Fester med gester

- ett projekt i kursen EITF10 - *Digitala Projekt (I).*

Robin Mellstrand

Kristoffer Frang

Abstract

Reading about a project where an accelerometer was used to control an mp3-player we came up with the idea of putting words to the everyday gestures we are making. The construction is based on an accelerometer that recognizes movement patterns in order to identify different gestures. For example can a raised hand be identified and fallowed by a speaker playing a recorded voice saying “hi”. However, due to lack of time we unfortunately didn’t have the time to set up the speaker so we limited our project to the movement pattern identification using LED light in order to acknowledge a specific movement. The system is also able to record new movement pattern for each LED light. There is a limit of four simultaneous recorded gestures.

Inledning

Projektets grundläggande syfte är att ljudsätta rörelser. Tanken är att man med olika handrörelser skall trigga igång en förinspelad ljudsnutt som motsvarar den gjorda rörelsen. Om man t.ex. höjer handen som i en hälsning skall en förinspelad höras som säger “hej!”.

Rörelsemönstren skall kunna spelas in med en recordfunktion som triggas med en knapp.

Systemet implementerades med hjälp av en accelerometer för att känna av accelerationen i x- och y-led. För att problemet skall bli hanterbart och ej för komplext valdes endast mätningar i två dimensioner trots att den använda accelerometer klarar av tre. Projektet begränsades till att endast kunna identifiera olika rörelsemönster och koppla dessa till lysdioder, snarare än till en högtalare. Detta eftersom att det efterhand blev uppenbart att tidsbördan för utveckling av mjukvaran för identifiering av rörelsemönster blev för stor. Max antal simultant inspelade rörelser sattes till fyra.

Teori

Igenkänning av rörelsemönster

Grundtanken för igenkännande av rörelser är att sampla värden som sedan jämförs med inspelade värden i en tabell. Stämmer ett värde i tabellen går programmet vidare och ser om även nästa samplade värde stammer mot nästa värde I tabellen. Om alla samplade värden stämmer mot hela tabellen har en rörelse identifierats. Skulle ett samplat värde inte stämma överens med tabellvärde åtegår programmet att jämföra nya samplade värden med det första i tabellen igen. Nedan följer en lista med de olika stegen.

1. Programmet loopar I en oändlig loop där vi samplar ett nytt värde från accelerometern I varje varv.

2. I loopen kontrolleras först om record-knappen är nedtryckt. Då går vi till recordläge. Mer om det senare.

3. Sen kontrolleras om ”rörelseväljar”-knappen är nedtryckt. Den anger vilken rörelse som skall spelas in nästa gång record-knappen nedtrycks. Trycks ”rörelseväljar”-knappen ned väljs ”nästa” rörelse i ordning och den korrespenderande lampan lyses upp en kort stund för att konfirmera detta.

4. Vidare kontrollerar loopen om det samplade värdet stämmer överens med aktuellt tabellvärde i de fyra tabellerna som motsvarar de fyra rörelser som kan spelas in. Det finns alltså en if-sats för varje tabell. Varje tabell innehåller 4 värden.

5. Stämmer det samplade värdet med det aktuella värdet i någon av tabellerna sätts det aktuella värdet till nästa i tabellen och nästa samplade värdet blir jämfört med detta. Skulle tabellen vara

slut och alltså alla värden i tabellen är jämförda är rörelsen igenkänd och motsvarande diod tänds.

Inspelning av rörelsemönster

För att spela in rörelser måste inspelningstabell först väljas. Detta görs genom ”rörelseväljar”-knappen beskriven ovan. Record-knappen trycks sedan ner och programmet går in i inspelningsfunktionen. Programmet ställer sig sedan i vila och väntar på att accelerometern är helt stilla. När den uppfattat att indikeras detta med att en diod tänds och sedan börjar den lyssna på nästa rörelse. När rörelse uppfattats börjar programmet samla in värden och spara i den tidigare valda tabellen. När inspelningen är klar återgår programmet till huvudloopen.

Genomförande och problem på vägen

Nedan följer en mer detaljerad beskrivning av de olika momenten, förändringar vi var tvugna att göra samt de problem som uppkom på vägen.

Få korrekta värden från accelerometern

En av de stora bryderierna var att få korrekta värden från accelerometern. Komponenten gav analoga utslag till processorns A/D-omvandlare som skall tolka om detta till digitala signaler. Till en början uppvisade den helt irrationella mätvärden som inte korresponderade mot rörelserna den borde ha känt igen, ibland visade den inga värden alls och ibland så visade den samma värden i X-led och Y-led hela tiden.

Det var ett ganska stort problem att förstå alla ingående delar, register och funktioner hos komponenten. Till en början resulterade samtliga rörelser i enbart högsta utslag eller lägsta utslag (enbart ettor eller enbart nollor i reslutatregistrena). Detta berodde på att vi ej insett vikten av referensspänningen till A/D-omvandlaren i processorn. Referensspänningen avgör inom vilka intervall omvandlingarna sker, halva referensspänningen anses som viloläge hos accelerometern. Sedan ger rörelser antingen ett positivt eller negativt utslag därifrån. Vi började med att laborera med den inbyggda referensspänningen på 2.56V men då detta gav oss dåliga värden insåg vi att vi måste ha samma referensspänning till A/D-omvandlaren som spänningen accelerometern går på.

Nästa problem var A/D-omvandlingens relativt oexakta natur. Eftersom X-värdena och Y-värdena lästes av vid olika pinnar på processorn krävdes det att vi läste in från olika portar om vartannat. Detta fungerar dock inte i praktiken eftersom första omvandlingen efter en omställning av portarna (som skedde mellan varje omvandling) blev felaktig och manualen till processorn rådde oss att alltid kasta detta värde. Detta löste vi genom att läsa in två värden för varje omvandling och enbart använda det andra. Vidare klarar AD-omvandlaren ungefär 5000 omvandlingar/s (enligt empiriska mätningar) varför vi lät varje inläst

värde till programmet motsvara ett genomsnitt av 1000 omvandlingar. Det gav oss mer exakta mätningar som kompenserade för små fel i rörelsen samt gav oss den tiden vi behövde för att kunna genomföra en hel rörelse innan resultattabellen var full. Resultattabellen består, som beskrivet nedan, endast av 4 värden vilket gav oss ungefär en sekund på oss att genomföra den aktuella gesten. Detta visade sig vara tillräckligt för de något enkla rörelser vi använde oss av.

Nästa problem med accelerometern var upplösningen på de olika resultatregistren. Resultaten sparas i två register på 8+2 bitar. Till en början använde vi oss endast av de 8 mest signifikanta bitarna i resultatet, men det visade sig att detta gav ytterst små utslag mellan olika rörelser. Dock löste sig detta någorlunda när vi lade till de två minst signifikanta bitarna till resultatet och fick en upplösning på 4x av det föregående.

När vi lyckats fundera ut ovanstående och reda ut vilka register som behövde aktiveras för att en omvandling ska ske på rätt sätt och vilka register som behövdes läsas i vilken ordning så började accelerometern ge bra mätvärden. Detta var en viktig milstolpe i projektet och ökade optimismen för att vi faktiskt skulle kunna ro iland projektet, något vi tvivlat på många gånger under resans gång.

Igenkänning av rörelsemönster

Vi dividerade länge om hur många mätvärden som skulle innefattas av en rörelse. Från början hade vi en tabellstorlek på 30, alltså 30 värden skulle stämma överens för att känna igen en rörelse. Vi insåg slutligen att detta var på tok för många och att det skulle krävas för hög precision återgivningen av rörelsemönstret för att den skulle matcha alla tabellvärden. Den färdigställda produkten har bara fyra mätvärden som skall stämma. Detta visades vara en bra kompromiss mellan lättigenkännlighet hos rörelsemönstret och risken att fel rörelse skulle identifieras.

Vi disskuterade också om vi skulle mäta fyra momentana värden under rörelsen kontra använda snittvärden för de fyra delarna. Vi kom fram till att det enda rimliga var att räkna ut snittvärden för rörelsen. Alltså att vi delar upp rörelsen I fyra delar där varje värde för varje del är ett snit av accelerationerna under den delen.

Vidare använder vi spann som värdet skall gälla inom för att det skall räknas som godkänt och stämma överens med det i tabellen.

Spänningstransfomering till accelerometern

Att accelerometern krävde en matningsspänning på ~3.3V och övriga komponenter 5V kändes till en början som ganska trivialt. Detta visade sig senare bli ett relativt stort problem.

Vi fick en spänningsregulator av vår handledare som skulle ordna detta men i det första försöket använde vi oss, av misstag, av databladet för en snarlik produkt som använde sig av annan slags teori och kopplingsschema. Detta resulterade självfallet i att vi inte fick den utspänningen vi hade räknat med. Efter konsultering med handledaren insåg vi detta och försökte med rätt kopplingsschema lösa detta. Detta fungerade inte heller och efter många bryderier ansåg vi att komponenten var trasig. Så efter ett

komponentbyte och ytterligare modifieringar av kopplingsschemat fick vi, till vår stora lycka, rätt spänning från komponenten.

Hårdvaruprogrammering med ett nytt språk

Då projektgruppens erfarenheter att programmera i språket C var trappsteget i början relativt stort när vi startade programvaruutvecklingen. C är relativt likt Java så många funktioner kände vi igen, men saker som variabel - och funktionsdeklarationer spelade helt plötsligt mycket större roll. Vidare uppstod många bryderier kring kommunikationen mellan processorns portar och register. Frågorna rörde ofta vilka portar skall läsas och vilka man kan skriva till, och hur man åstadkommer detta. Detta rodde vi iland tack vare hjälpsamma tips på olika forum och noggrant studerande av manualen till processorn.

Med själva programvaruutvecklingen till produkten hade vi till en början förhoppningar att kunna använda oss av modifierad kod från befintliga projekt på internet. Detta grusades dock ganska snabbt då vi upptäckte att det upplägg vi tänkt oss knappt fanns med liknande komponenter vi använt oss av. Alltså fick vi göra koden från grunden helt ur eget huvud. Detta visade sig, som tur var, inte vara ett så stort problem som vi trott från början. Vi lyckades ganska snabbt få till ett utkast på en kod som innehöll grundfunktionerna och som uppförde sig ungefär som vi tänkt. Självklart krävdes dock många modifieringar på koden under resans gång.

Resultat

Att ambitionerna för detta projekt var lite i överkant insåg vi redan under de inledande projektveckorna. Eftersom våra kunskaper inledningsvis var bristfälliga stötte vi på flera oförutsatta problem och allt tog längre tid än beräknat. Funktionaliteten för systemet är därför mer begränsad än vad vi tänkt oss från början.

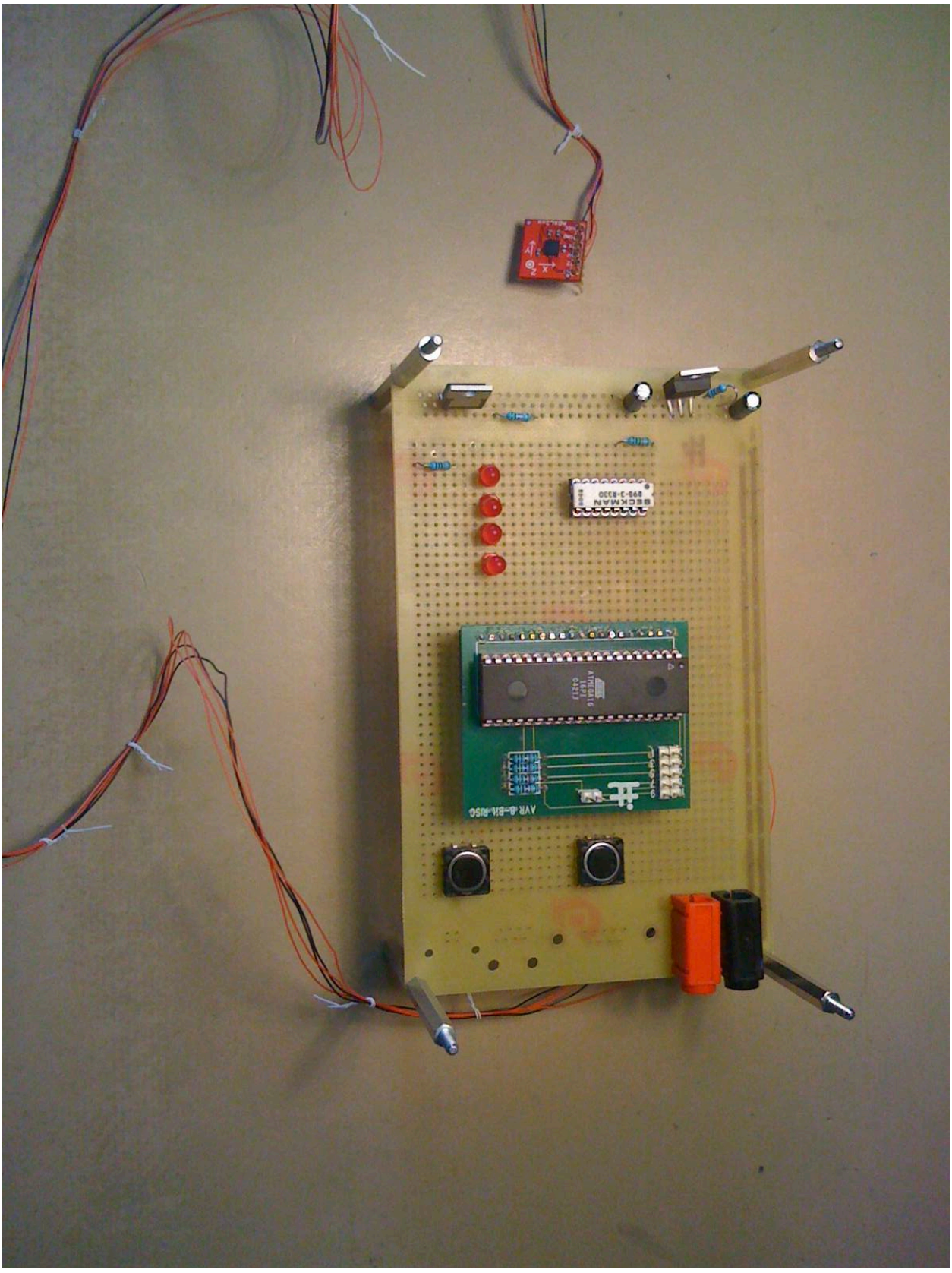
Systemet är fullt kapablet att spela in valfri rörelse. Det finns en speciell inspelningsknapp och en knapp för val av rörelse att spela över. När systemet gått in i inspelningsläge skall accelerometern först hållas stilla för att sedan starta inspelningen när man börjar rörelsen. Inspelningen fortgår sedan i ca en sekund. Detta fungerar mycket väl och det är enkelt att spela in avsedd rörelse. Varje steg under inspelningen bekräftast med tänd lysdiod.

Att känna igen inspelad rörelse fungerar väl för enklare rörelser. Försöker man återskapa en komplicerad rörelse är det svårt att få systemet att reagera. Detta eftersom vi var tvugna att väga generösitet i jämförelsen mellan inspelad rörelse och återskapad, kontra att fel rörelse inte skall kännas igen av misstag. Användaren är tvungen att göra rörelsen lika snabbt som inspelningen. Gör man t.ex. en cirkel vid inspelningen på en sekund känner inte systemet igen denna om användaren gör samma cirkel på två sekunder.

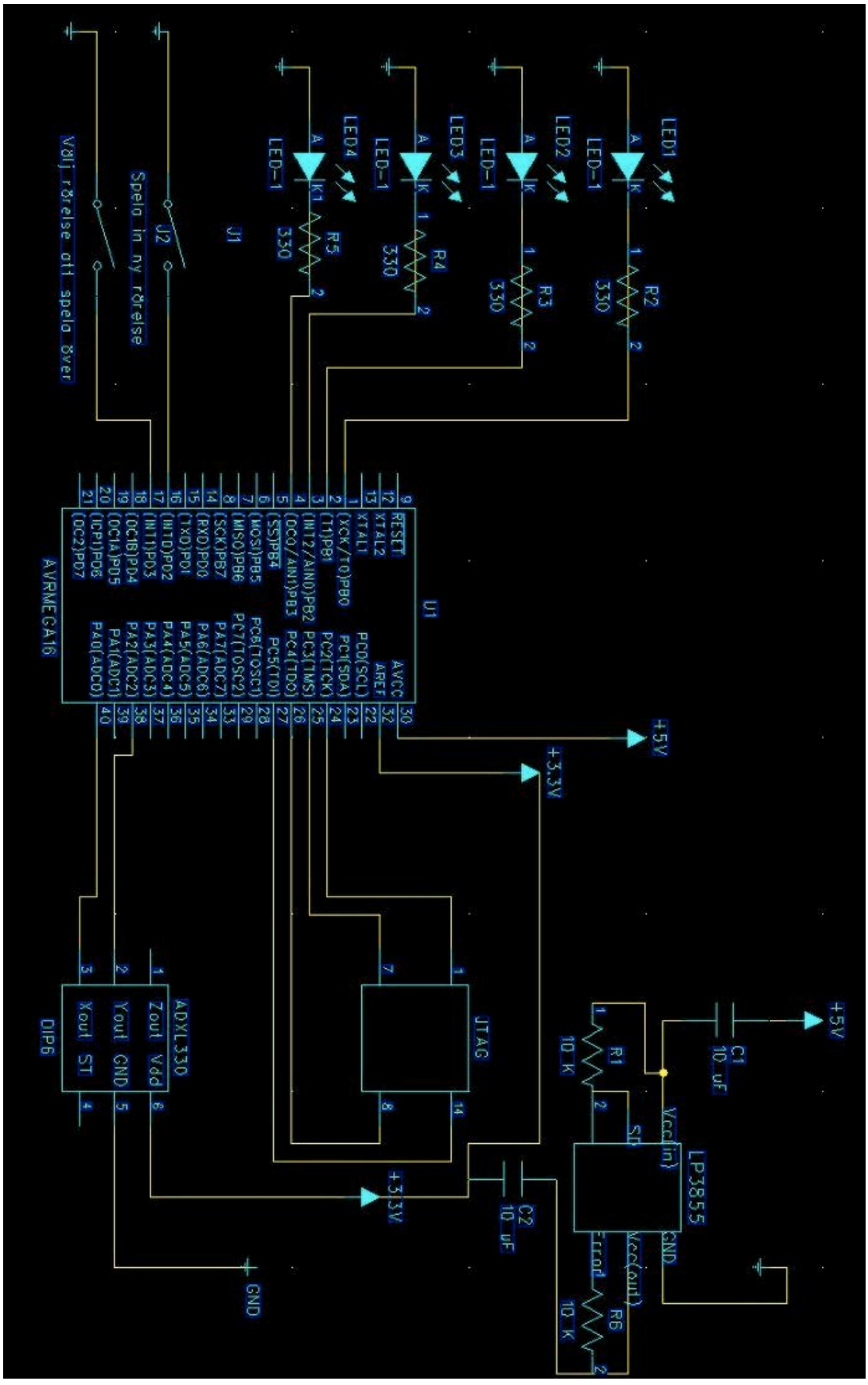
Känner systemet igen en rörelse bekräftas detta genom att motsvarande lysdiod tänds ett par sekunder. Det finns alltså ingen högtalare kopplad till systemet som spelar upp ljud baserat på rörelse som vi tänkte från början. Det fanns inte tid nog att implementera det i systemet.

Systemet mäter bara rörelser i två dimensioner eftersom vi ansåg att det inte var tillräckligt betydande för funktionaliteten att använda tre och därmed öka komplexiteten.

Bilaga 1 - Konstruktionsbild



Bilaga 2 - Kopplingsschema



Bilaga 3 - Kod

[Huvudprogram]

```
#include <avr/io.h>
#include <U:/vadvivill/AVR035.h>
#include <util/delay.h>

int COUNTERA = 0;
int COUNTERB = 0;
int COUNTERC = 0;
int COUNTERD = 0;
int tableSize = 4;
int TABLEA[2][4]; //klarar ej av att ha tableSize variabeln som dimension.
int TABLEB[2][4];
int TABLEC[2][4];
int TABLED[2][4];
int jmfvardeMin = -20;
int jmfvardeMax = 20;
int selectedMove = 1;
int divider = 1000; //Antal omvandlade ACC-v%rden per anrop av getValues().
int count = 0;
long value[2];
int recordCounter = 0;
int tempRecordX;
int tempRecordY;
int oldX = 0;
int oldY = 0;
int count;

void main(void) {

	CLEARBIT(ADMUX,REFS0);
	CLEARBIT(ADMUX,REFS1);
	SETBIT(PORTD, PD2);//L%s input frÂn knapp.
	SETBIT(PORTD, PD3);//L%s input frÂn knapp 2.


	DDRB = 0xFF; //aktivera alla lampor
	PORTB = 0x00;
	DDRD = 0x00;  //Data direction in pÅ hela PORTD (direction in)
	//PORTD = 0x08; //L%s input frÂn knapp.
	int countx = 0;

	for(;;) {

		getValues(); //L%ser in v%rden frÂn accelerometern och sparar i value
		if (PIND == 0xF7) {  //Om knapp1 nedtryckt -> gÅ till record
			record();

		}
		if (PIND == 0xFB) {  //Om knapp2 nedtryckt -> ange vilken r^relse som ska spelas
in.
			selectedMove = selectMove(selectedMove);
			PORTB = 0x00;
		}

		countx = 0;
		if (compareValuesA() == tableSize-1) {
			PORTB = 0x01; //T%nd lampa 1
			COUNTERA = 0;
			_delay_ms(1200);
			PORTB = 0x00;
		}

		if (compareValuesB() == tableSize-1) {
			PORTB = 0x02; //T%nd lampa 2
			COUNTERB = 0;
			_delay_ms(1200);
```

```
                        PORTB = 0x00;

                }
                if (compareValuesC() == tableSize-1) {
                        PORTB = 0x04; //T%nd lampa 3
                        COUNTERC = 0;
                        _delay_ms(1200);
                        PORTB = 0x00;
                }
                if (compareValuesD() == tableSize-1) {
                        PORTB = 0x08; //T%nd lampa 4
                        COUNTERD = 0;
                        _delay_ms(1200);
                        PORTB = 0x00;
                }
        }
}

int compareValuesA (){

        int tempX = TABLEA[0][COUNTERA] - value[0];
        int tempY = TABLEA[1][COUNTERA] - value[1];

        if(tempX > jmfvardeMin && tempX < jmfvardeMax
           && tempY > jmfvardeMin && tempY < jmfvardeMax  )  {
                COUNTERA++;
                return COUNTERA;
        }
        COUNTERA = 0;
        return 0;
}

int compareValuesB () {

        int tempX = TABLEB[0][COUNTERB] - value[0];
        int tempY = TABLEB[1][COUNTERB] - value[1];

        if(tempX > jmfvardeMin && tempX < jmfvardeMax
           && tempY > jmfvardeMin && tempY < jmfvardeMax  ) {
                COUNTERB++;
                return COUNTERB;
        }
        COUNTERB = 0;
        return 0;
}

int compareValuesC (){
        int tempX = TABLEC[0][COUNTERC] - value[0];
        int tempY = TABLEC[1][COUNTERC] - value[1];

        if(tempX > jmfvardeMin && tempX < jmfvardeMax
           && tempY > jmfvardeMin && tempY < jmfvardeMax ) {
                COUNTERC++;
                return COUNTERC;
        }
        COUNTERC = 0;
        return 0;
}

int compareValuesD () {
        int tempX = TABLED[0][COUNTERD] - value[0];
        int tempY = TABLED[1][COUNTERD] - value[1];

        if(tempX > jmfvardeMin && tempX < jmfvardeMax
           && tempY > jmfvardeMin && tempY < jmfvardeMax ) {
                COUNTERD++;
                return COUNTERD;

        }
        COUNTERD = 0;
        return 0;
}
```

```
void record() {

	recordCounter = 0;
	tempRecordX = 0;
	tempRecordY = 0;
	oldX = 0;
	oldY = 0;
	count = 0;

	for(;;) {

		oldX = value[0];
		oldY = value[1];
		getValues();
		tempRecordX = value[0] - oldX;
		tempRecordY = value[1] - oldY;
		PORTB = 0x01;


		if(tempRecordX > -2 && tempRecordX < 2 //Kollar om den ligger still
		   && tempRecordY > -2 && tempRecordY < 2 ) {
			recordCounter++;

		}
		if(recordCounter > 2) {
			PORTB = 0x02;

			if((tempRecordX < -2) || (tempRecordX > 2) ||
			   (tempRecordY < -2) || (tempRecordY > 2) ){
				/*
				 count = 0;
				 PORTB = 0x01;
				 _delay_ms(1000);

				 */
				PORTB = 0x04;
				break;
			}
		}
	}
	for(;;) {

		getValues();
		//PORTB = 0x02;
		if (count == tableSize) {

			PORTB = 0x00;
			return;
		}
		if (selectedMove == 0) {
			TABLEA[0][count] = value[0];
			TABLEA[1][count] = value[1];
			count++;
		}
		if (selectedMove == 1) {
			TABLEB[0][count] = value[0];
			TABLEB[1][count] = value[1];
			count++;
		}
		if (selectedMove == 2) {
			TABLEC[0][count] = value[0];
			TABLEC[1][count] = value[1];
			count++;
		}
		if (selectedMove == 3) {
			TABLED[0][count] = value[0];
			TABLED[1][count] = value[1];
			count++;
		}

	}
```

```
}

void getValues() {
	int countx = 0;
	value[0] = 0;
	value[1] = 0;
	for(;;) {
		value[0] = value[0] + konvX(); //L%ser in v%rden frÅn accelerometern
		value[1] = value[1] + konvY(); //L%ser in v%rden frÅn
accelerometern
		countx++;
		if (countx == divider) {
			value[0] = value[0]/divider;
			value[1] = value[1]/divider;
			break;
		}
	}
	return;
}
```

[Metod som läser in X-värden från Accelerometern]

```
#include <avr/io.h>
#include <U:/vadvivill/AVR035.h>


konvX() {
	//	SETBIT(ADMUX,ADLAR); //Ger 8-bits precision
	SETBIT(ADCSRA,ADEN);  //M^jligg^r A/D-konvertering
	SETBIT(ADCSRA,ADSC); //Startar A/D-konvertering
	int trash;
	if (! CHECKBIT(ADCSRA, ADSC)) {
		CLEARBIT(ADCSRA,ADEN);  //St%nger av A/D-konv.
		CLEARBIT(ADCSRA,ADATE);
		trash = ADC;
	}
	int temp1;
	int temp2;

	SETBIT(ADCSRA,ADEN);  //M^jligg^r A/D-konvertering
	SETBIT(ADCSRA,ADSC); //Startar A/D-konvertering
	int answerX;
	for (;;) {

		if (! CHECKBIT(ADCSRA, ADSC)) {
			CLEARBIT(ADCSRA,ADEN);  //St%nger av A/D-konv.
			CLEARBIT(ADCSRA,ADATE);
			temp1 = ADCL;
			temp2 = ADCH;

			answerX = ((temp2 & 3) << 8) + temp1;
			break;
		}


	}

	return answerX;
}
```

[Metod som läser in Y-värden från Accelerometern]

```
#include <avr/io.h>
#include <U:/vadvivill/AVR035.h>

konvY() {
	SETBIT(ADMUX,MUX1); //L‰ser frÂn ADC0 (Y)
	//	SETBIT(ADMUX,ADLAR); //Ger 8-bits precision
	SETBIT(ADCSRA,ADEN);  //M^jligg^r A/D-konvertering
	SETBIT(ADCSRA,ADSC); //Startar A/D-konvertering
	int trash;
	if (! CHECKBIT(ADCSRA, ADSC)) {
		CLEARBIT(ADCSRA,ADEN);  //St‰nger av A/D-konv.
		CLEARBIT(ADCSRA,ADATE);
		trash = ADC;
	}


	SETBIT(ADCSRA,ADEN);  //M^jligg^r A/D-konvertering
	SETBIT(ADCSRA,ADSC); //Startar A/D-konvertering
	int answerY;
	int temp1;
	int temp2;
	for (;;) {

		if (! CHECKBIT(ADCSRA, ADSC)) {
			CLEARBIT(ADCSRA,ADEN);//St‰nger av A/D-konv.
			CLEARBIT(ADCSRA,ADATE);
			temp1 = ADCL;
			temp2 = ADCH;
			answerY = ((temp2 & 3) << 8) + temp1; //Sparar de mest sign. bitarna i en
variabel.
			break;
		}
	}

	CLEARBIT(ADMUX, MUX1);
	return answerY;
}
```

[Metod som ändrar vald rörelse att spela in]

```
#include <avr/io.h>
#include <U:/vadvivill/AVR035.h>
#include <util/delay.h>

int selectMove(int selectedMove) {

	if(selectedMove == 0) {
		PORTB = 0x02;
		_delay_ms(800);
		return 1;
	}
	if(selectedMove == 1) {
		PORTB = 0x04;
		_delay_ms(800);
		return 2;
	}
	if(selectedMove == 2) {
		PORTB = 0x08;
		_delay_ms(800);
		return 3;
	}
	if(selectedMove == 3) {
		PORTB = 0x01;
		_delay_ms(800);
		return 0;
	} }
```