Digitala Projekt

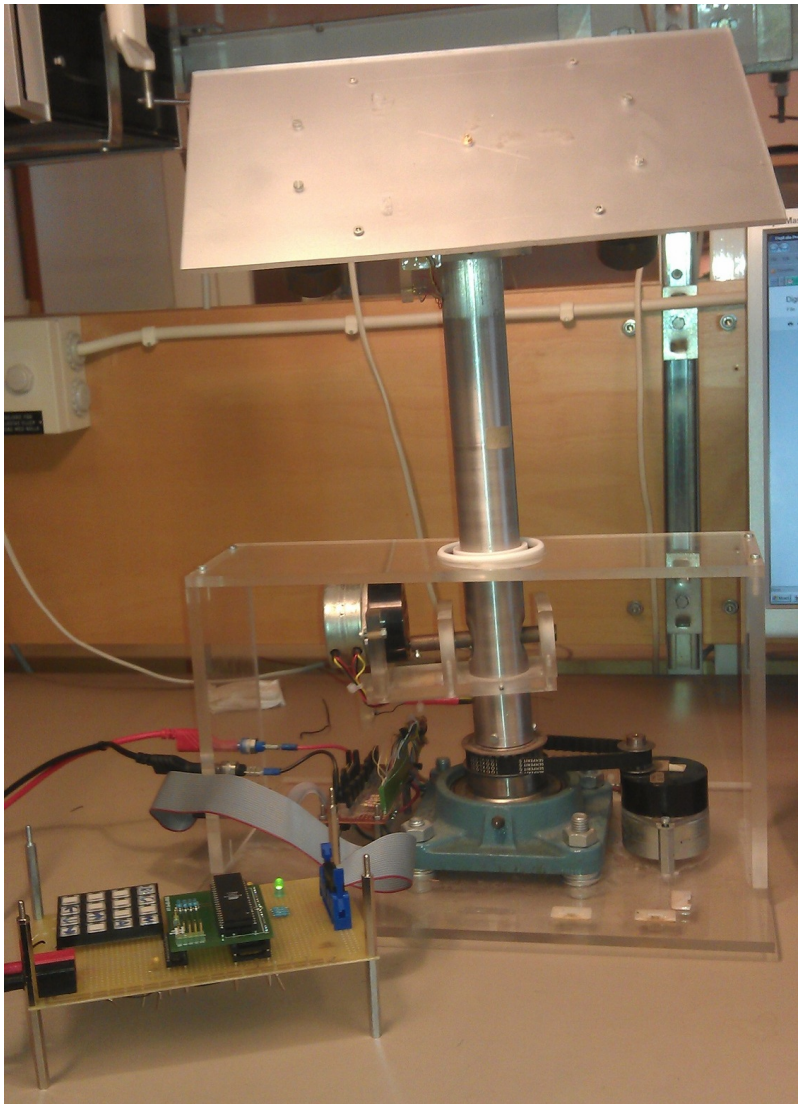
Ljuskänslig riktningsautomatik

av

Rasmus Persson 890510-3571 - dt08rp9@student.lth.se
Max Åkesson 880214-3936 - dt07ma4@student.lth.se

Grupp 6

Handledare:
Bertil Lindvall



1. Abstract

This project resulted in a working prototype for an automatic photosensitive directioning.
It was achieved by writing a requirements specification, drawing a model of the controller unit, constructing the controller unit and developing of software for the controller unit. The automatic photosensitive directioning rotates and tilts to maximize the light on the top plate. It is also possible to calibrate and manually control the apparatus.

2. Innehållsförteckning

3. Inledning

3.1 Syfte

Detta projekt genomfördes i kursen digitala och analoga projekt. Valet av ljuskänslig riktningsautomatik gjordes för att det verkade intressant med en apparat som rörde på sig. Även processen att bygga upp en styrenhet från grunden med endast dess komponenter och sedan programmera den för att kontrollera en annan maskin var intressant. Det skulle även leda till insikt i hur analog till digital konvertering används tillsammans med andra funktioner som finns i processorer. Att genomföra ett projekt på egen hand innebär också att man lär sig vilka delar som ingår i ett projekt och hur man går till väga för att lösa de problem som uppstår. Det ska också syfta till att förbättra rapportskrivandet och presentation av genomfört projekt.

3.2 Mål

Målet med projektet är att få riktningsautomatiken att hitta och rikta in sig mot en ljuskälla. Riktningsautomatiken ska även följa ljuskällan om den förflyttas. Detta uppnås genom att använda en processor som kör ett mindre program. Den ska även kunna kontrolleras och kalibreras via en knappsats. Information till användaren lämnas via lysdioder som kodar för olika lägen.

4. Metod

Först skrivs en kravspecifikation där funktioner och krav fastställs för riktningsautomatiken. Därefter ska styrenheten skissas upp med alla kopplingar utritade. Konstruktionen byggs sedan enligt skissen; lödas, viras och testas så att allting fungerar. När konstruktionen är klar så kopplas den via en AVR JTAG MKII ihop med en dator som används för att skriva koden som processorn ska använda sig av. Efter att koden är komplett och testad så att riktningsautomatiken följer kravspecifikationen så är den redo för redovisning.

4.1 Kravspecifikation

- Riktningsautomatiken ska ställa in sig så att ljusintensiteten är så jämn över de fyra fototransistorerna som möjligt.
- Plattan ska gå att styra med knappsats. Med fyra knappar ska det gå att snurra och tilta.
- Vid ändläge för rotationen ska plattan snurra ett halvt varv tillbaka.
- Systemet ska via räknare hålla reda på positionen i tilt-led.
- Vid för lite ljus på plattana ska systemet gå till utgångsläget.
- Det ska finnas en knapp för att välja mellan automatisk styrning och manuell styrning via knappsats.
- Det ska finnas en återställningsknapp som ställer motorerna i utgångsläget.
- Extra: Det ska finnas en knapp där man kan välja om systemet ska söka efter maximal eller minimal ljusintensitet.

4.2 Hårdvara

Hårdvaran som använts listas nedan med kort beskrivning.

Riktningsautomatik med två stegmotorer, fem fototransistorer och tre ändlägesgivare. På institutionen finns en riktningsautomatik som är uppbyggd kring två stegmotorer vilka kan

rotera och tilta en platta på vilken fem fototransistorer sitter monterade. För att undvika överrotation och -tiltning sitter det tre ändlägesgivare, en för rotationen och två för tiltningen.

ATmega16 är en 8 bitars processor som arbetar i upp till 16 MHz. Den innehåller 16 kB programminne och 1 kB arbetsminne. Den har även åtta A/D-omvandlare och tre räknare.

16-knappars knappsats valdes för att undvika att få för få knappar om en mindre knappsats hade valts.

MM74C922 16-key encoder: för att läsa knappsatsen användes en knappenkoder som ger en fyrabitars utsignal som kodar för en knapp. Den ger även en "data available"-signal då en knapp trycks ner.

Tre stycken lysdioder kommer att användas för att ge feedback till användaren; en röd, en gul och en grön.

För att kommunicera med processorn användes en AVR JTAG MKII som kopplades in till datorn via USB. Den anslöts till processorn via fyra reserverade pinnar. Via JTAG går det att ladda över programmet till processorns programminne, läsa processorns register och stegvis exekvera programmet i felsökningssyfte.

4.3 Mjukvara

Mjukvaran utvecklades i AVR Studio 4 som använde AVR JTAG MKII för att kommunicera med processorn. Där kördes debugging för att stegvis köra koden och se vad som blev fel. Det var också i AVR Studio 4 som koden kompilerades och överfördes till processorns programminne.

Koden är uppbyggd av en main-metod som innehåller en oändlig loop där programmet snurrar runt så länge processorn är på. Vid uppstart utförs en kalibrering då ändläge och mittläge för rotation och tilt fastställs. Därefter hämtas värden från fototransitorerna som jämförs och resulterar i ett förflyttningssteg mot ljuset. Rotationsrörelsen har högst prioritet och när ljusintensiteten är jämn i rotationsled börjar tiltningen för att utjämna skillnader i tiltled.

Ett avbrott genereras då en knapp på knappsatsen trycks ner. I avbrottsrutinen kontrolleras vilken knapp som blev nertryckt och koden kopplad till den knappen körs. Det finns åtta fungerande knappar; fyra funktionsknappar och fyra pilknappar. Den första funktionsknappen är "Reset", som letar upp ett ändläge och ställer riktningsautomatiken i mittläge både för rotation och tiltning. Den andra knappen är "Center" som ställer riktningsautomatiken i mittläge utan att kalibrera mot ändpunkterna. De två sista knapparna används för att växla mellan automatisk och manuell styrning. Pilknapparna används för att rotera och tilta riktningsautomatiken manuellt.

4.4 Konstruktion

Komponenterna monterades på ett kretskort. Drivspänningen och jordsladdarna lödades fast för att säkerställa hållbarheten på kopplingarna. Även motstånd, dioder och kondensatorer lödades fast för att inte trilla av kretskortet. Signalsladdarna virades kring pinnarna med en virpistol enligt tidigare fastställt kopplingsschema. Anslutning till riktningsautomatiken gjordes via en 20-pinnars flatkabel. Matspänningen till kretskortet är 5V och drivspänningen till stegmotorerna är 12V.

5. Resultat

Genom att kolla de olika värdena för fototransistorerna bestäms i vilken riktning ljuset befinner sig. Plattan roterar mot de mest ljusintesiva fototransistorernas position gentemot varandra tills skillnaden ligger inom ett gränsvärde. Då är plattan riktad mot ljuset. På grund av fototransistorernas ojämna värden och känslighet för riktning fungerar programmet endast då ljuset är ganska starkt och inte har för stor vinkel mot plattan.

Eftersom rotationerna görs en åt gången så blir förflyttning i mer än en dimension lite hackig. Den går med andra ord inte i diagonal riktning, utan stegar i de olika riktningarna en åt gången.

Dioderna indikerar vad riktningsautomatiken gör för tillfället. Vid initiering så lyser alla dioderna samtidigt tills den är redo att användas. Vid normal automatisk styrning så lyser grön diod när maximal ljus har hittats, gul då den tiltar och röd då den roterar. Om grön och gul diod lyser samtidigt indikerar det att det är för lite ljus på fototransistorerna. Då manuell styrning används lyser den gröna och den röda dioden. Vid centrering blinkar dioderna i ordningen grön, gul, röd och då återställning utför så blinkar dioderna i ordningen grön, gul, röd, gul, grön.

	Grön	Gul	Röd
Färdig	X	-	-
Tiltar	-	X	-
Roterar	-	-	X
För lite ljus	X	X	-
Manuell styrning	X	-	X
Reserverad	-	X	X
Initiering	X	X	X
Centrering	1	2	3
Återställning	1, 5	2, 4	3

Tabell 1: Förklaring till dioderna.

6. Diskussion

Ett problem som uppstod var att värdena för ljusintensiteten endast varierar i intervallet mellan inget ljus och en övre gräns som bestäms med hjälp av resistanser. Resistanserna valdes så att intervallet för ljusstyrkan ligger kring normal kontorsbelysning. Det gör att om ljusintensiteten blir för låg så blir skillnaderna mellan fototransistorerna för små för A/D-omvandlingen att skilja på. Fototransistorerna är också väldigt rikningskänsliga så ljus som infaller med för stor vinkel registreras inte och leder därmed inte till någon justering.

Det finns utrymme för optimering av den algoritm som används för att bestämma nästa förflyttning av plattan. Det skulle vara möjligt att justera steghastigheten beroende på

skillnaden mellan de olika fototransistorerna. Det går också att justera initieringsförfarandet så att rotation och tiltning utförs samtidigt och på så sätt påskynda initieringen.

Det skulle vara möjligt att räkna ut positionen av ljuset beroende på värdena av fototransistorena, och på så sätt direkt rikta plattan mot ljuset. Det skulle dock antingen betyda att plattan inte är speciellt bra på att hitta ljuskällor som rör på sig eller att beräkningen av positionen måste upprepas med ett kort intervall. Det innebär att det inte skulle funnits någon skillnad alls mot nuvarande funktion, endast att mer beräkningar behöver utföras.

7. Sammanfattning

Detta projekt har resulterat i en prototyp till en ljuskänslig riktningsautomatik. Det har åstadkommits genom att skriva en kravspecifikation, skissa en modell av styrenheten, konstruktion av styrenheten och utveckling av mjukvara som körs på styrenheten.

Appendix - Kod

```
#include <avr/interrupt.h>
#include <avr/io.h>
#define F_CPU 8000000UL
#include <util/delay.h>
#include <stdio.h>
#include <math.h>
#define t 5.0
int halflap = 5;
int rotation = 0;
int tilthalf = 0;
int angle = 0;
int getADC(int pin);
void CWrotate(double d);
void CCWrotate(double d);
void tilta01();
void tilta23();
void CheckEndpoint(int direction);
void reset();
void center();

int main(void)
{
/*
	Initialize pins for input and output
*/
	ADCSRA = 0x80;
	DDRA = 0x70;
	PORTA = 0x70;
	DDRB = 0xff;
	PORTC = 0xff;
	PORTD = 0xff;
	DDRD = 0x00;
	int pt[4];
/*
	Initialize device and find endpoints and middle for rotation.
*/

	double init = 5.0;
	while(!(PIND&0x01)){
		CWrotate(init);
	}
	CCWrotate(init);
	CCWrotate(init);
	CCWrotate(init);
	CCWrotate(init);
	CCWrotate(init);
	halflap = 5;
	while(!(PIND&0x01)){
		CCWrotate(init);
		halflap++;
	}
	halflap = halflap/2;
```

```
        for (int n = halflap; n > 0; n--){
                CWrotate(init);
        }
        rotation = halflap;
/*
        Find endpoints and middle for tilt.
*/
        while(!(PIND&0x02)){
                tilta01();
        }
        while(!(PIND&0x04)){
                tilta23();
                tilthalf++;
        }
        tilthalf = tilthalf/2;
        for (int n = tilthalf; n > 0; n--){
                tilta01();
        }
        angle = tilthalf;

/*
        Main loop for running program in automatic mode.
*/
        int total;
        while(1){
                for (int n = 0; n < 4; n++){
                        pt[n] = getADC(n);
                }
                total = (pt[0]+pt[1]+pt[2]+pt[3])/3;
                if (total < 50){
                        while(rotation < halflap){
                                CWrotate(t);
                                PORTA = 0x30;
                        }
                        while(rotation > halflap){
                                CCWrotate(t);
                                PORTA = 0x30;
                        }
                        while(angle < tilthalf){
                                tilta23();
                                PORTA = 0x30;
                        }
                        while(angle > tilthalf){
                                tilta01();
                                PORTA = 0x30;
                        }
                        PORTA = 0x60;
                } else if (((pt[1]+pt[2]) - (pt[0]+pt[3]) < total) &&
                        (pt[1]+pt[2] - (pt[0]+pt[3]) > -total)){
                        PORTA = 0x20;
                        if ((pt[0]+pt[1] - (pt[2]+pt[3]) < total) &&
                           (pt[0]+pt[1] - (pt[2]+pt[3]) > -total)){
                                PORTA = 0x40;
                        } else if (pt[0]+pt[1] > pt[2]+pt[3]){
```

```
                                        if (!(PIND&0x02)){
                                                tilta01();
                                        } else {
                                                angle = 0;
                                                PORTA = 0x40;
                                        }
                                } else {
                                        if (!(PIND&0x04)){
                                                tilta23();
                                        } else {
                                                angle = tilthalf*2;
                                                PORTA = 0x40;
                                        }
                                }
                        } else if (((pt[1]+pt[2]) > (pt[0]+pt[3])) && (angle > tilthalf)){
                                        PORTA = 0x10;
                                        CheckEndpoint(0);
                                        CCWrotate(t);
                        } else  if (((pt[1]+pt[2]) < (pt[0]+pt[3])) && (angle < tilthalf)){
                                        PORTA = 0x10;
                                        CheckEndpoint(0);
                                        CCWrotate(t);
                        } else {
                                        PORTA = 0x10;
                                        CheckEndpoint(1);
                                        CWrotate(t);
                        }
                }
}

/*
        Get value of A/D-conversion from pin pin
*/
int getADC(int pin)
{
        ADMUX = pin+32;
        ADCSRA = 0xc0;
        while (ADCSRA != 0x90);
        return ADCH*4;
}

/*
        Clockwise rotation
*/
void CWrotate(double d)
{
        rotation++;
        PORTB = 0x60;
        _delay_ms(d);
        PORTB = 0xa0;
        _delay_ms(d);
        PORTB = 0x90;
        _delay_ms(d);
        PORTB = 0x50;
```

```
		_delay_ms(d);
}

/*
		Counter clockwise rotation
*/
void CCWrotate(double d)
{
		rotation--;
		PORTB = 0x50;
		_delay_ms(d);
		PORTB = 0x90;
		_delay_ms(d);
		PORTB = 0xa0;
		_delay_ms(d);
		PORTB = 0x60;
		_delay_ms(d);
}

/*
		Tilt towards the side with phototransistor 0 and 1
*/
void tilta01()
{
		angle--;
		PORTB = 0x06;
		_delay_ms(t);
		PORTB = 0x0a;
		_delay_ms(t);
		PORTB = 0x09;
		_delay_ms(t);
		PORTB = 0x05;
		_delay_ms(t);
}

/*
		Tilt towards the side with phototransistor 2 and 3
*/
void tilta23()
{
		angle++;
		PORTB = 0x05;
		_delay_ms(t);
		PORTB = 0x09;
		_delay_ms(t);
		PORTB = 0x0a;
		_delay_ms(t);
		PORTB = 0x06;
		_delay_ms(t);
}

/*
*		Checks to see if the rotational end point has been reached, and rotates to center
*		position if so.
```

```
*/
void CheckEndpoint(int direction)
{
	int init = 5.0;
	if(PIND&0x01){
		if(direction){
			for(int i = 0; i < halflap + 10; i++){
				CCWrotate(init);
			}
		}else{
			for(int i = 0; i < halflap + 10; i++){
				CWrotate(init);
			}
		}
	}
}

/*
	Interrupt routine. Check which button has been pressed.
*/
ISR(INT1_vect)
{
	int pin = PIND&0xf0;
	switch(pin) {
		case 176:
			center();
			break;
		case 240:
			reset();
			break;
		case 48:
			PORTA = 0x50;
			while(1){
				if ((PIND&0xf0) == 0x20){
					break;
				}
				if ((PIND&0xf0) == 0xb0){
					center();
				}
				if ((PIND&0xf8) == 0xf8){
					reset();
				}
				while((PIND&0xf8) == 0x18){ /* right button*/
					if(rotation < (halflap*2)){
						CWrotate(t);
					}
				}
				while((PIND&0xf8) == 0x98){ /* left button*/
					if(rotation > 0){
						CCWrotate(t);
					}
				}
				while((PIND&0xf8) == 0x48){ /* up button*/
					if (!(PIND&0x04))
```

```
                                        tilta23();
                                }
                                while((PIND&0xf8) == 0x58){ /* down button*/
                                        if (!(PIND&0x02))
                                        tilta01();
                                }
                        }
                default :
                        break;
        }
}

/*
        Resets the positional variables and centers all positions
*/
void reset()
{
        int count = 0;
        byte colors[4] = {0x10, 0x20, 0x40, 0x20};
        while(!(PIND&0x01)){
                CCWrotate(t);
                count++;
                PORTA = colors[(count/3)%4];
        }
        rotation = 0;
        while(rotation < halflap){
                CWrotate(t);
                count++;
                PORTA = colors[(count/3)%4];
        }
        while(!(PIND&0x02)){
                tilta01();
                count++;
                PORTA = colors[(count/3)%4];
        }
        angle = 0;
        while(angle < tilthalf){
                tilta23();
                count++;
                PORTA = colors[(count/3)%4];
        }
        PORTA = 0x40;
}

/*
        Return to center both for tilt and rotation.
*/
void center()
{
        int count = 0;
        byte colors[3] = {0x40, 0x20, 0x10};
        if (rotation < halflap){
                while (rotation < halflap){
                        CWrotate(t);
```

```
                count++;
                PORTA = colors[(count/3)%3];
            }
        } else {
            while (rotation > halflap){
                CCWrotate(t);
                count++;
                PORTA = colors[(count/3)%3];
            }
        }
        if (angle < tilthalf){
            while (angle < tilthalf){
                tilta23();
                count++;
                PORTA = colors[(count/3)%3];
            }
        } else {
            while (angle > tilthalf){
                tilta01();
                count++;
                PORTA = colors[(count/3)%3];
            }
        }
        PORTA = 0x40;
}
```