Saldus novads

Druvas vidusskola

Sensoru skaita ietekme uz līnijsekotāja robota kustību

Zinātniski pētnieciskais darbs informātikā

Darba autori:
Raivis Strogonovs,
Toms Friķis
11.e klase

Darba vadītājs:
Informātikas skolotājs **Normunds Svētiņš**

Saldus 2010

Saturs

Ievads 4
1. Līnijsekotājs robots 5
1.1. Līnijsekotāju roboti un to izmantošana robotu sacensībās 5
1.2. Līnijsekotāja robota darbības princips 5
1.3. Līnijsekotāja robota sensoru veidi 6
1.4. Datu apstrāde ar fototranzistoru un infrasarkano diodi 6
1.5. Līnijsekotāja robota algoritmi atkarībā no sensoru skaita 7
2. Robotu izveides priekšnosacījumi 8
3. Elektronikas izveide 9
3.1. Analogo signālu pārveidotājs (ADC) 9
3.1.1. ADC nepieciešamība robotam 9
3.1.2. Automātiskā konvertēšana 9
3.1.3. Manuālā konvertēšana 10
3.2. Impulsu platuma modulācija (PWM) 11
3.2.1. Pirmais mēģinājums pielietot PWM uz gaismas diodes (LED) 11
3.2.2. PWM ģenerēšana izmantojot pārtraukumus 12
3.2.3. PWM ģenerēšana izmantojot *fast PWM* un WGM metodi 13
3.3. 74hc595 pārbīdes reģistrs 15
3.3.1. Pārbīdes reģistra funkcija 15
3.3.2. Pirmais mēģinājums izmantot *porta paplašinātāju* 15
3.3.3. Programmas pirmkoda optimizēšana 16
3.4. Septiņu segmentu displejs 17
3.4.1. Īsumā par displeju 17
3.4.2. Pirmais eksperiments ar displeju 17
3.5. Motoru vadības bloks L293D 18

3.5.1. L293D nozīme robotam 18
3.5.2. Kā izmantot L293D 18
3.5.3. Problēmas ar L293D 19
4. Mehānikas izveide 20
4.1. Piedziņa 20
4.2. Līdzstrāvas elektriskie motori 20
4.3. Riteņi 21
4.4. Virsbūve 21
5. Robotu testēšana 22
Secinājumi 24
Uzziņas avotu saraksts 25
Anotācija 26
Annotation 27
Pielikumi 28

Ievads

Lai atvieglotu cilvēku dzīvi, jau kopš pagājušā gadsimta 60-tajiem gadiem zinātnieki sāka izgatavot robotus. Robotika kā datorinženierijas apakšnozare mūsdienās kalpo dažādu darbu automatizācijai. Lai jaunietis kļūtu par veiksmīgu datorinženieri, jau skolas gados ir vēlams izprast vienkārša robota uzbūvi un darbības principus.

Darba mērķis: Apkopot galvenās elektronikas teorijas, kuras ir vajadzīgas līnijsekotāja robota izveidei. Izveidot līnijsekotāju robotu un aprakstīt tā izveides procesu. Uzrakstīt robota vadības programmu, pielāgot to dažādam sensoru skaitam un testēšanas rezultātā noteikt labāko sensoru skaitu līnijsekotājam robotam.

Darba novitāte: Šis darbs var tikt izmantots kā teorētiskais un praktiskais materiāls robotikas pamatu apguvei vidusskolās. Šāds pētījums Druvas vidusskolā tiek veikts pirmo reizi.

Darba uzdevumi:

- Iepazīties ar teoriju, kura ir nepieciešama līnijsekotāja robota izveidei.
- Sagādāt nepieciešamos materiālus un komponentes robota izveidei.
- Izgatavot robota elektronisko shēmu.
- Izgatavot robota mehānisko daļu.
- Izgatavot korpusu un savienot robota elektronisko un mehānisko daļu.
- Uzrakstīt robota vadības programmu.
- Izgatavot trasi, uz kuras var veikt eksperimentus ar līnijsekotāju robotiem.
- Mainot robota sensoru skaitu, pārbaudīt darbībā robotus ar vienu, diviem, trijiem un pieciem sensoriem.
- Veikt robotu ātrdarbības salīdzināšanu, atkarībā no sensoru skaita.

Darbā izmantotās metodes: Informācijas studēšana un eksperimenti.

Darba hipotēze:

Līnijsekotājs robots kurš izmantos 5 sensorus, iespējams nodrošinās stabilāku, ātrāku vadīšanu un pagriezienu izņemšanu, nekā līnijsekotājs ar 2 vai 3 sensoriem.

Darba struktūra: 1. nodaļā ir īsi izklāstīta nepieciešamā teorija, lai izveidotu funkcionējošu līnijsekotāju robotu. 2., 3. un 4. nodaļās ir aprakstīts, kā mums izdevās teoriju pielietot praksē. 5. nodaļā ir aprakstīta robotu testēšana. Pielikumā ir funkcionējoša robota elektriskās shēmas, C++ pirmkods, 2 robotu attēli, 2 līnijsekotāja robota trašu attēli un DVD disks ar robotu testēšanas videomateriāliem.

1. Līnijsekotājs robots

1.1. Līnijsekotāju roboti un to izmantošana robotu sacensībās

Līnijsekotājs robots ir robots, kura galvenais uzdevums ir iespējami ātri un precīzi sekot novilktai līnijai.

Pirmās Robotikas sacensības Latvijā notika 2007 gadā un tajās startēja 23 roboti. Tajās bija tikai divas disciplīnas, līnijsekotāju robotu un sumo robotu sacensības. Līnijsekotāja robota galvenais uzdevums, ir „neizlidot" no trases (nepazaudēt līniju) un finišēt ar iespējami labāko apļa laiku. Sumo robotu disciplīnas uzdevums bija izgrūst pretinieku no sacensību apļa. Pirmajās līnijsekotāju sacensībās piedalījās 17 roboti, no kuriem veiksmīgi finišēja seši roboti. Nākošajās robotikas sacensībās „Robotika 2008" jau bija četras sacensību disciplīnas – līnijasekotāji bez un ar šķērsli, 3kg sumo un Roomba sumo roboti, kuriem par pamatu tiek izmantots putekļu sūcējs Roomba, ko dalībnieks pārveido par cīņas robotu. Dalībnieki bija iedalīti dažādās kategorijās – skolēnu, studentu un inženieru. Pēdējās notikušajās sacensībās startēja 53 roboti no Latvijas, no kuriem 22 bija līnijas sekotāji. Salīdzinot ar pirmajām robotikas sacensībām ir vērojams interesentu pieaugums. „Robotika 2009" sacensības notika piecās disciplīnās – visas iepriekšējo gadu disciplīnas un mini sumo disciplīna. Šajās sacensībās dalībnieki netika dalīti klasēs.

Pirms piedalīties sacensībās dalībniekiem obligāti ir jāiepazīstas ar sacensību noteikumiem, jo katrā sacensību disciplīnā ir savs sacensību nolikums. Ar tiem var iepazīties sacensību mājas lapas http://www.robotika.lv sadaļā disciplīnas un noteikumi. Robotikas sacensībās drīkst piedalīties tikai ar paša konstruētiem robotiem.

1.2. Līnijsekotāja robota darbības princips

Parasti līnijsekotāja robota trasi izveido uz baltas pamatnes uzkrāsojot melnu līniju. Lai robots varētu sekot līnijai, tam vajadzīgas „acis". Acu vietā tiek izmantoti sensori. Uztverto informāciju par trases krāsu sensori nogādā attiecīgajām ierīcēm, kuras šo informāciju apstrādā un tālāk ar programmas palīdzību tiek vadīti 2 robota motori (katrs savam ritenim). Mainot robota riteņu kustības ātrumu var veikt robota pagriezienus un regulēt tā gaitu.

1.3. Līnijsekotāja robota sensoru veidi

Par sensoriem robotos parasti izmanto kādu no uzskaitītajiem variantiem:

- Fototranzistors un infrasarkanā diode.
- Infrasarkanais sensors un infrasarkanā diode.
- Webkamera.

Fototranzistors un infrasarkanā diode ir pats vienkāršākais un arī lētākais variants. Tomēr šo sensoru signālu var ietekmēt apkārtējā gaisma, piemēram fotoaparāta zibspuldze.

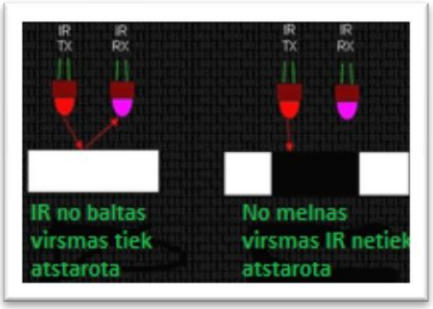
Variants ar infrasarkano sensoru un infrasarkanā diodi ir nedaudz dārgāks, bet to pilnīgi neietekmē apkārtējā gaisma. Šis robots var labi pārvietoties pa līniju arī pilnīgā tumsā.

Modelis ar vebkameru ir ļoti precīzs un var labi pārvietoties pa dažādu krāsu līnijām, bet šī robota kustībai ir grūti uzrakstīt atbilstošu algoritmu. Bez tam robots var labi pārvietoties tikai spožā apgaismojumā.

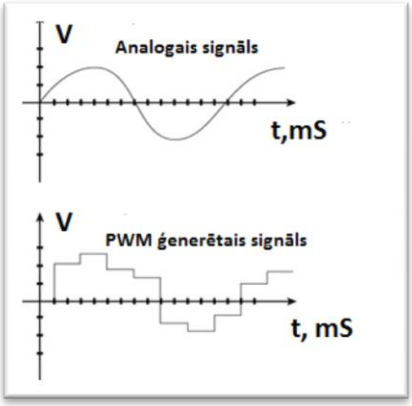
1.4. Datu apstrāde ar fototranzistoru un infrasarkano diodi

Kā līnijsekotājs izmanto sensorus un kā tiek veikta datu apstrāde?

Vispirms pret trases virsmu ar infrasarkanās diodes palīdzību tiek raidīts stars (skat. 1. att.). Atstaroto staru uztver fototranzistors un sūta tālāk strāvas analogo impulsu (spriegumu). Šo analogo signālu uztver ADC (analog to digital converter), kas pārveido analogo signālu par digitālo (skat. 2. att). Tālāk nepieciešams mikrokontrolieris, kas analizē no ADC iegūtos datus un nosaka krāsu, no kādas atstarojas IR stars. No melnās krāsas IR stars neatstarojas, tādēļ līnijsekotājs visbiežāk seko melnai līnijai. Tādā gadījumā mikrokontrolierim vairs neinteresē atstarošanas krāsa, bet tikai tas vai signāls ir atstarots vai nē. Izejot no šiem diviem stāvokļiem mikrokontrolieris tālāk padod strāvu nepieciešamajiem motoriem. Parasti neatstarotu signālu apzīmē ar digitālo 1, bet atstaroto ar 0.



1. att. Līnijas uztveršana ar fototranzistoriem.



2. att. Atšķirība starp PWM signālu un analogo signālu.

1.5. Līnijsekotāja robota algoritmi atkarībā no sensoru skaita

Līnijsekotājos robotos ir iespējams izmantot dažādu sensoru skaitu. Turpmāk tiks apskatīti 3 populārākie sensoru izmantošanas veidi.

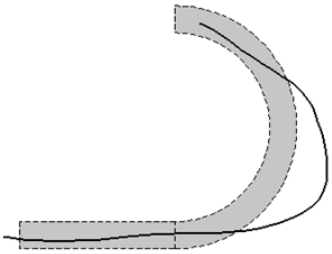
1. Pats vienkāršākais algoritms ar vienu sensoru un tikai ar diviem iespējamiem stāvokļiem (skat. 3. att.):
   1. 1 – redzu līniju;
   2. 0 – neredzu līniju.

3. att. Viena sensora trajektorija.

2. Nedaudz sarežģītāks ar 3 sensoriem (skat. 4. att.):
   1. 001 – gandrīz neredzu līniju, griezties vairāk pa labi;
   2. 010 – līnija ir pa vidu, virzīties pilnā jaudā uz priekšu;
   3. 011 – esmu nedaudz nost no līnijas, griezties pa labi;
   4. 100 – gandrīz neredzu līniju, griezties vairāk pa kreisi;
   5. 110 – līnija atrodas nedaudz pa kreisi, pagriezties nedaudz pa kreisi;
   6. 111 – esmu krustojumā, virzīties pilnā jaudā uz priekšu.

4. att. Sensoru trajektorijas, kad lieto 3 sensorus.

3. Pats sarežģītākais ir veids ar 5 sensoriem. Ar 5 sensoriem ir iespējams ātrāk izbraukt līkumus, it īpaši asus līkumus (skat. 5. att.). Tāpat 5 sensori palīdz ātrāk izlīdzināt robota kustību taisnajos trases posmos:
   1. 11111 – krustojums, virzīties pilnā jaudā uz priekšu;
   2. 00001– gandrīz pazaudēta līnija, samazināt ātrumu un strauji griezties pa labi;
   3. 11100 – straujš līkums pa kreisi, griezties pa kreisi;
   4. 00010 – labās malas sensors ir pa vidu līnijai, griezties pa labi;
   5. 00110– līnija atrodas nedaudz pa labi, pagriezties nedaudz pa labi;
   6. 00100– līnija ir pa vidu, virzīties pilnā jaudā uz priekšu;
   7. 01100 – līnija atrodas nedaudz pa kreisi, pagriezties nedaudz pa kreisi;
   8. 01000 – kreisās malas sensors ir līnijas vidū, griezties pa kreisi;
   9. 00111 – straujš līkums pa labi, griezties pa labi;
   10. 10000 – gandrīz pazaudēta līnija, samazināt ātrumu un strauji griezties pa kreisi.



5. att. Līnijsekotāja robota trajektorija, ja jāizbrauc ass līkums, lietojot 5 sensorus.

2. Robotu izveides priekšnosacījumi

Lai izveidotu savu robotu, bija nepieciešamas teorētiskas zināšanas un arī praktiskas iemaņas. Tāpēc mēs izmantojām iespējas, kas tiek piedāvātas robotu konstruēšanas iesācējiem.

Pirmo līnijsekotāju mēs (darba autori) uzbūvējām kopīgi, Rīgas Tehniskās universitātes Studentu parlamenta, organizētajā pasākumā „Tehniskās jaunrades dienas 2009" Liepājas posmā. Iesākumā bija lekcija par līnijsekotāju robotiem un elektronikas pamatprincipiem. Roboti bija jābūvē komandās pa četriem cilvēkiem. Mēs bijām komandā ar diviem zēniem no Liepājas. Darbus sadalījām atbilstoši prasmēm – mēs lodējām detaļas galvenajā un sensoru platē, bet mūsu komandas biedri taisīja mehānisko daļu. Robota izveide bija salīdzinoši vienkārša, jo tam jau bija sagatavota galvenā plate un ieprogrammēts mikrokontrolieris.

Līnijsekotājus būvēja 11 komandas, bet uzbūvēja tikai 6. Sacensībās mēs ierindojāmies ceturtajā vietā, mūsuprāt mēs varējām startēt labāk, ja nebūtu pieļāvuši vairākas nozīmīgas kļūdas. Šis robots veica savu galveno uzdevumu sekot līnijai, bet nedarīja to pietiekami labi.

No elektroniskās puses nekādu kļūdu nebija, kļūdas bija tikai mehāniskajā daļā. Viena no kļūdām bija tā, ka līnijsekotājs bija uzbūvēts par garu. Robota vidusdaļa bija ieliekusies un saskārās ar trases virsmu, kas ievērojami palielināja berzes koeficientu. To vajadzēja būvēt kompaktāku, lai tas spētu labāk manevrēt. Arī priekšā izvietotā sensoru plate bija novietota pārāk zemu un tas bremzēja kustību. Visas šīs kļūdas ievērojami palēninātja robota kustības ātrumu. Būvējot nākošos līnijsekotājus mēs ievērojām iepriekš pieļautās kļūdas un atkārtoti tās nepieļāvām.

Pirmā robota būvēšana bija ļoti vienkārša, jo tika lietota gatava shēmas plate. Tas tomēr nelikās pietiekoši interesanti un tāpēc nolēmām izveidot savus robotus pašu rokām no sākuma līdz beigām. Turpmākajās sadaļās mēģināsim smalki aprakstīt izveides procesu gan elektronikā, gan mehānikā. Galveno robota elektronisko shēmu var skatīt 2. pielikumā, sensoru plates elektronisko shēmu 3. pielikumā, bet pilnu robota vadības pirmkodu 1. pielikumā.

3. Elektronikas izveide

3.1. Analogo signālu pārveidotājs (ADC)

3.1.1. ADC nepieciešamība robotam

Kā sensorus robotā izmantojām fototranzistorus un infrasarkanās diodes. To darbības principi ir aprakstīti nodaļā 1.4. Tas nozīmē to, ka mūsu robotā ir bija nepieciešams ADC.

3.1.2. Automātiskā konvertēšana

Pirmajā reizē, mēs izmantojām automātiskās konvertēšanas metodi. Saskaņā ar teoriju mums bija jāuzstāda konvertācija 50 – 200 kHz, un to var izdarīt **ADCSRA** reģistrā[1]. Lai varētu salīdzināt cik lielas ir strāvas izmaiņas, Atmega168[2] vajag atskaites punktu. Kā atskaites punktu mēs izmantojām 5V, kurus pievadījām pie **AREF** porta. Normālā režīmā ADC izvada 10 bitu skaitli, bet mūsu mikrokontrolieris spēj uzturēt tikai 8 bitu. Mums nācās uzstādīt ADC tā, lai tas izdotu 8 bitu skaitli. *Datu lapu* lasot uzzinājām, ka ir iespējama automātiskā konvertēšana un to varēja ieslēgt, reģistram **ADCSRA** piešķirot 1 pie **ADATE** un **ADIE**. Katru reizi kad tiek konvertēts analogais signāls par digitālu signālu, mikrokontrolieris izsauca pārtraukumu *ISR(ADC_vect)* un sāk jaunu konvertāciju. Robota vadībai izvēlējāmies *C*++ programmēšanas valodu, jo tā mums bija pazīstama un vienkāršāka par Assembler valodu. Tālāk redzama ADC automātiskās konvertācijas programma *C*++ valodā.

```
Int main()
{
   ADCSRA |= (1 << ADPS1) | (1 << ADPS0); // ADC 50khz - 200khz
   ADMUX |= (1 << REFS0); // Norādām  atskaites punktu AREF
  ADMUX |= (1 << ADLAR); // Pārveidojam par 8 bitu skaitli
  ADCSRA |= (1 << ADATE); //automātiskā konvertēšana
  ADCSRA |= (1 << ADIE); //Ieslēdzam pārtraukumu
  ADCSRA |= (1 << ADEN);  // Ieslēdzam ADC
  ADCSRA |= (1 << ADSC);  // Sākam konvertāciju
  While(1) { }
}
```

Vēlāk uzzinājām, ka automātiskā ADC konvertēšana notiek izmantojot vienu kanālu. Bet, lietojot 5 sensorus, mums ir nepieciešami 5 kanāli, jo katra fototranzistora signāls ir jāapstrādā atsevišķi.

[1] Informāciju par reģistriem var atrast Atmel Atmega 168 datu lapā
[2] 8 bitu mikrokontrolieris, kuru izmantojam robota vadībā

3.1.3. Manuālā konvertēšana

Manuāli konvertējot analogo signālu, nāksies izmantot galveno procesoru. Uzstādījumos izslēdzām **ADATE** un **ADIE** jeb automātisko konvertēšanu un pārtraukumus. Izmantojot automātisko konvertēšanu, katru reizi kad signāls tika konvertēts, sākās nākamā konvertācija. Manuālajā konvertācijā, tagad pašiem jāieslēdz konvertācija, tad jāsagaida līdz tā tiks pabeigta, un tas ir jādara katram kanālam atsevišķi. Tas nozīmē, ka mikrokontrolieris nevar veikt vairākas konvertācijas reizē. Pēc vairākiem mēģinājumiem, mēs nonācām pie šādas programmas:

```
uint16_t ReadADC(uint8_t ch) //Funkcija, kas pieprasītajam kanālam veiks konvertāciju
{
  ADMUX &= 0xF8; // Izdzēšam iepriekšējās konvertācijas datus no operatīvās atmiņas
  ADMUX |= ch; //Kādai no operatīvās atmiņas šūnām, piešķiram jauno kanālu.
  ADCSRA|=(1<<ADSC); //Sākam konvertāciju vienam kanālam
  while(!(ADCSRA & (1<<ADIF))); //Gaidām līdz konvertācija tiks pabeigta
  ADCSRA|=(1<<ADIF); //Izdzēšam ierakstus reģistrā ADIF
  return(ADCH); //Atgriežam konvertācijas datus
}
Int main()
{
   ADCSRA |= (1 << ADPS1) | (1 << ADPS0); // ADC 50khz - 200khz

   ADMUX |= (1 << REFS0); // Norādām  atskaites punktu AREF
  ADMUX |= (1 << ADLAR); // Pārveidojam par 8 bitu skaitli
  ADCSRA |= (1 << ADATE); //Automātiskā konvertēšana
  ADCSRA |= (1 << ADSC);  // Sākam konvertāciju
Uint8_t ch0,ch1,ch2
While(1)
{
  Ch0 = ReadADC(0); //Pirmā kanāla dati, tiks uzglabāti mainīgajā ch0
  Ch1 = ReadADC(1);
  Ch2 = ReadADC(2);
}
}
```

Centāmies izdomāt algoritmu, kas neizmantotu galveno procesoru. Ne vienmēr tas ir iespējams. Šī pieeja pilda savu funkciju, iespējams pat labāk nekā tad, kad izmantotu pārtraukumus, jo izmantojot galveno procesoru tiks ātrāk apstrādāti dati no fototranzistoriem un ātrāk padotas attiecīgās komandas motoriem.

3.2. Impulsu platuma modulācija (PWM)

Impulsa platuma modulāciju izmanto, lai ģenerētu analogo signālu. Vienkāršāk sakot, tas regulē elektrisko spriegumu (V). Noteiktā laika periodā, uz īsu brīdi, mikrokontrolieris padod strāvu motoram (+5V), un tad to atslēdz. Ļoti ātri (uz dažām μs) slēdzot iekšā un ārā motoru, mēs varam regulēt tā apgriezienus.

3.2.1. Pirmais mēģinājums pielietot PWM uz gaismas diodes (LED)

Lai uzprogrammētu PWM, pirms tam ir jāapgūst mikrokontroliera taimeri. Katram mikrokontrolierim ir savādāki taimeri. Mūsu robots izmanto Atmel Atmega 168 (turpmāk Atmega168). Šim mikrokontrolierim ir 3 taimeri – viens 16 bitu un divi 8 bitu taimeri. Taimera ieslēgšana mums nesagādāja problēmas. Sākumā mēs pielietojām 8 bitu taimeri 0 (timer0). Lai šo taimeri ieslēgtu, reģistram **TCCR0B** programmā vajag piešķirt bitu **CS00**. Programmā tas izskatījās šādi:

```
TCCR0B |= (1 << CS00);
```

CS00 nozīmē ka taimeris darbosies ar **FCPU** (mikrokontroliera procesora) ātrumu. Pirmajā reizē mūsu **FCPU** bija 1MHz. Kad **CS00** ir vienāds ar 1, taimeris ir aktīvs. Tagad reģistrs **TCNT0** skaita no 0 līdz 255, jo taimeris 0 ir 8 bitu taimeris, un mainīsies līdz 255, jo $2^8 = 256$. Kad bijām iemācījušies palaist taimerus, mēs uzprogrammējām vienkāršu PWM. Pirmā PWM koda fragments izskatījās šādi:

```
if(TCNT0 > cik) PORTB = 0b00000000; //Ja TCNT0 ir lielāks par cik, tad izslēgt visas gaismas diodes.
else PORTB = 0b11111111; //Citādi tās ieslēgt!
```

Mainīgais **cik** ir atbildīgs par gaismas diodes spožumu. Šo spožumu var aprēķināt pēc šīs formulas $gaišums = \frac{cik}{255} * 100\%$. Piemēram, ja mums **cik=128**, tad gaišums būtu aptuveni 50%, jeb motors strādā uz pusi no maksimālās jaudas.

Bet, šī metode izmanto galveno procesoru, lai ģenerētu PWM. Robotam šis vienkāršais algoritms pilnībā neder:

1. Jo galvenajam procesoram vajag apstrādāt sensora datus
2. Vadīt ekrānu
3. Vadīt motorus

Jo vairāk uzdevumus veiks galvenais procesors, jo lēnāk tiks apstrādāti dati un PWM signāls netiks pietiekami ātri ģenerēts. Motoriem nevis samazināsies apgriezieni, bet tiks apstādināts un tad atkal palaists. Vienkāršāk sakot, galvenais procesors ir par lēnu lai uzturētu visas šīs darbības.

3.2.2. PWM ģenerēšana izmantojot pārtraukumus

Pēc daļēji veiksmīga PWM , mēs apguvām, kā var izdarīt to pašu, neizmantojot galveno procesoru. Šī metode saucās *pārtraukumi*.No datu lapas mēs uzzinājām, ka Atmega168 uztur 26 pārtraukumus.

Šajā metodē, izmanto iebūvēto taimera procesoru.. Šobrīd mēs esam iemācījušies izmantot *Compare on match* un *Overflow* pārtraukumus.

Overflow pārtraukums pārbauda **TCNT0**, ja šis reģistrs ir sasniedzis maksimālo vērtību (šajā gadījumā 255) tiek izsaukta *funkcija* **TIMER0_OVF_vect**. Programma izskatās šādi:

```
#include <avr/interrupt.h> // Ielādējām pārtraukumu bibliotēku
Int main()
{
  TCCR0B |= (1<<CS00);
  TIMSK0 |= (1<<TOIE0); // Aktivizējam Overflow pārtraukumu
sei(); //Aktivizējam globālos pārtraukumus
}
ISR(TIMER0_OVF_vect)
{
  PORTB = ~PORTB; //Apgriežam bitus pretējā stāvoklī 0 >> 1 un 1 >> 0
}
```

Programmas piemērā, katru reizi kad **TCNT0** *pārplūdīs*, jeb būs vienāds ar 255, tiks ieslēgta vai izslēgta gaismas diode.

Compare on match pārtraukuma gadījumā, funkcija tiek izsaukta nevis, kad pārplūst **TCNT0**, bet kad **OCR0x** vērtība ir vienāda ar **TCNT0**. Uz viena taimera var būt divi **OCR0x** reģistri, atmegas168 gadījumā – OCR0A un OCR0B. Programma ar šādu pārtraukumu izskatās šādi:

```
#include <avr/interrupt.h>
Int main()
{
  TCCR0B |= (1<<CS00);
  TIMSK0 |= (1<<OCIE0A); // Aktivizējam compare on match pārtraukumu, un mūsu reģistrs salīdzināšanai būs
OCR0A
OCR0A = 128;
sei(); //Aktivizējam globālos pārtraukumus
}
ISR(TIMER0_COMPA_vect) //Tiek izsaukts, kad OCR0A = TCNT0
{
  PORTB = ~PORTB; //Apgriežam bitus pretējā stāvoklī 0 >> 1 un 1 >> 0
}
```

Programmas piemērā katru reizi, kad **TCNT0 = 128** izsauks *pārtraukumu* **TIMER0_COMPA_vect**, tas apgriezīs bitus pretējā stāvoklī.

Kad bijām apguvuši šīs divas metodes, mēs izdomājām, kā uz šiem pārtraukumiem uzprogrammēt PWM. Kad **TCNT0=OCR0A**, mēs izslēdzam gaismas diodes, bet, kad notiek *Overflow* jeb **TCNT0=255**, mēs tās ieslēdzam, tādā veidā ģenerējot PWM. Jaunā programma, PWM signāla ģenerēšanai izskatījās šādi:

```
#include <avr/interrupt.h>
Int main()
{
  TCCR0B |= (1<<CS00);
  TIMSK0 |= (1<<OCIE0A) | (1<<TOIE0); // Aktivizējam abus pārtrakumus
OCR0A = 128; //šis reģistrs ir tas pats, kas mainīgais cik
sei(); //Aktivizējam globālos pārtraukumus
}
ISR(TIMER0_COMPA_vect) //Tiek izsaukts, kad OCR0A = TCNT0
{
  PORTB &= ~PORTB; //Izslēdzam visas gaismas diodes
}
ISR(TIMER0_OVF_vect)
{
  PORTB |= PORTB; // Ieslēdzam visas gaismas diodes
}
```

Pēc eksperimenta secinājām, ka šī metode ir vēl sliktāka nekā pirmā. Programmā kļūdu nav, bet mikrokontrolieris nespēj pietiekami ātri izsaukt pārtraukumus. Ar gaismas diodēm tas bija viegli pamanāms.

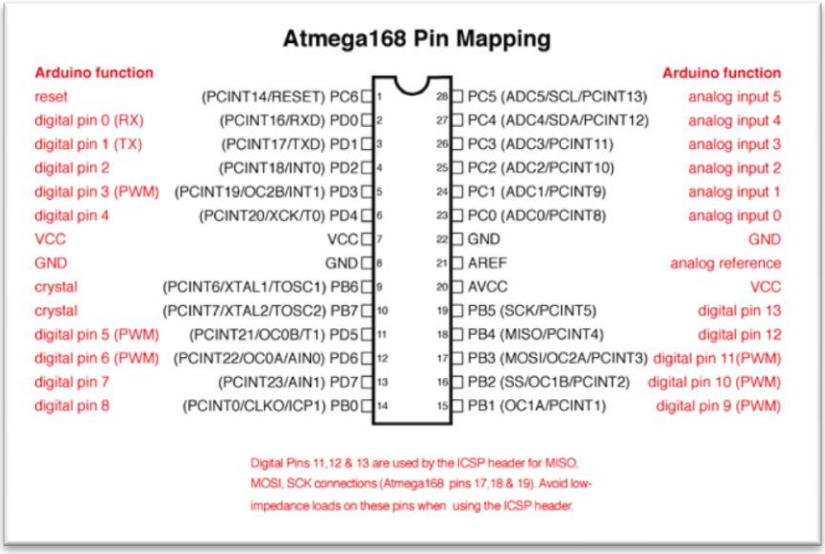
3.2.3. PWM ģenerēšana izmantojot *fast PWM* un WGM metodi

Pirmās divas PWM ģenerēšanas metodes varam uzskatīt par neizdevušām. Trīs lietas, labas lietas. Šoreiz izmantojām *Waveform generation mode* (Viļņa veida ģenerēšanas režīms) un *fast PWM*, kas izrādījās ir iebūvēts Atmega168 mikrokontrolierī. Uzkonfigurējot **TCCR0B** un **TCCR0A** reģistrus, var aktivizēt automātiskos PWM izvada portus un Atmega168 tādi ir 6, bet mums vajadzīgi tikai divi.

Pēc teorijas katram reģistram **OCR0x, OCR1x, OCR2x** ir savs PWM izvada ports (skat. 6. att.). Tā kā mēs izmantojam taimeri 0 (timer0), šī taimera reģistram **OCR0A** atbilst iebūvētais PWM ports **OC0A**, kurš pēc dotās shēmas ir 12. ports. Savukārt **OCR0B** iebūvētais PWM ports ir **OC0B**, kurš shēmā ir 11. Šīs metodes galvenā priekšrocība ir tā, ka minimāli tiek izmantoti procesora resursi, jo varam iztikt bez *pārtraukumiem***.** Tas ļoti paātrina PWM, jo iepriekšējā eksperimentā, izmantojot *pārtraukumus*, ar gaismas diožu

palīdzību varējām novērtēt cik lēni tas strādā. Izpētījām Atmega168 *datu lapu* un nokonfigurējām taimeri 0 *Fast PWM* režīmā. Šis režīms pats ģenerē šo PWM signālu.

Ar to konfigurācija nebeidzas, mikrokontroliera atmiņā vilnis ir izveidojies, bet netiek automātiski izvadīts. Lai to izvadītu, mums taimerī 0 bija jāieslēdz *Compare output mode* (salīdzinot izvadīt režīms). Lai šo režīmu nokonfigurētu, ir vajadzīgi vēl divi reģistri – **COM0x1** un **COM0x0**. Pēc PWM teorijas un iepriekšējiem eksperimentiem, kad **OCR0x** > **TCNT0,** mums vajag, lai gaismas diode vai motors ir izslēgts, bet kad **OCR0x < TCNT0** gaismas diode vai motors ir jāieslēdz. Vadoties pēc iepriekš minētajiem nosacījumiem un Atmega168 *datu lapas*, noskaidrojām, ka **COM0x1=1** un **COMx0=0**. Balstoties uz *fast PWM* un WGM zināšanām, tika izgatavota jauna programma:



6. att. Atmega168 portu funkcijas.

```
#include <avr/io.h>
Int main()
{
  TCCR0A |= (1 << WGM01) | (1 << WGM00) | (1<<COM0A1); //Taimeri 0 nokonfigurējam par fast PWM un nokonfigurējam automātisko izvadīšanu
  TCCR0B |= (1<<CS00); //Ieslēdzam taimeri 0
  OCR0A = 128; //Samazinām elektrisko spriegumu par 50%
While(1) { } //Bezgalīgais cikls
}
```

Beidzot mums izdevās uzprogrammēt PWM ģeneratoru, kas strādā minimāli izmantojot galveno procesoru un neizmantojot *pārtraukumus*. Tagad mēs spējam vadīt motora apgriezienus. Apgriezienu regulēšana ļaus robotam ātrāk izpildīt pagriezienus, tam nevajadzēs pilnībā apstāties, lai veiktu pagrieziena manevru. Aprēķināt nepieciešamo **OCR0x** vērtību motora jaudas regulēšanai var pēc formulas:

$OCR0x = \frac{Jauda\ \%}{100} * 255$. Tomēr jāatceras, ka **OCR0x** var būt tikai vesels nenegatīvs skaitlis.

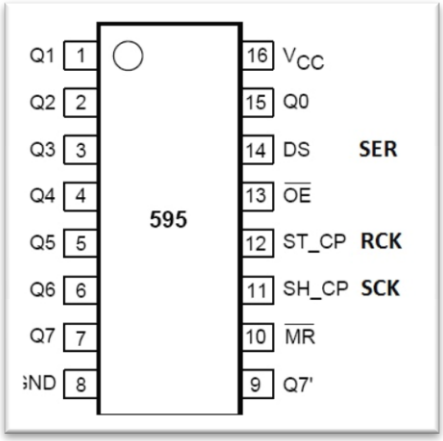
3.3. 74hc595 pārbīdes reģistrs

3.3.1. Pārbīdes reģistra funkcija

Katram mikrokontrolierim ir ierobežots izvades un ievades portu skaits. Piemēram, Atmega168 ir tikai 23 IO (ievades un izvades) porti. Ja vajadzīgs lielāks regulējamo ierīču skaits, tad lieto izvades pārbīdes reģistrus. Šis pārbīdes reģistri tiek dēvēti arī par *portu paplašinātājiem.* Piemēram, izmantojot 74hc595, ar trīs Atmega168 portiem var atsevišķi kontrolēt vairāk kā 1000 gaismas diodes. Robota sensoru kalibrēšanai mēs bijām nolēmuši izmantot displeju. Ja nebūtu *porta paplašinātāja*, mums nepietiktu porti, lai vadītu displeju. Lielākā daļa porti tiek izmantoti motoru vadībai un sensoriem.

3.3.2. Pirmais mēģinājums izmantot *porta paplašinātāju*

Izpētot *datu lapu,* mēs uzzinājām, ka portu paplašinātāja 74hc595 mikrokontrolierim ir astoņi izvadi. Lai palielinātu izvadu skaitu, mikrokontrolierus var savienot vienu aiz otra. Lai nosūtītu datus mikrokontrolieriem (74hc595) izmanto trīs portus – SCK, RCK un SER (skat. 7. att.). Ja mēs gribam gaismas diodi ieslēgt, tad vispirms SER portam pievadam +5V un apstiprinām izvēli, atslēdzot un pievadot +5V SCK portam. Šo procesu mēs atkārtojam tik reizes, cik mums ir izvadi. Piemēram, ja mums ir divi portu paplašinātāji 74hc595, un katram no tiem ir 8 izvadi, tad procesu atkārtojam 16 reizes. Kad visi biti nosūtīti, portam RCK pievadam +5V, un tad visiem trīs piešķiram 0V, jo ir jāsagatavo portu paplašinātājs nākamajiem datiem.



7. att. 8 bitu pārbīdes reģistra portu izvietojums

Mūsuprāt, koncepts ir ļoti vienkāršs, izmantojot *char* (8bitu mainīgais, kas spēj uzturēt vienu simbolu) masīvu, mēs uzrakstījām programmu. Mūsu programma salīdzina vai attiecīgais *char* masīva elements ir 0 vai 1, un atkarībā no šī stāvokļa dod signālu SER portam.

```
void writeout(char biti[16])
{
  for(int unsigned i=0; i<Skaits; i++) //Ciklā programma ieies 16 reizes, jo eksperimentā izmantojam divus 74hc595
  {
    if(biti[i] == '1') {   //Ja i'tais bits ir 1
      PORTB = (1<< SER); //tad SER padodam +5v
        PORTB = (1<< SCK); //Apstiprinām savu izvēli
```

```
    }
    else //Ja i'tais bits nav 1
    {
      PORTB = (1>> SER); //SER portam pievadam 0v
        PORTB = (1<< SCK);    //Apstiprinām izvēli
    }
        PORTD |= (1 << RCK);
  }
```

Šī metode būtu laba, ja vienīgi viens *char* elements neaizņemtu 8 bitus. Tas nozīmē ka 16 *char* elementi aizņem 128 bitus jeb 16 baitus. Mūsu Atmega168 ir ierobežots atmiņas daudzums - tikai 4 KB. Tāpēc nospriedām, ka šī metode neder!

3.3.3. Programmas pirmkoda optimizēšana

Kā jau iepriekš minējām, izmantojot *char* masīvu tiek aizņemts pārāk daudz mikrokontroliera atmiņas. Tā kā Atmega168 ir 8 bitu mikrokontrolieris, tad mums radās ideja, izmantot vienu 8 bitu mainīgo, kurš sevī ietvertu 8 portu jeb gaismas diožu stāvokļus. Lai Atmega168 ar šādu metodi uzglabātu 16 portu stāvokli, vajag tikai 16 bitus jeb 2 baitus!

Pirmie testi jaunajai programmai bija veiksmīgi. Ar trim portiem varējām vadīt 16 gaismas diodes. Izveidojām testēšanas plati ar četriem 74hc595 pārbīdes reģistriem, pārveidojām programmas pirmkodu, lai tas nosūtītu 32 stāvokļus. Uz 16 izvadiem viss strādāja pareizi, bet, kad pārbaudījām programmu ar 32 izvadiem, iedegās nepareizās diodes. Vienkārši sakot, dati tika izvadīti nepareizi. Ar citu programmētāju palīdzību atradām kļūdu, mēs nepareizi lietojām *bit shift* jeb bitu pārbīdes metodi. Tad, kad mums vajadzēja izmainīt stāvokli tikai SER portam, mēs izmainījām arī RCK stāvokli. Apguvuši *bit shift,* izlabojām kļūdu un uzrakstījām pirmkodu, kas strādāja bez kļūdām.

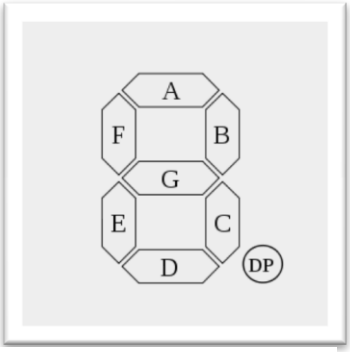
```
void write_595(uint8_t data)
{
  for(int i=0; i<Skaits; i++)
  {
        if((data & (1<<i)) )   // Ja bita numurs „data" ir viens tad
        {
              PORTD |= (1<<SER); //Gaismas diodi ieslēdzam
        }
        Else //ja ir 0
        {
              PORTD &= ~(1<<SER); //Gaismas diodi izslēdzam
        }
        PORTD &= ~(1 << SCK);
```

```
            PORTD |= (1<<SCK);
        }
      PORTD |= (1 << RCK);
        PORTD &= ~(1 << RCK);
        PORTD &= ~(1 << SCK);
        PORTD &= ~(1 << SER);
    }
```

3.4. Septiņu segmentu displejs

3.4.1. Īsumā par displeju

Lai kalibrētu sensorus, mikrokontrolierim jāizvada dati cilvēkam saprotamā valodā. Visvienkāršākais ekrāna veids ir LED 7 segmentu displejs, bet ir vēl LCD (šķidro kristālu displejs), mirdzizlādes zīmju indikātors, plazma displejs, e-tintes displejs un citi. Viens kvadrāts 7 segmentu displejam sastāv no septiņiem segmentiem, un šos segmentus dēvē ar burtiem no A līdz G (skat. 8. att.). Populārākie 7 segmentu displeji sastāv no gaismas diodēm. Datu attēlošana notiek pavisam vienkārši, katram skaitlim ir jāiededz savi segmenti, piemēram, lai attēlotu ciparu viens, ir jāiededzina B un C segments.



8. att. Katra segmenta nosaukums, displejam.

3.4.2. Pirmais eksperiments ar displeju

No veca satelīta uztvērēja, izlodējām 4x7 segmentu displeju (skat. 9. att. un 4. pielikumu). Lai samazinātu izvadu skaitu, šim displejam segmenti tiek dalīti tā, ka atsevišķi var vadīt tikai kvadrātu. Tas nozīmē, ka informācijas attēlošana notiek lielā ātrumā ieslēdzot un izslēdzot kvadrātus. Tā arī mēs darījām, ieslēdzām kvadrātus ar frekvenci 31250 Hz, bet displejs dega ļoti blāvi. Mēs domājām, ka jāpalielina frekvence un palielinājām uz 250000 Hz. Spožums palielinājās, bet ne pietiekami. Izrādās, ka nevajadzēja palielināt frekvenci, bet to samazināt jeb attēlot ciparu, lai cilvēka acs šo ciparu ieraudzītu. Tagad mūsu ekrāna frekvence ir aptuveni 60 Hz. Lai samazinātu aizņemto IO portu skaitu, displeju regulējam izmantojot pārbīdes reģistru – 74hc595.

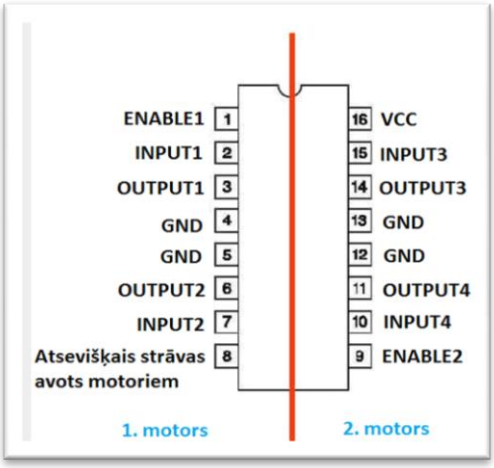
9. att. Vecs satelīta uztvērēja 7 segmentu displejs.

3.5. Motoru vadības bloks L293D

3.5.1. L293D nozīme robotam

Pašā sākumā mēs domājām motoru slēgt tieši pie *Atmega168,* bet ar motoriem ir vairākas problēmas. Viena no problēmām ir tā, ka motori, iedarbinot tos no miera stāvokļa, patērē ļoti daudz strāvas. Tāpēc ieslēdzot motorus var nepietikt strāva mikrokontrolierim. Šī parādība saucās „pīķa efekts", un šo „pīķa efektu" var atrisināt divos veidos, izmantojot elektrolītisko kondensatoru vai pievienojot motoriem atsevišķu strāvas avotu.

Ar elektrolītisko kondensatoru mēs kompensējam problēmu, kad baterijai trūkst kapacitāte, lai ieslēgtu motoru un nodrošinātu ar strāvu mikrokontrolieri. Savukārt ar L293D, mēs nodrošinām atsevišķu strāvas avotu motoriem (skat. 10. att.). Mēs izvēlējāmies atdalīt strāvas avotu, jo plānojam piedalīties sacensībās ar jaudīgākiem motoriem, piemēram, ar 7.5V motoriem, bet Atmega168 maksimālais spriegums ir 5V.



10. att. L293D portu veidi.

3.5.2. Kā izmantot L293D

Principā L293D ir ļoti viegli lietot. L293D spēj vadīt divus motorus. Ja mēs gribam vadīt **pirmo motoru**, mums jāpadod signāls portam **ENABLE1**. Pēc tam mēs varam norādīt uz kuru pusi griezīsies motors, pievadot attiecīgos bitus portiem **INPUT1** un **INPUT2** (skat. 1. Tabulu)**.**

1. tabula

Iespējamie motora stāvokļi mainot porta INPUT1 un INPUT2 bitus

INPUT1	INPUT2	Rezultāts
0V	0v	Motors negriežas
0V	5v	Motors griežas pretēji pulksteņrādītāja kustības virzienam
5V	0v	Motors griežas pulksteņrādītāja kustības virzienā
5V	5v	Motors negriežas

Otru motoru arī var regulēt tieši tāpat kā pirmo, vienīgi jāmaina stāvokļi portiem – **ENABLE2, INPUT3, INPUT4.**

3.5.3. Problēmas ar L293D

Mums izdevās pieslēgt motorus pie L293D, bet nevarējām kontrolēt motoru apgriezienus. Mēs visādi izmēģinājām, padevām PWM signālu gan pie **INPUT1,** gan **INPUT2**, nekādi panākumi, vienmēr motors traucās ar pilnu jaudu. Problēmas atrisinājumu meklējām L293D *datu lapā*. Izrādījās ka PWM signāls ir jāpadod **ENABLE** portiem.

Tad, kad robotu izveidojām, tam motori negriezās. Mēs veicām mērījumus visā shēmā un elektriskais spriegums visur bija optimāls 4,7 V. Tomēr robots nekustējās. Vēlreiz pārlasījām *datu lapu* motoru vadības blokam L293D un noskaidrojām, ka atsevišķā strāvas avotā elektriskais spriegums (VS) nedrīkst būt zemāks par L293D barošanās elektrisko spriegumu (VSS).

4. Mehānikas izveide

4.1. Piedziņa

Sākumā mums bija izvēle iegādāties servo motorus vai līdzstrāvas motorus, bet mēs nolēmām, ka servo motori pirmajam robotam būs pārāk dārgi. Tāpēc nolēmām iegādāties līdzstrāvas motorus. Pirmā problēma bija tāda, ka vienkāršam līdzstrāvas motoram ir ļoti liels apgriezienu skaits, tāpēc mums bija jāizdomā kā tos samazināt. Sākumā bija ideja iegādāties motorus ar iebūvētiem pārnesumiem, kas samazina apgriezienu skaitu. Tā nebija slikta ideja, bet aizbraucot uz „Tehniskās Jaunrades Dienām" mēs ieraudzījām citu risinājumu šai problēmai. Robota piedziņā izmantojām dubulto pārnesumkārbu (skat 11.att.), kurai vienā korpusā ir iebūvēti divi līdzstrāvas motori un katram motoram savs pārnesums. Šos pārnesumus var salikt četrās attiecībās – 344.2:1, 114.7:1, 38.2:1 un 12.7:1.

11. att. Dubultā ātrumkārba, kas spēj vadīt atsevišķi divus motorus.

Tukšgaitā mūsu dzinēji veic 12300 apgriezienus minūtē, bet ar šo neatkarīgo pārnesumkārbu apgriezienus varam regulēt, attiecīgi uz – 38, 115, 345, 1039 apgriezieniem minūtē. Tas ir ļoti labi, jo, izvēloties motorus bez pārnesumiem, tikai ar impulsa platuma modulāciju, mēs nevarētu pietiekami precīzi noregulēt motora ātrumu.

Mēs izvēlējāmies otro ātrumu, 115 apgriezienus minūtē, jo pirmais ir mazliet par lēnu, bet ar trešo un ceturto būs grūti noturēties trasē, jo tie būs nedaudz par ātru.

Informāciju par griezes momentu var apskatīties pārnesumkārbas līdzi esošajā informācijas lapā. Mūsu kārbas salikumā tas ir 0.079 Nm (Ņūtonmetri).

4.2. Līdzstrāvas elektriskie motori

Mēs saviem robotiem izmantojam līdzstrāvas motorus (skat 12. att.), kas bija iekļauti pārnesumkārbas komplektā. Šos motorus, nepareizi pievienojot pie elektriskās strāvas, nav iespējams sabojāt, jo atkarībā no tā, kur būs pievienots pozitīvais un negatīvais pols mainīsies tikai motora ass griešanās virziens. Motoriem ir jāpievada to noteiktais spriegums, jo, ja elektriskais spriegums būs pa mazu, tad tā jauda netiks pilnībā izmantota, bet, ja spriegums būs lielāks, tad motora

12. att. Līdzstrāvas motors ar elektrisko spriegumu 3V.

kalpošanas laiks būs daudz īsāks. Iegādājoties šāda veida dzinējus uzmanība ir jāpievērš diviem galvenajiem faktoriem – elektriskajam spriegumam un griezes momentam.

Ja spriegums būs lielāks, tad arī griezes moments būs lielāks. Ja spriegums būs pārāk liels, tad arī barošanā vajadzēs izmantot vairākus barošanas elementus, kas ievērojami palielinās svaru. Parasti robotikā izmanto 6-12 voltu dzinējus. Mēs pagaidām izmantojam 3 voltu dzinējus, bet kad piedalīsimies robotikas sacensībās, tad noteikti izmantosim jaudīgākus motorus.

4.3. Riteņi

Ar riteņiem mums nekādu problēmu nebija, jo tos pasūtījām no tā paša ražotāja, no kura sūtījām pārnesumkārbu. Mūsu gadījumā (skat 13.att.), riteņi ar pārnesumkārbu ir ļoti viegli savienojami, bet ja riteņi nav pirkti no viena ražotāja, tad gan var rasties problēmas ar to piestiprināšanu pie pārnesumkārbas ass.

13. att. „Tamiya racing" ritenis, kura diametrs ir 5,50 cm.

Riteņiem ir jābūt ar mīkstu gumijas apvalku, lai robots neslīdētu un uzsākot kustību neizslīdētu. Ar lielākiem riteņiem būs sasniedzams lielāks ātrums nekā ar mazākiem, bet, ja būs izmantoti jaudīgi dzinēji, tad pat ar ļoti maziem riteņiem var sasniegt ļoti lielu ātrumu. Lielākiem riteņiem ir nepieciešama lielāka jauda lai uzsāktu kustību. Lai aprēķinātu kustības ātrumu, vispirms ir jāaprēķina riteņa apkārtmērs pēc formulas - $C=2\pi R$. Pēc tam tas ir jāreizina ar apgriezienu skaitu minūtē. Mūsu gadījumā sanāk šādi – 2*3.14*2.75*115=1986.05 centimetri/minūtē. Tas ir apmēram 0.33 m/s.

4.4. Virsbūve

Virsbūvi katrs var izgatavot tādu, kādu pats vēlas, bet, ja būs vēlēšanās startēt robotikas sacensībās, tad noteikti nolikumā ir jāizlasa, kādi ir maksimāli noteiktie līnijsekotāja gabarīti. Mūsu gadījumā visu var būvēt virsū pārnesumkārbai un tāpēc nerodas problēmas. Bet, ja piedziņai tiks izmantoti divi neatkarīgi motori, kuri nav iebūvēti vienā korpusā, tad noteikti vajadzēs piedomāt pie karkasa uzbūves. Visas robota komponentes var iebūvēt kādā kārbā vai pielodēt pie shēmas plates. Ļoti interesanti varētu būt, ja līnijsekotāju iebūvētu kādā vecā rotaļu automašīnas modeļa korpusā. Protams, labāk izvēlēties vieglākus materiālus, lai robots būtu vieglāks un kustīgāks.

5. Robotu testēšana

Vairāku mēnešu kopīga darba rezultātā tika izveidoti 2 roboti, kuriem tika piešķirti arī vārdi. Raivja robots tika nosaukts par Gutsy, bet Toma robots par Jaunty. Viena robota izmaksas bija aptuveni 35 Ls.

Robotu mehāniskās daļas ir atšķirīgas, jo katrs no darba autoriem sava robota mehānisko daļu veidoja pats. Būtiskākā atšķirība ir tā, ka Raivja robotam ir nedaudz mazāks attālums no sensoru plates līdz riteņu asij (aptuveni 3 mm). Jau iepriekš paredzējām, ka tas varētu ietekmēt robota rezultātus dažādās trasēs ar dažādu sensoru skaitu. Robotu galvenās plates ir vienādas un arī vadības programma abiem robotiem sākumā bija viena.

Sensoru platē ir ielodēti 5 sensori, bet testos bijām nolēmuši izmantot dažādus sensoru skaita variantus. Sensoru skaitu nolēmām mainīt ar programmas palīdzību. Lai to varētu ērti izdarīt, bija nepieciešams izveidot ierīci, kas atļauj no datora manīt robota programmu (programmators). Šāda ierīci arī izveidojām, izmantojot datora paralēlo portu. Ierīces shēmu var redzēt 7. pielikumā. Mūsu programmators sastāv no 2 daļām, no kurām viena ir iebūvēta galvenajā platē. Tas nodrošina veiksmīgu pieslēgšanos robotam, neizņemot mikrokontrolieri.

Roboti tika testēti 2 trasēs. Pirmā trase tika izveidota pietiekami vienkārša, bet otrajā trasē tika izveidots krustojums un iekļauts pagrieziens taisnā leņķī.

Robotu ātrdarbību testējām abās trasēs ar 1, 2, 3 un 5 sensoriem. Vienkāršajā trasē abi roboti strādāja ar vienādu programmu, bet sarežģītajā trasē Toma robotam bija nepieciešams pielāgot programmu 5 sensoru algoritmam, jo programma sākotnēji tika izveidota Raivja robotam.

Visus testus veicām 3 reizes vienā un otrā virzienā ar katru sensoru skaitu. Apkopojām vidējos apļu laikus (skatīt 2. tabulā).

2. tabula

Apļu laiki abiem robotiem atkarībā no sensoru skaita un trases

Sensoru skaits	1.trase (Gutsy)	1.trase (Jaunty)	2. trase (Gutsy)	2. trase (Jaunty)
5	25.41	27.01	28.32	27.48
3	25.69	27.56	27.85	26.65
2	40.57	41.15	43.99	38.57
1	43.65	41.27	nebeidza trasi	nebeidza trasi

Skatoties tabulā, vērojami interesanti fakti.

Atšķirības starp 3 un 5 sensoriem ir ļoti niecīgas. Tomēr redzams, ka robots ar 5 sensoriem pārvietojas ātrāk un stabilāk, ja trasē nav asi pagriezieni. Skatīt 8. pielikuma videofailu „5sensori_gutsy_pirma_trase.avi". Savukārt ar 3 sensoriem robots ātrāk noreaģēja asos pagriezienos, bet nevarēja ātri nostabilizēties taisnajos trases posmos. Skatīt 8. pielikuma videofailu „3sensori_gutsy_pirma_trase.avi".

Labāku reakciju uz asiem līkumiem 3 sensoru gadījumā var skaidrot ar to, ka galējais sensors dod signālu asam pagriezienam, bet 5 sensoru gadījumā šis pats sensors dod signālu kustības nostabilizēšanai. Tieši tā paša iemesla dēļ, nelielos pagriezienos un taisnajos posmos, 5 sensoru robots būs ātrāks.

Robotam ar 2 sensoriem bija ļoti lēns apļa laiks, jo tas nevarēja atšķirt taisnos posmus no pagriezieniem un brauca ļoti izteiktos „zigzagos". Skatīt 8. pielikuma videofailu „2sensori_jaunty_otra_trase.avi".

Robotam ar 1 sensoru bija vēl lēnāks apļa laiks, bet pirmajā trasē tas brauca precīzi pa līniju. Skatīt 8. pielikuma videofailu „1sensors_gutsy_pirma_trase.avi". Savukārt, otrajā trasē abi roboti izbrauca no trases un nespēja tajā atgriezties. To mēs skaidrojam ar otrās trases melnās līnijas slikto kvalitāti. „1sensors_gutsy_nav_pietiekami_melna_linija.avi".

Jau minējām, ka otrajā trasē Toma algoritmam bija nepieciešams veikt izmaiņas 5 sensoru variantam. 8. pielikuma videofailā „5sensori_jaunty_ar_gutsy_algoritmu.avi" var redzēt kā Toma robots kļūdaini pārvietojas ar Raivja robota algoritmu.

Toma robota uzvaru otrajā trasē, bet zaudējumu pirmajā trasē skaidrojam ar Toma robota mehāniskās konstrukcijas (sensoru plates attālums līdz riteņu asij) labāku piemērotību otrajai trasei.

Secinājumi

1. Sava pirmā robota izveides process ir ļoti darbietilpīgs, jo tam veltījām aptuveni 5 mēnešus pa 10-15 stundām nedēļā.
2. Darba procesā pārliecinājāmies, ka datorinženiera profesija ir mums piemērota un veikt šo darbu mums bija ļoti interesanti.
3. Ar izgatavotajiem robotiem esam nolēmuši piedalīties robotikas sacensībās, kas notiks 2010. gada 1. maijā.
4. Robota kustību būtiski ietekmē sensoru plates novietojums līdz riteņu asij.
5. Robota kustības ātrums būtiski atšķiras dažādās trasēs.
6. Robota kustības algoritms ir jāpielāgo robota mehāniskajām īpatnībām.
7. Pētījuma rezultāti daļēji apstiprina izvirzīto hipotēzi, ka robots izmantojot 5 sensorus, brauks ātrāk un stabilāk, nekā izmantojot 3 vai 2 sensorus.

Uzziņas avotu saraksts

1. http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf - skatīts 15.09.2009
2. http://www.wrighthobbies.net/guides/linefollower.htm – skatīts 07.12.2009
3. http://extremeelectronics.co.in/avr-tutorials/timers-in-compare-mode-part-i/ - skatīts 10.12.2009
4. http://extremeelectronics.co.in/avr-tutorials/introduction-to-pwm-pulse-width-modulation/ - skatīts 11.12.2009
5. http://extremeelectronics.co.in/avr-tutorials/programming-in-c-tips-for-embedded-development/ - skatīts 15.12.2009
6. http://www.syntax.com.tw/proddata/IC/IC-74595.JPG - skatīts 26.12.2009
7. http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf - skatīts 27.12.2009
8. http://www.windmeadow.com/node/20 - skatīts 26.12.2009
9. http://extremeelectronics.co.in/avr-tutorials/using-the-analog-to-digital-converter/ - skatīts 03.01.2010
10. http://www.freewebs.com/isuru-c/motor%20driver.JPG –skatīts 01.02.2010
11. http://www.datasheetcatalog.com/datasheets_pdf/L/2/9/3/L293D.shtml - skatīts 09.02.2010
12. http://www.notepad.lv/viewtopic.php?t=1781 – skatīts 01.02.2009

Anotācija

Zinātniski pētnieciskā darba temats – Sensoru skaita ietekme uz līnijsekotāja robota kustību.

Darba izpildītāji – Saldus novada Druvas vidusskolas 11. E. Klases skolnieki Raivis Strogonovs un Toms Friķis.

Darba vadītājs – informātikas skolotājs Normunds Svētiņš.

Izstrādātais darbs sastāv no 25 lapaspusēm, 13 attēliem, 2 tabulām un 8 pielikumiem.

Zinātniski pētnieciskā darba mērķis:

Apkopot galvenās elektronikas teorijas, kuras ir vajadzīgas līnijsekotāja robota izveidei. Izveidot līnijsekotāju robotu un aprakstīt tā izveides procesu. Uzrakstīt un testēšanas rezultātā noteikt labāko līnijsekotāja robota darbības algoritmu.

Izvirzītie darba uzdevumi:

1. Iepazīties ar teoriju, kura ir nepieciešama līnijsekotājs robota izveidei.
2. Izgatavot katram savu līnijsekotāju robotu.
3. Izgatavot trasi, uz kuras var veikt eksperimentus ar līnijsekotāju robotiem.
4. Mainot robota sensoru skaitu un uzrakstot atbilstošus algoritmus, pārbaudīt darbībā robotus ar diviem, trijiem un pieciem sensoriem.
5. Veikt visu algoritmu un tiem atbilstošo robotu darbību salīdzināšanu.

Darba hipotēze:

Līnijsekotājs robots kurš izmantos 5 sensorus, brauks stabilāk un ātrāk, nekā līnijsekotājs ar 2 vai 3 sensoriem.

Zinātniski pētnieciskā darba metodes: Informācijas studēšana un eksperimenti.

Pētījuma rezultāti daļēji apstiprina izvirzīto hipotēzi, ka robots izmantojot 5 sensorus, brauks ātrāk un stabilāk, nekā izmantojot 3 vai 2 sensorus.

Annotation

Scientific research subject – Number of sensors influence on line follower robot's, movement.

Work performers – Saldus district Druvas High school 11. E. Class students Raivis Strogonovs and Toms Friķis.

Supervisor - Informatics Teacher Normunds Svētiņš

The developed work consists of 25 pages, 13 pictures, 2 tables and 8 attachments.

Scientific research aims:

Summarize the key theories of electronics, which are necessary in creation of line follower robot. Create line follower robot and describe the creation process. Write and test algorithms for line follower robot.

Tasks of our work:

1. Research theories, which are necessary in line follower robot creation.
2. Make a line follower robot for each of us.
3. Make the track, on which to carry out experiments with line follower robots.
4. Changing the robot's number of sensor and typing the appropriate algorithms, to test the operation of robots with two, three and five sensors.
5. Take all the algorithms and compare robots effectiveness.

Hypothesis of work:

Line follower robot that uses 5 sensors will be moving more stable and faster than line follower with 2 or 3 sensors.

Scientific research methods: Information studying and experiments.

The results of studies partly confirm the hypothesis, that a robot using 5 sensors will be moving faster and more stable than using 3 or 2 sensors.

Pielikumi

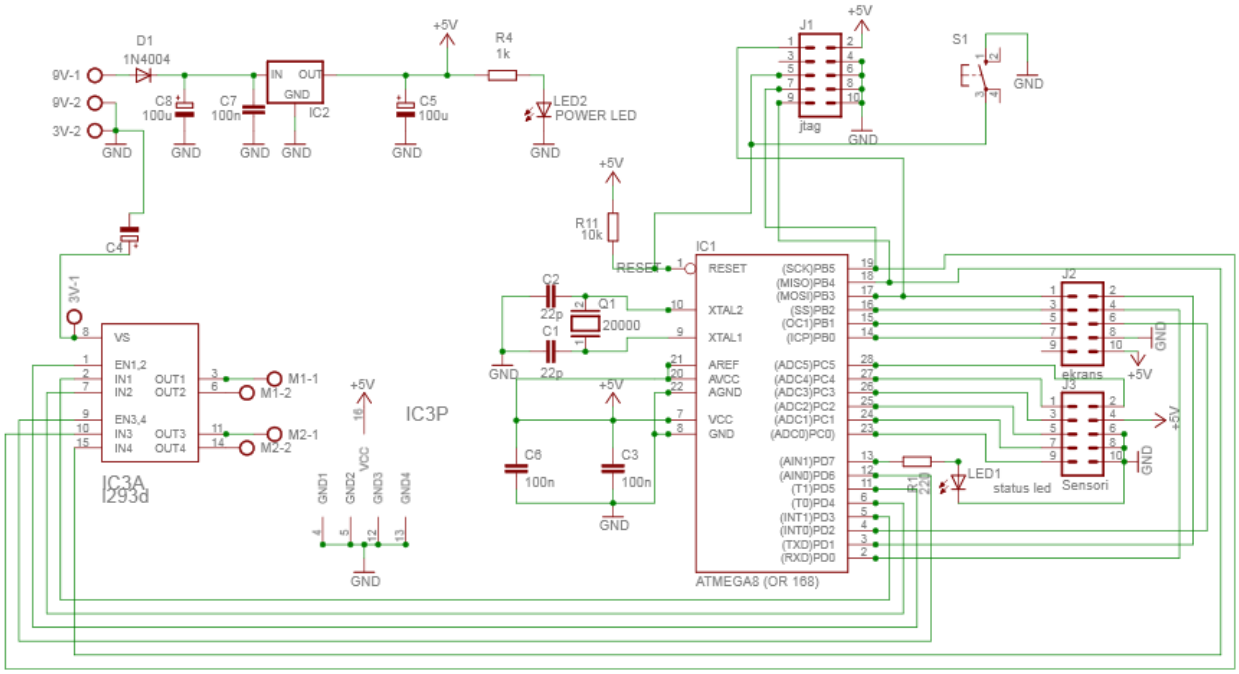
```
#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>


volatile unsigned char fourdisp[4];
volatile unsigned char numbers[10] = {0b00111111, 0b00000011,0b01101101,0b01100111,
0b01010011,0b01110110,0b01111110,0b00100011,0b01111111,0b01110111 };

volatile uint8_t column[4] = {1<<0, 1<<1, 1<<2, 1<<3};
volatile uint8_t seq,count;
volatile int ss,mm,hh;
volatile char buffer[16];
volatile uint8_t data,where,adcs;

const int SCK = 0;
const int RCK = 1;
const int SER = 2;

const int Skaits = 8;
const uint8_t jutigums[5] = {170,170,170,170,170}; //2. 235 5. 235

void write7_uni(char *exp)
{

   int sk,len;
   char temp;
   len = strlen ( exp );

   if(len==4)
   {
                temp = exp[3];
                sk = atoi( &temp );
                fourdisp[0] = numbers[sk];
                temp = exp[2];
                sk = atoi( &temp );
                fourdisp[1] = numbers[sk];
                temp = exp[1];
                sk = atoi( &temp );
                fourdisp[2] = numbers[sk];
                temp = exp[0];
                sk = atoi( &temp );
                fourdisp[3] = numbers[sk];
   }
   if(len == 3)
   {
                temp = exp[2];
                sk = atoi( &temp );
                fourdisp[0] = numbers[sk];
                temp = exp[1];
                sk = atoi( &temp );
                fourdisp[1] = numbers[sk];
                temp = exp[0];
                sk = atoi( &temp );
                fourdisp[2] = numbers[sk];
                fourdisp[3] = 0b00000000;
   }
   if(len==2)
   {
                temp = exp[1];
                sk = atoi( &temp );
                fourdisp[0] = numbers[sk];
                temp = exp[0];
                sk = atoi( &temp );
                fourdisp[1] = numbers[sk];
                fourdisp[2] = 0b00000000;
                fourdisp[3] = 0b00000000;
   }
   if(len == 1)
   {
                temp = exp[0];
                sk = atoi( &temp );
                fourdisp[0] = numbers[sk];
```

```
                fourdisp[1] = 0b00000000;;
                fourdisp[2] = 0b00000000;
                fourdisp[3] = 0b00000000;
    }
}

void write_595(uint8_t data)
{


    for(int i=0; i<Skaits; i++)
    {

                if((data & (1<<i)) )
                {
                            PORTD |= (1<<SER);
                }
                else
                {
                            PORTD &= ~(1<<SER);
                }
                PORTD &= ~(1 << SCK);
                PORTD |= (1<<SCK);

    }

 PORTD |= (1 << RCK);


    PORTD &= ~(1 << RCK);
    PORTD &= ~(1 << SCK);
    PORTD &= ~(1 << SER);
}


uint16_t ReadADC(uint8_t ch)
{
   ADMUX &= 0xF8;
    ADMUX |= ch;

   ADCSRA|=(1<<ADSC);

   while(!(ADCSRA & (1<<ADIF)));
   ADCSRA|=(1<<ADIF);

   return(ADCH);
}


void sensors(uint8_t adc_data, uint8_t ch)
{
    if(adc_data >= jutigums[ch])
    {
                data |= (1<<ch);
    }
    else
    {
                data &= ~(1<<ch);
    }


}

void goAhead()
{
    OCR0A = 225; //190 200
    OCR0B = 225;
}
void turn_left()
{

    OCR0A = 225;
    OCR0B = 150;
}
void turn_right()
{

    OCR0A = 150; //3. 140
```

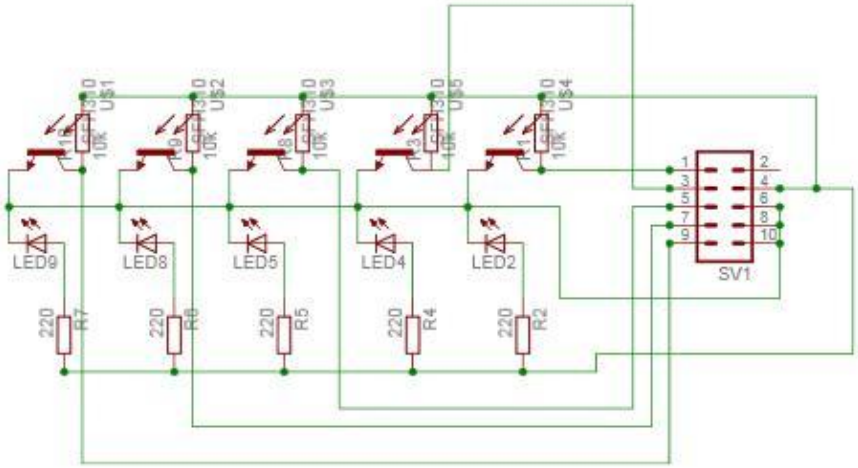
```
    OCR0B = 225; //3. 190
}

void turn_left_middle()
{
    OCR0A = 225;
    OCR0B = 150;
}

void turn_right_middle()
{
    OCR0B = 150;
    OCR0A = 225;
}

void turn_left_fast()
{
    OCR0A = 195; //3. 180
    OCR0B = 0;
}
void turn_right_fast()
{
    OCR0A = 0;
    OCR0B = 195;
}
void getSignals()
{
                sensors(ReadADC(0),0);
                sensors(ReadADC(1),1);
                sensors(ReadADC(2),2);
                sensors(ReadADC(3),3);
                sensors(ReadADC(4),4);

                adcs = ReadADC(1);
                sprintf(buffer,"%d", adcs);
                write7_uni(buffer);
}

int main()
{
    seq = 0;
    TCCR1B |= (1 << CS01) | (1 << WGM12);

     TIMSK1 |= ((1<<OCIE1A));

    OCR1A = 500;

    PORTD |= column[0];

    TCCR0A |= (1 << WGM01) | (1 << WGM00) | (1<<COM0A1) | (1<<COM0B1);
    TCCR0B |= (1<<CS00);
    TCNT0 = 0;

    ADCSRA |= (1 << ADPS1) | (1 << ADPS0);

    ADMUX |= (1 << REFS0);
    ADMUX |= (1 << ADLAR);
    ADCSRA |= (1 << ADEN);  // Enable ADC
    ADCSRA |= (1 << ADSC);  // Start A2D Conversions
    count = 0;
    ss=0;
    mm=0;
    hh=0;
    sei();

    DDRB = 0b11111111;
    DDRD = 0b11111111;

    PORTD |= (1<<3);
    PORTD &=~(1<<4);
    PORTB |= (1<<5);
    PORTB &=~(1<<4);
    while(1)
    {

                getSignals();
```

```
		//5 sensori
		if(data == 0b00000001 || data == 0b00000011)
		{
			turn_right_fast();
			where = 0x00;
			where |= (1<<0);
		}
		if(data == 0b00010000 || data == 0b00011000)
		{
			turn_left_fast();
			where = 0x00;
			where |= (1<<1);
		}
		if(data == 0b00001000) turn_left();
		if(data == 0b00000010) turn_right();
		if(data == 0b00000111)
		{
			turn_right_fast();
			_delay_ms(100);
		}
		if(data == 0b00011100)
		{
			turn_left_fast();
			_delay_ms(100);
		}
		if(data==0b000000100)
		{
			goAhead();
		}
		if(data==0b000001100) turn_left();
		if(data==0b000000110) turn_right();
		if(data==0b000011111) goAhead();


		if(data == 0b00000010) PORTD |= (1<<7);
		else PORTD &=~(1<<7);

	}

}

ISR(TIMER1_COMPA_vect)
{

				if(seq+1 < 4) { seq++;
				PORTB &= ~(column[seq-1]); }
				else { seq = 0;
				PORTB &= ~(column[3]);}

					write_595(fourdisp[seq]);
					PORTB |= column[seq];

}
```

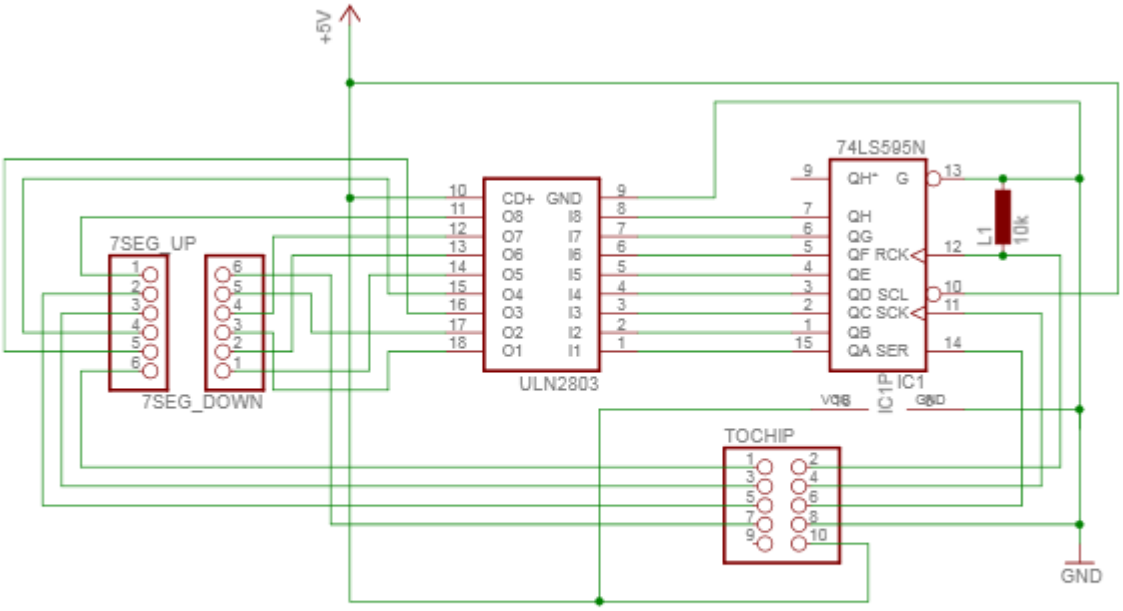
Robota pirmkods, kur tiek izmantoti 5 sensori.



Galvenās plates elektroniskā shēma.

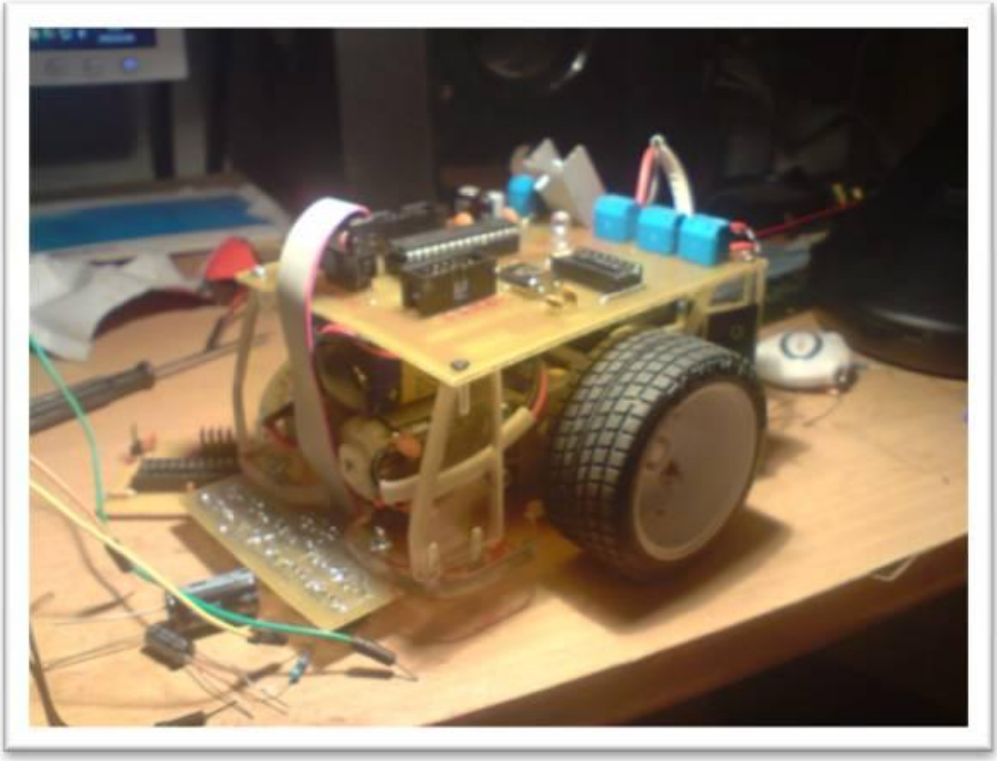


Sensoru plates elektroniskā shēma.

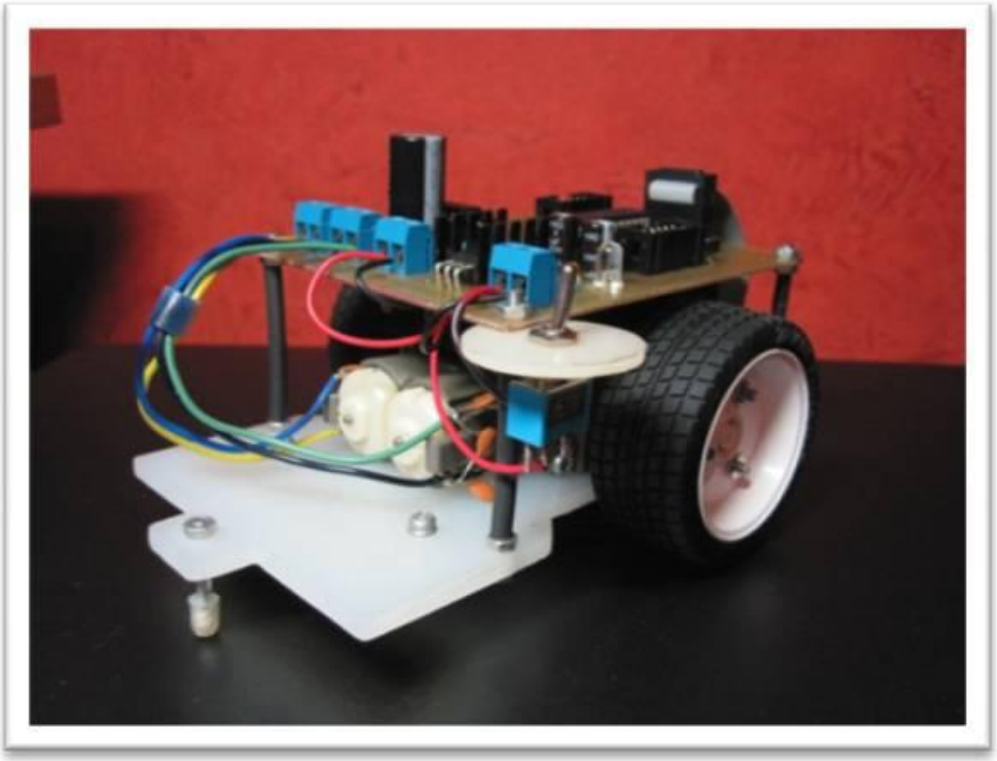


7 segmentu displeja elektroniskā shēma.

5. pielikums

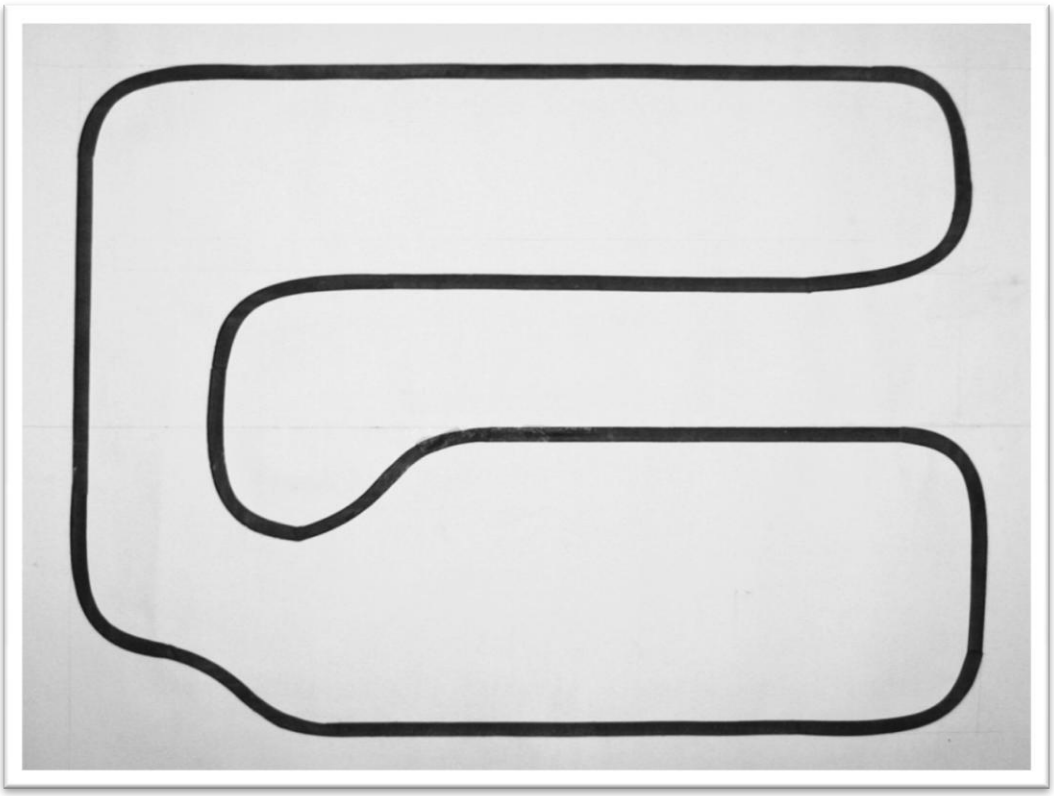


Raivja robots „Gutsy”.

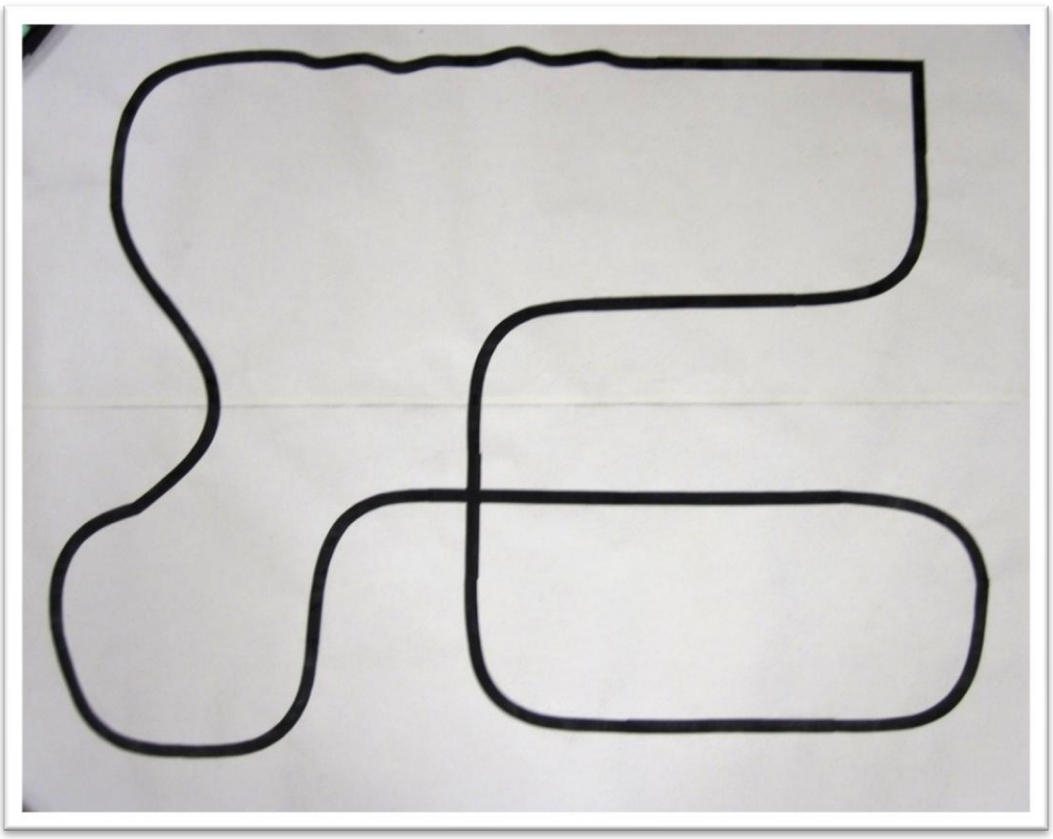


Toma robots „Jaunty”.

6. pielikums

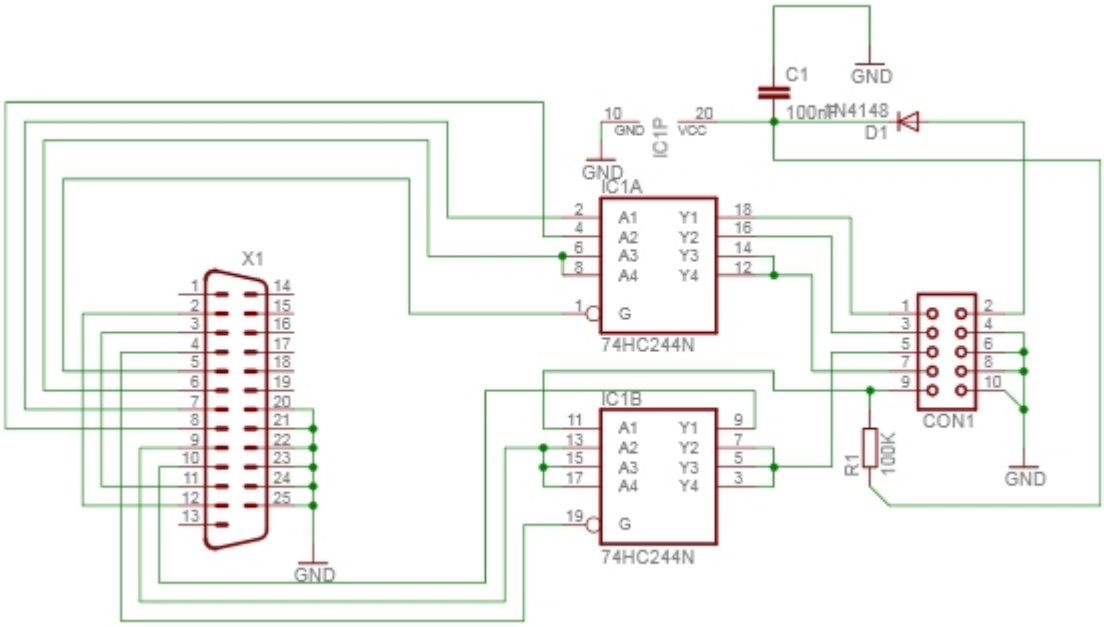


1. trase.



2. trase.

7. pielikums



Programmatora elektriskā shēma.

8. pielikums