 0000\ 1101,$$
$$A\ \hat{}\ B = 0011\ 0001$$

Bitwise operator NE (~, NOT):

Tablica 4. Logička stanja operatora NE, ~ [10]

Logička tablica operatora ~ (*NOT*)	
Ulaz	Izlaz
A	~A
0	1
1	0

Pod pretpostavkom da je ulaz *A=60*, a *B=13*. Rezultat *~A = 195*, *~B = 242*. U binarnom obliku:

$$A = 60 = 0011\ 1100,$$
$$B = 13 = 0000\ 1101,$$
$$\sim A = 1100\ 0011,$$
$$\sim B = 1111\ 0010$$

Bitwise pomicanje bitova u lijevu stranu (<<) :

Rezultat pomaka vrijednosti *A=60* i *B=13* za dva bita u lijevu stranu jest *A = 240*, *B = 52*. U binarnom obliku:

$$A = 60 = 0011\ 1100,$$
$$B = 13 = 0000\ 1101,$$
$$A \ll 2 = 1111\ 0000,$$
$$B \ll 2 = 0011\ 0100$$

Bitwise pomicanje bitova u desnu stranu (>>) :

Rezultat pomaka vrijednosti *A=60* i *B=13* za dva bita u desnu stranu jest *A = 15*, *B = 3*. U binarnom obliku:

$$A = 60 = 0011\ 1100,$$
$$B = 13 = 0000\ 1101,$$
$$A \ll 2 = 0000\ 1111,$$
$$B \ll 2 = 0000\ 0011$$

3.4.1.2. Funkcija za pretvorbu RGB565 u RGB

```
char blue = 0, red = 0, green = 0;
void RGB565(char lobyte, char hibyte) {
	red = 0;
	green = 0;
	blue = 0;
	red = (hibyte & 0xF8) >> 3;
	green = ((hibyte & 0x7) << 3) | ((lobyte & 0xE0) >> 5);
	blue = lobyte & 0x1f;
}
```

Ako pozovemo funkciju *RGB565(A, B)* sa *A* i *B* varijablama, tok izvršavanja je sljedeći:

$$lobyte = A = 60 = 0011\ 1100,$$
$$hibyte = B = 13 = 0000\ 1101,$$
$$blue = 0011\ 1100\ \&\ 0001\ 1111,$$
$$blue = 0001\ 1100 = 28,$$
$$red\ = (0000\ 1101\ \&\ 1111\ 1000) \gg 3,$$
$$red = 0000\ 1000 \gg 3,$$
$$red = 0000\ 0001 = 1,$$
$$green = \big((0000\ 1101\ \&\ 0000\ 0111) \ll 3\big)\ |\ \big((0011\ 1100\ \&\ 1110\ 0000) \gg 5\big),$$
$$green = (0000\ 0101 \ll 3)\ |\ (0010\ 0000 \gg 5),$$
$$green = 0010\ 1000\ |\ 0000\ 0001,$$
$$green = 0010\ 1001 = 41$$

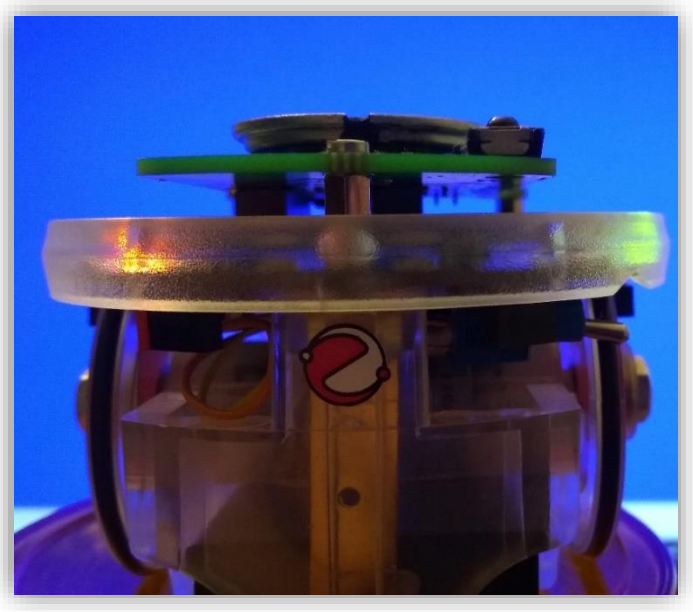
Funkcija radi tako da joj se iz *buffera* pošalju *low-byte* i *high-byte*, a ona prema njima izračunava *red*, *green*, *blue* varijable. Za isti piksel parovi *hibyte i lobyte* su: `RGB565(buffer[1], buffer[0])`, odnosno `RGB565(buffer[parni broj + 1], buffer[parni broj])`.

3.4.1.3. Automatski balans bijele boje i automatska ekspozicija (auto white balance, auto exposure)

Kamera automatski ima uključeni balans bijele boje i automatsku ekspoziciju, što otežava obradu boja. Kamera za istu sliku kroz vrijeme mijenja intenzitet crvene, zelene i plave boje.

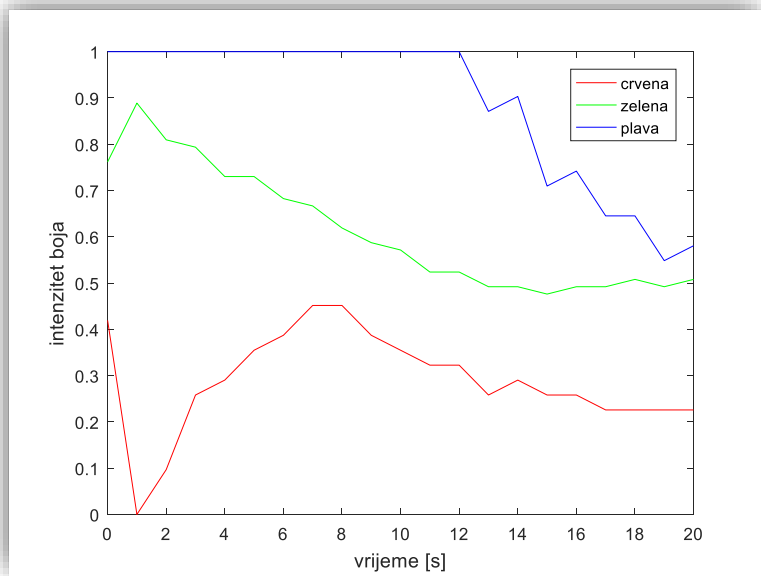
Funkciju za promjenu postavki (*`e_poxxxx_set_awb_ae(ABBB, AE)`*) potrebno je staviti prije funkcije *`e_poxxxx_write_cam_registers()`*, vrijednosti su *0* za isključeno i *1* za uključeno.

Radi demonstracije razlike boje u vremenu, postavio sam robot pred ekran računala koji je ispunjen plavom bojom (Slika 41.) i snimio boje s različitim postavkama. Vrijeme uzorkovanja je *1s*. Obzirom da zelena boja ima veću rezoluciju od crvene i plave, u grafovima koji slijede, boje su podijeljene s vlastititim maksimumom.



Slika 41. Robot pred ekranom računala za testiranje postavki automatskog balansa bijele boje i automatske ekspozicije

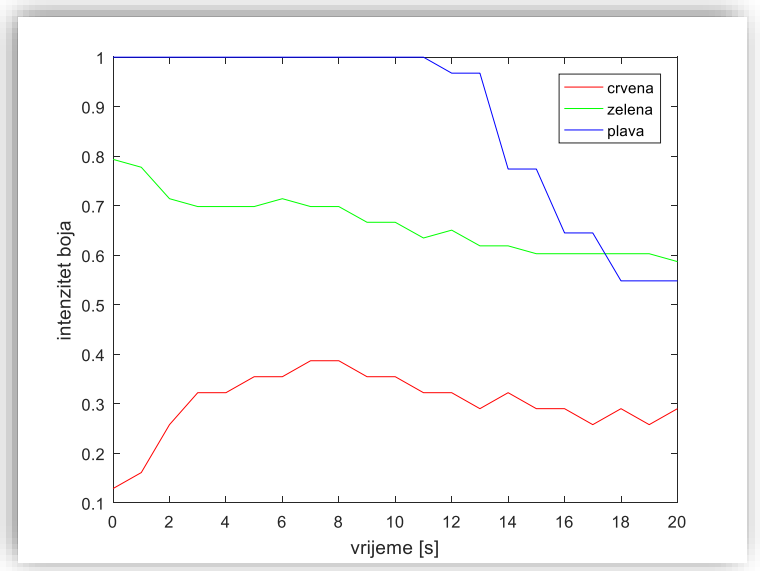
Sa standardnim postavkama ili sa `e_poxxxx_set_awb_ae(1, 1)`, vidi se velika promjena prikazivanja boja (Slika 42).



Slika 42. Intenzitet boja s uključenim automatski balansom bijele boje i automatskom ekspozicijom

S postavkama podešenim tako da je uključen automatski balans boje i isključena automatska ekspozicija i dalje se vidi velika promjena boja (Slika 43). Čak dolazi do problema da se intenzitet plave boje spusti ispod razine zelene boje.

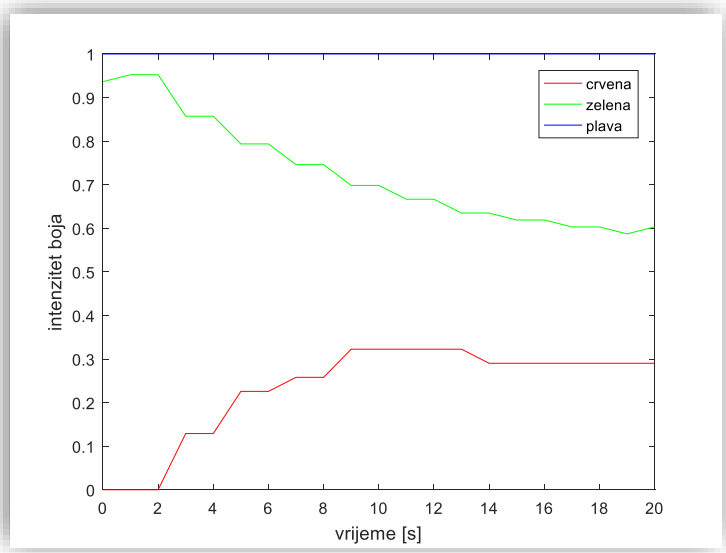
Postavke kamere u ovom slučaju su: `e_poxxxx_set_awb_ae(1, 0)`.



Slika 43. Intenzitet boja s uključenim automatski balansom bijele boje i isključenom automatskom ekspozicijom

Kada se isključi automatski balans bijele boje, intenzitet boja se i dalje mijenja kroz vrijeme, ali s ovakvom karakteristikom može se sa sigurnošću raspoznati da je plava boja dominantna (Slika 44).

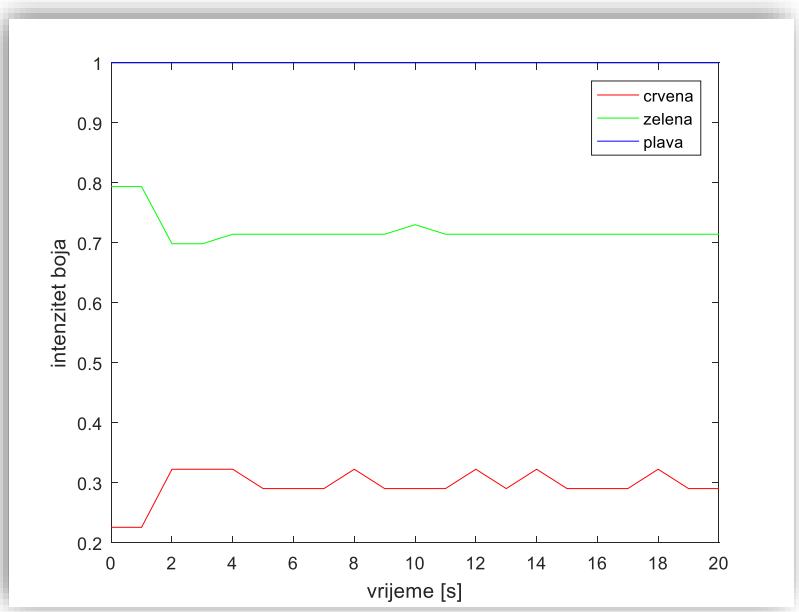
Postavke kamere u ovom slučaju su: `e_poxxxx_set_awb_ae(0, 1)`.



Slika 44. Intenzitet boja s isključenim automatskim balansom bijele boje i uključenom automatskom ekspozicijom

Kada se isključe sve automatske postavke kamere, dobije se karakteristika s kojom je najlakše raditi za raspoznavanje boja, kao što se vidi na slici (Slika 45).

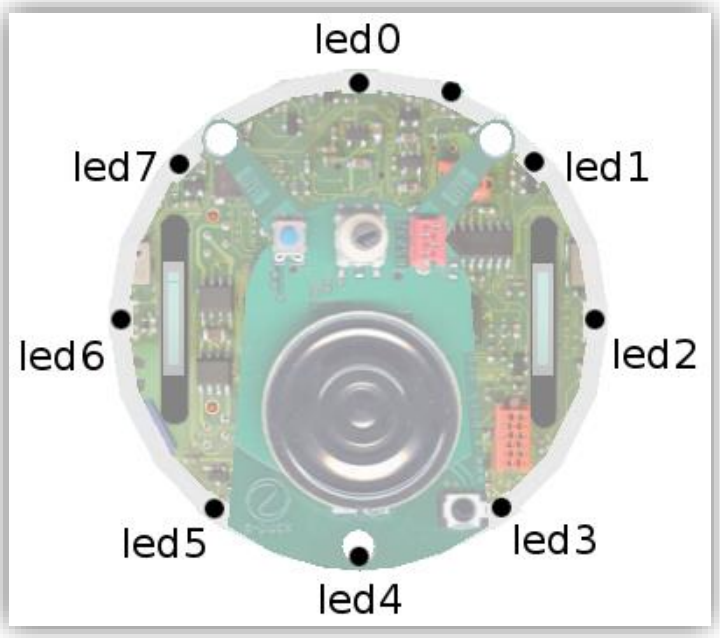
Postavke kamere u ovom slučaju su: `e_poxxxx_set_awb_ae(0, 0)`.



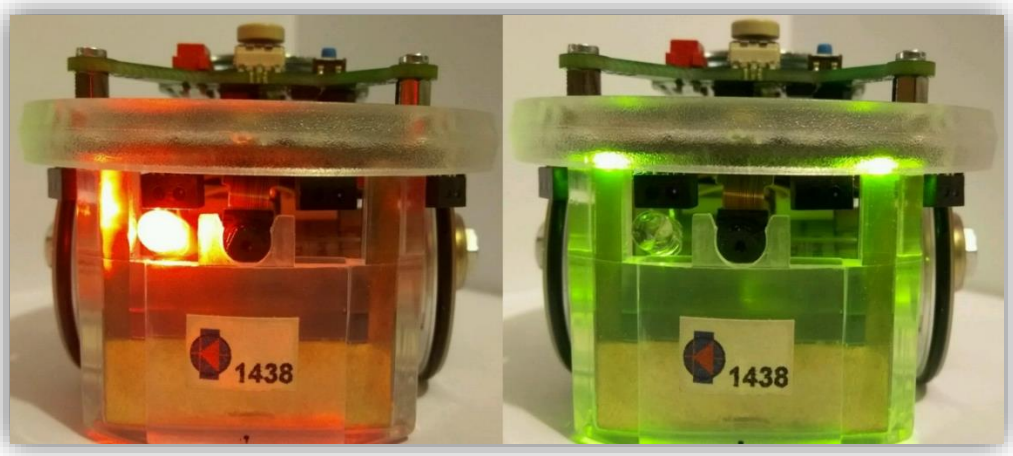
Slika 45. Intenzitet boja s isključenim automatskim balansom bijele boje i isključenom automatskom ekspozicijom

3.5. LED (light-emitting diode) prsten

Po cijelom prstenu *e-puck* robota postavljeno je LED crveno osvjetljenje, koje omogućuje interakciju s kamerom. Osim LED prstena postoji još jedna crvena LED dioda na prednjoj strani te zelena LED dioda u transparentnom kućištu. Položaj LED dioda po prstenu prikazan je na slici (Slika 46).



Slika 46. Položaj *LED* dioda



Slika 47. Lijevo robot s upaljenom prednjom LED diodom i desno s upaljenom body LED diodom

3.5.1. Kod za upravljanje LED diodama

```
#include "p30F6014A.h"
#include "../library/utility/utility.h"
#include "../library/motor_led/e_epuck_ports.h"
#include "../library/motor_led/e_init_port.h"
#include "../library/motor_led/advance_one_timer/e_led.h"

main() {
	e_init_port();
	int selector = getselector();
	switch (selector) {
	case 0: {
			e_set_front_led(1);
			e_set_body_led(1);
			int i = 0;
			for (i = 0; i < 8; i++) {
				e_set_led(i, 1);
			}
			wait(1000000);
			e_set_front_led(0);
			e_set_body_led(0);
			for (i = 0; i < 8; i++) {
				e_set_led(i, 0);
			}
		}
		break;
	}

	case 15: {
		unsigned int i = 0;
		for (i = 0; i<1000000; i++) {
			asm("nop");
		}
		break;
	}
	}
	}
	while (1);
}
```

Ovaj programski isječak prikazuje paljenje svake *LED* dioda robota zasebno, te nakon jedne sekunde čekanja, njihovo gašenje obrnutim redoslijedom. Varijabla *i* odgovara broju *LED* diode na slici (Slika 46), a položaj *front* i *body LED* dioda može se vidjeti na slici (Slika 47). Logička vrijednost *1* označava upaljenu *LED* diodu, a logička vrijednos *0* ugašenu.

Za brže paljenje ili gašenje svih *LED* dioda po prstenu robota, koristi se ista funkcija kao i za paljenje ili gašenje zasebne *LED* diode, s tom razlikom da se kao broj *LED* diode upisuje bilo koji broj koji *ne pripada* nijednoj od trenutnih *LED* dioda robota; npr. kako bismo upalili sve *LED* diode odjednom, možemo koristiti funkciju `e_set_led(9, 1)`.

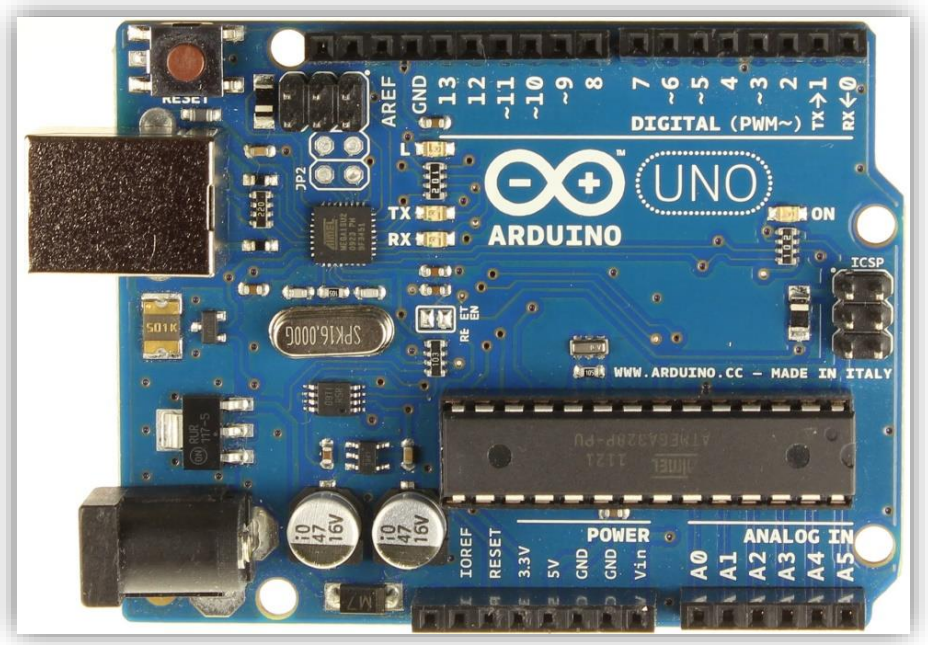
4. IZRADA SENZORA ZA DETEKCIJU PODA

E-puck roboti sami po sebi nemaju infracrveni senzor za detekciju poda. Obzirom da je *e-puck open-source* projekt, postoji već gotovo rješenje, ali je dokumentacija loše rezolucije i elektroničke sheme je teško pouzdano pročitati. Senzor je baziran na *microchip-ovom* mikrokontroleru za koji nemam programator (potreban je *MPLAB ICD 2 programator*). Cijena gotovog senzora je *300CHF* [11].

Ovo rješenje nije mi odgovaralo te sam projektirao i izradio vlastiti senzor, baziran na *AVR* mikrokontroleru *ATmega328* koji je zbog svoje popularnosti izrazito jeftin.

Mikrokontroler senzora čita analogne vrijednosti infracrvenih senzora i šalje stanja preko I^2C (*Inter-Integrated Circuit*) serijske veze mikrokontroleru robota.

Za izradu programa mikrokontrolera senzora koristio sam *Arudino* programski jezik, a samo programiranje *bootloadera* na mikrokontroler napravio sam pomoću *Arduino Uno* razvojne ploče (Slika 48).

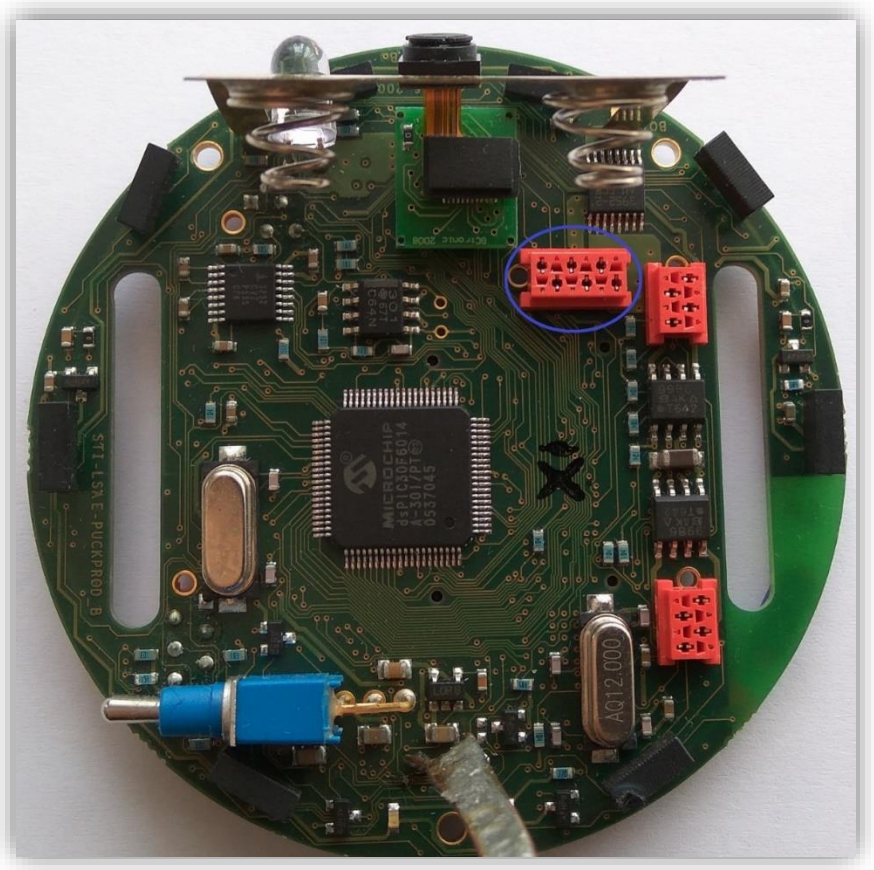


Slika 48. Arduino Uno

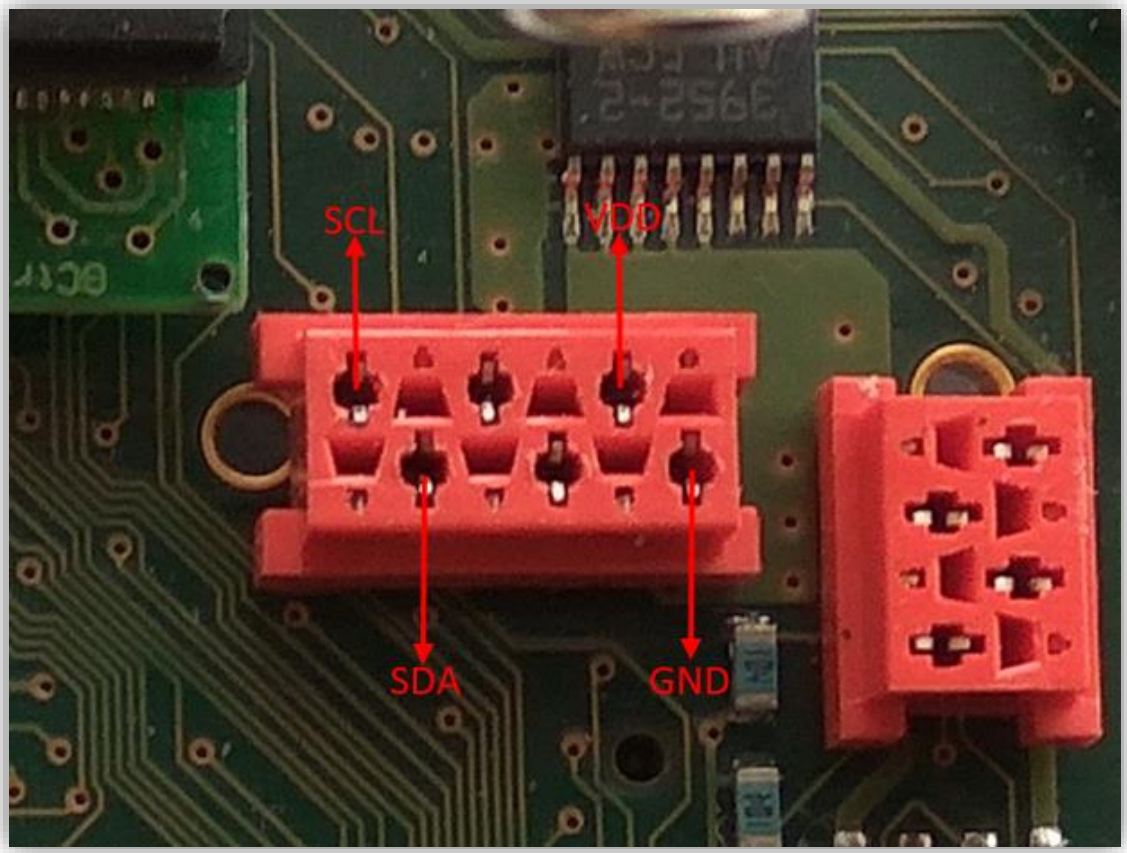
Za same senzore sam koristio *Everlight ITR8307/TR8*, a glavni razlog za njihov odabir je njihova mala dimenzija. Na slici (Slika 49) se može vidjeti nedostatak prostora.

Slika 49. E-puck robot s ugrađenim senzorom

Konektor za I^2C komunikaciju nalazi se s donje strane tiskane pločice *e-pucka* i potrebno je rastaviti robot da bi se došlo do njega. Konektor je jedini konektor sa šest pinova i označen je na slijedećoj slici (Slika 50).



Slika 50. Konektor s pinovima za I²C komunikaciju



Slika 51. Konektor s označenim pinovima

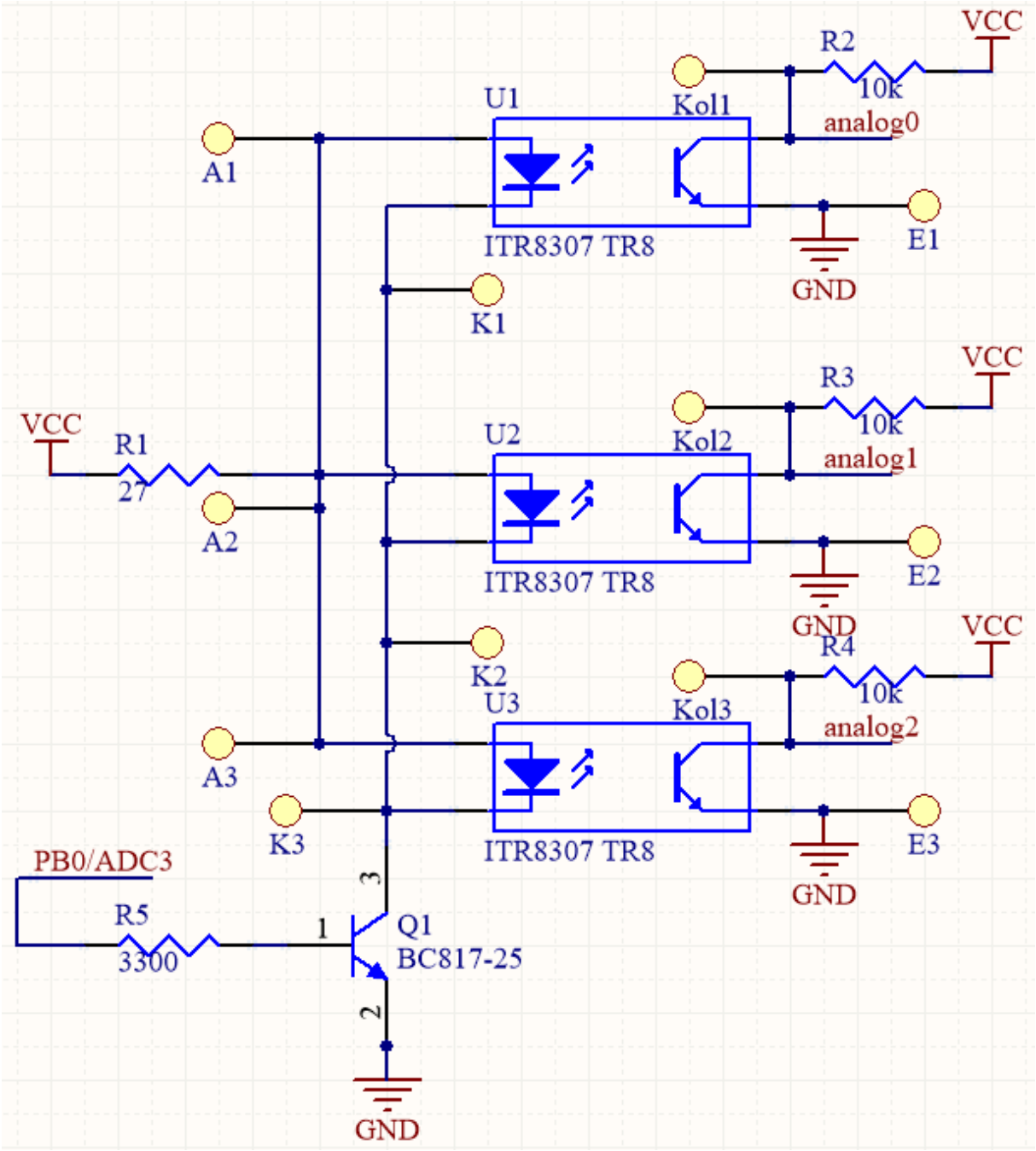
Na prethodnoj slici (Slika 51) prikazan je konektor i pinovi na koje moramo spojiti senzor. *Vdd* i *GND* pinovi su za napajanje senzora, a *SCL (clock line)* i *SDA (data line)* za komunikaciju preko I^2C sabirnice. Konektor na slici (Slika 51) isto je orijentiran kao konektor na slici na kojoj se vidi cijela pločica (Slika 50).

4.1. Shema i tiskana pločica senzora

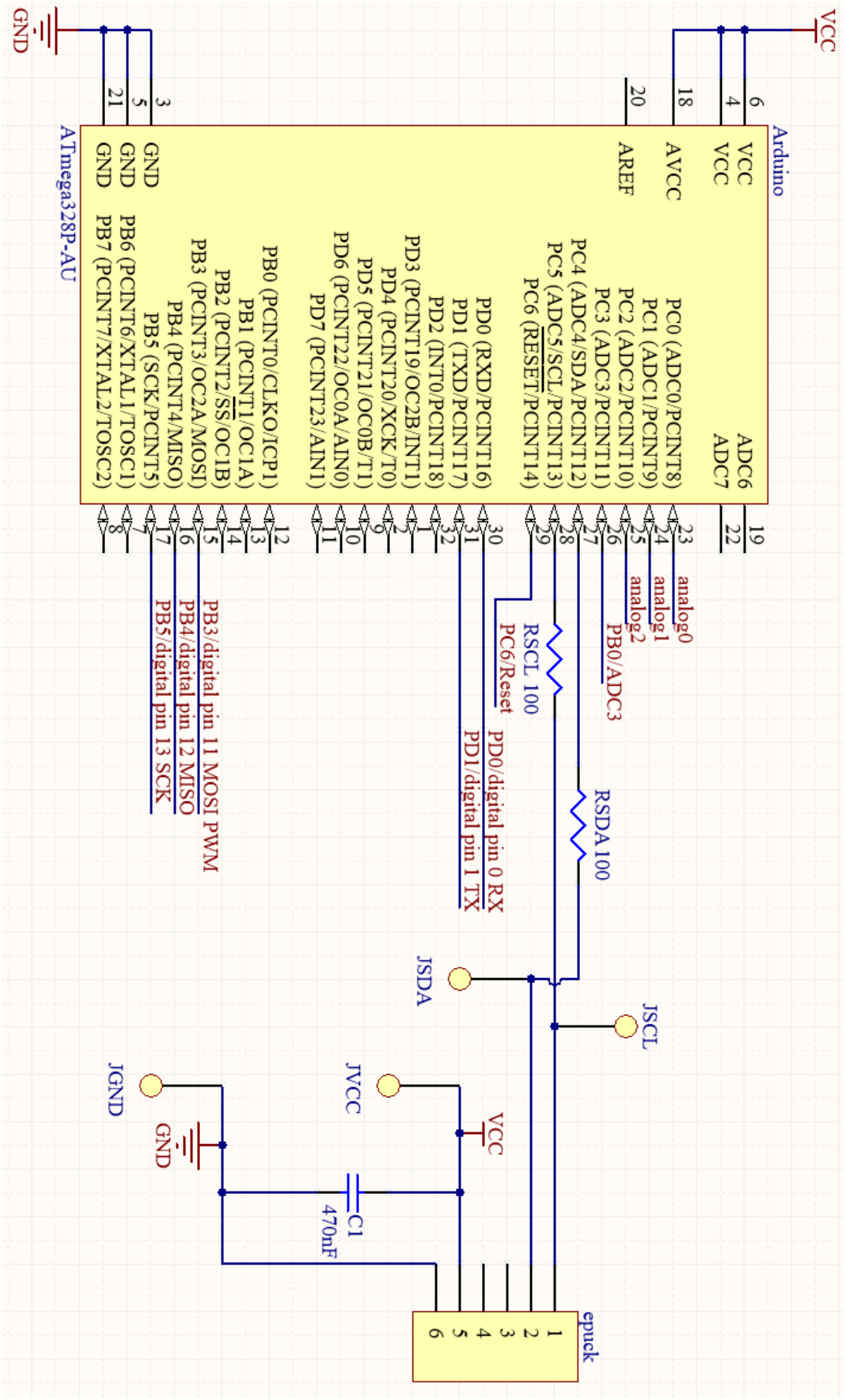
Shemu i PCB pločicu radio sam u programu *Altium Designer 16.0.5 (Build 271)*.

Sve datoteke potrebne za izradu tiskane pločice zajedno sa shemom nalaze se na linku: Ground Senzor Altium. Preporučana debljina pločice je *1mm*.

Radi bolje preglednosti shema je podijeljena u dva dijela (Slika 52, Slika 53). Prvi dio sheme (Slika 52) prikazuje senzore (*Everlight ITR8307/TR8*). Drugi dio sheme (Slika 53) prikazuje način na koji je mikrokontroler (*ATmega 328*) spojen na napajanje i konektor robota. Kako bih uštedio na prostoru, nisam stavio vanjski oscilator, nego koristim interni u mikrokontroleru (to su ukupno tri komponente manje, oscilator i dva kondezatora).

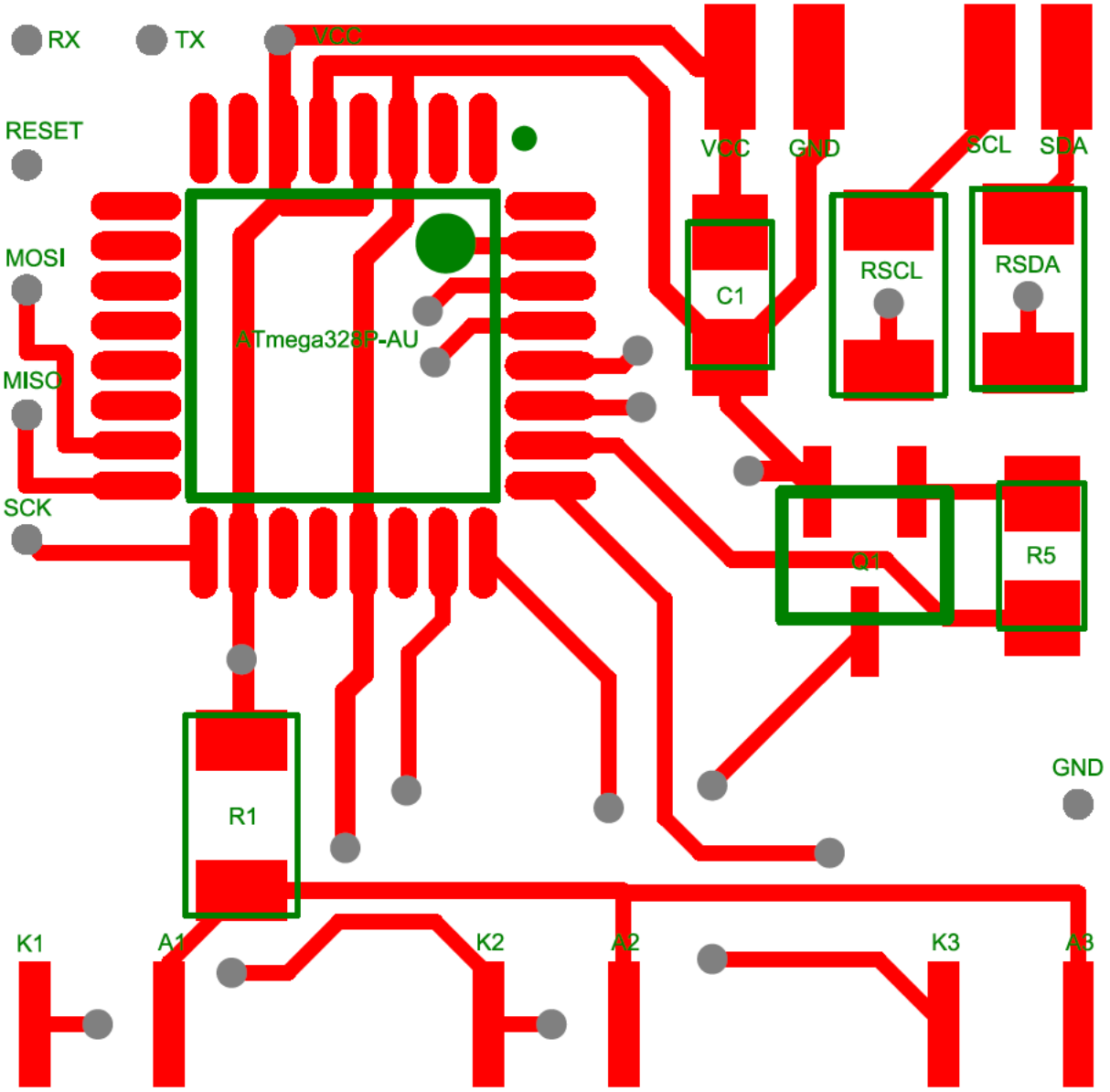


Slika 52. Prvi dio sheme senzora

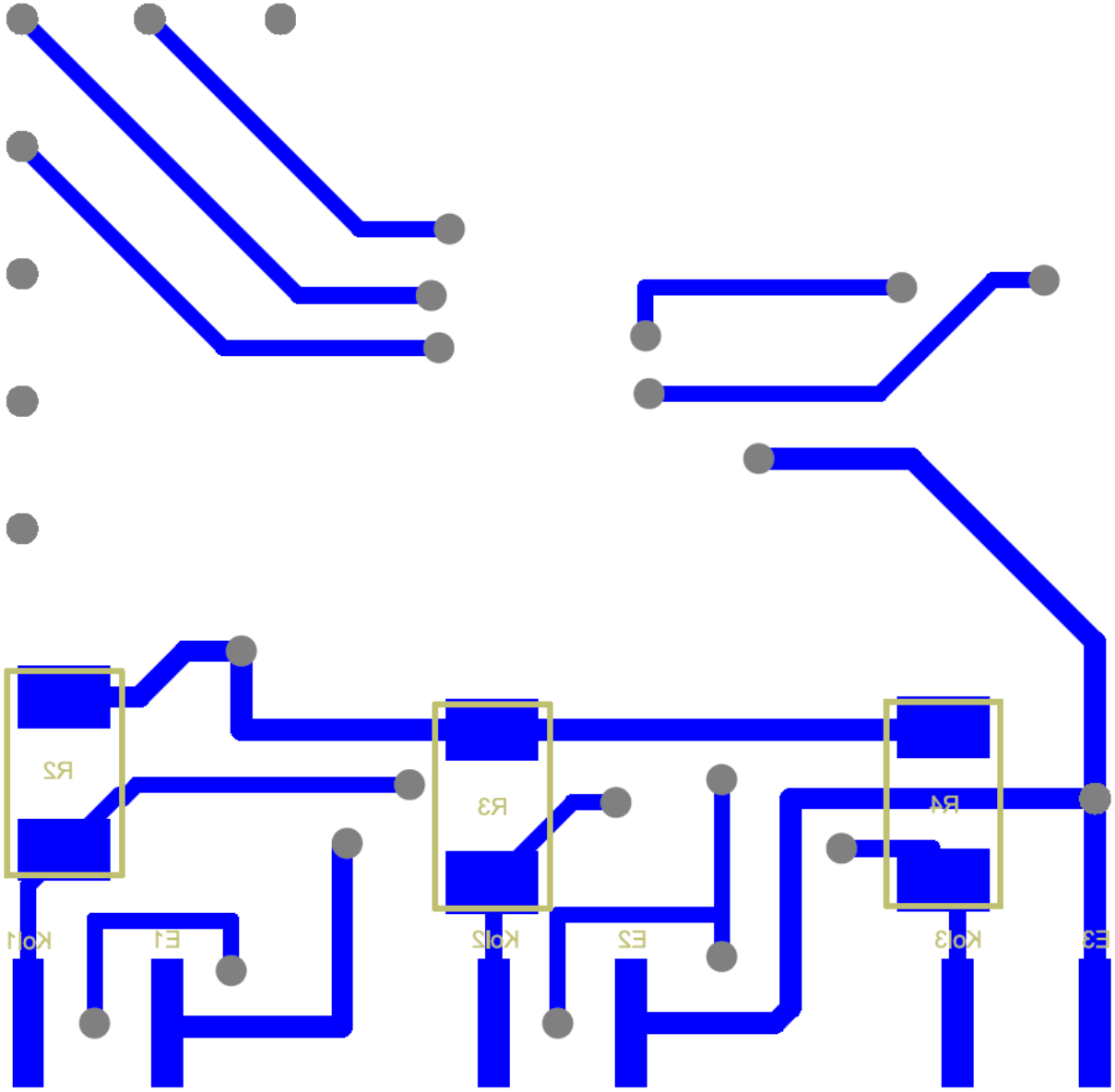


Slika 53. Drugi dio sheme senzora

Slike koje slijede (Slika 54, Slika 55) prikazuju gornju i donju stranu pločice, te kako su vodovi i komponente postavljene. Sami senzori nekonvencionalno su postavljeni, uzduž *debljine* pločice (Slika 49).



Slika 54. Gornja strana pločice



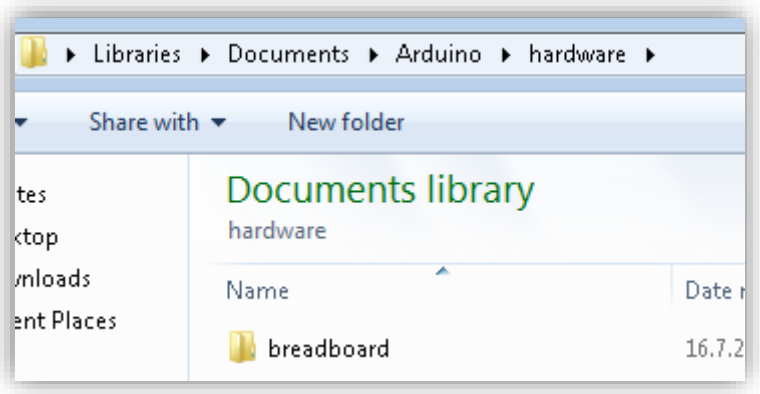
Slika 55. Donja strana pločice

4.2. Programiranje *bootloadera* i programa na senzor

Za programiranje mikrokontrolera senzora potrebno je imati funkcionalan *Arduino Uno* (Slika 48). Potrebno je imati *Arduino Software*, koji je moguće preuzeti na www.arduino.cc. Verzija korištena u ovom radu jest Arduino 1.6.8.zip.

Potrebno je preuzeti i breadboard-1-6-x.zip [12] te otpakirati sadržaj zip datoteke u *Documents/Arduino/hardware*.

Ukoliko nedostaje neka mapa potrebno ju je ručno napraviti. Sve treba izgledati kao na slici (Slika 56).



Slika 56. Arduino kopiranje *bootloadera*

Potrebno je spojiti *Arduino* na računalo, otpakirati i pokrenuti *Arduino Software*. U izborniku *Tools* potrebno je podesiti *Board* i *Port*.

Board je u mom slučaju *"Arduino/Genuino Uno"*, a *Port: "COM3"*. Port na kojem se nalazi Arduino moguće je otkriti na isti način kao i port za *e-puck*, što je opisano u poglavlju (1.5), a prikazano na slici (Slika 23).

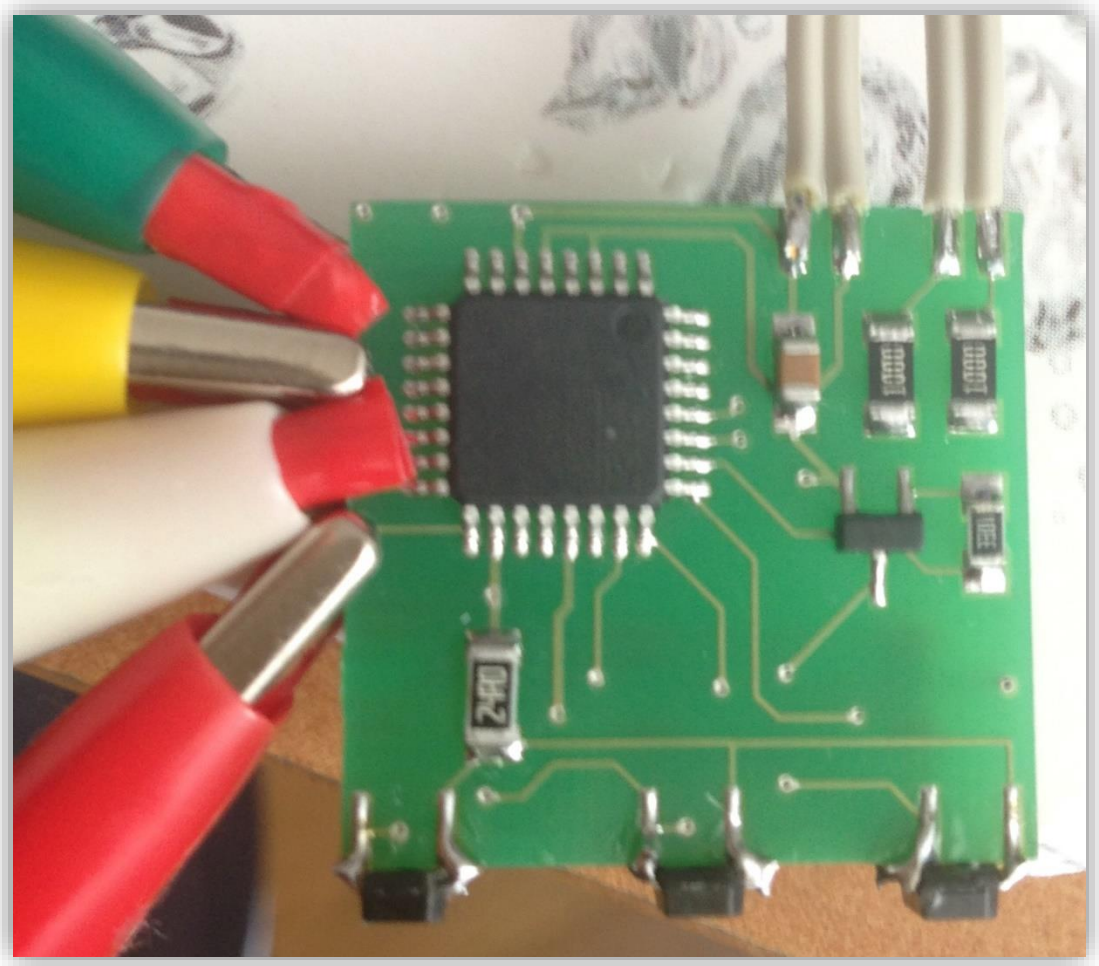
U izborniku odabrati *File -> Examples -> ArduinoISP -> ArduinoISP*, pa *Sketch -> Upload*. Pričekati dok se program ne prebaci na *Arduino*, te isključiti *Arduino* iz računala, kako se pri spajanju slučajno ne bi napravio kratki spoj.

Za programiranje *bootloadera* potrebno je spojiti *Arduino* i senzor na određeni način, što je opisano u tablici koja slijedi (Tablica 6).

Tablica 5. Spajanje za programiranje *bootloadera* [12]

Arduino pin	Senzor
10	RESET
11	MOSI
12	MISO
13	SCK
5V	VCC
GND	GND

RESET, MOSI, MISO, SCK, VCC, GND pinovi označeni su na slici (Slika 54). Spajanje je rađeno preko hvataljki, a može se vidjeti na slici (Slika 57). Svaka hvataljka izolirana je s jedne strane, kako ne bi došlo do kratkog spoja.



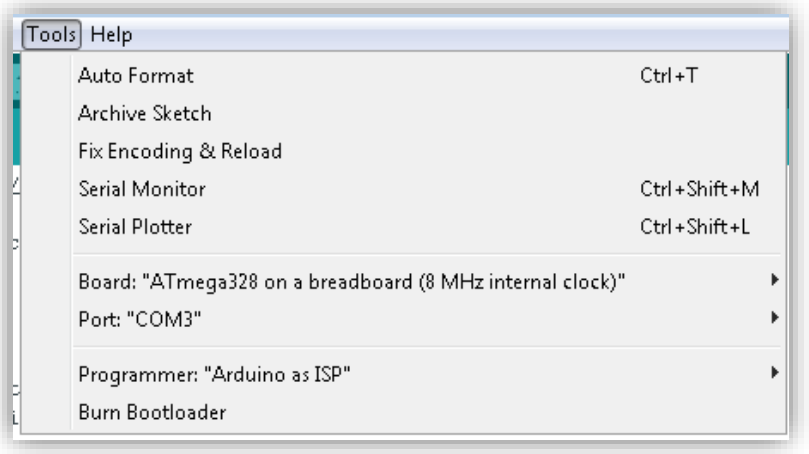
Slika 57. Spajanje senzora za programiranje *bootloadera*

Nakon što je sve dobro spojeno (provjeriti multimetrom) potrebno je opet uključiti *Arduino* u računalo.

Postaviti slijedeće opcije u *Arduino* programu:

- *Tools -> Board -> "ATmega328 on a breadboard (8 MHz internal clock)"*
- *Tools ->Programmer -> "Arduino as ISP"*
- *Tools -> Burn Bootloader*

Sve treba biti kao na slijedećoj slici (Slika 58).



Slika 58. Postavke za programiranje *bootloadera*

Nakon odabiranja opcije *Burn Bootloader*, potrebno je pričekati programiranje *bootloadera*. potom preuzeti program I2C_Slave_Ground_Sensor.zip, otpakirati ga i otvoriti u *Arduino* programu. Nakon toga, s *Arduino Uno* pločice maknuti mikrokontroler te spojiti *Arduino* razvojnu ploču sa senzorom tla, prema tablici koja slijedi (Tablica 6).

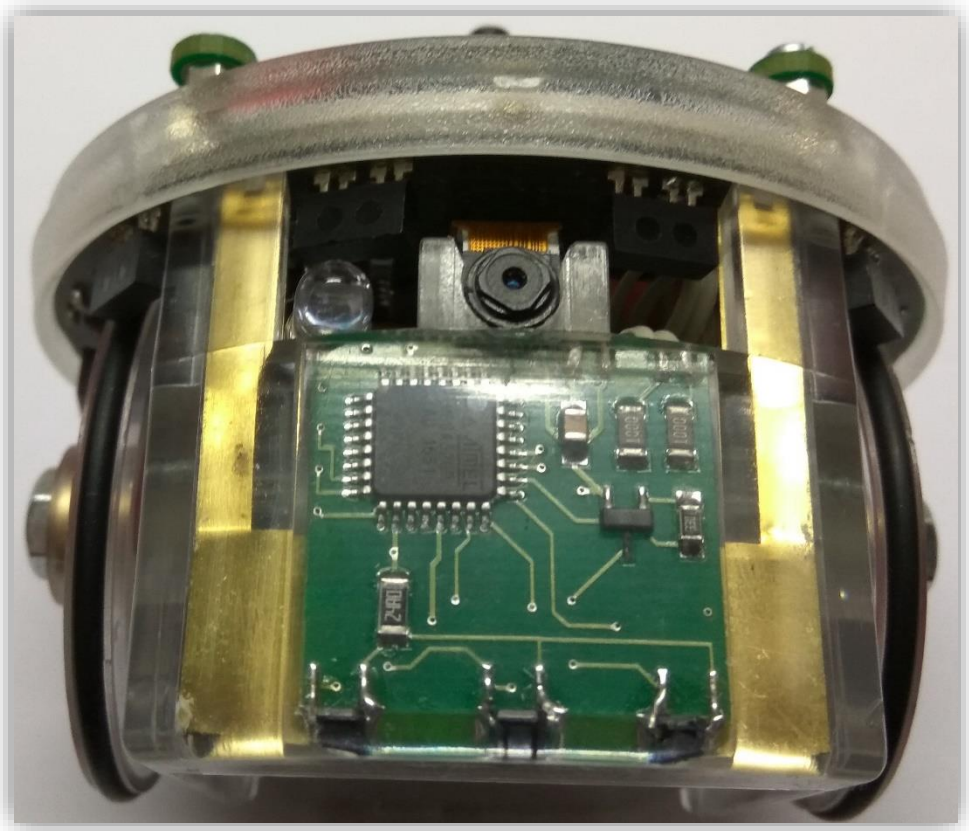
Tablica 6. Spajanje Arduina i senzora za prebacivanje programa [12]

Arduino pin	Senzor
RX 0	RX
TX 1	TX
RESET	RESET
5V	VCC
GND	GND

Kada su *Arduino* razvojna ploča i senzor tla spojeni, treba provjeriti jesu li opcije u programu i dalje ispravno postavljene:

- *Bord: "ATmega328 on a breadboard (8MHz internal clock)"*
- *Port:* odgovarajući *COMXX* port postavljen.

Ukoliko je sve u redu, odabrati opciju *Sketch -> Upload.* Senzor je sada uspješno programiran te se može ugraditi u robota.



Slika 59. Ugrađeni senzor

4.3. Kod kojim se programira senzor

```
#include <Wire.h>
char aread[3] = { 0, 0, 0 };
int count = 0;

void setup() {
	Wire.begin(0x8 >> 1); //i2c adresa #8
	Wire.onRequest(requestEvent);
	pinMode(A3, OUTPUT);
	digitalWrite(A3, LOW);
}

void loop() {

}

void requestEvent() {
	digitalWrite(A3, HIGH);
	delayMicroseconds(10);

	if (count == 0) {
		int val = analogRead(0);
		aread[0] = map(val, 0, 1023, 0, 127);

		val = analogRead(1);
		aread[1] = map(val, 0, 1023, 0, 127);

		val = analogRead(2);
		aread[2] = map(val, 0, 1023, 0, 127);

		digitalWrite(A3, LOW);
		Wire.write(aread[count]);
		count = 1;
	}
	else if (count == 1) {
		Wire.write(aread[count]);
		count = 2;
	}
	else if (count == 2) {
		Wire.write(aread[count]);
		count = 0;
	}
}
```

4.4. Kod za čitanje stanja senzora

```
#include <stdio.h>
#include "p30F6014A.h"
#include "../../library/motor_led/e_epuck_ports.h"
#include "../../library/motor_led/e_init_port.h"
#include "../../library/utility/utility.h"
#include "../../library/uart/e_uart_char.h"
#include "../../library/I2C/e_I2C_master_module.h"
#include "../../library/I2C/e_I2C_protocol.h"

int main(void) {
	e_init_port();
	e_init_uart1();
	e_i2c_init();

	unsigned char buffer[100];
	unsigned char desni, srednji, lijevi;
	desni = 255;
	srednji = 255;
	lijevi = 255;

	int selector = getselector();
		switch (selector) {
		case 0: {
			while (1) {
				e_i2c_enable();
				desni = e_i2cp_read(0x8, 1);
				srednji = e_i2cp_read(0x8, 1);
				lijevi = e_i2cp_read(0x8, 1);
				e_i2cp_disable();

				sprintf(buffer, "I2C: desni:%d, srednji:%d, lijevi:%d\r\n",
desni, srednji, lijevi);
				e_send_uart1_char(buffer, strlen(buffer));
				while (e_uart1_sending());
				wait(1000000);
			}
			break;
		}
		case 15: {
			wait(1000000);
			break;
		}
	while (1);
	}
}
```

Funkcije za korištenje I^2C komunikacije:

- `e_i2c_init()` – inicijalizacija
- `e_i2c_enable()` – početak I^2C komunikacije
- `e_i2cp_read(0x8, 1)` – čitanje jednog bajta s uređaja čija je adresa 0x8

Kada robot započne I^2C komunikaciju i prvi put zatraži podatke, mikrokontroler na senzoru upali senzore, očita vrijednost svih senzora i vrati vrijednost desnog senzora. Kada robot drugi put zatraži podatke, mikrokontroler vraća vrijednosti srednjeg senzora, a kada robot zatraži podatke treći put, mikrokontroler vraća vrijednosti lijevog senzora. To znači da je za očitanje vrijednosti sa sva tri senzora potrebno pozvati funkciju `e_i2cp_read(0x8, 1)` tri puta.

Vrijednosti senzora se kreću od *0* do *127*. Vrijednost *0* označava maksimalnu refleksiju, a vrijednost *127* da senzor ne detektira ništa.

5. PRIMJERI

U poglavlju koje slijedi, biti će pobliže opisane osnovne naredbe *e-puck* robota. Sav kod napisan je unutar projekta *Clean*, čije stvaranje je objašnjeno u poglavlju 1.4, a priložen je uz diplomski rad, kao i na dropbox linku epuck_primjeri.zip.

5.1. Paljenje i gašenje LED indikatora

U ovom programu pokazano je manipuliranje LED indikatora.

Prvo se pale zeleni LED indikatori u kućištu i LED sa oznakom 0 na prstenu, prema slici (Slika 46). Zatim se redom pale ostali LED indikatori te se potom gase istim redom. LED indikator s oznakom 9 radi obrnuto od zelenih LED indikatora na kućištu.

```
#include <stdio.h>
#include <stdlib.h>
#include "p30F6014A.h"

#include "../library/motor_led/e_epuck_ports.h"
#include "../library/motor_led/e_init_port.h"
#include "../library/utility/utility.h"
#include "motor_led/advance_one_timer/e_led.h"

int main() {
    e_init_port(); // inicijalizacija portova
    unsigned int led = 0;

    int selector = getselector(); // očitanje stanje selektora
    switch (selector) {

    // logika za paljenje/gašenje LED indikatora
    case 0: {
       while (1) {
           led = led ^ 1; // bitwise XOR za promjenu led varijable iz 0 u 1 i obrnuto
           e_set_front_led(led ^ 1);     // pali, gasi LED9 (1 = upali, 0 = ugasi)
           e_set_body_led(led);   // pali, gasi LED kućišta (1 = upali, 0 = ugasi)

           // za LED na prstenu, i označava trenutni LED (0-7), a varijabla led
označava da li je upaljeno ili ugašeno
           for (i = 0; i<8; i++) {
              e_set_led(i, led);
              wait(1000000);      // čeka 1s
           }
       }
       break;}

    case 15: { // čekanje za lakše reprogramiranje robota
       unsigned int i = 0;
       for (i = 0; i<1000000; i++) {
          asm("nop");
       }
       break;}
    }
    while (1);
    return(0);
}
```

5.2. Motori

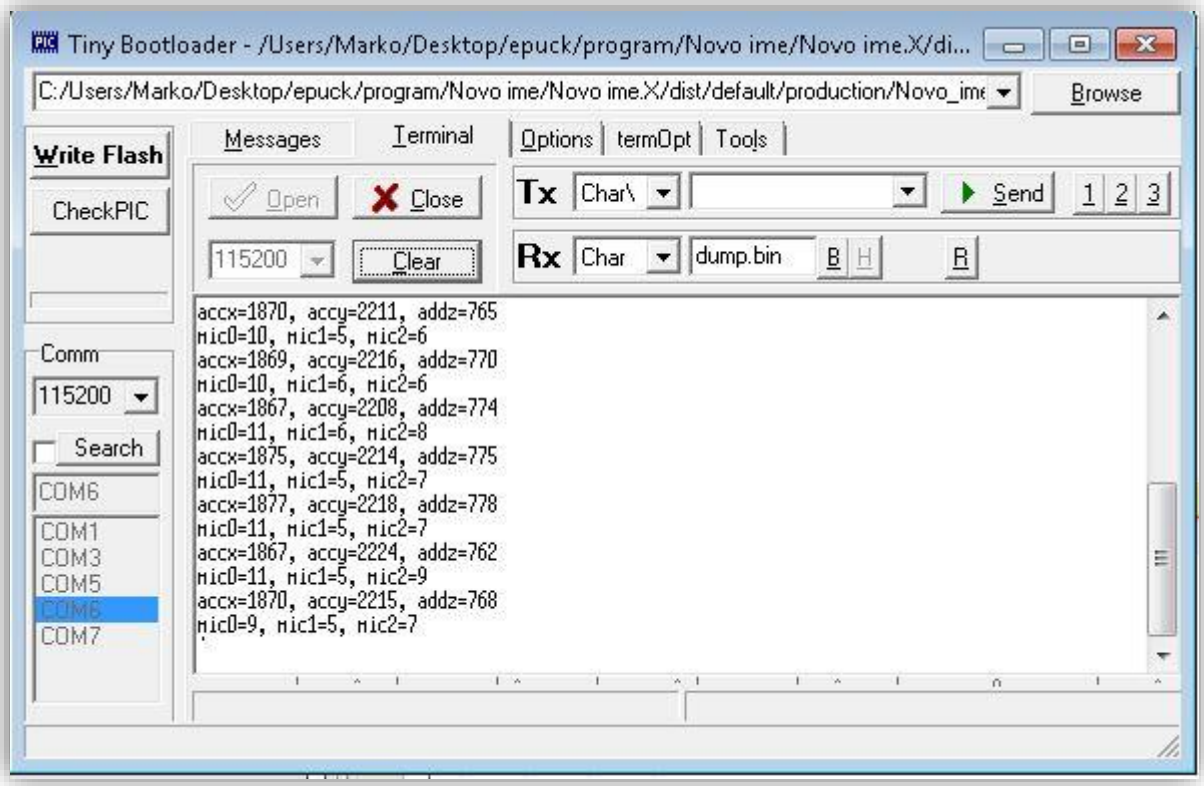
Program koji pokazuje manipulaciju *e-puck* motorima. Program radi tako da *e-puck* upali motore, napravi 1000 koraka te potom stane.

```
#include <stdio.h>
#include <stdlib.h>
#include "p30F6014A.h"

#include "../library/motor_led/e_epuck_ports.h"
#include "../library/motor_led/e_init_port.h"
#include "../library/utility/utility.h"
#include "../library/motor_led/advance_one_timer/fast_agenda/e_led.h"
#include "../library/motor_led/advance_one_timer/fast_agenda/e_motors.h"
#include "../library/motor_led/advance_one_timer/fast_agenda/e_agenda_fast.h"

int main() {
	e_init_port();
	e_init_motors(); // inicijalizacija motora
	e_start_agendas_processing(); // inicijalizacija agendas_processing, potrebno za
motore ako koristimo e_motors.h iz fast_agenda mape
	e_set_steps_left(0); // nuliranje prijeđenih koraka
	e_set_steps_right(0);

	int selector = getselector();
	switch (selector) {

	// logika za pokretanje motora
	case 0: {
		// postavljanje brzine motora na maksimum (1000)
		e_set_speed_left(1000);
		e_set_speed_right(1000);

		// čitanje broja koraka i čekanje da se napravi 1000 koraka
		while (e_get_steps_left() < 1000 && e_get_steps_right() < 1000);

		// gašenje motora
		e_set_speed_left(0);
		e_set_speed_right(0);
		break;
	}

	case 15: { // čekanje za lakše reprogramiranje robota
		unsigned int i = 0;
		for (i = 0; i < 1000000; i++) {
			asm("nop");
		}
		break;
	}
	}
	while (1);
	return(0);
}
```

5.3. Bluetooth, senzor ubrzanja, mikrofon

Program čita x, y i z os senzora ubrzanja i glasnoću na sva tri mikrofona i podatke preko *bluetootha* šalje na računalo.

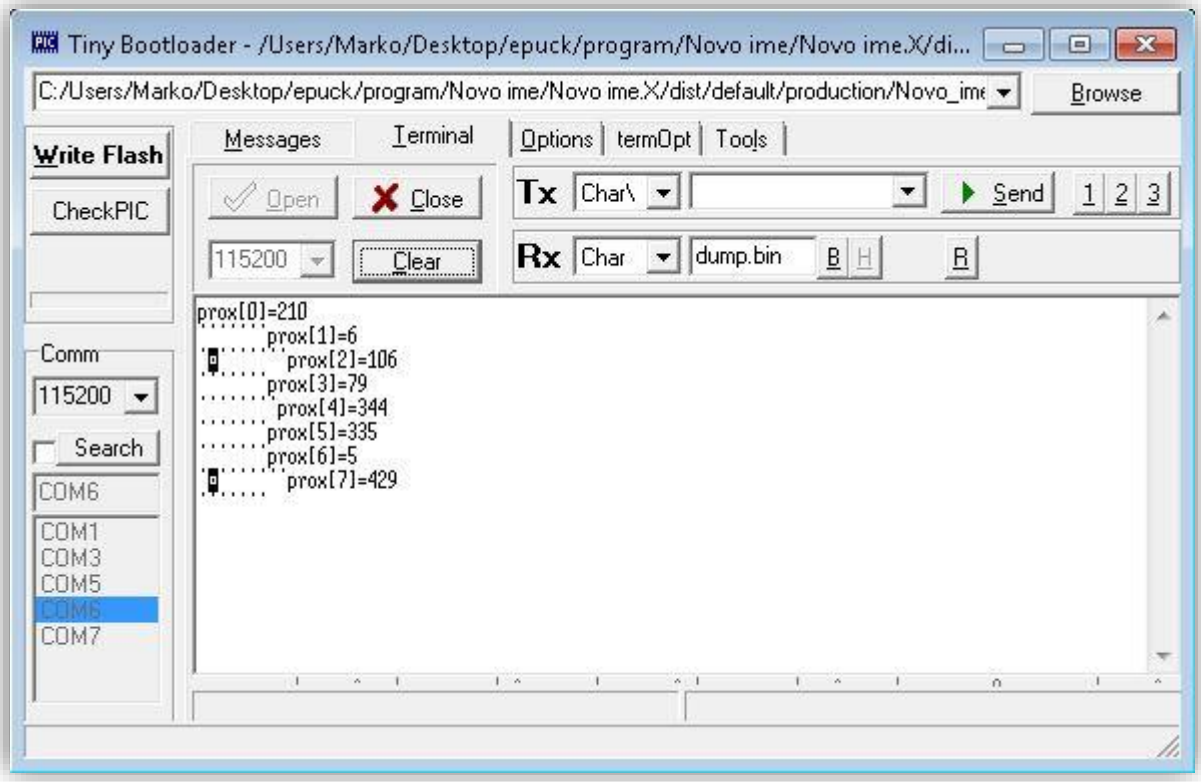
```
#include <stdio.h>
#include <stdlib.h>
#include "p30F6014A.h"

#include "../library/motor_led/e_epuck_ports.h"
#include "../library/motor_led/e_init_port.h"
#include "../library/utility/utility.h"
#include "../library/uart/e_uart_char.h"
#include "../library/a_d/advance_ad_scan/e_acc.h"
#include "../library/a_d/advance_ad_scan/e_ad_conv.h"
#include "../library/a_d/advance_ad_scan/e_micro.h"
#include "../library/a_d/advance_ad_scan/e_prox.h"

int main()
{
	e_init_port();
	e_init_uart1(); // inicijalizacija komunikacije s računalom
	e_init_ad_scan(ALL_ADC); // inicijalizacija analogno-digitalne konverzije

	int accx, accy, accz, mic0, mic1, mic2;
	char buffer[80] = { ' ' };

	int selector = getselector();
	switch (selector) {

	case 0: {
		while (1) {
			accx = e_get_acc(0); // čitanje stanja senzora ubrzanja
			accy = e_get_acc(1);
			accz = e_get_acc(2);
			mic0 = e_get_micro_volume(0); // čitanje glasnoće mikrofona
			mic1 = e_get_micro_volume(1);
			mic2 = e_get_micro_volume(2);

			// pretvaranje string-a u polje character-a (buffer)
			sprintf(buffer, "accx=%d, accy=%d, addz=%d\r\nmic0=%d, mic1=%d,
mic2=%d\r\n", accx, accy, accz, mic0, mic1, mic2);

			// slanje 80 znakova polja u varijabli buffer
			e_send_uart1_char(buffer, 80);

			while (e_uart1_sending()); // čekanje da se sve pošalje
			wait(2000000); // čekanje 2s
		}
		break;}

	case 15: { // čekanje za lakše reprogramiranje robota
		unsigned int i = 0;
		for (i = 0; i < 1000000; i++) {
			asm("nop");
		}
		break;}
	}
	while (1);
	return (0);
}
```



Slika 60. Vrijednosti senzora ubrzanja i mikrofona

5.4. Senzori blizine

```
#include <stdio.h>
#include <stdlib.h>
#include "p30F6014A.h"

#include "../library/motor_led/e_epuck_ports.h"
#include "../library/motor_led/e_init_port.h"
#include "../library/utility/utility.h"
#include "../library/uart/e_uart_char.h"
#include "../library/a_d/advance_ad_scan/e_ad_conv.h"
#include "../library/a_d/advance_ad_scan/e_prox.h"

int main()
{
	e_init_port();
	e_init_uart1();
	e_init_ad_scan(ALL_ADC);

	unsigned int i = 0, prox[8] = { 0 };
	char buffer[80] = { ' ' };

	int selector = getselector();
	switch (selector) {
	case 0: {
		for (i = 0; i < 8; i++){
			prox[i] = e_get_prox(i);   //čitanje senzora blizine
			sprintf(buffer, "prox[%d]=%d\r\n", i, prox[i]);
			e_send_uart1_char(buffer, 20);
			while (e_uart1_sending());
		}
		wait(2000000);
		break;
	}

	case 15: { // čekanje za lakše reprogramiranje robota
		for (i = 0; i < 1000000; i++) {
			asm("nop");
		}
		break;
	}
	}
	while (1);
	return (0);
}
```



Slika 61. Vrijednosti senzora blizine

5.5. Bluetooth komunikacija između dva *e-puck* robota

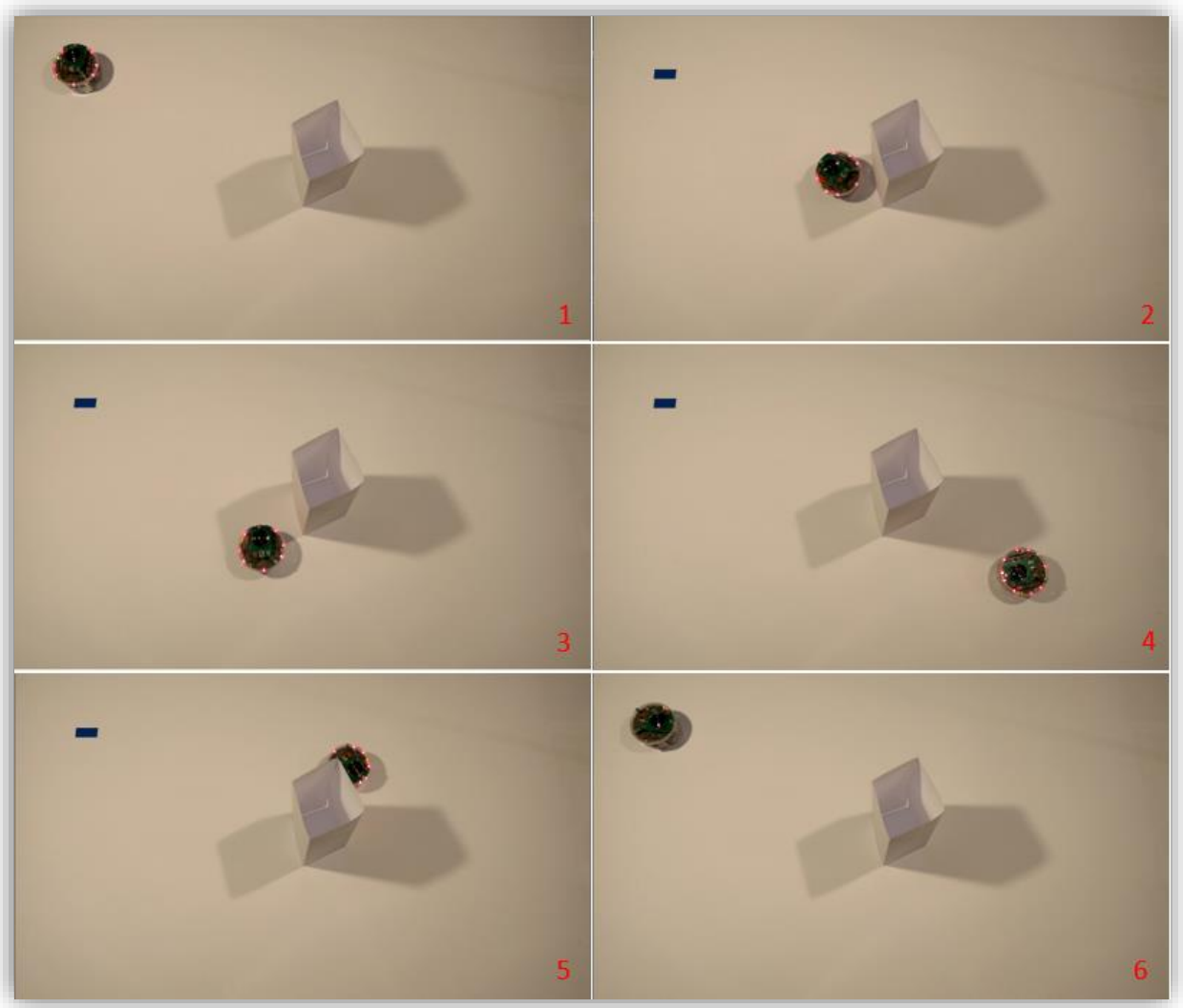
Program demonstrira komunikaciju između dva robota, jedan u *master*, drugi u *slave* ulozi. Nakon što uspostave komunikaciju, *master* robot šalje naredbu *slave* robotu u obliku polja znakova: redni broj LED indikatora te oznaku da li LED indikator treba upaliti ili ugasiti. Npr: "3,1" znači da treba upaliti LED indikator na mjestu 3.

```
#include "p30F6014A.h"
#include "stdio.h"
#include "string.h"
#include <stdlib.h>

#include <motor_led/e_init_port.h>
#include <motor_led/advance_one_timer/e_led.h>
#include <uart/e_uart_char.h>
#include <bluetooth/e_bluetooth.h>
#include <motor_led/e_epuck_ports.h>
#include <utility/utility.h>
#include "memory.h"

// funkcija koja šalje slave-u naredbu
void setSlaveLed(int ledNr, int onOff) {
	char slaveBuffer[10];
	sprintf(slaveBuffer, "%x,%x\r", ledNr, onOff);
	e_bt_send_SPP_data(slaveBuffer, 5);
}

int main() {

	char buffer[BUFFER_SIZE];
	unsigned char selector = 0;
	unsigned char retValue = 0;
	int i = 0;
	unsigned char masterSentData = 0;

	e_init_port();
	e_init_uart1();
	e_init_uart2(BAUD115200);

	selector = getselector();
	while (1) {

		switch (selector) {
		case 0: // master
			if (masterSentData == 0) { // pošalji naredbe samo jednom
				masterSentData = 1;
				do {
					e_set_body_led(1);
					// spoji se na prvi detektirani e-puck
					retValue = e_bt_connect_epuck();
					e_set_body_led(0);
				} while (retValue != 0);

				for (i = 0; i<7; i++) {
					e_set_body_led(1);
					// upali LED indikator slave-u
					setSlaveLed(i, 1);
					e_set_body_led(0);
```

```
                                        // čekaj odgovor od slave-a
                                        while (!e_bt_recv_SPP_data(buffer));
                                        wait(2000); // čekaj 2s
                                }

                                // važno - otpusti bluetooth resurse
                                e_bt_release_SPP_link();
                                e_bt_restore_pin();
                        }
                        break;

                case 1: // slave
                        while (1) {
                                // počisti zadnju naredbu od mastera
                                memset(buffer, 0x0, 50);
                                // čekaj novu naredbu od mastera
                                e_bt_recv_SPP_data(buffer);

                                // pročitaj naredbu od mastera
                                int nrLed = buffer[0] - 48; // pretvori char u int
                                int onOff = buffer[2] - 48;

                                // upali/ugasi odabrani LED indikator
                                e_set_led(nrLed, onOff);
                        }
                        break;

                // otpusti bluetooth resurse ako se dogodio neki problem u komunikaciji
                case 2:
                        e_bt_release_SPP_link();
                        e_bt_restore_pin();
                        break;
                }

        }

        while (1);
        return 0;

}
```

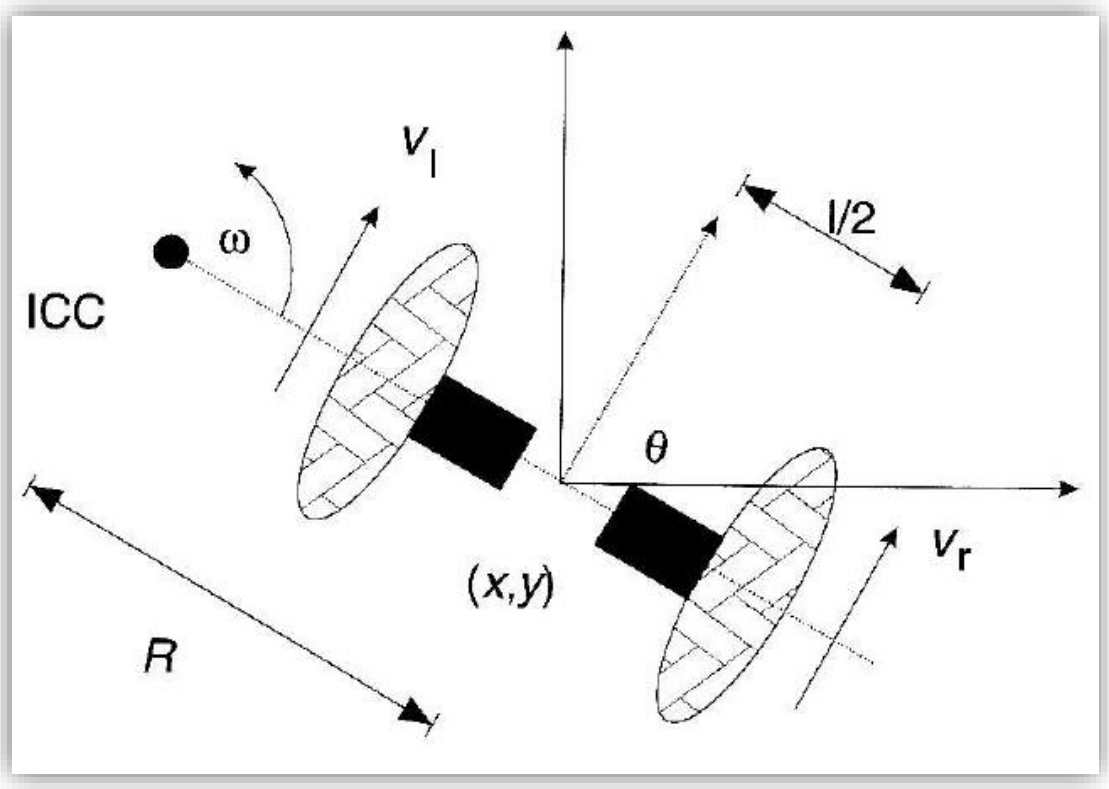
5.6. Zaobilaženje prepreke

Robot se nalazi na početnoj poziciji, cilj mu je doći do krajnje pozicije (krajnja pozicija nalazi se 600mm desno i 300mm prema dolje), a ukoliko robot dođe do prepreke, usporit će i pratiti prepreku dok mu više ne smeta. Nakon svega robot se vraća u početnu poziciju (Slika 62).



Slika 62. Zaobilaženje prepreke: 1) robot u početnoj poziciji; 2) robot kada uoči prepreku; 3) robot kada se dovoljno udalji od prepreke; 4) robot u krajnjoj poziciji; 5) robot u povratku zaobilazi prepreku; 6) robot kako se vratio na početak.

5.6.1. Kinematika robota



Slika 63. Kinematika mobilnog robota s diferencijalnim pogonom [13]

Jednadžbe za upravljanje translacijskom (v) i kutnom brzinom (ω) robota:

$$\dot{x} = v\cos(\theta) \quad (3)$$

$$\dot{y} = v\sin(\theta) \quad (4)$$

$$\dot{\theta} = \omega \quad (5)$$

Obzirom da ne možemo direktno upravljati s v *i* ω, koristimo dodatne jednadžbe:

$$\dot{x} = \frac{1}{2}(v_r + v_l)\cos(\theta) \quad (6)$$

$$\dot{y} = \frac{1}{2}(v_r + v_l)\sin(\theta) \quad (7)$$

$$\dot{\theta} = \frac{1}{l}(v_r - v_l) \quad (8)$$

Kada izjednačimo izraze (3), (4), (5) s izrazima (6), (7), (8):

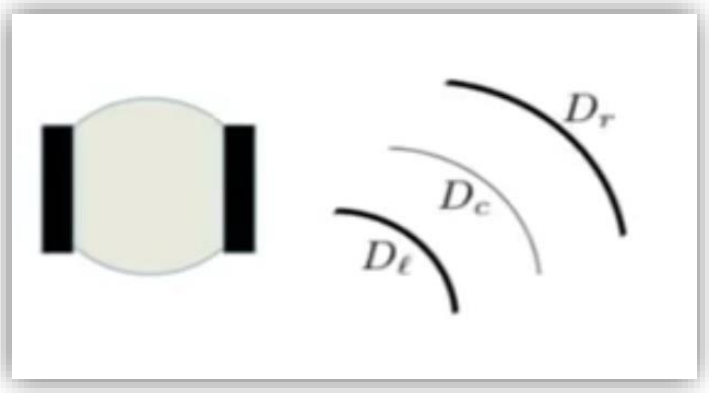
$$v_r = v + \frac{\omega l}{2} \quad (9)$$

$$v_l = v - \frac{\omega l}{2} \quad (10)$$

$$\omega = \frac{1}{l}(v_r - v_l) \quad (11)$$

5.6.2. Proračun položaja robota

Robot nema senzore na kotačima, ali posjeduje koračne motore te praćenjem koraka možemo pratiti položaj i orijentaciju robota u prostoru.



Slika 64. Luk kotača pri skretanju [14]

$$D_c = \frac{D_l + D_r}{2} \quad (12)$$

$$x' = x + D_c \cos(\theta) \quad (13)$$

$$y' = y + D_c \sin(\theta) \quad (14)$$

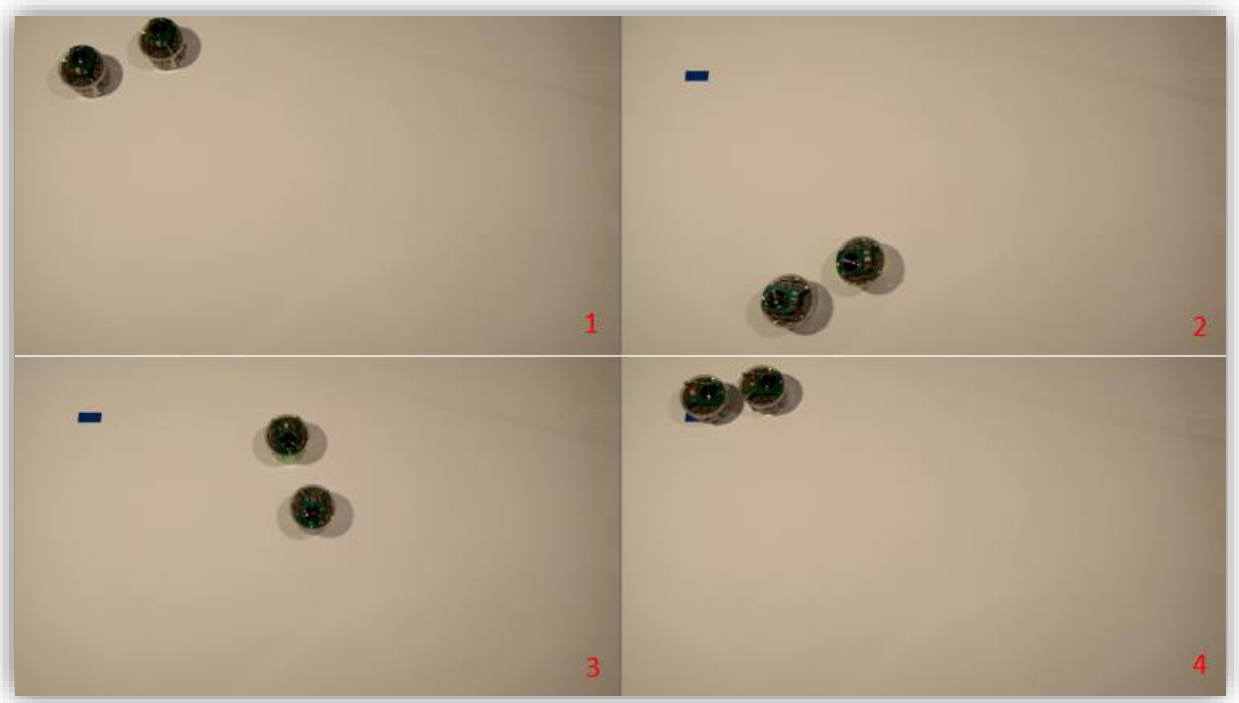
$$\theta' = \theta + \frac{D_r - D_l}{l} \quad (15)$$

$$\Delta N_j = N_j' - N_j, \qquad j = r \; ili \; l \quad (16)$$

$$D_j = 2\pi r \frac{\Delta N_j}{N_j} \quad (17)$$

5.7. Praćenje robota

Vođa radi pravokutnu kretnju i vraća se na početak, drugi ga robot prati.



Slika 65. Praćenje robota: 1) početna pozicija; 2) i 3) vođa robot putuje, drugi robot ga prati; 4) krajnja pozicija

5.8. Prepoznavanje boja

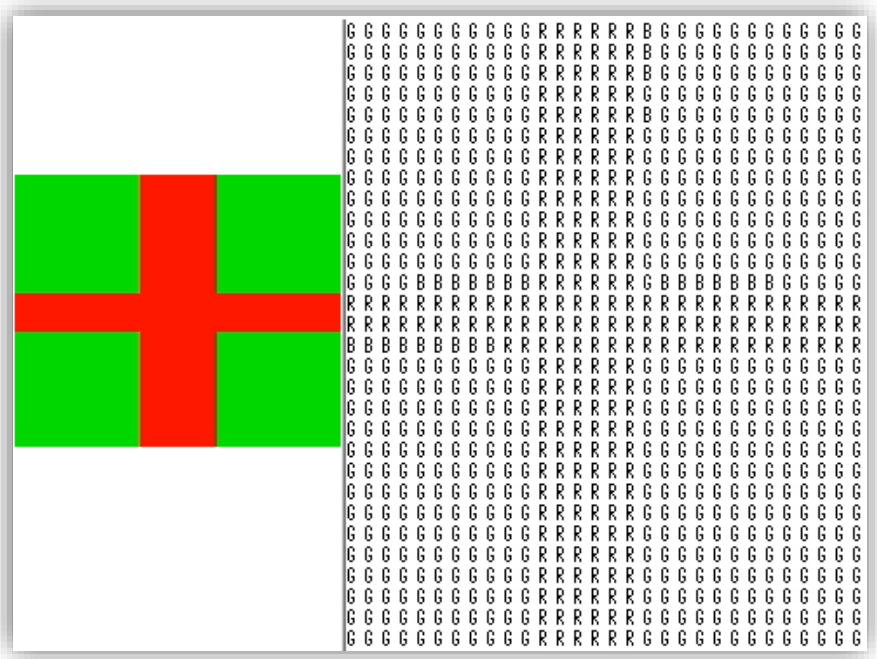
Robot je usmjeren prema ekranu računala i mijenja koje su *LED* diode upaljene u odnosu na boju na ekranu (za crvenu boju upaljene su *LED0*, *LED1*, *LED7*, za zelenu je upaljena *body* zelena dioda, i za plavu su upaljene *LED3*, *LED4*, *LED5*).

Slika 66. Prepoznavanje boja: robot je uspješno prepoznao 1) zelenu boju, 2) plavu boju i 3) crvenu boju

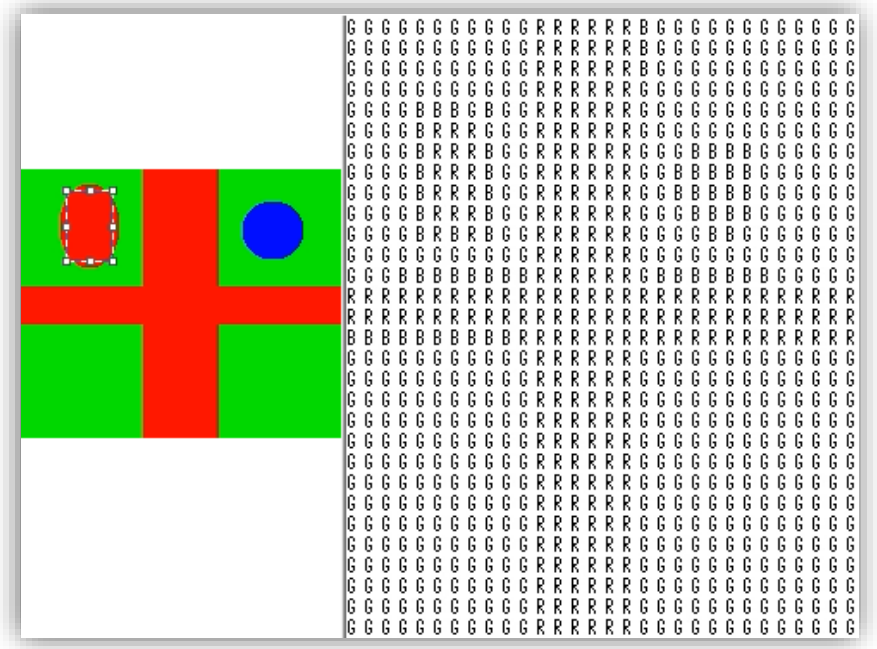
5.9. Obrada i prikazivanje slike s ekrana računala

Robot je usmjeren prema ekranu računala u centar sjecišta crvenih crta i ispisuje sliku u tekstualnom obliku. Za svaki piksel ispiše dominantnu boju (R - crveno, G - zeleno, P - plavo).

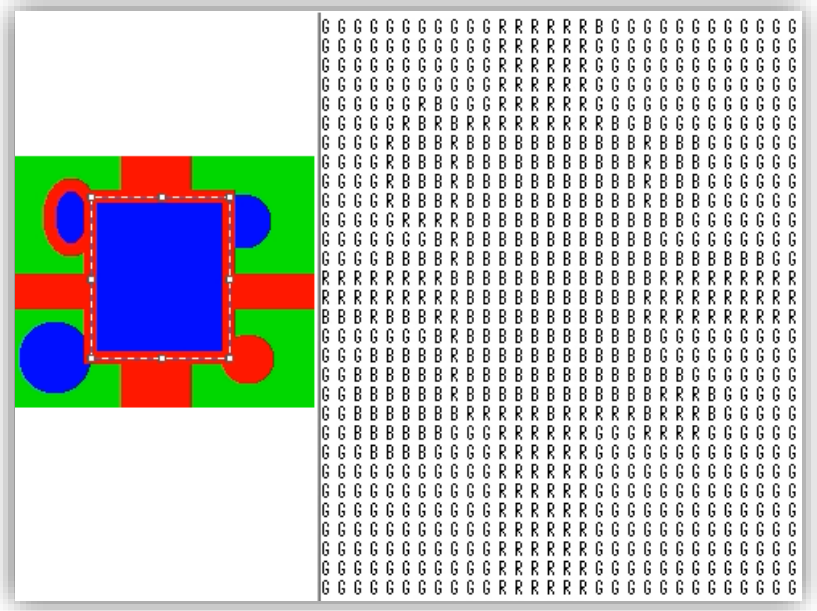
Slika i ispis za tu sliku mogu se vidjeti na slikama koje slijede (Slika 67, Slika 68, Slika 69)



Slika 67. Robot je uspješno prepoznao boje na slici



Slika 68. Robot je uspješno prepoznao boje na slici



Slika 69. Robot je uspješno prepoznao boje na slici

ZAKLJUČAK

U ovom diplomskom radu istraživane su prednosti i mane *e-puck* robota kao edukacijske platforme za mobilne robote. Glavni nedostatak ovih robota jest nedostatak dokumentacije, što čini početak rada i postavljanje radne okoline za njihovo programiranje izrazito teškim. Također, robot zna imati neočekivana ponašanja programske prirode (ne ponaša se sukladno s programskim kodom, ponovno gašenje/paljenje rješava problem), kao i manje hardverske nedostatke (npr. gubitak kontakta s baterijom uslijed vibracija).

Hardverski dio robota je sasvim dovoljan za obradu podataka sa senzora i upravljanje izlaznim uređajima, no vrlo ograničen u radu s kamerom. Veliku prednost daje mu bežična *bluetooth* komunikacija, koja omogućava spajanje s računalom, a samim time korištenje puno jačih resursa za obrađivanje podataka te donošenje upravljačkih odluka. *Bluetooth* komunikacija također omogućava spajanje i međusobnu komunikaciju robota kako bi se mogli kreirati kompleksniji programski algoritmi.

Unatoč tome što je *e-puck* platforma otvorena za razvoj (*open-source*), unutar zadnje četiri godine ne postoji gotovo nikakav razvoj, a postojeća gotova rješenja su često nestabilna.

No, iako *e-puck* mobilni robot ima svojih nedostataka, kada se dovoljno upozna s platformom, sam razvoj programskog koda i upravljačkih algoritama je poprilično zabavan te ispunjava svoju *edukacijsku* svrhu.

Nadam se da će ovaj rad omogućiti budućim korisnicima lako postavljanje radne okoline i jednostavno upoznavanje programske platforme kako bi mogli brzo početi s pisanjem programskog koda i stvaranjem upravljačkih algoritama.

PRILOZI

I. Programski kod

II. DVD-R disk

LITERATURA

[1] *https://en.wikipedia.org/wiki/E-puck_mobile_robot*, pristupljeno studeni 2016.

[2] *http://www.eu-robotics-sme.org/gctronic/*, pristupljeno studeni 2016.

[3] *http://www.e-puck.org/*, pristupljeno studeni 2016.

[4] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stephane Magnenat, Jean-Christophe Zufferey, Dario Floreano, Alcherio Martinoli, "The e-puck, a Robot Designed for Education in Engineering", *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, vol. 1, num. 1, p. 59-65, Portugal: Instituto Politécnico de Castelo Branco*, 2009.

[5] *http://www.gctronic.com/doc/index.php/E-Puck*, pristupljeno studeni 2016.

[6] *NXP MMA7260QT*, pristupljeno studeni 2016.

[7] *https://www.theimagingsource.com/support/documentation/ic-imaging-control-cpp/PixelformatRGB565.htm*, pristupljeno studeni 2016.

[8] *https://wiguna149.wordpress.com*, pristupljeno studeni 2016.

[9] *https://www.cyberbotics.com/item?id=14*, pristupljeno studeni 2016.

[10] https://hr.wikipedia.org/wiki/Logi%C4%8Dki_sklopovi, pristupljeno studeni 2016.

[11] https://www.cyberbotics.com/e-puck, pristupljeno studeni 2016.

[12] https://www.arduino.cc/en/Tutorial/ArduinoToBreadboard, pristupljeno studeni 2016.

[13] Dudek, G., Jenkin, M.: Computational Principles of Mobile Robotics, Cambridge University Press, Cambridge , 2010.

[14] https://www.youtube.com/watch?v=aE7RQNhwnPQ, pristupljeno studeni 2016.

PROGRAMSKI KOD

Zaobilaženje prepreke

(selektor mora biti na poziciji 0):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "p30F6014A.h"
#include "../library/motor_led/e_epuck_ports.h"
#include "../library/motor_led/e_init_port.h"
#include "../library/utility/utility.h"
#include "uart/e_uart_char.h"
#include "I2C/e_I2C_master_module.h"
#include "../library/I2C/e_I2C_protocol.h"
#include "../library/motor_led/advance_one_timer/fast_agenda/e_led.h"
#include "../library/motor_led/advance_one_timer/fast_agenda/e_motors.h"
#include "../library/motor_led/advance_one_timer/fast_agenda/e_agenda_fast.h"
#include "a_d/advance_ad_scan/e_ad_conv.h"
#include "a_d/advance_ad_scan/e_prox.h"

#define PI 3.14159265

unsigned char buffer[50];

long Nleftold = 0, Nrightold = 0;
double x = 0, y = 0, fi = 0, xold = 0, yold = 0, fiold = 0;
void updateposition(void) { //Funckija za praćenje pozicije i orijentacije x, y, Fi
	long Nleft, Nright;
	Nleft = e_get_steps_left();
	Nright = e_get_steps_right();
	float Dc, Dl, Dr;
	Dl = PI * 41 * (Nleft - Nleftold) / 1000;
	Dr = PI * 41 * (Nright - Nrightold) / 1000;
	Dc = (Dl + Dr) / 2;
	x = xold + Dc*cos(fi);
	y = yold + Dc*sin(fi);
	fi = fiold + (Dr - Dl) / 53;

	xold = x;
	yold = y;
	fiold = fi;

	Nleftold = Nleft;
	Nrightold = Nright;
}

int xd = 0, yd = 0;
float v0 = 0;
float ufi = 0, Efi = 0;
float fid, efi;
float kPfi = 25, kIfi = 0.01;
void PIDfi(void) {  //PI regulator za omegu
	updateposition();
	fid = atan((yd - y) / (xd - x));  //Računanje kuta cilja
	if (x > xd) {
		fid = fid + PI;
		fid = atan2(sin(fid), cos(fid));
	}
	efi = fid - fi;	//Računanje pogreške kuta
	efi = atan2(sin(efi), cos(efi));  //Osiguranje da je pogreška unutar granica
[-Pi, Pi]
```

```
	Efi = Efi + efi;
	ufi = kPfi*efi + kIfi*Efi;
}

float Ev0 = 0, ev;
void PIDv0(void) {  //PI regulator za brzinu
	float deltax, deltay;
	float kPv0 = 1, kIv0 = 0.001;
	updateposition();
	deltax = xd - x;    //Računanje razlike udaljenosti
	deltay = yd - y;
	ev = sqrt(pow(deltax, 2) + pow(deltay, 2));    //Računanje pogreške
	Ev0 = Ev0 + ev;
	v0 = kPv0*ev + kIv0*Ev0;
	if (v0 >70) { //Podašavanje maksimalne brzine
		v0 = 70;
	}
}

void drive(float v, float omega) {	//Preračunavanje brzina u korake/s
	float vl, vr;
	int Nvl, Nvr;
	vl = v - (omega * 53) / (41);
	vr = v + (omega * 53) / (41);
	Nvl = 1000 / 41 / PI*vl;
	Nvr = 1000 / 41 / PI*vr;
	if (Nvl > 1001) {
		Nvl = 1000;
	}
	if (Nvr > 1001) {
		Nvr = 1000;
	}
	e_set_speed_left(Nvl);
	e_set_speed_right(Nvr);
}

float fio = 0;
int flago = 0;
void obstacles_kut(void) { //Definiranje kuta prepreke
	fio = 0;
	flago = 0;
	int i, prox[8];
	for (i = 0; i<8; i++) {
		prox[i] = e_get_calibrated_prox(i);
		wait(10);
	}
	if (prox[0] >100 && prox[7]>200) {
		fio = 0;
		flago = 1;
	}
	else if (prox[0] > 100 && prox[1] >200) {
		fio = -0.5236;
		flago = 1;
	}
	else if (prox[7] > 200 && prox[6] >200) {
		fio = 0.5236;
		flago = 1;
	}
	else if (prox[1] > 200 && prox[2] >180) {
		fio = -1.1781;
		flago = 1;
	}
	else if (prox[6] > 200 && prox[5] >200) {
		fio = 1.1781;
		flago = 1;
```

```
	}
	else if (prox[3] > 100 && prox[4] > 200) {
		fio = 3.14159;
		flago = 1;
	}
	else if (prox[0] > 100) {
		fio = -0.2618;
		flago = 1;
	}
	else if (prox[7] > 210) {
		fio = 0.2618;
		flago = 1;
	}
	else if (prox[1] > 200) {
		fio = -0.7854;
		flago = 1;
	}
	else if (prox[6] > 200) {
		fio = 0.7854;
		flago = 1;
	}
	else if (prox[2] > 80) {
		fio = -1.5708;
		flago = 1;
	}
	else if (prox[5] > 100) {
		fio = 1.5708;
		flago = 1;
	}
	else if (prox[3] > 150) {
		fio = -2.618;
		flago = 1;
	}
	else if (prox[4] > 200) {
		fio = 2.618;
		flago = 2;
	}
}
void obstacles_reg(void) { //Proračun kutne brzine u slučaju prepreke
	float fiop, fiom;
	do {
		if (sqrt(fio*fio) < 1.5708) {
			fiop = fio + PI / 2;
			fiom = fio - PI / 2;
			if (sqrt(fiop*fiop) < sqrt(fiom*fiom)) { //Provjera u kojem je
smjeru manji kut za zaobilaženje prepreke
				fid = fiop + fi;
			}
			else {
				fid = fiom + fi;
			}
			v0 = 10;
		}
		else if (sqrt(fio*fio) >= 1.5708) {
			v0 = 35;
		}
		updateposition();
		efi = fid - fi;
		efi = atan2(sin(efi), cos(efi));  //Osiguranje da je pogreška unutar
granica [-Pi, Pi]
		ufi = 35 * efi;
		drive(v0, ufi);
		updateposition();
		obstacles_kut();
		wait(100000);
```

```
	} while (flago != 0);	//Izvršavanje petlje, sve dok senzori detektiraju
prepreku
}



int main(void) {
	e_init_port();
	e_init_uart1();
	e_i2c_init();
	e_init_ad_scan(ALL_ADC);

	e_init_motors();
	e_start_agendas_processing();
	e_calibrate_ir();
	e_set_speed_left(0);
	e_set_speed_right(0);
	e_set_steps_left(0);
	e_set_steps_right(0);

	int selector = getselector();
	sprintf(buffer, "Selector position: %d\r\n", selector);
	e_send_uart1_char(buffer, strlen(buffer));
	while (e_uart1_sending());

	switch (selector) {

	case 0: {
		xd = 300;	//Postavljanje koordinate cilja
		yd = 600;
		v0 = 60;
		updateposition();
		int i;
		for (i = 0; i<8; i++) {	//Paljenje LED dioda na prstenu
			e_set_led(i, 1);
		}
		wait(1000);
		e_calibrate_ir();	//Kalibracija senzora blizine
		wait(100000);
		do {
			obstacles_kut();
			PIDv0();
			PIDfi();
			drive(v0, ufi);
			if (flago == 1) {
				obstacles_reg();
			}
			wait(200000);
		} while (ev > 10);	//Izvršavanje petlje sve dok je pogreška veća od 10

		xd = 0;	//Postavljanje koordinata na početnu poziciju
		yd = 0;
		v0 = 60;
		updateposition();
		do {
			obstacles_kut();
			PIDv0();
			PIDfi();
			drive(v0, ufi);
			if (flago == 1) {
				obstacles_reg();
			}
			wait(200000);
		} while (ev > 10);	//Izvršavanje petlje sve dok je pogreška veća od 10
```

```
                e_set_speed_left(0); //Gašenje motora
                e_set_speed_right(0);
                for (i = 0; i<8; i++) {     //Gašenje LED dioda
                        e_set_led(i, 0);
                }
                break;
        }

        case 15: {
                int i;
                for (i = 0; i < 1000000; i++) {
                        asm("nop");
                }
                break;
        }
        }
        while (1);
}
```

Prepoznavanje boja, praćenje svjetla, obrada i prikazivanje slike s ekrana računala

Selektor na poziciji 0 za prepoznavanje boja, 1 za praćenje svjetla, 2 za obradu i prikazivanje slike s ekrana računala.

```
#include "p30F6014A.h"
#include "stdio.h"
#include "string.h"
#include "math.h"
#include <time.h>
#include <codec/e_sound.h>
#include <motor_led/e_init_port.h>
#include <motor_led/advance_one_timer/e_led.h>
#include <motor_led/advance_one_timer/e_motors.h>
#include <motor_led/advance_one_timer/e_agenda.h>
#include <motor_led/advance_one_timer/e_remote_control.h>
#include <camera/fast_2_timer/e_poxxxx.h>
#include <uart/e_uart_char.h>
#include <a_d/advance_ad_scan/e_ad_conv.h>
#include <a_d/advance_ad_scan/e_prox.h>
#include <a_d/advance_ad_scan/e_acc.h>
#include <a_d/advance_ad_scan/e_micro.h>
#include <bluetooth/e_bluetooth.h>
#include <motor_led/e_epuck_ports.h>
#include <acc_gyro/e_lsm330.h>
#include <I2C/e_I2C_protocol.h>
#include "memory.h"

#define PI 3.14159265358979

char unsigned buffer[1800];
char temp[100];

char blue = 0, red = 0, green = 0;
void colors(char lobyte, char hibyte) {  //Pretvorba iz RGB565 u RGB
	red = 0;
	green = 0;
	blue = 0;
	red = (hibyte & 0xF8) >> 3;
	green = ((hibyte & 0x7) << 3) | ((lobyte & 0xE0) >> 5);
	blue = lobyte & 0x1f;
}

int flag2 = 0;
float r = 0, g = 0, b = 0;
void boje(void) {    //Uzorkovanje slike i obrada podataka
	int rr = 0, gg = 0, bb = 0;
	int cam_height = 20, cam_width = 20;
	rr = 0;
	gg = 0;
	bb = 0;
	if (flag2 == 0) {
		e_poxxxx_config_cam((ARRAY_WIDTH - cam_width) / 2, (ARRAY_HEIGHT -
cam_height) / 2, cam_width, cam_height, 1, 1, RGB_565_MODE);   //Postavke kamere
		e_poxxxx_set_awb_ae(0, 0); //Dodatne postavke kamere
		e_poxxxx_write_cam_registers(); //Zapisivanje postavki u kameru
		flag2 = 1;
	}
	e_poxxxx_launch_capture(&buffer[0]);    //Čitanje piksela i spremanje u buffer
	while (!e_poxxxx_is_img_ready());    //Čekanje da prethodna funkcija sve izvrši
	int i;
	for (i = 0; i<cam_width*cam_height * 2; i = i + 2) {  //Zbrajanje pojedinih boja
svih piksela
```

```
			colors(buffer[i + 1], buffer[i]);
			rr = rr + red;
			gg = gg + green;
			bb = bb + blue;
		}
		r = (rr / (31.0*cam_width*cam_height))*100.0; //Preračunavanje inteziteta boja u
postotak
		g = (gg / (63.0*cam_width*cam_height))*100.0;
		b = (bb / (31.0*cam_width*cam_height))*100.0;
}

int main(void) {
		e_init_port();
		e_init_uart1();
		e_i2c_init();
		e_init_ad_scan(ALL_ADC);

		e_init_motors();
		e_start_agendas_processing();
		e_set_steps_left(0);
		e_set_steps_right(0);

		int selector = getselector();
		sprintf(temp, "Selector position: %d\r\n", selector);
		e_send_uart1_char(temp, strlen(temp));
		while (e_uart1_sending());

		switch (selector) {

		case 0: {	//Prepoznavanje boja
			e_poxxxx_init_cam();
			while (1) {
				boje(); //Funkcija za uzorkovanje i obradu slike
				e_led_clear();  //Gašenje LED dioda
				e_set_body_led(0);
				if (r > 5 + g && r > 5 + b) { //Paljenje LED dioda ako je određena
boja veća za 5%
					e_set_led(0, 1);
					e_set_led(1, 1);
					e_set_led(7, 1);
				}
				else if (g > 5 + r && g > 5 + b) {
					e_set_body_led(1);
				}
				else if (b > 5 + r && b > 5 + g) {
					e_set_led(3, 1);
					e_set_led(4, 1);
					e_set_led(5, 1);
				}
			}

			break;
		}

		case 1: {	//Praćenje svjetla
			e_poxxxx_init_cam();
			e_poxxxx_config_cam((ARRAY_WIDTH - 8) / 2, (ARRAY_HEIGHT - 60 * 8) / 2,
8, 60 * 8, 8, 8, GREY_SCALE_MODE);   //Postavke kamere
			e_poxxxx_set_mirror(1, 0); //Dodatne postavke kamere
			e_poxxxx_set_awb_ae(0, 0); //Dodatne postavke kamere
			e_poxxxx_write_cam_registers(); //Zapisivanje postavki u kameru

			while (1) {
				e_poxxxx_launch_capture(&buffer[0]);    //Čitanje piksela s kamere
u buffer
```

```
                    while (!e_poxxxx_is_img_ready());
                    int i = 0, max_light = 0, light_location = 0;
                    for (i = 30; i<60; i = i + 1) { //Traženje najsvjetlijeg piksela
                          if (buffer[i] > max_light) {
                                max_light = buffer[i];
                                light_location = i;
                          }
                    }
                    for (i = 29; i >= 0; i = i - 1) { //Traženje najsvjetlijeg piksela
                          if (buffer[i] > max_light) {
                                max_light = buffer[i];
                                light_location = i;
                          }
                    }

                    e_set_speed_left(10 * (light_location - 30));   //Promjena kutne
brzine robota prema najsvjetlijem pikselu
                    e_set_speed_right(-10 * (light_location - 30));
             }

             break;
      }

      case 2: {    //Obrada i prikazivanje slike s ekrana računala
             e_poxxxx_init_cam();
             e_poxxxx_config_cam((ARRAY_WIDTH - 30 * 8) / 2, (ARRAY_HEIGHT - 30 * 8) /
2, 30 * 8, 30 * 8, 8, 8, RGB_565_MODE);    //Postavke kamere
             e_poxxxx_set_mirror(1, 0); //Dodatne postavke kamere
             e_poxxxx_set_awb_ae(0, 0); //Dodatne postavke kamere
             e_poxxxx_write_cam_registers(); //Zapisivanje postavki u kameru
             wait(1000000);
             while (1) {
                    e_poxxxx_launch_capture(&buffer[0]);    //Čitanje piksela s kamere
u buffer
                    while (!e_poxxxx_is_img_ready());
                    int i = 0, counter = 0, j = 0;
                    float red2 = 0, blue2 = 0, green2 = 0;
                    char bla[30 * 30];
                    for (i = 0; i<30 * 30 * 2; i = i + 2) {     //Obrada podataka iz
buffera
                          colors(buffer[i + 1], buffer[i]);
                          red2 = (red / 31.0) * 100;
                          blue2 = (blue / 31.0) * 100;
                          green2 = (green / 63.0) * 100;
                          if ((red2>blue2 + 5) && (red2>green2 + 5)) {
                                bla[counter] = 'R';
                          }
                          else if ((green2>red2 + 5) && (green2>blue2 + 10)) {
                                bla[counter] = 'G';
                          }
                          else {
                                bla[counter] = 'B';
                          }
                          counter = counter + 1;
                    }
                    for (i = 0; i < 30; i++) {      //Ispisivanje slike na ekran
računala
                          for (j = 0; j < 30; j++) {
                                sprintf(temp, "%c ", bla[30 * j + i]);
                                e_send_uart1_char(temp, strlen(temp));
                                while (e_uart1_sending());
                          }
                          sprintf(temp, "\r\n");
                          e_send_uart1_char(temp, strlen(temp));
                          while (e_uart1_sending());
```

```
                        }
                        wait(3000000);  //Čekanje, kako se slika ne bi prečesto mijenjala
                        for (i = 0; i<30; i++) {    //Ispisivanje novog reda
                               sprintf(temp, "\r\n");
                               e_send_uart1_char(temp, strlen(temp));
                               while (e_uart1_sending());
                        }
                }
                break;
        }
        case 15:
                wait(2000000);
        }
        while (1);
}
```

Praćenje robota

Selektor na poziciji 0 za robota koji vodi, selektor na poziciji 1 za robota koji prati.

```
#include <e_ad_conv.h>
#include <e_init_port.h>
#include <e_epuck_ports.h>
#include <e_uart_char.h>
#include <e_led.h>
#include <e_led.h>
#include <e_motors.h>
#include <e_agenda.h>
#include <stdio.h>
#include <stdlib.h>
#include <ircom.h>
#include <btcom.h>
#include <math.h>
#include <string.h>

int selector;
long int lastClock;
char buffer[200];
#define PI 3.14159265

void wait(long num) {
	long i;
	for (i = 0; i<num; i++);
}

long Nleftold = 0, Nrightold = 0;
double x = 0, y = 0, fi = 0, xold = 0, yold = 0, fiold = 0;
void updateposition(void) {//Funckija za praćenje pozicije i orijentacije x, y, Fi
	long Nleft, Nright;
	Nleft = e_get_steps_left();
	Nright = e_get_steps_right();
	float Dc, Dl, Dr;
	Dl = PI * 41 * (Nleft - Nleftold) / 1000;
	Dr = PI * 41 * (Nright - Nrightold) / 1000;
	Dc = (Dl + Dr) / 2;
	x = xold + Dc*cos(fi);
	y = yold + Dc*sin(fi);
	fi = fiold + (Dr - Dl) / 53;

	xold = x;
	yold = y;
	fiold = fi;

	Nleftold = Nleft;
	Nrightold = Nright;
}

int xd = 0, yd = 0;
float v0 = 0;
float ufi = 0, Efi = 0;
float fid, efi;
float kPfi = 35, kIfi = 0.01;
void PIDfi(void) {  //PI regulator za omegu
	updateposition();
	fid = atan((yd - y) / (xd - x));
	if (x > xd) {
		fid = fid + PI;
		fid = atan2(sin(fid), cos(fid));
	}
	efi = fid - fi;
```

```
	efi = atan2(sin(efi), cos(efi));
	Efi = Efi + efi;
	ufi = kPfi*efi + kIfi*Efi;
}
float irdistance = 0, irangle = 0;
int irdistance2 = 0;

float Ev0 = 0, ev;
int v0max = 60;
void PIDv0(void) {  //PI regulator za brzinu
	float deltax, deltay;
	float kPv0 = 1, kIv0 = 0;
	updateposition();
	deltax = xd - x;
	deltay = yd - y;
	ev = sqrt(pow(deltax, 2) + pow(deltay, 2));
	Ev0 = Ev0 + ev;
	v0 = kPv0*ev + kIv0*Ev0;
	if (v0 >v0max) {
		v0 = v0max;
	}
}


void drive(float v, float omega) {	//Preračunavanje brzina u korake/s
	float vl, vr;
	int Nvl, Nvr;
	vl = v - (omega * 53) / (41);
	vr = v + (omega * 53) / (41);
	Nvl = 1000 / 41 / PI*vl;
	Nvr = 1000 / 41 / PI*vr;
	if (Nvl > 1001) {
		Nvl = 1000;
	}
	if (Nvr > 1001) {
		Nvr = 1000;
	}
	e_set_speed_left(Nvl);
	e_set_speed_right(Nvr);
}

int getselector()
{
	return SELECTOR0 + 2 * SELECTOR1 + 4 * SELECTOR2 + 8 * SELECTOR3;
}

void irget(void) {  //Funkcija za računanje udaljenosti i kuta drugog robota
	ircomDisableProximity();
	lastClock = ircomGetTime();
	IrcomMessage msg;
	ircomPopMessage(&msg);
	if (msg.error == 0) {
		irdistance = msg.distance;
		irangle = msg.direction;
		irangle = atan2(sin(irangle), cos(irangle));
	}
	ircomEnableProximity();
}

void irsend(void) { //Paljenje, gašenje senzora blizine, kako bi 'irget()' mogao
detektirati svjetlo i izračunati kut
	int i;
	ircomStart();
	ircomEnableContinuousListening();
	ircomListen();
```

```
	ircomResetTime();
	ircomDisableProximity();
	e_set_body_led(1);
	for (i = 0; i<2; i++) {
		lastClock = ircomGetTime();
		ircomSend(1);
		while (ircomSendDone() == 0);
	}
	e_set_body_led(0);
	ircomEnableProximity();
	ircomStopListening();
	ircomStop();
}

int main()
{
	e_init_port();
	e_init_ad_scan();
	e_init_uart1();
	e_led_clear();
	e_init_motors();
	e_start_agendas_processing();
	e_calibrate_ir();

	int selector = getselector();
	sprintf(buffer, "Selector position: %d\r\n", selector);
	e_send_uart1_char(buffer, strlen(buffer));
	while (e_uart1_sending());
	wait(10000000);
	switch (selector) {

	case 1: {	//robot koji vodi
		v0max = 60;
		xd = 300;	//Kretanje u različite točke
		yd = 0;
		v0 = 60;
		int i;
		for (i = 0; i<4; i++) {
			if (i == 0) {
				xd = 300;
				yd = 0;
			}
			else if (i == 1) {
				xd = 300;
				yd = 300;
			}
			else if (i == 2) {
				xd = 0;
				yd = 300;
			}
			else if (i == 3) {
				xd = 0;
				yd = 0;
			}
			updateposition();
			wait(100000);
			do {
				PIDv0();
				PIDfi();
				drive(v0, ufi);
				irsend();
				wait(100000);
			} while (ev > 10);
		}
```

```
		e_set_speed_left(0);
		e_set_speed_right(0);
		while (1) {
			irsend();
			wait(500000);
		}
		for (i = 0; i<8; i++) {
			e_set_led(i, 0);
		}
		break;
	}

	case 2: {    //robot koji prati
		ircomStart();
		ircomEnableContinuousListening();
		ircomListen();
		ircomResetTime();
		v0max = 100;
		while (1) {
			irget();
			if (irdistance > 9) {
				irdistance2 = irdistance - 9;
			}
			else {
				irdistance2 = 0;
			}
			v0 = 18 * irdistance2;
			if (v0 >90) {
				v0 = 90;
			}
			ufi = 20 * irangle;
			drive(v0, ufi);
			wait(50000);

		}
		break;
	}

	case 15: {
		wait(2000000);
		break;
	}
	}
	while (1);
}
```