Mikrokontrollerid ja praktiline robootika
Aruanne

Koostas: Riho Külaots
Maria Vita
Sigrid Münter
Jaanus Karjane
Juhendaja:

2009
Tallinn

1. Sisukord

2. Ülesanne

Kaks võstkonda võistlevad korraga sumoringis oma konstrueeritud iRobot Roomba Vacuum Cleaneri sumorobotitega. Üks matš kestab 3 minutit. Matši võidab meeskond, kelle robot suudab vastase kahel korral kolmest ringist välja lükata või saab 2 „Yuhkoh-“ punkti. Võistlusringi juures võib olla korraga mõlemast võistkonnast vaid üks meeskonnaliige, kes sätivad robotit stardiks kõrvuti ja käivitavad nad. Stardis peavad robotid peale käivitamist paigal püsima 5 sekundit.
Robotil peavad kõik mõõtmed peale kõrguse säilima, roboti kaal ei tohi erineda originaalroboti kaalust rohkem kui 30% ja maksimumkaal on 3,8 kg. Energiaallikana võib kasutada ainult originaalakut.

3. Ideed

a. „Nägemine“

Roomba sumoroboti ehitamisel oli vastavalt ülesande püsitusele eesmärgiks, et robot suudaks teise endasuguse lokaliseerida nind teda rünnata. Vastase leidmiseks on tarvis anda robotile silmad, milleks otsustasime kasutada Sharp infrapunasensoreid (IR sensor) põhiliselt, kuna neid meil jagus ja nad sobisid kokku kontrollerplaadiga. Paigaldasime sensori roombale esimese küljele keskele. Peagi jõudsime arusaamisele, et ühest sensorist jääb väheks, kuna teist robotit rünnates on lükkamise jõud suurem teda otse rünnates, nurga lükkates võib ründaja teisest lihtsalt mööda libiseda või teine eest lihtsalt ära sõita. Seega paigutasime Roobma esimesel küljele 2 sensorit, et oleks võimalik korrigeerida sõidusuuna otse teise roboti suunas. Kahe sensori paigaldamisel oli plussiks veel, et vastast on võimalik kiiremini jälitada: kui vastane sõidab roboti eest läbi, siis üks sensor saab lihtsalt aru, et tema eest sõideti läbi ning peab hakkama suvalises suunas sõitma, et vastane leida. Seevastu kahe sensoriga saab on võimalik tuvastada, millises suunas vastane sõitis. Vastavalt, kumba sensoriga vastast viimati nähti, selle suunas hakatakse sõitma, et jõuda vastaseni võimalikult kiiresti. Lisaks kahele sensorile esiküljel lisasime IR sensori Roomba tagumisele küljele, et veel suurendada võimalust vastast kiiresti märgata. Tagumise sensoriga vaenlast märgates, pean robot end ümber keerama ning esimeste sensoritega end vaenlase poole juhtima.

b. Kaalueelis

Mõningased teadmisest füüsikast ja peamiselt testvõitlused teiste robotitega tegid selgeks, et suurema massiga robot suudab kergema välja lükata. Seega paigutasime roboti sisemusse rauatüki, millega viisime kaalu maksimumi lähedale ehk ligemale 3,8 kg. Lisaraskusega tekkis aga ka probleem. Nimelt asetsevad originaalis Roobma rattad külgedel ning ees, millest tingituna läks kaalujaotus paigast. Lisaraskuseta ei suutnud robot üle ääre sõites end tagasi väljakule tagurdada. Lisaraskus tuli sellest tulenevalt paigutada tagumisele otsale, kuid siis tekkis probleem robotil ründele asudes. Ründama hakkas robot suurema kiirusega kui otsimisrežiimil ning kiirendades tõusis roboti esiosa õhku ja ta „vaatas taevasse“ ja vastasel võis õnnestuda roboti vaateväljast kaduda. Lahenduseks paigaldasime roboti tagumise külje alla lisaratta, mis ei

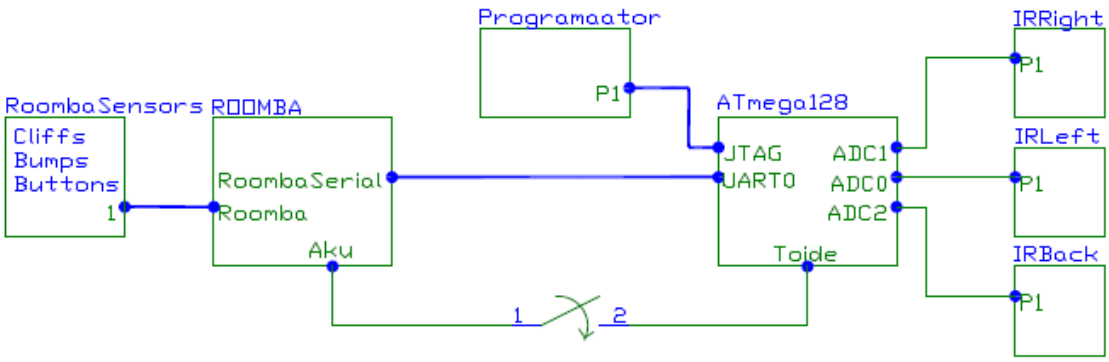
lasknud tagumisel ostsal vastu maad vajuda. Kuid võistluseelsel õhtul selgus katsetamise käigus, et lisakaal omab kõige suuremat effektiivsust roomba teljel, ning seetõttu viisime raskuse roomba tagaosast roomba teljele. See parandas roboti võimet teisi lükata, ning kadus vajadus ka tagumisele rattale, mis võistluspäeval saigi eemaldatud, et vastasrobotite sahkade poolt ülestõstetud roomba ei jääks tagumisele rattale kandma.

4. Spetsifikatsioon

a. Roboti kirjeldus

Roboti valmistasime IRobot Roomba ® 416 baasil, mis on 8,89 cm kõrge ja mille diameeter on 33 cm. Algselt kaalus roomba 2,9 kg. Võistlusel oli lubatud roomba originaalkaalu muuta ± 30%. Maksimaalseks kaaluks seati 3,8 kg. Meie andsimegi oma robotile maksimaalse võimaliku kaalu, sest korduvate katsetuste käigus osutus, et suurema massiga robotitel on eelis väiksema massiga omade ees. Massi suurendasime metallplaadiga, mille panime roboti alumisse osasse keskele. Robotilt eemaldasime tolmukogumiskasti. Paigaldasime robotile ette 2 ja taha ühe IR anduri. Need olid mõeldud vastase nägemiseks. Esialgu olid meie robotil andurid vaid ees, kui hiljem lisasime ka taha ühe, kuna nii ei pidanud ta tegema täispööret, et kõike enda ümber toimuvat jälgida. Eesolevate andurite paigaldamiseks eemaldasime tüki roboti esipaneelist. Robotil olid algselt andurid ka bumperi küljes ja allosas. Paigutasime robotile taha ühe väikese metallist ratta juurde, et tagada paremat stabiilsust. Originaalsuse huvides värvisime roboti küünelakiga roosaks.

b. Elektroonika üldskeem



c. Algoritmi selgitus

i. Ülesanne

Roboti algoritm põhines eesmärgil vastane võistlusringist välja lükata ning ise ringi sisse jääda.

ii. Algoritmi töökäik

Roboti käivitamise järgselt luuakse kõigepealt ühendus Roomba ROI(Roomba Open Interface) ja robotil kasutatava ATMega128 plaadi vahel. Ehk siis viiakse Roomba sellisesse olekusse(Full Mode), kus ta võtab liikumiskäske vastu läbi ROI.

Algoritm on jagatud nelja osasse: Esimeses osas juhitakse roomba ründamist, teises osas on kirjeldatud roboti lähivõitluse algoritm, kolmandas osas on roomba algoritm väljakult väljasõtmise vastu ning neljas osa on roomba vastase otsimise algoritm.

Võistlusalgoritmi käivitudes oodatakse kõigepealt startkäsku - HandleLocalCommands(). Startkäsu saades oodatakse reeglitekohaselt 5 sekundit ning seejärel alles alustatakse peaalgoritmiga.
Iga tüskli alguses uuendatakse kõigi kasutuses olevate sensorite andmeid - UpdateSenosrdata(). Seejärel tegutseb roomba vastavalt senosrite infole.
Kõige suurema prioriteediga tegevuseks on võistlusväljakult väljasõidu takistamine. Kui roomba juhutb sellisesse olukorda sattuma, siis keelatakse ründamine ja vastase otsimine ning üritatakse ohutusse alasse jõuda. Vastavad lipud on: SafeToSearch = false; SafeToAttack = false.
Järgmiseks prioriteediks võib lugeda Bumpereid. Juhul kui bumperi sensorid näitavad, et roomba on kellegi vastus, siis keelatakse vastase otsimine IR senorite kaudu - SafeToReadFrontIR = false. Niikaua, kuni kõrgema prioriteediga tegeust ei esine(väljakult väljasõit), lükatakse vastast.
Järgmiseks madalamaks prioriteediks on vastase ründamine kasutades IR andureid. Kui vastast nähakse pikema distatsi pealt, siis liigutakse vastase poole, vajadusel korrigeerides suunda. Ning IR’e kasutatakse seni, kuni ei esine kõrgema prioriteediga tegevust(väljakult väljasõit, bumper).
Viimaseks, kõige madalama prioriteediga tegevuseks, on vastase otsimine IR anduritega. Juhul, kui roomba on ohutus alas ning ei ole bumperiga vastasel vastas ja ei ole vastast näinud teatud aja (LASTSEENTIMER) siis hakatakse vastast otsima – ehk antud algoritmi korral kohapeal keerutama - aegajalt suunda muutes. Ning seda tehakse seni, kuni ei esine mõni kõrgema prioriteediga tegevus.

Kogu eelpool kirjeldatud tegevus toimub tsükliliselt seni, kuni saadakse Stopkäsk – ehk läbi ROI saadetakse roombale roomba.PowerOff() käsk.

Täpsem ja põhjalikum kirjeldus algoritmist on kirjas lähtekoodis.

iii. Algoritmi puudused

Algoritmi suurimaks puuduseks võib pidada vastase otsimist(kohapeal keerutamine) ning joonelt taganemist. Nimetatud puudused andsid tunda võistluspäeval, kui selgus, et joonelt taganemine peaks olema veidi kavalamalt lahendatud.

d. Programmikoodi tähtsamad osad

Kirjeldatud on mõningad osad programmist. Täpsema kirjelduse leiab lähtekoodist.

i. Sensorite info

Üheks tähtsamaks osaks võib lugeda sensorite info kogumist:

```
void Program::UpdateSenosrdata()
{
        // using only subsets 1 and 2
        roomba.ReadSensors(SUBSET_1);
        roomba.ReadSensors(SUBSET_2);

        IRLeft =  irLeft.GetDistanceInCM(5);
        IRRight = irRight.GetDistanceInCM(5);
        IRBack =  irBack.GetDistanceInCM(5);

        BumpRight = roomba.Sensors.BumpRight;
        BumpLeft  = roomba.Sensors.BumpLeft;

        CliffLeft= roomba.Sensors.CliffLeft;
        CliffFrontLeft = roomba.Sensors.CliffFrontLeft;
        CliffFrontRight= roomba.Sensors.CliffFrontRight;
        CliffRight = roomba.Sensors.CliffRight;
        VirtualWall = roomba.Sensors.VirtualWall;

        MaxIsPressed   = roomba.Sensors.Max;
        CleanIsPressed = roomba.Sensors.Clean;
        SpotIsPressed  = roomba.Sensors.Spot;
        PowerIsPressed = roomba.Sensors.Power;
}
```

ii. Ründamine

Vastase ründamine kasutades IR sensoreid(näites on kood kõvasti lihtsustatud)

```
// Attack, if it is safe
if(SafeToAttack && SafeToReadFrontIR) {

// seeing enemy?
if(IRLeft < ENEMY_IN_FRONT && IRRight < ENEMY_IN_FRONT) {
        roomba.Drive(FORWARDSPEEED, STRAIGHT);}

else if(IRLeft < ENEMY_IN_FRONT && IRRight > ENEMY_IN_FRONT) {
        roomba.Drive(FORWARDSPEEED, CCW + 3*SLIGHT_TURN);}

else if(IRLeft > ENEMY_IN_FRONT && IRRight < ENEMY_IN_FRONT) {
        roomba.Drive(FORWARDSPEEED, CW - 3*SLIGHT_TURN); }

else if(IRBack  < ENEMY_IN_BACK  &&
```

```
        IRLeft  > ENEMY_IN_FRONT &&
        IRRight > ENEMY_IN_FRONT &&
        SafeToReadBackIR)
{
        /* kood*/
}
```

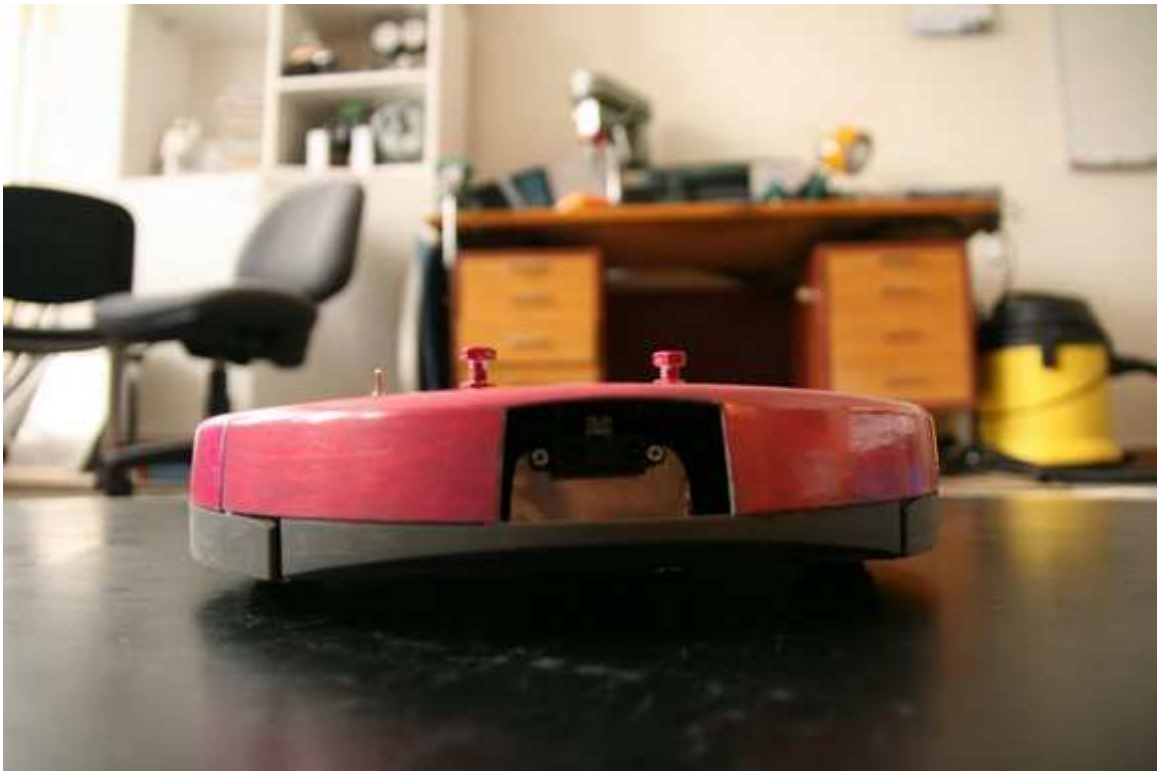
iii. Joonelt\äärelt taganemine

Väljaku ääre pealt taganemine. (Kood on kõvasti lihtsustatud)

```
if( (CliffLeft || CliffFrontLeft || CliffFrontRight || CliffRight) && IRBack <
ENEMY_IN_BACK) {
        // drive backwards until enemy is behind
}
//over border?
else if(CliffLeft && CliffFrontLeft && CliffFrontRight && CliffRight) {
        roomba.Drive(BACKWARDDSPEEED, STRAIGHT);}

else if(CliffFrontLeft && CliffFrontRight) {
        roomba.Drive(BACKWARDDSPEEED, STRAIGHT); }

else if(CliffLeft){
        roomba.Drive(BACKWARDDSPEEED, -BORDERANGLE); }

else if(CliffFrontLeft){
        roomba.Drive(BACKWARDDSPEEED, STRAIGHT);}

else if(CliffFrontRight){
        roomba.Drive(BACKWARDDSPEEED, STRAIGHT);}

else if(CliffRight){
        roomba.Drive(BACKWARDDSPEEED, BORDERANGLE);}
else {
// Not over border.
}
```

e. Fotod

Pink End. Eest on näha avad, mille lõikasime sensorite paigaldamiseks. Mutrid keskel hoiavad üleval terasplaati, mille asetasime raskuskeskmesse, et robot suudaks oma massiga teisi roboteid platsilt efektiivsemalt välja lükata.



Roboti tagaküljelt on näha sensor, mille paigaldasime, et robot suudaks kiiremini reageerida, kui vastast peaks näha olema või kui ta peaks lähenema selja tagant.

5. Kokkuvõte, võistlustulemused, järeldused

Võistlustel jagati robotid loosiga paaridesse. 2 robotit võistlesid omavahe 3 korda järjest, iga kord niikaua kuni 1 robot teise ringist välja ajas või maksimaalselt 3 min. Kokkuvõttes pääses meie robot finaali, kuid jäi neljandaks, sest finaalis kaotas ta 1:2. Päris palju tuli võistluse ajal ette hetki, kus 2 robotit jäid vastamisi üksteist nügima, ilma paigalt liikumata, sest nende massid ja tegutsemistaktika olid samad. Sel juhul sai tihti saatuslikuks aeg. Mitmed Leedu võistkonnad olid oma robotitel tagumise osa sahaks disaininud ja see töötas päris hästi nende kasuks. Seetõttu enne finaali eemaldasime oma robotil tagumise metallist ratta, et tagada mõningast kaitset sahkade vastu. Kahjuks sellest väga palju kasu ei olnud, sest finaalis pidi ta võistlema ka ilma sahata robotiga, mis osutus meie omast paremaks.
Kokkuvõttes võib öelda, et antud kursus avardas tunduvalt meie silmaringi robotite osas ja äratas ka suuremat huvi asjaga edasi tegeleda. Antud programmi raames tehtud robotit võiks kindlasti edasi töödelda nii mehaanika kui ka tarkvara osas.

6. Kasutatud kirjandus

Kasutud materjalid või viited materjalidele asuvad:

http://www.robotiklubi.ee/kursused/roomba_sumo

Lisa 1.

Põhiprogrammi kood

```
/*****************************************************************************\

        Sumo Roomba
        Baltic Robot Sumo Cup 2009

        Jaanus Karjane
        TUT Robotics Club
        Spring 2009
        Tallinn

\*****************************************************************************/

#include "common/Main.h"

#define object_modifier extern
#include "common/HardwareObjects.h"
#undef object_modifier

// speed values
#define FORWARDSPEEED           500
#define BACKWARDDSPEEED -500
#define BUMPERSPEED     500

// turning values
#define BORDERANGLE     225
#define SLIGHT_TURN             100

#define ON                      1
#define OFF                     0

// IR values
#define ENEMY_IN_FRONT    60            //  50+ is OK
#define ENEMY_IN_BACK     35       // <20 is quite shortranged

// timers
#define LASTSEENTIMER   2000
#define SEARCHINGTIMER  2000
#define BACKTIMER       600
#define ONCLIFFTIMER    1200

#define START_DELAY     5000

// enemy directions
#define INBACK                  5
```

```
#define UNKNOWN          4
#define INFRONT                  3
#define INLEFT           2
#define INRIGHT           1
#define NOTSEEN           0

// turning directions
#define LEFT             0
#define RIGHT            1
#define NOTTURNING       2

// led debuging
//#define LEDDEBUG



/**
 * Robot main program.
 */
void Program::Run()
{
        roomba.Init();
        Reset();

        //roomba.PlaySong();

        // testing cycle
        while(false) {test();}

        while (true)
        {
                // put roomba to Full Mode, if not already in it. - avoids roomba from stoping
when lifted up.
                roomba.FullMode();

                // onboard buttons?
                HandleLocalCommands();

                // fight!
                if(FightingAllowed)
                {
                        /*
                         * update sensors data: IRs, bumpers, cliffs
                         */

                        UpdateSenosrdata();


        /**********************************************************************
***********/
```

```
        /*****************************************************************************
***********/

                        /*
                         * Attack Mode with IR's
                         */

                        // Attack, if it is safe
                        if(SafeToAttack && SafeToReadFrontIR)
                        {
                                // If roomba saw an enemy 'BACKTIMER'ms ago, allow to read
back ir again
                                if(IRBack > ENEMY_IN_BACK && clock.GetTimestamp() >
BackTimer + BACKTIMER) SafeToReadBackIR = true;

                                // seeing enemy?
                                if(IRLeft < ENEMY_IN_FRONT && IRRight <
ENEMY_IN_FRONT)
                                {
                                        LastSeen = INFRONT;

                                        roomba.Drive(FORWARDSPEEED, STRAIGHT);
                                        TurningDirection = NOTTURNING;

                                        SafeToSearch = false;
                                        SafeToReadBackIR = true;

                                        // When was the opponent last seen?
                                        LastSeenTimer = clock.GetTimestamp();

                                        #ifdef LEDDEBUG
                                                roomba.LED.Spot(ON);
                                                roomba.LED.Max(ON);
                                                roomba.LED.DirtDetect(OFF);
                                        #endif
                                }
                                else if(IRLeft < ENEMY_IN_FRONT && IRRight >
ENEMY_IN_FRONT)
                                {
                                        LastSeen = INLEFT;

                                        roomba.Drive(FORWARDSPEEED, CCW +
3*SLIGHT_TURN);
                                        TurningDirection = LEFT;

                                        SafeToSearch = false;
                                        SafeToReadBackIR = true;

                                        // When was the opponent last seen?
```

```
                LastSeenTimer = clock.GetTimestamp();

                #ifdef LEDDEBUG
                        roomba.LED.Spot(ON);
                        roomba.LED.Max(OFF);
                        roomba.LED.DirtDetect(OFF);
                #endif
        }
        else if(IRLeft > ENEMY_IN_FRONT && IRRight < ENEMY_IN_FRONT)
        {
                LastSeen = INRIGHT;

                roomba.Drive(FORWARDSPEEED, CW - 3*SLIGHT_TURN);
                TurningDirection = RIGHT;

                SafeToSearch = false;
                SafeToReadBackIR = true;

                // When was the opponent last seen?
                LastSeenTimer = clock.GetTimestamp();

                #ifdef LEDDEBUG
                        roomba.LED.Spot(OFF);
                        roomba.LED.Max(ON);
                        roomba.LED.DirtDetect(OFF);
                #endif
        }
        // roomba saw enemy in back, not in front?
        else if(IRBack  < ENEMY_IN_BACK  &&
                        IRLeft  > ENEMY_IN_FRONT &&
                        IRRight > ENEMY_IN_FRONT &&
                        SafeToReadBackIR)
        {
                //roomba.LED.DirtDetect(ON);

                // 'SafeToReadBackIR' is enabled when 'BACKTIMER' is full, or 'LASTSEENTIMER' is full, or front IRs saw enemy
                SafeToReadBackIR = false;
                SafeToSearch = false;
                LastSeen = INBACK;

                // mark time when enemy was seen in back.
                BackTimer = clock.GetTimestamp();

                // When was the opponent last seen?
                LastSeenTimer = clock.GetTimestamp();
```

```
                                        // It is faster to turn against enemy by swaping turning
directions.
                                        if(TurningDirection == RIGHT)
                                        {
                                                roomba.Drive(FORWARDSPEEED, CCW);
                                                TurningDirection = LEFT;
                                        }
                                        else if(TurningDirection == LEFT)
                                        {
                                                roomba.Drive(FORWARDSPEEED, CW);
                                                TurningDirection = RIGHT;
                                        }
                                        else
                                        {
                                                // else what?
                                        }

                                        #ifdef LEDDEBUG
                                                roomba.LED.Spot(ON);
                                                roomba.LED.Max(ON);
                                                roomba.LED.Clean(ON);
                                        #endif
                                }
                                else
                                {
                                        // LastSeen is set to "NOTSEEN" only at the beginning
of the fight.
                                       // it's value will change only when roomba sees an enemy.
                                        // look for enemy, if it was recently been seen by
Roomba

                                        #ifdef LEDDEBUG
                                                roomba.LED.Spot(OFF);
                                                roomba.LED.Max(OFF);
                                                roomba.LED.DirtDetect(ON);
                                        #endif

                                        if( (LastSeen != NOTSEEN) && (clock.GetTimestamp()
< LastSeenTimer + LASTSEENTIMER))
                                        {
                                                if(LastSeen == INLEFT)
                                                {
                                                        roomba.Drive(FORWARDSPEEED,
CCW);
                                                        TurningDirection = LEFT;
                                                        SafeToSearch = false;
                                                }
                                                else if(LastSeen == INRIGHT)
                                                {
                                                       roomba.Drive(FORWARDSPEEED, CW);
```

```
                                        TurningDirection = RIGHT;
                                        SafeToSearch = false;
                                }
                                else if(LastSeen == INFRONT)
                                {
                                        roomba.Drive(FORWARDSPEEED,
STRAIGHT);
                                        TurningDirection = NOTTURNING;
                                        SafeToSearch = false;
                                }
                                else if(LastSeen == INBACK)
                                {
                                        if(TurningDirection == LEFT)
                                        {

        roomba.Drive(FORWARDSPEEED, CCW);
                                                TurningDirection = LEFT;
                                                SafeToSearch = false;
                                                //roomba.LED.Spot(ON);
                                        }
                                        else if(TurningDirection == RIGHT)
                                        {

        roomba.Drive(FORWARDSPEEED, CW);
                                                TurningDirection = RIGHT;
                                                SafeToSearch = false;
                                                //roomba.LED.Max(ON);
                                        }
                                }
                        }
                        else
                        {
                                // enemy was not seen for 2000 ms or roomba has
been just started(LastSeen == NOTSEEN), return to SafeToSearch mode.
                                SafeToSearch = true;
                                SafeToReadBackIR = true;
                                //roomba.LED.DirtDetect(OFF);

                                // Mark time, when searching started.
                                SearchingTimer = clock.GetTimestamp();

                                // Set LastSeen direction to unknown
                                LastSeen = UNKNOWN;
                        }
                }
        }


        /*****************************************************************
**********/
```

```
        /*****************************************************************************
***********/

                        /*
                         * BUMPERS
                         * Bumpers are red only when roomba is not over border.
                         */

                        // if both bumpers are set, disable IR reading and attack
                        if(BumpRight && BumpLeft && !OnCliff)
                        {
                                /*
                                if(clock.GetTimestamp() > BumperTimer + 3000)
                                {
                                        roomba.Drive(-BUMPERSPEED, CW);
                                        BumperTimer = clock.GetTimestamp();
                                }
                                else
                                {
                                        roomba.Drive(BUMPERSPEED, STRAIGHT);
                                }
                                */
                                roomba.Drive(BUMPERSPEED, STRAIGHT);
                                TurningDirection = NOTTURNING;

                                SafeToReadFrontIR = false;
                                SafeToAttack = true;
                                LastSeen = INFRONT;

                                #ifdef LEDDEBUG
                                        roomba.LED.Status(ON, ON);
                                #endif
                        }
                        // otherwise, if not on cliff, check again for bumperes.
                        else if(!OnCliff)
                        {
                                // if right bumper is set, disable IR reading and slightly turn right
and attack.
                                if(BumpRight)
                                {
                                        roomba.Drive(BUMPERSPEED, CW -
SLIGHT_TURN);
                                        TurningDirection = RIGHT;

                                        SafeToAttack = true;
                                        SafeToReadFrontIR = false;
                                        LastSeen = INRIGHT;

                                        LastSeenTimer = clock.GetTimestamp();
```

```
                        #ifdef LEDDEBUG
                                roomba.LED.Status(ON, OFF);
                        #endif
                }
                // otherwise enable SafeToAttack flag
                else
                {
                        SafeToAttack = true;
                        SafeToReadFrontIR = true;

                        #ifdef LEDDEBUG
                                roomba.LED.Status(OFF, OFF);
                        #endif
                }
                // if left bumper is set, disable IR reading and slightly turn left
and attack.
                if(BumpLeft)
                {
                        roomba.Drive(BUMPERSPEED, CCW +
SLIGHT_TURN);
                        TurningDirection = LEFT;
                        SafeToAttack = true;
                        SafeToReadFrontIR = false;

                        // enemys last seen direction and time.
                        LastSeen = INLEFT;
                        LastSeenTimer = clock.GetTimestamp();

                        #ifdef LEDDEBUG
                                roomba.LED.Status(OFF, ON);
                        #endif
                }
                // otherwise enable SafeToAttack flag
                else
                {
                        SafeToAttack = true;
                        SafeToReadFrontIR = true;

                        #ifdef LEDDEBUG
                                roomba.LED.Status(OFF, OFF);
                        #endif
                }

        }
        // If on cliff, enable SafeToReadFrontIR flag, and disable SafetoAttack
flag.
        // SafetoAttack is enabled, when not on cliff.
        else
        {
```

```
                        SafeToReadFrontIR = true;
                        SafeToAttack = false;
                }



        //*************************************************************************
**********

        //*************************************************************************
**********

                        /*
                         * CLIFF MODE
                         */

                        // over border and being pushed?
                        if( (CliffLeft || CliffFrontLeft || CliffFrontRight || CliffRight) &&
IRBack < ENEMY_IN_BACK)
                        {
                                // drive backwards until enemy is behind
                                do
                                {
                                        IRBack =  irBack.GetDistanceInCM(5);

                                        roomba.FullMode();
                                        roomba.Drive(BACKWARDDSPEEED, STRAIGHT);
                                        roomba.LED.DirtDetect(ON);

                                } while (IRBack < ENEMY_IN_BACK);

                                roomba.LED.DirtDetect(OFF);
                                OnCliffTimer = clock.GetTimestamp();

                                TurningDirection = NOTTURNING;

                                SafeToSearch = false;
                                SafeToAttack = true;
                                //
                                OnCliff      = false;
                        }
                        //over border?
                        else if(CliffLeft && CliffFrontLeft && CliffFrontRight && CliffRight)
                        {
                                // put roomba to Full Mode, if not already in it. - avoids roomba
from stoping when lifted up.
                                roomba.FullMode();

                                OnCliffTimer = clock.GetTimestamp();
```

```
                roomba.Drive(BACKWARDDSPEEED, STRAIGHT);
                TurningDirection = NOTTURNING;

                SafeToSearch = false;
                SafeToAttack = false;
                OnCliff      = true;
        }
        else if(CliffFrontLeft && CliffFrontRight)
        {
                // put roomba to Full Mode, if not already in it.
                roomba.FullMode();

                OnCliffTimer = clock.GetTimestamp();
                roomba.Drive(BACKWARDDSPEEED, STRAIGHT);
                TurningDirection = NOTTURNING;

                SafeToSearch = false;
                SafeToAttack = false;
                OnCliff      = true;
        }
        else if(CliffLeft)
        {
                // put roomba to Full Mode, if not already in it.
                roomba.FullMode();

                OnCliffTimer = clock.GetTimestamp();
                roomba.Drive(BACKWARDDSPEEED, -BORDERANGLE);
//CW);//;
                TurningDirection = LEFT;

                SafeToSearch = false;
                SafeToAttack = false;
                OnCliff      = true;
        }
        else if(CliffFrontLeft)
        {
                // put roomba to Full Mode, if not already in it.
                roomba.FullMode();

                OnCliffTimer = clock.GetTimestamp();
                roomba.Drive(BACKWARDDSPEEED, STRAIGHT);
                TurningDirection = NOTTURNING;

                SafeToSearch = false;
                SafeToAttack = false;
                OnCliff      = true;
        }
        else if(CliffFrontRight)
        {
                // put roomba to Full Mode, if not already in it.
```

```
                        roomba.FullMode();

                        OnCliffTimer = clock.GetTimestamp();
                        roomba.Drive(BACKWARDDSPEEED, STRAIGHT);
                        TurningDirection = NOTTURNING;

                        SafeToSearch = false;
                        SafeToAttack = false;
                        OnCliff      = true;
                }
                else if(CliffRight)
                {
                        // put roomba to Full Mode, if not already in it.
                        roomba.FullMode();

                        OnCliffTimer = clock.GetTimestamp();
                        roomba.Drive(BACKWARDDSPEEED,
BORDERANGLE);//CCW);//
                        TurningDirection = RIGHT;

                        SafeToSearch = false;
                        SafeToAttack = false;
                        OnCliff      = true;
                }
                else
                {
                        // Not over border.
                        OnCliff = false;

                        // safe to attack ?
                        // enable SafeToAttack flag, if roomba has not been over border
for 'ONCLIFFTIMER' ms
                        if(clock.GetTimestamp() > OnCliffTimer + ONCLIFFTIMER)
                        {
                                SafeToAttack = true;
                        }
                }


        //**********************************************************************
**********

        //**********************************************************************
**********

                        /*
                         * SEARCH MODE
                         */

                        // If roomba is not over border
```

```
                        // SafeToSearch is only enabled, when enemy has not been seen for
'LASTSEENTIMER' ms
                        // or roomba has just been started
                        if(SafeToSearch && !OnCliff)
                        {
                                // If roomba has looked for enemy for 'SEARCHINGTIMER' ms
swap turning directions.
                                if(clock.GetTimestamp() > SearchingTimer +
SEARCHINGTIMER)
                                {
                                        // swap searching directions
                                        SearchDirection = (SearchDirection == (uint16_t)CW) ?
CCW : CW;
                                        // mark turning direction
                                       TurningDirection = (SearchDirection == (uint16_t)CW) ?
RIGHT : LEFT;
                                        // mark time when directions swapped
                                        SearchingTimer = clock.GetTimestamp();
                                }

                                roomba.Drive(FORWARDSPEEED, SearchDirection);
                        }


        //*****************************************************************************
**********

        //*****************************************************************************
**********

                } // FightingAllowed

        } // main while
} // run

void Program::UpdateSenosrdata()
{
        // using only subsets 1 and 2
        roomba.ReadSensors(SUBSET_1);
        roomba.ReadSensors(SUBSET_2);

        //IRLeft = (666 - irLeft.GetRawDistance(5)) / 2;
        IRLeft =  irLeft.GetDistanceInCM(5);
        IRRight = irRight.GetDistanceInCM(5);
        IRBack =  irBack.GetDistanceInCM(5);

        BumpRight = roomba.Sensors.BumpRight;
        BumpLeft  = roomba.Sensors.BumpLeft;
        //WheeldropRight = roomba.Sensors.WheeldropRight;
        //WheeldropLeft  = roomba.Sensors.WheeldropLeft;
```

```
	//WheeldropCaster= roomba.Sensors.WheeldropCaster;

	//Wall = roomba.Sensors.Wall;
	CliffLeft= roomba.Sensors.CliffLeft;
	CliffFrontLeft = roomba.Sensors.CliffFrontLeft;
	CliffFrontRight= roomba.Sensors.CliffFrontRight;
	CliffRight = roomba.Sensors.CliffRight;
	VirtualWall = roomba.Sensors.VirtualWall;

	//SideBrush = roomba.Sensors.SideBrush;
	//Vacuum = roomba.Sensors.Vacuum;
	//MainBrush = roomba.Sensors.MainBrush;
	//DriveRight = roomba.Sensors.DriveRight;
	//DriveLeft = roomba.Sensors.DriveLeft;

	MaxIsPressed   = roomba.Sensors.Max;
	CleanIsPressed = roomba.Sensors.Clean;
	SpotIsPressed  = roomba.Sensors.Spot;
	PowerIsPressed = roomba.Sensors.Power;
}

void Program::HandleLocalCommands()
{
	if(!FightingAllowed)
	{
		// buttons are only in subset 2
		roomba.ReadSensors(SUBSET_2);

		MaxIsPressed   = roomba.Sensors.Max;
		CleanIsPressed = roomba.Sensors.Clean;
		SpotIsPressed  = roomba.Sensors.Spot;
		PowerIsPressed = roomba.Sensors.Power;

		// start attacking to right
		if(MaxIsPressed)
		{
			FightingAllowed = true;
			SearchDirection = CW;
			TurningDirection = RIGHT;

			// Starting delay
			roomba.LED.DirtDetect(ON);
			clock.Delay(START_DELAY);
			roomba.LED.DirtDetect(OFF);
		}
		// start attacking to left
		if(SpotIsPressed)
		{
			FightingAllowed = true;
			SearchDirection = CCW;
```

```
                        TurningDirection = LEFT;

                        // Starting delay
                        roomba.LED.DirtDetect(ON);
                        clock.Delay(START_DELAY);
                        roomba.LED.DirtDetect(OFF);
                }
        }

        /*
         * Shut down?
         */

        if(CleanIsPressed)
        {
                Shutdown();
        }

}

void Program::Reset()
{
        // Reset all default values.
        FightingAllowed   = false;
        SafeToSearch      = true;
        SafeToAttack      = true;
        SafeToReadFrontIR = true;
        OnCliff           = false;
        SafeToReadBackIR  = true;
        LastSeen          = NOTSEEN;
        TurningDirection  = 3;          // 3 a is a random undefined value.

        // get timestamp to mark the time when roomba started.
        SearchingTimer = clock.GetTimestamp();

        // ???
        BackTimer = clock.GetTimestamp();

        #ifndef LEDDEBUG
                roomba.LED.Spot(OFF);
                roomba.LED.Max(OFF);
                roomba.LED.Clean(OFF);
                roomba.LED.DirtDetect(OFF);
                roomba.LED.Status(OFF, OFF);
        #endif
}

void Program::test()
{
        UpdateSenosrdata();
```

```
        //clock.Delay(100);

        if(CleanIsPressed)
        {
                Shutdown();
        }
}

void Program::Shutdown()
{
        roomba.Drive(0, STRAIGHT);
        roomba.PowerOff();
}
```