UNIVERZITET U BEOGRADU

MATEMATIČKI FAKULTET

Master rad

Algoritmi za praćenje linije za Arduino robota sa IC senzorima

Mentor:

Prof. dr Miodrag Živković

Student:

Aleksandar Milanović

Beograd, avgust 2016. godine

Sadržaj

1. Uvod

1.1 Aktuelnost i značaj teme

Ideja o automatima, autonomnim mašinama koje obavljaju predefinisanu sekvencu poslova, javlja se još u najranijim kulturama. Termin *robot* se prvi put pojavljuje u literaturi da opiše automate u predstavi R.U.R. češkog pisca Karela Čapeka.[1] Prvi humanoidni roboti se pojavljuje dvadesetih i tridesetih godina 20. veka. Humanoidni robot Elektro je prvi put predstavljen 1939. godine u Njujorku na svetskoj izložbi. Elektro je mogao da hoda kada mu se izda glasovna komanda, priča oko sedam stotina reči, pomera svoju glavu i ruke i vrši složenije operacije kao što je pušenje cigareta i naduvanje balona.[2]

Moderni elektronski autonomni roboti se pojavljuju posle drugog svetskog rata. Prvi elektronski autonomni robot sa kompleksnim ponašanjem je napravio V. G. Volter na Burdenovom Neurološkom Institutu u Bristolu 1948. i 1949. godine. Volterovi roboti su bili u obliku kornjače i imali su mogućnost da se kreću ka izvoru svetlosti. Na taj način su mogli da nađu put do izvora struje ukoliko im nivo baterije padne na kritičan nivo.[3]

Tokom vremena dolazi do razvoja robota za upotrebu u industriji i vojsci. U poslednjoj dekadi, zbog sve niže cene autonomnih robota, sve je vidljivija njihova upotreba u svakodnevnom životu, u domaćinstvima, industriji i vojsci.

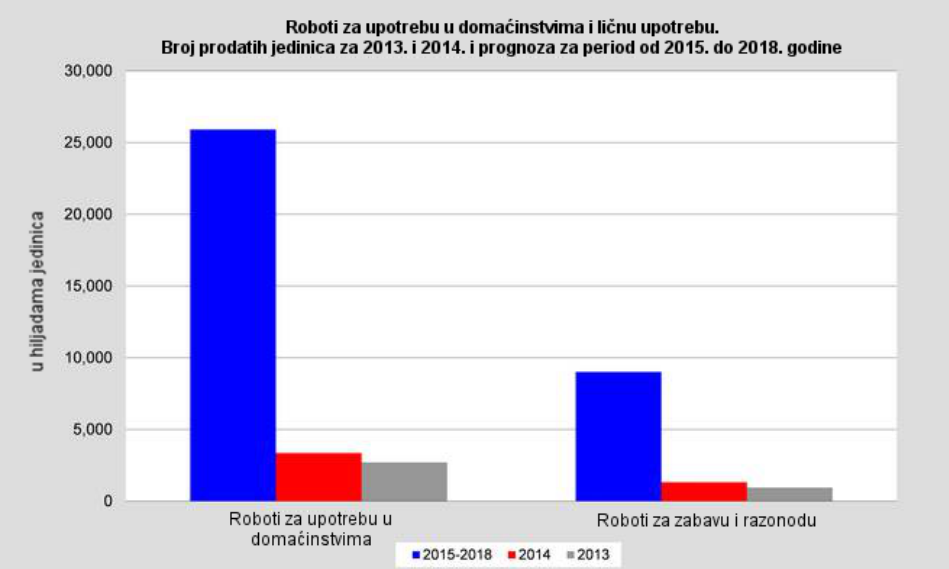
Prva šira upotreba robota u vojne svrhe se dogodila tokom Drugog svetskog rata. Nemačke i Sovjetske snage su imale svoja vozila kojima je bilo moguće upravljati daljinski i koristiti ih u borbenim dejstvima. Stvari nisu mnogo odmakle u

[1] Klíma, Ivan ,Karel Čapek:Life and Work. Catbird Press, 2002 ISBN 0945774532, (strana 260).

[2] http://www.reffell.org.uk/people/ericrobot.php

[3] http://www.cerebromente.org.br/n09/historia/greywalter_i.htm

narednim dekadama. Tek sredinom devedesetih godina dvadesetog veka, aktiviranjem sistema za globalno pozicioniranje (GPS) i razvojem telekomunikacija dolazi do naglog razvoja robota (dronova).[4] Vojske počinju da upotrebljavaju autonomne robote prilikom uklanjanja eksplozivnih naprava. Dronovi se sve više koriste u vojnim operacijama. Od 2005. do 2012. godine broj zemalja koje poseduju letelice sa daljinskim upravljanjem je skočio sa 40 na 75.[5] Predviđa se da će globalno tržište vojnih robota porasti sa 13.55 milijardi američih dolara u 2015. godini na 21.11 milijardu američkih dolara do 2020. godine (videti [3]).



SLIKA 1: BROJ PRODATIH ROBOTA ZA UPOTREBU U DOMAĆINSTVIMA (VIDETI [1])

U domaćinstvima se sve češće viđaju roboti. Najveći udeo tih robota čine roboti koji služe za usisavanje, čišćenje podova, održavanje travnatih površina, čišćenje prozora i slično. U poslednjih par godina beleži se značajan rast prodaje robota koji služe kao ispomoć hendikepiranim licima. U 2014. godini je prodato oko 4.7 miliona robota za ličnu upotrebu i upotrebu u domaćinstvima u vrednosti od 2.2 milijarde američkih dolara, što predstavlja rast od 28% u odnosu na 2013. godinu. Procenjuje se da će u periodu do 2018. godine biti prodato oko 25.9 miliona robota

[4] http://www.historynet.com/drones-dont-die-a-history-of-military-robotics.htm

[5] http://www.gao.gov/products/GAO-12-536

za upotrebu u domaćinstvima i da će tržište dostići vrednost od 12.2 milijarde dolara (slika 1, videti [1]).

Najzustupljeniji roboti u upotrebi su industrijski roboti. Industrijski roboti su roboti koji se koriste za proizvodnju drugih dobara. Od 2010. godine beleži se trend rasta tražnje za industrijskim robotima. Od 2010. do 2014. godine, godinu za godinom raste prodaja industrijskih robota za 17%. Najveći korisnici robota su automobilska, elektro i elektronska industrija, a zemlje sa najvećom zastupljenošću su Japan, Nemačka, Sjedinjene Američke Države i Republika Koreja. Predviđa se da će do kraja 2018. godine broj instaliranih robota rasti u proseku 15% iz godine u godinu (videti [2]).

Tržište robota je samo u poslednjih par godina neverovatno poraslo. Procenjuje se da će globalna potrošnja robotike i povezanih usluga skočiti sa trenutnih 71 milijarde američkih dolara na 135.4 milijarde američkih dolara do kraja 2019. godine. Robotizacija i automatizacija će drastično promeniti svet u narednim godinama i izazvaće velike promene širom sveta. Izvesno je da će doći do rasprostranjene upotrebe robota, ne samo u "tradicionalnim" granama kao što su automobilska, već i u zdravstvu, logistici, poljoprivredi, obrazovanju i drugim oblastima (videti [4]).

Rast upotrebe robota će izazvati i određene društvene i socijalne promene. Postoje različita viđenja kako će se proces robotizacije odvijati. Neki istraživači ocenjuju da će doći do smanjenja radnih mesta i socijalnih nemira, dok drugi na osnovu istorijskih podataka i prethodnih tehnoloških revolucija predviđaju da će doći do promene u strukturi poslova i smanjenja manje plaćenih poslova, ali i pojava novih radnih mesta u sektorima koji će baviti održavanjem i unapređivanjem robota.[6] [7]

[6] http://www.roboticstomorrow.com/article/2013/10/impacts-of-robotics-on-employment-safety-quality-productivity-and-efficiency/204/

[7] http://www.pewinternet.org/2014/08/06/future-of-jobs/

Osnova za ovaj rad je upravo robot. Postoji veliki broj robota koji se mogu koristiti kao primer u ovom radu. Mnogi roboti nastaju kao delo entuzijasta u oblasti robotike. Najveći broj tih robota koristi već gotove hardverske platforme - mikrokontrolere, integrisane računare za programiranje ponašanje robota. Ostatak robota zavisi od autora. Motori za pokretanje robota, senzori za daljinu i kamere i drugi delovi se priključuju po želji autora ili prema nameni robota. U ovom radu koristi se programibilni robot Arduino.

1.2 Tema, struktura i metodologija rada

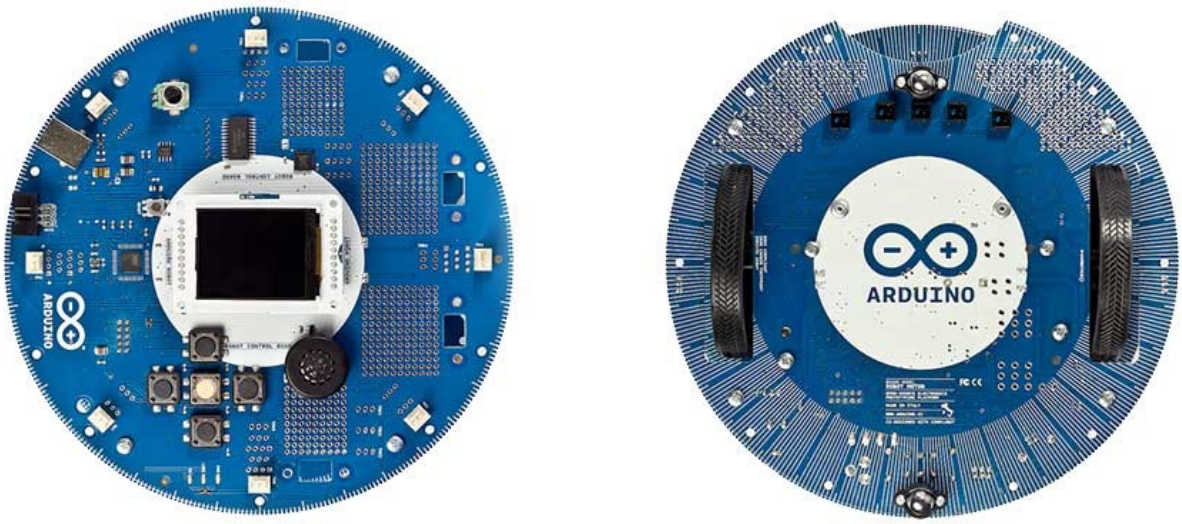
Sa ciljem da se predstave mogućnosti platforme Arduino robot kroz rad su u realizovani algoritmi za praćenje linije kod robota sa IC (infracrvenim) senzorima i to je ujedno i suština ovog rada. Ovaj rad je osmišljen kao uvod u razvoj na platformi Arduino robot. Prvo poglavlje se odnosi na osnove programiranja na platformi Arduino. U ovom poglavlju izlažu se osnove i način funkcionisanja platforme Arduino. Pored toga opisuju se sve funkcionalnosti platforme Arduino robot i njene mogućnosti.

Drugo poglavlje je usmereno na algoritme za praćenje linije kod robota sa IC senzorima. Pomoću platforme Arduino robot biće realizovani osnovni algoritmi za praćenje linije i izvršiće se njihova uporedna analiza. Navedeni pristup bi trebalo da pruži uvid u mogućnosti koje platforma Arduino poseduje i osnovu za izgradnju budućih projekata na istoj, kojima bi se iskoristili njeni puni potencijali. Nakon obrade nabrojanih zadataka sledi zaključivanje.

Korišćeni metodi za prikupljanje podataka su eksperiment i posmatranje tokom odabranih testiranja, a ostali podaci vezani za temu su prikupljeni sa zvaničnog sajta Arduino robota i iz radova koji su se bavili algoritmima za praćenje linija robota sa IC senzorima od kojih se izdvajaju sledeći autori: Sluka J., Anirudh S. N., Aravind K., Tarun M, pred svih ostalih navedenih u referencama. Prilikom obrade prikupljenih podataka i izrade rada biće opisani načini na koji su prikupljeni podaci i vršiće se poređenje dobijenih rezultata.

1.3 Šta je Arduino robot?

Arduino je kompanija koja se bavi proizvodnjom hardvera i softvera. Najčuvenija je po hardverskoj platformi po kojoj je i dobila ime. Platforma Arduino je nastala 2005. godine kao projekat za studente na Institutu za dizajn interakcija u gradu Ivrea u Italiji. Tokom vremena Arduino je prerastao u kompaniju koja proizvodi različite vrste čipova. Među njenim proizvodima je i Arduino robot (videti [5]).

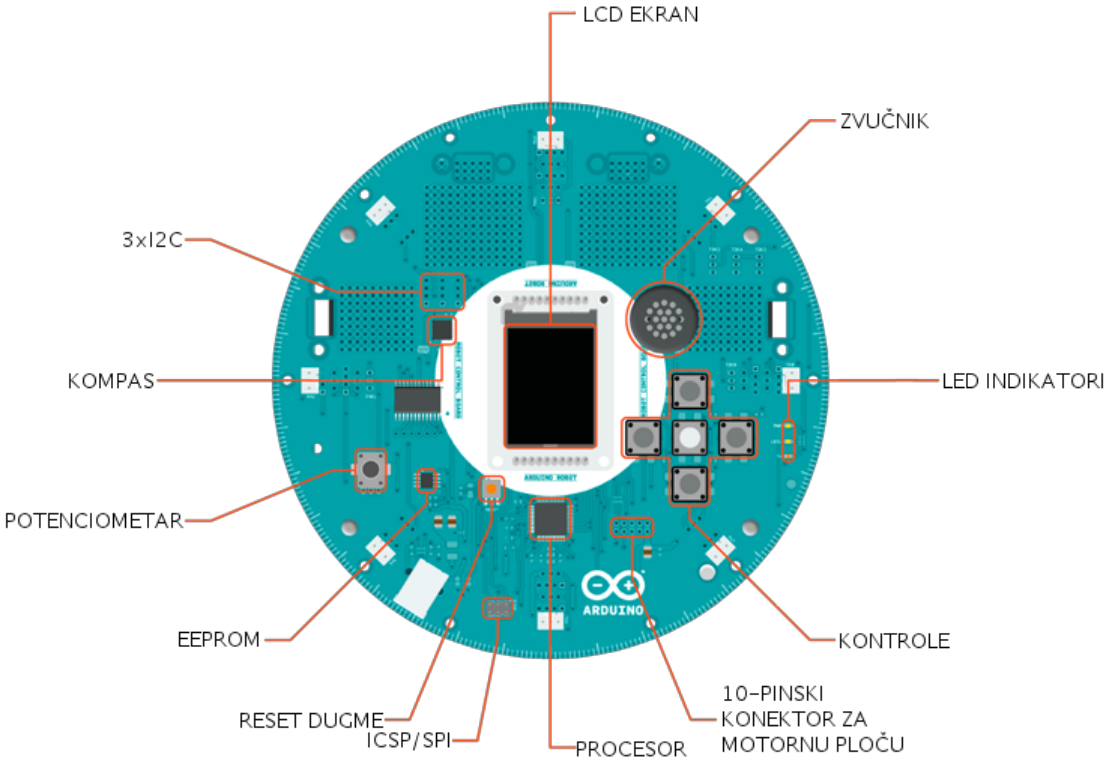


SLIKA 2: KONTROLNA I MOTORNA PLOČA

Arduino robot je jedini zvanični robot kompanije Arduino. Robot se sastoji od dve integrisane ploče (slika 2). Donja ploča se naziva motornom pločom i jedini zadatak joj je da kontroliše rad motora koji pokreću robota. Gornja ploča služi za čitanje podataka sa senzora i kontrolisanje robota. Obe ploče su mikrokontrolerske ploče bazirane na mikrokontroleru ATmega32u4.[8]

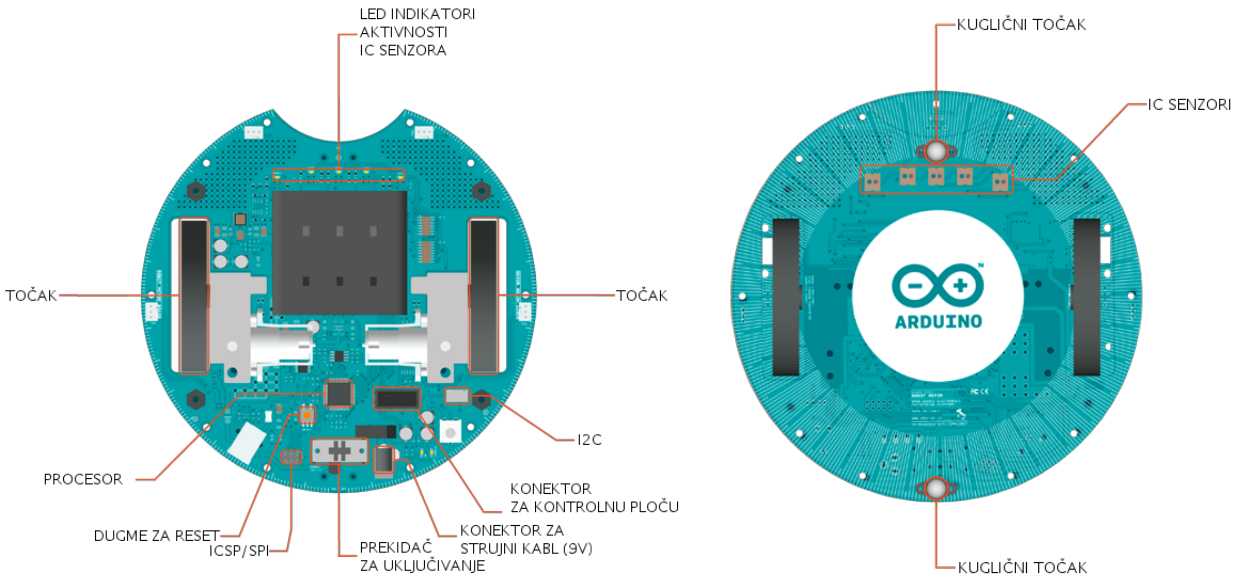
Kontrolna ploča pored standardnih procesora i EEPROMa sadrži dugmad za kontrolu, kompas, zvučnik, potenciometar i druge ulaze i priključke za mnoge senzore koje podržava platforma Arduino (slika 3).

[8] https://www.arduino.cc/en/Main/Robot



SLIKA 3: STRUKTURA KONTROLNE PLOČE

Motorna ploča pored standardnih procesora i EEPROMa sadrži motore za točkove, pet infracrvenih senzora za prepoznavanje linije na podu (slika 4).



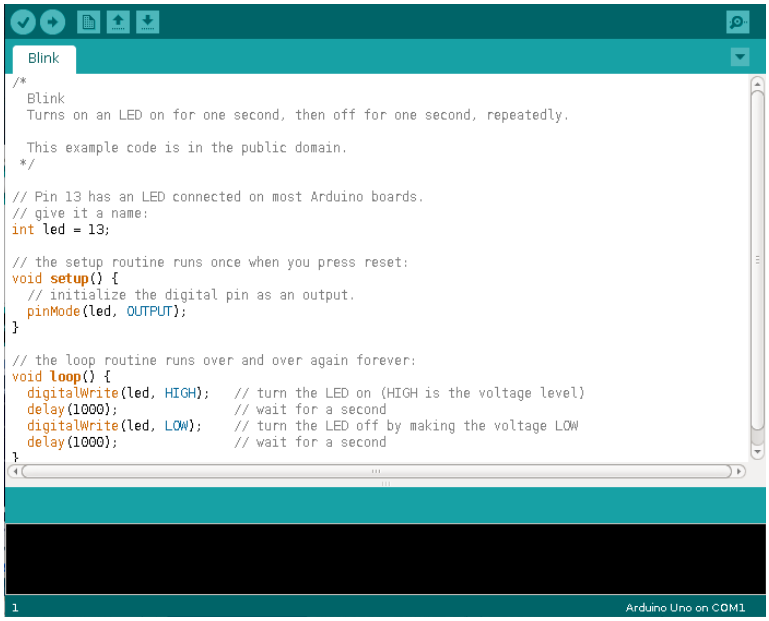
SLIKA 4: STRUKTURA MOTORNE PLOČE

2. Osnove programiranja na platformi Arduino

2.1 Programski jezik i alati

Za rad na platformi Arduino potrebno je koristiti jezik koji je prilagođen toj platformi. Zapravo ne radi se o programskom jeziku u punom smislu tih reči, već o podskupu funkcija programskog jezika C/C++.

Napisani kod se naziva skicom (*eng. sketch*). Skica prolazi kroz Arduino parser koji vrši neke male promene nad kodom (na primer automatski generiše prototipove funkcija) i potom se taj kod prosleđuje C/C++ kompajleru (avr-g++). Sve konstrukcije koje su podržane od strane avr-g++ kompajlera trebalo bi da rade bez problema na platformi Arduino[9].



SLIKA 5: RAZVOJNO OKRUŽENJE ARDUINO

Celokupan proces pisanja koda, kompajliranja i instalacije za Arduino uređaje vrši se u razvojnom okruženju Arduino (slika 5).

[9] https://www.arduino.cc/en/Main/FAQ

2.2 Osnovna struktura Arduino programa

Osnovna struktura Arduino programa je prilično jednostavna. Svaki Arduino program se sastoji iz dve funkcije (kod 1).

```
void setup()
{
      // Komande
}

void loop()
{
      // Komande
}
```

KOD 1: OSNOVNA STRUKTURA PROGRAMA ZA PLATFORMU ARDUINO

Prva funkcija **setup()** služi kao pogodno mesto za deklarisanje svih promenljivih, inicijalizaciju svih potrebnih sistema na platformi Arduino ili podešavanje svih parametra pre izvršavanja glavnog dela programa. Ova funkcija se izvršava samo jednom prilikom pokretanja programa (videti [7]).

Na Arduino robotu u funkciji **setup()** se naječešće inicijalizuju sve biblioteke koje su potrebne u daljem izvršavanju programa.

Druga funkcija **loop()** je mesto na kome se nalaze komande koje se neprekidno izvršavaju kao, na primer, čitanje podataka sa ulaza, reagovanje na određene vrednosti, pisanja podataka na izlaz. Kao što joj ime kaže, ova funkcija se izvršava neprekidno sve dok traje izvršavanje programa (videti [7]).

Na Arduino robotu funkcija **loop()** se koristi za čitanje ulaznih vrednosti sa senzora, reagovanje na njih podešavanjem brzine motora robota, ispisivanjem vrednosti na ekran i slično.

2.3 Osnovni tipovi podataka

Arduino jezik podržava nekoliko osnovnih tipova podataka:

- **bool** - jednobitni tip podataka. Može imati vrednost **true** (tačno) ili **false** (netačno).
- **char** - osmobitni tip podataka koj ima vrednost u opsegu -128 do 127.
- **byte** - tip podataka koji ima osmobitnu vrednost u opsegu 0 - 255.
- **int** - celobrojni šesnaestobitni tip podataka koji može da ima vrednost u opsegu od -32768 - 32767.
- **long** - celobrojni tridesetdvobitni tip podataka koji može da ima vrednost u opsegu od -2147483648 - 2147483647.
- **float** - tip podataka sa pokretnim zarezom u tridesetdvobitnom formatu. Ovaj tip podataka može sadržati vrednost u opsegu od -3.4028235E+38 - 3.4028235E +38.

Sva izračunavanja koja uključuju brojeve sa pokretnim zarezom su sporija u odnosu na izračunavanja koja uključuje ostale vrste tipove podataka, jer su na platformi Arduino realizovane softverski. Takođe, operacije sa brojevima sa pokretnim zarezom nisu toliko precizne (do 6, 7 decimale preciznosti) (videti [8]).

Tipovi int, long i char mogu imati kvalifikator **unsigned** koji označava da promenljiva tog tipa mogu imati samo pozitivne vrednosti. U tom slučaju važe sledeći opsezi:

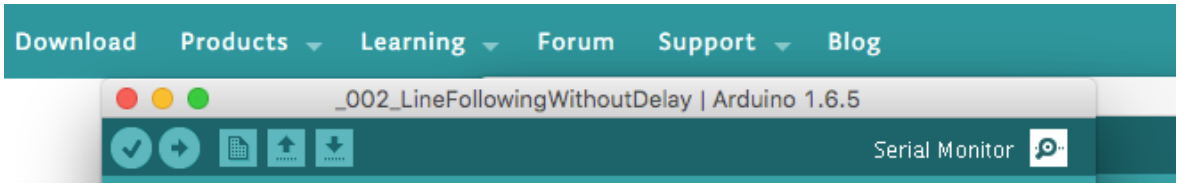
- **unsigned char** - opseg od 0 - 255. Isto kao tip podataka **byte**.
- **unsigned int** - opseg od 0 - 65535
- **unsigned long** - opseg od 0 - 4294967295.

2.4 Standardne funkcije u Arduino biblioteci

Pored standardnih funkcija iz programskog jezika C koje se mogu koristiti, postoje takođe neke jako korisne funkcije u osnovnim Arduino bibliotekama koje se često koriste, a koje bi trebalo spomenuti. Takođe u ovom delu pored standardnih funkcija iz biblioteke Arduino biće navedene i neke funkcije koje su izdvojene u narednim poglavljima, a specifične su za Arduino robota.

Funkcija **delay(time)** je funkcija koja ima ulogu da pauzira izvršavanje programa za broj milisekundi navedenih u argumentu funkcije. Argument funkcije je pozitivan 32-bitan broj (**long**). Ukoliko koristimo funkciju **delay** u funkciji **loop**, dok se robot kreće, **delay** neće zaustaviti kretanje robota, samo će zaustaviti izvršavanje dalje navedenih funkcija (videti [6]).

Ukoliko je potrebno nešto podrobnije istražiti, debagovati, najprimitivniji način za to je ispisivanje poruka na standardni izlaz. Jedini način da te poruke postanu vidljive je da se proslede i prikažu na računaru. Ukoliko postoji potreba da se poruka prosledi računaru potrebno je da se otvori veza sa računarom i to se čini pomoću funkcije **Serial.begin(speed)**. Ova funkcija se poziva u **setup** funkciji programa. Kao argument navodi se brzina prenosa podataka u bitovima po sekundi. Brzina od 9600 bitova po sekundi se najčešće koristi u praksi. **Serial.println** i **Serial.print** su funkcije koje šalju podatke na serijski port u ASCII obliku. Sve što se pošalje ka računaru moguće je videti u odeljku "**Serial Monitor**" koji se nalazi u Arduino razvojnom okruženju (slika 6).



SLIKA 6: SERIAL MONITOR OPCIJA U ARDUINO RAZVOJNOM OKRUŽENJU

U okviru Arduino biblioteke za robote osnovna funkcije za pomeranje robota je funkcija **Robot.motorsWrite(speedLeft, speedRight)**. Prvi argument označava željenu brzinu levog motora, dok druga označava željenu brzinu desnog motora. Ove vrednosti mogu biti u opsegu od -255 do 255. Ukoliko se prosledi negativna vrednost, motor će se pokretati unazad, a ukoliko se prosledi pozitivna vrednost, motori će se pokretati napred. Robot se zaustavlja pozivanjem funkcije **Robot.motorsStop()** koja zaustavlja oba motora.

Robot.turn(degrees) i **Robot.pointTo(degrees)** su takođe korisne funkcije za pomeranje robota iz biblioteke Arduino za robote. **Robot.turn(degrees)** je funkcija koja rotira robota za određeni broj stepeni u odnosu na trenutnu poziciju. Ukoliko se prosledi vrednost koja je negativna robot će se zarotirati u smeru suprotnom od kazaljke na satu, a ukoliko se prosledi pozitivna zarotiraće se u smeru kazaljke na satu. **Robot.pointTo(degrees)** je funkcija čije je uloga da zarotira robota ka određenom pravcu korišćenjem ugrađenog kompasa. Kao argument se unosi pravac u obliku stepena kao vrednost u opsegu od 0 do 359. Razlika između funkcija je u tome što **Robot.pointTo(degrees)** rotira ka apsolutnom pravcu, dok **Robot.turn(degrees)** rotira relativno u odnosu na trenutni pravac.

Vrlo značajne za upravljanje kretanjem robota su i funkcije koje očitavaju vrednosti sa senzora za detektovanje linije. Vrednosti senzora se nalaze u nizu **Robot.IRarray[]**. Niz sadrži pet elemenata. Svaki od tih elemenata odgovara tačno jednom infracrvenom senzoru i predstavlja celobrojnu vrednost u opsegu od 0 do 1023. Vrednost 0 označava da ispod tog senzora se nalazi površina koja ne odbija svetlost, dok vrednost 1023 označava da ispod datog senzora površina reflektuje svu svetlost. Ukoliko se nacrta crna linija na beloj površini, vrednosti senzora iznad crne linije će biti približne nuli. Ove vrednosti se ne ažuriraju automatski. Ažuriranje vrednosti niza se postiže pozivanjem funkcije **Robot.updateIR()** na kontrolnoj ploči Arduino robota, odnosno **IRread()** na motornoj ploči. **IRread**

prima jedan argument koji označava sa kog senzora se vrši očitavanje vrednosti. Funkcija vraća celobrojnu vrednost od 0 do 1023.

Da bi se koristile gore navedene i druge funkcije Arduino biblioteke za robote potrebno je da se uključi zaglavlje "ArduinoRobot.h" i pozove funkcija **Robot.begin()** u **setup** funkciji programa. Ova funkcija inicijalizuje sve specifične delove Arduino robota.

2.5 Programiranje motorne ploče Arduino Robota

Kao što je ranije pomenuto Arduino robot se sastoji iz dve ploče, kontrolne ploče i motorne ploče. Ove dve ploče su povezane pomoću kabla i komuniciraju međusobno. Motorna ploča je zamišljena kao konačni automat, a kontrolna ploča kao "mozak" čitavog robota koji ima zadatak da menja stanja motorne ploče i zadaje pojedinačne komande ili zahteva dostavljanje podataka od motorne ploče.

Stanje motorne ploče kod Arduino robota se naziva režimom rada motorne ploče. Motorna ploča Arduino robota ima nekoliko predefinisanih režima rada, ali je takođe moguće definisati novi. Režim rada motorne ploče se može promeniti pozivanjem funkcije **setMode(mode)** na kontrolnoj ploči sa identifikatorom režima rada kao argumentom ove funkcije. Identifikator je celobrojni tip podataka.

Da bi se definisao novi režim rada potrebno je da se uradi sledeće:

1. Definiše se režim rada i označi se jedinstvenom celobrojnom vrednošću (identifikatorom) u fajlovima *ArduinoRobotMotorBoard.h* i *ArduinoRobot.h (primer #define IME_REZIMA broj)*. Oba ova fajla se mogu naći u fajlovima koji su instalirani prilikom instalacije Arduino integrisanog okruženja za razvoj. Pored tih fajlova u istom folderu postoje i drugi fajlovi koje čine osnovu operativnog softvera Arduino robota.

2. U funkciji **process()** koju se može naći u fajlu *ArduinoRobotMotorBoard.cpp* dodaje se kod koji će se izvršavati ukoliko je izabran režim rada koji je upravo definisan.
3. Poslednja stvar koja je potrebna jeste da se izabere novi režim rada na motornoj ploči. To se postiže tako što se u programu na kontrolnoj ploči poziva funkcija **setMode()** koja kao argument prima identifikator novog režima rada.

Pored režima rada, mogu se definisati i nove komande na motornoj ploči. Komande se mogu podeliti na dve vrste:

- Naredbene komande (ili jednosmerne), koje predstavljaju komande koje kontrolna ploča prosleđuje motornoj koja ih potom izvršava. Motorna ploča ne šalje nikakve podatke kontrolnoj ploči. Primer za ovu vrstu komandi je **motorsWrite()**. Ako se pogleda kod ove funkcije može se videti da kontrolna ploča prvo prosleđuje tip komande, a zatim i argumente komande - brzinu levog i desnog motora. Na osnovu tipa komande motorna ploča zna da očekuje brzinu levog i desnog motora i prosleđuje te vrednosti na prave "adrese".
- Upitne komande (ili dvosmerne) su komande koje kontrolna ploče prosleđuje motornoj ploči sa namerom da im motorna ploča na njih odgovori. Primer takve komande je već navedena **updateIR()** komanda. Kontrolna ploča inicira komunikaciju tako što pošalje komandu za ažuriranje vrednosti senzora. Motorna ploča čita komandu i vrednosti senzora, a potom prosleđuje tražene podatke kontrolnoj ploči.

Ukoliko postoji potreba da se definiše nova komanda potrebno je izvršiti sledeće korake:

1. U fajlovima "libraries/Robot_Control/ArduinoRobot.h" i "libraries/Robot_motor/ArduinoRobotMotorBoard.h" definišu se jedinstveni celobrojni identifikatori komandi. (#define IME_KOMANDE broj).

2. Ukoliko je komanda upitna potrebno je da se definiše identifikator povratne komande. Praksa je da naziv povratne komande bude istovetan identifikatoru upitne komande sa sufiksom "_RE".
3. Kod za slanje komande mora prvo poslati identifikator komande, a potom jedan za drugim parametre potrebne za izvršavanje te komande. Parametri mogu biti tipa **byte** ili **int**. Na Arduino kontrolnoj ploči robota slanje parametra i identifikatora komande se obavlja pozivom funkcije **writeByte** i **writeInt**. Pošto se navedu sve informacije potrebne za izvršavanje komande, slanje komande se obavlja pozivom funkcije **sendData**(primer - kod 2).

```
void RobotControl::setMode(uint8_t mode){
     messageOut.writeByte(COMMAND_SWITCH_MODE);
     messageOut.writeByte(mode);
     messageOut.sendData();
}
```

KOD 2: FUNKCIJA ZA PROMENU REŽIMA RADA ARDUINO ROBOTA

4. Parsiranje komande na motornoj ploči se vrši u funkciji **parseCommand** u fajlu ArduinoRobotMotorBoard.cpp (kod 3).

```
uint8_t command=messageIn.readByte();
switch(command){
     …
     case COMMAND_SWITCH_MODE:
          modeName=messageIn.readByte();
          setMode(modeName);
          break;
     …
}
```

KOD 3: FUNKCIJA ZA PARSIRANJE KOMANDE

3. Algoritmi za praćenje linije

3.1 Osnovni algoritam za praćenje linije

Da bi robot pratio ucrtanu liniju na što efikasniji način, potrebno je da se izabere dobar algoritam. U ovom poglavlju polazi se od osnovnog algoritma, a zatim se razmatraju njegove usavršene verzije. Algoritmi se zasnivaju na činjenici da robot mora posedovati senzore za detektovanje linije i da je jedan od tih senzora na početku izvršavanja algoritma iznad linije koju treba da prati. Takođe, linija mora biti zatvorena. U slučaju da nije i da robot izađe van njenog okvira ne garantuje se dalje ponašanje robota.

Arduino Robot koji se koristi u ovom primeru ima pet senzora za praćenje linije. Senzori funkcionišu tako što emituju infracrveni snop svetlosti i potom detektuju količinu svetlosti koja se vratila natrag do senzora. Ukoliko je površina ispod senzora crna vratiće se mala količina svetlosti, a ukoliko je bela vratiće se većina emitovane infracrvene svetlosti. Kod Arduino Robota senzori vraćaju niz brojeva, po jedan broj za svaki senzor, u opsegu od 0 do 1023. Vrednost 0 označava da se senzor nalazi iznad tamne površine, 1023 iznad svetle površine.

Za najjednostavniji algoritam se mogu koristi samo tri centralna senzora.
Algoritam se sastoji od sledećih koraka (videti [12]):

1. Pročitaju se podaci sa senzora.
2. Ukoliko je senzor u sredini detektovao liniju robot se pomera napred.
3. Ukoliko je desni senzor pozicioniran iznad linije levi motor dobija veću brzinu i robot se rotira udesno.
4. Ukoliko je levi senzor pozicioniran iznad linije desni motor dobija veću brzinu i robot se rotira ulevo.
5. Ponavlja se prvi korak 1.

```
Algoritam Osnovni(motorSpeed)
Ulaz: motorSpeed (integer), sensorThreshold (integer)
while (true) do
     A := readSensorValues();
     if A[middleSensorId] < sensorThreshold
          setLeftMotorSpeed(motorSpeed);
          setRightMotorSpeed(motorSpeed);
     else if A[rightSensorId] < sensorThreshold
          setLeftMotorSpeed(motorSpeed);
          setRightMotorSpeed(0);
     else if A[leftSensorId] < sensorThreshold
          setLeftMotorSpeed(0);
          setRightMotorSpeed(motorSpeed);
```

KOD 4: PSEUDO KOD OSNOVNOG ALGORITMA ZA PRAĆENJE LINIJA

Ovaj algoritam se može implementirati na kontrolnoj ploči na sledeći način:

```
#include <ArduinoRobot.h>

int motorSpeed = 0;
int sensorThreshold = 0;

void setup() {
  Serial.begin(9600);
  Robot.begin();
  Robot.beginTFT();
  motorSpeed = map(readInputValue(), 0, 1023, 0, 255);
  sensorThreshold = readInputValue();
}

void loop() {
  Robot.updateIR();

  if (Robot.IRarray[2] < sensorThreshold) {
    Robot.motorsWrite(motorSpeed, motorSpeed);
  }
  else if (Robot.IRarray[1] < sensorThreshold) {
    Robot.motorsWrite(motorSpeed, 0);
  }
  else if (Robot.IRarray[3] < sensorThreshold) {
    Robot.motorsWrite(0, motorSpeed);
  }
}

int readInputValue()
{
  while (true) {
    if (Robot.keyboardRead() == BUTTON_MIDDLE) { delay(100);
break; }
```

```
    Robot.debugPrint(Robot.knobRead());
  }

  return Robot.knobRead();
}
```

KOD 5: OSNOVNI ALGORITAM ZA PRAĆENJE LINIJE NA KONTROLNOJ PLOČI ARDUINO ROBOTA

Ukoliko bi se pokrenuo ovaj algoritam na kontrolnoj ploči Arduino robota, može se primetiti da algoritam ne samo da nije savršen, nego i ne radi. Robot se ponaša kao da liniju uopšte i ne vidi i često se kreće pravolinijski bez ikakvog menjanja pravca.

Debagovanjem se može videti da je problem u vrednostima senzora za praćenje linije. Vrednosti u toku kretanja ne odgovaraju realnim vrednostima koje robot treba da vrati. Daljim istraživanjem dolazi se do uzroka problema. Pre nego što se pročita vrednost željenog senzora (iz niza **Robot.IRarray[]**) potrebno je pozvati funkciju **Robot.updateIR()** koja praktično treba da popuni gore pomenuti niz sa trenutnim vrednostima senzora. Problem je u tome što kada se pozove funkcija **Robot.updateIR(),** vrednosti senzora za praćenje linije se ne ažuriraju istog trenutka. Razlog zašto je tako može se videti u funkciji koja služi za čitanje vrednosti senzora na kontrolnoj ploči (kod 6, 7).

```
void RobotControl::updateIR()
{
  messageOut.writeByte(COMMAND_READ_IR);
  messageOut.sendData();
  delay(10);
  if(messageIn.receiveData())
  {
    if(messageIn.readByte()==COMMAND_READ_IR_RE)
    {
      for(int i=0;i<5;i++) { IRarray[i]=messageIn.readInt();}
    }
  }
}
```

KOD 6: FUNKCIJA ZA AŽURIRANJE PODATAKA SA SENZORA NA KONTROLNOJ PLOČI

```
void RobotMotorBoard::_readIR()
{
  int value;
  messageOut.writeByte(COMMAND_READ_IR_RE);
  for(int i=0;i<5;i++)
  {
    value=_IRread(i);
    messageOut.writeInt(value);
  }
  messageOut.sendData();
}
```

KOD 7: FUNKCIJA ZA SLANJE PODATAKA SA SENZORA NA MOTORNOJ PLOČI

Prilikom pozivanja funkcije **Robot.updateIR()**, kontrolna ploča šalje komandu za čitanje vrednosti senzora motornoj ploči i čeka 10 milisekundi. U toku tih 10 milisekundi, dok kontrolna ploča prelazi na čekanje, motorna ploča dobija komandu za čitanje vrednosti senzora i šalje podatke sa senzora kontrolnoj ploči. Nakon čekanja od 10 milisekundi kontrolna ploča počinje da čita vrednosti sa motorne ploče i upisuje ih u niz kome se kasnije pristupa.

Ovakav način rada prouzrokuje kašnjenje podataka sa motorne ploče u trajanju od najmanje 10 milisekundi što predstavlja značajan vremenski period ukoliko je potrebno da robot reaguje pravovremeno na promene pravca linije. Zbog gore navedenih razloga postoje dva način da se implementira dobar algoritam za praćenje linije. Prvi je da se implementira na kontrolnoj ploči ali da se motori robota zaustavljaju, svaki put dok se vrednosti senzora na kontrolnoj ploči ne ažuriraju (u daljem tekstu Algoritam 1, videti kod 8), ili drugi da se implementira na motornoj ploči gde neće biti zadrške u dostavljanju vrednosti senzora za praćenje linije (u daljem tekstu Algoritam 2).

```
Algoritam Algoritam1(motorSpeed)
Ulaz: motorSpeed (integer), sensorThreshold (integer),
delayTime(integer)
while (true) do
     A := readSensorValues();
     wait(delayTime);
     setLeftMotorSpeed(0);
     setRightMotorSpeed(0);
     if A[middleSensorId] < sensorThreshold
```

```
            setLeftMotorSpeed(motorSpeed);
            setRightMotorSpeed(motorSpeed);
        else if A[rightSensorId] < sensorThreshold
            setLeftMotorSpeed(motorSpeed);
            setRightMotorSpeed(0);
        else if A[leftSensorId] < sensorThreshold
            setLeftMotorSpeed(0);
            setRightMotorSpeed(motorSpeed);
```

KOD 8: PSEUDO KOD ALGORITMA 1

Algoritam 1 na motornoj ploči Arduino robota se može implementirati na sledeći način:

```
#include <ArduinoRobot.h>

int motorSpeed = 0;
int sensorThreshold = 0;
const int delayTime = 25;

void setup() {
  Serial.begin(9600);
  Robot.begin();
  Robot.beginTFT();

  motorSpeed = map(readInputValue(), 0, 1023, 0, 255);
  sensorThreshold = readInputValue();
}

void loop() {
  Robot.updateIR();

  delay(delayTime);
  Robot.motorsStop();

  if (Robot.IRarray[2] < sensorThreshold) {
    Robot.motorsWrite(motorSpeed, motorSpeed);
  }
  else if (Robot.IRarray[1] < sensorThreshold) {
    Robot.motorsWrite(motorSpeed, 0);
  }
  else if (Robot.IRarray[3] < sensorThreshold) {
    Robot.motorsWrite(0, motorSpeed);
  }
}

int readInputValue()
{
  while (true) {
```

```
    if (Robot.keyboardRead() == BUTTON_MIDDLE) { delay(100);
break; }
    Robot.debugPrint(Robot.knobRead());
  }

  return Robot.knobRead();
}
```

KOD 9: IMPLEMENTACIJA ALGORITMA 1 NA KONTROLNOJ PLOČI ARDUINO ROBOTA

Za vrednost konstante **delayTime** uzimamo bilo koju vrednost koja je veća od 10 milisekundi potrebnih da se ažuriraju vrednosti senzora na kontrolnoj ploči. U ovom primeru koristi se vrednost od 25 milisekundi.

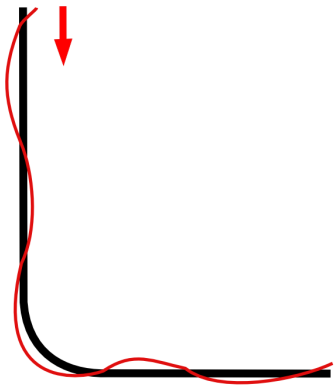
Problem kod Algoritma 1 je što se motori stopiraju pri svakoj evaluaciji podataka sa senzora. Ovakav algoritam ispunjava zadatak praćenja linije, ali nije najbrži ni najefikasniji. Da bi se postigla veća brzina, potrebno je da motori robota rade bez prekida, a to je moguće samo ako se osnovni algoritam implementira na motornoj ploči. Kod za ovakvo rešenje (Algoritam 2) izgleda ovako:

```
#include <ArduinoRobotMotorBoard.h>
const int motorSpeed = 110;
const int sensorThreshold = 250;
void setup() {
  Serial.begin(9600);
  RobotMotor.begin();
}

void loop() {
  if (RobotMotor.IRread(2) < sensorThreshold)
  {
    RobotMotor.motorsWrite(motorSpeed, motorSpeed);
  }
  else if (RobotMotor.IRread(1) < sensorThreshold)
  {
    RobotMotor.motorsWrite(motorSpeed, 0);
  }
  else if (RobotMotor.IRread(3) < sensorThreshold)
  {
    RobotMotor.motorsWrite(0, motorSpeed);
  }
}
```

KOD 10: OSNOVNI ALGORITAM ZA PRAĆENJE LINIJE NA MOTORNOJ PLOČI ARDUINO ROBOTA

Zbog specifičnosti implementacije algoritama na motornoj ploči Arduino robota, i kompleksnosti koda za čitanje vrednosti sa standardnih ulaza, implementacija ne podržava menjanje vrednosti brzine motora i granične vrednosti senzora.

Implementacija Algoritma 2 radi bez problema na motornoj ploči robota. Robot prati ucrtanu liniju, ali to radi uz primetnu dozu osciliranja (povremenog odstupanja robota od ucrtane linije, slika 7). Osciliranje je moguće regulisati smanjivanjem brzine robota ili drugačijom implementacijom algoritma koja će biti predstavljena u sledećim poglavljima.



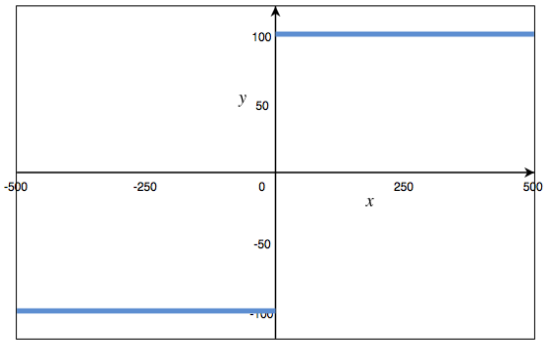
SLIKA 7: PRIMER OSCILIRANJA ROBOTA OD UCRTANE PUTANJE

3.2 Proporcionalni algoritam za praćenje linije

Zapaža se da Algoritam 2 sastoji od akcija koje se izvršavaju kada se očitavanja senzora nađu u nekom opsegu. U Algoritmu 2 ukoliko vrednost levog senzora pokazuje da se nalazi iznad linije vrši se korekcija robota tako što se povećava brzina desnog motora i ispravlja robot sve dok središnji senzor ne dođe direktno iznad linije koja se prati. Posmatranjem senzora sa desne strane, vrši se ispravljanje robota na sličan način. Poenta je u balansiranju između vrednosti senzora sa leve strane i desne strane. U slučaju da očitavanja sa jedne strane imaju nesrazmerno veće vrednosti vrši se balansiranje tako što se smanjuje brzina motora sa te strane. Neka je trenutna brzina levog motora M_L, *a* trenutna brzina desnog motora M_D. Vrednost svih senzora za praćenje linije sa leve strane može se

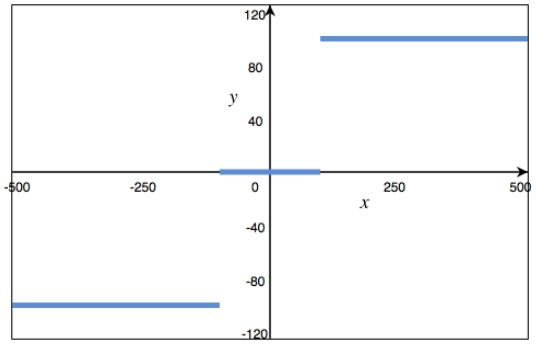
obeležiti sa SL_i, a sa desne strane SD_i. Trenutna brzina levog i desnog motora mogu se izraziti preko razlike očitanih vrednosti senzora leve i desne strane (slika 8).

Ukoliko bi se ovo predstavilo na grafiku, može se predstaviti kao zavisnost razlike brzine motora (odnosno zaokreta) u odnosu na razliku suma vrednosti očitanih sa senzora sa leve i desne strane (vrednost greške). Jedinica mere za grešku je ista kao jedinica mere za očitavanje senzora. Kod Arduino robota opseg zbira vrednosti levih senzora je od 0 do 2046, pa je mogući opseg greške od -2046 do 2046.



SLIKA 8: ZAVISNOST ZAOKRETA OD RAZLIKE VREDNOSTI SENZORA SA LEVE I DESNE STRANE (X - VREDNOST GREŠKE, Y - VREDNOST ZAOKRETA)

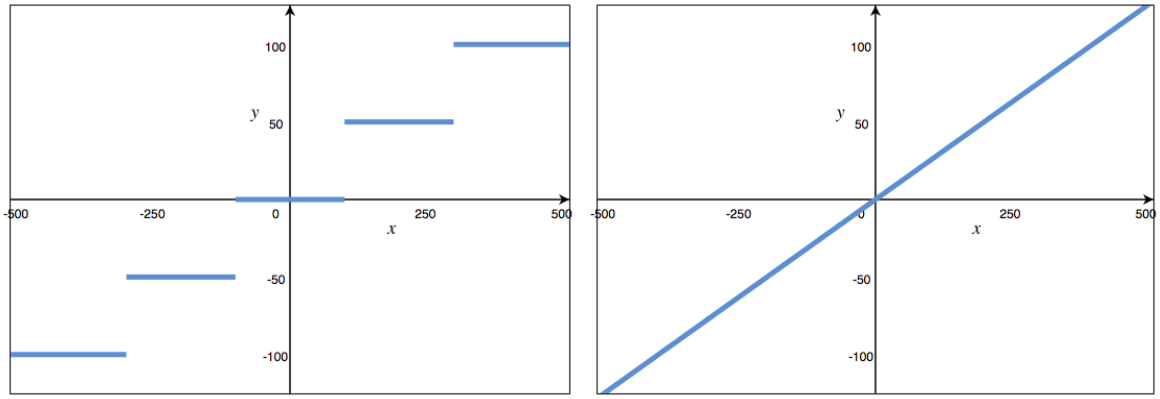
Kada se robotom upravlja pomoću prethodnih algoritama, pošto su sume svih vrednosti senzora sa leve strane i desne strane retko jednake, često se događa da robot izlazi van okvira linije. Taj problem može se rešiti uvođenjem praga tolerancije između razlike sume senzora sa leve i sume senzora sa desne strane (slika 9).



SLIKA 9: PRAG TOLERANCIJE RAZLIKE IZMEĐU SUME VREDNOSTI SENZORA SA LEVE I DESNE STRANE ROBOTA (X - VREDNOST GREŠKE, Y - VREDNOST ZAOKRETA)

Radi povećanja preciznost i smanjenja oscilovanja, tj. izlaženja robota van okvira linije, može se dodati još par opsega i na taj način smanjiti razlika u brzinama levog i desnog motora na manju vrednost kada je razlika između suma senzora jedne i druge strane manja. Što više opsega postoji to je preciznost veća i dolazi do manjeg osciliranja robota.

Zarad daljeg povećanja preciznosti može se nastaviti sa smanjivanjem opsega i na taj način će se dobiti grafik sličan grafiku linearne zavisnosti (slika 10). Ovo predstavlja osnovu proporcionalnog algoritma za praćenje linije (algoritma P). Kod ovog algoritma vrednost odstupanja brzine motora od željene brzine motore, u daljem tekstu vrednost zaokreta - *Turn*, je direktno proporcionalna razlici suma vrednosti senzora sa desne i leve strane robota, u daljem tekstu greške - *error* (nastale izlaženjem sa linije) (videti [10]).



SLIKA 10: VEĆI BROJ OPSEGA (X - VREDNOST GREŠKE, Y - VREDNOST ZAOKRETA)

Vrednost zaokreta se može predstaviti na sledeći način:

$$Turn = K_P \cdot error$$

Brzina motora se može izraziti kao (videti [9]):

$$M_L = T_P - Turn$$
$$M_R = T_P + Turn$$

T_P i K_P su ulazni parametri koji se mogu menjati u zavisnosti od implementacije i načina korišćenja algoritma. T_P predstavlja ciljanu brzinu kojom se kreće robot. U

zavisnosti od same linije zavise i koje se vrednosti mogu koristiti za ova dva parametra. Ukoliko je linija kretanja pretežno prava, bez krivina, mogu se koristiti velike vrednosti za T_P, dok vrednosti K_P mogu biti male. U slučaju da linija kretanja ima veliki broj krivina, uvek će postojati neka maksimalna vrednost koju T_P može da ima.

Problem kod algoritma P, kao i kod većine prethodnih, jeste da zahteva da robot reaguje trenutno na komande koje su mu prethodno date. Međutim, motorima na robotu je, kao i na drugim sistemima, potrebno određeno vreme da reaguju na komande. Ukoliko je brzina motora podešena na određenu vrednost, biće potrebno i do nekoliko desetinki da motor počne da radi na željenoj brzini. Ta mala kašnjenja smanjuju preciznost algoritma.

3.3 Proporcionalno integralni algoritam za praćenje linije

Da bi se popravio odziv robota na promene pravca u liniji kretanja, uvodi se novi deo, u daljem tekstu - *integral*. *Integral* predstavlja sumu grešaka nastalih dosadašnjim izvršavanjem programa. Svaki put kada se očitaju vrednosti senzora i izračuna greška nastala u datom trenutku, sumira se trenutna greška sa greškama nastalim u prethodnim izračunavanjima i na taj način se dobija integral.

$$integral = integral + error$$

Vrednost *integrala* se uvrštava u formulu za izračunavanja zaokreta. Kao što je i slučaju kod algoritma P, vrednost integrala se množi koeficijentom K_I. Na ovaj način dobijamo proporcionalno integralni algoritam (algoritam PI).

$$Turn = K_P \cdot error + K_I \cdot integral$$

Ovaj način izračunavanja zaokreta pomaže posebno u otklanjanju malih grešaka. Ukoliko je vrednost greške mala i parametar K_P nema veliku vrednost, vrednost zaokreta, odnosno vrednost kojom se regulišu brzine oba motora, neće bitnije

uticati na putanju robota i tako će dolaziti do izlaženja robota van ucrtane linije. Ukoliko bi se povećala vrednost K_P dolazilo bi do prevelikog osciliranja robota u odnosu na ucrtanu liniju. To je trenutak kada integralni deo ovog algoritma dolazi do izražaja. Sve prethodne male greške se akumuliraju i on dobija takvu vrednost da može da utiče na putanju robota.

U cilju poboljšanja preciznosti potrebno je uračunati još jedan faktor, vreme. Prilikom svakog računanja greške, greška se množi sa vremenom (*dT*) koje je proteklo između prethodnog i trenutnog izračunavanja vrednosti integrala (videti [10]).

$$integral = integral + error \cdot dT$$

Ukoliko se izračunava *error* i *integral* u jednakim vremenskim intervalima vreme se može skloniti kao faktor u izračunavanja vrednosti integrala.

Postoji još jedan veoma vidljiv problem sa izračunavanjem vrednosti *integrala*. Kako vreme prolazi i vrednosti prethodnih grešaka se sumiraju, vrednost *integrala* će rasti sve više i više. Takve vrednosti mogu tokom vremena samo da povećaju nepreciznost robota i izazvati njegovo izlaženje van okvira željene linije.
Postoje dva načina putem kojih se rešava opisani problem:

- Može se vrednost *integrala* postaviti na 0 ukoliko je trenutna greška prilikom očitavanja 0 ili ispod neke vrednosti.
- Ili se, prilikom svakog izračunavanja, vrednost *integrala* može pomnožiti sa nekom vrednošću u opsegu od 0 do 1.

$$integral = m \cdot integral + error \cdot dT,\ 0 < m < 1$$

3.4 Proporcionalno integralni i diferencijalni algoritam

Algoritam PI trenutno se sastoji iz dva izraza proporcionalnog (P) koji pokušava da ispravi trenutnu grešku i integralnog (I) koji pokušava da ispravi prethodne

greške. Naravno nameće se pitanje da li se može dodati nešto algoritmu tako da se utiče na buduće greške i da se iste smanje.

Taj deo se naziva diferencijalni. Metoda za izračunavanje diferencijalnog izraza zasniva se na pretpostavci da je razlika između trenutne i buduće greške jednaka razlici između prethodne i trenutne. Ukoliko je prethodna greška 1, a trenutna 3, vrednost sledeće greške je 3 + (3 - 1) = 5. Ta činjenica se može opisati izrazom (videti [10]):

$$error_{next} = error_{current} + difference$$

Vrednost greške pri sledećem izračunavanju je jednaka zbiru vrednosti trenutne greške i vrednosti diferencijala. Vrednost diferencijala se može predstaviti kao razlika između trenutne i prethodne greške.

$$difference = error_{current} - error_{previous}$$

Vrednost diferencijala se radi bolje preciznosti, kao i kod proporcionalnog i integralnog izraza, množi sa konstantom K_D. Ukoliko je vreme između očitavanja vrednosti senzora nejednako, onda se zbog bolje preciznosti čitava vrednost diferencijala može podeliti vremenom između očitavanje vrednosti senzora (dT). U slučaju da je jednako, dT se može izostaviti. Uvrštavanjem diferencijalnog izraza u formulu za zaokret, formula se može napisati na sledeći način (videti [10]):

$$Turn = K_P \cdot error + K_I \cdot integral + K_D \cdot difference$$

Ovo predstavlja proporcionalno, integralni i diferencijalni algoritam (algoritam PID, videti kod 11). Ukoliko je trenutna greška gora nego prethodna diferencijalni term pokušava da je ispravi. Ako je trenutna greška bolja (manja) nego prethodna, diferencijalni term pokušava da zaustavi algoritam od daljeg ispravljanja greške. Diferencijalni deo je veoma značajan zbog poslednjeg slučaja. Ukoliko se greška približava nuli onda je poželjno da se zaustavi ispravljanje robota da slučajno robot ne bi izašao van okvira linije (videti [9]).

```
Algoritam Pid(motorSpeed, Kp, Ki, Kd, m)
Ulaz: motorSpeed (integer), Kp (float), Ki (float),
Kd(float), m(float)
begin
     previousError := 0;
     totalError := 0;
     while true do
          error := leftSensorsSum() - rightSensorsSum()
          totalError := m * totalError + error;
          turn = Kp * error + Ki * totalError + Kd * (error -
previousError);
          setLeftMotorSpeed(motorSpeed - turn);
          setRightMotorSpeed(motorSpeed + turn);
          previousError = error;
end
```

KOD 11: PSEUDO KOD ALGORITMA PID

3.5 Postupak podešavanje parametara u algoritmu PID

Za dobro upravljanje robotom moraju se pažljivo odabrati vrednosti za sva tri parametra K_P, K_I i K_D. Postoji nekoliko tehnika za izračunavanje parametera, jedna od poznatijih je "Ziegler-Nichols metoda" (videti [10], [11]).

1. Početne vrednosti K_I i K_D su $K_I = K_D = 0$. Na taj način isključuje se integralni i diferencijalni deo algoritma.
2. Željene brzina motora se postavlja na što manju vrednost.
3. Za početnu vrednost K_P uzima se količnik maksimalne brzine motora i maksimalne greške.
4. Pokreće se robot i posmatra njegovo kretanje. Ukoliko robot ne može da prati liniju, povećava se vrednost K_P. Ukoliko mnogo oscilira smanjuje se vrednost parametra K_P. Na taj način se traži prava vrednost za K_P, sve dok robot ne počne da prati liniju uz blago osciliranje. Takva vrednost K_P se označava kao K_C.
5. Potrebno je da izračunati dT i P_C. dT je vrednost između dva merenja i to ne bi trebalo da bude problem za izračunavanje. P_C je vrednost koja označava koliko traje jedna perioda oscilacije robota, od trenutka dok robot ode na jednu stranu, vrati se na drugu i ponovo se pozicionira na liniju.

6. Korišćenjem naredne tabele određuju se približne vrednosti ostalih parametara.

Vrsta algoritma	K_P	K_I	K_D
P	$0.5K_C$	0	0
PI	$0.45K_C$	$1.2 * K_P * dT / P_C$	0
PD	$0.8K_C$	0	$P_C * K_P / (8 * dT)$
PID	$0.6K_C$	$2 * K_P * dT / P_C$	$P_C * K_P / (8 * dT)$

7. Ukoliko robot i dalje ne prati dovoljno dobro ucrtanu liniju, fino se podešavaju vrednosti parametra sve dok se ne dobije najbolje ponašanje robota.

3.6 Implementacija algoritma PID na motornoj ploči Arduino robota

Prilikom implementacije ovog algoritma veoma je važna brzina kojom se reaguje na promenu vrednosti greške. Da bi se postigla najbolja preciznost najbolje je implementirati ovaj algoritam kao novi režim rada na Arduino motornoj ploči. Klasa PIDFollow sadrži kod koji pokreće novi režim rada za praćenje linije. Deklariše se klasa u fajlu PIDFollow.h (kod 12).

```
#ifndef PID_FOLLOW_H
#define PID_FOLLOW_H

#if ARDUINO >= 100
     #include "Arduino.h"
#else
     #include "WProgram.h"
#endif

class PIDFollow {
public:
     PIDFollow();
     void followLine();
private:
     int normalizedSensorValues[5];
     float totalError = 0;
     float previousError = -256;
     int minimalSensorValue;
```

```
    int maximalSensorValue;
};
#endif
```

KOD 12: PIDFOLLOW.H

Funkcija **followLine** sadrži kod za kontrolu robota u ovom režimu rada. Da bi se ovaj kod izvršavao potrebno je da se definiše identifikator novog režima rada i da se funkciju **followLine** pozove kada je novi režim rada izabran na motornoj ploči. Novi režim rada je definisan kao **MODE_PID_FOLLOW** i njegov identifikator je dodat u fajlovima ArduinoRobotMotorBoard.h i ArduinoRobot.h (kod 13).

```
//motor board modes
#define MODE_SIMPLE 0
#define MODE_LINE_FOLLOW 1
#define MODE_ADJUST_MOTOR 2
#define MODE_IR_CONTROL 3
#define MODE_PID_FOLLOW 4
```

KOD 13: LISTA IDENTIFIKATORA REŽIMA RADA

Pošto je definisan identifikator novog režima rada, potrebno je da se u funkciji **process** u fajlu ArduinoRobotMotorBoard.cpp pozove funkcija **followLine** kada je novi režim rada izabran (kod 14).

```
void RobotMotorBoard::process() {
    …

    else if (mode == MODE_PID_FOLLOW) {
        pidFollow->followLine();
    }
}
```

KOD 14: FUNKCIJA ZA IZVRŠAVANJE TRENUTNOG REŽIMA RADA

Funkcija **followLine** je definisana u fajlu PIDFollow.cpp (kod 15).

```
#include "PIDFollow.h"
#include "ArduinoRobotMotorBoard.h"

PIDFollow::PIDFollow() {
  minimalSensorValue = 0;
  maximalSensorValue = 1024;
}

void PIDFollow::followLine() {
  for (int i = 0; i < 5; i++) {
```

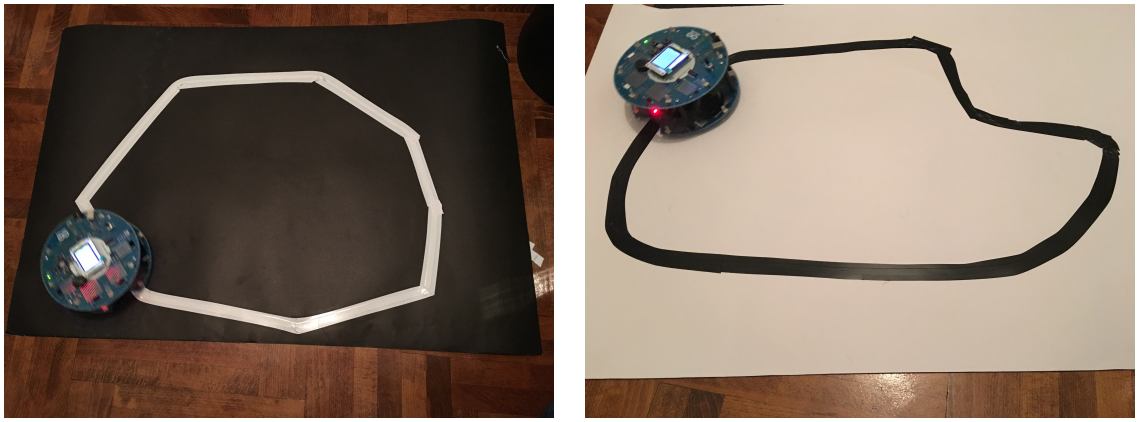
```
    normalizedSensorValues[i] = map(RobotMotor.IRread(i),
minimalSensorValue, maximalSensorValue, 0, 100);
  }

  float error = ((normalizedSensorValues[0] +
normalizedSensorValues[1]) -
    (normalizedSensorValues[3] + normalizedSensorValues[4]));

  // Smanjujemo veličinu ukupne greške množeći konstantom m,
koja je u ovom slučaju 2/3
  totalError = 2.0f / 3 * totalError + error;

  float kc = 1.275;
  float kp = 0.6 * kc;
  float ki = 2 * kp;
  float kd = 140 * kp;

  if (previousError < -255)
  {
    previousError = error;
  }

  int turn = kp * error + ki * totalError + kd * (error -
previousError);

  previousError = error;

  RobotMotor.motorsWrite(140 - turn, 140 + turn);
}
```

KOD 15: PIDFOLLOW.CPP

U prvom delu funkcije se čitaju vrednosti senzora i potom se mapiraju tako da budu u opsegu od 0 do 100. Potom se računa greška kao razliku između normalizovanih vrednosti senzora sa leve i desne strane od centralnog senzora. Posle svakog izračunavanja greške računa se ukupna greška, s tim što se umanjuje značaj prethodne akumulirane greške množenjem faktorom manjim od jedan, ali većim od nule. Posle svakog izračunavanja greške, pamte se vrednosti prethodne greške zbog potrebe računanja diferencijalnog dela formule. Vrednost zaokreta se računa po formuli iz algoritma PID i potom se koristi u funkciji za pokretanje motora robota. Vrednost parametra K_C se dobija kao količnik maksimalne vrednosti brzine motora (255) i maksimalne vrednosti greške (200). Na osnovu gore navedene tabele izračunavaju se parametri K_P, K_I, K_D. Proverom na primeru se

može utvrditi da li vrednosti parametra odgovaraju ucrtanoj liniji ili je potrebno malo modifikovati njihove vrednosti.

Algoritam se može proveriti u praksi tako što se napravi kružna linija na jednobojnoj podlozi crne ili bele boje. Važno je da boja linije bude u kontrastu u odnosu na boju pozadine. U ovom slučaju algoritam je testiran na hameru veličine A0 crne i bele boje sa linijom napravljenom od izolir trake širine približno 2,3 centimetra (slika 11). Bojenje hamera nekom bojom ne daje rezultate jer IC senzori zbog njihove konstrukcije ne prepoznaju razliku u boji između pozadine i linije. Kod Arduino robota pre bilo kakvog korišćenja potrebno je kalibrisati oba motora robota. To se postiže podešavanjem potenciometra na motornoj ploči robota i korišćenjem odgovarajućeg programa koji se nalazi u sastavu Arduino biblioteke. Prilikom pokretanja algoritma važno je da pozicioniramo centralni senzor iznad linije. Robot će nastaviti da se kreće sve dok se ne isključi. Željenu brzina se mora podesiti tako da odgovara liniji, u ovom slučaju (videti sliku 10) odabrana brzina je 140 od maksimalnih 255.



SLIKA 11: ARDUINO ROBOT NA UCRTANOJ LINIJI

4. Zaključak

Objašnjene su osnove platforme Arduino i Arduino robota, uvidom u osnovne programske jezike i alate, tipove podataka, strukturu Arduino programa i standardne funkcije. Objašnjen je izgled Arduino robota i celokupan proces pravljenja jedne aplikacije koja se pokreće na tom istom robotu. Prikazane su i osnovne metode za rad na ovim platformama, određene restrikcije i osnovne funkcionalnosti. Prikazan je način za pravljenje novog softvera i kako se može modifikovati postojeći softver motorne ploče Arduino robota. Sve navedeno pruža adekvatno upoznavanje sa platformom Arduino i uvod u mogućnosti koje ista pruža.

Predstavljeni su algoritmi za praćenje linije, kako u teoriji tako i u praksi korišćenjem Arduino robota. PID algoritam je prikazan kao najefikasniji algoritam koji omogućava da robot pri velikim brzinama glatko prati liniju bez bilo kakvih kratkotrajnih napuštanja iste. U tekstu je prikazan način implementacije algoritma, a ujedno i na koji se vrši brzo i fino podešavanja svih parametara PID algoritma. Istaknuta je važnost postojanja stabilne platforme za implementaciju algoritama za praćenje linije i brzog odziva komponenti na istim platformama.

Navedena realizacija algoritma za praćenje linije pruža okvir za uvid u dalji razvoj na platformi Arduino robot i njenu potencijalnu upotrebu. Postoje mnogi delovi platforme koji nisu pokriveni ovim radom, tj. ostali su neiskorišćeni, kao što su kompas, zvuk, ekran i slično, a koji se mogu koristiti za buduće projekte. Platforma takođe poseduje mogućnost za povezivanje sa drugim senzorima, koji se dalje mogu iskoristiti za razvoj različitih projekata i proizvoda.

5. Literatura

[1] International Federation of Robotics, World Robotics 2015 Service Robots, 2015, [Na internetu] Dostupno na: http://www.ifr.org/service-robots/statistics/

[2] International Federation of Robotics, World robotics 2015 Industrial Robots, 2015, [Na internetu] Dostupno na: http://www.ifr.org/industrial-robots/statistics/

[3] Markets and Markets Inc, Military Robots Market by Platform (Airborne, Naval, Land-based), Application (Warfield, Pick ‘n’ Place, Firefighting, Voice-controlled Robotic Vehicle, Metal Detector Robotic Vehicle, Others), & Region - Global Forecast to 2020, 2015, [Na internetu] Dostupno na: http://www.marketsandmarkets.com/Market-Reports/military-robots/market-245516013.html

[4] International Data Corporation, IDC's Worldwide Commercial Robotics Spending Guide Taxonomy, 2016s

[5] Hernando Barragán, The Untold History of Arduino, 2016, [Na internetu] Dostupno na: http://arduinohistory.github.io/

[6] Arduino, Arduino Language Reference, [Na internetu] Dostupno na: https://www.arduino.cc/en/Reference/HomePage

[7] Brian W. Evans, Arduino Programming Notebook, 2007, [Na internetu] Dostupno na http://playground.arduino.cc/uploads/Main/arduino_notebook_v1-1.pdf

[8] Brian Evans, Beginning Arduino Programming, Apress Media LLC, 2011.

[9] Anirudh Sunil Nath, Aravind Kumar, Tarun Malik, “Implementation Of PID Control To Reduce Wobbling In a Line Following Robot”, 2013, IJRET: International Journal of Research in Engineering and Technology

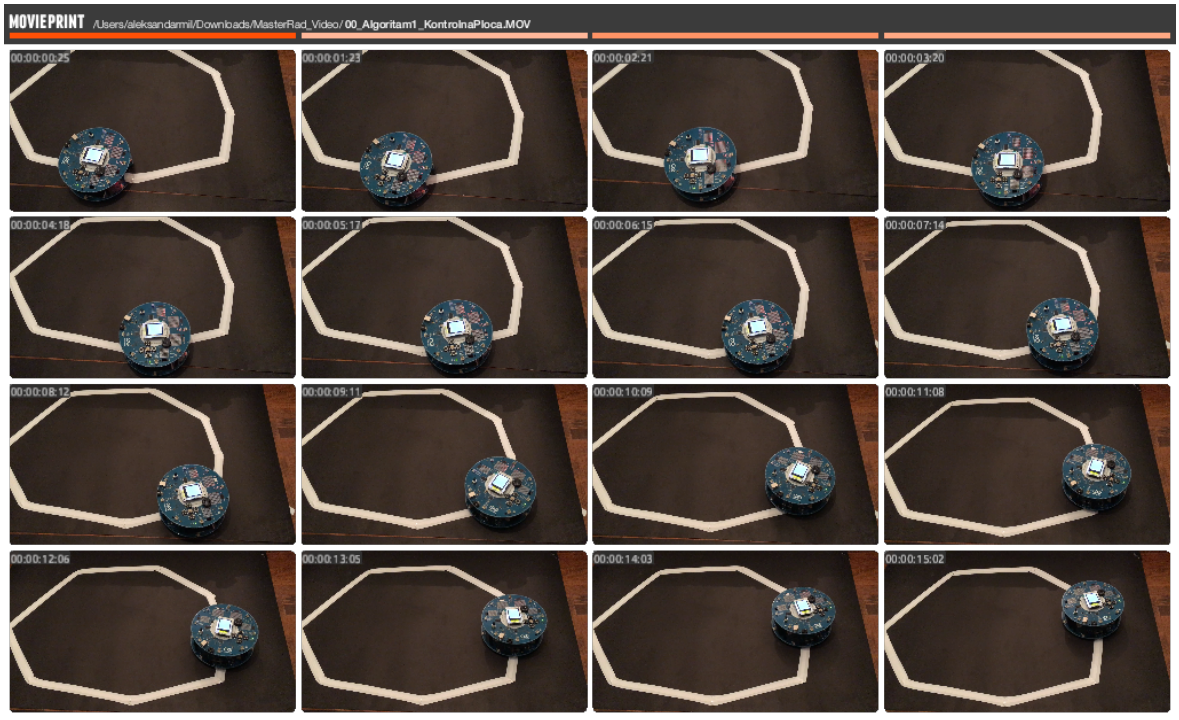
[10] Jim Sluka, A PID Controller For Lego Mindstorms Robots, 2009, [Na internetu] Dostupno na: http://inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html

[11] J.G. Ziegler and N. B. Nichols, Optimum settings for automatic controllers, Transactions of A.S.M.E., nov. 1942, 759-768

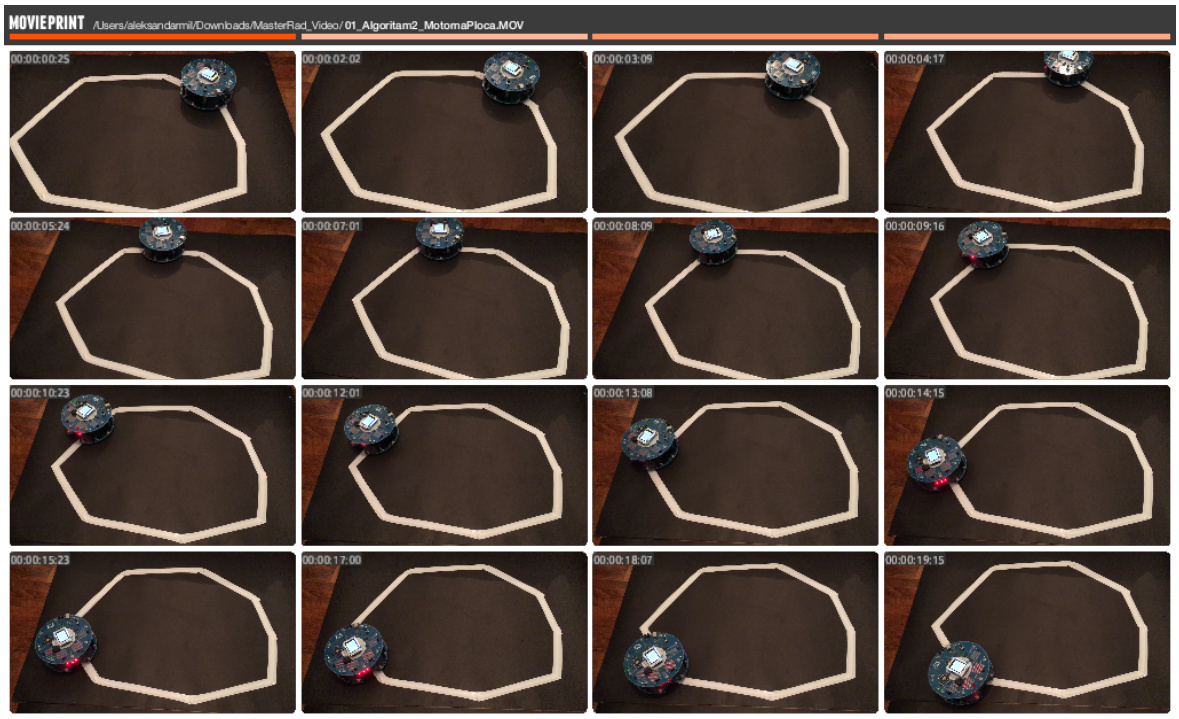
[12] John-David Warren, Josh Adams, Harald Molle, Arduino Robotics, Apress Media LLC, July 2011.

Prilog:

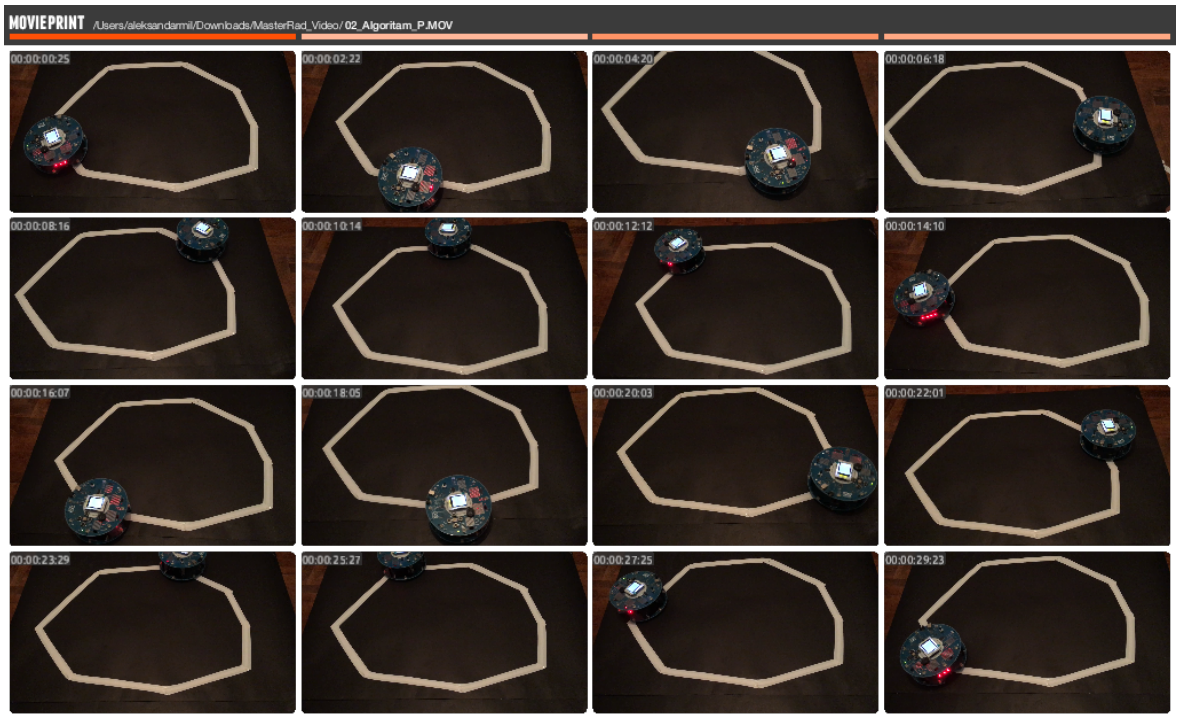
Video 1 - Osnovni algoritam sa zaustavljanjima implementiran na kontrolnoj ploči Arduino robota: https://goo.gl/AUZYWH



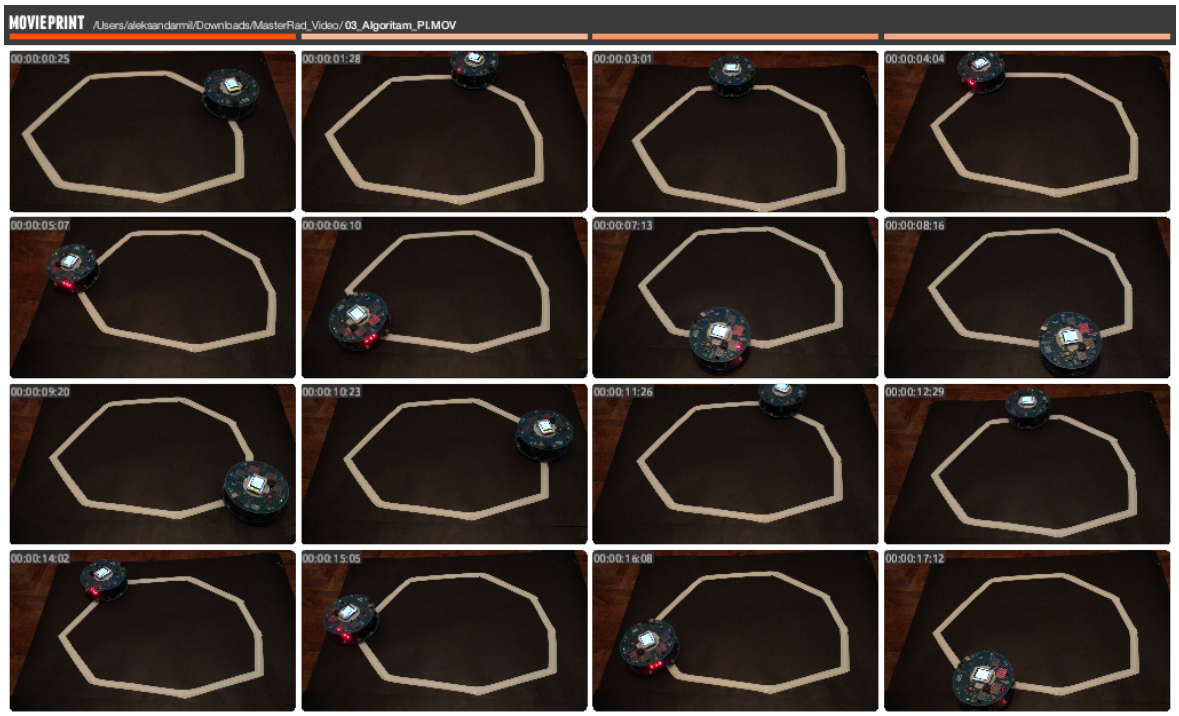
Video 2 - Osnovni algoritam implementiran na motornoj ploči Arduino robota: https://goo.gl/SHxqK9



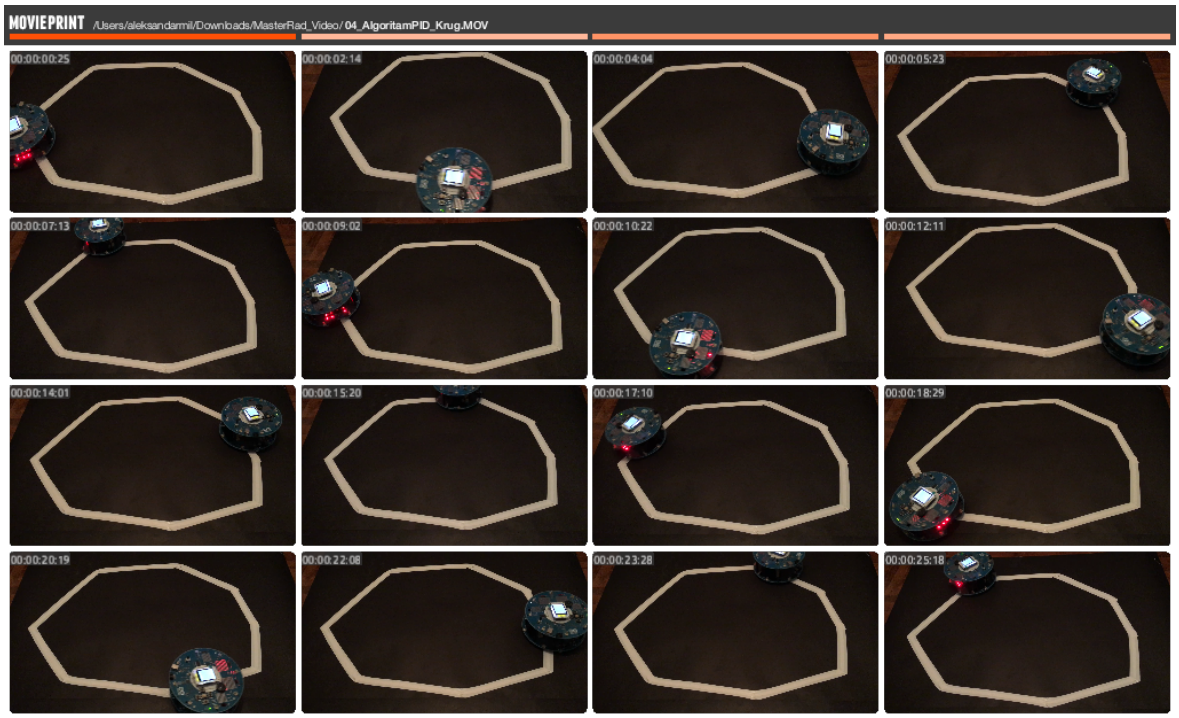
Video 3 - Algoritam P implementiran na motornoj ploči Arduino robota: https://goo.gl/CeyQT1



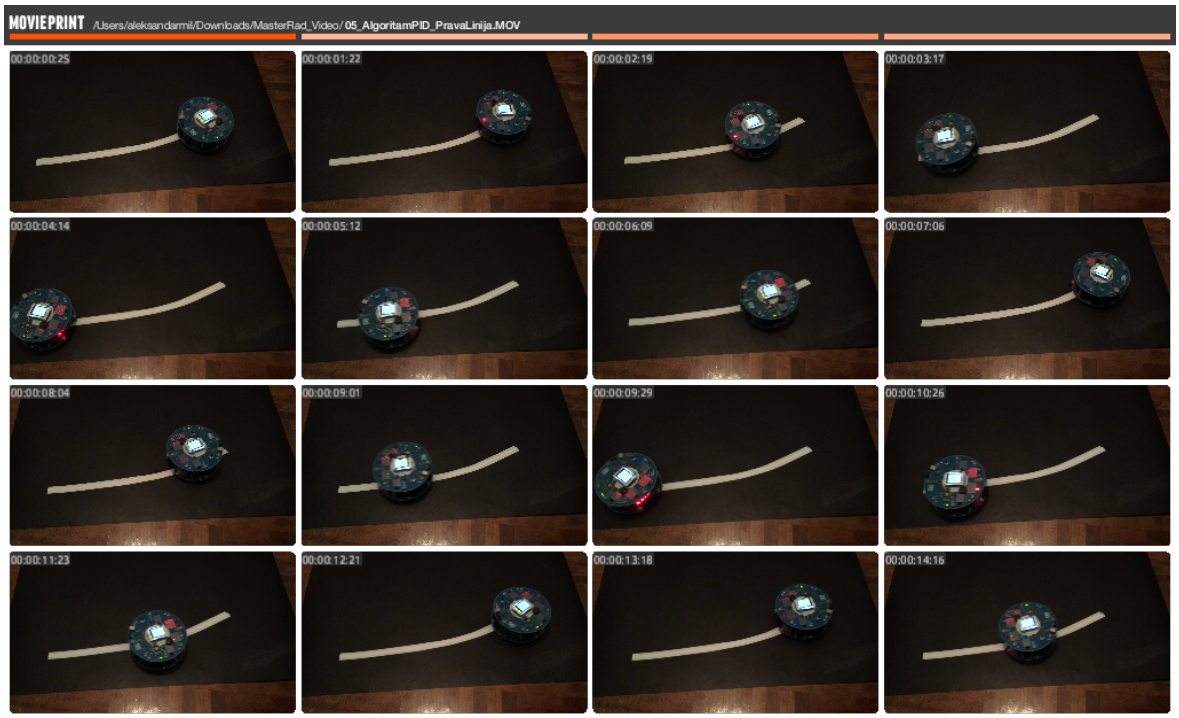
Video 4 - Algoritam PI implementiran na motornoj ploči Arduino robota: https://goo.gl/VUlz2K



Video 5 - Algoritam PID implementiran na motornoj ploči Arduino robota na kružnoj liniji: https://goo.gl/1Zw4Tr



Video 6 - Algoritam PID implementiran na motornoj ploči Arduino robota na pravoj liniji: https://goo.gl/7AqVPp



Kod - Primeri iz rada: https://goo.gl/hQRNG1