Arduino Montierungssteuerung

Was tut man, wenn man eine ältere hochwertige Montierung oder gar eine selbstgebaute besitzt, die man mit modernen Schrittmotoren ausstatten möchte, aber keine passende Steuerung hat?

Eine neue Steuerung für über 800€ kaufen? Auf dem Gebrauchtmarkt mehrere hundert Euro ausgeben? GoTo Funktionalität bezahlen, obwohl man selbst nur einen Tangentialarm in Deklination hat?

Lange Zeit musste ich daher mit einer halbfunktionierenden, alten Steuerung auskommen und beim Wechsel zwischen meiner 30 Jahre alten RUPP Montierung und der Selbstbaumontierung von Rolf Eckert mühsam die Steuerung umkonfigurieren.

Bis ich mich vor einiger Zeit getraut habe, eine Steuerung basierend auf dem Arduino Mikroprozessor zu basteln. Eine Steuerung selber basteln? Au weia, Achim und Elektronik! Ich wusste ja, wie das normalerweise bei mir endet. O.K., kurz im Internet informiert und ein Herz gefasst.

Grundkenntnisse waren vorhanden, Ohmsches Gesetz, Widerstände, Kondensatoren und auch ein Lötkolben waren mir nicht fremd. Und wenn es nicht so einfach wäre, würde ich nun auch nicht hier sitzen und einen Artikel schreiben...

Ein Arduino ist ein Mikroprozessor, den man einfach ausgedrückt benutzen kann, um verschiedenste Sachen zu steuern. Entwickelt wurde er Ende 2005 von Prof. Massimo Banzi und David Cuartielles am Interaction Design Institute Ivrea (IDII) in Italien. Namensgebend war ein Studentenlokal nahe des IDII, welche nach dem italienischen König Arduino (um 1000 n. Ch.) benannt wurde. Zur eigentlichen Hardware gehört auch Entwicklungssoftware mit einer grossen Bibliothek und eine große, aktive Community.

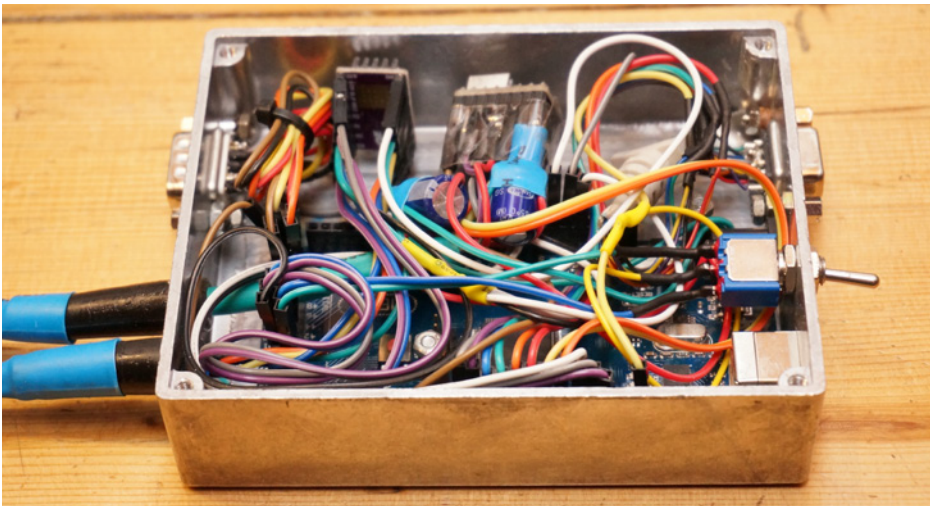
Der von mir verwendete Arduino Mega kostet als Nachbau ca. 15€ und hat einen 8bit Prozessor mit 16 MHz und 54 digitalen Ein- und Ausgängen. Ein Arduino Uno mit weniger Ein- und Ausgängen würde es für eine einfache Steuerung auch tun. Wo wir schon bei der Hardware sind, was braucht man noch? Eine Aufsteckplatine („Shield“) mit den Schrittmotortreibern (10€ bis 30€), Kabel mit Steckanschlüssen (ca. 5€) und ggf. Taster, Stecker, Gehäuse für die Steuerung und die Handbox.

Bleiben wir nun erst einmal bei der Hardware und schauen, was da zu tun ist. Den Aufbau kann man sehr gut in einzelne Schritte aufteilen, diese dann unabhängig voneinander testen. So habe ich zuerst die Handsteuerbox mit den 4 Richtungen angeschlossen. Beispiele für den Anschluss der Taster und die Programmierung (dazu später mehr) gibt es zuhauf im Internet. Zum Testen der Taster hab ich keinen Motor samt Treiberplatine benutzt, sondern einfach die interne LED auf dem Arduino bei Tastendruck aufleuchten lassen. So sammelt man erste Erfahrungen und die weiteren Arbeitsschritte bauen darauf auf. An welchen Ein/Ausgangspins man alles anschließt ist egal, die Pins werden später per Softwarebefehle angesprochen.

Der nächste Schritt ist der Anschluss der Motortreiber. Diese wandeln die Step-Befehle von der Steuerung in Impulse für den Schrittmotor um. Außerdem kann man dort auch noch die Feinheit der Schritte einstellen, von Vollschritten je nach Treiber bis zu 1/32 Schritt per Schrittbefehl. An diese Motortreiber werden die Schrittmotoren angeschlossen.

Im Prinzip ist man danach schon fertig, kann aber auch noch einen Autoguidereingang einbauen (analog zum Handtaster), eine Status-LED und weitere Schalter z.B. für verschiedene Nachführ- oder Aufsuchgeschwindigkeiten.

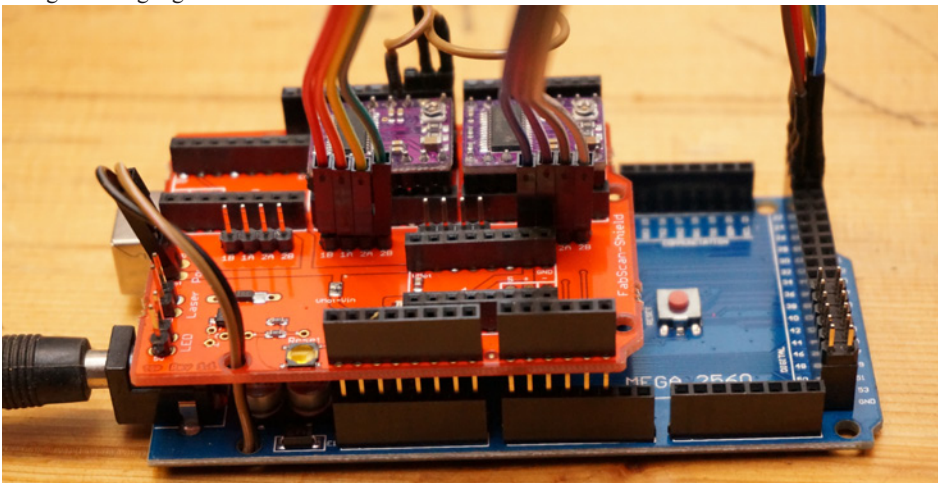
Hat man alles (frei) verdrahtet sieht es so aus:



Komplette Steuerung mit 15 poligen Steckanschluss für den Handtaster (rechts) und entsprechendem Autoguidereingang links.

Au weia, das soll jemand nachbauen? Nein, es geht nämlich viel einfacher! Für die Arduinos gibt es nämlich Aufsteckplatinen als fertige Bauteile, sogenannte Shields. Für uns interessant ist das sog. FabScan-Shield mit gleich 4 Motortreibern. Eigentlich ist es für einen 3D Scanner entwickelt worden, leistet aber auch für unsere Zwecke gute Dienste. Entweder kauft man das Shield unbestückt für 10€ zum selber löten, oder für ca. 30 € komplett bestückt. Leider wusste ich bei meiner ersten Steuerung noch nichts davon...

Mit dem FabScan-Shield sieht die Steuerung schon wesentlich ordentlicher aus. Oben jeweils die Ausgänge für die Motoren und rechts die Kabel für die Handbox. Und da wir einen Arduino Mega benutzt haben bleiben noch jede Menge Ein-/Ausgabepins unbelegt für Erweiterungen wie den Autoguidereingang.



Einfache Steuerung mit Anschluss für den Handtaster (rechts) und den 2 Anschlüssen für die Motoren. Zwei Steckplätze für Motortreiber sind auf dem Shield unbestückt. Links die Stromversorgung (12V) und dahinter der USB-Anschluss zum laden der Software

Nun fragt man sich, alles zusammengesteckt und gelötet, wie läuft das Teil denn nun eigentlich?

Klar, die Software fehlt noch!

Geschrieben wird die Software in der Programmiersprache „C“. Aber keine Sorge, das ist recht einfach, man braucht nur wenige Befehle. Und im Internet gibt es viele sehr gute Tutorials.

Die Software wird in einem einfachen Editor geschrieben, der bei der Arduino Umgebung neben vielen Beispielprogrammen dabei ist. Man schreibt das Programm, Sketch genannt, auf seinem Rechner, kompiliert es und lädt es per USB-Kabel auf den Arduino. Dort bleibt es gespeichert und läuft von nun an auch ohne PC sobald der Arduino mit Strom versorgt wird.

Der Sketch ist in zwei Funktionen gegliedert. Die Funktion „setup“ wird beim Starten aufgerufen. Dort wird z. B. angegeben, welche Pins als Ein- oder Ausgang benutzt werden. Die Handsteuerbox belegt Eingangspins, die Motortreiber Ausgangspins. Der Befehl hierzu lautet pinMode(11, OUTPUT); bzw. pinMode(11, INPUT_PULLUP); , hier beispielsweise der 11. Pin. Bei INPUT_PULLUP wird zusätzlich ein interner Widerstand benutzt, den man ansonsten zusätzlich dran löten müsste, um definierte Zustände (LOW/HIGH entspr. AUS/EIN) zu erhalten.

Die Hauptfunktion ist die „loop“, eine Schleife die ständig wiederholt durchlaufen wird. Dort frägt man z. B. einen Pin der Handsteuerung ab (digitalRead(pin);) und schickt ggf. bei gedrücktem Taster einen entsprechenden Befehl an den Motortreiber, um einen Schritt zu machen. Der Befehl hierzu lautet digitalWrite(pin, LOW); gefolgt von digitalWrite(pin, HIGH);

Und wann bzw. wie oft macht man Schritte? Das kommt auf das Getriebe an: man kann sich das Intervall einfach über die Getriebeübersetzung und die Dauer eines Tagen ausrechnen (in Mikrosekunden). In den einfachen Beispielen für Schrittmotorsteuerungen wird hierzu zwischen den Schritten einfach eine entsprechend lange Pause eingelegt.

Nun ja, das ist weder effektiv noch funktioniert es bei Zweiachsensteuerungen. Stellen wir uns die „loop“ als Fließband vor, an dem 2 Arbeiter sitzen. Diese bewegen die Motoren jeweils um einen Schritt weiter, wenn der entsprechende Befehl auf dem Fließband liegt und an ihnen vorbei kommt. Mit der Pausenfunktion aus den Beispielprogrammen wird dann aber das gesamte Fließband angehalten, der zweite Arbeiter muss dann ebenfalls warten, obwohl vielleicht ein Befehl für ihn auf dem Fließband liegt. So stören sich Rektaszensions- und Deklinationsbefehle, die Nachführung wird nicht funktionieren. Von weiteren Befehlen wie Autoguider und LED-Statusanzeigen usw. mal ganz abgesehen...

Und wie macht man das nun? Ganz einfach, wenn man in bestimmten Intervallen einen Schritt machen möchte, misst man die Zeit, die seit dem letzten Schritt vergangen ist, bis die Intervalldauer erreicht ist. Die beiden Arbeiter arbeiten ja immer die selben Befehle ab, nämlich einen Motorschritt machen. Also schauen sie jedes mal, wenn das Fließband an ihnen vorbeikommt, wieviel Zeit vergangen ist und machen dann einen Schritt oder halt nicht. So stören sich die einzelnen Steuerungsbefehle z. B. Rektaszension oder Deklination von dem Handtaster nicht gegenseitig.

Man kann die seit dem Start des Arduinos vergangene Zeit über die Befehle millis(); und micros(); abfragen und somit entsprechend die Motoren ansprechen, wenn ihr jeweiliges Intervall abgelaufen ist.

Ich möchte hier keine komplette Bauanleitung anbieten sondern nur anregen, es selber mal zu versuchen. Daher wird auch nicht auf noch fehlende Details wie die Einstellung des Motorstoms (steht in den Datenblättern zu den Motortreibern) oder der Konfiguration der Schrittweite (Mikroschritte) eingegangen. Auf Anfrage versende ich gerne den Sketch per Mail und gebe Hilfestellung.

Und wie geht es weiter? Einen der freien Motortreiber werde ich zukünftig benutzen, um einen Fokussiermotor anzusteuern, am liebsten auch noch mit automatischer temperaturabhängiger Korrektur. Einen Temperatursensor an einen analogen Eingang anzuschließen ist ganz einfach. Der Fantasie, wie man die Steuerung mit weiteren Modulen ausstattet, sind keine Grenzen gesetzt.

Auf die Spitze getrieben hat es sicher Howard Dutton mit seiner OnStep Steuerung (Link unten). Diese GoTo Steuerung kommuniziert per Bluetooth-Modul mit einem Android-Smartphone, von dem aus man die Montierung per Planetariumsprogramm steuern kann. Die Hardware ist einfach aufgebaut und kostengünstig, den Sketch kann man sich kostenlos herunterladen.

Achim Schaller

Arduino:	http://www.arduino.org/
Fabscan shield:	http://hci.rwth-aachen.de/fabscan_shield
Bausatz:	http://www.watterott.com/de/Arduino-FabScan-Shield
OnStep:	http://www.stellarjourney.com/index.php?r=site/equipment_onstep