Remote Controlled Robot

David Pajar

Rimon Sawa

**Goal:**

* Control a robot using RF control
* Implement an AVR and RF module
* Move robot forward, reverse, left, right and stop

**Deliverables:**

Remote controlled robot based on ATmega328P + RF + PC

The goal of this project will be to control a zumo robot (provided fully assembled) using RF control. The zomo robot will be equipped and controlled with an AVR + RF module. The host system will consist of an AVR+RF module connected to the PC through UART interface. Commands will be issued from the PC through the terminal or visual GUI to move the robot move forward, reverse, left, right and stop.

# I. COMPONENTS

## A. ATmega328p

The ATmega328p Xplained Mini is an 8-bit microcontroller from Atmel that can be programmed with either Atmel Studio or Arduino IDE using C/C++, AVR assembly, or Arduino. The code will allow users to manipulate the board’s 32 GPIOs, 6 PWMs, 3 timer/counters, 1kB EEPROM, 2kB SDRAM, I2C interface, SPI interface, 10-bit ADCs, and up to an internal 16 MHz clock.

## B. RF Remote

This 4-button keyfob remote goes with our three basic 315MHz RF receiver modules. It will work with all of them, sending out one of four commands that match up with the four outputs. Its small and light weight and will work up to 25 feet away depending on line-of-sight and obstructions. Since this is just a transmitter, if you have multiple receivers, it will turn all of them on and off at the same time (there is no sub-addressing)

## C. Gear motors

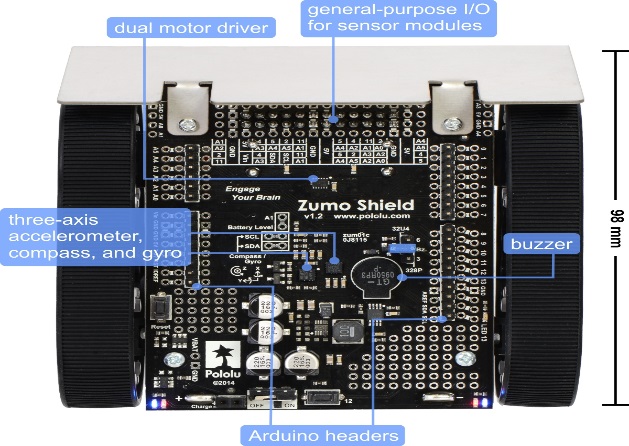
## D. 2 \* 3.7V battery

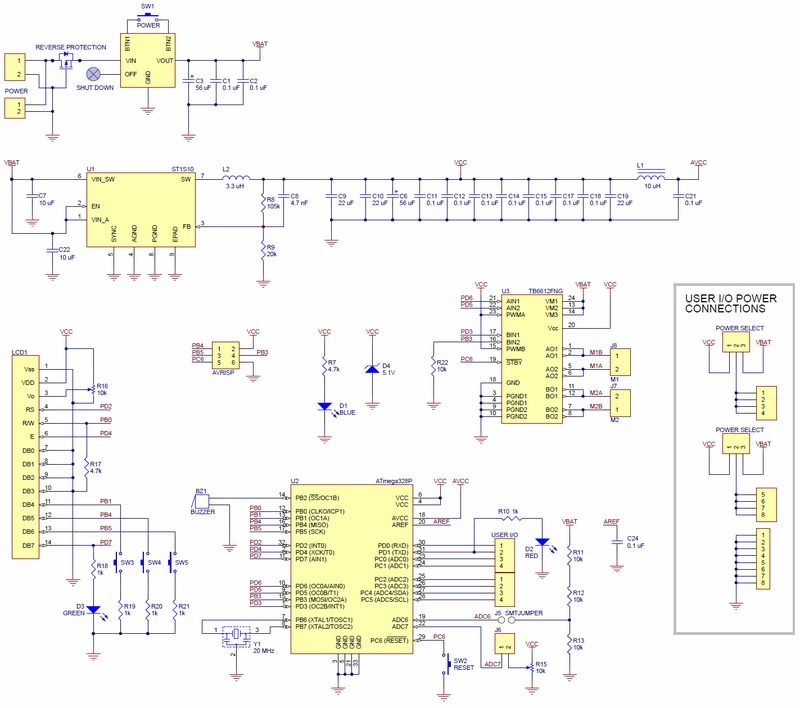
To power microcontroller as well as motor module.

**E. Model X Motor Driver module**

Module is used to connect all four DC motors and to be able to control them.

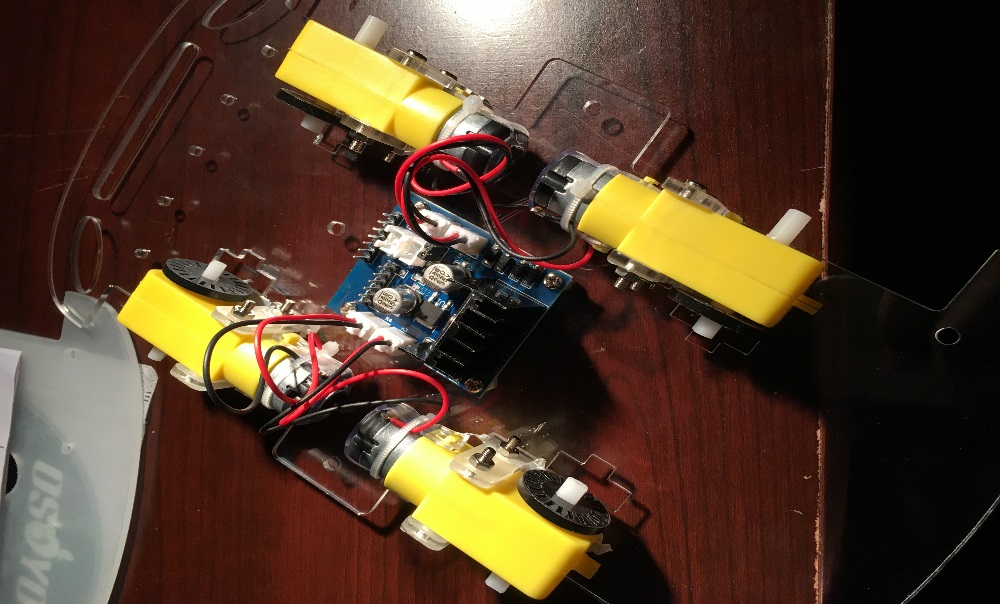
III. Schematics





# IV. IMPLEMENTATION

Implementing the RF module to work with the atmega328p board. Once configured, we are able to control the robot.

V. SNAPSHOTS AND SCREENSHOTS 

VI. Code

Main:

#define F\_CPU 8000000UL

/\*\*

\*/

#define F\_CPU 8000000UL //Set clk speed to 8 Mhz

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdlib.h>

#include "drive\_motor.h"

#include "uart.h"

int main(void){

// initialize WIFI

initUART();

init\_motor();

}

Drive Motor

#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);

// 255

TCCR0A = (1<<COM0A1)|(1<<COM0B1)|(1<<WGM00)|(1<<WGM01); // non-inverting MODE

TCCR0B = (1<<CS00);

TCCR2A = (1<<COM2A1)|(1<<COM2B1)|(1<<WGM20)|(1<<WGM21); TCCR2B = (1<<CS20);

}

//forwards and backwards movement

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;

OCR0B = (tempL)<255?tempL:255; // change duty cycle

OCR2A = 0;

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

}

else if (tempL>=0 && tempR<0){

OCR0A = 0;

OCR0B = (tempL)<255?tempL:255; // change duty cycle

OCR2A = (-tempR)<255?(-tempR):255;

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

Uart.h

#ifndef UART\_H

#define UART\_H

#define F\_CPU 8000000UL //set clk speed to 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 RX

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

# VII. LINKS

David Pajar github: [www.github.com/pajard1](http://www.github.com/pajard1)

Rimon Sawa github: [www.github.com/rsawa91](http://www.github.com/rsawa91)

Youtube Video:

Demo: <https://youtu.be/EYaKe_ahoAY>

Presentation: https://www.youtube.com/watch?v=kU5Ig5Q5mPs

## VIII. CONCLUSION

We were able to control the robot using the atmega328p along with the RF module. We displayed the movements moving forward, backward, left, right and stopping.