

How to use a STM32F4Discovery MMOS controller together with a cheap servo driver and motor.

Video with working system at <https://youtu.be/-2qgV8PRJo4>

Download essential files for the project (all freeware/demos) at
<https://mega.nz/#!t2RnzLIK!UW6Pfvne498bXAKCqI2CpsQBpc2ApDAcoz7eL2U-XU>

Homepage of the MMOS force feedback controller:
<https://forum.virtualracing.org/showthread.php/92420-DIY-USB-Force-Feedback-Controller>

Why did I make this? Wanting to make a cheap direct drive FFB wheel, I went and bought a very cheap closed loop driver (Leadshine HBS86H clone) and motor set on Taobao (about 100 USD altogether). The plan was to use it together with the MMOS Controller (<https://forum.virtualracing.org/showthread.php/92420-DIY-USB-Force-Feedback-Controller>). But it soon became clear that the two weren't compatible. The MMOS controller outputs a PWM signal to indicate the level of torque that the driver should produce, but the cheap driver I bought doesn't accept that sort of signal. It only accepts a binary direction (DIR) signal and a pulse signal (PUL) that indicates how many steps the motor should rotate. Torque can only be set by sending hex-commands to the driver via a serial connection. This is usually done via a serial connection with a PC using a setup program called Protuner.

The solution to this problem was to send the PWM control signal from the MMOS controller (Disco) to an Arduino, which would then convert the signal to something that the driver can accept. So the Arduino measures the length of the PWM signal from the Disco (which indicates the amount of torque needed) and sends a corresponding command to the driver via serial connection. Furthermore, every time the direction of the torque requested from the Disco changes, it changes the DIR signal that is sent to the driver and sends a number of pulses (PUL), so that the driver makes the motor pull in the right direction.

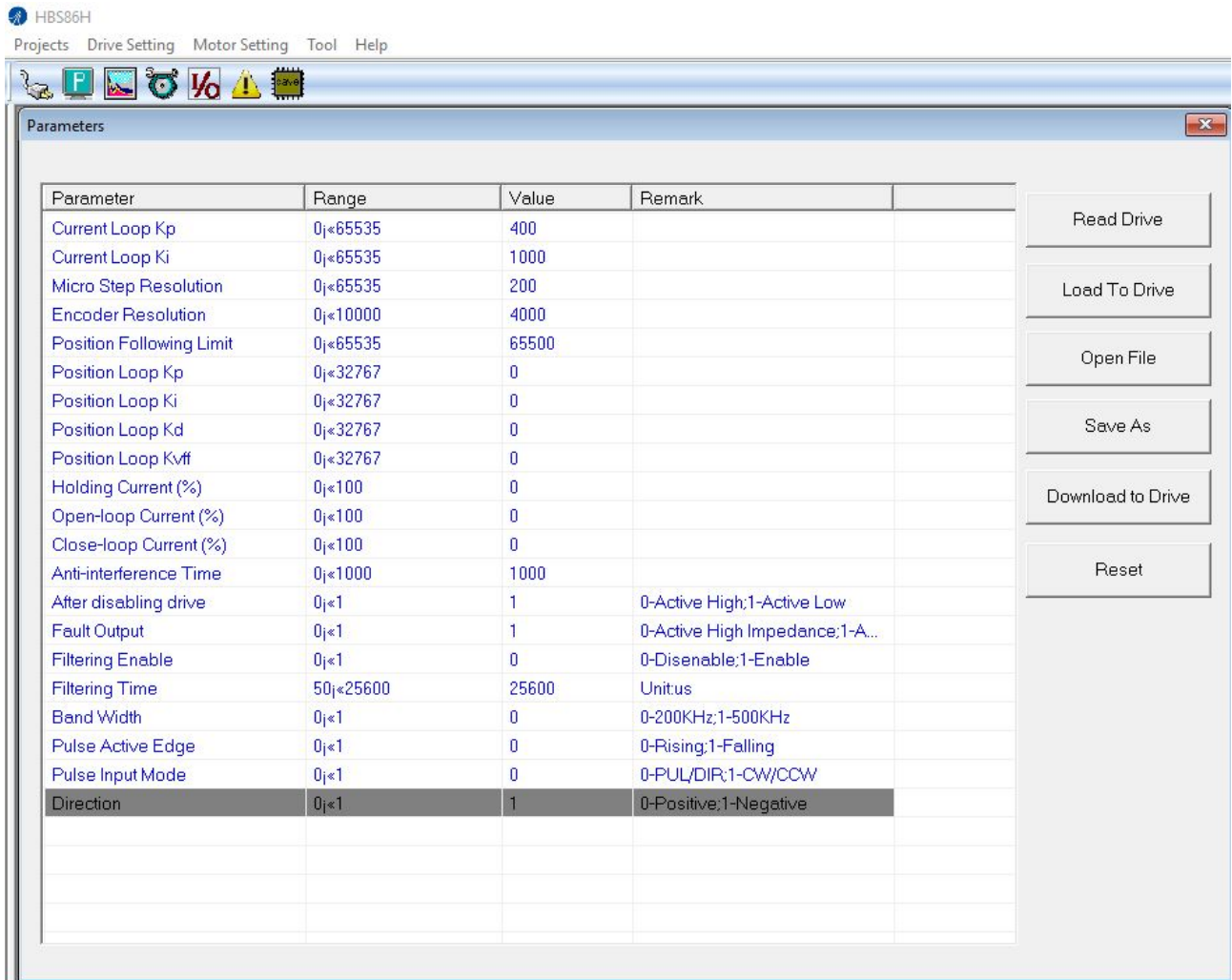
Setting up the system

using a Leadshine HBS86H driver or possibly a ES-D1008 driver (they should be similar)

Driver + motor + encoder

Connect motor, encoder and driver according to instruction manual.

Install the Protuner software, connect your PC via serial to the driver and set the following values:



Press 'Download to Drive' to save the values permanently on the driver. Turn the driver off and on, press 'Read Drive' to check if the values have been saved properly on the driver.

STM32F4Discovery

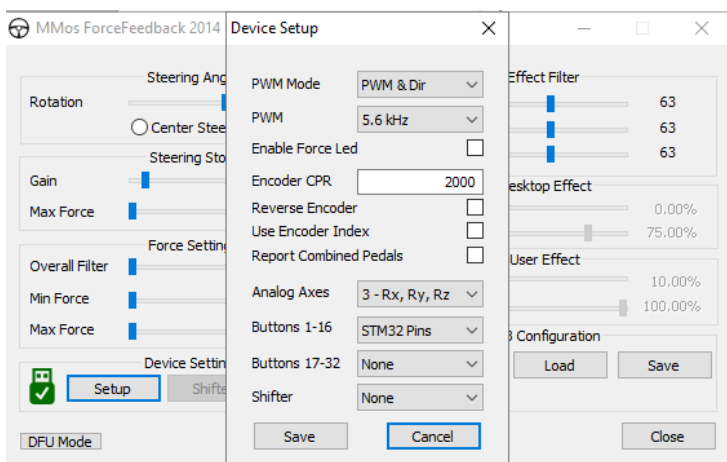
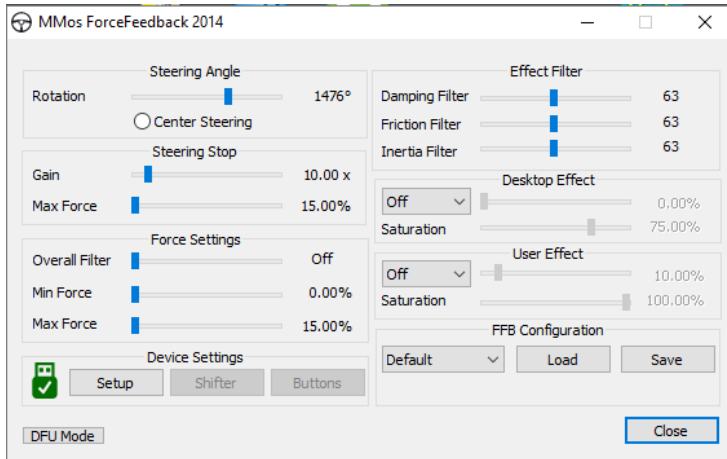
Install the MMOS controller firmware on the STM32F4Discovery board as described in 'OpenSimwheel-Tutorial.pdf' section 4.4:

1. connect **mini-USB** cable to Disco board
2. install **STM32 ST-LINK utility**
3. connect
4. erase chip
5. program and verify
6. select MMos hex-file
7. start flashing
8. disconnect
9. unplug **mini-USB** cable

Connect both USB connections on the Discovery board to a PC.

Run the program MMosForceFeedback2014.exe. Check that the icon in the lower left corner is green.

Set the below settings



Arduino

If you are not using a Leadshine HBS86H driver or another compatible driver, the hex-commands in the below sketch will probably not work. In that case you will need to connect the driver to a PC using the serial connection, use the Protuner software to send the appropriate torque commands to the driver and use a program like Free Serial Analyzer (<https://freeserialanalyzer.com/>) to detect and copy the hex commands that are sent via serial to the driver. Then these hex-commands should be inserted in the section marked **red** in the below sketch. Please note that in the individual hex numbers must start with 0x, so the values obtained from Serial Analyzer may need to be amended.

Please note that for each torque level two commands are sent: One for adjusting the value Position Loop Kp and another for adjusting the value Close-Loop Current (called `CloLoopxx` and `Kpxxxx` in the below sketch). This gives the quickest and most dynamic response.

Here's the sketch that I'm using with my HBS86H driver:

```

// Interface for using the MMOS controller software with a cheap driver+motor.
// Connects between the STM32F4Discovery and the servo driver.
// More information on the MMOS FFB software on
https://forum.virtualracing.org/showthread.php/92420-DIY-USB-Force-Feedback-Controller
// The torque hex commands in this sketch work with the Leadshine HBS86H driver and possibly ES-
D1008 (they should be similar).

// Disco pin PA0 to driver EA+,
// Disco pin PA1 to driver EB+
// Disco GND to driver EGND
// Disco GND to Arduino GND
// Arduino pin 3 to Disco pin PE9
// Arduino pin 4 to Disco pin PE11
// Arduino pin 9 to servo driver PUL+
// Arduino pin 13 to servo driver DIR+
// Arduino GND to servo driver PUL- and DIR-

// micro step resolution must be set to 200

byte PWM_OUT_PIN = 3;
int pwm_value;
int pwm_valueSum;
int pwm_value1;
int pwm_value2;
int pwm_value3;
int pwm_value4;
int pwm_value5;
int pwm_value6;
int pwm_value7;
int pwm_value8;
int pwm_value9;
int pwm_value0;

int delayshort = 1;
int delaylong = 8;

byte DIR_IN_PIN = 4;
int Dir1 = 0;
int Dir2 = 0;
byte STEP_OUT_PIN = 9;
byte DIR_OUT_PIN = 13;

void setup()
{
  Serial.begin(9600);
  pinMode(PWM_OUT_PIN, INPUT);
  pinMode(DIR_IN_PIN, INPUT);
  pinMode(STEP_OUT_PIN, OUTPUT);
  pinMode(DIR_OUT_PIN, OUTPUT);

  // initialization

  delay(7000); // wait seven seconds while the user turns on the driver
  byte Kp000 [] = {0x01, 0x06, 0x00, 0x06, 0x00, 0x00, 0x69, 0xCB}; // set the wheel force to zero

  Serial.write(Kp000, 8);

  Serial.println("");

  // set the wheel 1000 steps offcenter
  for(int i=0; i < 1000; i++){
    digitalWrite(STEP_OUT_PIN, HIGH);
    delayMicroseconds(6);
    digitalWrite(STEP_OUT_PIN, LOW);
    delayMicroseconds(6);
  }
}

void loop() {

```

```

Dir2=Dir1;
Dir1=digitalRead(DIR_IN_PIN);

// take 10 samples of the length of the PWM signal coming from the Disco
pwm_value0 = pulseIn(PWM_OUT_PIN, HIGH);
delay (delayshort);
pwm_value1 = pulseIn(PWM_OUT_PIN, HIGH);
delay (delayshort);
pwm_value2 = pulseIn(PWM_OUT_PIN, HIGH);
delay (delayshort);
pwm_value3 = pulseIn(PWM_OUT_PIN, HIGH);
delay (delayshort);
pwm_value4 = pulseIn(PWM_OUT_PIN, HIGH);
delay (delayshort);
pwm_value5 = pulseIn(PWM_OUT_PIN, HIGH);
delay (delayshort);
pwm_value6 = pulseIn(PWM_OUT_PIN, HIGH);
delay (delayshort);
pwm_value7 = pulseIn(PWM_OUT_PIN, HIGH);
delay (delayshort);
pwm_value8 = pulseIn(PWM_OUT_PIN, HIGH);
delay (delayshort);
pwm_value9 = pulseIn(PWM_OUT_PIN, HIGH);
delay (delayshort);

//make an average of the 10 values
pwm_valueSum = pwm_value0 + pwm_value1 + pwm_value2 + pwm_value3 + pwm_value4 + pwm_value5 +
pwm_value6 + pwm_value7 + pwm_value8 + pwm_value9 ;
pwm_value = pwm_valueSum / 10;

// determine if the directional signal from the Disco has changed, if it has, then set torque to
zero and tell driver to turn 2000 steps in the new direction.
if (Dir2!=Dir1) {
    byte Kp000 [] = {0x01, 0x06, 0x00, 0x06, 0x00, 0x00, 0x69, 0xCB};
    byte CloLoop00 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x00, 0x28, 0x1B};

    Serial.write(Kp000, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop00, 8);
    Serial.println("");

    digitalWrite(DIR_OUT_PIN, Dir1);

    for(int i=0; i < 2000; i++){

        digitalWrite(STEP_OUT_PIN, HIGH);
        delayMicroseconds(6);
        digitalWrite(STEP_OUT_PIN, LOW);
        delayMicroseconds(6);
    }
}

// Define the level of torque for a given length of the pwm signal.
int Level00=0;
int Level01=2;
int Level02=5;
int Level03=7;
int Level04=10;
int Level05=12;
int Level06=14;
int Level07=17;
int Level08=19;
int Level09=22;
int Level10=24;
int Level11=26;
int Level12=29;
int Level13=31;
int Level14=33;
int Level15=36;
int Level16=38;
int Level17=41;

```

```

int Level18=43;
int Level19=45;
int Level20=48;
int Level21=50;
int Level22=53;

// define the hex commands that will be sent to the driver via serial to control the torque

byte CloLoop00 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x00, 0x28, 0x1B};
byte CloLoop10 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x0A, 0xA8, 0x1C};
byte CloLoop20 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x14, 0x28, 0x14};
byte CloLoop30 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x1E, 0xA8, 0x13};
byte CloLoop35 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x23, 0x69, 0xC2};
byte CloLoop40 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x28, 0x28, 0x05};
byte CloLoop45 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x2D, 0xE8, 0x06};
byte CloLoop50 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x32, 0xA9, 0xCE};
byte CloLoop55 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x37, 0x69, 0xCD};
byte CloLoop60 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x3C, 0x28, 0x0A};
byte CloLoop70 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x46, 0xA9, 0xE9};
byte CloLoop80 [] = {0x01, 0x06, 0x00, 0x52, 0x00, 0x50, 0x28, 0x27};
byte Kp0000 [] = {0x01, 0x06, 0x00, 0x06, 0x00, 0x00, 0x69, 0xCB};
byte Kp0100 [] = {0x01, 0x06, 0x00, 0x06, 0x00, 0x64, 0x68, 0x20};
byte Kp0200 [] = {0x01, 0x06, 0x00, 0x06, 0x00, 0xC8, 0x68, 0x5D};
byte Kp0300 [] = {0x01, 0x06, 0x00, 0x06, 0x01, 0x2C, 0x69, 0x86};
byte Kp0400 [] = {0x01, 0x06, 0x00, 0x06, 0x01, 0x90, 0x68, 0x37};
byte Kp0450 [] = {0x01, 0x06, 0x00, 0x06, 0x01, 0xC2, 0xE9, 0xCA};
byte Kp0500 [] = {0x01, 0x06, 0x00, 0x06, 0x01, 0xF4, 0x69, 0xDC};
byte Kp0550 [] = {0x01, 0x06, 0x00, 0x06, 0x02, 0x26, 0xE9, 0x71};
byte Kp0600 [] = {0x01, 0x06, 0x00, 0x06, 0x02, 0x58, 0x69, 0x51};
byte Kp0650 [] = {0x01, 0x06, 0x00, 0x06, 0x02, 0x8A, 0xE9, 0x0C};
byte Kp0700 [] = {0x01, 0x06, 0x00, 0x06, 0x02, 0xBC, 0x69, 0x1A};
byte Kp0800 [] = {0x01, 0x06, 0x00, 0x06, 0x03, 0x20, 0x68, 0xE3};
byte Kp0900 [] = {0x01, 0x06, 0x00, 0x06, 0x03, 0x84, 0x69, 0x58};
byte Kp1000 [] = {0x01, 0x06, 0x00, 0x06, 0x03, 0xE8, 0x69, 0x75};
byte Kp1200 [] = {0x01, 0x06, 0x00, 0x06, 0x04, 0xB0, 0x6A, 0xBF};
byte Kp1500 [] = {0x01, 0x06, 0x00, 0x06, 0x05, 0xDC, 0x6B, 0x02};
byte Kp1600 [] = {0x01, 0x06, 0x00, 0x06, 0x06, 0x40, 0x6B, 0x9B};
byte Kp3200 [] = {0x01, 0x06, 0x00, 0x06, 0x0C, 0x80, 0x6D, 0x6B};

// send the torque commands that correspond to the length of the pwm signal.

if ( pwm_value <= Level100) {

    Serial.write(Kp0000, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop00, 8);
    Serial.println("");

}

if (pwm_value > Level100  && pwm_value <= Level101) {

    Serial.write(Kp0100, 8);
    Serial.println("");
    delay (delaylong); // The drive can only accept about 50 commands per second. This delay
ensures that both commands are applied.
    Serial.write(CloLoop10, 8);
    Serial.println("");

}

if (pwm_value > Level101  && pwm_value <= Level102) {

    Serial.write(Kp0200, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop10, 8);
    Serial.println("");

```

```

}
if (pwm_value > Level02  && pwm_value <= Level03) {

    Serial.write(Kp0200, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop20, 8);
    Serial.println("");

}
if (pwm_value > Level03  && pwm_value <= Level04) {
    Serial.write(Kp0300, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop20, 8);
    Serial.println("");

}
if (pwm_value > Level04  && pwm_value <= Level05) {

    Serial.write(Kp0300, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop30, 8);
    Serial.println("");

}
if (pwm_value > Level05  && pwm_value <= Level06) {
    Serial.write(Kp0400, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop30, 8);
    Serial.println("");

}

if (pwm_value > Level06  && pwm_value <= Level07) {
    Serial.write(Kp0400, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop35, 8);
    Serial.println("");

}

if (pwm_value > Level07  && pwm_value <= Level08) {
    Serial.write(Kp0450, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop35, 8);
    Serial.println("");

}
if (pwm_value > Level08  && pwm_value <= Level09) {
    Serial.write(Kp0450, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop40, 8);
    Serial.println("");

}
if (pwm_value > Level09  && pwm_value <= Level10) {
    Serial.write(Kp0500, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop40, 8);
    Serial.println("");

}

```

```

if (pwm_value > Level10  && pwm_value <= Level11) {
    Serial.write(Kp0500, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop45, 8);
    Serial.println("");
}

if (pwm_value > Level11  && pwm_value <= Level12) {

    Serial.write(Kp0600, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop45, 8);
    Serial.println("");
}
if (pwm_value > Level12  && pwm_value <= Level13) {
    Serial.write(Kp0600, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop50, 8);
    Serial.println("");
}
if (pwm_value > Level13  && pwm_value <= Level14) {

    Serial.write(Kp0700, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop50, 8);
    Serial.println("");
}
if (pwm_value > Level14  && pwm_value <= Level15) {
    Serial.write(Kp0700, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop55, 8);
    Serial.println("");
}

if (pwm_value > Level15  && pwm_value <= Level16) {
    Serial.write(Kp0800, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop55, 8);
    Serial.println("");
}

if (pwm_value > Level16  && pwm_value <= Level17) {
    Serial.write(Kp0800, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop60, 8);
    Serial.println("");
}

if (pwm_value > Level17  && pwm_value <= Level18) {
    Serial.write(Kp0900, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop60, 8);
    Serial.println("");
}

```



```

if (pwm_value > Level18  && pwm_value <= Level19) {
    Serial.write(Kp0900, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop70, 8);
    Serial.println("");

}

if (pwm_value > Level19  && pwm_value <= Level20) {
    Serial.write(Kp1000, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop70, 8);
    Serial.println("");

}


if (pwm_value > Level20  && pwm_value <= Level21) {
    Serial.write(Kp1000, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop80, 8);
    Serial.println("");

}


if (pwm_value > Level21  && pwm_value <= Level22) {
    Serial.write(Kp1200, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop80, 8);
    Serial.println("");

}


if (pwm_value > Level22 ) {

    Serial.write(Kp1600, 8);
    Serial.println("");
    delay (delaylong);
    Serial.write(CloLoop80, 8);
    Serial.println("");

}
}

```

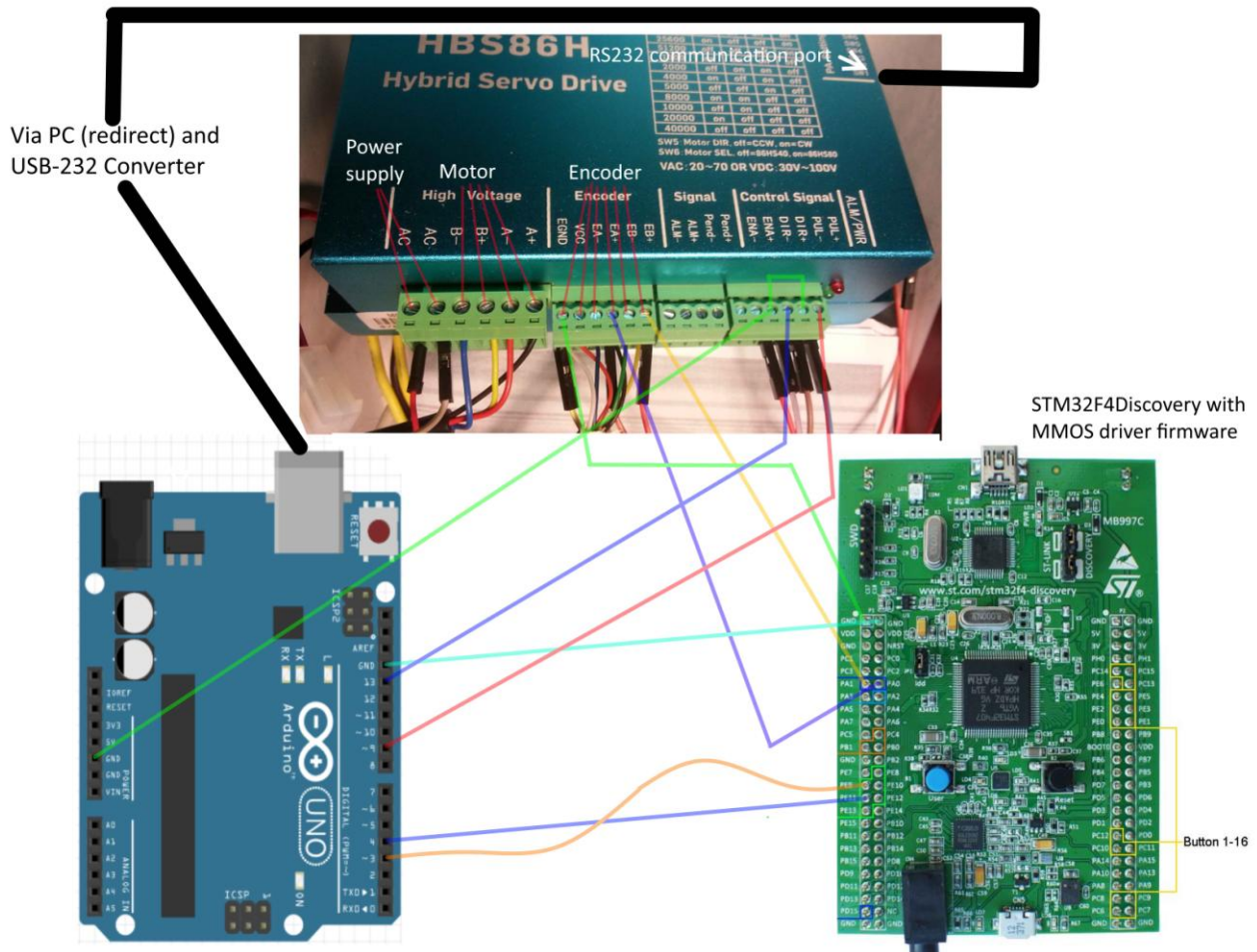
Upload the sketch to a standard Arduino Uno.

Establish serial connection between Arduino and HBS86H drive

Install VSPE (Virtual Serial Ports Emulator) on a PC. Connect the Arduino to a USB port on the PC and connect the HBS86H servo driver to the PC (either directly via RS-232 serial cable or via a USB-232 converter, as described in the manual for the driver). Open VSPE, select Device > Create... > Serial Redirector > Next. Specify the two COM ports that the HBS86H servo driver and the Arduino are connected to, and specify the speed under Settings (9600 baud for the Arduino, 38400 baud for the HBS86H). All other settings can be left as they are.

Connecting the Arduino, the STM32F4Discovery and the drive.

Connect the Arduino, the STM32F4Discovery and the drive as show below.



Startup sequence:

1. Set the wheel in straight ahead position
2. Stop the serial redirect connection in VSPE (press the black square).
3. Turn off the HBS86H servo drive (disconnect power)
4. Reset the Arduino and the STM32F4Discovery. After a delay of 7 seconds the Arduino will send initialization signals to the HBS86H servo drive, so the rest of the startup sequence has to be done within 7 seconds.
5. Turn on the HBS86H servo drive
6. Start the connection in VSPE (press the green triangle).

If connection has been established, the TX and RX LED's on the Arduino should blink almost simultaneously 7 seconds after the reset.