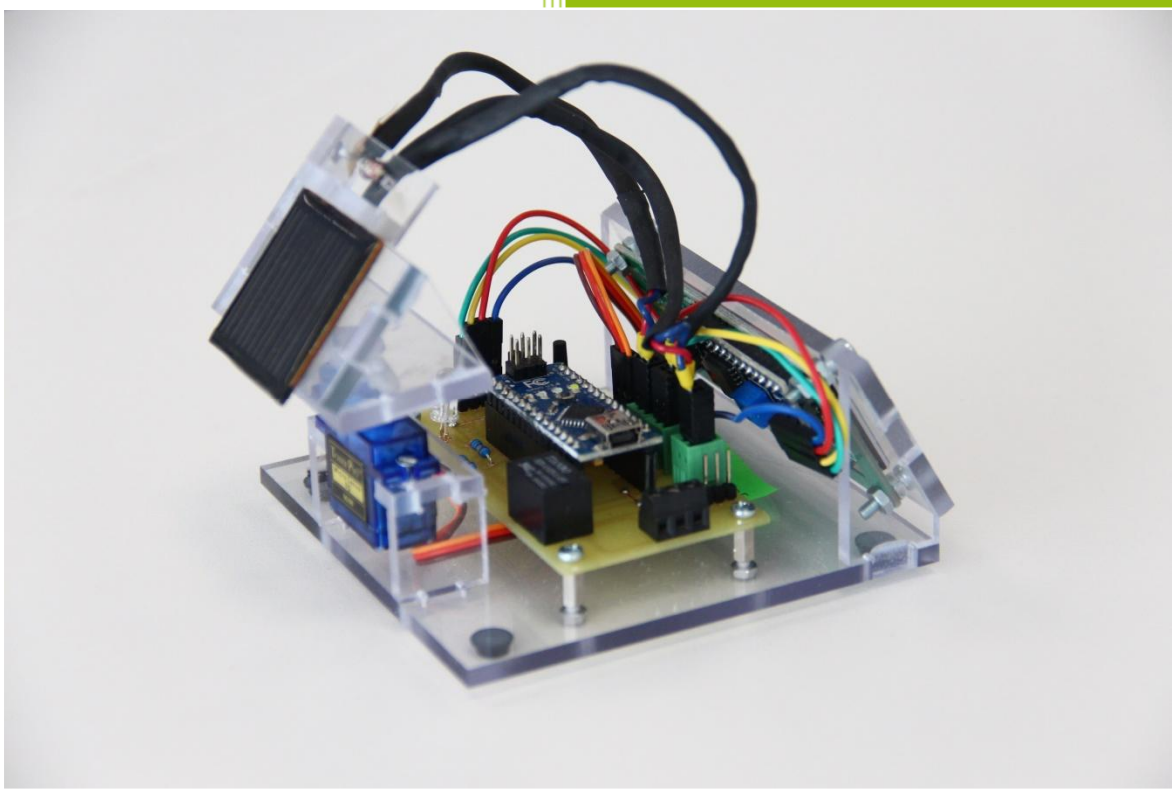
Projekt

Leitfaden

Solarpanel-Nachführung



Phoenix Contact GmbH & Co. KG
Professional Education (PED)
Lars Christian Gleichmann
30.05.2018

Inhaltsverzeichnis

1. Projektvorstellung 1
1.1 Projektziele 1
1.2 Material 1
2. Praktische Umsetzung des Projekts 2
2.1 Montage des Gehäuses 2
2.1.1 Gehäuse zusammenstecken 2
2.1.2 Leiterplattenhalterung 4
2.1.3 Kleben 5
2.1.4 Bildschirmmontage 6
2.1.5 Motormontage 7
2.1.6 Solarpanelhalterung 8
2.2 Die Platine 9
2.2.1 Der Schaltplan 9
2.2.2 Das Platinenlayout 9
2.2.3 Auflöten der Bauteile 11
3. Die Schaltung und ihre Bestandteile 12
3.1 Der Mikrocontroller 12
3.1.1 Was ist ein Mikrocontroller? 12
3.1.2 Der ATmega328 12
3.2 Die Lichtsensorik 13
3.3 Der Stellmotor 14
3.4 Das Relais 15
3.4.1 Taster 16
3.4.2 Die RGB-LED 17
4. Programmierung 18
4.1 Entwicklungsumgebung und Programmiersprache 18
4.2 Klassen 18
4.3 Variablen 19
4.4 Funktionen 20
4.5 Neues Projekt 20
4.6 Bibliotheken 21
4.7 Befehle 22
4.8 Das Programm schreiben 22
4.9 Programm brennen 22
4.10 Hintergrundwissen: Was passiert mit meinem Programm? 23

5. Experimente 24
I. Befehlsverzeichnis für Solarpanelnachführung 1
II. Beispiel für Quelltext des fertigen Programms 3

1. Projektvorstellung

1.1 Projektziele

Im Folgenden soll eine Solarpanelnachführung gebaut und programmiert werden. Ziel dieser Baugruppe ist, eine Photovoltaikzelle mithilfe eines Motors optimal zur Sonne auszurichten. Außerdem soll die erzeugte Spannung gemessen werden, was Rückschlüsse auf die Intensität der jeweiligen Sonneneinstrahlung zulässt.
Das gesamte System soll hierbei autark arbeiten und alle relevanten Werte sollen auf einem LC-Display angezeigt werden.

Dem Projektteilnehmer soll grundlegendes Wissen zum Thema Elektrotechnik und Informatik anhand dieses Projektes praktisch vermittelt werden, sodass die Funktionsweise aller Bauteile bekannt ist und diese auch über das Projekt hinweg verwendet werden können. Des Weiteren soll sich der Teilnehmer nach dem Projekt in die Lage versetzt fühlen, Platinen zu löten und einen Mikrocontroller im prozeduralen Stil selbstständig zu programmieren.

1.2 Material

Für die praktische Umsetzung benötigen wir folgende Bauteile **(bitte auf Vollständigkeit prüfen)**:

Anzahl	Artikel	Verwendung
1	Leiterplatte	Leiterplatte
4	Schrauben M2,5x4mm	Leiterplatte
4	Abstandsbolzen M2,5x5mm	Leiterplatte
4	Mutter M2,5	Leiterplatte
1	Display I2C 1602 Serial Green LCD Display Module	Display
8	Mutter M3	Display
4	Fächerscheibe M3	Display
4	Schraube M3x12mm	Display
1	Gehäusebausatz	Gehäuse
4	Gehäusefuß 120-6041	Gehäuse
1	Servo TGY-R5180MG	Servomotor
1	Solar Cell Module YH-39X35	Solarzelle 4V/35mA
8	Durchkontaktierung Easy Contact 0,6mm	Leiterplatte
1	Widerstand 100 Ω	Leiterplatte
2	Widerstand 68 Ω	Leiterplatte
2	Widerstand 10 kΩ	Leiterplatte
2	Taster 700336	Leiterplatte
3	Buchse MC 0,5 - 2,5	Leiterplatte
3	Stecker FK-MC 0,5/2-ST-2,5	Leiterplatte
1	Relais Tianbo HJR-4102E-L-05V	Leiterplatte
1	LED LL 5-8000RGB	Leiterplatte
2	Conrad Buchsenleiste 15pol	Leiterplatte
2	Stiftleiste 3pol	Leiterplatte
1	Stiftleiste 4pol	Leiterplatte Display
1	Buchse PTSM 0,5/2-HV-2,5-THR R32	Leiterplatte Akku

1	Stecker PTSM 0,5/2P2,5	Leiterplatte Akku
1	Jumper 2,54 GL SW	Leiterplatte
1	Stiftleiste 4Pol	Leiterplatte Relais
2	A9060	Photowiderstand
1	LED LL 5 – 8000RGB	RGB-LED

Zur Durchführung der Arbeiten wird außerdem benötigt:

- 1x Mini-USB-Kabel
- Leitungen
- Abisolierzange
- Computer mit Arduino IDE 1.8.4 oder besser
- Lötkolben mit Lötzinn und Zubehör
- Silicon oder Komponentenkleber

Für Experiment 2 wird zusätzlich benötigt:

- 1x Widerstand 50 Ohm
- 1x Widerstand 100 Ohm
- 1x LED rot
- 1x LED grün
- Steckbrett und Leitungen

2. Praktische Umsetzung des Projekts

2.1 Montage des Gehäuses

Das Gehäuse unserer Solarpanelnachführung wurde bereits durch einen Kunststofffräser gefertigt und muss nur noch montiert werden. Dafür sind folgende Schritte erforderlich:

2.1.1 Gehäuse zusammenstecken

Die von der Fräse gefertigten Bauteile (siehe Abbildung 1) müssen zunächst richtig zusammengesteckt werden. Dies sollte ohne Kraftaufwand möglich sein. Das so entstandene Gehäuse sollte dann so aussehen, wie es Abbildung 2 zeigt.

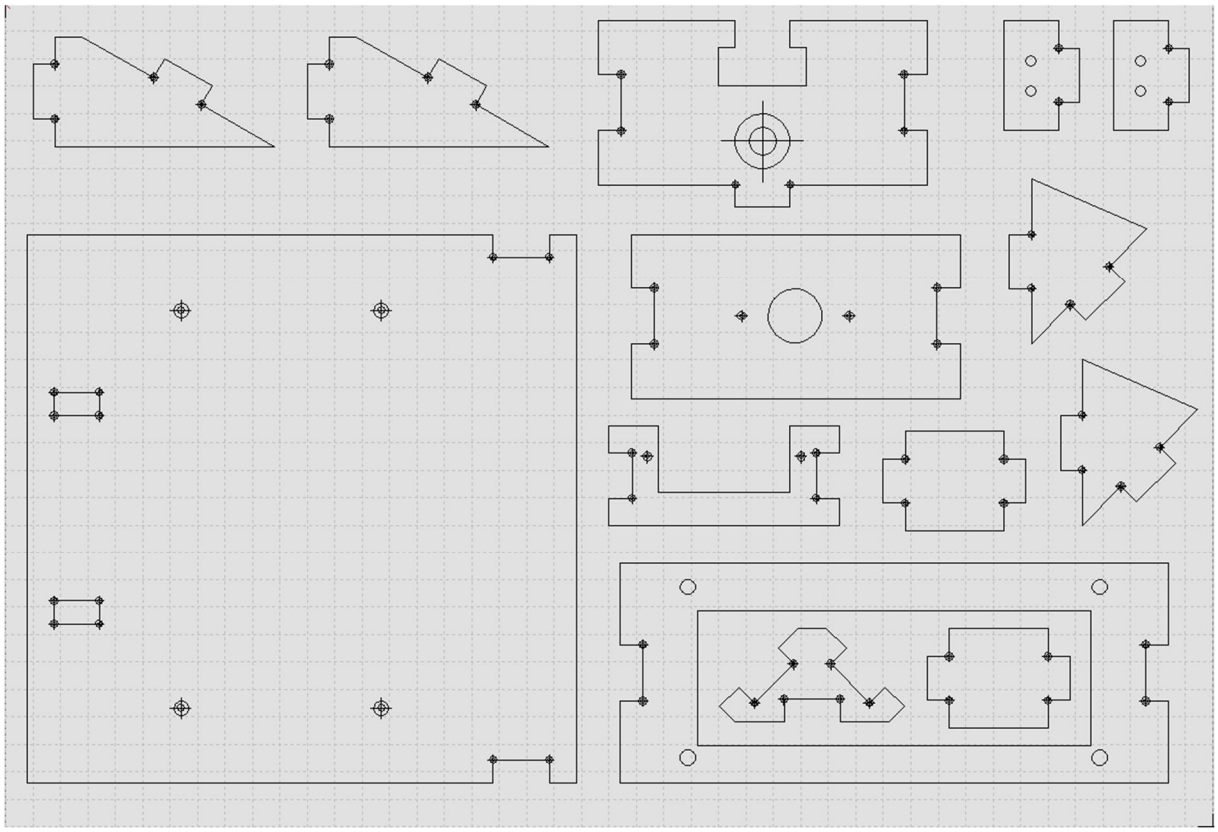


Abbildung 1 - Fräsvorlage

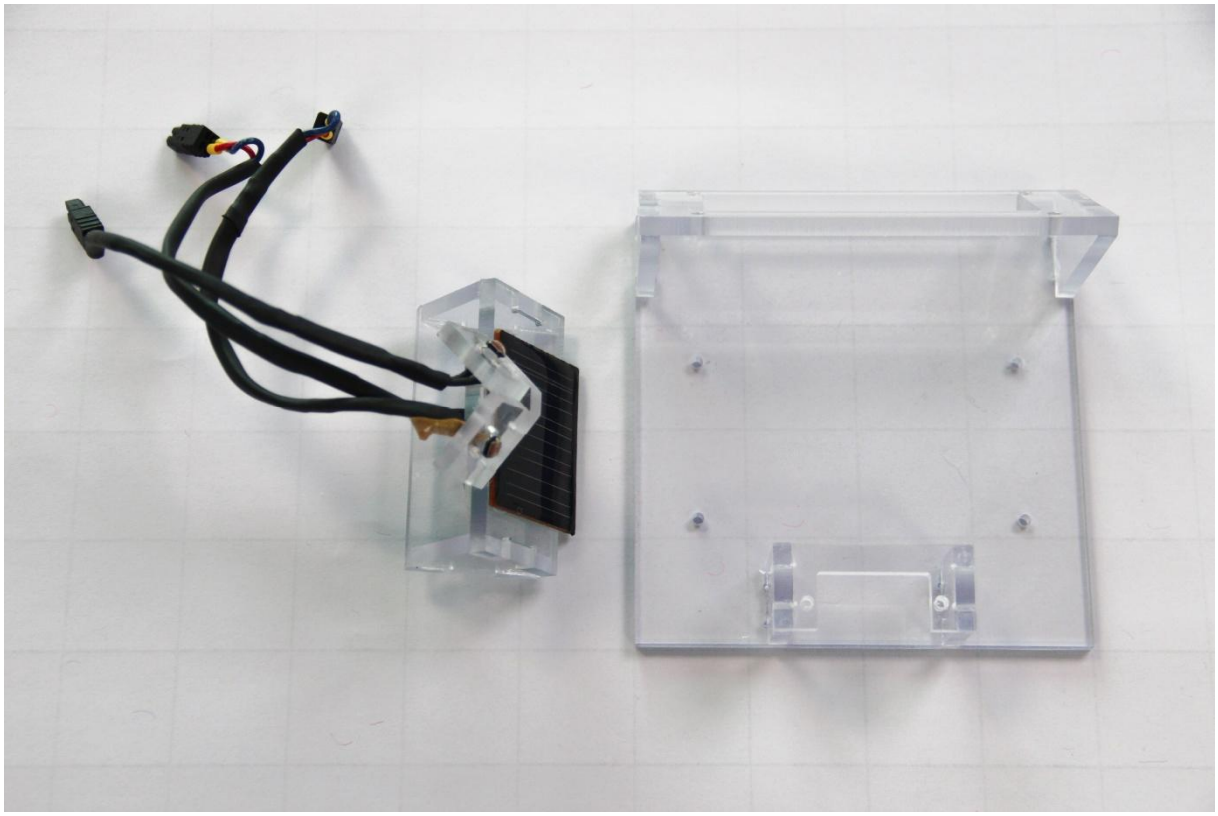


Abbildung 2 - Montiertes Gehäuse

2.1.2 Leiterplattenhalterung

Damit wir später die Leiterplatte einfach montieren können, ohne das Gehäuse zu beschädigen, bringen wir an der Bodenplatte vier Abstandsbolzen M2,5x5mm an, die wir auf der Unterseite mit Muttern der Größe M2,5 (nach ISO 1502 Norm) befestigen.

Die Leiterplatte kann so später einfach mit vier weiteren M2,5x5mm Linsenkopfschrauben befestigt werden.

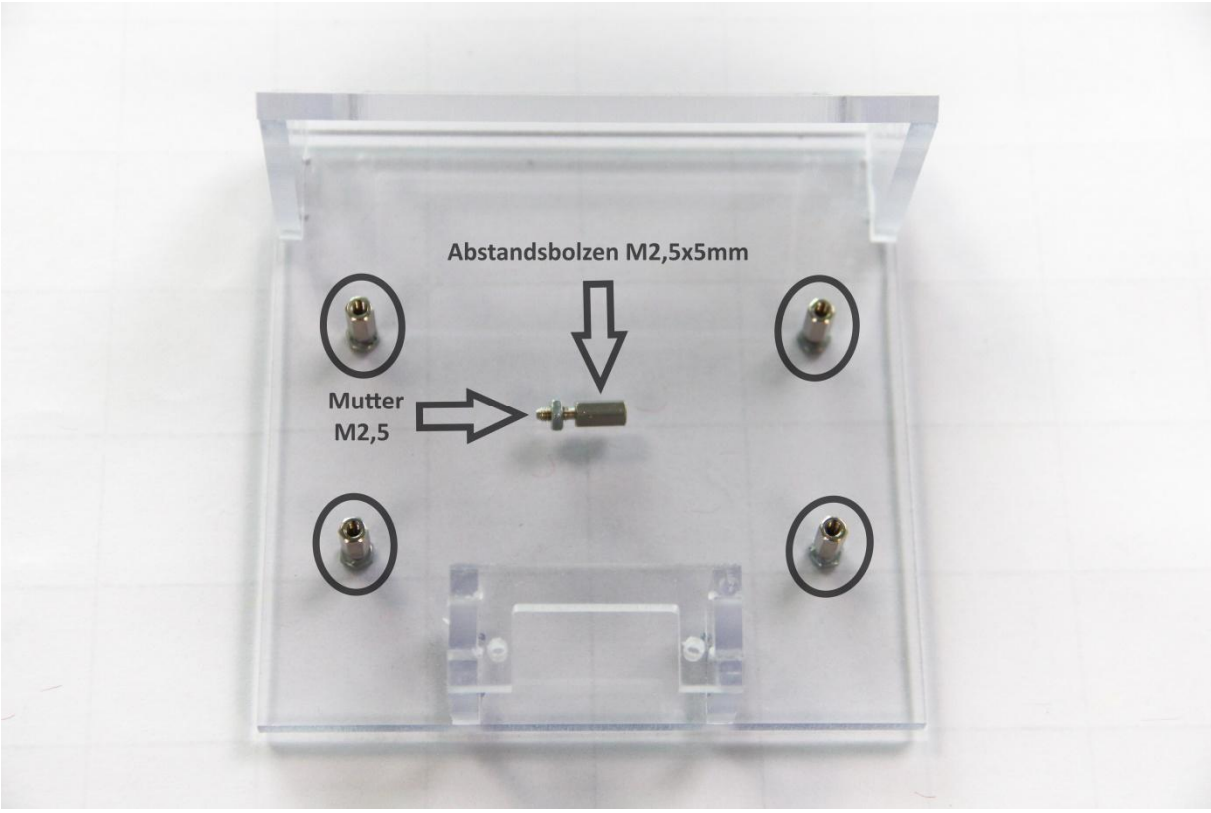


Abbildung 3 – Leiterplattenhalterung mit Abstandsbolzen

2.1.3 Kleben

Um die Steckverbindungen nun dauerhaft und stabil zu verbinden, verwenden wir Silicon. Dieses muss mit einer Spritze in die Steckstellen eingeführt werden und dort zirka 5 Stunden aushärten. In der Zwischenzeit können Vorexperimente und weitere Montageschritte durchgeführt werden.

Alternativ ist (je nach Arbeitsschutzbestimmungen vor Ort) auch die Verwendung eines geeigneten Kunststoff- oder Komponentenklebers denkbar.

Abbildung 4 - Klebestellen

Achtung:

Trotz seiner langen Aushärtungszeit verhält sich Silicon wie Kleber. Dementsprechend sollten ungewollte Rückstände, sowie Körperkontakt möglichst vermieden werden!

2.1.4 Bildschirmmontage

Als Nächstes montieren wir den Bildschirm. Dieser sollte bereits in das vorgefertigte Gehäuse passen (anderenfalls muss mit einer Pfeile nachgearbeitet werden), sodass er nur an vier Schrauben befestigt werden muss.

Die vier Befestigungen bestehen jeweils aus einer Schraube mit M3-Gewinde, einer Mutter hinter der Gehäusewand, einer Fächerscheibe vor dem Bildschirm und einer weiteren Mutter hinter dem Bildschirm (siehe Abbildung 5). Ein handfestes Anziehen der Muttern ist vollkommen ausreichend.

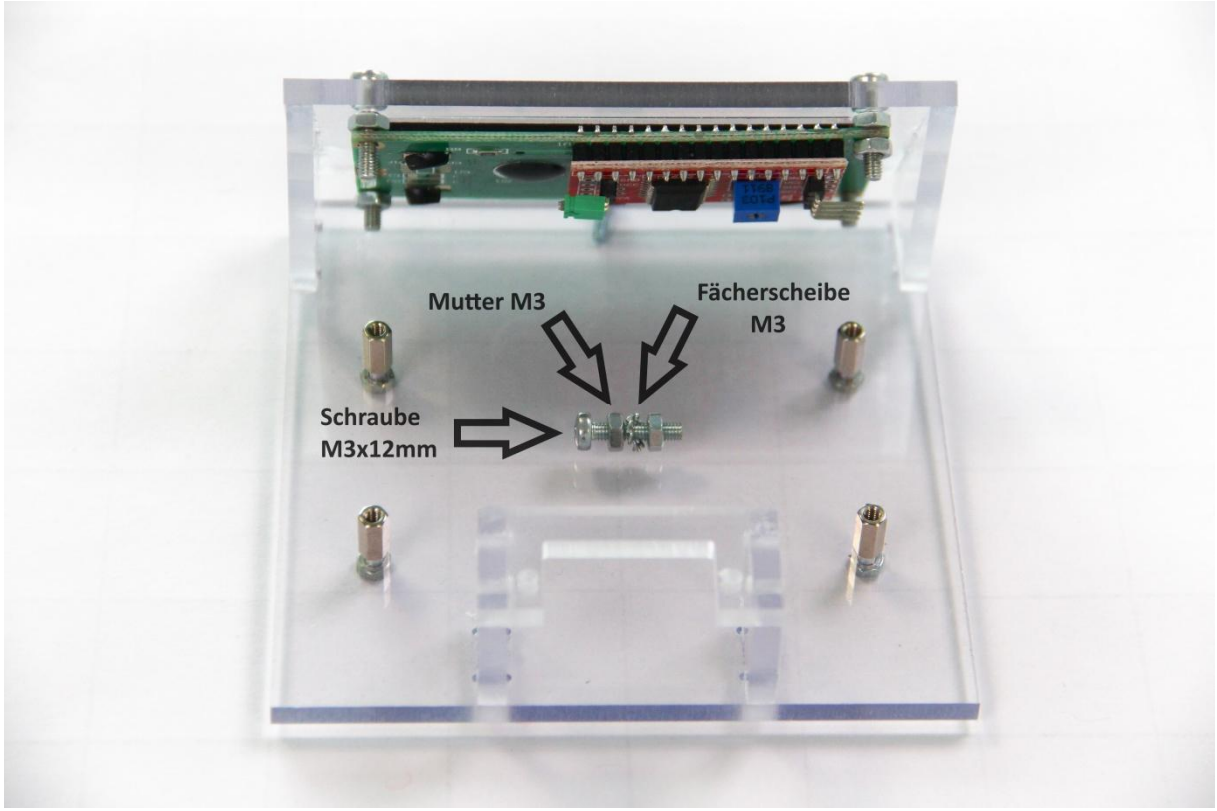


Abbildung 5 - Bildschirmmontage

2.1.5 Motormontage

Nun soll auf der anderen Seite des Gehäuses der Motor montiert werden.

Da die Fräsmaschine keine Gewinde bohren kann, muss dies manuell mit einem M2-Handbohrer erfolgen. Danach kann der Motor mit zwei M2-Schrauben befestigt werden.

Achtung:

Sollte sich der Gewindebohrer nur unter äußerster Gewalteinwirkung in das Kernloch ziehen, so muss dieses mit einer Handbohrmaschine und einem 1,7mm Bohrer aufgebohrt werden.

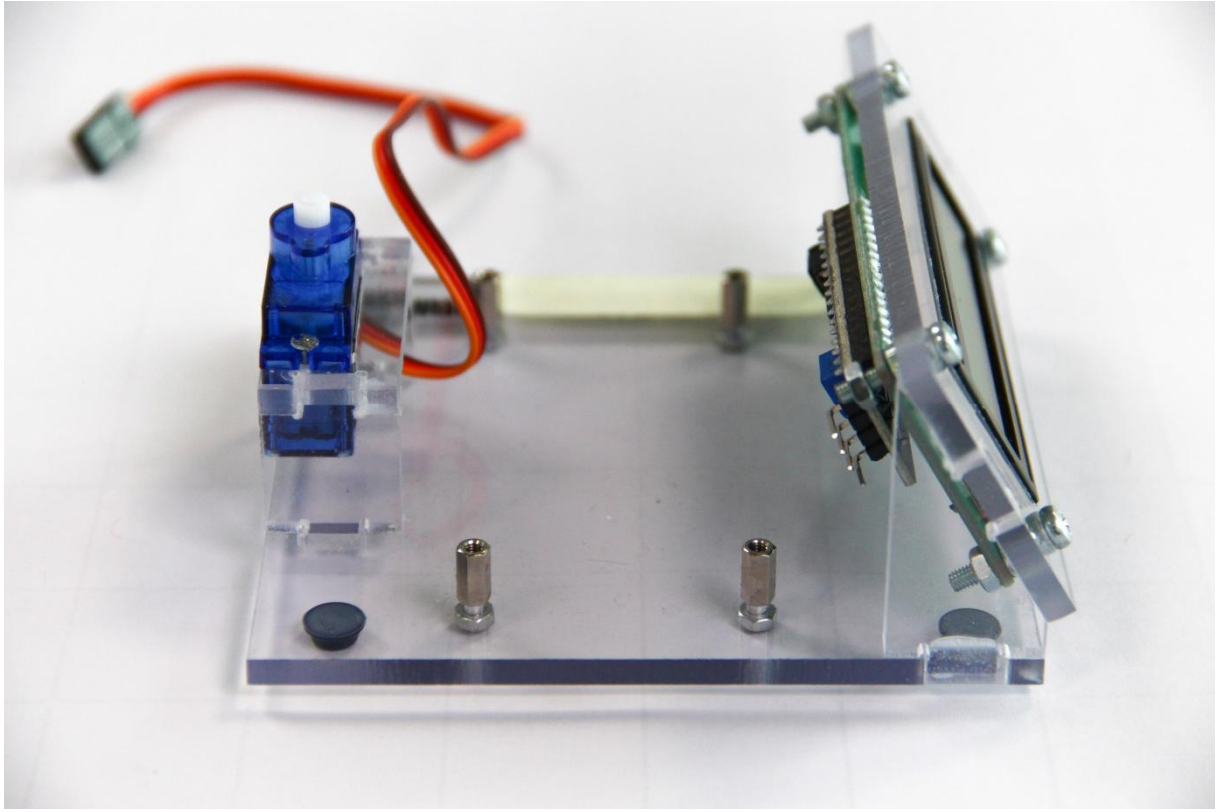


Abbildung 6 - Fertig montierter Motor

2.1.6 Solarpanelhalterung

Als nächstes muss die Halterung für das Solarpanel befestigt werden. Hierfür werden ein Servostern (Plastikkreuz) und zwei passende Schrauben (M2) benötigt, die beide im Motorzubehör (kleine Plastiktüte) enthalten sind.

Die beiden Schrauben müssen von unten durch den Servostern gedreht und dann in die dafür vorgebohrten Löcher in der Halterung geschraubt werden.

Danach wird der Servostern auf den Motor gesetzt und kann durch das dafür vorgesehene Loch in der Solarpanelhalterung mithilfe einer kleinen Schraube (ebenfalls im Motorzubehör) befestigt werden.

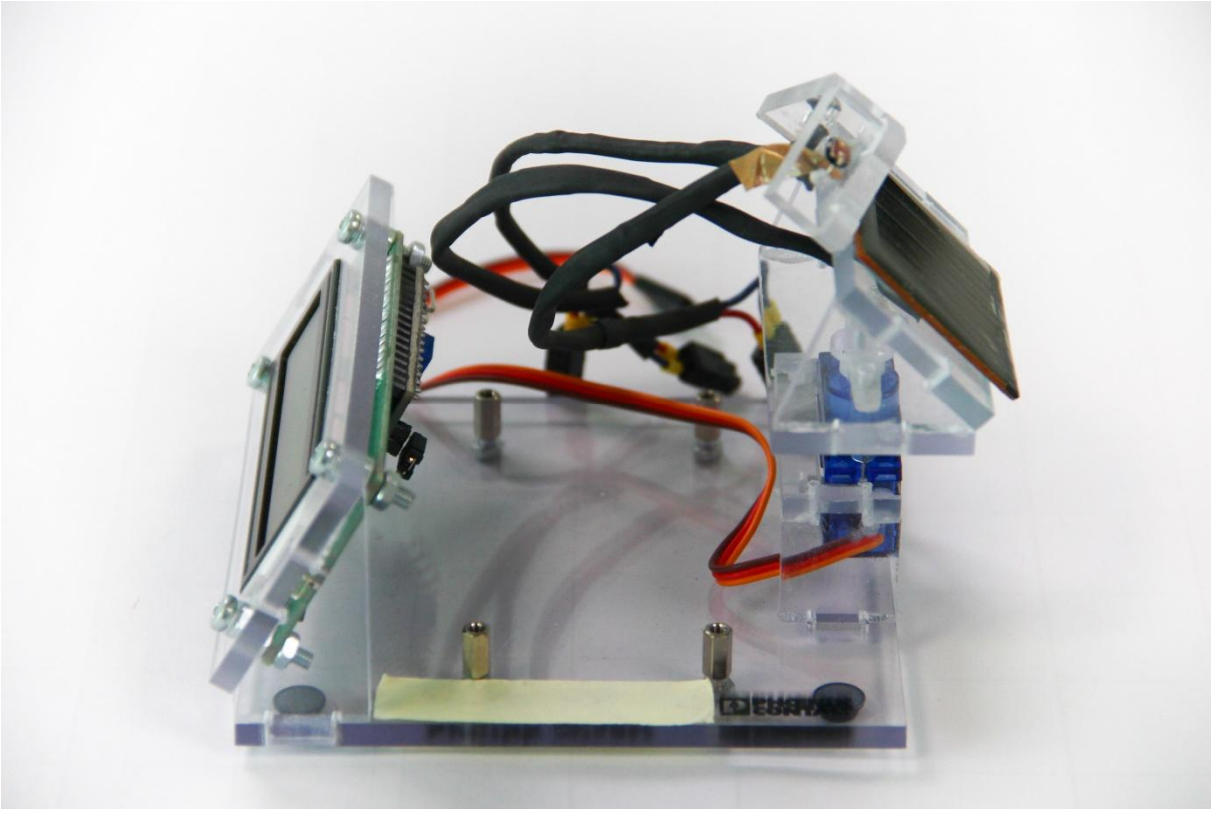


Abbildung 7 - Fertig montierte Solarpanelhalterung auf dem Servomotor

2.2 Die Platine

2.2.1 Der Schaltplan

Ohne eine elektrische Schaltung würde unsere Solarpanelnachführung nur gut aussehen, sich aber weder programmieren lassen, noch bewegen.

Für das Projekt benötigen wir zunächst einen Schaltplan, der die Eigenschaften (Größe, Ansteuerungsart etc.) der einzelnen Komponenten berücksichtigt. Dieser könnte beispielsweise so aussehen:

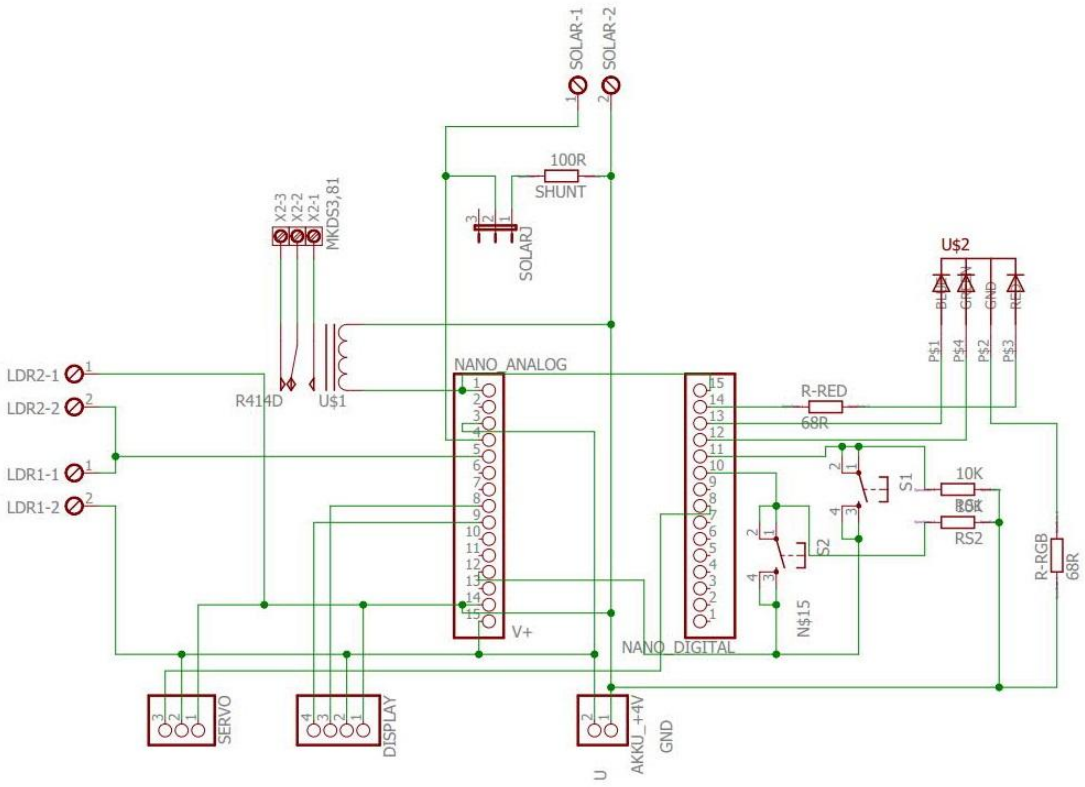


Abbildung 8 - Schaltplan der Solarpanelnachführung

2.2.2 Das Platinenlayout

Damit wir diesen Schaltplan nun nicht selbst auf einem Steckbrett stecken müssen, verwenden wir eine fertig gefräste Platine. Das Layout für eine solche Platine lässt sich relativ einfach in einem CAD-Programm, wie Eagle erstellen. Das fertig erstellte Layout wird dann in Leiterbahnen und Bauteile unterteilt.

Die Platine selbst besteht aus einem nicht-leitenden Grundmaterial, das von zwei Seiten mit gut leitendem Kupfer beklebt ist. An den Stellen, wo dieses Kupfer nicht leiten soll, wurde es vom Fräser entfernt.

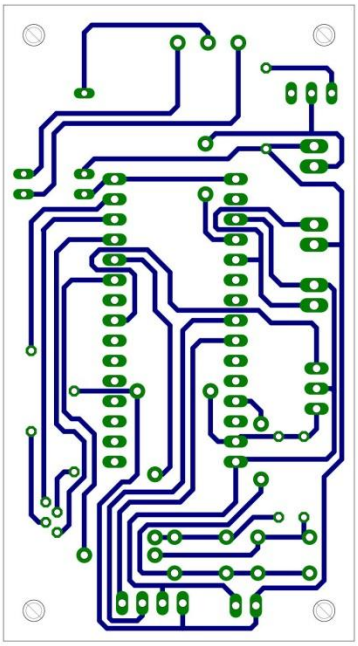


Abbildung 12 - Platinenunterseite

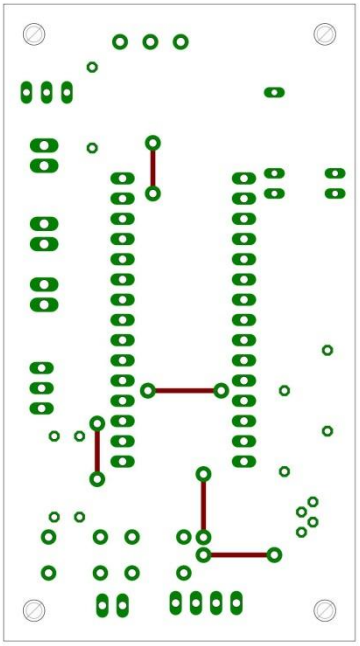


Abbildung 9 – Platinenoberseite

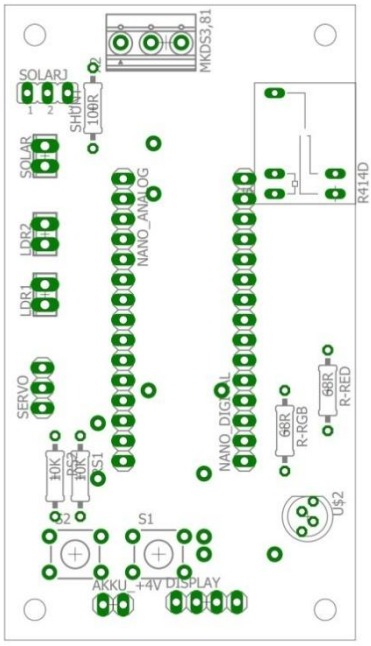


Abbildung 11 - Bestückungsplan

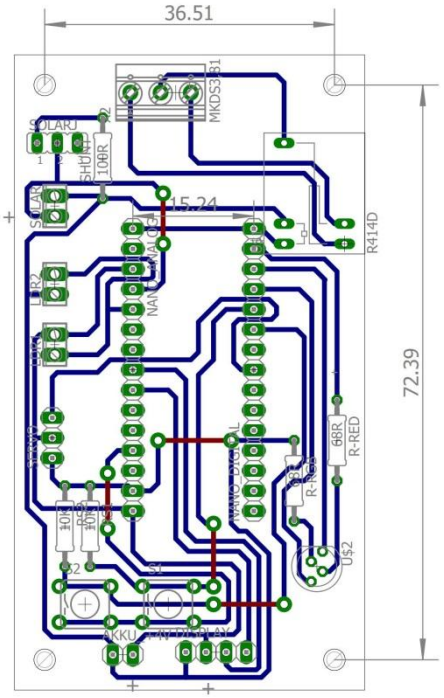


Abbildung 10 - Platinenlayout

2.2.3 Auflöten der Bauteile

Die Bauteile müssen gemäß dem Bestückungsplan (siehe Abbildung 11) aufgelötet werden.

Dafür empfiehlt sich die Verwendung bleifreien Lötzinns (Zinn-Silber-Kupfer-Legierung) mit einer Löttemperatur von 315 °C.

Alle Bauteile dieser Platine befinden sich auf der Oberseite (auf der sich nur wenige Leiterbahnen befinden) und müssen somit von der Unterseite der Platine her gelötet werden.

Beim Löten ist darauf zu achten, dass sowohl der anzulötende Draht, als auch die Lötstelle erhitzt werden und das Lötzinn nach dem Lötvorgang den eingelöteten Draht vollständig umschließt. Die Lötspitze sollte für eine bessere Wärmeübertragung möglichst groß gewählt werden, darf jedoch nicht mit anderen Lötkontakten in Berührung kommen.

Außerdem ist es wichtig, nach der Zugabe des Lötzinns zuerst das Lötzinn und dann die Lötspitze zu entfernen.

Bei unserer Platine ist zudem darauf zu achten, dass die RGB-LED richtig gepolt wird, da Dioden und somit auch Leuchtdioden nur in eine Richtung leitend sind. Wichtig ist daher vor allem, dass der längste Anschluss der LED als Kathode (Minuspol) an der richtigen Stelle eingelötet wird. Abbildung 13 ist hier zwingend zu beachten!

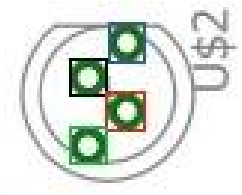


Abbildung 13 – Ausschnitt aus Abbildung 11, Lötplan für RGB-LED: blau = mittellanger, äußerer; schwarz = längster; rot = mittellanger, mittiger; grün = kürzester Draht

Achtung:

Der Mikrocontroller wird nicht auf die Platine gelötet!!!

Für ihn werden zwei Stiftleisten eingelötet, wo er später aufgesteckt werden kann.

Hinweis:

Normalerweise sollte man versuchen, Lötstellen aus bleifreiem Lot in der Elektronik immer bei 315 °C und nicht mit einer höheren Temperatur zu löten, damit empfindliche Bauteile nicht beschädigt werden.

Sollte diese Temperatur jedoch nicht ausreichend sein, kann die Löttemperatur etwas höher (ca. 340 °C) gewählt werden.

3. Die Schaltung und ihre Bestandteile

3.1 Der Mikrocontroller

3.1.1 Was ist ein Mikrocontroller?

Bei einem Mikrocontroller (kurz „MC“) handelt es sich um einen Computer, der sich auf einem einzigen Chip befindet (auch „**I**ntegrated **C**ircuit“, kurz „IC“ genannt). Intern besteht ein Mikrocontroller jedoch aus vielen einzelnen Bauteilen, die ihr von euren Rechnern zuhause kennt. Diese sind beispielsweise:

- Prozessor mit Rechenwerk (CPU mit ALU)
- Arbeitsspeicher (RAM)
- Nicht-flüchtiger Speicher (ROM; hauptsächlich zur Speicherung der Software)

Hinzu kommen je nach Anwendung noch diverse Peripherien, um mit der Außenwelt kommunizieren oder spezielle Aufgaben erfüllen zu können. Beispiele hierfür wären:

- Timer (PLLs und Frequenzteiler)
- Digitale Anschlüsse, General Purpose Outputs/ Inputs (GPIOs) genannt
- Spannungsmesser, sogenannte Analog-Digital-Wandler (ADCs)
- Analoge Ausgänge, sogenannte Digital-Analog-Wandler (DACs)
- PWM- bzw. PPM-Ausgänge zur Erzeugung präziser digitaler Signale bzw. Rechteckspannungen
- Bussysteme (z.B. USB, Ethernet, PCI, Interbus, Modbus, I²C, SPI etc.)

Im Gegensatz zu hardwaredefinierten Systemen, wie festen Schaltungen oder FPGAs („Field Programmable Arrays“), werden die Funktionen eines Mikrocontrollers erst durch eine Software definiert.

Im Praxiseinsatz sollte man bei der Wahl eines Mikrocontrollers nicht nur auf dessen Leistung, sondern auch seinen Energieverbrauch und Peripherien achten, um eine optimale Schaltung realisieren zu können.

3.1.2 Der ATmega328

Da unser Projekt mit einer relativ einfachen Schaltung auskommen soll und es sich um keine zeitkritische Anwendung handelt, verwenden auch wir einen Mikrocontroller.

Unsere Wahl fällt hierbei auf den ATmega328 von Atmel, da dieser dank der Arduino IDE einfach zu programmieren ist und alle benötigten Funktionen (PWM, PPM und ADC) vereint. Damit wir den Mikrocontroller nicht anlöten müssen und keine zusätzliche Schaltung zur Programmierung benötigen, verwenden wir eine Arduino Nano Platine, auf der diese Schaltungen, sowie eine Taktleitung bereits integriert sind.

3.2 Die Lichtsensorik

Damit unser Solarpanel sich immer zum hellsten und damit energiereichsten Punkt am Himmel drehen kann, benötigen wir eine Sensorik, die eben dies erfassen kann.

Der einfachste Weg hierfür ist die Verwendung von zwei lichtabhängigen Widerständen, sogenannten LDRs („**L**ight **D**ependent **R**esistor“), die im 45° Grad Winkel in jeweils eine der möglichen Drehrichtungen zeigen.

Diese Widerstände werden dann in Reihe geschaltet, wodurch sich der Spannungsabfall gemäß der Formel $U_{Ges} = U_{LDR\,1} + U_{LDR\,2}$ auf die beiden Widerstände verteilt.
Da die Spannung sich gemäß des Verhältnisses beider Widerstände zueinander verteilt, gilt für die Spannung über einem bestimmten Widerstand folgende Formel:

$$U_{1/2} = U_{Ges} \times \frac{R_{1/2}}{R_{Ges}}$$

Wenn wir nun den Spannungsabfall über einem der LDRs messen, können wir durch das Spannungsgesetz der Reihenschaltung den Spannungsabfall über den jeweils anderen LDR bestimmen.

Wir wollen den Spannungsabfall zwischen dem zweiten LDR und dem Minuspol des Akkus (Masse, GND, VSS oder V- genannt) bestimmen. Daher verbindet auf der Platine eine Leitung den zweiten Analog-Digitalwandler des Arduinos „A1“ und die Leitung zwischen beiden LDRs.

LDRs haben die Eigenschaft, dass ihr **Widerstand R bei höherer Lichteinstrahlung geringer** wird, wodurch hier der Spannungsabfall über ihnen kleiner wird.

Wenn nun beide LDRs gleichmäßig beleuchtet werden, ist ihr Widerstandsverhältnis zueinander gleich und somit fällt über beiden LDRs jeweils die Hälfte der Spannung ab (siehe Formel). Da in unserem Stromkreis eine Spannung von ca. 5 Volt anliegt, muss der Spannungsabfall über jedem LDR ungefähr 2,5 Volt betragen.

Wenn nun einer der LDRs verdunkelt wird, verschiebt sich der Spannungsabfall, was der Mikrocontroller mit dem Analog-Digital-Wandler erfassen kann.

Zu viel Theorie? Ein passendes Experiment gibt es auf Seite 27!

3.3 Der Stellmotor

Damit der Mikrocontroller die Solarzelle zur Sonne hinbewegen kann, benötigen wir einen Motor, der sich genau positionieren lässt. Hierfür bietet sich aufgrund seiner einfachen Ansteuerung ein sogenannter Servomotor an.

Ein Servomotor besteht aus drei Hauptbaugruppen. In ihm sind ein Elektromotor (üblicherweise mit Getriebe), ein Potentiometer (verstellbarer Widerstand) und eine Steuereinheit integriert. Die Steuereinheit erfasst über das Potentiometer die aktuelle Position der Motorachse. Sollte diese nicht mit dem über ein Steuersignal empfangenen Soll-Wert übereinstimmen, lässt die Steuereinheit den Motor so drehen, dass er seine Zielposition erreicht und stoppt ihn dort.

Das Steuersignal müssen wir mit unserem Mikrocontroller erzeugen. Bei dem Signal handelt es sich um ein sogenanntes PPM-Signal (**P**uls**p**ausen**m**odulation bzw. **P**ulse-**P**osition-**M**odulation). Bei diesem Verfahren erzeugt der Mikrocontroller Impulse, die alle dieselbe Länge (herstellerabhängig 18 – 21ms) haben. Die Pausenzeiten zwischen den gesendeten Impulsen variieren jedoch. Mithilfe der Pausenlänge zwischen den Impulsen lässt sich so eine analoge Information digital übertragen.

Der Servomotor bewegt sich in einem Bereich von 0 bis 180 Grad und die Pausenlänge liegt zwischen einer und zwei Millisekunden, weshalb der Motor sich bei einer 1ms Pause auf 0 Grad, bei 1,5ms Pause auf 90 Grad und bei 2ms Pause auf 180 Grad dreht.

Da nicht jeder Pin an unserem Mikrocontroller ein ausreichend genaues PPM-Signal generieren kann, verwenden wir in unserem Projekt zur Ansteuerung des Servomotors Pin 9 bzw. „D5“. Geeignete Pins werden auf den Arduino-Platinen oft mit einem „~“-Zeichen markiert.

Außerdem müssen wir das rote Kabel (V+) und das schwarze Kabel (V-) des Servos mit der Spannungsversorgung verbinden.

Ein Experiment zur Ansteuerung des Motors gibt es auf Seite 27!

3.4 Das Relais

Ein Mikrocontroller hat nur eine geringe zulässige Stromaufnahme und arbeitet meist mit einer möglichst kleinen Spannung, um einer Wärmeentwicklung entgegen zu wirken und zu verhindern, dass kleinste Bauelemente auf dem Controller beschädigt werden.

Wenn nun jedoch ein Motor, eine Glühbirne oder ein anderes Bauteil, welches viel Energie benötigt und somit auch einen hohen Stromfluss verursachen würde angesteuert werden soll, ist ein bloßer Mikrocontroller nicht ausreichend, da dieser solche Geräte nicht mit Strom versorgen könnte und wahrscheinlich sogar zerstört werden würde.

Daher besteht eine Schaltung mit Mikrocontrollern und Verbrauchern ab einer bestimmten Größe immer aus **Steuerstromkreisen** mit geringer Spannung, die der Informationsübermittlung dienen und **Laststromkreisen**, die die benötigte Energie für die Aktoren (Motoren, Lampen, Heizelemente etc.) liefern.

Damit die höhere Spannung eines Laststromkreises nicht auf einen Steuerkreis überspringen kann, müssen beide Kreise galvanisch, also elektrisch voneinander getrennt sein, und dennoch muss der Steuerkreis Einfluss auf die Aktoren im Laststromkreis nehmen können.

Eine Möglichkeit dies zu realisieren stellt ein Relais da. Dabei handelt es sich um einen Schalter, der jedoch nicht mechanisch, sondern elektrisch bewegt werden kann. Dies funktioniert mithilfe von Elektromagneten.

Je nach Bauweise unterscheidet man zwischen Öffnern („normal closed“, kurz NC), die bei Ansteuerung den Laststromkreis unterbrechen, Schließern („normal open“, kurz NO), die das Gegenteil bewirken und Wechslern, die zwischen zwei Laststromkreisen umschalten können.

Auf unserer Platine ist ein Wechsler verbaut. Dieser könnte je nach Beschaltung als Öffner, Schließer oder zum Wechseln zwischen zwei Stromkreisen (z.B. zwischen einer roten und einer grünen Ampel) genutzt werden.

In der Industrie werden oft sogenannte „Schütze“ eingesetzt. Dabei handelt es sich um spezielle Relais, die die Verbindungen im Laststromkreis immer jeweils an zwei Stellen schließen oder unterbrechen, was eine höhere (Prozess-) Sicherheit zur Folge hat.

Für unser späteres Projekt wird kein Relais mehr benötigt - es soll rein dem Verständnis elektronischer Anlagen dienen.

Ein Experiment hierzu befindet sich auf Seite 28!

3.4.1 Taster

Die meisten elektronischen Schaltungen lassen sich durch menschliche Einwirkung beeinflussen. Dies kann beispielsweise durch einen Knopfdruck, einen Mausklick oder eine Tastatur geschehen. Alle diese Möglichkeiten bedienen sich eines sogenannten Tasters, eines Schalters, der bei Druck einen Stromkreis schließt und ihn bei Freigabe wieder öffnet.

Auch zwei solcher Taster sind auf unserer Platine verbaut. Wenn diese nicht gedrückt werden, trennen sie einen Steuerkreis auf und es liegen 0 Volt Spannung an, was der Mikrocontroller mithilfe seiner digitalen Eingänge (GPIOs) leicht als logische 0 erkennen kann.

Wird der Schalter nun gedrückt, fällt über einem Widerstand hinter dem Schalter eine Spannung ab, die am GPIO anliegt, was als logische 1 ausgewertet werden kann. Bei nur einem Widerstand fällt die gesamte Spannung des Steuerkreises – hier also 5 Volt - am Widerstand ab.

Ein Experiment zum Üben mit den Tastern findest du später auf Seite 28!

3.4.2 Die RGB-LED

Leuchtdioden sind aus elektrischen Schaltungen nicht mehr wegzudenken. Im Gegensatz zu normalen Glühlampen ist ihre Verlustleistung (vor allem in Form von Wärme) deutlich geringer und mit ihnen lassen sich klar definierte Lichtwellen erzeugen. Außerdem besitzen sie die Eigenschaft, Strom nur in einer bestimmten Richtung zu leiten.

Auf unserer Platine ist eine sogenannte RGB-LED verbaut. Dabei handelt es sich genau genommen um drei LEDs, mit den Farben rot (λ = 636nm), grün (λ = 636nm) und blau (λ = 472nm, die in einem einzigen Gehäuse verbaut sind.

Aus einer Kombination dieser Farben, können theoretisch alle anderen sichtbaren Farben erzeugt werden. Hierfür müssen wir jedoch die Stärke der einzelnen Teilfarben, also ihre Leuchtkraft anpassen können.

Dies würde uns eventuell durch eine analoge Spannungsänderung gelingen. Allerdings ist die Erzeugung eines genauen analogen Signals mithilfe eines Mikrocontrollers sehr schwer.

Stattdessen machen wir uns die Pulsweitenmodulation (PWM) zu nutzten.

Die Länge eines solchen rechteckförmigen Signals ist klar als Periodendauer T definiert und wird in Sekunden, häufig aber auch als Frequenz in Hertz angegeben. In diesem Zeitraum wechselt das Signal von einem hohen Spannungslevel (hier 5 Volt) auf ein niedriges Spannungslevel (üblicherweise 0 Volt).

Umso länger sich das Signal auf dem höheren Level befindet, desto höher ist der sogenannte Tastgrad (engl: „duty circle“). Dieser gibt an, wieviel Prozent der Zeit der Periodendauer sich das Signal auf dem höheren Level befindet.
Mit der so erzeugten Rechteckspannung lassen sich nicht nur digitale Signale übermitteln, sondern mit ihr lässt sich auch die Energiezufuhr die LED-Farben kontrollieren, denn die gelieferte Energie entspricht dem Integral der Leistung P. Erhöhen wir also unseren Tastgrad, wird die jeweils angesteuerte Farbe heller. Mithilfe der drei Pluspole unserer LED (für jede Farbe einer), können wir nun nahezu präzise jede Farbenkombination erzeugen.

Die RGB-LED anzusteuern kannst du praktisch in einem Experiment auf Seite 29 üben!

4. Programmierung

4.1 Entwicklungsumgebung und Programmiersprache

Das Programmieren geschieht über eine Programmiersprache.

Programmieren lässt sich der ATmega328, wie fast alle Mikrocontroller in Assembler oder C. Mithilfe der Arduino IDE („**I**ntegrated **D**evelopment **E**nvironment“), einer extra für diese Plattform entwickelten Entwicklungsumgebung lässt sich der Controller jedoch auch in einem einfacheren C++ Dialekt programmieren. Da die sogenannten „Sketches“ – wie die Programmquelltexte genannt werden – einfacher zu verstehen sind, als die jeweiligen Alternativen, verwenden wir für unser Projekt die **Arduino IDE 1.8 oder höher**.

Bei C++ bzw. dem Arduino Dialekt handelt es sich um objektorientierte Programmiersprachen, in denen Klassen angelegt werden können, die verschiedene Eigenschaften eines Objektes als sogenannte Variablen speichern und dem Programmierer verschiedene Funktionen eines Objektes zur Verfügung stellen.

Beispiel: So könnte ein „Hund“ als Klasse angelegt werden, die dann sehr wahrscheinlich die Variablen „Geschlecht“, „Rasse“ und „Fellfarbe“, so wie die Funktionen „bellen“, „springen“ und „rennen“ enthalten würden. Objektorientierte Programmiersprachen sollen somit gewissermaßen unsere natürliche Umwelt wiederspiegeln.

4.2 Klassen

Während bei den meisten objektorientierten Sprachen Klassen als ein übliches Mittel zur besseren Übersichtlichkeit des Quelltextes verwenden, verzichtet die Arduino IDE bewusst auf diese Möglichkeit, da die Sketches meist recht übersichtlich bleiben und aufgrund des kleinen Speichers der meisten Controller ohnehin in ihrer Größe begrenzt sind.

Die Arduino IDE legt also nur eine Klasse mit allen Funktionen und Variablen an, die für den Programmierer unsichtbar bleibt.

4.3 Variablen

Variablen dienen zur Speicherung von Werten, die später im Programm nochmals benötigt werden (für Berechnungen, zum Speichern, zum Vergleichen etc.).

Beim Anlegen einer Variable muss dem Controller zunächst mitgeteilt werden, welche Art von Wert gespeichert werden soll, denn davon hängt unter anderem die Größe des Speichers ab, die für diesen Wert reserviert werden muss. Die Verwendung folgender Variabletypen ist für unser Projekt sinnvoll:

Typ	Beschreibung	Speicherbedarf
int	Ein „Integer“ ist eine Ganzzahl mit Vorzeichen zwischen -32.786 und +32.786	16 Bit
long	Speichert große Ganzzahlen mit Vorzeichen von ca. -2 bis +2 Milliarden	32 Bit
float	Kommazahl (z.B. für Brüche)	32 Bit
char	Einzelner Buchstabe	8 Bit
String	Speichert eine Buchstabenkette (Wort bis ganze Sätze)	
bool	Speichert wahr oder falsch (1 oder 0)	1 Bit

Grundsätzlich sollte natürlich immer der Variabletyp verwendet werden, indem alle nötigen Werte gespeichert werden können, der aber einen möglichst geringen Speicherbedarf hat.

Damit das Programm nun weiß, dass es einen Speicherplatz reservieren muss, wird eine Variable wie folgt „deklariert“:

```
Variabletyp Name;
```

Die Deklaration oder Zuweisung einer Variablen muss immer mit „;“ beendet werden. Der Name ist frei wählbar, sollte dabei aber möglichst kurz und prägnant gehalten werden, denn um der Variable nun einen Wert zuzuweisen würde man einfach schreiben:

```
Name = x;
```

Beispiele:

```
int zahl;
zahl = 1;

String text;
text = "Hallo Welt";
```

oder auch:

```
float Prozent = 0.15;
```

Achtung:

Bei der Benennung und späteren Verwendung von Variablen muss Groß- und Kleinschreibung beachtet werden. Auch darf ein Name nie mit einer Zahl beginnen!

4.4 Funktionen

Grundsätzlich sollten Codeabschnitte, die sich wiederholen als Funktion deklariert werden. In eine Funktion können Werte in Form von Variablen übergeben werden, damit diese dort verarbeitet werden können und Funktionen können auch einen bestimmten Wert zurückgeben. Grundsätzlich werden Funktionen wie folgt deklariert:

```
Rückgabetyp Funktionsname (Variabletyp1 Variablename1, Variabletyp2
Variablename2,…) {



        return Wert;
}
```

Für den Rückgabetypen können dieselben Typen, wie für Variablen gewählt werden. Der Funktionsname ist ebenfalls, wie bei Variablen frei wählbar. Alle Anweisungen müssen sich zwischen den geschweiften Klammern („{ }“) und vor dem „return“ befinden.

Beispiel:

```
int add (int Wert1, int Wert2) {
  return Wert1 + Wert2;
}
```

Eine besondere Art von Funktion lässt sich mit dem Typ „void“ („Nichts“) realisieren. Diese Art Funktion benötigt keinen „return“-Befehl, da sie rein zum Ausführen von Befehlen dient.

4.5 Neues Projekt

Startet man nun die Arduino IDE und legt über den Menüpunkt „Datei à Neu“ ein neues Projekt an, so enthält dieses direkt zwei Funktionen des Types „void“, also ohne Rückgabewert.

Die erste vorbereitete Funktion heißt „setup“. Diese wird direkt nach Start des Mikrocontrollers einmal ausgeführt. Hier sollten daher alle Befehle eingefügt werden, die zur Initialisierung des Systems notwendig sind.

Die zweite Funktion heißt „loop“. Sie wird nach Beendigung des Setups unendlich oft, also in einer Schleife ausgeführt. In diese Funktion sollten alle Befehle eingefügt werden, die der Mikrocontroller regelmäßig ausführen soll.

4.6 Bibliotheken

Standartmäßig und für den Benutzer verborgen integriert die IDE bereits einige Standartbefehle (z.B. analogRead(*Port*) zum einlesen der analogen Eingänge).

Damit alle weiteren Befehle verwendet werden können, müssen diese zunächst aus Quelltextsammlungen, sogenannten Bibliotheken („Libraries“) importiert werden.

Für alle von uns verwendeten elektronischen Komponenten (Servomotor, LCD-Display, USB) wurden bereits Bibliotheken entwickelt, die wir kostenlos im Internet zur Verfügung gestellt bekommen und einfach über einen #include-Befehl zu unserem Code hinzufügen können.

Bibliotheken lassen sich einfach installieren, indem man sie in den Ordner „C:\\…\Documents\Arduino\libraries\“ entpackt bzw. kopiert und die IDE einfach neu startet.

Um die Befehle nun im Quelltext verwendbar zu machen, müssen wir vor allem anderen Code und außerhalb von geschweiften Klammern für jede Bibliothek ein „#include *Bibliothekname*;“ schreiben.

Wir benötigen folgende Bibliotheken:

- LiquidCrystal_I2C zur Ansteuerung des Displays über I²C-Bus
- Wire (standartmäßig installiert) zur Verwendung von UART/ USB und I²C
- Servo (standartmäßig installiert) zur Ansteuerung des Servos

Die ersten drei Zeilen des Quellcodes müssen demnach letztlich so aussehen:

```
#include Wire;
#include Servo;
#include LiquidCrystal_I2C;
```

Alle zur Programmierung benötigten Befehle sollten nach der Inklusion der Bibliotheken zur Verfügung stehen. Diese findet ihr im Anhang dieses Leitfadens.

Achtung:

Der Name („Namespace“) der Bibliothek ist nicht zwangsweise der Name des eingefügten Ordners!

4.7 Befehle

Um einen inkludierten Befehl oder eine geschriebene Funktion nun auszuführen, müssen diese aufgerufen werden. Dabei ist zu beachten, welche Argumente, also welche Werte der Befehl benötigt. Diese Argumente müssen dann in der richtigen Reihenfolge in Klammern hinter dem eigentlichen Befehl übergeben werden:

```
Befehl(Argument1, Argument2, …);
```

Bei Befehlen, die einen Wert zurückgeben, also externen Funktionen, ist es oft sinnvoll, den Rückgabewert in einer Variable zu speichern:

```
Variablename = Befehl(Argument1, Argument2, …);
```

Beispiele:

```
int value = analogRead(A1);
Serial.println(value);
```

4.8 Das Programm schreiben

Mit dem oben erworbenen Wissen können wir nun ein Programm schreiben.

Für das Programm werden normalerweise nur die Befehle im Anhang (siehe Befehlsliste) benötigt, die durch die genannten Bibliotheken integriert wurden. Solltest du jedoch weitere Befehle oder Erklärungen benötigen, so findest du diese auch ausführlich in der offiziellen Dokumentation der Arduino IDE: https://www.arduino.cc/reference/en/

Probiere es einfach mal aus!!! Eine mögliche Lösung findest du im Anhang.

4.9 Programm brennen

Damit wir unser geschriebenes Programm nun auf den Mikrocontroller aufspielen, also „brennen“ können, müssen wir zunächst einige Vorbereitungen treffen:

1. Der Arduino Nano muss über ein Mini-USB-Kabel mit dem Computer verbunden werden.
2. Auf dem Computer muss die Software „Arduino IDE“ gestartet werden.
3. Die Version der IDE sollte 1.8 oder höher sein (sonst Version erneuern).
4. Unter dem Menüpunkt „Werkzeuge“ müssen folgende Optionen eingestellt sein:
    a. Board: Arduino Nano
    b. Port: Com1 à bei Fehlfunktion ändern (Gerätemanager befragen)
    c. Programmer: AVRISP mkII
5. Nun kann die Software geschrieben und mit dem Rechtspfeil im Schnellzugriff („Hochladen“) auf den Mikrocontroller „gebrannt“ werden. Die Software sollte nun automatisch starten!

4.10 Hintergrundwissen: Was passiert mit meinem Programm?

Das in C++ oder einer anderen Sprache geschriebenen Programm ist für einen Mikrocontroller noch nicht verständlich.

Daher muss es vor dem Hochladen in einen Maschinencode (die berühmten Nullen und Einsen) übersetzt werden. Dies geschieht beim sogenannten „Kompilieren“ durch ein dafür geschriebenes Programm, den sogenannten „Compiler“.

Dabei prüft der Compiler zunächst, ob der Syntax der Programmiersprache eingehalten wurde, das heißt, ob alle Befehle und Zeichen richtig geschrieben wurden und an korrekter Stelle stehen (z.B. nach jedem Befehl ein „;“).
Bei einigen Compilern folgt danach noch eine inhaltliche (semantische) Überprüfung.

Sollte der Compiler einen Fehler feststellen, wird der Vorgang abgebrochen und ein Fehler an die Entwicklungsumgebung zurückgegeben.

Anderenfalls übersetzt der Compiler den Code in eine Reihenfolge aus Einsen und Nullen. Prinzipiell handelt es sich dabei einfach nur um Schaltzustände des Prozessors.

Im Falle des ATmegas328 ist jeder Befehl exakt 16 Bits, also 16 Einsen oder Nullen lang. Neben einem bestimmten Befehl (z.B. Addieren, Subtrahieren, Kopieren etc.) enthalten einige Befehlsketten auch Registeradressen. Beim Addieren müssen beispielsweise zunächst zwei Zahlen aus einer bestimmten Speicherstelle (einem Register) ausgelesen, um dann addiert zu werden.

Die entstandene Binärdatei wird vom Compiler über einen Datenbus (hier USB) in die Register des Mikrocontrollers geschrieben.

Ein zeitlich klar definiertes Taktsignal sorgt danach dafür, dass alle 16-Bit-Befehle nacheinander verarbeitet werden.

Das Programm wird nun ausgeführt!!!

5. Experimente

Das folgende Kapitel enthält einfache Experimente, die dem besseren Verständnis der einzelnen Bauteile auf der Platine, sowie als Programmierübungen dienen sollen.

Experiment 1 – Spannungsmessung:

Bestimme die Spannung über dem zweiten LDR, indem du den ADC A1 des Arduinos ausließt.

Vorgehensweise:

Generell lässt sich ein analoger Spannungswert an allen analogen Eingängen auslesen, die immer mit „A*0*“ – A*X* bezeichnet sind. Um den Wert des Analog-Digital-Konverters zu erhalten, lässt sich der Befehl „**analogRead(A*X*);**“ anwenden. Dieser gibt einen ganzzahligen Wert (Integer) zwischen 0 und 1023 (also 10-Bit) aus, wobei 0 auch 0 Volt und 1023 dem vollen Spannungswert, also 5 Volt entsprechen würde.

Hinweis:

Um dir das Ergebnis anzeigen zu lassen, kannst du die Werte über USB an deinen Computer senden und in der Arduino IDE unter **Werkzeuge à Serieller Plotter** als Graph anzeigen lassen.
Füge dafür im Setup() deiner Software den Befehl „**Serial.begin(9600);**“ ein. Im Loop() kannst du nun mit dem Befehl „**Serial.println(„Beispieltext“);**“ Werte übermitteln.

Experiment 2 – Servotest:

Versuche nun den Servo über die Servobibliothek der Arduino IDE anzusteuern, sodass er sich von einer Seite zur anderen Seite und zurück dreht.

Gehe dabei davon aus, dass…

- … alle Leitungen bereits auf der Platine enthalten sind.
- … die **Signalleitung** des Servos mit **Pin 9** des Mikrocontrollers verbunden ist.
- … der Servo sich nur zwischen 10° und 170° drehen sollte, um ein Durchdrehen zu vermeiden.

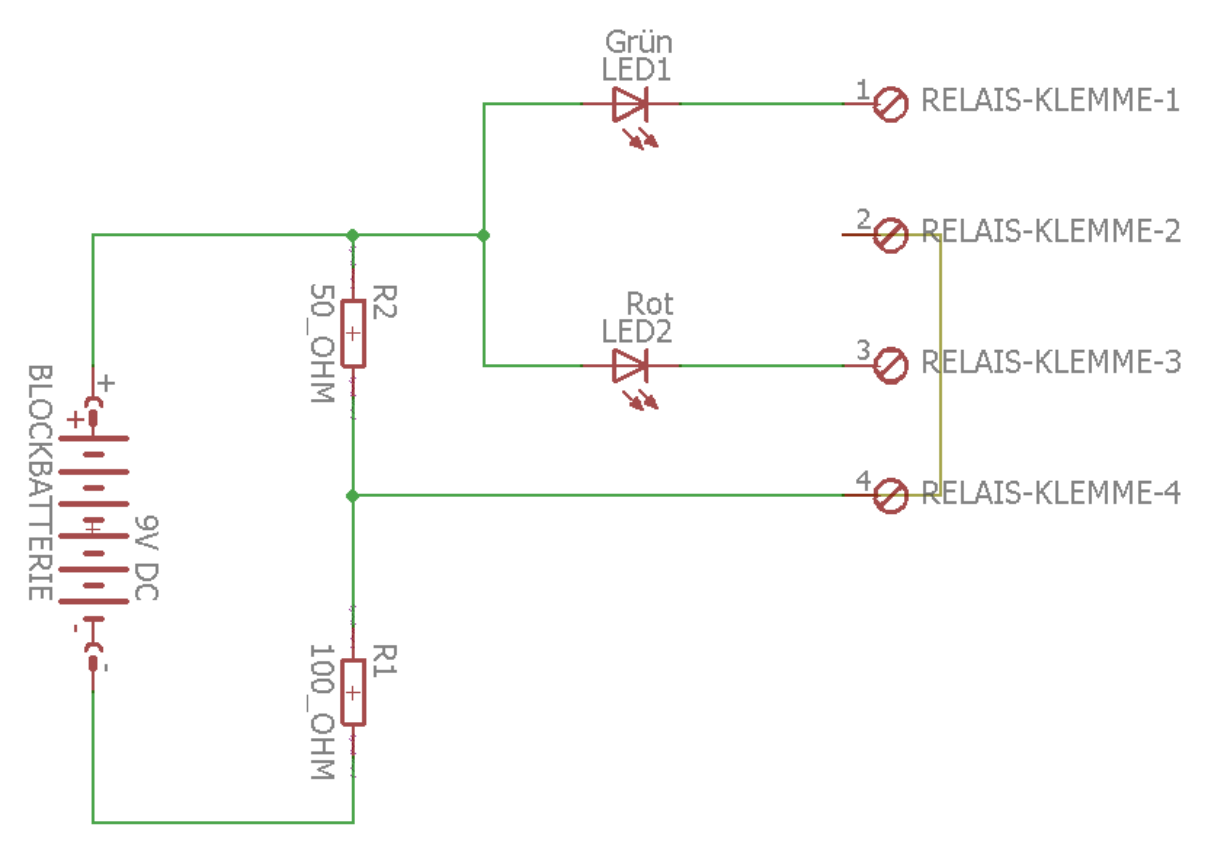
Du benötigst für diesen Versuch keine weitere Schaltung!

Experiment 3 – Ampel:

Verwende nun folgende Bauteile, um dir eine Ampelschaltung aufzubauen:

- 1x Widerstand mit 100 Ohm
- 1x Widerstand mit 50 Ohm
- 1x 9V Blockbatterie mit Batterieclip
- 1x LED rot
- 1x LED grün

Baue nun folgende Schaltung auf einem Steckbrett auf:



Hinweis: Anschluss 2 und 4 der Platine sind bereits verbunden!

Wenn du alles richtig gemacht hast, sollte nun eine LED leuchten. Schreibe nun eine Software, die…

- … das **Relais** über **Pin 16** des Arduinos ansteuert und so den Zustand des Wechslers ändert.
- … abwechselnd die rote und die grüne LED durch Wechsel des Relaiszustandes zum Leuchten bringt.

Kleine Herausforderung:

Nutzte nun die verbauten Taster, um zwischen den beiden LEDs umzuschalten.

Experiment 4 –Disco:

Bringe nun die RGB-LED auf der Platine in verschiedenen Farben zum Leuchten. Alle dafür benötigten Leiterbahnen sind bereits integriert, sodass du nur den Mikrocontroller programmieren musst (siehe Befehlsliste im Anhang).

Nutzte zur Steuerung der Helligkeit die Pulsweitenmodulation (PWM) an den Pins…

- 15 für rot
- 14 für grün
- 13 für blau

Kleine Herausforderung:

Erzeuge einen Farbverlauf mithilfe einer Schleife in deinem Programm.

I. Befehlsverzeichnis für Solarpanelnachführung

Die folgende Liste enthält alle zur Programmierung benötigten Befehle. Alle unterstrichenden Befehle müssen direkt am Anfang des Programmes stehen, sofern die anderen Befehle aus dem jeweiligen Tabellenabschnitt verfügbar sein sollen.

Wichtig:

- Unterstrichende **„#include“-Befehle** müssen am Anfang stehen, damit alle anderen Befehle aus den jeweiligen Tabellenabschnitt verfügbar sein!
- Hinter jedem Befehl muss ein „**;**“ stehen!

Deklaration von der Variablen		
Deklaration	Beispiel	Erklärung
int	int xy = 0;	Speichert eine 16 Bit Zahl zwischen -32.768 und 32.768
float	float xy = 0;	Speichert eine 32 Bit Kommazahl
double	double xy = 0;	Speichert eine 64 Bit Kommazahl (präziser als float)
Ein- und Ausgabefunktionen		
Befehl	Beispiel	Erklärung
int analogRead(String ADC-Bezeichner);	variable = analogRead(A0);	Gibt eine Zahl zwischen 0 und 1023 zurück. Diese entspricht dem Wert der aktuell am Analog-Digital-Wandler anliegenden Spannung. 0 entspricht hier 0 Volt und 1023 entspricht 5V.
int digitalRead(int Anschlussnummer);	digitalRead(9);	Fragt ab, ob an einem Anschluss Spannung anliegt oder nicht und gibt 1 oder 0 zurück.
digitalWrite(int Anschlussnummer, HIGH oder LOW);	digitalWrite(9, HIGH);	Legt an Pin 9 eine Spannung von 5 Volt (HIGH) oder 0 Volt (LOW) an.
analogWrite(int Anschlussnummer, 0 bis 255);	analogWrite(9,125);	Erzeugt ein PWM-Signal mit einem Tastgrad zwischen 0 und 100 %. 0 entspricht hierbei 0 % und 255 entspricht 100%.
Funktionen zur Steuerung des Liquid Crystal Displays		
Syntax	Beispiel	Erklärung
#include <Wire.h>		Bindet die Bibliothek ein, die dem Arduino eine Kommunikation über I2C erlaubt.

#include <LiquidCrystal_I2C.h>		Macht alle Befehle, die benötigt werden, um den Bildschirm anzusteuern zugreifbar.
LiquidCrystal_I2C name(I2C-Adresse, Spalten, Zeilen);	LiquidCrystal_I2C lcd(0x27, 16, 2);	Legt im System einen LCD-Display an (Initialisiert ein Objekt), dessen Eigenschaften sich steuern lassen.
name.begin();	lcd.begin();	Startet Verbindung zu LCD-Display über I2C-Bus
name.backlight();	lcd.backlight();	Aktiviert die Hintergrundbeleuchtung des LCDs. Der Kontrast muss manuell eingestellt werden!!!
name.print(Text);	lcd.print("TEST");	Gibt an der aktuellen Position des imaginären Bildschirmcursors den angegebenen Text aus.
name.setCursor(Spalte, Reihe);	lcd.setCursor(0,15);	Setzt den imaginären Bildschirmcursor an die angegebene Stelle. Die Nummerierung der Zeilen und Spalten beginnt bei 0!!!
Befehle zur Ansteuerung des Servomotors		
Syntax	Beispiel	Erklärung
#include <Servo.h>;		Lädt alle Befehle, die zur Ansteuerung des Servomotors benötigt werden.
Servo name;	Servo motor1;	Legt einen Servo als Objekt an.
name.attach(int Anschlussnummer);	motor1.attach(9);	Weißt dem Motor einen Anschluss zu. Dabei muss es sich um einen PWM-Anschluss handeln. In unserem Fall kommt dafür Pin 9, also PD5 zum Einsatz.
name.write(int Winkel);	motor1.write(90);	Lässt den Servo die angegebene Position anfahren. Diese darf zwischen 0 und 180 Grad liegen.
Befehle Datenübertragung per USB		
Syntax	Beispiel	Erklärung
Serial.begin(long Baudrate);	Serial.begin(9600);	Gibt an, wie viele Bits pro Sekunde übermittelt werden. Zur Kommunikation über USB sind 9600 Bits/s Standard.
Serial.println(String Nachricht);	Serial.println("Hallo Welt!");	Sendet eine Nachricht mit anschließendem Zeilenumbruch.
Serial.print(String Nachricht);	Serial.print("Hallo...");	Sendet eine Nachricht.

II. Beispiel für Quelltext des fertigen Programms

Bei dem folgenden Programmcode handelt es sich um ein Beispiel für ein vollständiges Programm, das alle Anforderungen vereint, aber nur ein Beispiel für viele Lösungsansätze ist:

```
/*

+++++++++++++++++++Solarpanelnachfuehrung++++++++++++++++++++++++++++++++++
+
 Diese Software dient zur Steuerung einer Solarpanelnachfuehrung ueber
einen Arduino Nano (ATmega328).
 Dafuer wird das Verhaeltnis des Spannungsabfalls ueber zwei analogen
Lichtsensoren (LDRs) ermittelt.
 Ist dieses 1:1, dann behaelt die Solarzelle ihre Position bei und laedt
einen Akku.
 Die umgewandelte Solarenergie soll hier ebenfalls bestimmt werden.
 Ist das Verhaeltnis der Lichtsensoren hingegen nicht gleich, soll ein
Servomotor die Photovoltaikzelle
 in Richtung des Sensors bewegen, der einen kleineren Spannungsabfall hat,
auf dessen Seite es also heller ist.
 So soll die Photovoltaikzelle immer nach der Sonne ausgerichtet werden.

 Created by Lars Christian Gleichmann
 For Phoenix Contact GmbH and Co. KG
 Professional Education (PED)

 14.08.2017
 */

// Variablen zur Speicherung der gemessenen Spannungen beider Analog-
Digital-Wandler-Werte deklarieren:
int valLDRadc = 0;
int valPadc = 0;
double mW = 0;

//Taster anlegen:
const int button1 = 8;     // Taster 1 ist an Pin 12 angeschlossen
const int button2 =  7;    // Taster 2 ist an Pin 11 angeschlossen

int r = 11;
int g = 9; // the PWM pin the LED is attached to
int b = 10;

int relayPin = 16;

//Variablen zum Speichern des Buttonzustands anlegen:
int buttonState1 = 0;     //Variablen zum Einlesen des aktuellen Zustandes
int buttonState2 = 0;
int buttonPressed1 = 0;   //Variablen zum Merken, dass ein Knopf gedrueckt
wurde
int buttonPressed2 = 0;

// Da nicht alle 10 Millisekunden der Bildschirm umgestellt und ueber USB
eine Nachricht an einen Computer gesendet werden soll (um Fehler zu
vermeiden und ein Flackern zu verhindern).
int timestamp = 0;

// Biblotheken fuer den Bildschirm einbinden ("Wire.h" importiert alle
Funktionen des I2C-Bus):
#include <Wire.h>
```

```
#include <LiquidCrystal_I2C.h>

// Neues Bildschirmobjekt mit I2C-Bus-Adresse (0x3F), Spalten (16) und
Zeilen (2) des LCD-Displays anlegen:
LiquidCrystal_I2C lcd(0x3F, 16, 2);

// Bibliothek zur Ansteuerung von Servomotoren laden.
#include <Servo.h>
// Einen Servomotor als Objekt mit vollem Funktionsumfang anlegen:
Servo motor;

// Variable fuer Motorposition deklarieren.
// Der Anfangswert entspricht 90, was einer geraden Ausrichtung von 0 Grad
des Solarpanels entspricht.
int mpos = 90;

// Initialisierung - Diese Funktion wird direkt beim Start des Controllers
ausgefuehrt:
void setup() {

  // lcd.init(); // Achtung: Bei Fehlermeldung ausdokumentieren!
  // lcd.begin(16,2); // Achtung: Bei Fehlermeldung ausdokumentieren!

  // LEDs als Ausgaenge festlegen:
  pinMode(r, OUTPUT);
  pinMode(b, OUTPUT);
  pinMode(g, OUTPUT);

  // LED in Betriebsmodus schalten:
  analogWrite(r, 0);
  analogWrite(b, 0);
  analogWrite(g, 255);

  // Relais als Ausgang festlegen:
  pinMode(relayPin, OUTPUT);

  //Knoepfe als Eingaenge definieren:
  pinMode(button1, INPUT);
  pinMode(button2, INPUT);

  // Verbindung zum LCD aufnehmen:
  lcd.begin();
  // Hintergrundbeleuchtung des LCD anschalten:
  lcd.backlight();
  // Maske anlegen, in der spaeter die "_" gegen Werte ausgetauscht werden.
  // Erste Zeile schreiben:
  lcd.print("Winkel: ___ Grad");
  // Cursor in zweite Zeile bewegen (Informatiker zaehlen immer ab 0!!!)
  lcd.setCursor(0,1);
  // Zweite Zeile schreiben:
  lcd.print("L:_.__V  S:_.__V");

  // USB-Verbindung einrichten (zu Fehlerbehebungszwecken bei defekten
Bildschirm):
  Serial.begin(9600);

  // Die Position des Servomotors soll ueber Pulsweitenmodulation (PWM)
gesteuert werden.
  // Das PWM-Signal wird durch folgenden Befehl am Pin D9 des Arduinos
erzeugt:
  motor.attach(5);
```

```
  // Der Befehl "write(int WINKEL)" stellt das PWM-Signal so ein, dass der
Motor auf die gewuenschte Position faehrt.
  // Hier wird der Motor zunaechst zu eingestellt, dass das Solarpanel
gerade ausgerichtet ist:
  motor.write(mpos);
  //delay(10000);
}

// Die Funktion "loop" wird in einer Schleife durch den Arduino so lange
ausgefuehrt, bis dieser umprogrammiert oder vom Strom getrennt wird:
void loop() {

    //Knopfzustand abfragen:
    buttonState1 = digitalRead(button1);
    buttonState2 = digitalRead(button2);

    //Merken, wenn der Knopf gedrueckt wurde:
    if (buttonState1 == 1) {
      buttonPressed1 = 1;
    }
    if (buttonState2 == 1) {
      buttonPressed2 = 1;
    }

    // Der Befehl "analogRead(PIN)" liest die an einem der Analog-Digital-
Wandler anliegende Spannung ein.
    // Am Pin A6 wird die Spannung ueber einen der LDRs gemessen:
    valLDRadc = analogRead(A1);

    // Der Spannungsteiler aus beiden Lichtsensoren (LDRs) wird auf eine
Spannungsverschiebung geprueft:
    if (valLDRadc < 494) {
      mpos = mpos + 1;
      // Damit der Motor nicht ueberdreht, darf der Winkel nicht groesser
als 160 Grad betragen.
      if (mpos >= 171) {
        mpos =171;
      }
      motor.write(mpos);
    } else if (valLDRadc > 529) {
      mpos = mpos - 1;
      // Der Winkel darf auch nicht kleiner als 50 Grad sein, damit der
Motor nicht durchdreht.
      if (mpos <= 90) {
          mpos = 90;
        }
      motor.write(mpos);
    }

    // Wenn der Timestamp sich glatt durch 1000 (X modulo 1000; Rest von x
durch 1000) teilen laesst, ist eine Sekunde vergangen.
    if (timestamp % 1000 == 0) {
      // Die selbstgeschriebene Funktion "report()" wird aufgerufen, um die
Anzeige auf den neusten Stand zu bringen.
      report();
      // Der Timestamp wird auf 0 zurueckgesetzt, damit es zu keinen
Ueberlauf des Speichers (8 Bit) kommt.
      timestamp = 0;

    }

   Serial.println(timestamp);
```

```
    // Die erneute Ausfuehrung der Schleife wird um 10 Millisekunden
verzoegert, damit der Analog-Digitalwandler sauber arbeitet.
    delay(10);
    // Der Timestamp (Millisekunden seit letztem Bericht an den Bildschirm)
wird dementsprechend erhoeht:
    timestamp = timestamp + 10;

}

void report() {
    lcd.clear();
     Serial.println(4);
    lcd.setCursor(2,1);
     Serial.println(5);
    lcd.print(String(adcToVolt(valLDRadc), DEC));
     Serial.println(6);
    lcd.setCursor(11,1);
     Serial.println(7);
    lcd.print(String(adcToVolt(valPadc), DEC));
    Serial.println(8);
    //Melden, wenn Knopf gedrueckt wurde:
    /*if (buttonPressed1 == 1) {
      pause();
      buttonPressed1 = 0;
      buttonState1 = 0;
    }*/

    /*if (buttonPressed2 == 1) {
      if (digitalRead(relayPin) == 1) {
        digitalWrite(relayPin, LOW);
      } else {
        digitalWrite(relayPin, HIGH);
      }
      buttonPressed2 = 0;
      buttonState2 = 0;
    }*/
    Serial.println(1);

    // Die Spannung, die an der Photovoltaikzelle anliegt, wird am Pin A7
gemessen:
    valPadc = analogRead(A0);
    Serial.println(2);
    // Aktuelle Messwerte zu Fehlerbehebungszwecken per USB und UART an
einen anderen Computer senden:
    Serial.println("Spannung MW: " + String(adcToVolt(valPadc), DEC) +
"V");
    mW = adcToVolt(valPadc);
    Serial.println("--> Leistung: " + String(mW, DEC) + "V");
    Serial.println("Spannung LDR2: " + String(adcToVolt(valLDRadc), DEC) +
"V");

    // Die Maske des LCD-Bildschirmes fuellen (siehe oben):

    lcd.setCursor(0,0);

    lcd.print(mpos - 90);
    Serial.println(3);
    // Abfrage zur optisch richtigen Positionierung des Schriftzuges
"Grad".
   /* if (mpos - 90 <= -10) {
```

```
      Serial.println(31);
      lcd.print(" Grad");
    } else if (mpos - 90 <= -1) {
       Serial.println(32);
      lcd.print("  Grad");
    } else if (mpos -90 <= 9) {
       Serial.println(33);
      lcd.print("   Grad");
    } else {
       Serial.println(34);
      lcd.print("  Grad");
    }*/


}

void nullpos() {
  // Die Maske des LCD-Bildschirmes fuellen (siehe oben):
  lcd.setCursor(8,0);
  lcd.print(mpos - 90);
  lcd.setCursor(9,0);
  lcd.print("  ");

  analogWrite(r, 0);
  analogWrite(b, 255);
  analogWrite(g, 0);

  mpos = 90;
  motor.write(mpos);

  for (int i = 10; i > 0; i--) {
    lcd.setCursor(0,1);
    lcd.print("Noch " + String(i) + "               ");
    delay(1000);
  }
  buttonPressed2 = 0;
  analogWrite(r, 0);
  analogWrite(b, 0);
  analogWrite(g, 255);
  // Maske anlegen, in der spaeter die "_" gegen Werte ausgetauscht werden.
  // Erste Zeile schreiben:
  lcd.print("Winkel: ___ Grad");
  // Cursor in zweite Zeile bewegen (Informatiker zaehlen immer ab 0!!!)
  lcd.setCursor(0,1);
  // Zweite Zeile schreiben:
  lcd.print("L:_.__V  S:_.__V");
}

void pause() {
  analogWrite(r, 250);
  analogWrite(b, 255);
  analogWrite(g, 0);
  buttonState1 = 0;
  buttonPressed1 = 0;
  lcd.clear();
  // Maske anlegen, in der spaeter die "_" gegen Werte ausgetauscht werden.
  // Erste Zeile schreiben:
  lcd.print("Pause!");
  // Cursor in zweite Zeile bewegen (Informatiker zaehlen immer ab 0!!!)
  lcd.setCursor(0,1);
  // Zweite Zeile schreiben:
  lcd.print("L:_.__V  S:___V ");
```

```
  while (digitalRead(button1) == 0) {
    delay(2);
  }

  lcd.setCursor(0,0);
  // Maske anlegen, in der spaeter die "_" gegen Werte ausgetauscht werden.
  // Erste Zeile schreiben:
  lcd.print("Winkel: ___ Grad");
  // Cursor in zweite Zeile bewegen (Informatiker zaehlen immer ab 0!!!)
  buttonState1 = 0;
  buttonPressed1 = 0;

  analogWrite(r, 0);
  analogWrite(b, 0);
  analogWrite(g, 255);
  loop();
}

// Diese Funktion wandelt den Messwert des Analog-Digital-Wandlers, der
zwischen 0 und 1023 liegt in einen absoluten Spannungswert in Volt um:
float adcToVolt(int value) {
  // (Die Zahl entspricht der Spannung von 5 Volt durch 1023)
  return 0.00488758553274 * value;
}
```