DOCUMENTATION

WINTER WORKSHOP (PHASE 2)

Problem Statement: **GRID FOLLOWER WITH OBSTACLE DETECTION**

- Mentors: SAKSHI AGARWAL
  ADITYA NARAYAN
  ABHINAV PACHAURI

DAY 1

Basics of C++

- **Syntax of for loop, while loop, do while**
- **Function call**
- **Operations on a byte**

Set – (a| (1<<n))

Clear- (a&~ (1<<n))

Toggle-(a^ (1<<n))

- **Binary operators**

AND (&)

OR (|)

XOR (^)

NOT (~)

LEFT SHIFT (<<)

RIGHT SHIFT (>>)

TOPIC DISCUSSED

1. Power Supply

2. Sensor System

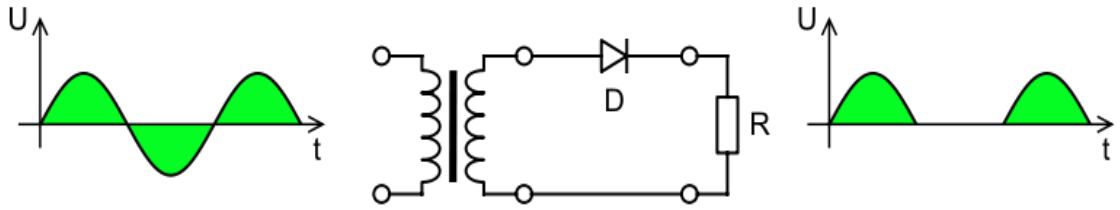
3. Microcontroller

4. **Locomotion System**

✓ POWER SUPPLY

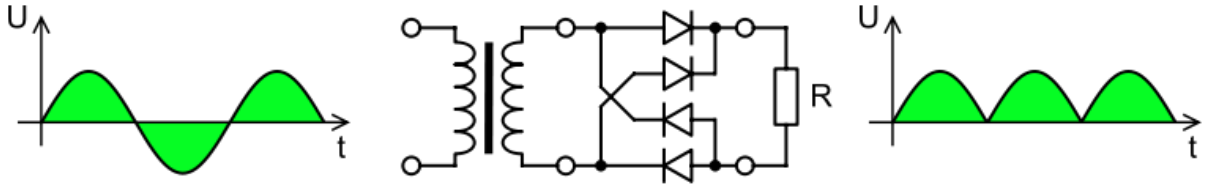
HALF WAVE RECTIFIER:

In half wave rectification of a single-phase supply, either the positive or negative half of the AC wave is passed, while the other half is blocked. The frequency of the output voltage in half wave rectifier is equal to that of input voltage.



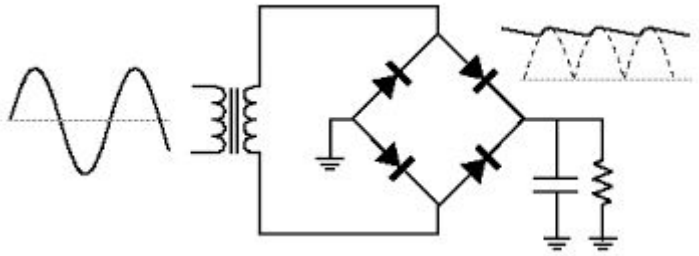
FULL WAVE RECTIFIER:

A full-wave rectifier converts the whole of the input waveform to one of constant polarity (positive or negative) at its output.



BRIDGE RECTIFIER:

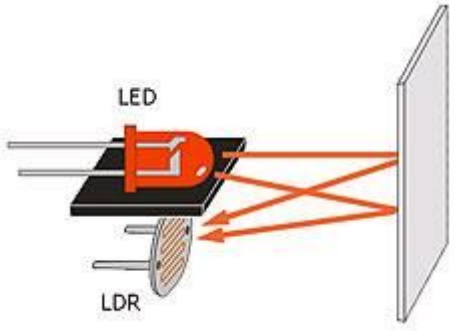
Another type of circuit that produces the same output waveform as the full wave rectifier circuit is that of the Full Wave Bridge Rectifier. This type of single phase rectifier uses four individual rectifying diodes connected in a closed loop "bridge" configuration to produce the desired output.



SENSOR

LDRs or Light Dependent Resistors are used in light/dark sensor circuits. Normally the resistance of an LDR is very high, but when they are illuminated with light resistance drops dramatically.

- Resistance of LDR is inversely proportional to intensity of light.



It is used to detect the white and black path and helps the robot to move in the correct path.

DAY 2

Topic discussed are:

- **Peripherals of microcontroller**
- **Register**
- **ADC**
- **Timer**

  Mode of operation

  PWN

Phase current

MICROCONTROLLER

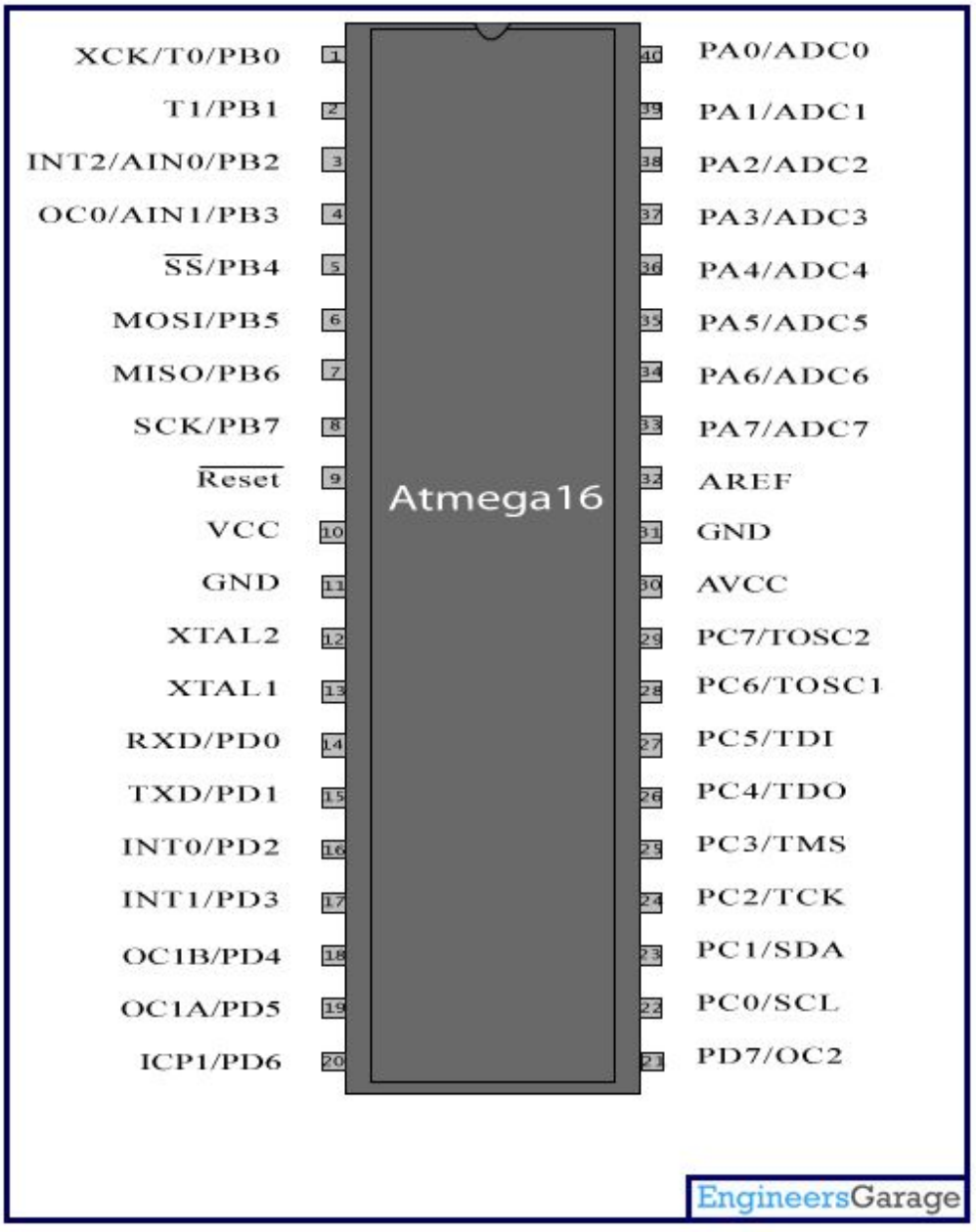
A microcontroller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals.

A micro-controller is a single integrated circuit, commonly with the following features:

- Central processing unit - ranging from small and simple 4-bit processors to complex 32- or 64-bit processors
- Volatile memory (RAM) for data storage
- ROM, EPROM, EEPROM or Flash memory for program and operating parameter storage
- Discrete input and output bits, allowing control or detection of the logic state of an individual package pin
- Serial input/output such as serial ports (UARTs)
- peripherals such as timers, event counters, PWM generators
- Clock generator - often an oscillator for a quartz timing crystal, resonator or RC circuit
- many include analog-to-digital converters, some include digital-to-analog converters

- in-circuit programming and debugging support

MICROCONTROLLER----ATMEGA16



MEMORIES IN MICROCONTROLLER

Types of RAM

The RAM family includes two important memory devices: static RAM (SRAM) and dynamic RAM (DRAM). The primary difference between them is the lifetime of the data they store. SRAM retains its contents as long as electrical power is applied to the chip. If the power is turned off or lost temporarily, its contents will be lost forever. DRAM, on the other hand, has an extremely short data lifetime-typically about four milliseconds.

Types of ROM

- **PROM (programmable ROM)**
- **EPROM (erasable-and-programmable ROM):**

To erase an EPROM, you simply expose the device to a strong source of ultraviolet light.

- **EEPROM(electronically erasable and programmable read only memory**):

EEPROMs are electrically-erasable-and-programmable. Internally, they are similar to EPROMs, but the erase operation is accomplished electrically, rather than by exposure to ultraviolet light. Any byte within an EEPROM may be erased and rewritten.

Pin Descriptions

VCC - Digital supply voltage.

GND - Ground.

Port A (PA7..PA0) - Port A serves as the analog inputs to the A/D Converter.

Port B (PB7..PB0) - Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit).

Port C (PC7..PC0) - Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit).

Port D (PD7..PD0) - Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit).

XTAL1 - Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

XTAL2 - Output from the inverting Oscillator amplifier.

AVCC - AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to V_{CC}, even if the ADC is not used.

AREF - AREF is the analog reference pin for the A/D Converter.

REGISTERS

Each port is controlled by three registers, which are also defined variables in the arduino language.

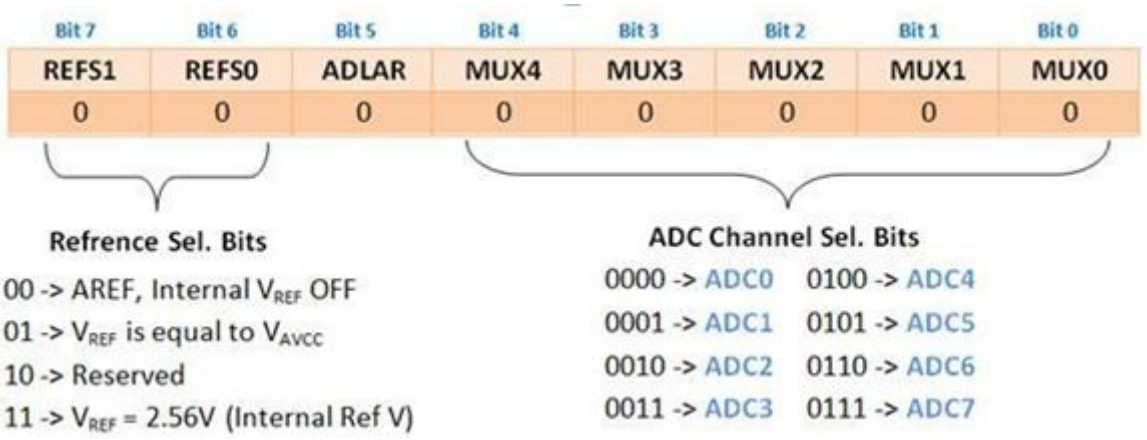
- The DDR register, determines whether the pin is an INPUT or OUTPUT.
- The PORT register controls whether the pin is HIGH or LOW
- PIN register reads the state of INPUT pins set to input with pinMode ().

Registers

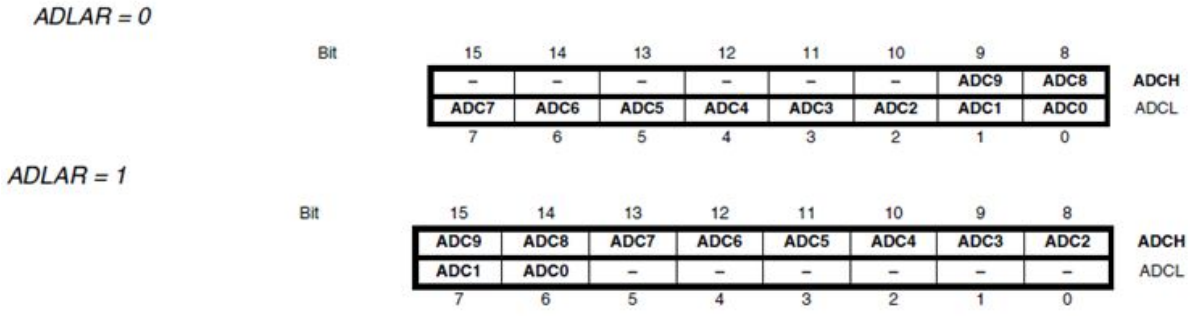
1. ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

2 ADMUX



ADLAR bit when set to 1 gives the left adjusted result in data registers ADCH and ADCL. These help to get the required precision in the output.



DAY 3

Timer

Each timer has different modes of operations:

- **Normal Mode**
- **Fast PWM**
- **Phase Correct PWM**
- **Clear Timer on Compare Match (CTC) Mode**

Normal Mode

This is the simplest mode of operation. For 8-bit timer TCNT register starts from BOTTOM (0) and counts till TOP (255).

Fast PWM

In this mode the OCn pin (for 8-bit timers, OC0 and OC2) is the output pin for PWM of duty-cycle dependent on the value of OCRn register. The TCNT value goes from BOTTOM (0) to TOP (255). When the value of TCNT matches with that of the OCR register, the OC pin can be reset/set according to the mode selected (non-inverting/ inverting).

Phase Correct PWM

In this mode the TCNT increases from BOTTOM to MAX and then falls from MAX to BOTTOM. So in this case the duty cycle will remain same as that in the case of the Fast PWM but frequency is halved.

Clear Timer on Compare Match (CTC) Mode

In this mode we can set the value of the TOP. So if we set the value of TOP as 100 instead of 255(default TOP value), the TCNT counts from BOTTOM (0) to 100 and then again drops to BOTTOM (0). So, if the counts of TCNT decrease then the time period of the oscillation decreases. In this mode the OCn pin (for 8-bit timers, OC0 and OC2) is the output pin for PWM.

Duty Cycle:

Duty Cycle is the percentage of time in a time period for which the Voltage is HIGH.

DAY 4

TIMER

The input clock of microcontroller and operation of the timer is independent of the program execution. All the Atmel microcontrollers have Timers as an inbuilt peripheral.

ATmega8 comes with two 8-bit and one 16-bit timer. This means that there are 3 sets of timers, each with the ability to count at different rates. The two 8-bit counters can count to 255 while the 16- bit counter

can count to 65,536. The simplest timer in Atmeag8 is TIMER0 with an 8-bit resolution (0-255). Timers can run asynchronous to the main AVR core hence timers are totally independent of CPU. A timer is usually specified by the maximum value to which it can count, beyond which it overflows and resets to zero.

The Timer16 User Module employs from two digital PSoC blocks, each contributing 8 bits to the total resolution. The consecutive blocks are linked so their internal carry, terminal count and compare signals are synchronously chained. This concatenates the 8-bit Count, Period and Compare registers (Data registers DR0, DR1 and DR2, respectively) from block to block to provide the required resolution. In this way, the 16-bit timer operates as a single monolithic synchronous timer.

- sensor circuit was made to detect node
- code for grid following was discussed

DAY 5

INTERRUPT

An interrupt is an unexpected hardware initiated subroutine call or jump that temporarily suspends the running of the current program.

Interrupts occur when a peripheral device asserts an interrupt input pin of the micro-processor. Provided the interrupt is permitted, it will be acknowledged by the processor at the end of the current memory cycle. The processor then services the interrupt by branching to a special service routine written to handle that particular interrupt. Upon servicing the device, the processor is then instructed to continue with what is was doing previously by use of the "return from interrupt" instruction.

Interrupts fall into two major categories, maskable and non-maskable. An interrupt may be either EDGE or LEVEL sensitive.

The processor can inhibit certain types of interrupts by use of a special interrupt mask bit. This mask bit is part of the flags/condition code register, or a special interrupt register. If this bit is set, and an interrupt request occurs on the Interrupt Request input, it is ignored.

SONAR

- Sonar is a type of proximity sensor that uses sound to measure the relative distance between the sensor and objects in the environment. Sound waves are emitted and then detected after they bounce off objects in the environment. The time difference between emission and detection is used to determine the distance to an object.

- Microcontroller tells sonar to go. Then sonar emits a mostly inaudible sound, time passes, then detects the return echo. It then immediately sends a voltage signal to the microcontroller, which by keeping track of the time that passes, can calculate the distance of the object(s) detected.

DAY 6

CODE debug and threshold the sensor circuit.

DAY 7

Code for grid follower with sonar

```
#define threshold 85
#define forward 0b00100100
#define left_zero 0b00101000                //defining motor controls
#define right_zero 0b00010100
#define stall 0b00111100
#define back 0b00011000
#define F_CPU 16000000UL
#define left 0b00101100
#define right 0b00110100
#define east 0
#define north 1                              //defining directions
#define west 2
#define south 3
#include "LCD.h"                               //LCD display files
#include "LCD.c"
#include <string.h>
#include <avr/io.h>
```

```
#include <util/delay.h>
#include <avr/interrupt.h>

volatile uint8_t overflow;
int time;
int counter;                                    //global variables
int dist;
int dir;
int X = 0,Y = 0;

void motor_test()                               //to test motor
{
	PORTC = forward;
	_delay_ms(1000);

	PORTC = right;
	_delay_ms(1000);

	PORTC = right_zero;
	_delay_ms(1000);

	PORTC = left;
	_delay_ms(1000);

	PORTC = left_zero;
	_delay_ms(1000);

	PORTC = back;
	_delay_ms(1000);
}

uint8_t get_IR_sensor_value()                   //input from 6 LED LDR sensors
{
	uint8_t i;
	i = PINA;
	i = i&(0b00111111);                         //removing other pins value
	return i;
}

void follow_line(uint8_t i)                     //command to follow line on basis
or sensor values
{
	switch (i)
	{
		case 0b00001100:
		case 0b00011110:
		case 0b00111111:
		PORTC = forward;
		break;

		case 0b00000110:
		case 0b00001110:
		case 0b00000111:
		PORTC = left;
		break;

		case 0b00011000:
		case 0b00011100:
		case 0b00111000:
		PORTC = right;
		break;
```

```
                case 0b00110000:
                case 0b00100000:
                PORTC = right_zero;
                break;

                case 0b00000011:
                case 0b00000001:
                PORTC = left_zero;
                break;

                default:
                PORTC = forward;
                break;
        }

}

void initialize()
{
        DDRC = 0b11111101;                              //to output and input selections
        PORTC = (1<<PC1);
        DDRA = 0b00000000;
        DDRB = 0b11111111;
        DDRD = 0b11111111;

        ADMUX=(1<<REFS0)|(1<<ADLAR)|(1<<MUX1)|(1<<MUX2);          //filling ADC
registers
        ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);

        TCCR2=(1<<CS22)|(1<<CS21)|(1<<CS20);                      //filling timer
counter registers
        TIMSK|=(1<<TOIE0);
        TCCR1A=(1<<WGM10)|(1<<COM1A1)|(1<<COM1B1);
        TCCR1B=(1<<WGM12)|(1<<CS10)|(1<<CS11);

        OCR1A = 180;                                            //setting PWM to slow
down motors
        OCR1B = 180;

        dir=0;
        overflow = 0;
}

void next_node()                                                //for travelling between
nodes
{
        uint8_t a;
        while(1)
        {
                a = get_IR_sensor_value();
                if (a!=0b00111111){
                        follow_line(a);
                }

                else{
                        clear_display();
                        print_string("NODE DETECTED");
                        break;

                }
        }
}
```

```
int get_node_sensor_value()                                    //for node sensor values
{
	ADCSRA|=(1<<ADSC);
	while((ADCSRA &(1<<ADSC))!=0);
	int a = ADCH;
	if(a>threshold){
		return 0;
	}
	else{
		return 1;
	}

}

void get_to_node()                                        //travelling upto
{                                                          //nodes after detecting
	while(1){
		int b = get_node_sensor_value();
		if(b==0){
			int i=get_IR_sensor_value();
			follow_line(i);
		}
		else{
			PORTC = stall;
			coordinate_update();
			_delay_ms(1000);
			return;
		}
	}
}

void turn_right()                                  //turning on nodes
{
	int c = get_node_sensor_value();
	int d = get_IR_sensor_value();
	PORTC = right_zero;
	while(d){
		d=get_IR_sensor_value();
	}
	_delay_ms(10);
	while (c==0){
		c=get_node_sensor_value();
	}
	while(get_IR_sensor_value()==0);
	PORTC = stall;
}

void turn_left()
{
	int c = get_node_sensor_value();
	int d = get_IR_sensor_value();
	PORTC = left_zero;
	while(d){
		d=get_IR_sensor_value();
	}
	_delay_ms(10);
	while (c==0){
		c = get_node_sensor_value();
	}
	while(get_IR_sensor_value()==0);
	PORTC = stall;
```

```
}

void turn_around()
{
	turn_right();
	turn_right();
}

void change_dir(uint8_t v)                    //changing directions
{
	int a =  v - dir;
	switch(a){
		case 1:
		case -3:
		turn_right();
		break;

		case -1:
		case 3:
		turn_left();
		break;

		case 2:
		case -2:
		turn_around();
		break;

		default:
		break;
	}
	dir = v;
}

ISR(TIMER0_OVF_vect)                          //inerrupt on counter overflow
{
	overflow++;
}

void sonar()                                  //initialize sonar pins and triggering sonar
{
	overflow = 0;
	PORTC|=(1<<PC0);
	_delay_us(50);
	PORTC&=~(1<<PC0);
	while(!(PINC&(1<<PC1)));
	TCNT2 = 0;
	while ((PINC&(1<<PC1))&&TCNT2<245);
	print_integer(overflow);
	counter = overflow*256 + TCNT2;
	time = counter*(1024/16);
	clear_display();
	print_integer(counter);
	_delay_ms(200);
}

void coordinate_update()                     //updating coordinates on reaching node
{
	switch(dir)
	{
		case 0:
		Y++;
		break;
```

```
                case 1:
                X++;
                break;

                case 2:
                Y--;
                break;

                case 3:
                X--;
                break;
        }
        print_string("X=");
        print_integer(X);
        print_string(",Y=");
        print_integer(Y);
}

int obstacle_detection()                        //for detecting obstacles
{
        sonar();
        if (counter>80)
        {return 0;
        }
        else
        {
                return 1;
        }
}

int main(void)                      //main
{
        //motor_test();

        initialize();
        init_LCD();
        _delay_ms(1000);
        clear_display();

        sei();                          //enabling global interrupts

        while(1)
        {
                next_node();
                clear_display();
                print_string("NEXT NODE 1");
                get_to_node();
                clear_display();
                print_string("REACHED NODE 1");

                next_node();
                clear_display();
                print_string("NEXT NODE 2");
                get_to_node();
                clear_display();
                print_string("REACHED NODE 2");


                change_dir(1);
                if (obstacle_detection()){
                        PORTC = stall;
```

```
                        print_string("OBSTACLE");
                        break;
                }

                next_node();
                clear_display();
                print_string("NEXT NODE 3");
                get_to_node();
                clear_display();
                print_string("REACHED NODE 3");

                if (obstacle_detection()){
                        PORTC = stall;
                        clear_display();
                        print_string("OBSTACLE DECTECTED");
                        _delay_ms(1000);
                        turn_around();
                        break;

                }

        }
        return 0;
}
```