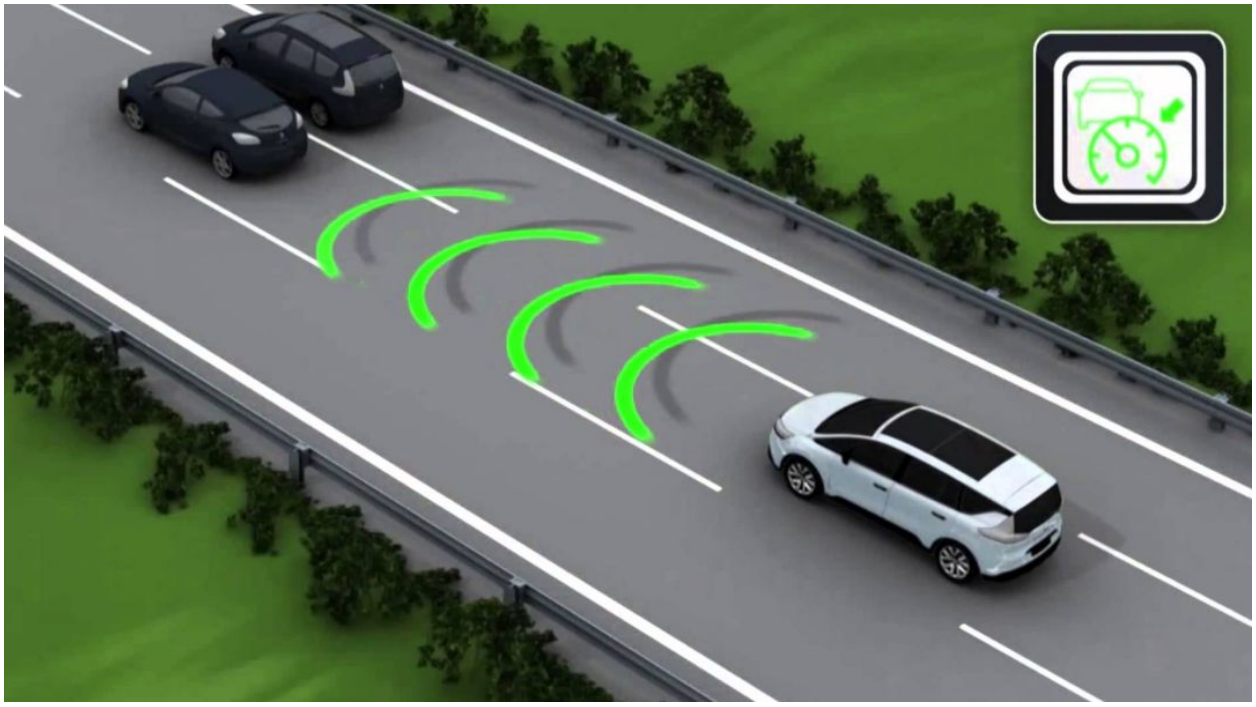
Adaptive Cruise Control Project



12/16/2019
Embedded Systems
Final Project Report
Olivia Ferda, Jiasen Zhou, Tomas Colorado

Project Summary

Similar to the adaptive cruise control feature found in a car, this robot moves forwards or backwards depending on how far away the object in front is. Using an ultrasonic sensor, the robot is able to detect the distance of objects that may lie ahead of it. Implementing PWM in Atmel SAM4L8 board, the motors underneath the robot will speed up or slow down according to what the sensor reads.

Project Objectives

1. Independent: The robot is meant to be independent; does not require power from outlets but from portable batteries

2. Adaptable: The robot is able to determine what the appropriate action is to the object in front is without help

Selected Hardware Platform

The control system of the car is built with one Atmel board, 3 9V-DC motors, 2 ultrasonic sensors and a protoboard with 2 TB6612 motor drivers. The motors are powered by a 9V battery and the Atmel board is powered by a 5V 1A battery.

Since the car needs to start working when the is turned on and shut down immediately, the Atmel board is the best choice. It also contains 5V output which can power the sensors directly. Even the Atmel cannot set the pwm directly, it also can be done by building a TC and TC_handler.

Sensors range meet the design request and it can send and detect thousands of times in 1 seconds, which makes the car changes as soon as the distance changes.

9V DC motors some with the robot kit. By calculation and testing the speed of motor, it meets the design requirements.

The TB6612 Motor Driver has two motor outputs, which means we can control 3 dc motor by only 2 drivers.

Hardware Design Description

Figure 1 belows shows the hardware design of the control system

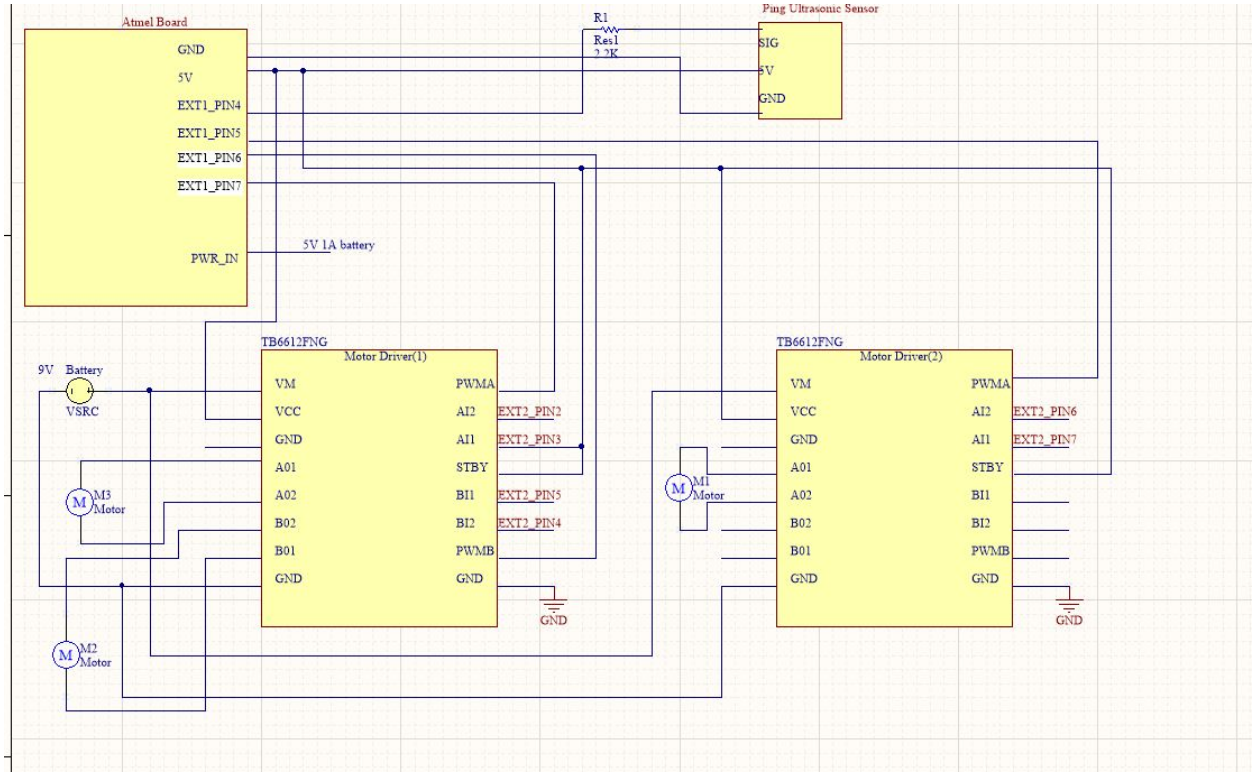
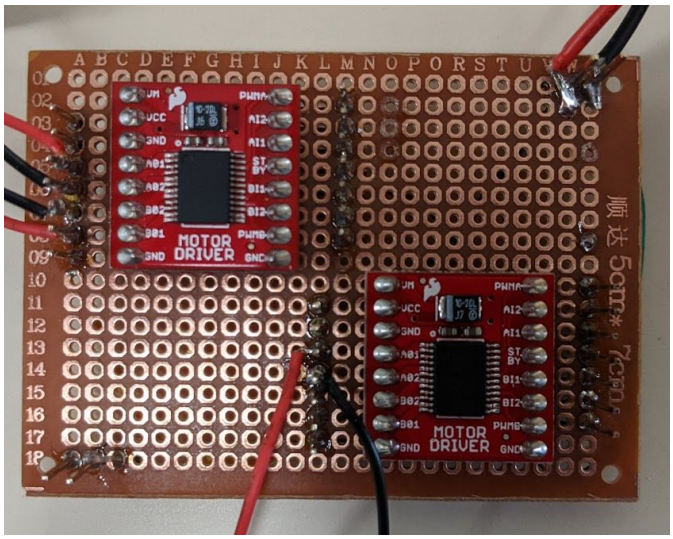


Figure 1. Hardware design schematic

The Atmel board and motor drivers are both powered by a 5V battery. The sensor is powered by the power pin on the Atmel board and pin EXT1_PIN4 is used for pulling the trigger and receiving the signal. PIN EXT1_PIN5, EXT1_PIN5, EXT1_PIN7 are connected to the pwm pins

on motor drivers to control the motors speed. PIN A01,A02,B01,B02 on driver boards are outputs that connect to the motors. PIN AI1, AI2, BI1, BI2 on driver boards are the inputs that used to control the orientation of the motors. STBY pin is the switch that enables or disables the H-bridge. PWM pins are inputs that get data from the Atmel board. VM is the power for motors. VCC is the power for the motor driver.
The ultrasonic sensor used on the car will automatically send a 40 KHz burst when the signal pin (trigger) is configured as an output. The signal pin is set low to high to low in order to send out the pulse. Then the same signal pin (echo) is set as an input for 40 us to receive the pulse. The timing from the trigger to the echo status is used for calculating the distance that the sensor is from an object.

Figure 2&3 belows shows the hardware circuit, every GND pin is connected. Since VCC and STBY pins use the same level of the voltage, we connected them together. And VM and GND are connected to the battery pin which is at the top right of the board (Figure 2).



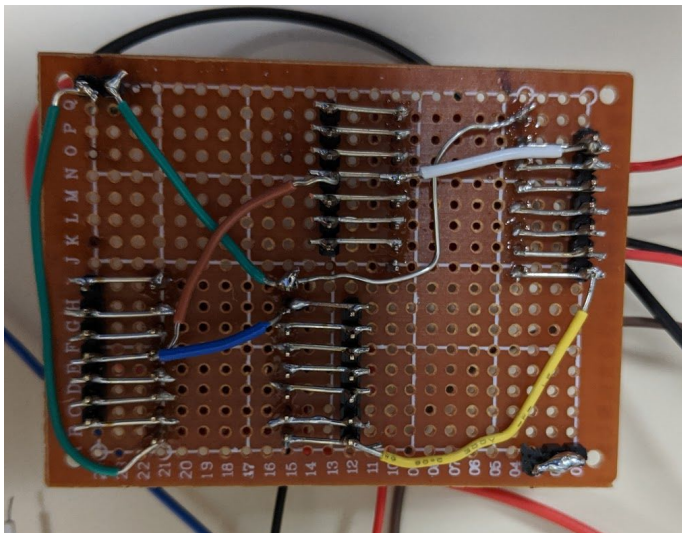


Figure 2&3 (respectively). Hardware Circuit

Chosen Software Platform
The chosen software platform was C in Atmel Studio which works best with the Atmel board. C worked best for the project design due to its speed, efficiency, and was used throughout most of the course.

Functions such as PWM function in utility.h was provided to help aid in the duty cycle calculations. C enabled us to create key functions and to be able to call them in main when needed.

Additionally, Putty console was important in the troubleshooting stages and printed out values at key locations in the code. This also helped determine whether an error was due to hardware or software when combining both components.

Software Design

Overall

Figure 4 is the block diagram which depicts the Ping ultrasonic sensor as the input to the Atmel board which is powered by a 5V power supply. The outputs are the three motors which are powered by a 9V battery and respond accordingly based on what the input is.

Figure 5 and Figure 6 displays the high-level diagram of the software design for the sensor and motors, respectively. In Figure 5, all of the variables for the sensor and motor are initialized and the appropriate functions are called. In the while(1) loop, there is another while loop (while n<5) which sends out a pulse and then receives the echo in order to calculate the start and stop time difference. This difference is then used to calculate the distance in centimeters using a calculation with the speed of sound. After the 5 distance values are captured, they are then averaged outside of the loop. The reasoning behind averaging five distance readings is so that the moves do not move abruptly if the sensor reads 20cm and then 300cm in the next pulse.

Figure 6 depicts the motor movement and speed based off of the average value calculated. For instance, if the average is less than 20cm, then one motor moves counter clockwise and then the other motor will move clockwise so that the motor movement matches. If the average is less than 50cm then both motors are at a 0% duty cycle. This is so that the robot is not constantly going forwards and back much like cruise control features. For ranges between 100cm and 350cm, the motors move in opposite directions of each other with increasing duty cycles the further that they are from an object.

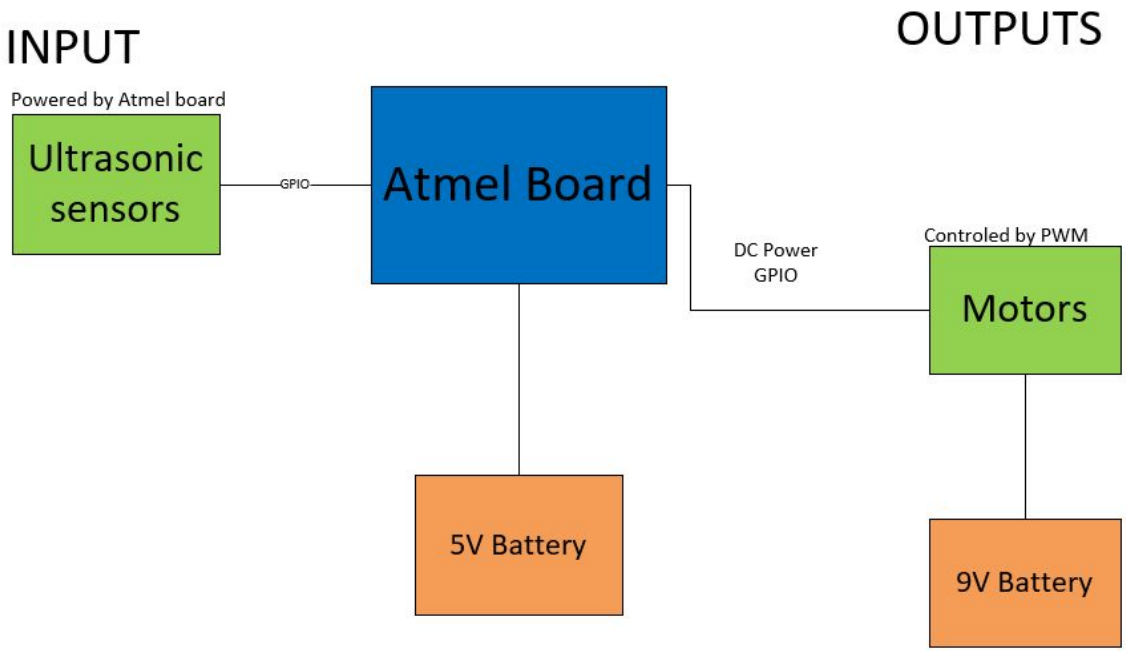

INPUT
OUTPUTS
Powered by Atmel board
Ultrasonic sensors
GPIO
Atmel Board
DC Power
GPIO
Controled by PWM
Motors
5V Battery
9V Battery

Figure 4

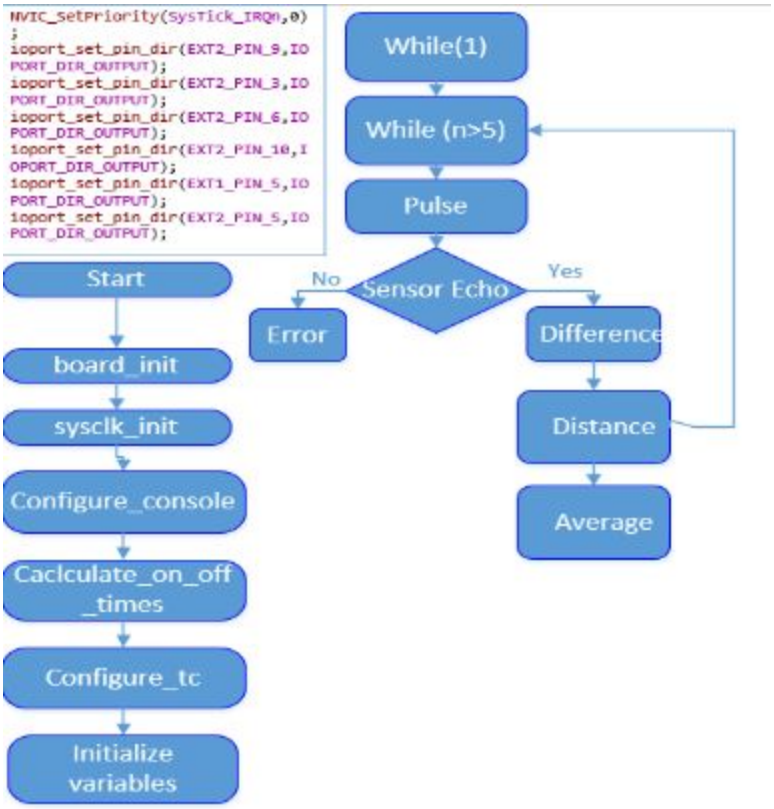


Figure 5

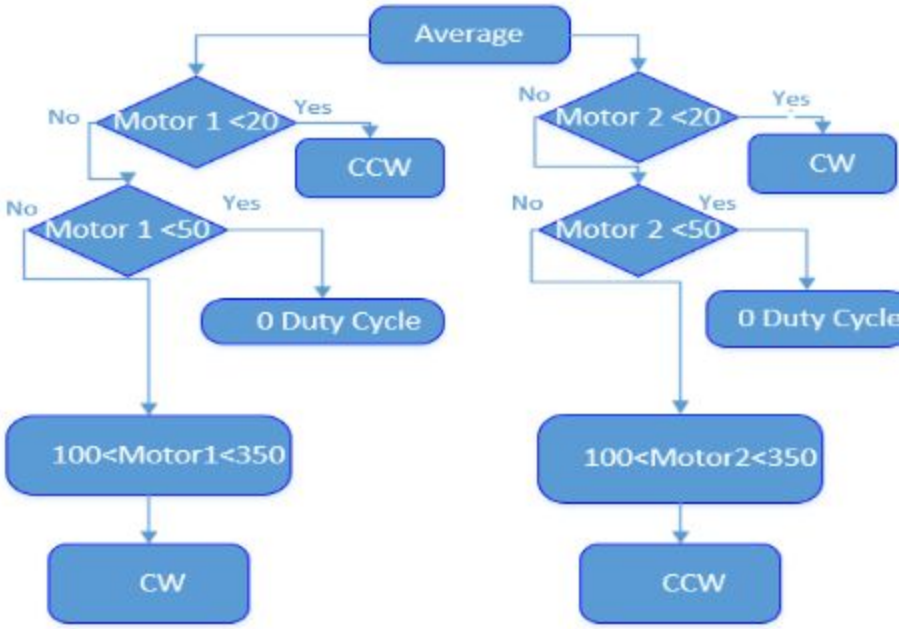


Figure 6

Sensor and Motor Interface

Table 1 shows the code references used to interface with the sensors and the motors.

Parts	**Code**	**Code source**
Sensor and Motors	**Main.c** **Utilities.h**	**https://github.com/olivia-ferda/Embedded-Team6**

Table 1. Sensor and Motor Code

Test Results

In conclusion of the project, the robot can successfully adapt to objects in front of it using the parts included above. The ultrasonic sensor is able to provide accurate measurements of detected objects in front of it. The protoboard built used to control the motors from the Atmel board with PWM came with complications, but was successfully implemented. The robot chassis that was bought provides a firm frame for all the components necessary to run the robot. Lastly, the power source used to power the ultrasonic sensor and atmel board was not appropriate for this project.

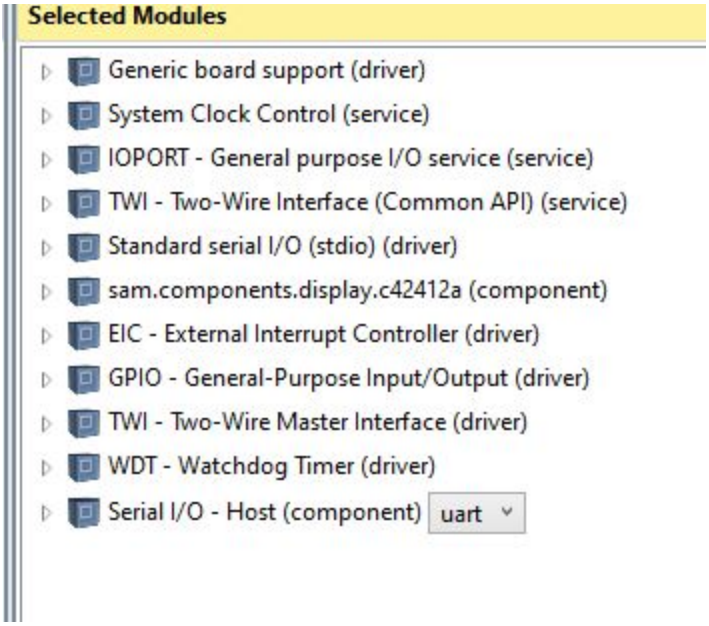
Lessons Learned

1. Voltage Dividers: Initially, we wanted to use a 12V source to power the 3 9V motors we had intended on using. To solve this, we implemented a voltage divider in the protoboard. The problem with this was: not only was the voltage divided, the amplitude was also cut down.
2. Datasheets: Despite reading the datasheets for the motors and atmel, we somewhat disregarded the limitations of how to power the ultrasonic sensor. The power source used to provide power to the atmel board and the ultrasonic sensor sent in too much amps that it burned out the sensor. Double check the datasheet to make sure none of your components go up in flames due to small mistakes.
3. Planning: creating a schedule was very beneficial and helped the group stay on track throughout the project. The problem was not taking the schedule as seriously as it should have been. Due to this lack of engagement, we were not able to plan how to connect all the different components together onto the robot chassis.
4. Cable Management: when testing the protoboard that we built, cables were a nuisance. Despite how annoying cables were, they were essential when trying to incorporate the atmel board and motors together. Having a proper system to connect the pins from the protoboard to the atmel board and motors would have greatly reduced the amount of time for testing.

Final Notes

To recreate the cruise control project:

1. Create a SAM4LC8C board project in Atmel Studio with a C/C ++ installed GCC C ASF Board Project.
2. Copy all of the code from the GitHub repository: https://github.com/olivia-ferda/Embedded-Team6/tree/master/Code/Embedded_Lab_A3_Ferda/src
3. Add all of the correct ASF Modules:



4. Run the project once all of the hardware components are connected

Appendix: Code

Main.c

```
#include <asf.h>
#include <utilities.h>
#include <conf_board.h>
#include <conf_clock.h>
#include <string.h>


int main (void)
{
   board_init();
   sysclk_init();
   configure_console();
   calculate_on_off_times();
   configure_tc();
```

```
SysTick_Config(sysclk_get_cpu_hz()/500000);

//Set priority of the SysTick Handler
NVIC_SetPriority(SysTick_IRQn,0);

//motor 1
ioport_set_pin_dir(EXT2_PIN_9,IOPORT_DIR_OUTPUT);//set motor direction A2 as output
ioport_set_pin_dir(EXT2_PIN_3,IOPORT_DIR_OUTPUT);// set motor direction A1 as output

//motor 2
ioport_set_pin_dir(EXT2_PIN_6,IOPORT_DIR_OUTPUT);//set motor direction B2 as output
ioport_set_pin_dir(EXT2_PIN_10,IOPORT_DIR_OUTPUT);// set motor direction B1 as output

ioport_set_pin_dir(EXT1_PIN_5,IOPORT_DIR_OUTPUT);//PWM for motor 1
ioport_set_pin_dir(EXT2_PIN_5,IOPORT_DIR_OUTPUT);//PWM for motor 2
printf("hi");
while(1) {
	int n = 0;
	float avg = 0;
	float avg2 = 0;
	while(n < 5){
		mdelay(25000);

		ioport_set_pin_dir(EXT1_PIN_4, IOPORT_DIR_OUTPUT);//trigger

		ioport_set_pin_level(EXT1_PIN_4,0);

		ioport_set_pin_level(EXT1_PIN_4,1);		// send out chirp from sensor
		mdelay(3);

		ioport_set_pin_level(EXT1_PIN_4,0);

		ioport_set_pin_dir(EXT1_PIN_4, IOPORT_DIR_INPUT);//echo

		while(ioport_get_pin_level(EXT1_PIN_4)==0){

		}
		startTime = ticks;

		while(ioport_get_pin_level(EXT1_PIN_4)==1){

		}
		stopTime = ticks;
```

```
        //distance calculations:
        difference = stopTime - startTime;

        rangeCm = (float)(difference * 343.0f*100.0f)/(1000000.0f);

        avg += rangeCm;

        n++;
}

mdelay(100);
avg = avg/5.0f;
printf("avg: %.2f \r\n", avg);

//motor logic based off of distance calculations:
if((avg) <= 20)
{
        //CCW:
        ioport_set_pin_level(EXT2_PIN_3,0);//low A1
        ioport_set_pin_level(EXT2_PIN_9,1);//high A2

        //CW:
        ioport_set_pin_level(EXT2_PIN_6,1);//low B2 black
        ioport_set_pin_level(EXT2_PIN_10,0);//high B1 red


        duty_cycle = 20;
        calculate_on_off_times();
        mdelay(10);
}

else if ((avg) <= 50){
        //CW
        ioport_set_pin_level(EXT2_PIN_3,1);//high
        ioport_set_pin_level(EXT2_PIN_9,0);//low

        //CCW:
        ioport_set_pin_level(EXT2_PIN_6,0);//low
        ioport_set_pin_level(EXT2_PIN_10,1);//high

        duty_cycle = 0;
        calculate_on_off_times();
```

```
            mdelay(10);
    }

    else if ((avg <= 100)){
            //CW
            ioport_set_pin_level(EXT2_PIN_3,1);//high
            ioport_set_pin_level(EXT2_PIN_9,0);//low

            //CCW
            ioport_set_pin_level(EXT2_PIN_6,0);//low
            ioport_set_pin_level(EXT2_PIN_10,1);//high

            duty_cycle = 17;
            calculate_on_off_times();
            mdelay(10);
    }

    else if ((avg <= 150)){
            //CW
            ioport_set_pin_level(EXT2_PIN_3,1);//high
            ioport_set_pin_level(EXT2_PIN_9,0);//low

            //CCW
            ioport_set_pin_level(EXT2_PIN_6,0);//low
            ioport_set_pin_level(EXT2_PIN_10,1);//high

            duty_cycle = 34;
            calculate_on_off_times();
            mdelay(10);
    }
    else if ((avg <= 200)){
            //CW
            ioport_set_pin_level(EXT2_PIN_3,1);//high
            ioport_set_pin_level(EXT2_PIN_9,0);//low

            //CCW
            ioport_set_pin_level(EXT2_PIN_6,0);//low
            ioport_set_pin_level(EXT2_PIN_10,1);//high

            duty_cycle = 51;
            calculate_on_off_times();
            mdelay(10);
    }
```

```
        else if ((avg <= 250)){
                //CW
                ioport_set_pin_level(EXT2_PIN_3,1);//high
                ioport_set_pin_level(EXT2_PIN_9,0);//low

                //CCW
                ioport_set_pin_level(EXT2_PIN_6,0);//low
                ioport_set_pin_level(EXT2_PIN_10,1);//high

                duty_cycle = 68;
                calculate_on_off_times();
                mdelay(10);
        }
        else if ((avg <= 350)){
                //CW
                ioport_set_pin_level(EXT2_PIN_3,1);//high
                ioport_set_pin_level(EXT2_PIN_9,0);//low

                //CCW
                ioport_set_pin_level(EXT2_PIN_6,0);//low
                ioport_set_pin_level(EXT2_PIN_10,1);//high

                duty_cycle = 85;
                calculate_on_off_times();
                mdelay(10);
        }

    }

    return 0;
}
```

Utilities.h

```
/*
 * utilities.h
 *
 * Created: 9/3/2019 4:42:14 PM
 *  Author: ferda
 */

#include <conf_uart_serial.h>


#include "conf_uart_serial.h"
#include "conf_c42412a_lcdca.h"
#include "core_cm4.h"
#include "c42412a_segmap.h"
#include "twim.h"

#ifndef UTILITIES_H_
#define UTILITIES_H_

volatile int num_detect;
volatile int mdelay_count;
volatile int ticks, ticks2;
volatile bool unit = true;          //default unit true is Celsius
volatile bool Led_status = false;         //lab A5 led
uint8_t  data_buf_tx[2];
uint8_t  data_buf_rx[2];

volatile int duty_cycle = 0;
volatile int on_time = 0;
volatile int off_time = 0;
volatile int tc_period = 0;
float tc_period_f = 0;
float desired_frequency = 2000.0f; //tc 2000
```

```
uint32_t tc_a_val = 0;

volatile int startTime, stopTime,startTime2, stopTime2, difference, difference2;
float rangeCm, rangeCm2;

static gpio_pin_callback_t GPIO_callback(void){
}

static void GPIO_setup(void){
	NVIC_SetPriority(EXT1_PIN_5,1);		//set the priority of GPIO

	gpio_enable_pin_interrupt (EXT1_PIN_5);		//enable interrupt

	gpio_set_pin_callback (EXT1_PIN_5, GPIO_callback, NVIC_GetPriority(25));	//make
a callback func

}


void calculate_on_off_times()
{
   static bool prev_stop = false;
   float tc_a_val_f = tc_period_f * ((float)duty_cycle)/100.0f;
   //printf("tc_a_val: %f\r\n", tc_a_val);
   tc_a_val = (uint32_t) tc_a_val_f; //values to write
   on_time = tc_a_val;
   off_time = tc_period - tc_a_val;
   if(prev_stop == true)
   {
	tc_start(TC0, 0);
   }
   if(duty_cycle == 30){
	prev_stop = true;
	//printf("on_time: %d\r\n", on_time);
	//printf("off_time: %d\r\n", off_time);
   }
   if(duty_cycle == 60){
	prev_stop = true;
	//printf("on_time: %d\r\n", on_time);
	//printf("off_time: %d\r\n", off_time);
   }
   if(duty_cycle == 100)
   {
```

```
            tc_stop(TC0, 0);
            ioport_set_pin_level(EXT1_PIN_5, true);
            ioport_set_pin_level(EXT2_PIN_5, true);
            prev_stop = true;
    }
    else if(duty_cycle == 0)
    {
            tc_stop(TC0, 0);
            ioport_set_pin_level(EXT1_PIN_5, false);
            ioport_set_pin_level(EXT2_PIN_5, false);
            prev_stop = true;
    }
    //printf("on = %d off = %d\r\n", on_time, off_time);
}



static void mdelay(uint32_t delay_ticks){
            uint32_t starting_tick;

            starting_tick = ticks;
            while((ticks - starting_tick)< delay_ticks);
}

void SysTick_Handler(void){
            ticks++;
    ticks2++;
            //printf("hi = %d ", ticks);
}

void configure_sys_tick(int ticks_per_second){
            SysTick_Config(sysclk_get_cpu_hz()/ticks_per_second);
}

static void configure_tc(void)
{
            uint32_t ul_tcclks = 1;
            //clock speed
            uint32_t ul_sysclk = sysclk_get_pba_hz();
            //TCO Config
            sysclk_enable_peripheral_clock(TC0);

            //Calculate the values you want to use for each timer counter
```

```
	tc_period_f = ((float)sysclk_get_pba_hz()/2.0f)/desired_frequency;
	tc_period = (uint32_t) tc_period_f;
	calculate_on_off_times();
	tc_init(TC0, 0, ul_tcclks | TC_CMR_CPCTRG); //3 timer channels option 1

	//enable interrupt
	NVIC_EnableIRQ((IRQn_Type) TC00_IRQn);
	tc_enable_interrupt(TC0, 0, TC_IER_CPCS);

	// start timer counter if you want the tc to start right away
	// in my case, I don't want the timer counter to start until
	// i get user input
	//		tc_start(TC0, 0);


}

//interrupt handler for each task
void TC00_Handler(void) //provided and runs every time interrupt occurs (stops main then
returns to same spot)
{
	static bool turn_on = true;
	tc_get_status(TC0, 0);
	tc_stop(TC0, 0);
	if(turn_on)
	{
	ioport_set_pin_level(EXT1_PIN_5, true);
	 ioport_set_pin_level(EXT2_PIN_5, true);
	tc_write_rc(TC0, 0, on_time);
	}

	else
	{
	ioport_set_pin_level(EXT1_PIN_5, false);
	 ioport_set_pin_level(EXT2_PIN_5, false);
	tc_write_rc(TC0, 0, off_time);
	}

	turn_on = !turn_on;
	tc_start(TC0, 0);

}
```

```
static void configure_console(void)
{
const usart_serial_options_t uart_serial_options =
{
.baudrate = CONF_UART_BAUDRATE,
.charlength = CONF_UART_CHAR_LENGTH,
.paritytype = CONF_UART_PARITY,
.stopbits = CONF_UART_STOP_BITS,
};
/* Configure console. */
stdio_serial_init(CONF_UART, &uart_serial_options);
}



#endif /* UTILITIES_H_ */
```