WeFee Bot

CpE 301 Final Project
May 4, 2017

Team #12
Reiner Dizon
Angel Solis

Abstract –

WeFee Bot is a Wi-Fi-operated, battery-operated robot vehicle system that avoids wall collisions and has sleep functionality to save battery power. The brain of the system is the Atmega328p microcontroller which manages all the moving parts of the robot.

1. Introduction

The goal of this project is to build a wirelessly user-controlled system which mimics the collision system in today's car but in the small scale. We implemented the system using the Atmega328p microcontroller chip as the main control unit for the system. This MCU controls and monitors all the peripherals of this system which is all listed in Table 1. It takes in input from ultrasonic distance sensor and ESP32 module. The information from these peripherals dictate the controls for the output, such as the DC motors, H-Bridge, passive buzzer, and LED.

The ultrasonic distance sensor measures distance by sending ultrasonic sound waves and measuring the time when the sound waves reflect onto the system. We used this sensor to measure the distance between the robot and the nearest obstacle, such as a wall. Based on this distance information, the system stops and backs itself up when it approaches an obstacle in front of it. As the robot approaches a wall, a red LED indicator light will turn on.

The ESP32 breakout module handles the Wi-Fi communication between the mobile application and itself. After processing the information from the smartphone, it then relays this information to the Atmega328p microcontroller. The user interacts with the system using a mobile application called Blynk on an Android device. This application enables us to create the graphical user interface for the controls of the robot, which was a joystick for direction and a button for sleep mode. Moreover, the application communicates with ESP32 module via Wi-Fi, so we processed the joystick information through the ESP32 before sending it to the Atmega328p MCU.

The DC motors control the two front wheels using the H-bridge driver which is connected to the Atmega328p chip. When the user decides to not operate the vehicle, the user can place the system into sleep mode to save battery life.

Table 1: List of Peripherals

Inputs	Outputs
Ultrasonic Distance Sensor ESP32 WiFi/Bluetooth Module	H-Bridge DC Motor Passive Buzzers LEDs

2. Implementation

The provided robot chassis kit is where the entire system is built upon. Since it came with an Arduino and shield kit, we removed those as they were irrelevant parts to our system. To have a common ground for the entire system, we wired a ground wire from the H-bridge drive and connected it to the ground line of the breadboard. Once we gutted the chassis kit, we worked on implementing different peripherals in software. Angel worked on the driving the motor using four PWM outputs from Timer's 0 and 2 which connects to the inputs of the H-bridge driver. Reiner worked on the detecting walls using the ultrasonic sensor by using Timer 1 to time the reflection of the ultrasonic waves and converting the time into distance. After we tested these codes, we transformed them into different header files and created a main function to implement the system. Using the Pin Change Interrupt, we implemented the sleep mode functionality for the system. To indicate sleep mode, we used microsecond delays to ring the buzzer before the system goes to sleep which is connected to a general I/O pin.

High-Level Hardware Schematic

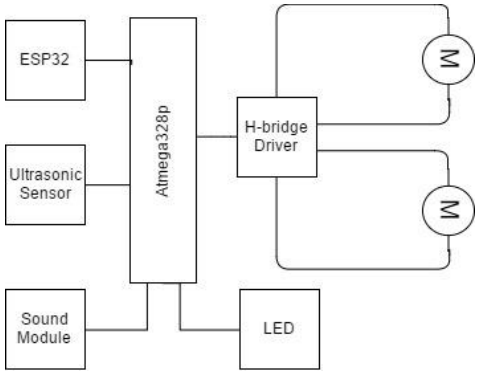


Fig 1 – High-Level Schematic

Figure 1 displays the high-level schematic of our system. It shows all of the peripherals that are connected to the Atmega328p microcontroller. This setup indicates that all these peripherals interact through the microcontroller and not to one another.

High-Level Software Flow Diagram (Main Function)

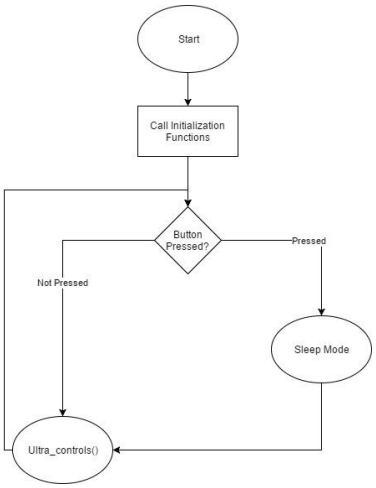


Fig 2 – Main Function

Figure 2 shows the high-level flow diagram of our software implementation. First, all the pins connected to the peripherals are initialized as well as necessary interrupts such as pin changes and timer overflow. As mentioned earlier, the GUI has a sleep mode button which is checked after the initialization if user presses it. If the user presses the button, the system will go to sleep until the user presses the button again. Otherwise, the main function calls the controls function to handle driving the motor as well as wall detection.

Flow Diagram for Other Functions

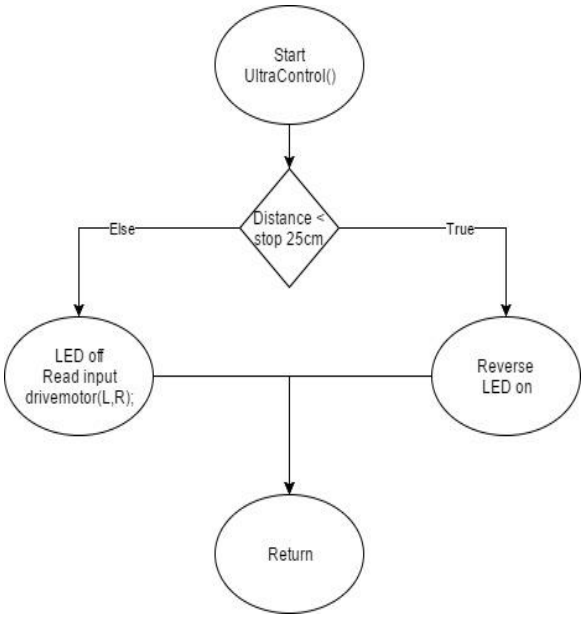


Fig 3 – Controls Function

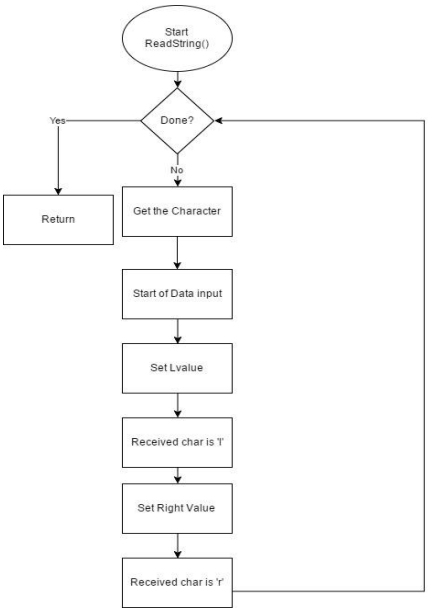


Fig 4 – Read String Function

The controls function (Figure 3) first checks the robot's distance to the nearest wall if it is 25 centimeters away. If that is the case, the robot reverses, and the LED indicator lights up until it backs up to a safe distance. Otherwise, the function turns off the LED indicator and reads the input from the ESP32 module.

When the ESP32 boots up, the module sends a string of irrelevant values to the Atmega328p. To ensure proper reading, the read function (Figure 4) waits for a '<' character from the ESP32. After the ESP32's initialization, the joystick data from the smartphone comes as two integer values: x and y direction values which the ESP32 receives. The ESP32's code also calculates the percentages for the left and right wheels. Then, the Wi-Fi module sends four bytes of information relating to this data: the left value followed by the 'l' character, then the right value followed by the 'r' character. After the Atmega receives these bytes, the read string function returns to the controls function which then drives the motor per joystick data.

The original code of the ESP32 sends a string of characters which prints the x and y directions in ASCII characters, instead of bytes of information. We modified this code slightly to accommodate for the read string function. The first implementation of the read string function made the system slow since we waited for parts of joystick data to come in, instead of all of it before returning to the controls function. Also, rather than cluttering the main function with all the code, we modularize the peripherals into separate header files to clean up code everywhere.

3. Experimental evaluation / Results

The PWM output to the left and right wheels of the robot determines direction of the robot. This setup mimics tank controls as opposed to the wheel-and-axle systems of cars today. Along with tank controls, the wall detection system prevents collision by reversing the system back and indicating the user that he or she does not have control of the vehicle until it backs up to a safe distance. With the addition of the sleep functionality, the user can put the robot to rest to save battery and to reset the system if it malfunctions. The use of UART communication is key to relay information from the smartphone application to the ESP32 and to the Atmega328p.

4. Conclusion

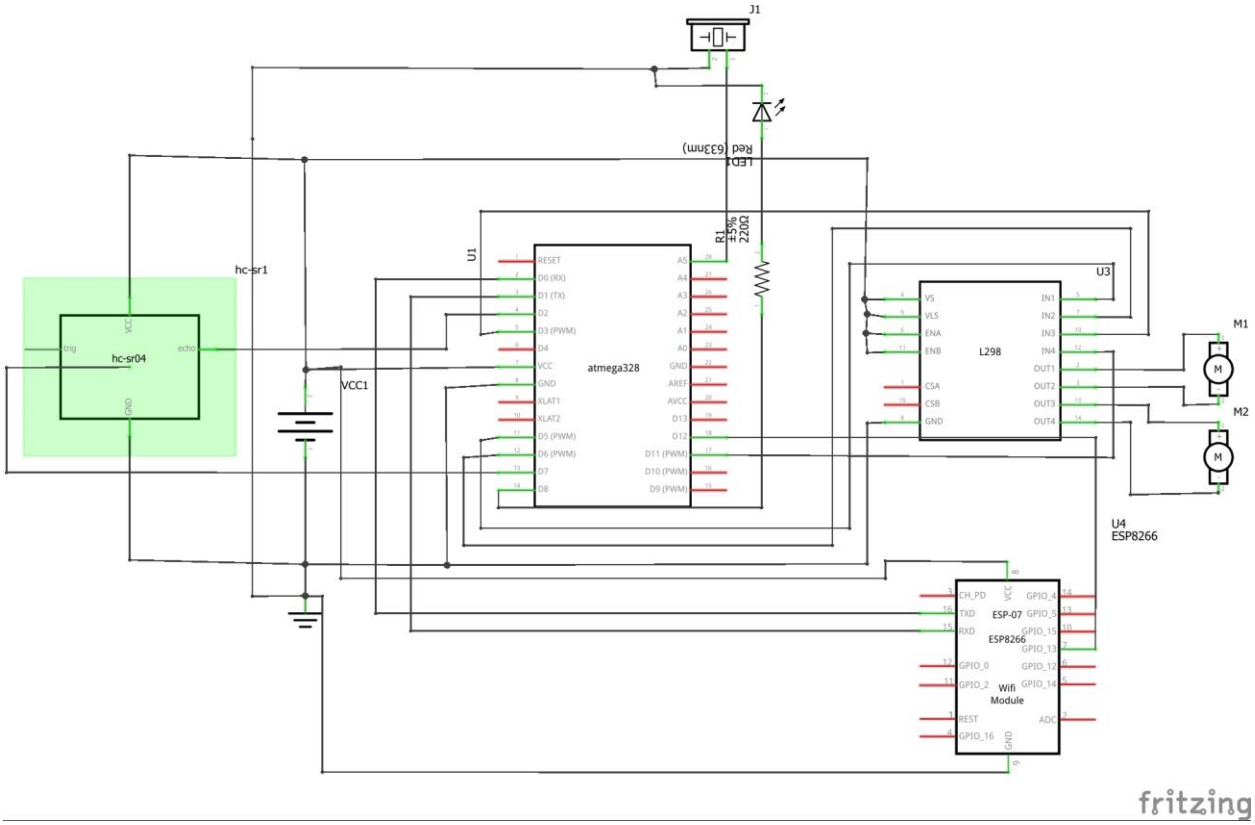
We created a system that expands on the functionalities of a typical RC car with the addition of a wall detection system and sleep mode functionality. Rather than using unsecured radio waves to control the system remotely, Wi-Fi allows for the remote control of the system from a farther distance than a RC system would reach.

With regards to the new development boards such as the ESP32 module, we do not recommend using them during their initial months out of the market. When we needed to use the ESP32, we had to plug it in to a computer so that it can initialize its UART functionality. Moreover, the libraries for its software often change as the makers continue to refine the features of the module. These continued changes made it difficult to find help for these new features. Partly due to the changing nature of the ESP32's libraries, we decided to use Wi-Fi rather than Bluetooth for communication as the Wi-Fi libraries were more readily available than Bluetooth's. Also, the Blynk application only supports Wi-Fi for the ESP32 breakout board currently which also affected our decisions in terms of wireless communication.

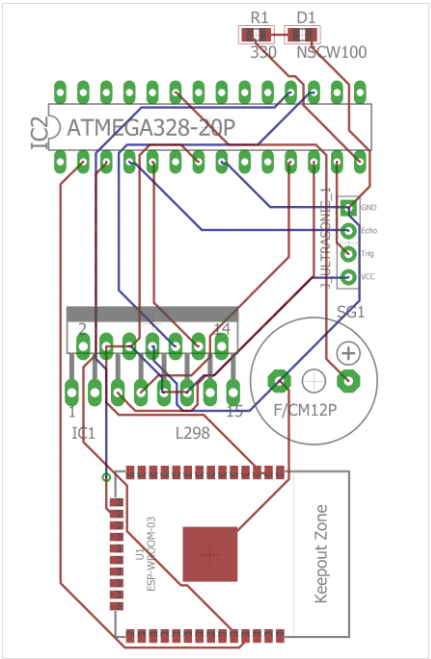
To improve direction controls, we would use a servo motor to precisely control the robot as opposed to the tank controls. The servo motor would only affect the direction of the middle wheel which is not connected to the DC motors. This setup would mean that only two PWMs control the DC motors to indicate forward and backward directions. One of the remaining PWM would interact with the servo motor. Along with this enhanced controls, we would eliminate the use of the breadboard. Instead, we would print a PCB of the system and solder all the wires onto it to improve the stability of the system along with its peripherals.

5. Appendix

- **Detailed Schematics**



- **PCB**



- **Bill of Materials**

Description	Manufacture	Distributor Part Numbe	Distributo	Web Link	Quantity	Cost (USD	Notes	Cost per K
Passive Buzzer	KY-006		DX	http://www.dx.com/	1.00	1.74		1.74
Robot Chassis	LYSB01LWPTI	B01LXY7CM3	amazon	https://www.amazon	1.00	14.69		14.69
Breadboard	EL-CP-004	B0084A7PI8	Amazon	https://www.amazon	1.00	4.97		4.97
Jumper wires set	IB400	B01EV70C78	Amazon	https://www.amazon	0.13	8.86	Kit price:	1.11
Red LED	YSL-R531R3d	COM-09590	SparkFun	https://www.sparkfu	1.00	0.35		0.35
Ultrasonic Sensor	HC-SR04	SEN-13959	SparkFun	https://www.sparkfu	1.00	3.95		3.95
							Cost per k	**26.81**

- **Code**

main.c

```
#define F_CPU 8000000UL       //!< clock speed of Atmega328p - 8MHz
#define BUTTON (PINB & (1<<PINB4)) // check for button press

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <stdlib.h>
#include "peripherals/drive_motor.h"
#include "peripherals/ultra_sensor.h"
#include "serial/uart.h"
#include "sleep.h"

int main(void){
      // initialize WEFEE
      initUART();
      init_motor();
      init_ultrasensor();
      init_PCINT();

      while(1){
            if(BUTTON)   //sleep mode check
                  good_night(); // NIGHT NIGHT

            ultra_controls();// call the do everything function
      }
}
```

peripherals/drive_motor.h

```
#ifndef DRIVE_MOTOR_H
#define DRIVE_MOTOR_H

#include <avr/io.h>

/**
* @defgroup driveMotorFunct Drive Motor
* @brief Enables the wheels to move in any direction
* @{
*/
```

```
#define PMW0A PORTD6
#define PMW0B PORTD5
#define PMW2A PORTB3
#define PMW2B PORTD3
//! Initialize motor pins
void init_motor(){
      DDRB |= (1<<PMW2A);
      DDRD |= (1<<PMW2B);
      DDRD |= (1<<PMW0B)|(1<<PMW0A);

      // top is 255
      TCCR0A = (1<<COM0A1)|(1<<COM0B1)|(1<<WGM00)|(1<<WGM01);           //
NON INVERTING MODE
      TCCR0B = (1<<CS00);
            // no Pre-scalar

      TCCR2A = (1<<COM2A1)|(1<<COM2B1)|(1<<WGM20)|(1<<WGM21);           //
NON INVERTING MODE
      TCCR2B = (1<<CS20);
            // no Pre-scalar

}

//! Allows left & right wheel movements (forwards, backwards)
void drive_motor(int left_pct, int right_pct){
      int tempL = (int)(left_pct/100.0 * 255);
      int tempR = (int)(right_pct/100.0 * 255);

      if (tempL>=0 && tempR>=0){
            OCR0A = 0;
      // no movement in other direction
            OCR0B = (tempL)<255?tempL:255;
      // change duty cycle
            OCR2A = 0;
      // no movement in other direction
            OCR2B = (tempR)<255?tempR:255;
      // change duty cycle

      }
      else if (tempL>=0 && tempR<0){
            OCR0A = 0;
      // no movement in other direction
            OCR0B = (tempL)<255?tempL:255;
      // change duty cycle
            OCR2A = (-tempR)<255?(-tempR):255;                           // no
movement in other direction
            OCR2B = 0;
      // change duty cycle

      }
      else if (tempL<0 && tempR>=0){
            OCR0A = (-tempL)<255?(-tempL):255;                           // no
movement in other direction
            OCR0B = 0;
      // change duty cycle
            OCR2A = 0;
      // no movement in other direction
```

```
            OCR2B = (tempR)<255?tempR:255;
      // change duty cycle

      }
      else{
            OCR0A = (-tempL)<255?-tempL:255;                              // no
movement in other direction
            OCR0B = 0;
      // change duty cycle
            OCR2A = (-tempR)<255?-tempR:255;                              // no
movement in other direction
            OCR2B = 0;
      // change duty cycle
      }
}
/**@}*/
#endif
```

peripherals/ultra_sensor.h

```
#ifndef ULTRA_SENSOR_H
#define ULTRA_SENSOR_H

#define F_CPU 8000000UL       //!< clock speed of Atmega328p - 8MHz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
#include "drive_motor.h"
#include "../serial/uart.h"

/**
* @defgroup ledIndic LED Wall Collision Warning
* @brief Defines the use of the LED wall collision warning system
* @{
*/
#define LED_PORT        PORTB0                                  //!< LED
port (PORTB0)
#define LED_ON                PORTB |= (1<<LED_PORT)        //!< turn on LED
(PORTB0)
#define LED_OFF               PORTB &= ~(1<<LED_PORT)       //!< turn off LED
(PORTB0)
/**@}*/

/**
* @defgroup ultraSenFunct Ultrasonic Distance Sensor
* @brief Allows the wall collision functionality of robot
* @{
*/

#define TRIG_PORT       PORTD7      //!< trigger port (PORTD7)
#define ECHO_PORT       PORTD2      //!< echo echo echo...echo port (PORTD2)

static volatile unsigned int  pulse;                  //!< pulse from
ultrasonic sensor
static volatile unsigned int  i;                      //!< state decider for
timer 1
```

```
static volatile unsigned int  distance;         //!< actual distance
calculated

int stop_dist = 25;                                                    //!<
boarder distance for backup

//! Initialize ultrasonic sensor pins
void init_ultrasensor(){
      pulse = 0;
      i = 0;

      DDRD |= (1<<TRIG_PORT);       // trigger output
      DDRD &= ~(1<<ECHO_PORT);      // echo input
      DDRB |= (1<<LED_PORT);        // LED indicator

      EIMSK |= (1<<INT0);                 // enable interrupt zero
      EICRA |= (1<<ISC00);          // positive edge
      TCCR1A = 0;                         // timer for distance t1

      sei();                                    // enable global interrupts
}

//! Detects wall collision & brakes accordingly
void ultra_controls(){ // controls everything on bot
      PORTD |= (1<<7);
      _delay_us(15);
      PORTD &= ~(1<<7);
      distance = pulse/58;
      _delay_ms(20);

      if(distance < stop_dist){
            drive_motor(-100, -100);
            LED_ON;
      }
      else{
            LED_OFF;
            readString();
            if(done)
                  drive_motor(l, r);
            else
                  drive_motor(0, 0);
      }
}

//! Internal Interrupt 0 ISR
ISR(INT0_vect){
      if(i == 1){
            TCCR1B = 0;
            pulse = TCNT1;
            TCNT1 = 0;
            i = 0;
      }
      if(i == 0){
            TCCR1B |= (1<<CS11);
            i = 1;
      }
      distance = pulse/58;
```

```
	if(distance < stop_dist){
		drive_motor(-100, -100);
		LED_ON;
	}
	else{
		drive_motor(l, r);
		LED_OFF;
	}
}
/**@}*/
#endif
```

serial/uart.h

```
#ifndef UART_H
#define UART_H

#define F_CPU 8000000UL		//!< clock speed of Atmega328p - 8MHz
#define BUFF_SIZE 25		//!< buffer size
#define BAUD  9600			//!< UART Baud Rate


#include <avr/io.h>
#include <util/delay.h>


/**
* @defgroup UARTfunct UART I/O
* @brief Enables the UART functionality in the microcontroller
* @{
*/

//! Initialize UART in the Atmega328p
volatile signed char receivedChar;
char charBuff[BUFF_SIZE];
volatile unsigned char charread;
int l, r;
char turnt = 0, done = 0;


void initUART(){
	unsigned int baudrate;

	// Set baud rate:  UBRR = [F_CPU/(16*BAUD)] -1
	baudrate = ((F_CPU/16)/BAUD) - 1;
	UBRR0H = (unsigned char) (baudrate >> 8);
	UBRR0L = (unsigned char) baudrate;

	UCSR0B |= (1 << RXEN0) | (1 << TXEN0);		// Enable receiver &
transmitter
	UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);  // Set data frame: 8 data
bits, 1 stop bit, no parity
}

//! Transmit/write one character to the output
void writeChar(unsigned char c) {
	UDR0 = c;		// Display character on serial (i.e., PuTTY)
terminal
```

```
	_delay_ms(10);		// delay for 10 ms
}

//! Transmit/write a NULL-terminated string to the output
void writestring(char *c){
	unsigned int i = 0;
	while(c[i] != 0)
		writeChar(c[i++]);
}

void readString(){
	done = 0;
	while (done == 0){
		if(UCSR0A & (1 << RXC0)){

			receivedChar = UDR0;			// Read the data
from the RX buffer
			charBuff[charread] = receivedChar;  // load char into
buffer

			if(receivedChar == '<')			// esp32 has
stopped sending gibberish
				return;

			else if(receivedChar == 'l')		// left value input
ready
				turnt = 1;

			else if(receivedChar == 'r')		// right value input
ready
				turnt = 0;

			else if(turnt){					// set
right
				r = (int)receivedChar;
				done = 1;
			}
			else{					// set left
				l = (int)receivedChar;
				done = 0;
			}
		}

	}
}

/**@}*/
#endif
```

sleep.h

```
#ifndef SLEEP_H
#define SLEEP_H

/**
* @defgroup soundIndic Sound Wall Collision Warning
* @brief Defines the use of the sound wall collision warning system
* @{
```

```
*/
#define SOUND_PORT                  PORTC5
      //!< sound port (PORTC5)
#define TONER_UPPER                 PORTC |= (1<<SOUND_PORT)      //!< tone
up sound (PORTC5)
#define TONE_DOWN_FOR_WHAT          PORTC &= ~(1<<SOUND_PORT)     //!< tone
down sound (PORTC5)
/**@}*/



void init_PCINT(){
      PCICR |= (1 << PCIE0);              // set PCIE0 to enable PCMSK0 scan
      PCMSK0 |= (1 << PCINT4);            // set PCINT4 to trigger an
interrupt on state change
      DDRC |= (1<<SOUND_PORT);            // sound indicator
      sei();
}

void good_night(){
      int i;
      for(i = 0; i < 600; i++){
            TONER_UPPER; //start tone
            _delay_ms(1);
            TONE_DOWN_FOR_WHAT; //mute tone
            _delay_ms(1);
      }

      set_sleep_mode(SLEEP_MODE_PWR_DOWN); // go to sleep
      cli();

      sleep_enable();
      sei();
      sleep_cpu();

      cli();
      PCICR |= (1 << PCIE0);     // set PCIE0 to enable PCMSK0 scan
      PCMSK0 |= (1 << PCINT4);   // set PCINT0 to trigger an interrupt on
state change
      sleep_disable();
      sei();
}
#endif
```

wifi.ino

```
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[] = "52f3c4e2349b42c68b94579fd313bf1e";

// Your WiFi credentials.
```

```
// Set password to "" for open networks.
char ssid[] = "";
char pass[] = "";

BLYNK_WRITE(V1){
  // variable declarations
  int x = param[0].asInt();         // x direction value
  int y = param[1].asInt();         // y direction value

  // turning data into percentages
  x /= 100;
  y /= 100;

  // left & right wheel percentage calculations
  x = -x;
  int v = (100-abs(x)) * y/100 + y;
  int w = (100-abs(y)) * x/100 + x;
  int r = (v+w) / 2;
  int l = (v-w) / 2;

  // sending joystick data onto Atmega328p
  Serial.write(l);
  Serial.write('l');
  Serial.write(r);
  Serial.write('r');
}

void setup(){
  Serial.begin(9600);                    // setup UART
  Blynk.begin(auth, ssid, pass);         // setup Wi-Fi authentication
  Serial.write('<');                     // send '<' for read ready
}

void loop(){
  Blynk.run();    // run Blynk
}
```