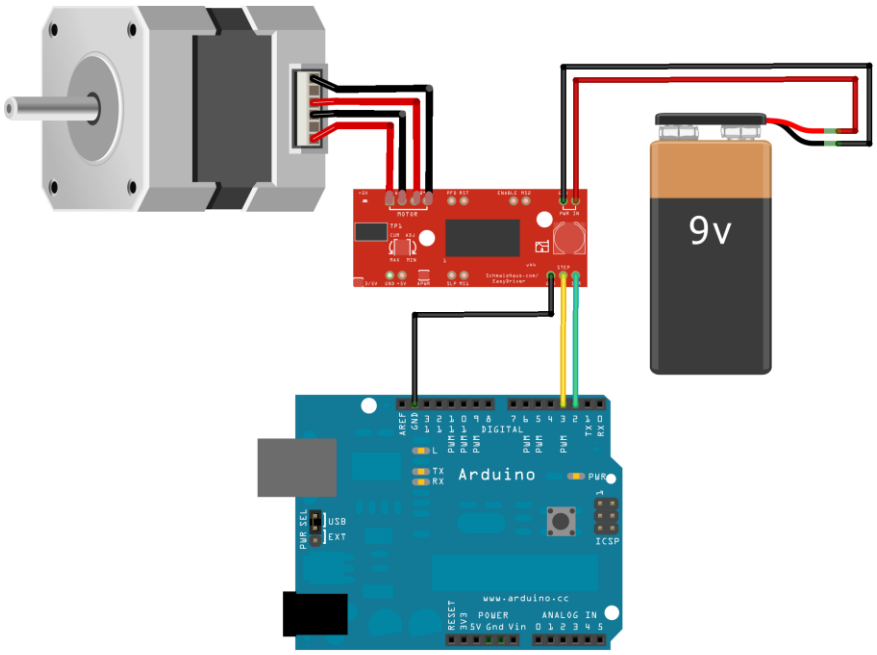
Arduino, stepper motors and Easydriver

Stepper motors or stepper motors are direct current motors, without brushes and are ideal if you want to integrate them in a certain application, which requires a certain speed of rotation or if you want the motor to rotate to a certain point and then keep its position. A direct current motor can be controlled to move in a certain direction with a given speed, which you do by applying a certain voltage to its terminals. The engine rotates how long there is applied voltage. However, you won't be able to control its exact rotation (for example, you don't have how to rotate it by a fixed 45 degrees, and then stop it).

The operation mode of stepper motors is different. One complete rotation of an engine stepper is made up of several steps, each step representing only a fraction of a complete rotation the engine. This is due to the internal construction, the rotor being composed of permanent magnets, and the stator from the windings. For this reason, a stepper motor can be controlled extremely precisely. It you can rotate, for example, by 1 degree to the left (that is, the 360th part of a complete rotation of the axis). Or you can rotate it 45 degrees to the right and then you can lock it. Of course, the control is also somewhat more complicated than in the case of a direct current motor. From luckily, using Arduino and a specialized driver, things become simple. In what follows we will features the use of EasyDriver (a specialized stepper motor driver) to control a stepper motor step by step



EasyDriver allows you to control the motor in very small steps. This technique is called microstepping, basically the driver divides a step into 8 microsteps. General purpose engines achieve a complete rotation in 200 steps or the angle of one step is 1.8°. But because EasyDriver shares a step in 8 microsteps, then 1600 microsteps are required for a complete rotation of the

motor. To stay it means that the motor can rotate with high precision. At high speeds, the engine develops a force reduced. Before putting the circuit into operation, you must be careful when choosing the voltage source, for that you must respect the engine parameters: supply voltage, and consumption. In general, the engines supplies at 12 V.

The other one important parameter is the intensity of the required current (there is a direct relationship between this and the force the engine; the more current the motor requires to operate, the more you have to wait that it will have a greater force). Before moving on to the code, I want to warn you that a stepper motor will heats up during operation, and this is normal. To work, through the motor coils electric current flows all the time, even when the engine is stationary (in blocking mode). As long as you can touch the engine with your hand, it is OK (60 - 80 degrees Celsius).

If you want to keep the engine cooler, you have two solutions. The first would be to reduce its current. On the EasyDriver board, there is a small potentiometer from which you can do this. Obviously, the lower the current, the better engine power is lower. The second thing you can do is that when the engine stops, do disable in the driver. To achieve this, connect the ENABLE pin on the EasyDriver board to a Arduino digital pin. When you lower the digital pin to zero, current will no longer flow through the motor (the engine will cool down). Obviously, the engine will be free to rotate at this moment (if there are external forces in your mechanical system). On a case-by-case basis, you may or may not apply this approach. For example, for a humanoid herding robot, you can't do that, because when you disable the driver, your robot will fall to the ground. But if it is about a system that moves a curtain, then no there is no problem if, after finishing moving the curtain, I disable the driver. The drapery will it remains in the current position, and the motor will not consume current and will not heat up.

The source code

The first lines of code are highlighted by two directives:

```
#define DIR_PIN 2

#define STEP_PIN 3
```

Wherever DIR_PIN is written in the code, the program refers to pin 2 of the Arduino. Same thing it is also valid for STEP_PIN.

```
void setup() {

pinMode(DIR_PIN, OUTPUT);

pinMode(STEP_PIN, OUTPUT);

}
```

Logical levels, pulses or bits, it depends how you prefer to call them from Arduino to EasyDriver so DIR_PIN and STEP_PIN will be set as output.

```
void loop(){
//rotate a specific number of degrees
rotateDeg(360, 1);
delay(1000);
rotateDeg(-360, 0.1); //reverse
delay(1000);
//rotate a specific number of microsteps (8 microsteps per step)
//a 200 step stepper would take 1600 micro steps for one full
//revolution
rotate(1600, 0.5);
delay(1000);
rotate(-1600, 0.25); //reverse
delay(1000);
}
```

Loop is a repetitive loop, it executes consecutively the lines of code inside it. In the loop you will notice that the rotatedDeg, rotated and delay functions are called. The delay function accepts a number as a parameter of milliseconds so delay(1000) means that the program will stay in place for one second.

```
void rotate(int steps, float speed){
//rotate a specific number of microsteps (8 microsteps per
//step) - (negative for reverse movement)
//speed is any number from .01 -> 1 with 1 being fastest - Slower
//is stronger
int dir = (steps > 0)? HIGH:LOW;
steps = abs(steps);
digitalWrite(DIR_PIN,dir);
float usDelay = (1/speed) * 70;
for(int i=0; i < steps; i++){
digitalWrite(STEP_PIN, HIGH);
delayMicroseconds(usDelay);
digitalWrite(STEP_PIN, LOW);
delayMicroseconds(usDelay);
}
}
```

The rotate function accepts int type steps and float type speed parameters. The derotation direction is set, if steps has a value greater than zero then dir will go to logical "1", conversely if steps is negative, dir will go to logical "0". The direction is sent to EasyDriver by:
digitalWrite(DIR_PIN,dir);
If dir is 1 then the motor will rotate in one direction and if dir goes to 0 the motor will change the direction of rotation.

The for loop transmits the number of steps. We observe a usDelay variable. Let's assume that speed = 1 (very high speed). Then usDelay would be equal to 70. But if speed = 0.5 then usDelay = 140. I have pointed this out because a longer delay is equivalent to a lower speed.

```
void rotateDeg(float deg, float speed){
//rotate a specific number of degrees (negitive for reverse
//movement)
//speed is any number from .01 -> 1 with 1 being fastest - Slower
//is stronger
int dir = (deg > 0)? HIGH:LOW;
digitalWrite(DIR_PIN,dir);
int steps = abs(deg)*(1/0.225);
float usDelay = (1/speed) * 70;
for(int i=0; i < steps; i++){
digitalWrite(STEP_PIN, HIGH);
delayMicroseconds(usDelay);
digitalWrite(STEP_PIN, LOW);
delayMicroseconds(usDelay);
}
}
```

The rotateDeg function is similar. The difference lies in the input parameters and in the line of code
convert the number of degrees into the number of steps:

```
int steps = abs(deg)*(1/0.225);
```

The rest of the lines work on the same principle.

```
#define DIR_PIN 2
#define STEP_PIN 3
void setup() {
pinMode(DIR_PIN, OUTPUT);
pinMode(STEP_PIN, OUTPUT);
}
void loop(){
//rotate a specific number of degrees
rotateDeg(360, 1);
delay(1000);
rotateDeg(-360, 0.1); //reverse
delay(1000);
//rotate a specific number of microsteps (8 microsteps per step)
//a 200 step stepper would take 1600 micro steps for one full
//revolution
rotate(1600, 0.5);
delay(1000);
rotate(-1600, 0.25); //reverse
```

```
delay(1000);
}
void rotate(int steps, float speed){
//rotate a specific number of microsteps (8 microsteps per
//step) - (negitive for reverse movement)
//speed is any number from .01 -> 1 with 1 being fastest - Slower
//is stronger
int dir = (steps > 0)? HIGH:LOW;
steps = abs(steps);
digitalWrite(DIR_PIN,dir);
float usDelay = (1/speed) * 70;
for(int i=0; i < steps; i++){
digitalWrite(STEP_PIN, HIGH);
delayMicroseconds(usDelay);
digitalWrite(STEP_PIN, LOW);
delayMicroseconds(usDelay);
}
}
void rotateDeg(float deg, float speed){
//rotate a specific number of degrees (negitive for reverse
//movement)
//speed is any number from .01 -> 1 with 1 being fastest - Slower
//is stronger
int dir = (deg > 0)? HIGH:LOW;
digitalWrite(DIR_PIN,dir);
int steps = abs(deg)*(1/0.225);
float usDelay = (1/speed) * 70;
for(int i=0; i < steps; i++){
digitalWrite(STEP_PIN, HIGH);
delayMicroseconds(usDelay);
digitalWrite(STEP_PIN, LOW);
delayMicroseconds(usDelay);
}
}
```