Wakeup Insurance

EEL4665C

Chris Dobson

Instructors:
A. Antonio Arroyo
Eric M. Schwartz

Teaching Assistants:
Tim Martin
Ryan Stevens

Table of Contents

Abstract

The purpose of Wakeup Insurance is to ensure that its owner wakes up on time. Wakeup Insurance will perform a variety of actions to wake up its owner, and then return to sleep mode and charge its battery.

Executive Summary

This project is designed to ensure that the robot's owner wakes up on time. This is done by waking up its owner with a buzzer and bright LEDS, then forcing them to chase after the robot. After the owner wakes up and disables the robot, the robot will then return to its base station to charge for the next day.

The main platform consists of two levels. The bottom level is larger and circular, to prevent getting caught on unseen obstacles and enable rotating in place. The platform was first designed in solid works and then cut out by hand. It is made out of .5 in plywood for high structural integrity. The top level is six inches above the bottom and square to make it easy to cut. Four threaded steel rods are used to hold the second level up.

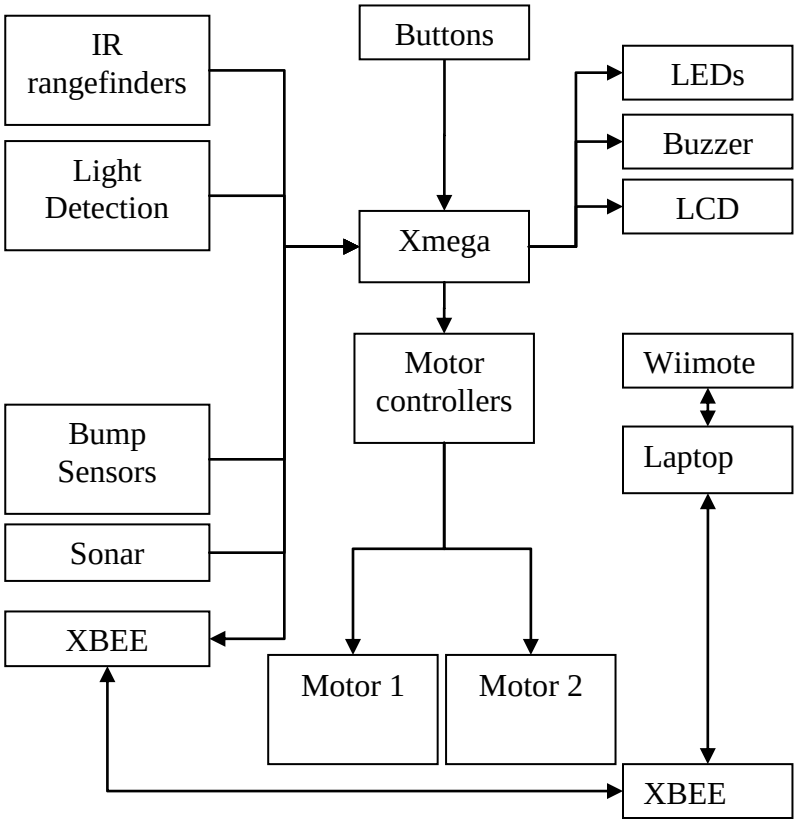
Obstacle avoidance uses three different sensors. A single sonar sensor is mounted on the front of the robot to detect anything in its path. Its wide detection range makes it ideal for this task. An IR range finder is used on either side of the robot to detect obstacles approaching from the sides. Finally, bump sensors are used as a last resort in the case of an unseen obstacle. Bump sensors are mounted on the front, back, right, and left of the robot.

The Special Sensor is a Wii remote, which is used to track the charging station. The Wii remote can detect an IR LED on the top of the charging station. It then communicates wirelessly to a laptop using Bluetooth. The laptop then interpreters this data and sends the results to the robot over XBEE. Using an IR to mark the charging station works well because of the lack of bright IR light sources in the robots operating range. This provides more reliable results than other methods such as color tracking which have many false positives in real world scenarios.

Introduction

This projects main goal is to solve the perpetual problem of waking up on time. A simple alarm clock or two is not enough to ensure that someone wakes up. Wakeup Insurance's goal is to eliminate this problem. The robot will first wake up the owner using a speaker or buzzer. If the room is dark, then bright LEDs will begin flashing. It will then randomly roam around the room, forcing the owner to get up and catch it. After it has been caught, the robot will return to a station to charge its battery for the next day.

Integrated System



The Epiphany DIY board is the base of this robot. It accepts input from the IR range finders, sonar range finder, and the CDS cell. GPIO pins are used to accept input from the buttons and Bump sensors. GPIO pins are also used to control the the LEDs, LCD, and buzzer. The motor driver integrated into the Epiphany board is used to directly power two DC motors.

A Wiimote is used to detect the location of the charging station. The Wiimote uses Bluetooth to communicate wirelessly with a laptop, which then reports back to the Epiphany DIY through XBEE.

Mobile Platform

The platform is a two level design. The top level contains the buttons, LEDs, CDS cell, Wii remote, and LCD screen. Everything else is mounted on the bottom platform.

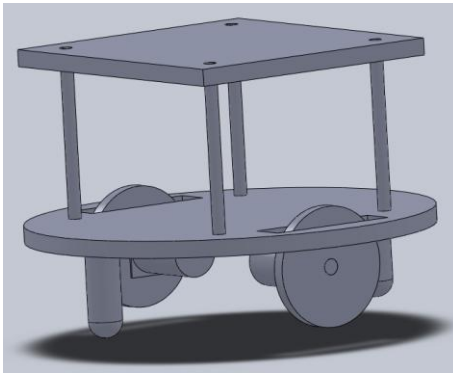


Figure MP.1

The base platform is circular in shape, with 2 cutouts for wheels. The circular shape prevents the robot from getting stuck in case of collisions. It also provides maximum maneuverability, enabling the robot to rotate in place. Half inch thick plywood was the material chosen to for both the bottom and top level. It provides exceptional strength, protecting the robot from taking damage in collisions. Threaded steel rods with nuts are used to hold up the top level. The steel rods provide more than enough structural strength to hold up the top level while using minimal space and providing some adjustability. The two casters are slightly shorter than the wheels, to ensure that the driven wheels always make contact with the ground.

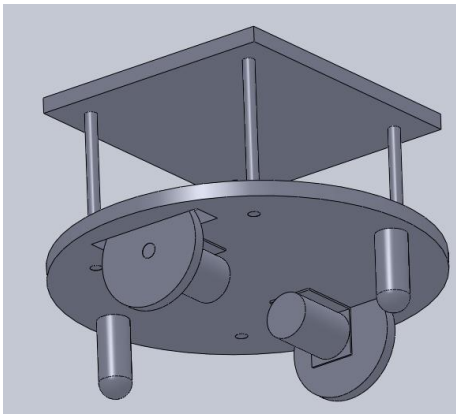


Figure MP.2

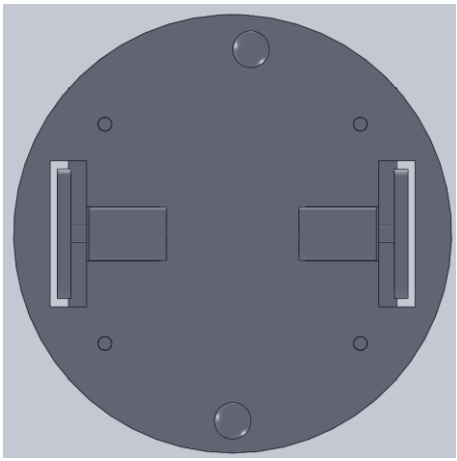


Figure MP.3

The top level also contains two springs (one above and one below the top level) used to make electrical contact with the base station. These springs are connected to a CPU controlled charging circuit to charge the battery.

Actuation

The platform will be driven by two motors attached to wheels near the center of the robot. The Motors have a torque of 200 oz·in, a max free run speed of 150 rpm, and a stall current of 5A. The motors have proven to provide more than enough torque to accelerate the robot. The top speed is also fast enough to force the owner to pay attention to the catch the robot and shut it off.

Sensors

There are five different sensors to be used.

Sharp GP2Y0A21YK IR range finding sensors are used to detect obstacles on the left and right sides of the robot. They are also part of basic obstacle avoidance. The IR sensor was placed on the platform at the height it will be mounted (3.5 inches from the ground). Two objects (a white piece of paper and a brown piece of cardboard) were each placed progressively farther from the sensor in one inch intervals. The voltage was

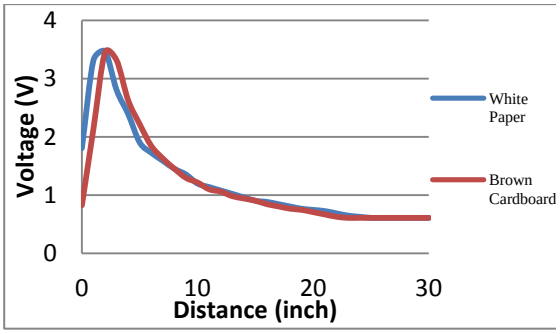


Figure S.1: Sharp IR Rangefinder

measured by the Epiphany DIY board and the hex result was printed out to the LCD. The hex values were then converted to voltages and the results are in figure S.1.

The Maxbotix LV-EZ0 ultrasonic range finder is used to detect obstacles directly in front of the platform. The ultrasonic range finder has a much greater detection width than the IR sensors, making it ideal for finding objects over a larger area. The ultrasonic sensor is able to detect even small objects anywhere in its range that the IR rangefinders were not able to see, making it the perfect solution.

Bump sensors (figure S.2) are used to report collisions with objects not detected by the IR sensors or the ultrasonic range finder. One bump sensor is mounted on the front and back, and one is mounted on each side.

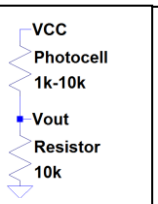


Figure S.3


Figure S.2: Bump Sensor

An ambient light sensor is used to detect whether the room is dark enough to use the LEDs to assist waking up the owner. The light sensor is mounted on the top platform. The circuit for this sensor is shown in figure S.3.

The Wiimote is used to detect the location of the base station. The base station has two IR LEDs, one above the other. The distance between these two points on the camera can be used to provide the distance to the base station. The Wiimote itself be located on the top platform.

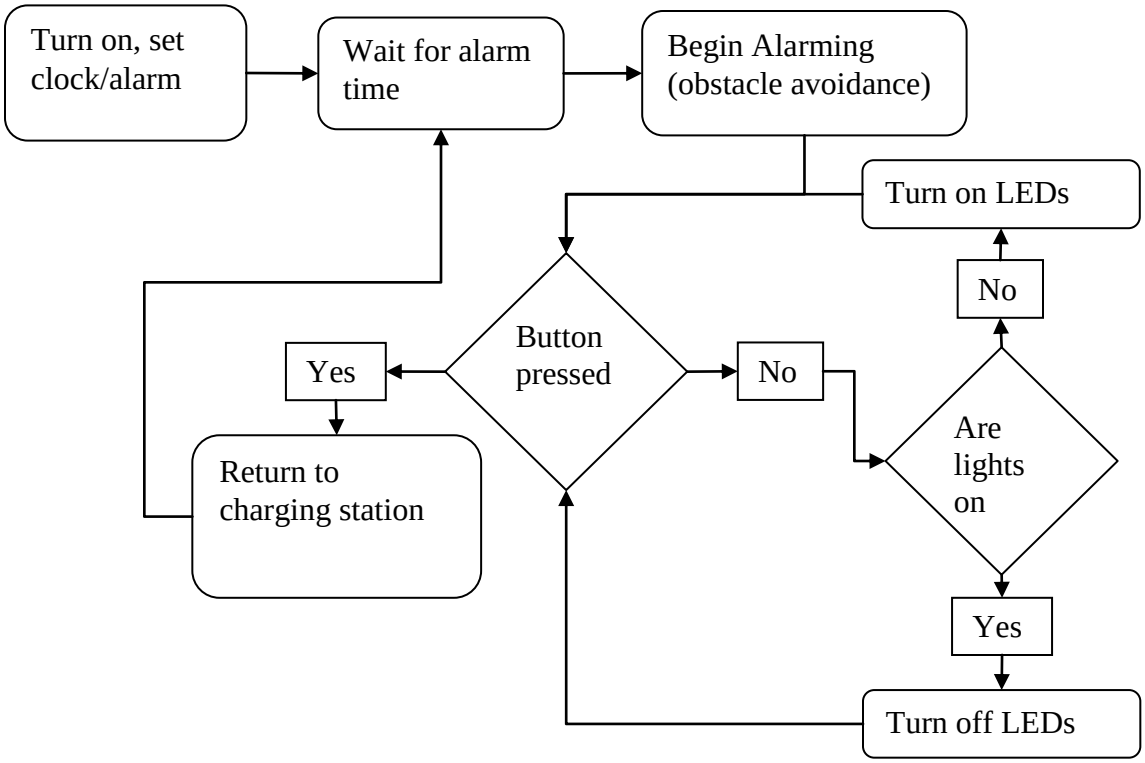
The Wiimote is connected to a computer using Bluetooth. A program called GlovePIE is used to interface with the Wiimote and output the camera information to a virtual joystick, the only output it supports. An additional program is then used to read the joystick data and send it to the microprocessor board over XBee

The range of the Wiimote was measured using a single LED as a worst case scenario. The LED provided a consistent reading up to 10 feet away from the LED. Multiple LED's or brighter LEDs could be used to further increase the detection range, but are not necessary in this instance.

The Output from the Wiimote was very consistent and clean, with virtually no jitter.

Behaviors

The first behavior is waiting mode. The robot waits and charge its battery until it is time to ring alarm. The next mode is alarm mode. The alarm will sound, the LED's turn on if it's dark enough, and obstacle avoidance will begin. This lasts until the owner presses a button on the robot. The next mode is returning to base, so it can enter waiting mode and recharge.



Experimental Layout and Results

The experiment provided a simulation of the robots intended purpose. The robot and its charging station were placed in an enclosed space (to simulate a room). The robot was then programmed to go off at a certain time. At the time, the robot began alarming and obstacle avoiding. After a minute or so of successfully moving randomly without hitting any obstacles, the large red button on the top of the robot was hit, causing it to seek its base station.

The robot then began randomly moving with its lights and buzzer turn off. While moving randomly, the Wii remote detected the base station. At that point the robot returned to the station, began charging, and waited for the next alarm time.

Conclusion

The most difficult part of the project was tracking the base station. Initially, the tracking method was undetermined and several different methods were tried. Several different arrays of photo detectors were initially experimented with, but a severe lack of precision prevented them

from being the used. The camera module was then removed from a Wii remote to be interfaced directly with the main board, but that also resulted in failure. The current solution was then determined and successfully implemented.

Another unforeseen problem was in obstacle avoidance. Initially, only IR range finders were to be used. Several crossing patterns and configurations were attempted, but the robot was too large for any of them to work. A sonar module was then ordered and worked perfectly in conjunction with the existing IR sensors.

After all the difficulties, the final product is a robot that accurately performs the tasks it was designed to do. The

Documentation

Epiphany DIY home page: https://sites.google.com/site/epiphanydiy/home

-Header files are located here

Epiphany DIY Google group: http://groups.google.com/group/epiphany-diy?hl=en

IMDL home page: http://mil.ufl.edu/5666

Code Appendix

ROBOT CODE:

```
/**
 * \file
 *
 * \brief Empty user application template
 *
 */

/*
 * Include header files for all drivers that have been imported from
 * AVR Software Framework (ASF).
 */
#include <asf.h>
#include <avr/io.h>
#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
#include <util/delay.h>
#include "motor.h"
#include "lcd.h"
#include "uart.h"

#define DbLedOn()           (PORTR.OUTCLR = 0x02)             //Turns the debug led on.
The led is connected with inverted logic
#define DbLedOff()          (PORTR.OUTSET = 0x02)             //Turns the debug led off.
The led is connected with inverted logic
#define DbLedToggle()       (PORTR.OUTTGL = 0x02)             //Toggles the debug led
off.  The led is connected with inverted logic
#define speedon                     0x300
#define superslow           0x230
#define speedturn           0x250
#define speedturnfast       0x270
```

```
#define speedoff            0
#define IRTHRESHOLD         70
#define SONARTHRESHOLD      16

void ADC_INTT(void); //this function initializes the ADC for single run mode
void ADC_Start_A3456(void); //this starts a ADC single run (only on ADC A though at the
moment)
unsigned int ADC_Returnn(char ADCLetter, char ChannelNumber, char PinNumber);      //this
function returns a result if there is one, and an error code if there is not
void Obstacle_Avoid(Bool* Obstacle_Avoidance_Turning);




int main (void)
{

       PORTF.DIRSET = 0x80;                                                  //charging control
       PORTF.OUTCLR = 0x80;                                                  //make sure it
starts out off


       PORTC.DIRSET = 0xC3;                                                  //configure the
output pins for speaker and LEDS
       PORTC.OUTCLR = 0xC3;                                                  //make sure they
start out off



       board_init(); /*This function originates in the file init.c, and is used to
initialize the Epiphany DIY
                                    motorInit() is declared within because by default you
the user should define what your
                                    motor setup is to prevent hurting the Epiphany.  You
can do this by
                                  */
       //RTC_DelayInit();//initializes the Real time clock this seems to actually take an
appreciable amount of time
       //servoControlInit();//initializes the servo module ***including enabling global
interrupts*** required for the servo control module
       //uartInit(&USARTC0,57600);//as can be seen in the schematic.  This uart is
connected to the USB port.  This function initializes this uart
       uartInit(&USARTE1,9600);//this should be the XBEE UART

       //stdout = &USB_str;//This function points stdout to the USB device.  So this
means printf will send data out the USB port.
       stdout = &lcd_str;
       //stdout = &Xbee_str;

       motorInit();
       //you should add any further initializations here
       LCDInit();
```

```
	ADCA.CTRLA =   0b00000001;        //bit 0 enables the ADC, 5-2 will later be used
to start a single conversion
	ADCA.CTRLB =   0b00000100;        //bits 1-2 set the resolution , and 10 is 8 bit.
(its easier to use for now) (table 25-2 p 303)
	ADCA.REFCTRL = 0b00010000;        //bits 4-5 control the ADC reference ( 01 is
internal Vcc/1.6, table 25-3 p304), bits 1 and 0 enable bandgap and temp inputs (off for
now)
	ADCA.PRESCALER = 0b00000011;      //bits 0-2 control the prescaler, set it to 011
for now (32), because opamps are limited at 1MHz (table 25-7 p 306)



	Bool Obstacle_Avoidance_Turning = 0;
	char Behavior = 2;          //0 = alarming, 1 = finding base station, 2 =
charging/waiting for action
	Bool first = true;
	int LEDTIMER = 0;

	Bool random_turning = 0;
	int    random_turning_timer = 0;
	int random_turning_alarm = 50;



			setMotorDuty(1,speedoff, MOTOR_DIR_NEUTRAL_gc);
			setMotorDuty(3,speedoff, MOTOR_DIR_NEUTRAL_gc);

	TCC0.PER = 3125;     //set the timer period to 1 second (default count up mode)
	TCC0.CTRLA = 7;             //set the timer prescaler to 1024 (this starts the
timer automatically)
	ADC_Start_A3456();

	while (1)
	{
		while (Behavior == 0)
		{
			if (TCC0.INTFLAGS & 0x01)
			{
				TCC0.INTFLAGS = 0x01;
				PORTC.OUTSET = 0x01;
				if (first)
				{
					LCDCommand(1);
					printf("Catch ME");
					first = false;
					setMotorDuty(1,speedon, MOTOR_DIR_BACKWARD_gc);
					setMotorDuty(3,speedon, MOTOR_DIR_BACKWARD_gc);
					_delay_ms(500);
				}
				if((LEDTIMER / 3) % 2)
					PORTC.OUTSET = 0x02;
				else
					PORTC.OUTCLR = 0x02;

				if((LEDTIMER / 4) % 2)
```

```
                        PORTC.OUTSET = 0x40;
                else
                        PORTC.OUTCLR = 0x40;

                if((LEDTIMER / 5) % 2)
                        PORTC.OUTSET = 0x80;
                else
                        PORTC.OUTCLR = 0x80;

                LEDTIMER++;

                Obstacle_Avoid(&Obstacle_Avoidance_Turning);
                if((PORTE.IN & 0x01) != 0)
                {
                        Behavior = 1;
                        first = true;
                        PORTC.OUTCLR = 0xC3;
                        setMotorDuty(1,speedoff, MOTOR_DIR_NEUTRAL_gc);
                        setMotorDuty(3,speedoff, MOTOR_DIR_NEUTRAL_gc);
                }
                DbLedToggle();

        }
}

while (Behavior == 1)
{
        if (TCC0.INTFLAGS & 0x01)
        {
                TCC0.INTFLAGS = 0x01;

                if (first && USARTE1.DATA != 0)
                {
                        LCDCommand(1);
                        printf("Looking for home");
                        first = false;
                }

                if (USARTE1.DATA == 0)
                {
                        LCDCommand(1);
                        printf ("WAIT FOR PC");
                        setMotorDuty(1,speedon, MOTOR_DIR_NEUTRAL_gc);
                        setMotorDuty(3,speedon, MOTOR_DIR_NEUTRAL_gc);
                }
                else if(USARTE1.DATA == 0x30)
                {
                        random_turning_timer++;

                        if (random_turning_timer >= random_turning_alarm)
                        {
                                if (random_turning == 0)
                                {
                                        random_turning = 1;
                                        random_turning_timer = 0;
                                        random_turning_alarm = rand() % 41 + 30;
                                        if (random_turning_alarm % 2 == 0)
                                        {
```

```
                                                setMotorDuty(1,speedturnfast,
MOTOR_DIR_FORWARD_gc);
                                                setMotorDuty(3,speedturnfast,
MOTOR_DIR_BACKWARD_gc);
                                        }
                                        if (random_turning_alarm % 2 != 0)
                                        {
                                                setMotorDuty(1,speedturnfast,
MOTOR_DIR_BACKWARD_gc);
                                                setMotorDuty(3,speedturnfast,
MOTOR_DIR_FORWARD_gc);
                                        }


                                }
                                else if (random_turning != 0)
                                {
                                        random_turning = 0;
                                        random_turning_timer = 0;
                                        random_turning_alarm = rand() % 26 + 25;
                                }
                        }

                        if (random_turning == 0)
                                Obstacle_Avoid(&Obstacle_Avoidance_Turning);

                }
                else if(USARTE1.DATA == 0x31)
                {
                        random_turning = 0;
                        setMotorDuty(1,superslow, MOTOR_DIR_FORWARD_gc);
                        setMotorDuty(3,superslow, MOTOR_DIR_BACKWARD_gc);
                }
                else if(USARTE1.DATA == 0x32)
                {
                        random_turning = 0;
                        setMotorDuty(1,superslow, MOTOR_DIR_BACKWARD_gc);
                        setMotorDuty(3,superslow, MOTOR_DIR_FORWARD_gc);
                }
                else if(USARTE1.DATA == 0x33 && ADCA.CH2.RESL > SONARTHRESHOLD
&& !(PORTD.IN & 1))
                {
                        random_turning = 0;
                        setMotorDuty(1,speedturn, MOTOR_DIR_FORWARD_gc);
                        setMotorDuty(3,speedturn, MOTOR_DIR_FORWARD_gc);
                }
                else if(USARTE1.DATA == 0x33 && ADCA.CH2.RESL <=
SONARTHRESHOLD && !(PORTD.IN & 1))
                {
                        random_turning = 0;
                        setMotorDuty(1,superslow, MOTOR_DIR_FORWARD_gc);
                        setMotorDuty(3,superslow, MOTOR_DIR_FORWARD_gc);
                }
                else if(USARTE1.DATA == 0x33 && PORTD.IN & 1)
                {
                        random_turning = 0;
                        setMotorDuty(1,speedoff, MOTOR_DIR_NEUTRAL_gc);
```

```
                                setMotorDuty(3,speedoff, MOTOR_DIR_NEUTRAL_gc);
                                Behavior = 2;
                                first = true;
                        }
                        DbLedToggle();
                }
        }

        while (Behavior == 2)
        {
                if (TCC0.INTFLAGS & 0x01)
                {
                        if (first)
                        {
                                LCDCommand(1);
                                printf("Charging");
                                first = false;
                                PORTF.OUTSET = 0x80;
                        }

                        TCC0.INTFLAGS = 0x01;
                        if (USARTE1.DATA == 0x34  || PORTE.IN & 0x01 != 0)
                        {
                                Behavior = 0;
                                PORTF.OUTCLR = 0x80;
                                first = true;
                        }
                }
        }
    }
}


void ADC_Start_A3456(void)  //this starts a ADC single run (only on ADC A though at the
moment)
{

    ADCA.CH0.INTFLAGS = 1;
    //setting a 1 clears the int flag (which is bit 1)
    ADCA.CH0.MUXCTRL = 3<<3;                                                    //
bits 0-1 control input mode (set to 01 for single ended positive input signal)
    ADCA.CH0.CTRL = 0b10000001;                                             //set bit 7
to start conversion on the channel,bits 3-6 control the input source (0-7 = pin 0-7); it
is different in internal input mode

    ADCA.CH1.INTFLAGS = 1;
    //setting a 1 clears the int flag (which is bit 1)
    ADCA.CH1.MUXCTRL = 4<<3;                                                    //
bits 0-1 control input mode (set to 01 for single ended positive input signal)
    ADCA.CH1.CTRL = 0b10000001;                                             //set bit 7
to start conversion on the channel,bits 3-6 control the input source (0-7 = pin 0-7); it
is different in internal input mode

    ADCA.CH2.INTFLAGS = 1;
    //setting a 1 clears the int flag (which is bit 1)
    ADCA.CH2.MUXCTRL = 5<<3;                                                    //
bits 0-1 control input mode (set to 01 for single ended positive input signal)
```

```
	ADCA.CH2.CTRL = 0b10000001;                                          //set bit 7
to start conversion on the channel,bits 3-6 control the input source (0-7 = pin 0-7); it
is different in internal input mode

	ADCA.CH3.INTFLAGS = 1;
	//setting a 1 clears the int flag (which is bit 1)
	ADCA.CH3.MUXCTRL = 6<<3;                                                    //
bits 0-1 control input mode (set to 01 for single ended positive input signal)
	ADCA.CH3.CTRL = 0b10000001;                                          //set bit 7
to start conversion on the channel,bits 3-6 control the input source (0-7 = pin 0-7); it
is different in internal input mode
}

unsigned int ADC_Returnn(char ADCLetter, char ChannelNumber, char PinNumber)	//this
function returns a result if there is one, and an error code if there is not
{
	if (ADCA.INTFLAGS == 1)
	{
		ADCA.INTFLAGS = 1;
		return ADCA.CH0.RESL;
	}
	else { return 0xffff;}
}

void Obstacle_Avoid(Bool* Obstacle_Avoidance_Turning)
{
	//LCDCommand(1);
	//printf ("%X - %X - %X\n%X - %X - %X", ADCA.CH1.RESL, ADCA.CH0.RESL,
USARTE1.DATA, ADCA.CH2.RESL, PORTD.IN, ADCA.CH3.RESL);
	if(ADCA.CH1.RESL <= IRTHRESHOLD && ADCA.CH0.RESL <= IRTHRESHOLD && ADCA.CH2.RESL >
SONARTHRESHOLD)
	{
		*Obstacle_Avoidance_Turning = false;
		setMotorDuty(1,speedon, MOTOR_DIR_FORWARD_gc);
		setMotorDuty(3,speedon, MOTOR_DIR_FORWARD_gc);
	}
	else if((ADCA.CH1.RESL > IRTHRESHOLD && ADCA.CH0.RESL > IRTHRESHOLD) ||
ADCA.CH2.RESL <= SONARTHRESHOLD)
	{
		*Obstacle_Avoidance_Turning = true;
		if( ADCA.CH1.RESL > ADCA.CH0.RESL)
		{
		setMotorDuty(1,speedturn, MOTOR_DIR_BACKWARD_gc);
		setMotorDuty(3,speedturn, MOTOR_DIR_FORWARD_gc);
		}
		else
		{
		setMotorDuty(1,speedturn, MOTOR_DIR_FORWARD_gc);
		setMotorDuty(3,speedturn, MOTOR_DIR_BACKWARD_gc);
		}
	}
	else if(ADCA.CH1.RESL <= IRTHRESHOLD && ADCA.CH0.RESL > IRTHRESHOLD &&
*Obstacle_Avoidance_Turning == false)
	{
		*Obstacle_Avoidance_Turning = true;
		setMotorDuty(1,speedturn, MOTOR_DIR_FORWARD_gc);
		setMotorDuty(3,speedturn, MOTOR_DIR_BACKWARD_gc);
	}
```

```
	else if(ADCA.CH1.RESL > IRTHRESHOLD && ADCA.CH0.RESL <= IRTHRESHOLD &&
*Obstacle_Avoidance_Turning == false)
	{
		*Obstacle_Avoidance_Turning = true;
		setMotorDuty(1,speedturn, MOTOR_DIR_BACKWARD_gc);
		setMotorDuty(3,speedturn, MOTOR_DIR_FORWARD_gc);
	}
	ADC_Start_A3456();
}
```