Institutionen för informationsteknologi

Projektarbete

Självgående robot

Tomas Nilsson

Fredrik Persson

2006-05-15

Abstract

This report is written as a part of the course "Digitala Projekt SK" given at the department of Information Technology. The purpose of the course is to, in a project, develop an application using knowledge gained during the education. In this project an automatically steering robot is constructed. It can move around in a room without hitting any objects. Furthermore it can be controlled via a TV-remote controller.

The robot construction is based on an ATMEL AVR Mega16 micro processor and uses two servo motors for steering and obstacle detection, and two electrical motors for propulsion. An IR-receiver is used for communication with the remote controller.

The project was successful even though one of the main goals wasn't reached due to interference problems caused mainly by the electrical motors.

Innehållsförteckning

Inledning

Kursen Digitala projekt utgörs helt utav ett projektarbete som skall sammanfattas i en projektrapport. Syftet med kursen är att illustrera industriellt utvecklingsarbete.

Målet med projektuppgiften är en prototyp för vidareutveckling med nödvändig dokumentation. Huvuddelen av kursen består i att konstruera, bygga och testa respektive konstruktion.

Vi har valt att utveckla en självgående robot vars syfte ursprungligen var att kunna köra runt obehindrat i inomhusmiljö och att kunna hitta andra robotar av samma typ. På grund av konstruktionsrelaterade hinder kunde det sist nämnda syftet ej uppfyllas. Istället har roboten fått en ny funktion som möjliggör fjärrstyrning med en vanlig TV-fjärrkontroll.

1 Kravspecifikation

1.1 Ursprunglig kravspecifikation

Robot skall:

- vara helt självgående
- ta sig runt hinder
- kunna söka upp en motståndare med hjälp av sin IR-sensor
- därefter kunna närma sig den detekterade motståndaren
- när den förlorar kontakt kunna uppta sökandet på nytt

1.2 Ny kravspecifikation

Robot skall:

- vara helt självgående
- ta sig runt hinder
- vara styrbar med hjälp TV-fjärrkontroll

2 Genomförande

2.1 Hårdvara

I Figur 2.1 finns ett enkelt blockschema över robotens hårdvara. Huvudkomponenten är AVR:en [1] vars uppgift är att samla in data från avståndssensorn och därefter besluta vad roboten skall göra.

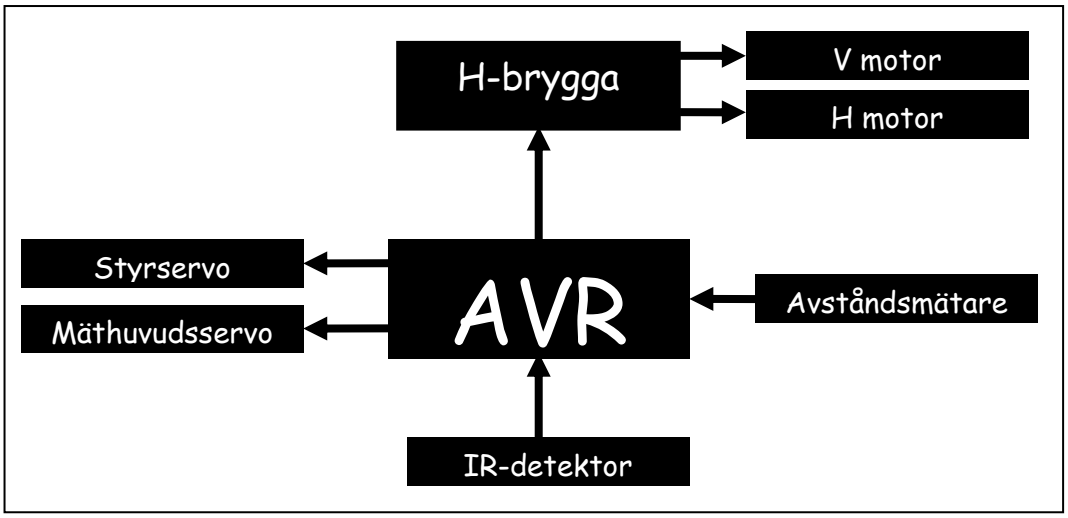
För att kunna styra elmotorerna som har en högre drivspänning än vad AVR:ens utportar kan leverera, används en H-brygga. Denna styrs av signaler direkt från AVR:en men matningsspänningen tas direkt från batteriet. H-bryggan fungerar enligt följande; Logiska signaler öppnar respektive stänger H-bryggan, d.v.s. H-bryggan kan bara släppa igenom eller stänga av all matningsspänning. Beroende av hur stor andel av periodtiden H-bryggan är öppen respektive stängd får motorerna olika hastighet. Denna teknik kallas för switchning.

För att kunna styra roboten med godtaglig precision används ett styrhjul. Detta riktas m.h.a. en servomotor. Den ursprungliga tanken var att riktningen skulle regleras genom att driva hjulen med olika hastigheter. Det visade sig sedermera att motorerna ej var av tillräcklig kvalitet för att med önskad precision kunna behålla önskad riktning.

Robotens enda sensor är en avståndsmätare baserad på IR-teknologi. Denna sitter monterad på en servo som vrider avståndsmätaren fram och tillbaka i en svepande rörelse. Via AVR:en ställs önskat läge in på mäthuvudsservon varpå en avståndsmätning sker. På så sätt vet man hur långt fältet är fritt i en given riktning.

IR-detektorn (se Figur 2.1) används för att ta emot kommandon från en TV-fjärrkontroll. Möjliga styrkommandon är:

- kör på egen hand
- stå still
- kör framåt
- kör bakåt
- kör fram åt vänster
- kör fram åt höger



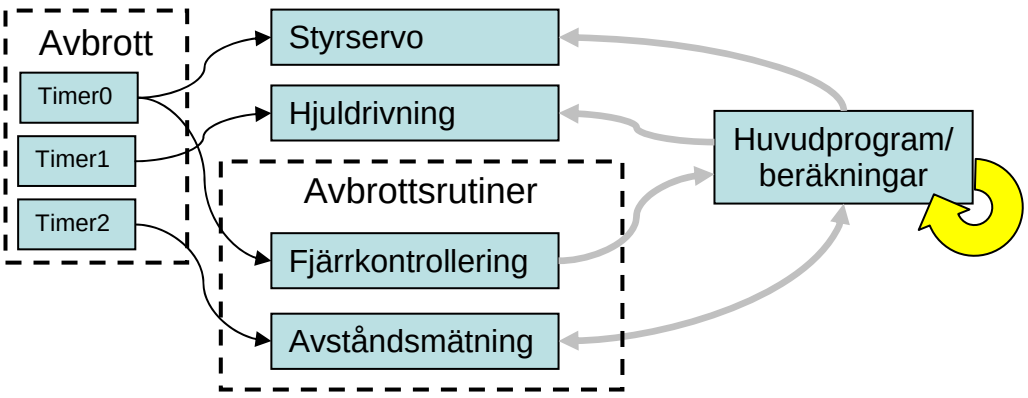
Figur 2.1: Blockschema över roboten.

2.2 Mjukvara

All mjukvara är skriven i språket C. Styrningen utgörs i stort sett av tre stycken timrar som triggar avbrottsrutiner och ett huvudprogram som utför beräkningar på insamlad data mellan avbrottsrutinernas exekvering, se Figur 2.2. Avbrottsrutinerna används enligt följande:

- En för att ta emot signaler från fjärrkontroll. Denna avbrottsrutin använder Timer0 för att mäta hur långa pulserna från fjärrkontrollen är. Denna information används sedan för att matcha mot kända pulssekvenser för olika knappar.

- En för att styra avståndsmätarsensorn samt att avläsa avståndssensorn i ett periodiskt intervall.

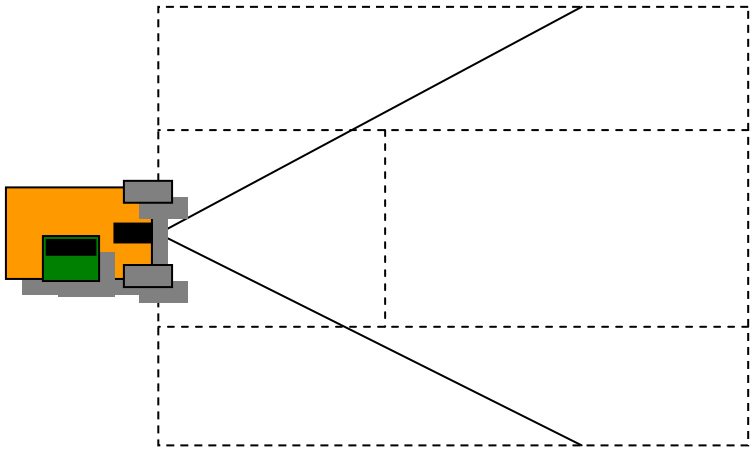


Figur 2.2: Blockschema över styrprogrammet.

2.2.1 Informationsinhämtning

Informationsinhämtningen sker periodiskt m.h.a. en timer. Var gång Timer2 (se Figur 2.2) genererar ett avbrott avläses avståndssensorns värde m.h.a. en av AVR:ens inbyggda A/D-omvandlare. Därefter skickas ett nytt börvärde till servon. Under tiden som löper mellan två avbrott hinner servon ställa sig i nästa position.

Figur 2.3 visar styrprogrammets beslutsunderlag, de solida strecken anger avståndsservons ändlägen och de streckade anger synfältets indelning. Beroende på inom vilken zon en mätpunkt hamnar beslutar styrprogrammet vilken hastighet och riktning roboten skall ha. För att minimera risken att köra fast backar roboten om föremål befinner sig inom den innersta streckade rutan.



Figur 2.3: Bills synfältsindelning; De solida strecken representerar synfältet d.v.s. de ändpunkter mellan vilka servon rör sig. De streckade linjerna representerar de gränser som styrprogrammet använder för att ta beslut angående hastighet och riktning.

2.2.2 Styrning

Eftersom roboten ursprungligen skulle styras endast genom att variera hastigheten på höger respektive vänster hjul görs det fortfarande trots att roboten nu även har ett styrhjul. Detta fyller dock fortfarande en funktion eftersom drivaxeln ej har någon differential. Tack vara styrhjulet har roboten mycket hög precision när den svänger, den avviker t.ex. väldigt lite från en rak linje när styrhjulet är ställt i mittläget.

Användaren kan ta över kontrollen över roboten m.h.a. en fjärrkontroll. När användaren tar kontrollen fortsätter avståndsmätaren att göra sina mätningar men informationen behandlas ej av styrprogrammet.

I Figur 2.2 ser man att Timer0 och Timer1 styr hjulservo respektive hjuldrivning utan att använda sig av en avbrottsrutin. Detta görs möjligt genom att AVR-processorn kan ändra en utsignal var gång en timer får ett givet värde. Det är mycket lämpligt att använda denna funktion då man slipper utnyttja avbrottsrutiner och därmed processortid för att utföra styrningarna.

3 Diskussion

Det stora problemet har genomgående varit störningar. Den största störningskällan är elmotorerna vilka genererar kraftiga variationer i systemets jordpunkt. För att motverka dessa störningar kopplades specialkondensatorer över motorerna. Detta motverkade en stor del av störningarna men inte tillräckligt för att ge en helt stabil jordpunkt.

Skälet till att roboten ej uppfyller den ursprungliga specifikationen d.v.s. att den ej har möjlighet att hitta andra robotar beror till största del på motorstörningarna. Ett annat problem är att strålningen från solen och även lysrör innehåller bl.a. IR-ljus. Dessa störkällor har dessutom betydligt högre (i alla fall vid direkt solsken) intensitet än vad man kan uppnå med en vanlig liten lysdiod.

För att komma runt problemen med solsken och lysrörsbelysning användes en avskärmning i form av en liten strut på mottagardioden. Denna fungerade bra men åtgärdade naturligtvis ej problemen med störningarna genererade av elmotorerna. Dessutom begränsade den IR-fotodiodens detektionsområde.

En annan lösning som också prövades var att byta ut IR-dioderna mot frekvensmodulerade dioder (liksom de som används i t.ex. TV-kontrollen). Detta medförde det motsatta problemet att roboten nu kunde detektera den andra roboten oavsett i vilken riktning fotodioden var ställd. Problemet kvarstod även om en strut sattes på fotodioden eftersom ljuset reflekterades i väggar o.d.

Vi fick ändå nytta av den frekvensmodulerade IR-dioden eftersom vi kunde använda denna för att ta emot signaler från TV-fjärrkontrollen. För detta ändamål lämpade den sig naturligtvis mycket bättre.

4 Referenslista

[1] Atmel Corporation 2003, Digitala projekt, ATmega16.

[2] Kernighan, Brian W., Ritchie, Dennis M., The C programming language, second edition, 1988

Main program

```
#include <avr/io.h>
#include <avr/interrupt.h>

#include <init.h>
#include <definitions.h>
#include <interrupts.h>
#include <functions.h>

// Main program
int main(void) {
        /* Wait for startbutton */

        PORTB = 1;
        while (PINB & 1) ;
        PORTB = 0;
        ir_command = 255;

        /* Do the initiations */
        ioinit ();

        /* Run this until powerloss */
        for (;;) {
                if (ir_command != 255) {
                        switch (ir_command) {
                                case 10: {
                                        stop_car++;
                                        stop_car %= 2;
                                        if (stop_car)
                                                setCar(Steer_S, Drive_F, Speed_Stop);
                                        } break;
                                case 2: {
                                        stop_car = 1;
                                        setCar(Steer_S, Drive_F, Speed_Fast);
                                        } break;
                                case 4: {
                                        stop_car = 1;
                                        setCar(Steer_L, Drive_F, Speed_Normal);
                                        } break;
                                case 6: {
                                        stop_car = 1;
                                        setCar(Steer_R, Drive_F, Speed_Normal);
                                        } break;
                                case 8: {
                                        stop_car = 1;
                                        setCar(Steer_S, Drive_B, Speed_Fast);
                                        } break;
                                case 5: {
                                        stop_car = 1;
                                        setCar(Steer_S, Drive_B, Speed_Stop);
                                        } break;
                        }
                        ir_command = 255;
                }

                /* Do calc when servo stands in middle and in endpoints */
                if (!stop_car && doCalc > 0) {
                        calcAreas();
                        drive();
                        doCalc = 0;             // Wait until intteruptroutine tells us to check again
                }
        }
        return 0;
}
```

Definitions

```
// Functions
void init(void);
void setCar(short steer_dir, short drive_dir, short speed);
void drive();
void calcAreas(void);
void match_ir_cmd(uint8_t time);
void zero_match();


// Motor definitions
#define Engine PORTC                            // Engine out ports
#define Init_Engine DDRC                        // Engine data direction reg
#define Pulse_L OCR1A                           // Puls length determiner (0 -> no puls, 255 -> all puls)
#define Pulse_R OCR1B

#define Forw_L PC7                                      // Bits to set for different directions
#define Back_L PC6
#define Forw_R PC1
#define Back_R PC0

#define Drive_F 1                                       // Drive directions
#define Drive_B 2                                       //

#define Steer_L 1                                       // Steer directions
#define Steer_S 2                                       //
#define Steer_R 3                                       //

#define Speed_UFast             220                     // Different speeds
#define Speed_SFast     180                     //
#define Speed_VFast     140                     //
#define Speed_Fast      100                     // Optimal normal pace
#define Speed_Normal    70                      // Optimal slow pace
#define Speed_Stop              0                       //

// Sensor Servo definitions
#define Timer2_Settings (1<<WGM21) | (0<<WGM20) | (0<<COM21) | (1<<COM20)

// Direction Servo Definitions
#define SteeringWheel   OCR0            // Where to write directins
#define Turn_VHRight    22                      // Very Hard Right (11)
#define Turn_HRight             30                      // Hard Right (15)
#define Turn_SRight             40                      // Soft Right (20)
#define Turn_VSRight    42                      // Very Soft Right (21)
#define Turn_Straight   44                      // Straight (22)
#define Turn_VSLeft             46                      // Very Soft Left (23)
#define Turn_SLeft              48                      // Soft Left (24)
#define Turn_HLeft              58                      // Hard Left (29)
#define Turn_VHLeft             68                      // Very Hard Left (34)

// Intelligence variables
volatile uint8_t distance[30];                          // "Distance to obstacles" vector
volatile uint8_t sight_L1, sight_L2, sight_L3; // Obstacle detection zones
volatile uint8_t sight_R1, sight_R2, sight_R3; // Obstacle detection zones

volatile uint8_t area3_first, doCalc; // Booleans

volatile uint8_t bBox[30], sBox[30], tBox[30];
volatile int8_t forceDir_area3;

// Servo variables
volatile int8_t pulse_width;                    // The length of the servo control pulse
volatile int8_t servo_Dir;

// IR Variables
volatile uint8_t stop_car;

volatile uint8_t counter;

volatile uint8_t last_time;

volatile uint8_t sync_array[11];
volatile uint8_t sync_pointer;

volatile uint8_t match[7];
volatile uint8_t ir_command;

volatile uint8_t button_2[9];
volatile uint8_t button_4[9];
volatile uint8_t button_5[9];
volatile uint8_t button_6[9];
volatile uint8_t button_8[9];
volatile uint8_t button_off[9];
volatile uint8_t button_0[9];

```

Functions

```
#include <definitions.h>
#include <math.h>


/* This function should be called whenever the
main program wishes to change direction or speed*/
void setCar(short steer_dir, short drive_dir, short speed) {
        /* Set direction on wheels */
        uint8_t vel_L, vel_R, dir_L, dir_R, steer;

        if (drive_dir == Drive_F) {
                dir_L = Forw_L;
                dir_R = Forw_R;
        } else {
                dir_L = Back_L;
                dir_R = Back_R;
        }

        /* Set direction on steering wheel and speed on drive wheels */
        if (steer_dir != Steer_S) {
                switch (speed) {
                        case Speed_UFast:
                                if (steer_dir == Steer_L) {
                                        steer = Turn_VSLeft;
                                        vel_L = Speed_SFast;
                                        vel_R = Speed_UFast;
                                }
                                else {
                                        steer = Turn_VSRight;
                                        vel_L = Speed_UFast;
                                        vel_R = Speed_SFast;
                                }
                                break;
                        case Speed_SFast:
                                if (steer_dir == Steer_L) {
                                        steer = Turn_VSLeft;
                                        vel_L = Speed_VFast;
                                        vel_R = Speed_SFast;
                                }
                                else {
                                        steer = Turn_VSRight;
                                        vel_L = Speed_SFast;
                                        vel_R = Speed_VFast;
                                }
                                break;
                        case Speed_VFast:
                                if (steer_dir == Steer_L) {
                                        steer = Turn_VSLeft;
                                        vel_L = Speed_Fast;
                                        vel_R = Speed_VFast;
                                }
                                else {
                                        steer = Turn_VSRight;
                                        vel_L = Speed_VFast;
                                        vel_R = Speed_Fast;
                                }
                                break;
                        case Speed_Fast:
                                if (steer_dir == Steer_L) {
                                        steer = Turn_SLeft;
                                        vel_L = Speed_Normal;
                                        vel_R = Speed_Fast;
                                }
                                else {
                                        steer = Turn_SRight;
                                        vel_L = Speed_Fast;
                                        vel_R = Speed_Normal;
                                }
                                break;
                        case Speed_Normal:
                                if (steer_dir == Steer_L) {
                                        steer = Turn_HLeft;
                                        vel_L = Speed_Normal;
                                        vel_R = Speed_Normal;
                                }
                                else {
                                        steer = Turn_HRight;
                                        vel_L = Speed_Normal;
                                        vel_R = Speed_Normal;
                                }
                                break;
                }
        } else {
                steer = Turn_Straight;
                vel_L = vel_R = speed;
        }

        Pulse_L = vel_L;
        Pulse_R = vel_R;

        SteeringWheel = steer;

        Engine = (1<<dir_L) | (1<<dir_R);       // Start wheels again

}
void drive() {
```

```
        /* Make choices based on how many points are
        found in each area and the preferred direction */
        uint8_t area_1 = sight_R1 + sight_L1;
        uint8_t area_2 = sight_R2 + sight_L2;
        uint8_t area_3 = sight_R3 + sight_L3;

        if (area_1+area_2+area_3 < 2) {
                setCar(Steer_S, Drive_F, Speed_SFast);
                forceDir_area3 = 0;
                area3_first = 1; // If something comes close again, car will stop
        }

        else {
                /* if something is in this area, first we stop then
                we take evasive action */
                if (area_3 > 1) {
                        /* If car has already been told to stop
                        try to go backwards out of the situation */
                        if (!area3_first) {
                                /* If bill is turning, he shoul keep on
                                                turning until he is free */
                                if (forceDir_area3 == 0) {
                                        if (sight_L3 > sight_R3) {
                                                forceDir_area3 = 1;
                                                setCar(Steer_L, Drive_B, Speed_Normal);
                                        } else {
                                                forceDir_area3 = -1;
                                                setCar(Steer_R, Drive_B, Speed_Normal);
                                        }
                                }
                        }
                        /* else tell car to stop */
                        else {
                                setCar(Steer_S, Drive_B, Speed_Stop);
                                area3_first = 0; // Don't stop again until we have been outside area3
                        }
                } else if (area_2 > 1) {
                        if (sight_L2 > sight_R2) {
                                setCar(Steer_R, Drive_F, Speed_Fast);
                        }

                        else {
                                setCar(Steer_L, Drive_F, Speed_Fast);
                        }

                        forceDir_area3 = 0;
                        area3_first = 1;
                }

                else if (area_1 > 1) {
                        setCar(Steer_S, Drive_F, Speed_VFast);
                        forceDir_area3 = 0;
                        area3_first = 1; // If something comes close again, car will stop
                }

                else
                        setCar(Steer_S, Drive_F, Speed_SFast);
        }
}

/* Calculate obstacle density for each area */
void calcAreas(void) {
        sight_R1 = sight_R2 = sight_R3 = sight_L1 = sight_L2 = sight_L3 = 0; // Zero counters
        int i, x;
        for(i = 0; i < 30; ++i) {               // Go through the 30 angles in the middle
                x = distance[i];                // Get distance for given angle
                if ((x > 15) && (x < 129)) {    // If point is within area of interest
                        if (i < 15) {           // Start with right side counters
                                if (x > bBox[i]) { // If point is within the Big Box
                                        if (x > tBox[i]) {// If point is within the Turn Box
                                                if (x > sBox[i]) {// If point is within the Small Box
                                                        sight_R3++;// Count Small Box Right Up
                                                } else {// Point is within the Turn Box Right Area
                                                        sight_R2++;// Count Turn Box Right Up
                                                }
                                        } else {        // Point is within the Big Box Right Area
                                                sight_R1++;             // Count Big Box Right Up
                                        }
                                }
                        } else {                        // Proceed with the left side counters
                                if (x > bBox[i]) {      // If point is within the Big Box
                                        if (x > tBox[i]) {// If point is within the Turn Box
                                                if (x > sBox[i]) {// If point is within the Small Box
                                                        sight_L3++;// Count Small Box Left Up
                                                } else {        // Point is within the Turn Box Left Area
                                                        sight_L2++;     // Count Turn Box Left Up
                                                }
                                        } else {        // Point is within the Big Box Left Area
                                                sight_L1++;     // Count Big Box Left Up
                                        }
                                }
                        }
                }
        }
}

void match_ir_cmd(uint8_t time) {
```

```
        if (match[0]) {
                if (time >= button_2[counter]-1 && time <= button_2[counter]+1) {
                        if (counter == 6) {
                                counter = 255;
                                zero_match();
                                ir_command = 2;
                        }
                } else {
                        match[0] = 0;
                }
        }

        if (match[1]) {
                if (time >= button_4[counter]-1 && time <= button_4[counter]+1) {
                        if (counter == 6) {
                                counter = 255;
                                zero_match();
                                ir_command = 4;
                        }
                } else {
                        match[1] = 0;
                }
        }

        if (match[2]) {
                if (time >= button_5[counter]-1 && time <= button_5[counter]+1) {
                        if (counter == 4) {
                                counter = 255;
                                zero_match();
                                ir_command = 5;
                        }
                } else {
                        match[2] = 0;
                }
        }

        if (match[3]) {
                if (time >= button_6[counter]-1 && time <= button_6[counter]+1) {
                        if (counter == 6) {
                                counter = 255;
                                zero_match();
                                ir_command = 6;
                        }
                } else {
                        match[3] = 0;
                }
        }

        if (match[4]) {
                if (time >= button_8[counter]-1 && time <= button_8[counter]+1) {
                        if (counter == 6) {
                                counter = 255;
                                zero_match();
                                ir_command = 8;
                        }
                } else {
                        match[4] = 0;
                }
        }

        if (match[5]) {
                if (time >= button_off[counter]-1 && time <= button_off[counter]+1) {
                        if (counter == 6) {
                                counter = 255;
                                zero_match();
                                ir_command = 10;
                        }
                } else {
                        match[5] = 0;
                }
        }

        if (match[6]) {
                if (time >= button_0[counter]-1 && time <= button_0[counter]+1) {
                        if (counter == 8) {
                                counter = 255;
                                zero_match();
                                ir_command = 0;
                        }
                } else {
                        match[6] = 0;
                }
        }

        counter++;
}

void zero_match() {
        int i;
        for (i = 0; i < 7; i++) {
                match[i] = 1;
        }
}
```

Initiations

```
#include <definitions.h>

void ioinit (void) {
        /* Wheel initiation */
        Init_Engine = (1<<Forw_L) | (1<<Back_L) | (1<<Forw_R) | (1<<Back_R);    // Set pins in DDRC
        setCar(Steer_S, Drive_F, Speed_Stop);           // Tell car to stand still at first

        /* Init ADC Väljer AVCC som ref-spänning (s215) */
        ADMUX = (0<<REFS1) | (1<<REFS0);
        /* Väljer ADC0 som in-pinne. (s216) */
        ADMUX |= (0<<MUX4) | (0<<MUX3) | (0<<MUX2) | (0<<MUX1) | (0<<MUX0);
        /* För att få resultatet VÄNSTERjusterat (0 är standard = högerjusterat) */
        ADMUX |= (1<<ADLAR);
        /*  Sätter ADC till Free Running Mode (Auto Trigger Enable) (s219) */
        ADCSRA |= (1<<ADATE);
        /* Sätter ADC-clk till (8M/8) 1000kHz (se s218) */
        ADCSRA |= (0<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
        /* Enables A/D-omvandlaren       */
        ADCSRA |= (1<<ADEN);
        /* Startar första omvandlingen*/
        ADCSRA |= (1<<ADSC);

        /* ----------------Init Timers ------------------*/

        /* ---Timer0 Direction Servo Control --*/
        /* Set pin OC0 as output */
        DDRB |= (1<<DDB3);
        /* Set timer to Fast PWM */
        TCCR0 |= (1<<WGM01) | (1<<WGM00);
        /* Set PWM to non-inverted mode */
        TCCR0 |= (1<<COM01) | (0<<COM00);
        /* Set prescaler to 256, with Fast PWM, length of period will be ~ 16ms */
        TCCR0 |= (1<<CS02) | (0<<CS01) | (0<<CS00);
        /* Set compare value 0 (stop) */
        SteeringWheel = Turn_Straight;

        /* -----Timer1 Enginge Control --------*/
        /* Set pin OC1B and OC1A (in that order) as output */
        DDRD |= (1<<PD4) | (1<<PD5);
        /* Set timer to PWM, Phase Correct, 8-bit (0x00FF as TOP-value) (page 111) */
        TCCR1B |= (0<<WGM13) | (0<<WGM12);
        TCCR1A |= (0<<WGM11) | (1<<WGM10);
        /* Set PWM to non-inverted mode (page 110) */
        TCCR1A |= (1<<COM1A1) | (1<<COM1B1) | (0<<COM1A0) | (0<<COM1B0);
        /* Set prescaler to 256(64 -> pwm = 245Hz) (page 112) */
        TCCR1B |= (1<<CS12) | (0<<CS11) | (0<<CS00);

        /* --Init Timer2 Sensor Servor Control-*/
        /* Set pin OC2 as output */
        DDRD |= (1<<DDD7);
        /*set the Clock Select Bits (s 83) */
        TCCR2 = Timer2_Settings | (1<<CS22) | (0<<CS21) | (0<<CS20);
        /*set Output Compare Match Interrupt Enable */
        TIMSK |= (1<<OCIE2);

        /*--- Set Interrupt things for IR---*/
        DDRD |= (0<<INT0);
        MCUCR |= (0<<ISC01) | (1<<ISC00);
        GICR |= (1<<INT0);

        /*set initial value for pulse_width ie straight forward and reset Servo*/
        pulse_width = 14;
        servo_Dir = 1;

        /* Intelligence variables */
        forceDir_area3 = area3_first = doCalc = 0;

        /* IR Variables */
        stop_car = 1;
        last_time = 0;
        sync_pointer = 0;
        ir_command = 255;
        counter = 255;

        /* Big (Outer) Box */
        uint8_t temp_bBox[30] = {39, 38, 35, 33, 29, 26, 21, 15, 16, 17,
                                 17, 17, 17, 18, 18, 17, 17, 17, 17, 17,
                                 16, 16, 15, 21, 28, 32, 35, 38, 40, 42};

        /* Small (Inner) Box */
        uint8_t temp_sBox[30] = {112, 111, 111, 111, 110, 108, 106, 102, 86, 69,
                                 70, 71, 73, 74, 74, 74, 74, 74, 74, 72,
                                 70, 86, 101, 106, 108, 109, 110, 111, 111, 111};

        /* Turn (Turn Area) Box */
        uint8_t temp_tBox[30] = {99, 95, 92, 86, 80, 70, 61, 50, 39, 29,
                                 19, 18, 17, 18, 18, 17, 17, 17, 17, 17,
                                 18, 39, 49, 61, 69, 80, 87, 92, 95, 99};

        int i;
        for (i = 0; i < 30; ++i) {
                bBox[i] = temp_bBox[i];
                sBox[i] = temp_sBox[i];
                tBox[i] = temp_tBox[i];
        }

        /* IR STUFF */
```

```
 98         uint8_t temp_sync_array[11] = {31, 14, 16, 13, 17, 13, 16, 13, 17, 13, 16};
 99         uint8_t temp_button_2[9]        = {13, 17, 13, 17, 27, 31, 13, 0, 0};
100         uint8_t temp_button_4[9]        = {13, 17, 27, 31, 13, 17, 13, 0, 0};
101         uint8_t temp_button_5[9]        = {13, 17, 27, 31, 27, 0, 0, 0, 0};
102         uint8_t temp_button_6[9]        = {13, 17, 27, 17, 13, 31, 13, 0, 0};
103         uint8_t temp_button_8[9]        = {27, 31, 13, 17, 13, 17, 13, 0, 0};
104         uint8_t temp_button_off[9]      = {27, 17, 13, 31, 13, 17, 13, 0, 0};
105         uint8_t temp_button_0[9]        = {13, 17, 13, 17, 13, 17, 13, 17, 13};
106
107         for (i = 0; i < 11; ++i) {
108                 sync_array[i] = temp_sync_array[i];
109         }
110
111         for (i = 0; i < 9; ++i) {
112                 button_2[i] = temp_button_2[i];
113                 button_4[i] = temp_button_4[i];
114                 button_5[i] = temp_button_5[i];
115                 button_6[i] = temp_button_6[i];
116                 button_8[i] = temp_button_8[i];
117                 button_off[i] = temp_button_off[i];
118                 button_0[i] = temp_button_0[i];
119         }
120
121         /* Enable global interrupts */
122         sei ();
123 }
```

Interrupt routines

```
#include <definitions.h>


/* Servomotion and sensor reading */
SIGNAL (SIG_OUTPUT_COMPARE2) {
        if (PIND & (1<<PIND7)) {
                /* Set prescaler to a low value (128) */
                TCCR2 = Timer2_Settings | (1<<CS22) | (0<<CS21) | (1<<CS20);

                /* Store a new value in the distance vector */
                distance[pulse_width] = ADCH;

                /* Determine direction and pulsewidth */
                if (pulse_width >= 29) {
                        pulse_width = 28;
                        servo_Dir = -1;
                        doCalc = 1;
                } else if (pulse_width <= 0) {
                        pulse_width = 1;
                        servo_Dir = 1;
                        doCalc = 3;
                } else {
                        pulse_width += servo_Dir;
                        if (pulse_width == 14)
                                doCalc = 2;
                        if (pulse_width == 21 || pulse_width == 7)
                                doCalc = 4;
                }

                /* set timer OCR2 to its value */
                OCR2 = 62 + pulse_width*2;
        } else {
                /* Set prescaler to a high value (1024) */
                TCCR2 = Timer2_Settings | (1<<CS22) | (1<<CS21) | (1<<CS20);

                /* set timer OCR2 to long puls */
                OCR2 = 120 - OCR2/8;
        }
}

/* Servomotion and sensor reading */
SIGNAL (SIG_INTERRUPT0) {
        uint8_t time, new_time;

        time = new_time = TCNT0;

        if (last_time > time)
                time += 256;

        time -= last_time;

        if (counter >= 0 && counter < 9) {
                match_ir_cmd(time);

        } else { // sync-fase
                if (time >= sync_array[sync_pointer]-1
                        && time <= sync_array[sync_pointer]+1) {
                        sync_pointer++;
                        if (sync_pointer == 11) {
                                sync_pointer = 0;
                                counter = 0;
                                zero_match();
                        }
                } else {
                        sync_pointer = 0;
                }
        }

        if (counter == 9) {
                counter = 10;
        }


        last_time = new_time;
}
```