Tallinna Tehnikaülikool

Teeme Ise 2011

ARUANNE

Toomas Ormisson

Mida õppisin:

- **Manuaalilt vajalike tehniliste andmete lugemist**
  Kuna riistvara programmeerimisel tuleb hästi kursis olla programmeeritava seadmega ja kuidas selle poole täpselt pöörduda, on mikrokontrolleri manuaal kõige olulisem tööriist seadme programmeerimisel.

- **Nende rakendamist riistvara programmeerimisel**
  Tehnilistest andmetest kogutud info põhjal saab teada milleks antud seade võimeline on ning mis siine pidi ja mis infot tuleb edastada, et mikrokontroller viiks läbi soovitud operatsioone.

- **Bittidega manipuleerimist C keeles**
  Kuigi binaartehtetega on kokku puututud juba diskreetses matemaatikas ja arvutid I tunnis, siis nende rakendamine programmeerimisel oli esialgu veidi võõras.

PID *(proportional–integral–derivative)* kontroll

PID on tagasisidestatud kontrollimismehhanism mis arvutab sensoritelt saadud info põhjal viga ja üritab seda jooksvalt parandada. Joonejärgimisroboti puhul määrab ta oma asukoha sensorite suhtes ja üritab püsida joone keskel.

00001	4	Robot asub joonest vasakul
00011	3	
00010	2	
00110	1	
00100	0	Robot asub täpselt joone keskel
01100	-1	
01000	-2	
11000	-3	
10000	-4	Robot asub joonest paremal
00000	-5 || +5	oleneb eelnevatest väärtustest

Roboti sensorid aitavad tal näha sihtpositsiooni ja tegelikku positsiooni ning määrata nende vahet ehk viga. Saadud vahe on proportsionaalne (P) joone suhtes. Veast võetud integraal (I) tähistab vea kuhjumist aja jooksul ning tuletis (D) näitab kui palju robot võbeleb küljelt küljele, mida kõrgem on tuletise väärtus seda ebastabiilsemalt robot sõidab. Koodis kasutatavad suurused Kp, Ki ja Kd on konstandid mis mõjutavad vastavalt P, I ja D väärtusi.

PID kontroll ei garanteeri joone optimaalset järgimist, ehk siis PID-i kasutamisel on vea teke paratamatu. Seepärast on oluline siluda Kp, Ki, ja Kd väärtusi, et minimeerida sõitmisel tekkivat viga. PID-i efektiivsust piiravad veel sensorite arv ja protsessori võimekus. PID-i seadistamine käib küll katse-eksituse meetodil kuid mõistlik oleks lähtuda kindlast seaduspärast:

1) Seada Kp=Ki=Kd=0 ning alustada kõigepealt Kp tuunimist. Selleks tuleks seada Kp esialgseks väärtuseks üks ning vähendada selle väärtust seni kuni robot suudab enamvähem joonel

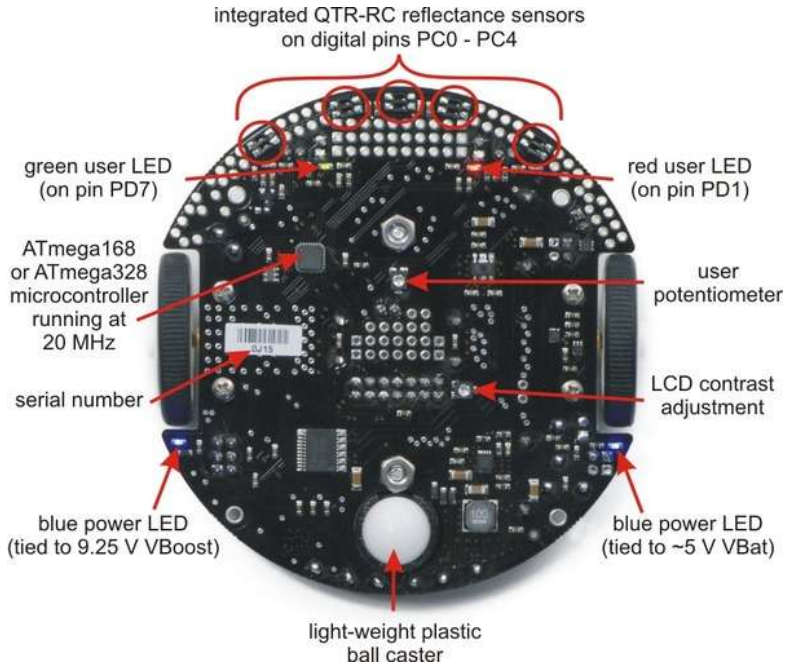
püsida. Kui robot sõigab joonest üle, tuleb Kp väärtust vähendada. Kui robot ei suuda nurkades keerata tuleb Kp väärtust tõsta.

2) Kui robot suudab enamvähem mööda joont sõita võib Kd panna võrduma ühega ning sammhaaval väärtusi tõsta kuni robot sujuvamalt sõidab.
3) Kui robot suudab juba suht sujuvalt sõita võib määrata Ki väärtuse, esialgu kuskil 0.5 ja 1 vahel. Kui väärtus on liiga kõrge liigub robot äkiliselt küljelt küljele, kui liiga madal, siis ei tohiks see eriti midagi muuta.

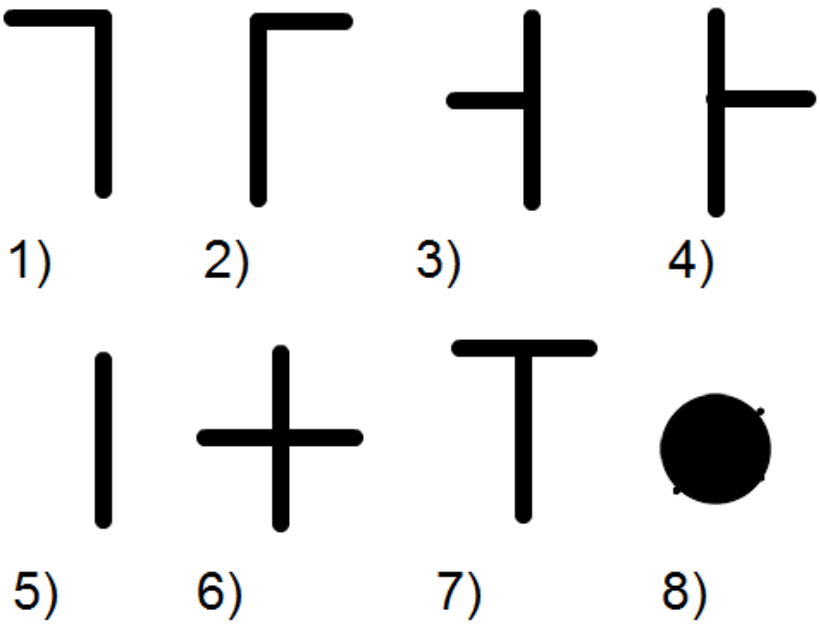
Oluline on meeles pidada, et ka roboti kiirus mõjutab PID-i efektiivsust. Kui PID-i on näiteks aeglase kiiruse juures seadistatud, siis suurematel kiirustel võib ta ettearvamatult käituda.

Algoritm:

Läbimaks edukalt labürinti saame kasutada kas vasaku- või paremakäereeglit. Antud juhul kasutab robot läbimiseks vasakukäereeglit ning robot eelistab alati vasakpööret alternatiividele. Kui robotil on võimalus valida kas minna otse või paremale eelistab robot minna otse.
Joone jälgimiseks kasutab robot viit sensorit, mis on allpool toodud joonisel ära märgitud PC0-PC4.



Labürinti läbides võib meil ette tulla kaheksa erijuhtumit kus robot peab otsustama mida edasi teha.



1) Robotil puudub valikuvõimalus ja ta saab liikuda vaid vasakule
2) Robotil puudub valikuvõimalus ja ta saab liikuda vaid paremale
3) Robot saab valida kas minna otse või teha vasakpööre
4) Robot saab valida kas minna otse või teha parempööre
5) Robot satub tupikusse ja saab liikuda tuldud teed pidi tagasi
6) Robot satub ristmikule ja saab minna kas otse, vasakule või paremale
7) Robot saab valida kas minna paremale või vasakule
8) Robot pääseb labürindist välja – jääb seisma.

Kuna roboti sensorid ei pruugi eristada esimest ja teist erijuhtumit vastavalt kolmandast ja neljandast, siis see tuleb lahendada funktsiooniga, mis laseb robotil paar sentimeetrit edasi sõita, et robot oleks võimeline otsustama, kas kasutada PID-i või erijuhtumite lahendamiseks mõeldud funktsiooni.

Roboti joone jälgimine on lahendatud PID-iga ning ristmikule jõudes jääb robot hetkeks seisma ja täidab funktsiooni, mis on mõeldud ristmikutel otsuste tegemiseks.
Antud funktsiooni juurde minnakse kui:

1) Üks äärmistest joonejälgimise sensoritest näeb joont
2) Mõlemad äärmised sensorid näevad joont
3) Robot ei näe enam joont

Ristmikute lahendamise funktsioon peaks välja nägema umbes selline:

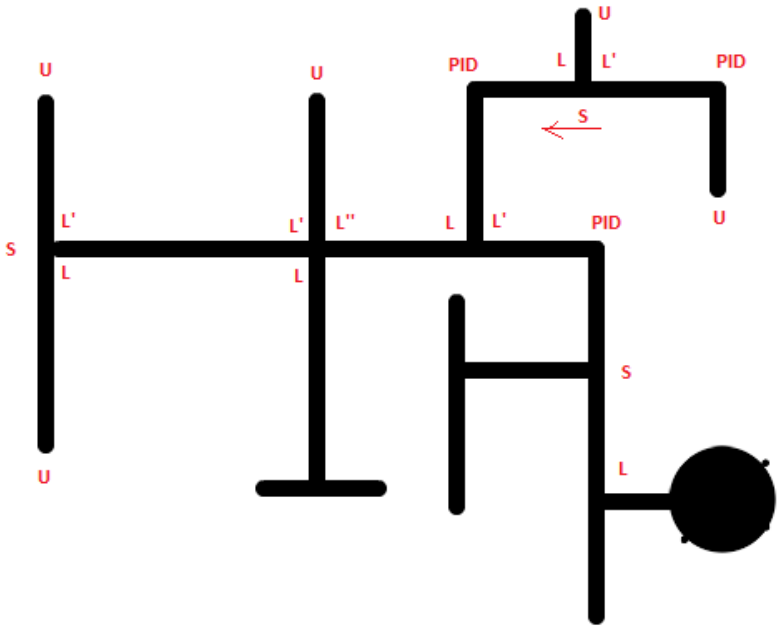
```
char select_turn(unsigned char found_left, unsigned char found_straight, unsigned char found_right){
/* found_left, found_straight jne on roboti koodis ära defineeritud selle järgi mida roboti sensorid parasjagu näevad  */
                if(found_left)
                        return 'L';
                else if(found_straight)
                        return 'S';
                else if(found_right)
                        return 'R';
                else
                        return 'U';
/*    Tähtedelt tagastatavad väärtused on defineeritud koodis mootorite pöörlemisena nt. „ found_left” puhul puhul käib
parem mootor mingi kiirusega X pärisuunas ning vasak mootor kiirusega X vastassuunas    */

        }
```

Jõudes funktsiooni lõppu salvestab robot *select_turn* funktsioonist saadud tähe mälumassiivi *path[]* ja lisab muutujale *path_length* ühe suurusjärgu juurde (*path_length += 1*). Seejärel vaatab robot, kas ta saab oma teekonda optimeerida, selle jaoks on eraldi funktsioon, mis näeb välja selline:

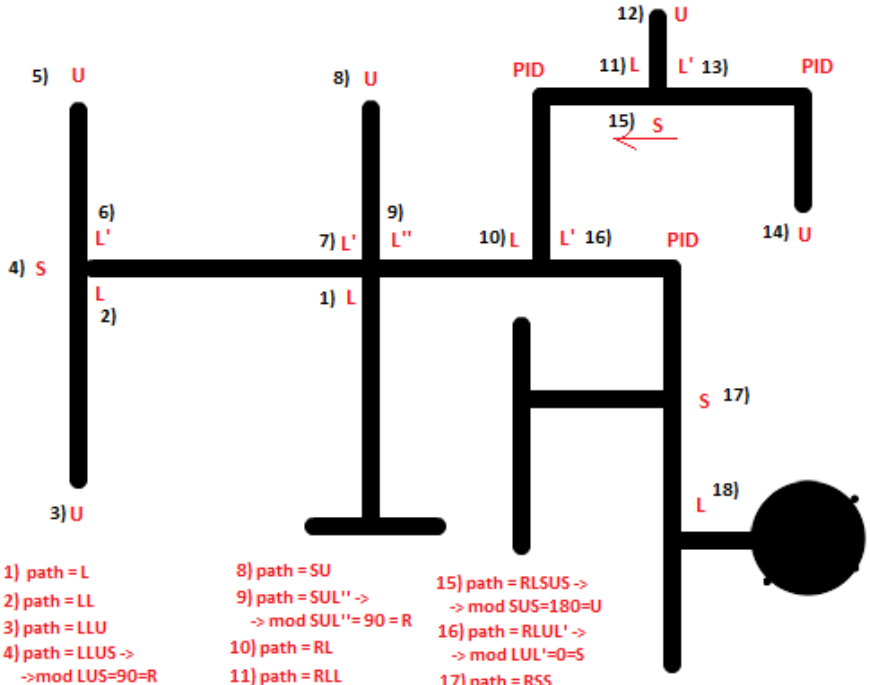
```
void simplify_path(){
/* Kui robot on näinud vähem kui kolme ristmiku või roboti mälu massiivis ei ole kirjas ühtegi ümberpööret millele järgneks veel üks pööre jätkab robot antud funktsiooni vahele. */
		if(path_length < 3 || path[path_length-1] != 'U')
			return;
		int total_angle = 0;
		int i;
		for(i=1;i<=3;i++){
/* Funktsioon teisendab roboti mälus olevad kolm järjestikust käiku kraadidesse ja salvestab nende summa muutujasse total_angle */
			switch(path[path_length-i]){
				case 'S'		total_angle += 0;		break;
				case 'R':		total_angle += 90;		break;
				case 'L':		total_angle += 270;		break;
				case 'U':		total_angle += 180;		break;
			}
		}
/* Funktsioon  võtab muutuja total_angle mooduli, mis vasakukäereeglit järgides peaks võrduma kas 0, 90 või 180 - kui moodul = 0, siis robot oleks pidanud 3 käiku tagasi vasaku pöörde asemel otse minema, kui mod=90, siis oleks pidanud paremale pöörama. mod = 180 tähistab U-pööret, ent see on vaid vahesamm õige raja leidmisel. */
		total_angle = total_angle % 360;

		switch(total_angle){
			case 0:		path[path_length - 3] = 'S';		break;
			case 90:	path[path_length - 3] = 'R';		break;
			case 180:	path[path_length - 3] = 'U';		break;
			case 270:	path[path_length - 3] = 'L';		break;


	}
/* kuna antud funktsiooni täidetakse vaid siis kui robot on teinud vale pöörde ja ülal on see vale pööre ära parandatud, on vaja kaks ülearust käiku roboti mälust kõrvaldada  */
	path_length -= 2;
}
```

Algoritmi visualiseerimiseks on allpool toodud paar joonist.

Rakendame labürindi peal kõigepealt vasakukäereeglit. Nii peaks robot labürindi esimesel katsel läbima:



Labürindi läbimise käigus optimeerib robot oma teekonda automaatselt simplify_path() funktsiooniga ning eemaldab valed käigud mälust:



Nagu näha on lõplik lihtsustatud teekond mille robot tulemuseks saab *path = RSSL*. Mis tähendab, et kõige optimaalsem teekond labürindi läbimiseks on:

1) Esimesel ristmikul pööre paremale
2) Teisest ristmikust otse üle sõita
3) Kolmandast ristmikust otse üle sõita
4) Neljandal ristmikul vasakpööre