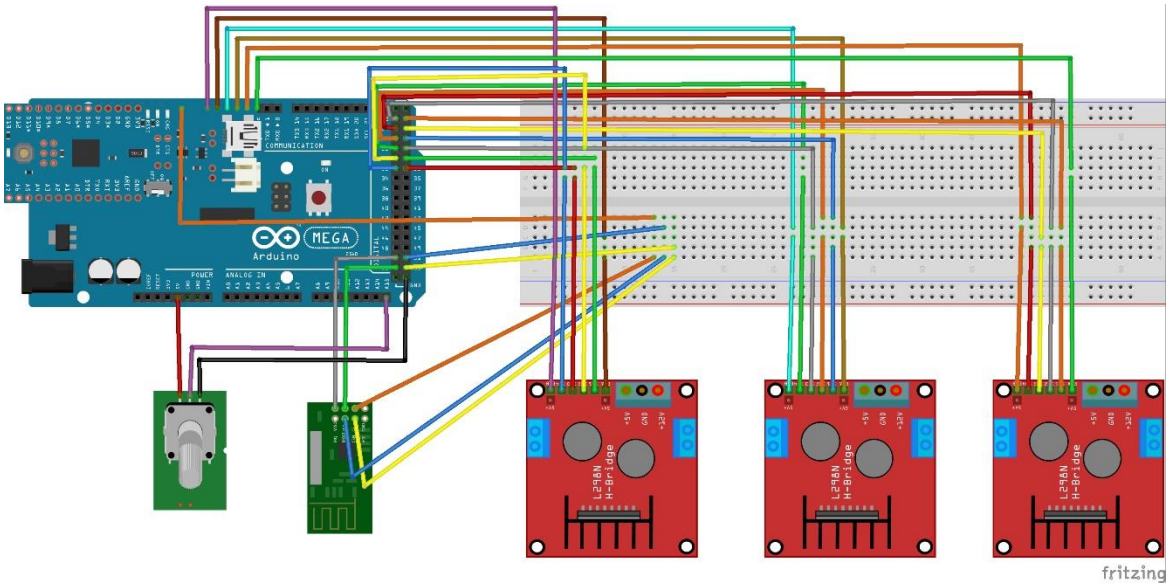
Proabot Boat Circuit Design and Program

Circuit Design:

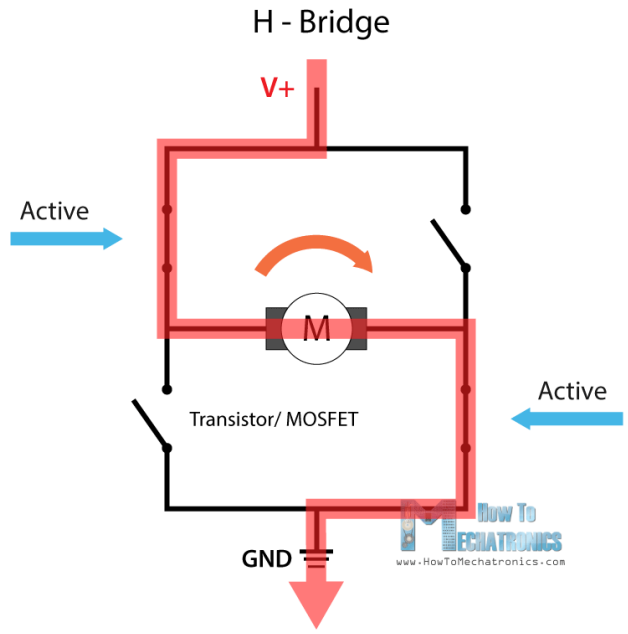


Components:

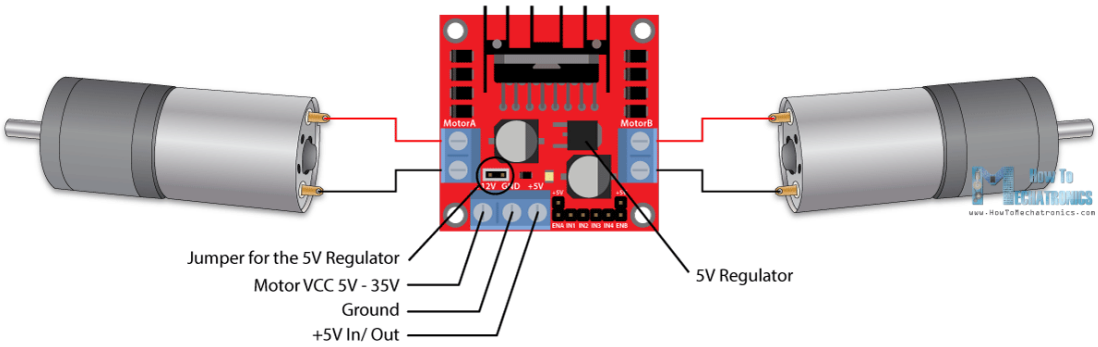
Arduino Mega 2560
L298N H-bridge DC motor driver *3
NRF24
Potentiometer

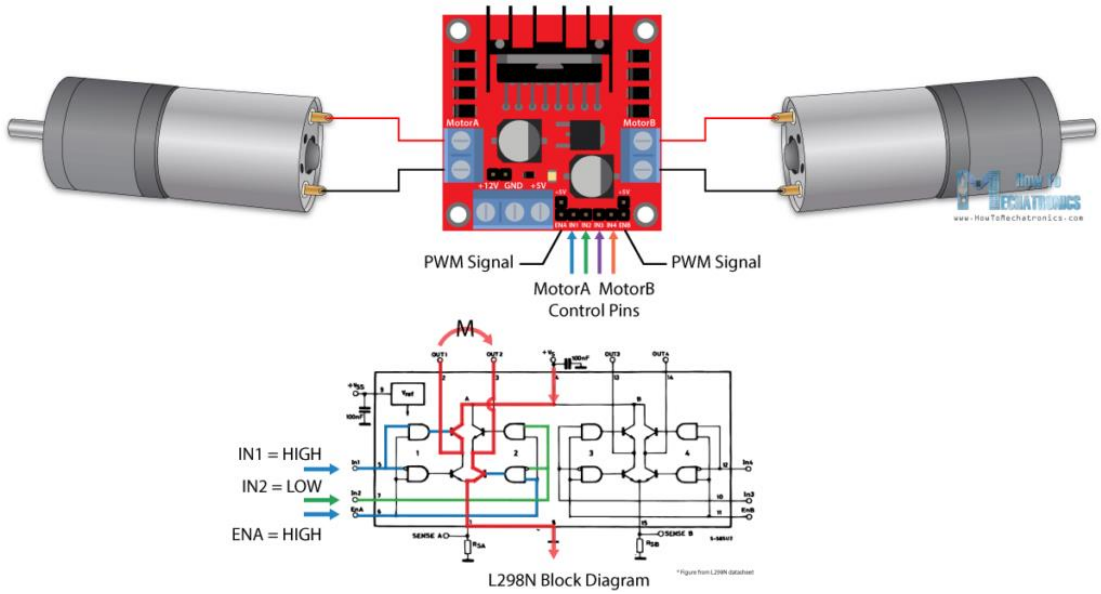
L298N H-bridge

Data Sheet: https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf
H-bridge:



L298N pins:





L298N Block Diagram

// The circuit uses 3 L298N to drive 6 motors.

Boat Program

Classes:

1. Motor
2. Rudder

Library:

1. <SPI.h>
2. <RF24.h>
3. <nRF24L01.h>
4. <PID_v1.h>

Class Motor:

```
/*
 * Motor.h
 *
 *  Created: Jan, 2018
 *      Author: Steven Hu
 *  >>Project Orthogonal  -Proabot
 *
 *  Known Issues:
 */

#ifndef MOTOR_H
#define MOTOR_H

#define DIRECT 0
#define REVERSE 1
#define STOP 5
#define HIGH_CURRENT 7

#define CURRENT_THRESHOLD 150

class Motor {
public:
    Motor(int pwm_p, int in1_p, int in2_p, int current_p);
    void set_pwm(int pwm);
    void change_to_direct();
    void change_to_reverse();
    void stop_motor();
    void sense_current();
    void ask_reboot();
```

```
    int get_direction();

/*
 * don't need to access pwm value when operating
 *motor, so pwm value is stored in rudder objects
 *as power
 */
private:
  int in1_p;
  int in2_p;
  int pwm_p;
  int direction;
  int current_p;

};

#endif /* MOTOR_MOTOR_H */
```

```
/*
 * Motor.cpp
 *
 *  Created: Jan, 2018
 *      Author: Steven Hu
 *  >>Project Orthogonal  -Proabot
 *
 *  Known Issues:
 *  1.current sensing is enforced. no way to bypass it. it does give an
option to restart the motor, but
 *    if issue remains, it will be stopped at the next cycle. should add a
method to bypass it.
 *  2.current reading for different motors are different.
 *
 */

#include <Arduino.h>
#include "Motor.h"



Motor::Motor(int pwm_p,int in1_p,int in2_p,int current_p)
    :in1_p(in1_p),in2_p(in2_p),pwm_p(pwm_p),direction(STOP),
current_p(current_p)
{
```

```
  //set up
  pinMode(pwm_p,OUTPUT);
  pinMode(in1_p,OUTPUT);
  pinMode(in2_p,OUTPUT);
  pinMode(current_p,INPUT);
}

void Motor::set_pwm(int pwm){
    if (direction!=HIGH_CURRENT)
        analogWrite(pwm_p,pwm);
}

void Motor::change_to_direct(){
    if(direction!=HIGH_CURRENT){
          digitalWrite(in1_p,HIGH);
          digitalWrite(in2_p,LOW);
          direction = DIRECT;
    }

}

void Motor::change_to_reverse(){
    if (direction!=HIGH_CURRENT){
        digitalWrite(in1_p,LOW);
        digitalWrite(in2_p,HIGH);
        direction = REVERSE;
    }
}

int Motor::get_direction(){
  return direction;
}

void Motor::stop_motor(){
  set_pwm(0);
  digitalWrite(in2_p,LOW);
  digitalWrite(in1_p,LOW);
  direction = STOP;
}

void Motor::sense_current(){
    int current = analogRead(current_p);
    Serial.print("\t# Current: ");
    Serial.println(current);
```

```
    if(direction!= HIGH_CURRENT && current>CURRENT_THRESHOLD){
        stop_motor();
        direction = HIGH_CURRENT;
        Serial.println("WARNING: HIGH_CURRENT[Motor Stopped.]");
    }

}

void Motor::ask_reboot()
{
    Serial.println("[R]eboot? ");
    if(Serial.read()=='R')
        direction = STOP;
}
```

Class Rudder:

```
/*
 * Rudder.h
 *
 *  Created: Jan, 2018
 *      Author: Steven Hu
 *  >>Project Orthogonal  -Proabot
 *
 *  Known Issues:
 *  1. motor and pid should be private. making them public gives too much
 *     control to the main script. but it's difficult to calibrate and
 *     experiment things with an extra method. They will stay public for
now.
 */

#ifndef RUDDER_H
#define RUDDER_H

#include "Motor.h"
#include <PID_v1.h>

class Rudder {
private:
    int position_p;
    double setpoint;
    double output;
    double position;
public:
```

```
    Rudder (int position_pin, int pwm_p, int in1_p, int in2_p, int
current_p);

    int choose_direction();

    void update_setpoint(double setpoint);
    void update_position();

    void Compute_and_Drive();

    int get_position();
    int get_output();
    int get_stat();
    void set_direction(int direction);
    //void check_current();

    void print();
    ~Rudder();
    Motor * motor;
    PID * pid;
};

#endif /* RUDDER_H */
```

```
/*
 * Rudder.cpp
 *
 *  Created: Jan, 2018
 *  Update:  April 16, 2018
 *      Author: Steven Hu
 *  >>Project Orthogonal  -Proabot
 *  Known Issues:
 *    1. position is given a voltage value which is not useful for display,
should
 *       add a mapping function to return degrees.
 *    2. If NRF24 doesn't receive anything, it will cause the whole system
to fail.
 *
 */

#include "Rudder.h"
#include <Arduino.h>

#define GOOD 15
```

```
#define CURRENT_SENSING_DOWN 20
#define POSITION_SENSING_DOWN 21
#define CURRENT_WARNING
#define TOLERATE 5
#define POSITION_LOWER_LIMIT  550
#define POSITION_UPPER_LIMIT  774

//Three parameters for PID controller
// For our purpose, we Kp should be in range(10,15)
double Kp=15, Ki=0, Kd=0;

/*
 * setpoint is set to 0 in the code. But 0 is actually not in the range of
 * possible values for the position. This value will be updated to a
 * correct value in the first loop.
 * SampleTime, OutputLimits are two settings that can be customized.
 */
Rudder::Rudder(int position_p, int pwm_p, int in1_p, int in2_p, int
current_p)
    :position_p(position_p), setpoint(0), output(0), position(0),
     motor(new Motor(pwm_p,in1_p,in2_p,current_p)),
     pid(new PID(&position, &output, &setpoint, Kp,Ki,Kd,DIRECT))

{
    update_position();
    pid->SetSampleTime(3000);
    pid->SetOutputLimits(-135,135);
    pid->SetMode(AUTOMATIC);

}
/*
 * analog value will not be stable. We could smooth it out
 * by assigning an average value of a consecutive 10 msec. But
 * since we are only checking it every thousand msec, that might
 * not be necessary. Note: delay cannot be used anywhere in the
 * program because it will cause the NRF module to stop working.
 * which essentially stops the program.
 */
void Rudder::update_position(){

    position = analogRead(position_p);
}

/*
```

```
 * Function is called after setpoint is received by the boat.
 * setpoint should be updated before Compute_and_Drive()
 */
void Rudder::update_setpoint(double setpoint_received){
    setpoint = setpoint_received;
}

/*
 * The function that calculates the output and operates the motor.
 * HIGH_CURRENT will be given to dir if during the last current
 * sensing process, a high current is detected. Serial monitor
 * will give an option to reboot the motor. Otherwise, the motor
 * will not move under any circumstances.
 *  If the difference between setpoint and position is smaller than
 * the TOLERATE range (see #define), motor is stopped.
 *  POSITION_UPPER_LIMIT gives the analog value range of the motor
 * potentiometer. Any position (value) that exceeds the range established
 * by UPPER and LOWER LIMIT will cause the motor to stop.
 */
void Rudder::Compute_and_Drive(){
    int dir = motor->get_direction();
    if (dir==HIGH_CURRENT){
        pid->SetMode(MANUAL);
        output = 0;
        motor->ask_reboot();
    }
       //compute_and_drive only operates when NOT(HIGH_CURRENT)
    else{
        update_position();
        if (abs(setpoint-position) < TOLERATE){
            motor->stop_motor();
            pid->SetMode(MANUAL);
            output = 0;
        }

        else{
            pid->SetMode(AUTOMATIC);
            pid->Compute(); //output is computed
            if (output<0){
                if(position<POSITION_LOWER_LIMIT+TOLERATE){
                    motor->stop_motor();
                    pid->SetMode(MANUAL);
                    output = 0;
                }
```

```
                else{
                    motor->change_to_reverse();
                    motor->set_pwm(-output);
                }
            }
            else{ //logical error
                if(position>POSITION_UPPER_LIMIT-TOLERATE){
                    motor->stop_motor();
                    pid->SetMode(MANUAL);
                    output = 0;
                }
                else{
                    motor->change_to_direct();
                    motor->set_pwm(output);
                }
            }
        }
    }
}

void Rudder::print(){
    Serial.print("\tSetpoint: ");
    Serial.println(setpoint);
    Serial.print("\tPosition: ");
    Serial.println(position);
    Serial.print("\tOutput: ");
    Serial.println(output);
    Serial.print("\t# Direction: ");
    Serial.println(motor->get_direction());
    // current_sensing is printed inside it's method. It should be called
right after this so that you can see which current it's showing. Read
issues for more detail.
}

int Rudder::get_position(){
    //update_position();
    return position;
}

int Rudder::get_output(){
  return output;
}

int Rudder::get_stat(){
```

```
    return motor->get_direction();
}

Rudder::~Rudder() {
    delete motor;
    delete pid;
}
```

Boat Progarm:

```
/*
 * boat_program_spring18.ino    March 2018
 * Author: Steven, Asis
 * >> Implemented:
 *   1.nrf  2.PID
 */
#include "Arduino.h"
#include <SPI.h>
#include <RF24.h>
#include <nRF24L01.h>
#include <PID_v1.h>
#include "Rudder.h"
#include "Motor.h"

double data_to_send[4] = {0,50, 1,1};
//[0] := r1_position, [1] := r2_position, [2] := r1_current, [3] :=
r2_current
double received_data[2];   //received_data[0] = r1_setpoint, [1] =
r2_setpoint

RF24 radio(9,53);
byte addresses[][6] = {"1Node","2Node"};
  //names of the two communication(2 directions)

Rudder r1(15,5,28,29,3);
Rudder r2(1,1,1,1,1);

void setup()
{
    Serial.begin(9600);
    pinMode(47,OUTPUT); //give pot power
    digitalWrite(47, HIGH);

    //initiate the radio object
    radio.begin();
```

```
    //Set the transmit power to lowest available to prevent power supply
related
    //issues
    radio.setPALevel(RF24_PA_MIN);

    //Set the speed of the transmission to the quickest available
    radio.setDataRate(RF24_2MBPS);

    //Use a channel unlikely to be used by Wifi, Microwave ovens etc
    radio.setChannel(124);

    //Open a writing and reading pipe on each radio, with opposite
addresses
    radio.openWritingPipe(addresses[0]);
    radio.openReadingPipe(1,addresses[1]);

}

void update_data(){
    r1.update_setpoint(received_data[0]);
    r2.update_setpoint(received_data[1]);
    data_to_send[0] = r1.get_position();
    data_to_send[1] = 50;
    if (r1.get_stat()==HIGH_CURRENT)
        data_to_send[2] = 7;

}
void operate_rudder(Rudder & r){
    r.update_position();
    r.Compute_and_Drive();
    r.print();
    r.motor->sense_current();
}

void loop()
{
    radio.startListening();
    unsigned long start_wait_time = millis();
    while(!radio.available()){
        if(millis() - start_wait_time > 5){
            Serial.println("Nothing received!");
            return;
        }
    }
```

```
    radio.read(&received_data, sizeof(received_data));
    Serial.print("Received: ");
    Serial.println(received_data[0]);

    update_data(); //update setpoint and data_to_send

    radio.stopListening();

    operate_rudder(r1);
    //operate_rudder(r2);

    radio.write(&data_to_send, sizeof(data_to_send));
    Serial.print("Sent: ");
    Serial.println(data_to_send[0]);
}
```