MOTOR CODE DOCUMENTATION

**Summer 2018**

**Jessica Chapman**

|  |  |  |
| --- | --- | --- |
| **Date** | **Editor** | **Reason** |
| 2018-08-29 | Jessica Chapman | Created Document |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  | **Table of Contents** |  |
| ***How it works*** | *1.1 Motor Setup* | *1.1.1 Comm and Datagrams* |
|  |  | *1.1.2 How to Use It* |
|  |  | *1.1.3 Improvement* |
|  |  | *1.1.4 Command Guide* |
|  |  |  |
| ***Programming*** | *2.1 Motor Code* | *2.1.1 How it Works* |
|  |  | *2.1.2 Editing Code* |
|  |  |  |
| ***Circuits*** | *3.1 Motor Circuit* | *3.1.1 List of Supplies* |
|  |  | *3.1.2 Pin Connection Guide* |
|  |  |  |
| ***Appendix*** | *4.1 Additional Information* | *4.1.1 List of Links* |
|  |  |  |

# 1.1 Motor Setup

# 

# 

## 1.1.1 Communication and Datagrams

The motor setup uses a TMC5130 board with a stepper motor and Arduino Mega. The TMC communicates with the Arduino using an SPI interface using a 40-bit datagram structure. The datagram is split into two parts: the address (8 bits) and the information to send (32 bits), with the MSB transmitting first, the MSB of the address (bit 39) will define whether it is going for read (0) or write (1) access. 0x80 can be added to the address to achieve write access instead of read access.



The SPI has four main signal types going from the master (Arduino) to control the slave (TMC5130). To hook up multiple motor controllers, they can be daisy chained together and enabled for communication using the CSN Pin. Note that the clocks must be set properly so that the communication speed is within range for the master and the slaves.

|  |  |
| --- | --- |
| SCK | Bus Clock Input |
| SDI | Serial Data Input |
| SDO | Serial Data Output |
| CSN | Chip Select Input (Active Low) |

## 1.1.2 How to Use the Program to Control a Motor

The loop in the Motor Communication INO file should be as fast as possible so as not to clog up the serial monitor, which can cause NULL problems in a python file attempting to communicate to the Arduino. Currently the loop takes about 30-200 µsec to complete.

|  |  |
| --- | --- |
| **Process** | **Time [µsecond]** |
| Feed in Serial Data | 20 |
| User Control Via Joystick | 100 |
| Seeking a Limit Switch | 12 |
| Is moving to a Position | 50 |

To use it, open up the Arduino IDE and then open the serial monitor. Change the Baud Rate to 57600 and the motor initialization information should appear on the screen. You should hear a small humming sound coming from the motor. Type in the command using the ‘Input’ under ‘List of Commands’ below, you can also find them in the Enum.h file (type in the number not the name). Make sure every command ends in a “;” and there are no additional spaces. If you get an e,unknown command; it likely means you have not formatted it correctly. If the motor does not respond and you do not hear the humming, make sure you are supplying 24V to the motor.

1. Open up Arduino IDE and Serial Monitor
2. Set Baud Rate to 57600
3. Send a command in the format [number],[\*args]; with no spaces, and a comma between each argument, with a semicolon at the end
4. An S,1; response means the command was properly received and processed and S,0; response means that the command was not properly received (check the amount of arguments or if you already have a process going on such as ‘Home’)

## 1.1.3 Suggestions for Improvement

* Better input command structure. There are safety features built in to send a failure status if the command is not correctly formatted or is missing an argument.
* Possibly building a GUI with python to call the different controls and such
* Hardware switch that overrides the current commands and goes straight to the Joystick control
* Checking motor is properly connected when sending commands (ie: code pings motor to see if it’s alive and then returns some sort of status bit or something)
* Output the position of the motor when using the joystick to find how far something is (maybe building an absolute and a relative position function)
* An easy set position on the joystick, click a button or something and the program sets the exact position of the motor as the home position
* An actual PCB / Enclosure, not just a messy breadboard. A box where you can just stick in an arduino cord to load code and the motor connections would be super nice

## 1.1.4 Command Guide

The setup uses a TMC5130 Board, Arduino Mega, Stepper Motor, and a Joystick. The code is based on a stepper motor with 256 microstep resolution. The motor is run off of 24V and the arduino can be controlled from the Arduino IDE Serial Monitor or a python script.



The setup also has the possibility for two limit switches to be added just above the motor connections, the switch configurations can be set using the commands below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Default Settings of Motor Directions and Limit Switches** | | | |
| **Forward Direction** | **Backward Direction** | **Right Limit Switch** | **Left Limit Switch** |
| Clockwise | Counter-Clockwise | Forward Hit | Backward Hit |

Note that the absolute position is described as the position relative to “Home” or point 0.

|  |  |  |
| --- | --- | --- |
| **List of Commands** | | |
| **Command** | **Input** | **Output** |
| Returns the 25-bit status from multiple registers. See table below for detailed description. | | |
| onRequestMotorStatus | 10; | m,[time],[bit0],[bit1],...,[bit24]; |
| Returns the 10-bit status from the Drive Status Register. It gives a means to measure mechanical load on the motor, a value of 0 signals the highest load (stall). | | |
| onRequestStallStatus | 11; | g,[time],[bit0],[bit1],...,[bit9]; |
| Changes the position specified to the current position of the motor (changes the step reference of the motor). The motor does not move to the specified position, but instead it becomes its position. | | |
| onRequestSetPosNoMove | 12; | d,[time],[oldPos],[newPos]; |
| Gets the ADC bits from the Arduino (Mega = 10 bit ADC). | | |
| onGetADCBits | 20; | A,[ADC]; |
| Gets the reference voltage of the Arduino (Mega = 5.0V) | | |
| onGetADCRefVolt | 21; | V,[5/3.3/2.56]; |
| Gets the absolute position of the motor in microsteps. | | |
| onGetXactual | 22; | x,[time],[Xactual]; |
| Gets the current velocity of the motor. | | |
| onGetVelocity | 23; | v,[time],[Velocity]; |
| Gets the current maximum acceleration that the motor can achieve to reach maximum velocity. | | |
| onGetAcceleration) | 24; | a,[time],[Acceleration]; |
| Gets the current maximum decceleration that the motor can achieve to reach 0 velocity. | | |
| onGetDeceleration | 25; | d,[time],[Decceleration]; |
| Gets the IHOLD\_IRUN (driver current control) power going to the motor | | |
| onGetPower | 26; | P,[time],[Power]; |
| Moves the motor forward at the speed specified continuously until the motor is stopped or a limit switch is pressed. Note that forward defaults to clockwise. | | |
| onConstantForward | 30,[speed]; | S,[1/0]; |
| Moves the motor backward at the speed specified continuously until the motor is stopped or a limit switch is pressed. Note that backward defaults to counterclockwise. | | |
| onConstantBackward | 31,[speed]; | S,[1/0]; |
| Moves to the absolute position specified. | | |
| onMovePosition | 32,[absolute position]; | S,[1/0]; |
| Moves a relative number of steps forward at the speed specified. | | |
| onMoveForward | 33,[number of steps],[speed]; | S,[1/0]; |
| Moves a relative number of steps backward at the speed specified. | | |
| onMoveBackward | 34,[number of steps],[speed]; | S,[1/0]; |
| Sets the maximum velocity achievable by the motor. For reference, the standard speed (ie: the speed of the motor used for onMovePosition) of the motor has been set to 100 000 µ/sec. 200 000 µ/sec is very fast, definitely do not exceed 300 000 µ/sec. | | |
| onVelocity | 35,[velocity]; | S,[1/0]; |
| Sets the maximum acceleration achievable by the motor. The default is set to 50 000 µ/sec2. | | |
| onAcceleration | 36,[acceleration]; | S,[1/0]; |
| Sets the maximum decceleration achievable by the motor. The default is set to 50 000 µ/sec2. | | |
| onDecceleration | 37,[decceleration]; | S,[1/0]; |
| Sets the hold and run power going to the motor. | | |
| onPower | 38,[holdpower],[runpower]; | S,[1/0]; |
| Sets the clockwise direction of the motor and the forward switch. See table below for detailed description. | | |
| onDirection | 39,[mode 1/2/3/4]; | S,[1/0]; |
| Enable joystick control. A constant moving command will override the JS enable and disable the JS. | | |
| onJSEnable | 40; | S,[1/0]; |
| Disable joystick control. | | |
| onJSDisable | 41; | S,[1/0]; |
| Stop the motor. Can be used while any other command is currently going. | | |
| onMotorStop | 42; | S,[1/0]; |
| Home the motor. The motor will run in the forward direction until it hits the forward limit switch and that will become the 0-point. | | |
| onMotorHome | 43; | S,[1/0]; |
| Seek a limit switch in the direction specified. | | |
| onSeek | 44,[direction to go]; | S,[1/0]; |
| Change the resolution of the motor, the possible settings are: 256, 128, 64, 32, 16, 8, 4, 2, 1. The motor defaults to 256. | | |
| onResolution | 45,[resolution]; | S,[1/0]; |
| Set whether the switches are active high or active low. 0 = active high, 1 = active low. | | |
| onActiveSettings | 46, [fwActive],[bwActive]; | S,[1/0]; |

|  |  |
| --- | --- |
| **Request Motor Status** | |
| **Bit Number** | **Status Bit** |
| 0 | status\_sg2 |
| 1 | status\_sg2\_event |
| 2 | status\_standstill |
| 3 | status\_velocity\_reached |
| 4 | status\_position\_reached |
| 5 | status\_position\_reached\_event |
| 6 | status\_stop\_l |
| 7 | status\_stop\_r |
| 8 | status\_stop\_l\_event |
| 9 | status\_stop\_r\_event |
| 10 | status\_latch\_l |
| 11 | status\_latch\_r |
| 12 | status\_openLoad\_A |
| 13 | status\_openLoad\_B |
| 14 | status\_shortToGround\_A |
| 15 | status\_shortToGround\_B |
| 16 | status\_overtemperatureWarning |
| 17 | status\_overtemperatureShutdown |
| 18 | status\_isReverse |
| 19 | status\_resetDetected |
| 20 | status\_driverError |
| 21 | status\_underVoltage |
| 22 | getIsForward |
| 23 | getIsPositionMode |
| 24 | getIsHomed |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Setting Motor and Switch Directions** | | | | |
| **Direction Mode** | **Forward Switch** | **Backward Switch** | **Forward Direction** | **Backward Direction** |
| 1 | Right | Left | CW | CCW |
| 2 | Right | Left | CCW | CW |
| 3 | Left | Right | CW | CCW |
| 4 | Left | Right | CCW | CW |

# 

# 2.1 Motor Code

# 

# 

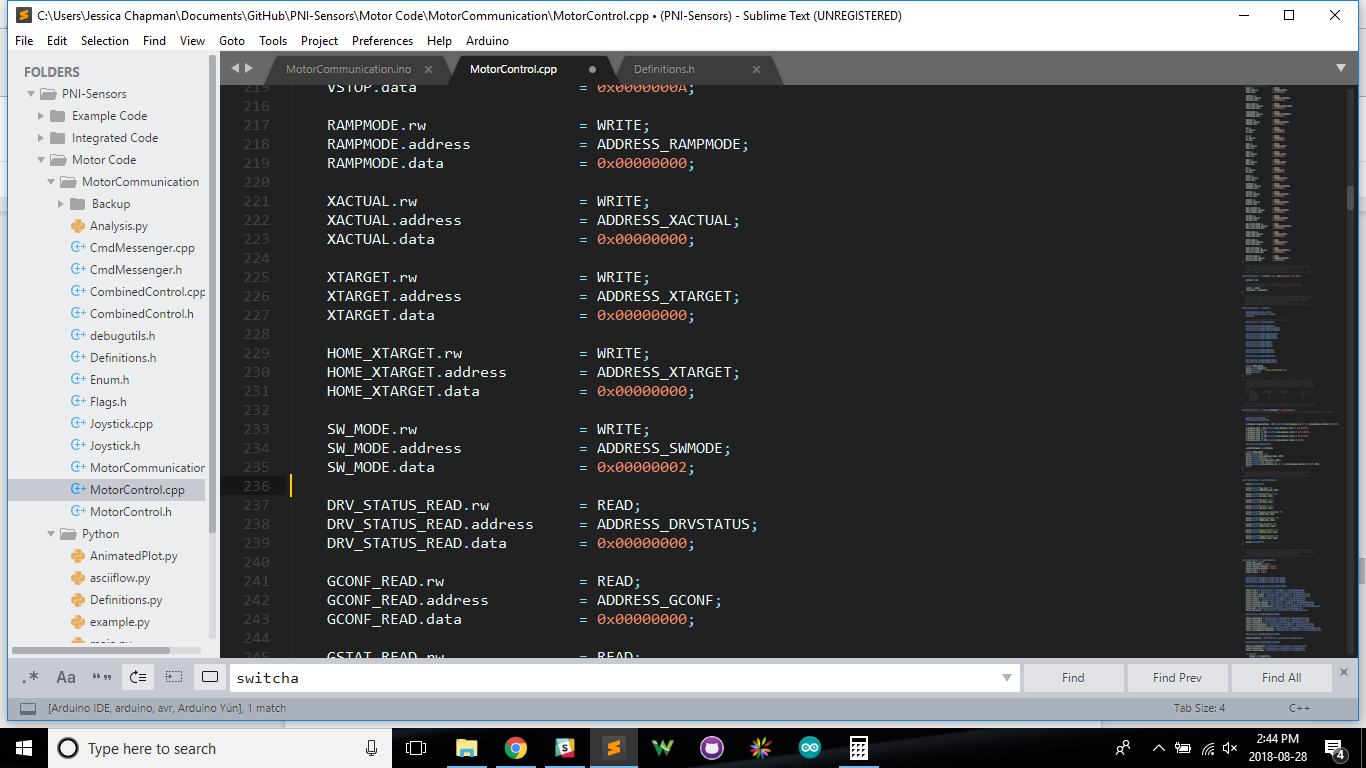
## 2.1.1 How It Works

The code has a 3-level hierarchy, starting with the INO file at the top and hardware control files at the bottom. The Combined Control file controls the interactions between the joystick and the motor. Additional hardware classes should be incorporated into the Combined Control file. The code is currently only designed to control one motor, but motors could be daisy-chained together with some small edits to the code to control the slave enable-disable for multiple motors.

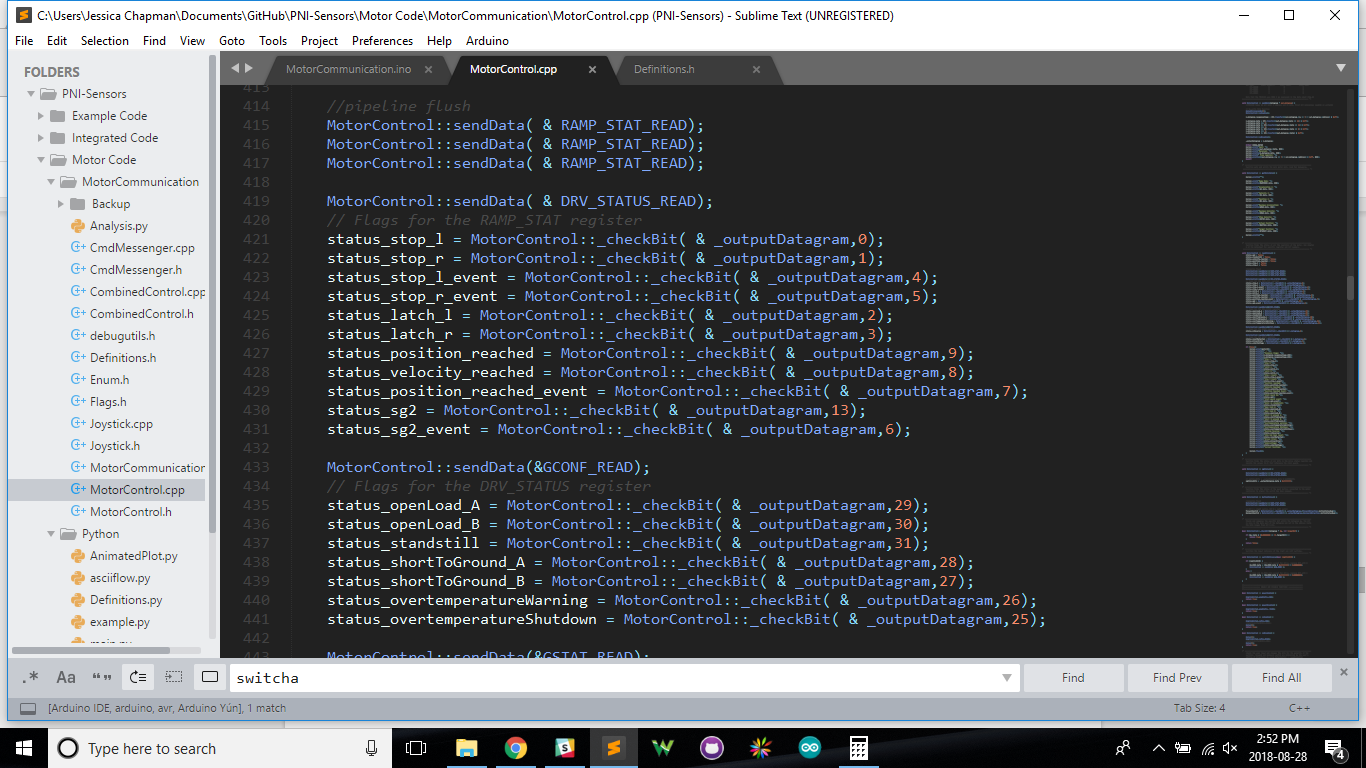


### Motor Control File

At the lowest level, the motor is controlled by registers which you send values to through an SPI interface between the motor controller and the Arduino. The program uses datagram structures which are split into three parts: the read or write access (rw) which is bit 39, the address which is bits 38-32, and the data which is bits 31-0 (the datagrams are MSB first).



When reading from the motor controller, when you send a read command, the command after will be the one holding the information from the query, this can be seen in the MotorControl :: readStatus() function.



Note that the CS Pin for the motor controller that the command is meant for should only be enabled while sending the command and should be disabled after.

In addition, anything that requires movement, direction, or the limit switches should be done using the directionControl Struct. The forward/backward directions and limit switches are software controlled in Motor Control (instead of changing the registers, which causes many problems when ccw -/-> left switch and cw -/-> right switch).

### Joystick Control File

The joystick can be used to control many different aspects of the motor, such as speed, power, acceleration, direction. Currently it is set to control the speed and direction of motors. The Joystick control does not follow the settings for the forward and backward switches and motor directions.

|  |  |
| --- | --- |
| **Current Joystick Control Settings** | |
| **Y-Direction** | **X-Direction** |
| Speed of Motor | Direction of Motor |

To save serial monitor time (sending bits to the register is a slow process), the Combined Control limits the values that will be updated such that the speed will only be updated if it is outside the range of +/-10% of the previous set reading.

## 2.1.2 Editing Code

To edit the code for more low level commands you need to use the registers provided in the definitions file and ensure the datagram has been set up (as that is how commands are sent to the motor). See [chapter 6](https://www.trinamic.com/fileadmin/assets/Products/ICs_Documents/TMC5130_datasheet_Rev1.15.pdf) for how to change the registers for what you need.

When you need to send information to the motor you should follow the format:

XTARGET.data = xtarget;

MotorControl :: sendData(&XTARGET);

To add new Serial Control Commands:

1. In the Enum.h file, add a definition ( \_\_DEFINITION\_\_ ) to the appropriate spot and assign a number
2. In MotorCommunication.ino under void attachCommandCallbacks() add your command using the format:

cmdMessenger.attach( \_\_DEFINITION\_\_, \_\_FUNCTION\_\_ );

1. Still in MotorCommunication.ino, add your function ( \_\_FUNCTION\_\_ )

To control the movement of the motor, I suggest calling one of the four already made functions (constForward, forward, constReverse, reverse) and using them in CombinedControl instead of making your own function in MotorControl; this is to keep from developing bugs in the software direction control. If you choose to make your own in MotorControl function, keep in mind the different ramp modes to use:

|  |  |  |
| --- | --- | --- |
| **Ramp Mode** | **Position Mode** | **Direction** |
| RAMP\_MODE0 | True | -- |
| RAMP\_MODE1 | False | Forward |
| RAMP\_MODE2 | False | Backward |
| RAMP\_MODE3 | -- | -- |

# 3.1 Motor Circuit

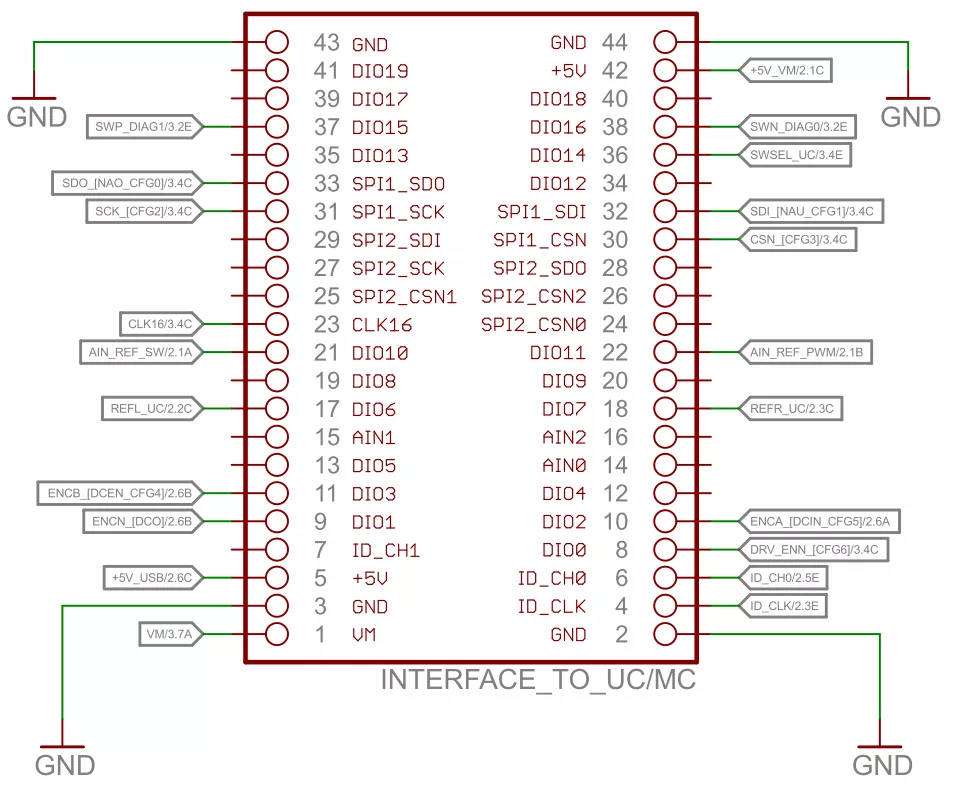
## 3.1.1 List of Supplies

|  |  |  |
| --- | --- | --- |
| **Supplies** | **Documentation** | **Additional Resources** |
| [TMC5130 Eval](https://www.trinamic.com/support/eval-kits/details/tmc5130-ev/) | [User Manual and Cmd Guide](https://www.trinamic.com/fileadmin/assets/Products/Eval_Documents/TMC5130_Eval_manual.pdf) | [Example Setup](http://blog.trinamic.com/2017/04/05/how-to-use-tmc5130-eval-with-your-arduino-mega/) |
| [Arduino Mega 2560](https://store.arduino.cc/usa/arduino-mega-2560-rev3) | [Schematic](https://www.arduino.cc/en/uploads/Main/arduino-mega2560_R3-sch.pdf) | [Pin Mapping](https://www.arduino.cc/en/Hacking/PinMapping2560) |
| [2-Axis Joystick](https://www.parallax.com/product/27800) | [Schematic](https://www.parallax.com/sites/default/files/downloads/27800-2-Axis-Joystick-Schematic-RevB.pdf) | [Arduino Code](https://www.arduino.cc/en/Tutorial/JoyStick) |

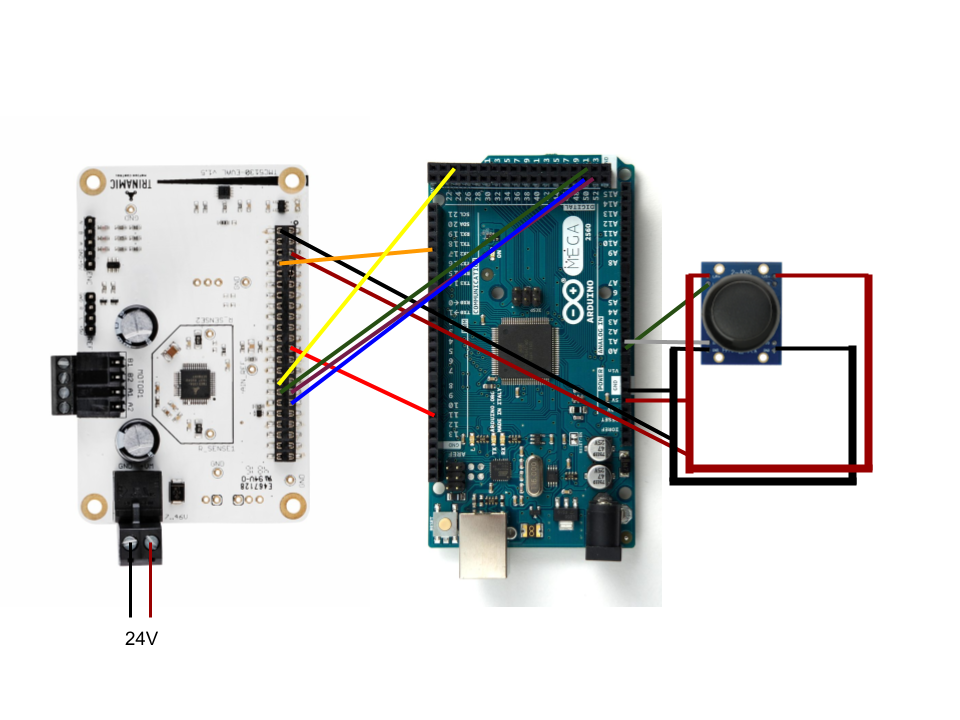
Additionally, the setup requires a [stepper motor](https://catalog.orientalmotor.com/item/cvd-2-phase-bipolar-stepper-motor-drivers/legacy-pkp-series-2-phase-bipolar-stepper-motors/pkp244d23a), a 24V Power Cord + jack, 2 limit switches, a breadboard or PCB, and some jumper cables. Toggle switches are also useful for enabling/disabling the joystick and for turning on/off the power to the motor.

## 3.1.2 Pin Connection Guide

The pins going from the Arduino to the TMC5130 can be connected with or without the pin connector piece and Evaluation Platform ([see page 5](https://www.trinamic.com/fileadmin/assets/Products/Eval_Documents/TMC5130_Eval_manual.pdf)), the diagrams below do not use the pin connector.

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Arduino Pin** | **Type** | **TMC5130 Pin** | **Type** | **Colour** |
| --- | GND | 2 | GND | Black |
| --- | 5V | 5 | +5V | Red |
| 11 | DIO | 23 | CLK6 | Red |
| 17 | DIO | 8 | DIO0 | Orange |
| 25 | CS | 30 | SPI1\_CSN | Yellow |
| 50 | MISO | 33 | SPI1\_SDO | Blue |
| 51 | MOSI | 32 | SPI1\_SDI | Purple |
| 52 | SCK | 31 | SPI1\_SCK | Green |

****

|  |  |  |
| --- | --- | --- |
| **Arduino Pin** | **Joystick** | **Colour** |
| GND | Both GND Pins | Black |
| 5V | L/R+ and U/D+ | Red |
| A0 | L/R | Green |
| A1 | U/D | Grey |

# 

# 4.1 Appendix

## 4.1.1 Relevant Links

|  |  |
| --- | --- |
| **GitHub** | https://github.com/ProjektEskie/PNI-Sensors/tree/master/Motor%20Code |
| **TMC5130 DataSheet** | https://www.trinamic.com/fileadmin/assets/Products/Eval\_Documents/TMC5130\_Eval\_manual.pdf |
| **TMC5130 Cmd Guide** | https://www.trinamic.com/fileadmin/assets/Products/ICs\_Documents/TMC5130\_datasheet\_Rev1.15.pdf |