Smart Solar Panel

Niklas Ackermann

Steffen Kraft

Jonas Scherz

Philipp Zoll

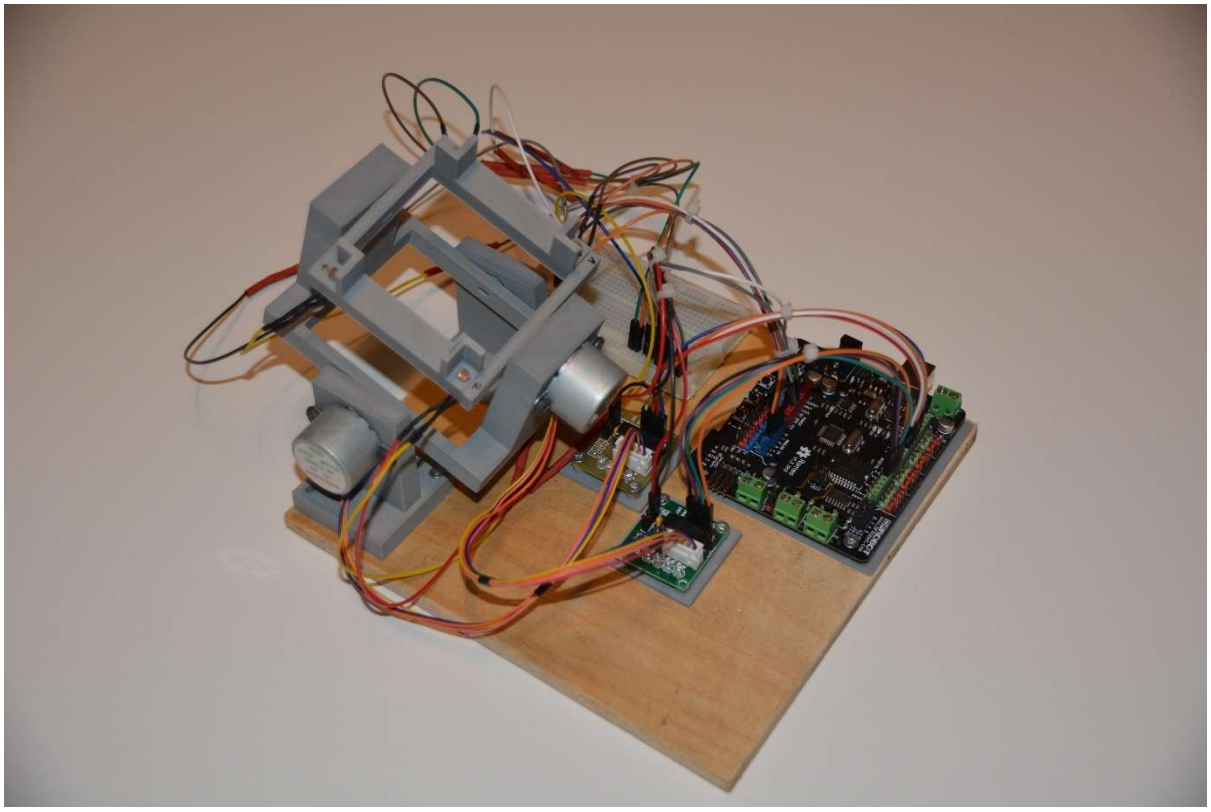


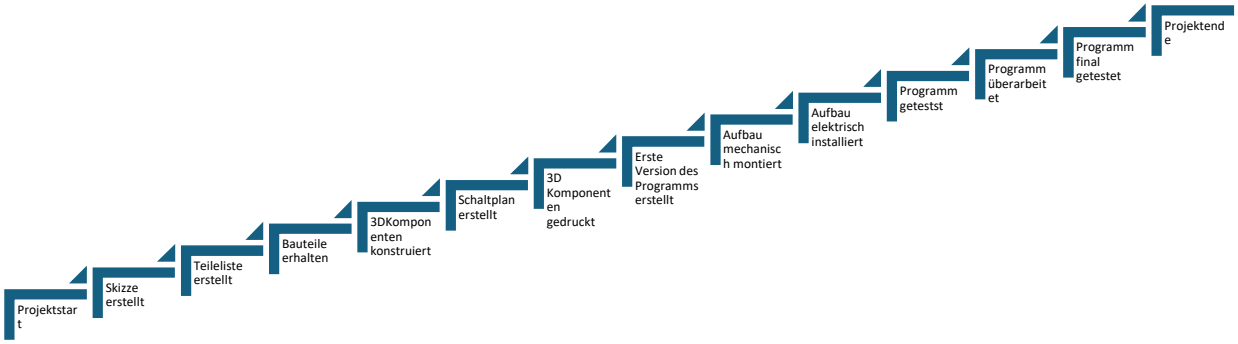
Abbildung 0: Modell

Inhalt

1. Projektziel

In der Vorlesung “Signalverarbeitung 2” wurde die Aufgabe gestellt, ein Projekt im Bereich Embedded Systems zu planen, umzusetzen und anschließend zu dokumentieren. Anfangs hatten wir eine kleine Ideenfindungsphase und haben uns dann schlussendlich dazu entschieden, eine smarte Solarplatte zu bauen. Solarenergie rückt immer mehr und mehr in den Vordergrund als alternative, grüne Energiequelle. Herkömmliche Installationen sind in einem Festen Winkel, auf einer festen Position auf beispielweise einem Hausdach installiert. Hier wird der maximale Strom immer nur zu bestimmen Tageszeiten, wenn der Winkel der Sonne perfekt passt, generiert. Hier setzt unser Projekt an. Wir möchten eine Konstruktion für eine Solarplatte bauen, welche sich automatisch perfekt zur Sonne ausrichtet und somit die zu dem Tageszeitpunkt maximal mögliche Energie liefert.

1.1 Zeitlicher Ablauf



2.Aufbau

Nachdem wir das Projektthema festgelegt hatten, haben wir uns Gedanken zu den benötigten Teilen gemacht. Hierzu haben wir folgende Materialien Liste festgelegt:

Beschreibung	**Anzahl**	**Link**
Arduino Uno	1	ARDUINO UNO: Arduino Uno Rev. 3, SMD-Variante, ATmega328, USB bei reichelt elektronik
Arduino Kabelsatz	1	ELEGOO Jumper Wire 40x 20cm, Male-Female, Kabel Steckbrücken 28AWG Draht-brücken für Arduino: Ama-zon.de: Gewerbe, Industrie & Wissenschaft
Arduino Steckboard	1	STECKBOARD 1K2V: Experi-mentier-Steckboard 640 - 200 Kontakte bei reichelt elektronik
Fotowiderstände	4	Otronic GL5516 LDR Lichtemp-findlicher Widerstand (Fotowi-derstand) - OTRONIC
Widerstände	4	1 - 4W 10K: Widerstand, Kohle-schicht, 10 kOhm, 0207, 250 mW, 5% bei reichelt elektronik
Servo-Motoren	2	SG90 9g Micro Servomotor | Roboter-Bausatz.de
Solarpanel mit Batterie	1	SOL-EXP 29100: Solar-Leucht-modul "LongLife" bei reichelt elektronik
3D-Druckmaterial* (Pla)	-	-

Während unseres Projekts fehlten einige der in der ursprünglichen Materialliste aufgeführten Komponenten, insbesondere das Solarmodul mit Batterie. Eine Überlegung, ein autarkes System zu entwickeln, erübrigte sich dadurch, und wir mussten auf eine kabelgebundene Stromversorgung ausweichen.

Zudem erhielten wir statt der angeforderten Servomotoren Schrittmotoren. Diese erforderten eine andere Ansteuerung und Anpassungen in Programmierung, was zu Verzögerungen führte.

Da die gerade genannten Komponenten fehlten, ist die ursprüngliche Materialliste nicht mehr aktuell.
*Siehe Fazit

2.1 Mechanischer Aufbau Solarplattenhalter

Während der Zusammenstellung unserer Teileliste haben wir uns auch Gedanken über das mechanische Modell gemacht, auf dem die Solarplatte platziert werden soll. Die Anforderung an das Modell war, dass die Solarplatte bis zu einem bestimmten Neigungswinkel in alle Richtungen geschwenkt werden kann. Dafür haben wir ein Modell mit zwei Drehachsen konzipiert, über die die Solarplatte sowohl vertikal als auch horizontal ausgerichtet werden kann. Außerdem wollten wir sicherstellen, dass bereits geringe Winkeländerungen der Lichtquelle dazu führen, dass sich die Konstruktion automatisch neu ausrichtet.

2.1.1 Zeichnungen

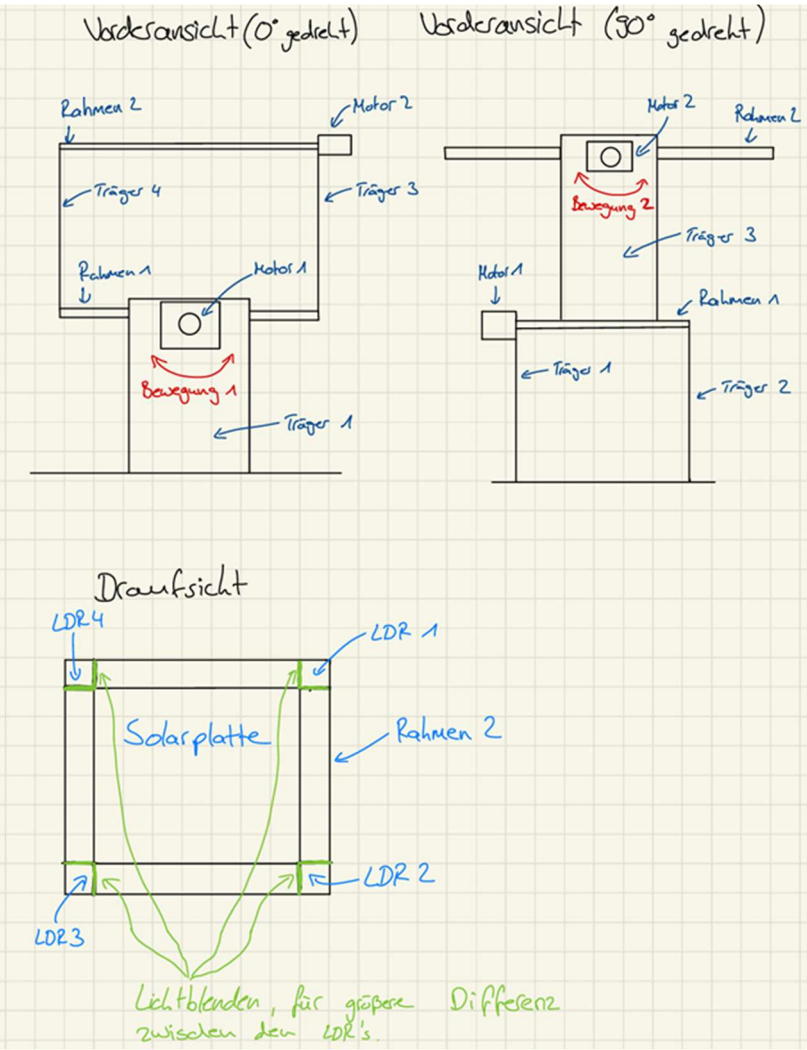


Abbildung 1: Skizze mechanisches Modell

Mit der obenstehenden Skizze haben wir die einzelnen Komponenten des Systems konstruiert. Die Konstruktion besteht aus einem Fundament, vier Trägern, zwei Rahmen, zwei Motoren, vier Lichtblenden und einer Solarplatte. Die Träger 1 und Träger 2 tragen den Rahmen 1, der mithilfe von Motor 1 positioniert und gedreht werden kann.

Auf dem Rahmen 1 sind zwei weitere Träger (Träger 3 und Träger 4) befestigt, die um jeweils 90 Grad zu Träger 1 und Träger 2 versetzt angeordnet sind. Diese Träger halten den Rahmen 2, der durch Motor 2 in alle gewünschten Richtungen positioniert werden kann. Diese Anordnung ermöglicht eine präzise zweiachsige Ausrichtung des Solarpanels.

Auf Rahmen 2 befinden sich Lichtblenden, die dafür sorgen, dass größere Unterschiede zwischen den Widerstandswerten der LDR-Sensoren gemessen werden können, wenn sich die Position der Lichtquelle ändert.

Der Entwicklungsprozess für das mechanische Konstrukt folgte einem iterativen Modell, das sich als äußerst sinnvoll erwiesen hat. Zu Beginn jeder Iteration haben wir eine klare Aufgabenstellung definiert, die anschließend von jedem Teammitglied während der Bearbeitungsphase individuell ausgearbeitet wurde. Nach Ablauf der Bearbeitungszeit haben wir die Ergebnisse gemeinsam diskutiert, analysiert und verglichen. Auf dieser Grundlage haben wir eine neue Aufgabenstellung entwickelt, die auf den bisherigen Erkenntnissen basierte.

Diesen iterativen Prozess haben wir so lange wiederholt, bis wir einstimmig zu dem Entschluss kamen, dass die erarbeitete Skizze unsere Projektidee vollständig und präzise widerspiegelt.

2.1.2 Konstruktion der einzelnen Komponenten

Die Konstruktion der einzelnen Komponenten unseres Projekts erfolgte mit der Software „Shapr3D“. Die Einarbeitung in diese Software erforderte eine längere Phase der Selbstschulung, da einige Teammitglieder bereits längere Zeit nicht mehr mit der Software gearbeitet hatten, während andere bislang keinerlei Erfahrung damit hatten.

2.1.2.1 Konstruktion Träger

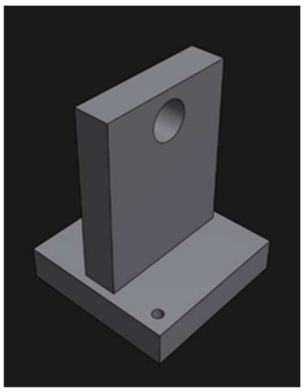


Abbildung 2: Träger unterer Rahmen

Abbildung 3: Träger unterer Rahmen

Die beiden obenstehenden Bilder zeigen den Träger. Vertikal nach unten sind zwei Bohrungen sichtbar, die dazu dienen, den gesamten Aufbau mithilfe von Schrauben oder anderen Verbindungselementen am Fundament zu befestigen. Zusätzlich ist eine größere, horizontal gebohrte Öffnung erkennbar. Diese dient dazu, einen Bolzen, der am Rahmen angebracht ist, durchzuschieben, um eine stabile Verbindung zwischen Träger und Rahmen herzustellen.

Der Motor wird direkt am Träger montiert, wobei der Flansch des Motors durch die Öffnung im Träger zeigt.

2.1.2.2 Konstruktion Rahmen mit Träger

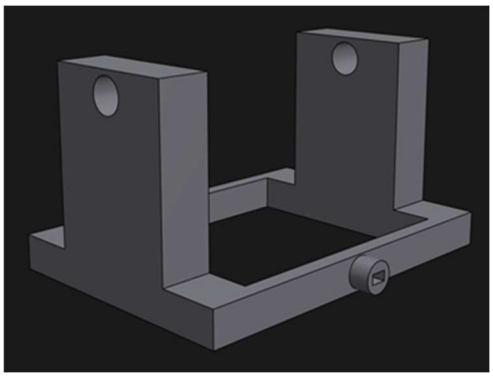


Abbildung 4: Rahmen mit Träger

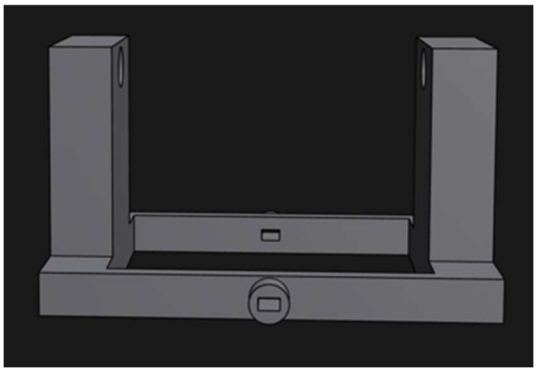


Abbildung 5: Rahmen mit Träger

Die beiden obenstehenden Bilder zeigen den Rahmen mit Träger. Dieser dient als Bindeglied zwischen den Trägern am Fundament und dem oberen Rahmen, auf dem später das Solarpanel montiert wird.

Der Rahmen mit Träger verfügt auf beiden Seiten über Bolzen mit einer Mulde. Wie bereits bei der Konstruktion der Träger beschrieben, werden diese Bolzen in die entsprechenden Löcher der Träger eingeführt, um eine stabile Verbindung zwischen Träger und Rahmen herzustellen. Zusätzlich wird der Flansch des Motors in die Mulde des Bolzens eingeführt, wodurch der Rahmen gedreht und positioniert werden kann.

Auf den Bildern ist zudem zu erkennen, dass sich auf dem Rahmen ausschließlich die Träger befinden, die für die Verbindung und Befestigung eines weiteren Rahmens vorgesehen sind.

2.1.2.3 Konstruktion Rahmen mit Lichtblenden

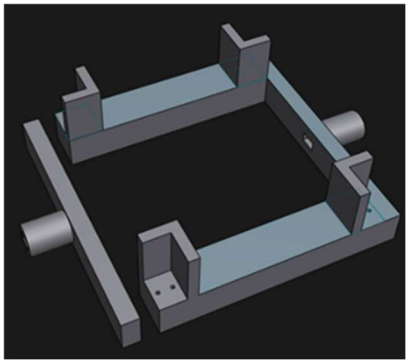


Abbildung 6: Rahmen mit Lichtblenden

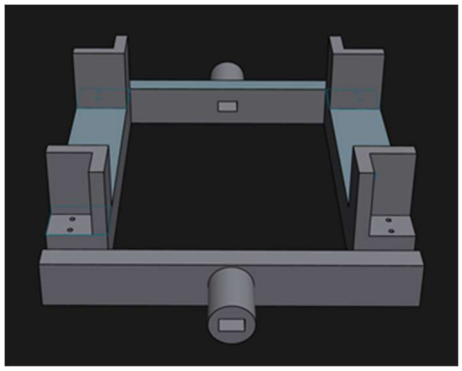


Abbildung 7: Rahmen mit Lichtblenden

Die beiden obenstehenden Bilder zeigen den Rahmen, auf dem später das Solarpanel montiert wird. Die Bolzen mit den Mulden an diesem Rahmen (Rahmen mit Lichtblenden) erfüllen die gleiche Funktion wie die Bolzen am mittleren Rahmen (Rahmen mit Träger).

Dieser Rahmen verfügt über keine Träger, ist jedoch mit Lichtblenden ausgestattet. In den Ecken des Rahmens befinden sich Bohrungen, die dazu dienen, die Kontakte der LDR-Sensoren durchzuführen. Der Rahmen besteht aus zwei Teilen, um die Montage zu erleichtern und den Aufbau effizienter zu gestalten. Hier kann man auch sehr gut die Lichtblenden erkennen, die nach außen hin geöffnet sind, weg von der Solarplatte bzw. der Mitte.

2.1.3 3D Druck Vorgang

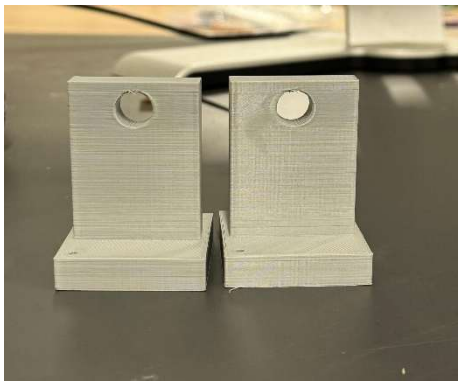


Abbildung 8: Gedruckte Träger

Die 3D-Modelle haben wir eigenständig mit unserem privaten 3D-Drucker gefertigt. Da wir bereits über Vorkenntnisse im 3D-Druck verfügten, war keine zusätzliche „Selbstschulungszeit“ erforderlich. Aufgrund der längeren Standzeit des Druckers war jedoch ein Workaround zur Justierung notwendig, bevor wir mit dem Druck beginnen konnten. Die Teile wurden anschließend erfolgreich gedruckt, auch wenn das leicht feuchte PLA zu einer nicht ganz perfekten Druckqualität führte.

2.2 Elektronischer Aufbau

Zu Beginn standen uns nur die LDRs, verschiedene Widerstände und der Arduino zur Verfügung. Daher haben wir uns zunächst mit diesen Komponenten auseinandergesetzt. Wir haben die Funktionsweise der LDRs im Detail untersucht: Ihr Widerstand sinkt, je mehr Licht auf sie einwirkt. Auf Grundlage dieses Wissens haben wir uns mit einem Spannungsteiler beschäftigt, der notwendig ist, um die Widerstandsänderung der LDRs bei Helligkeitsänderungen mit dem Arduino erfassen und verarbeiten zu können.

Das Prinzip funktioniert so: Die Spannung an einem festen Widerstand, der in Reihe mit dem LDR geschaltet ist, wird über einen Analogeingang des Arduino gemessen. Sinkt der Widerstand des LDRs, erhöht sich die Spannung am festen Widerstand. Dadurch können wir feststellen, welcher LDR mehr Licht misst.

Um den optimalen Wert für den festen Widerstand im Spannungsteiler zu ermitteln, haben wir eine Testschaltung aufgebaut und verschiedene Widerstandswerte getestet. Ziel war es, einen möglichst großen Messbereich (0 bis 5 V) am Analogeingang des Arduino abzudecken. Die besten Ergebnisse erzielten wir mit einem 10-kOhm-Widerstand.

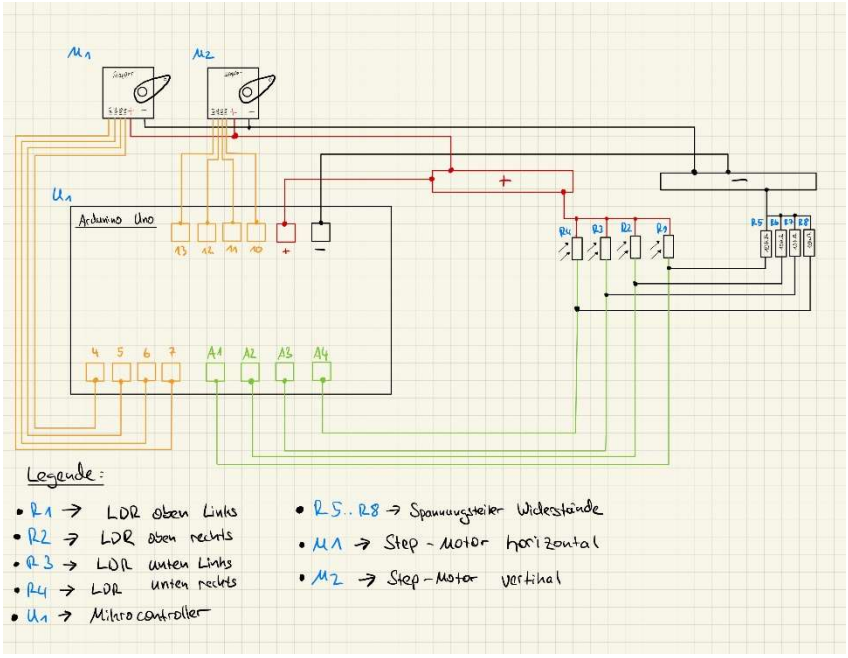


Abbildung 9: Schaltplan

Nachdem die Motoren verfügbar waren, hatten wir alle notwendigen Bauteile für den Schaltplan zusammen. Die Solarplatte wurde nicht in die Schaltung integriert, da der Fokus des Projekts auf der Bewegungsabfolge lag und uns keine Solarplatte zur Verfügung gestellt wurde. Auf dieser Grundlage haben wir den Schaltplan entwickelt. Die Informationen zum Anschließen der Motoren und anderer Komponenten haben wir den entsprechenden Datenblättern entnommen.

In diesem haben wir dann auch die später benötigte Pinbelegung für das Programm festgelegt.

Bauteilkennzeichnung	**Pin**
R1	A1
R2	A2
R3	A3
R5	A4
M1+	+
M1-	-
M1.IN1	7
M1.IN2	6
M1.IN3	5
M1.IN4	4
M2+	+
M2-	-
M2.IN1	13
M2.IN2	12
M2.IN3	11
M2.IN4	10

2.3 Zusammenbau des Prototyps

Für den Zusammenbau des Prototyps haben wir uns in der Werkstatt eines Gruppenmitglieds getroffen. Dort haben wir zunächst eine Grundplatte ausgemessen und zugesägt, auf der der gesamte Aufbau montiert werden sollte. Beim Zusammenbau sind wir schrittweise vorgegangen: Zunächst wurde die mechanische Konstruktion zusammengesteckt und teilweise verklebt, anschließend wurden die Motoren daran befestigt.

Im nächsten Schritt begann die Verdrahtung, bei der auch die LDRs angelötet wurden. Zum Schluss haben wir vor Ort das Programm erstmalig aufgespielt und getestet. Dabei fiel uns auf, dass einige Pins im Programm vertauscht waren und nicht mit dem Schaltplan übereinstimmten. Nach der Korrektur dieses Fehlers konnten wir weitere Tests durchführen.

Während der Tests stellten wir jedoch fest, dass im Programm ein grundlegender Fehler vorlag, den wir später in einem separaten Arbeitsschritt behoben haben.

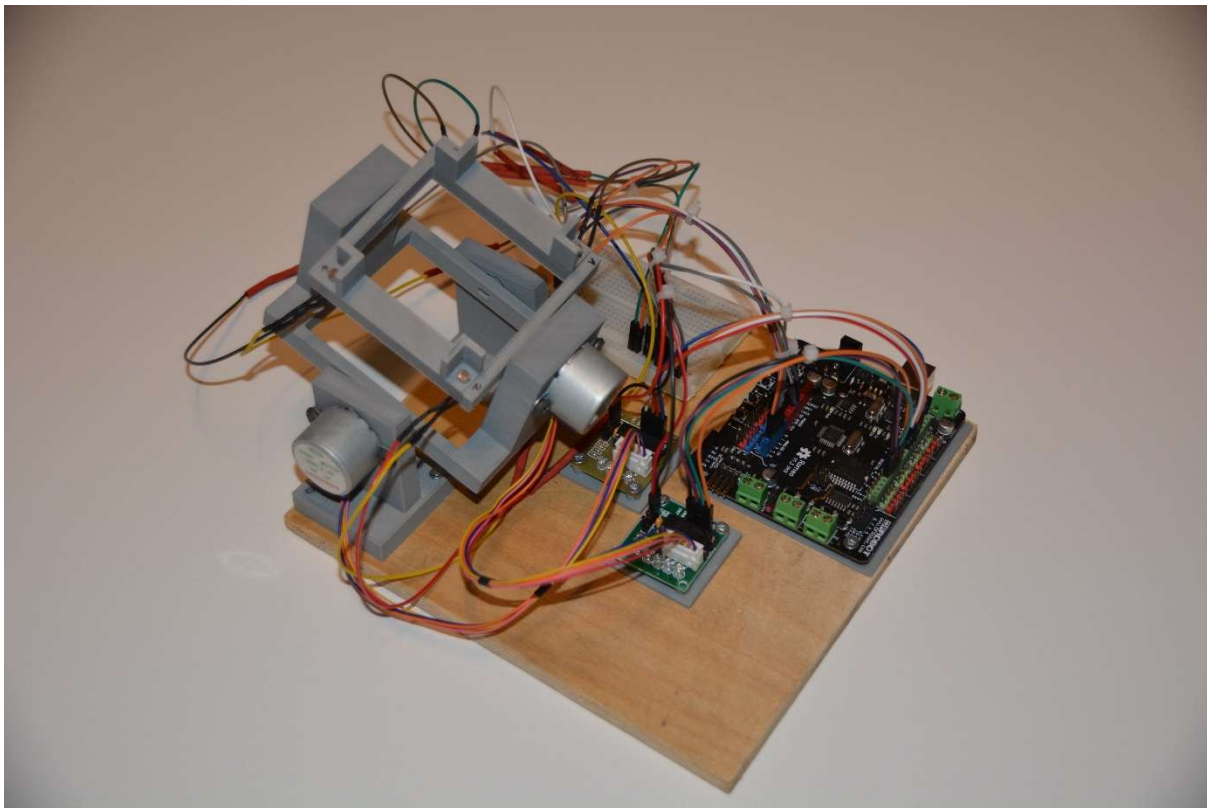


Abbildung 10: Fertiger Prototyp

3. Programmierung

Bei der Programmierung haben wir, ähnlich wie bei den mechanischen und elektrischen Tätigkeiten, einen iterativen Prozess angewandt. Dadurch konnten wir uns frühzeitig mit den einzelnen Komponenten unseres Modells auseinandersetzen. So haben wir beispielsweise zunächst ein kleines Programm zum Auslesen und Vergleichen der LDRs entwickelt. Ebenso haben wir ein Beispielskript für die Stepper-Motoren erstellt, um ihr Verhalten zu untersuchen. Aus diesen Experimenten ist schließlich unsere finale Implementierung entstanden.

3.1 Abstraktion des Programmes

Ablaufdiagramm unseres Programms:

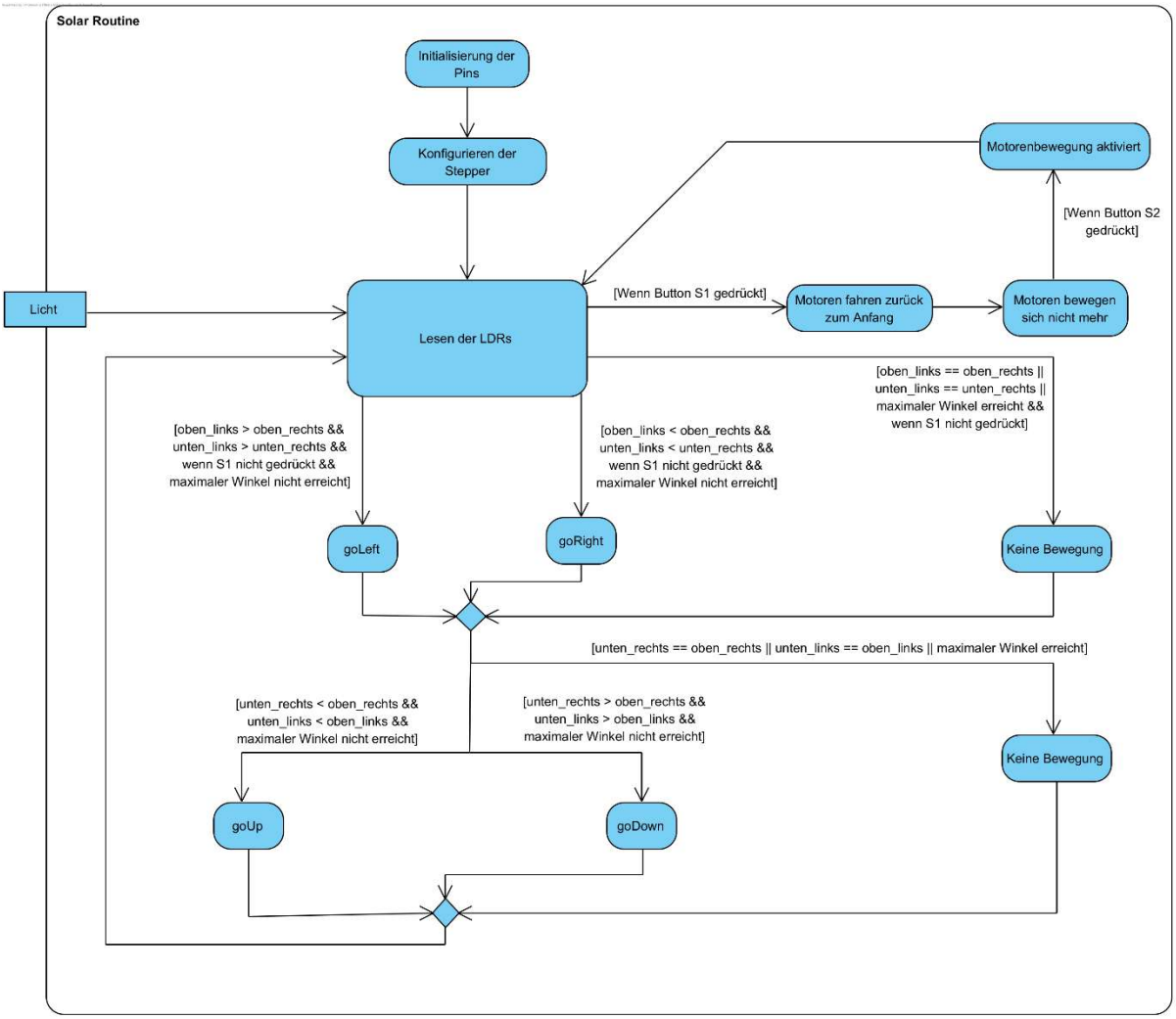


Abbildung 11: Ablaufdiagramm

Das Ablaufdiagramm kann wie folgt zusammengefasst werden:

1. **Initialisierung**: Zu Beginn werden die Pins initialisiert und die Stepper konfiguriert.
2. **Hauptlogik**: Das Programm überprüft kontinuierlich die Eingabewerte an den LDRs. Es werden dann Entscheidungen basierend auf diesen Werten getroffen.
3. **Bedingte Aktionen**: Abhängig von den ausgewerteten Bedingungen wird das Gestell in verschiedene Richtungen bewegt (z. B. links, rechts, oben, unten) oder bleibt inaktiv, falls keine Aktion erforderlich ist.
4. **Benutzerinteraktion**: Buttons können das Verhalten des Programms beeinflussen, indem sie bestimmte Funktionen auslösen, wie z. B. das Zurücksetzen auf den Anfangszustand oder das Aktivieren der Motorbewegung.
5. **Endzustände**: Der Hauptteil des Programmes ist in einer sich ständig widerholenden Schleife. Aber ein annähender Endzustand kann erreicht werden, z. B. wenn kein weiterer Bewegungsbedarf besteht.

Das Diagramm zeigt die logische Struktur und Aktionen des Programms.

3.2 Implementierung

```
#include <AccelStepper.h>

// 1
int oben_links = 0;
// 2
int oben_rechts = 0;
// 3
int unten_links = 0;
// 4
int unten_rechts = 0;

int ldr_oben_links = A1;
int ldr_oben_rechts = A2;
int ldr_unten_links = A3;
int ldr_unten_rechts = A4;
// Infos und Variablen für die Stepper
const byte Fullstep = 4;
const byte Halfstep = 8;
const short fullRevolution = 2038;
const float SteppDegree = 11.32;  // Halfstep 11.32 - Fullstep 5.66
//----------------------Pins IN1-IN3-IN2-IN4
AccelStepper stepperH(Halfstep, 7, 5, 6, 4);
AccelStepper stepperV(Halfstep, 13, 11, 12, 10);

float maxDegree = 45; //Maximale Auslenkung
long maxMoveRev = maxDegree * SteppDegree; //Maximale Auslenkung in Stepps
bool automatic = true;

const int NUM_KEYS = 5;
int adc_key_val[NUM_KEYS] = {
  30, 150, 360, 535, 760
};
int adc_key_in;
int key = -1;
int oldkey = -1;
```

Abbildung 12: Initialisierung

Zu Beginn des Programms initialisieren und deklarieren wir die Variablen, die im weiteren Verlauf des Programms verwendet werden.

- **oben_links**, **oben_rechts**, **unten_rechts** und **unten_links** sind die Variablen, denen die aktuell ausgelesenen Werte der LDRs zugewiesen werden.
- Die gleichen Werte mit dem Präfix **ldr_** davor geben die Pins an, an denen die LDRs angeschlossen sind.
- **Fullstep** und **Halfstep** sind Konstanten, die die Betriebsart des Schrittmotors definieren. **Fullstep** bedeutet, dass der Motor bei jeder Sequenz einen vollständigen Schritt

ausführt, während **Halfstep** kleinere Schritte ermöglicht, die eine genauere Bewegung erlauben.

- **MaxDegree** bestimmt die maximale Auslenkung unserer Konstruktion in Grad.
- **MaxMoveRev** legt die maximale Auslenkung in Schritten (Steps) fest.
- **Automatic** wird verwendet, um bei einem Nullpunkt-Fahren alle anderen Bewegungen zu stoppen, indem es auf **false** gesetzt wird.
- **NUM_KEY** definiert, wie viele Tasten an unserem Arduino angeschlossen sind.
- **adc_key_val** gibt an, welchen Wert jeder Taster hat.
- **adc_key_in** enthält den aktuell ausgelesenen Wert eines Tasters.
- **Key** ist der Wert des aktuell ausgelesenen Tasters.
- **OldKey** speichert den zuletzt ausgelesenen Wert eines Tasters.

```
void setup() {

  pinMode(ldr_oben_links, INPUT);
  pinMode(ldr_oben_rechts, INPUT);
  pinMode(ldr_unten_links, INPUT);
  pinMode(ldr_unten_rechts, INPUT);

  stepperH.setMaxSpeed(1000.0);     // set the maximum speed
  stepperH.setAcceleration(3000);    // set acceleration
  stepperH.setSpeed(600);           // set initial speed
  stepperH.setCurrentPosition(0);   // set position

  stepperV.setMaxSpeed(1000.0);     // set the maximum speed
  stepperV.setAcceleration(3000);    // set acceleration
  stepperV.setSpeed(600);           // set initial speed
  stepperV.setCurrentPosition(0);   // set position

  Serial.begin(9600);
}
```

Abbildung 13: Setup

In unserem Setup Initialisieren wir unsere Pins als Input Pins, die dann die Spannung der LDRS als Input bekommen. Anschließend konfigurieren wir die Bewegungsgeschwindigkeit und den Nullpunkt der beiden Schrittmotoren.

Der Nullpunkt wird zu diesem Zeitpunkt gesetzt, da wir davon ausgehen, dass das Gestell sich in einer neutralen Position befindet und nicht ausgelenkt ist. Dieser Nullpunkt dient später als Referenz, um die aktuelle Auslenkung zu berechnen und sicherzustellen, dass die maximale Auslenkung nicht überschritten wird.

```
void loop() {
  readAndMapLdrs();

  Serial.print("LDR Wert oben_links: ");
  Serial.println(oben_links);  // debug value
  Serial.print("LDR Wert oben_rechts: ");
  Serial.println(oben_rechts);  // debug value
  Serial.print("LDR Wert unten_links: ");
  Serial.println(unten_links);  // debug value
  Serial.print("LDR Wert unten_rechts: ");
  Serial.println(unten_rechts);  // debug value


  buttonRoutine();

  delay(50);
  if (automatic) {
    // When no movment happens, wair for the specified delay
    if (!ldrRoutine()) {
      delay(1000);
    }
  }
}
```

Abbildung 14: Loop

Dies ist die Kernschleife, die kontinuierlich durchlaufen wird, solange der Arduino läuft.

- Zunächst wird die Methode **readAndMapLdrs()** aufgerufen, um die Spannungswerte der LDRs auszulesen und auf nutzbare Werte zu mappen.
- Anschließend werden die gelesenen Werte zur Überprüfung für Testzwecke ausgegeben.
- Danach wird **buttonRoutine()** aufgerufen, um zu überprüfen, ob einer der Aktionsknöpfe gedrückt wurde. Wenn **automatic** nicht **false** ist, wird anschließend **ldrRoutine()** ausgeführt.
- Die Methode **ldrRoutine()** gibt zurück, ob sich das Gestell bewegt hat oder nicht. Falls keine Bewegung stattgefunden hat, wird ein **Delay** von 1000 Millisekunden eingefügt. Dies spart Energie, da das Gestell bereits optimal zur Sonne ausgerichtet ist und sich für eine Weile nicht bewegen muss.

```
/*
 Lese die Werte der LDR's und map sie auf niedrigere Werte,
 dass kleine Unterschiede keine großen Auswirkungen haben
*/
void readAndMapLdrs() {
  oben_links = analogRead(ldr_oben_links);
  oben_rechts = analogRead(ldr_oben_rechts);
  unten_links = analogRead(ldr_unten_links);
  unten_rechts = analogRead(ldr_unten_rechts);

  oben_links = map(oben_links, 0, 1023, 0, 50);
  oben_rechts = map(oben_rechts, 0, 1023, 0, 50);
  unten_links = map(unten_links, 0, 1023, 0, 50);
  unten_rechts = map(unten_rechts, 0, 1023, 0, 50);
}
```

Abbildung 1: ReadAndMapLdrs

Die Methode **readAndMapLdrs** liest die Werte unserer LDRs aus und mappt diese auf einen Bereich, der sich besser für Vergleiche eignet.

```
// Routine um Funktionen via Buttons auszulösen.
void buttonRoutine() {
  adc_key_in = analogRead(7);  // read the value from the sensor
  /* get the key */
  key = get_key(adc_key_in);     // convert into key press
  if (key != oldkey) {           // if keypress is detected
    delay(50);                   // wait for debounce time
    adc_key_in = analogRead(7);  // read the value from the sensor
    key = get_key(adc_key_in);   // convert into key press
    if (key != oldkey) {
      oldkey = key;
      if (key >= 0) {
        switch (key) {
          case 0:
            Serial.println("Button S1 pressed");
            goToBeginning();
            Serial.println("Automatische Bewegung ausgeschaltet");
            break;
          case 1:
            Serial.println("Button S2 pressed");
            Serial.println("Automatische Bewegung eingeschaltet");
            automatic = true;
            delay(50);
            break;
          case 2:
            Serial.println("Button S3 pressed");
            break;
          case 3:
            Serial.println("Button S4 pressed");
            break;
          case 4:
            Serial.println("Button S5 pressed");
            break;
          default:
            Serial.println("Something went wrong");
        }
      }
    }
  }
}
```

Abbildung 16: ButtonRoutine

Die Methode **buttonRoutine** legt fest, welche Funktion den einzelnen Tastern zugewiesen ist, um eine manuelle Interaktion mit unserem Gestell zu ermöglichen. Auf dem Romeo V1.3-Board sind fünf integrierte Taster vorhanden, die alle mit dem analogen Pin 7 verbunden sind.

- Mit **S1** werden die Motoren auf ihren Nullpunkt zurückgefahren, und die automatische Bewegung wird deaktiviert.
- Mit **S2** wird die automatische Bewegung der Motoren wieder aktiviert.

Die Methode ist so gestaltet, dass das Gedrückthalten eines Tasters keinen Einfluss hat. Ein neuer Befehl kann erst ausgeführt werden, wenn der vorherige vollständig abgeschlossen ist

und der Taster erneut betätigt wird. Dies stellt sicher, dass jede Aktion nur einmal ausgelöst wird.

Da alle Taster mit demselben analogen Pin (Pin 7) verbunden sind, wird der jeweils gedrückte Taster anhand der unterschiedlichen analogen Werte identifiziert, die je nach Taster erzeugt werden.

```
// Convert ADC value to key number
int get_key(unsigned int input) {
  int k;
  for (k = 0; k < NUM_KEYS; k++) {
    if (input < adc_key_val[k]) {
      return k;
    }
  }
  if (k >= NUM_KEYS)
    k = -1;  // No valid key pressed
  return k;
```

Abbildung 17: Get_key

Die Methode **get_key** dient dazu, herauszufinden, welcher Taster gedrückt wurde. Dazu wird das zuvor deklarierte **adc_key_val**-Array verwendet, das die möglichen analogen Werte der Taster enthält.

Die Methode iteriert durch das Array und prüft, ob der übergebene Wert kleiner ist als der aktuelle Wert im Array. Sobald dies der Fall ist, wird die Anzahl der Iterationen zurückgegeben, um so den gedrückten Taster zu identifizieren.

Da die Werte der Taster im Array in aufsteigender Reihenfolge angeordnet sind, ermöglicht dieser Ansatz eine einfache und effiziente Zuordnung.

```
// Funktion um die Motoren auf ihren Nullpunkt zu fahren
void goToBeginning() {
  automatic = false;
  Serial.println("goToBeginning");

  stepperV.stop();
  delay(10);
  stepperH.stop();
  delay(10);

  stepperV.moveTo(0);
  stepperH.moveTo(0);

  while (stepperV.currentPosition() != stepperV.targetPosition()) {
    stepperV.run();

    delay(20);
  }
  while (stepperH.currentPosition() != stepperH.targetPosition()) {
    stepperH.run();

    delay(20);
  }
}
```

Abbildung 18: GoToBeginning

Die Methode **goToBeginning** wird durch das Drücken des Tasters **S1** ausgelöst. Sie sorgt dafür, dass beide Motoren nacheinander auf ihre Nullposition zurückfahren.

```
bool ldrRoutine() {
  bool moved = false;

  // Einfacher Vergleich auf dem oberen Gestell
  if (oben_links < oben_rechts && unten_links < unten_rechts) {
    moved = moveStepper(stepperH, 1, "goLeft");
  } else if (oben_links > oben_rechts && unten_links > unten_rechts) {
    moved = moveStepper(stepperH, -1, "goRight");
  } else {
    Serial.println("Keine horizontale Bewegung nötig");
  }

  // Einfacher Vergleich auf dem unteren Gestell
  if (unten_rechts < oben_rechts && unten_links < oben_links) {
    moved = moveStepper(stepperV, -1, "goUp");
  } else if (unten_rechts > oben_rechts && unten_links > oben_links) {
    moved = moveStepper(stepperV, 1, "goDown");
  } else {
    Serial.println("Keine vertikalen Bewegung nötig");
  }
  return moved;
}
```

Abbildung 19: LdrRoutine

Die Methode **ldrRoutine** verwendet die zuvor ausgelesenen Werte der LDRs (Lichtsensoren), um zu bestimmen, in welche Richtung sich die Motoren bewegen sollen. Basierend auf dieser Entscheidung werden die Motoren jeweils um einen Schritt (Step) in die entsprechende Richtung bewegt. Zusätzlich gibt die Methode zurück, ob eine Bewegung stattgefunden hat.

```
/**
*
* Move Method for steppers with Border check.
* @return true if movment was done
*/
bool moveStepper(AccelStepper& stepper, int move_steps, char* console_txt) {
  Serial.println(console_txt);

  bool stepperRuns = stepper.isRunning();

  // check border
  if (abs(stepper.currentPosition() + move_steps) > maxMoveRev && !stepperRuns) {
    return false;
  }

  // set new target if stepper is not running
  if (!stepperRuns) {
    // use move, because it will automatically calculate the position relative from the actual value
    stepper.move(move_steps);
  }

  // trigger stepper run
  stepper.run();
  delay(5);
  return true;
}
```

Abbildung 20: MoveStepper

Die Methode **moveStepper** bietet eine allgemeine Möglichkeit, einen Motor um eine bestimmte Anzahl von Schritten (Steps) zu bewegen. Sie liefert einen Rückgabewert, der angibt, ob der Motor tatsächlich bewegt wurde.

Dazu wird überprüft, ob die aktuelle Position des Motors zusammen mit der Anzahl der Schritte, die der Motor bewegen soll, den von uns definierten maximalen Winkel von 45 Grad überschreitet. Ist dies nicht der Fall, bewegt sich der Motor um die übergebene Anzahl von Schritten.

4. Fazit

4.1 Probleme mit dem Arduino

Zu Beginn hatten wir Schwierigkeiten, uns mit dem vom Projekt bereitgestellten Arduino zu verbinden. Um das Problem zu lösen, testeten wir einen eigenen Arduino, doch auch dieser funktionierte nicht. Daher fragten wir nach einem Ersatzgerät und erhielten schließlich den Romeo V1.3, mit dem die Verbindung dann erfolgreich hergestellt werden konnte.

4.2 Probleme mit den Schrittmotoren

In der Materialliste hatten wir ursprünglich Servomotoren angefordert, jedoch erhielten wir stattdessen Schrittmotoren. In früheren Versionen unseres Programms haben wir den maximalen Winkel direkt zu Beginn an den Motor übergeben, um sicherzustellen, dass dieser Wert nicht überschritten wird. Diese Lösung funktionierte jedoch nicht wie erwartet: Wenn eine neue Zielposition übergeben wurde, bevor die alte Zielposition erreicht war, fuhr der Motor zunächst zur alten Position, bevor er die neue Zielposition ansteuerte.

Um dieses Problem zu beheben, haben wir die aktuelle Version des Programms angepasst. Der Schrittmotor bewegt sich nun in kleinen Schritten zur Zielposition. Dadurch wird sichergestellt, dass die Zielposition ohne Umwege erreicht wird.

Abschließend lässt sich sagen, dass wir mit diesem Projekt viel Spaß hatten und auch einiges lernen konnten.