ТЕХНОЛОГИЧЕСКО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ

към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

КУРСОВ ПРОЕКТ

Реализация на 3D повърхност чрез бутала с линейно задвижване

Дипломант:

Мартин Дацев 11Б

Телерик Арсов 11Б

Научен ръководител:

Росен Витанов

СОФИЯ

2018

СЪДЪРЖАНИЕ

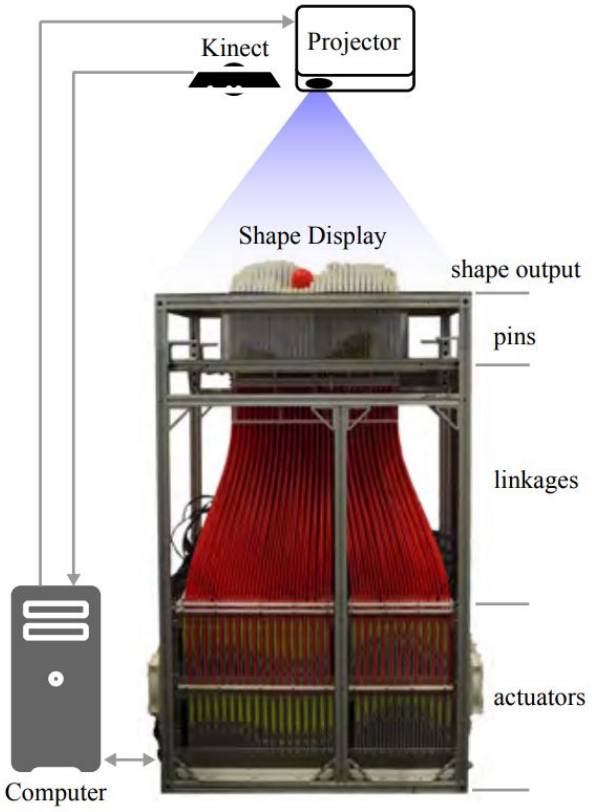
УВОД

Идеята за проекта дойде от вече реализиран проект (inFORM на MiT). Целта на проекта е да се направи повърхност която да може да си променя формата. Това се постига след със бутала с линейно задвижване наредени в някаква форма и ардуино което да ги контролира. В кода за сега е направено буталата да се движат така че да репрезентират математически функции. Освен че нашия проект може да се разширява безкрайно (стига да има ресурсите), той е изцяло open source. Приложенията на проекта са безкрайно много: помагане в обучението чрез пресъздаване на математически функции, интерактивна маса за дома и работата, репрезентация на 3D модели в реалния свят и много други. Също така сме създали и симулатор на проекта, чрез който може да се разработват по-бързо различни функционалности със произволен мащаб без да трябва да се тестват на реалния хардуер.

ПЪРВА ГЛАВА: ПРЕГЛЕД НА ПОДОБНИТЕ ПРОДУКТИ НА ПАЗАРА

1.1 Преглед на подобните продукти

Няма такива продукти на пазара. Всички подобни продукти са само прототипи от различни университети. Също така повечето са много големи и непрактични за комерсиална употреба все още. Един пример за такава система е inFORM на MiT (фигура 1). Тяхната реализация има много голяма основа и е доста обемиста. Също така не е мащабируема, защото всички части са наредени в изработена рамка. Използват се много части доста от които са скъпи. Те включват бутала за всеки пин, kinect, projector, свързващи х. връзки и пр. Също така няма възможност за промяна на конфигурацията (т.е да се подредят в форма освен в квадратна решетка). Освен това проектът им(и кода за него) е затворен и засега не е достъпен до публична употреба.



Фигура 1. структура на inForm.

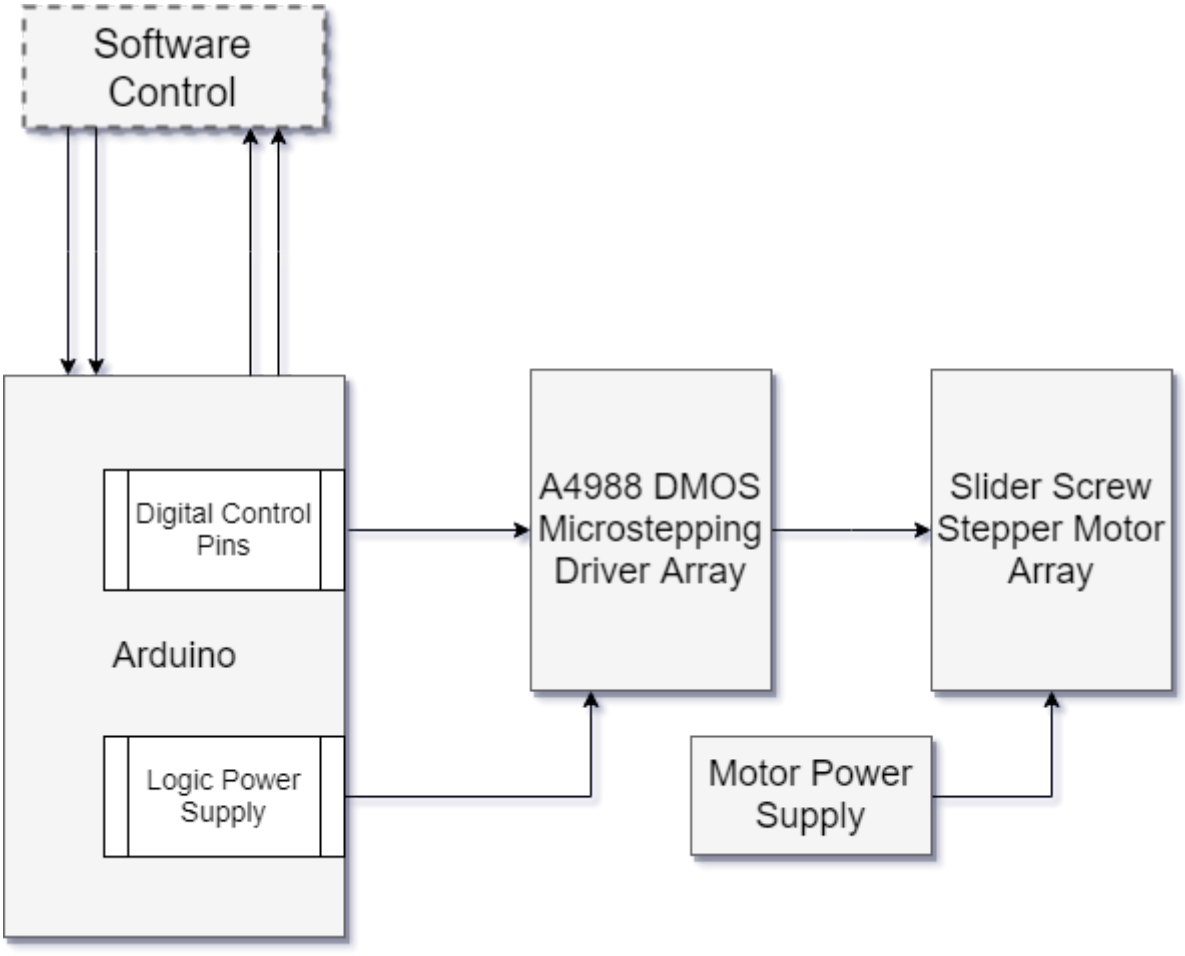
1.2 Нашият проект

За разлика от проекта им нашата реализация е много по-компактна и по-икономична. Частите са много по-евтини и лесно достъпни. Също така е open source и лесно мащабируема. Използваме 5V Mini Slider Screw Stepper Motor 2-Phase 4-Wire Micro Stepper, които представляват нашите бутала. За тяхното контролиране използваме A4988 Stepper Motor Driver With Driver Carrier, който изисква 8-35V power supply за самия мотор, което ние предоставяме с външна батерия и 3-5.5V power supply за самия драйвер, което бива предоставено от ардуиното. Използваме и arduino mega поради големия брой пинове които има. Освен това сме направили и платка по поръчка, чието предназначение е да свързва мотора със драйвера за по-лесна инсталация и употреба.

Самото съоръжение представлява наредени в някаква форма бутала които се задвижват от ардуиното. Мащабируемостта се изразява в това че лесно може да правим инсталацията по голяма, тъй като моторите ни са компактни и лесни за инсталиране и кода е направен, така че лесно да може се променя конфигурацията. Все още нямаме kinect който да възприема движенията на потребителя или прожектор, което е нещо което може да се предприеме за бъдещо развитие.

ВТОРА ГЛАВА: СТРУКТУРА НА ПРОЕКТА И ИЗПОЛЗВАНИТЕ КОМПОНЕНТИ

2.1 Блок схема

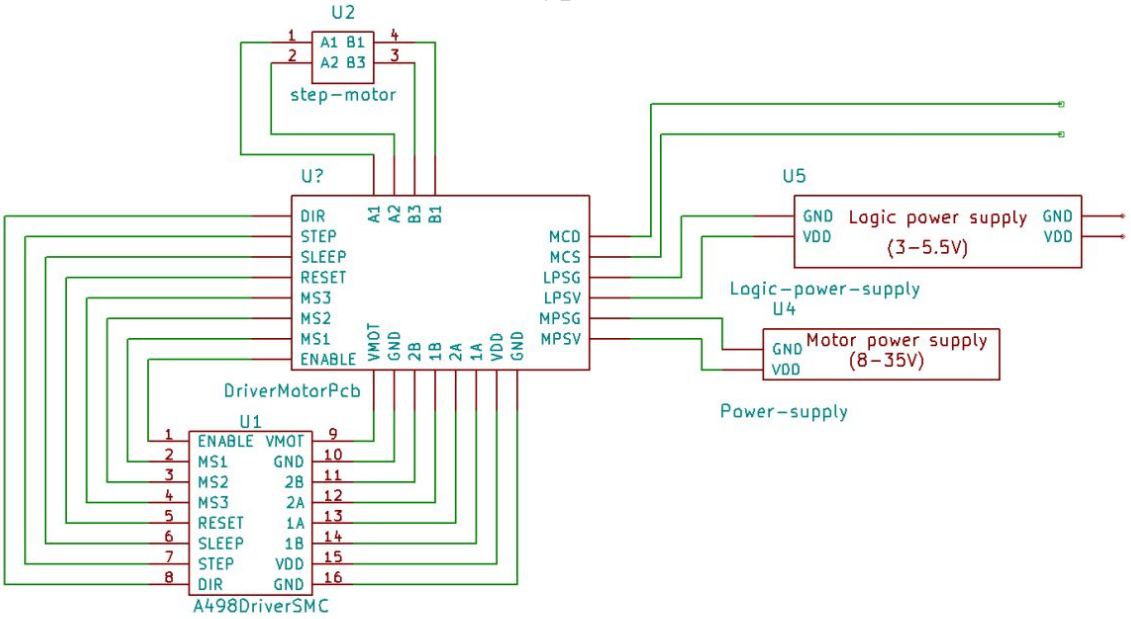


2.2 ИЗПОЛЗВАНИТЕ КОМПОНЕНТИ

2.2.1 Мотор

Мотора е модел 5V Mini Slider Screw Stepper Motor 2-Phase 4-Wire Micro Stepper. За него няма предоставен datasheet. Има 4 пина, 2 за всяка намотка.

Управлява се чрез A4988 Stepper Motor Driver Carrier. Свързването на двата компонента е показано на **Фигура 2**.

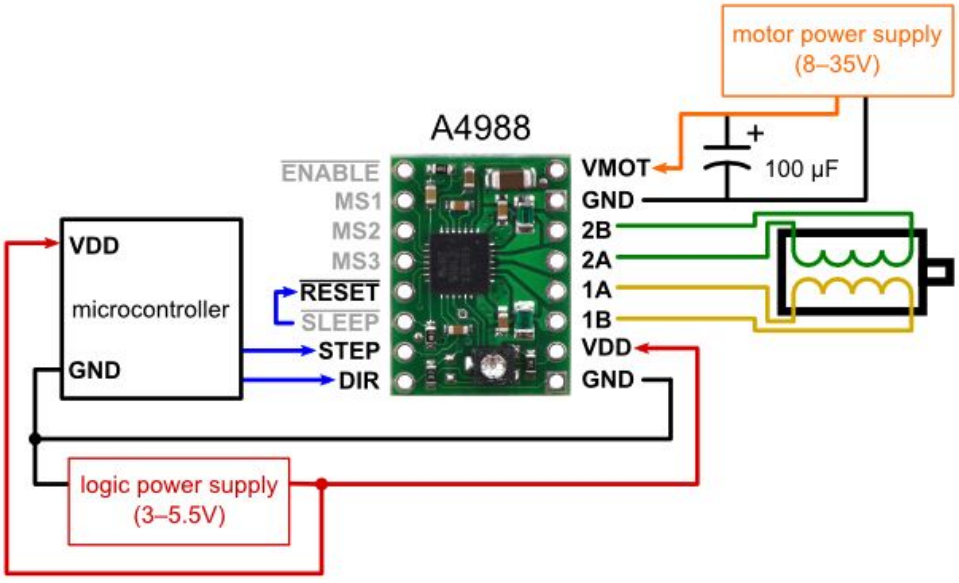


Фигура 2

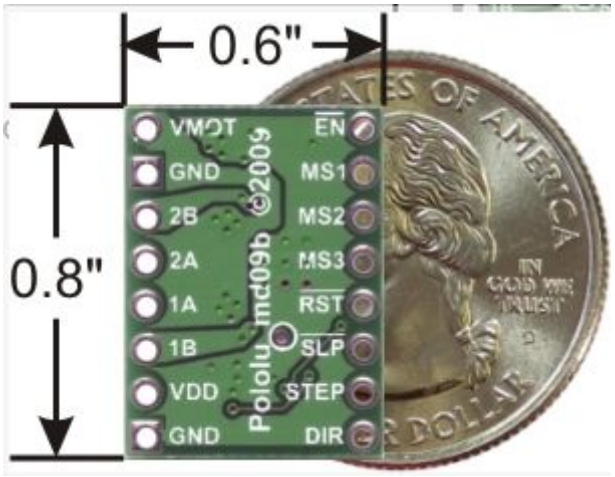
2.2.2 Драйвер

Драйвера е Allegro’s A4988 DMOS Microstepping Driver. Той е поставен във carrier board. Проста диаграма на свързване на driver carriera със микроконтролер и мотора може да се види на **Фигура 3.А.** Размерите на drive carrier-а са 0.8” и 0.6” както е показано на **Фигура 3.Б**. Ето и няколко от функционалностите на драйвера:

- Лесен интерфейс за управление на посоката и стъпката.
- Пет различни вида стъпки, които позволяват по-прецзно движение на моторите, отколкото е възможно без драйвер: full-step, half-step, quarter-step, eighth-step, and sixteenth-step
- Регулируемия контрол над тока позволя да се зададе максималния изходен ток чрез потенциометър, което позволява да се използват напрежения над допустимите за мотора, за да се постигне по-голямо темпо на стъпката.



Фигура 3.А Диаграма на свързване на драйвера.



Фигура 3.Б Размерности на driver carrier

2.2.2.1 Захранване

Драйвера се нуждае от Logic power supply (3 до 5.5V) да бъде свързан на VDD и GND пиновете и Motor power supply (8 до 35 V), който се използва да захранва мотора и трябва да бъде свързан на VMOT и GND пиновете. Драйвера работи със ток не по голям от 4A, както е показано във неговия datasheet.

2.2.2.2 Размера на стъпката (и микростъпката)

Размера на стъпката се определя от MS1, MS2, и MS3 пиновете.

На **Фигура 3.B** може да се види таблица за по-ясно контролиране на стъпката. За по-опростено свързване за сега тези 3 пина сме ги свързали с HIGH чрез самата платка, която сме си направили за свързване на мотора със драйвера, arduino-то и захранванията.

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

Фигура 3.B Таблица за контрол на стъпката

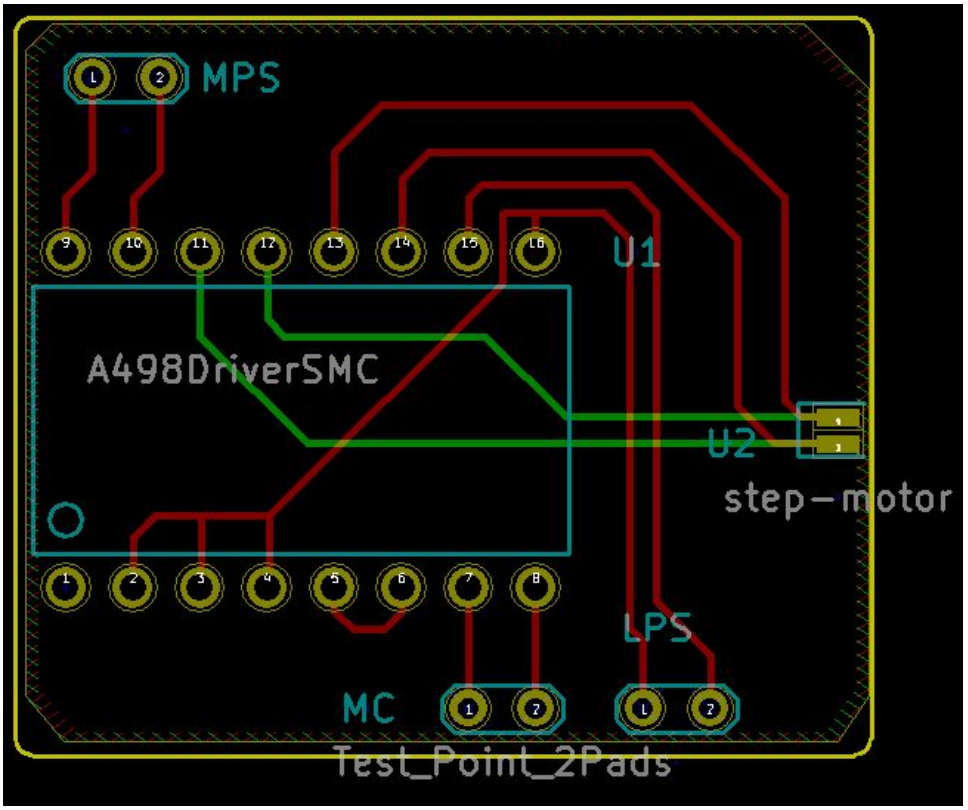
2.2.2.3 Контролни пинове

Всеки импулс към STEP пина съответства на една микростъпка на мотора във посока избрана от DIR пина. Тези два пина винаги трябва да са свързани. За въртене в една посока DIR пина може да се свърже директно към VDD (за посока нагоре) или GND (за посока надолу) пиновете. За да се активира драйвера трябва $\overline{SLEEP}$ пина да се свърже със $\overline{RESET}$ пина, което сме направили върху платката ни.

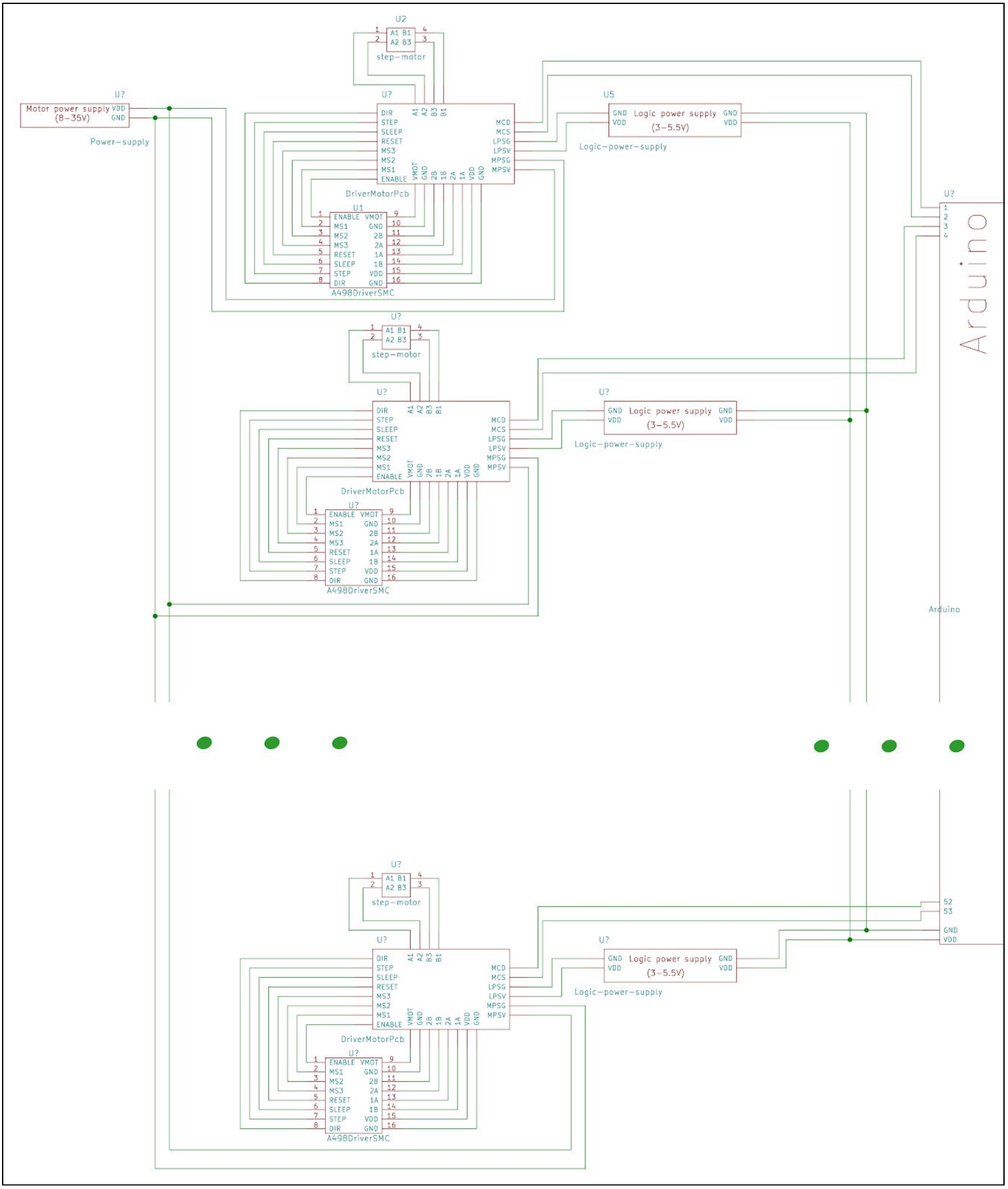
2.2.3 Custom pcb

Направена е и платка по поръчка която да свърже лесно захранването, arduino-то, мотора и драйвера (Показана на **Фигура 4.A**). Електрическа схема на всички компоненти нужни за един мотор, свързани, може да се види на **Фигура 2.** Всяко моторче се свързва по този начин със драйвера. Нареждат се във формата на квадрат, но има възможност и за други конфигурации. Лесно могат да се добавят още компоненти стига да има достатъчно пинове. Driver carrier case-а се

монтира through hole върху pcb-то. Докато мотора се слага чрез surface mount и от двете страни за по-лесно слагане в желаната ориентация. По принцип самия мотор е направен да се монтира through hole, но ние сме направили платката, така че да го монтираме чрез surface mount (по два пина от всяка страна на платката), за да може когато pcb-то е в хоризонтално положения, мотора да е във вертикално. На **Фигура 4.Б** може да се види схема на свързване за всички мотори към ардуиното.



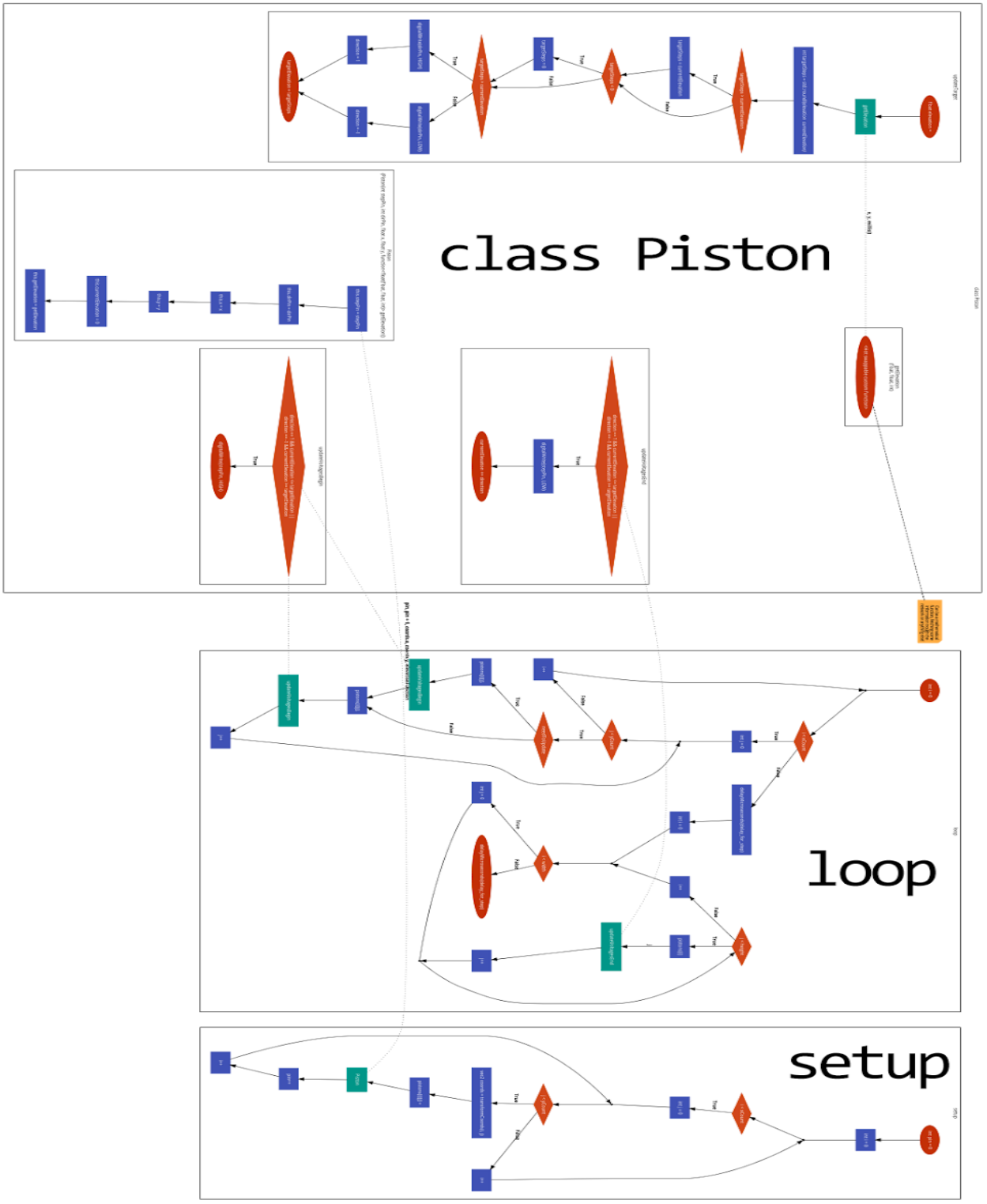
Фигура 4.А PCB



Фигура 4.Б Принципна електрическа схема на свързване на проекта.

ТРЕТА ГЛАВА: БЛОК СХЕМА НА СОФТУЕРА. ОПИСАНИЕ И СТРУКТУРА НА КОДА.

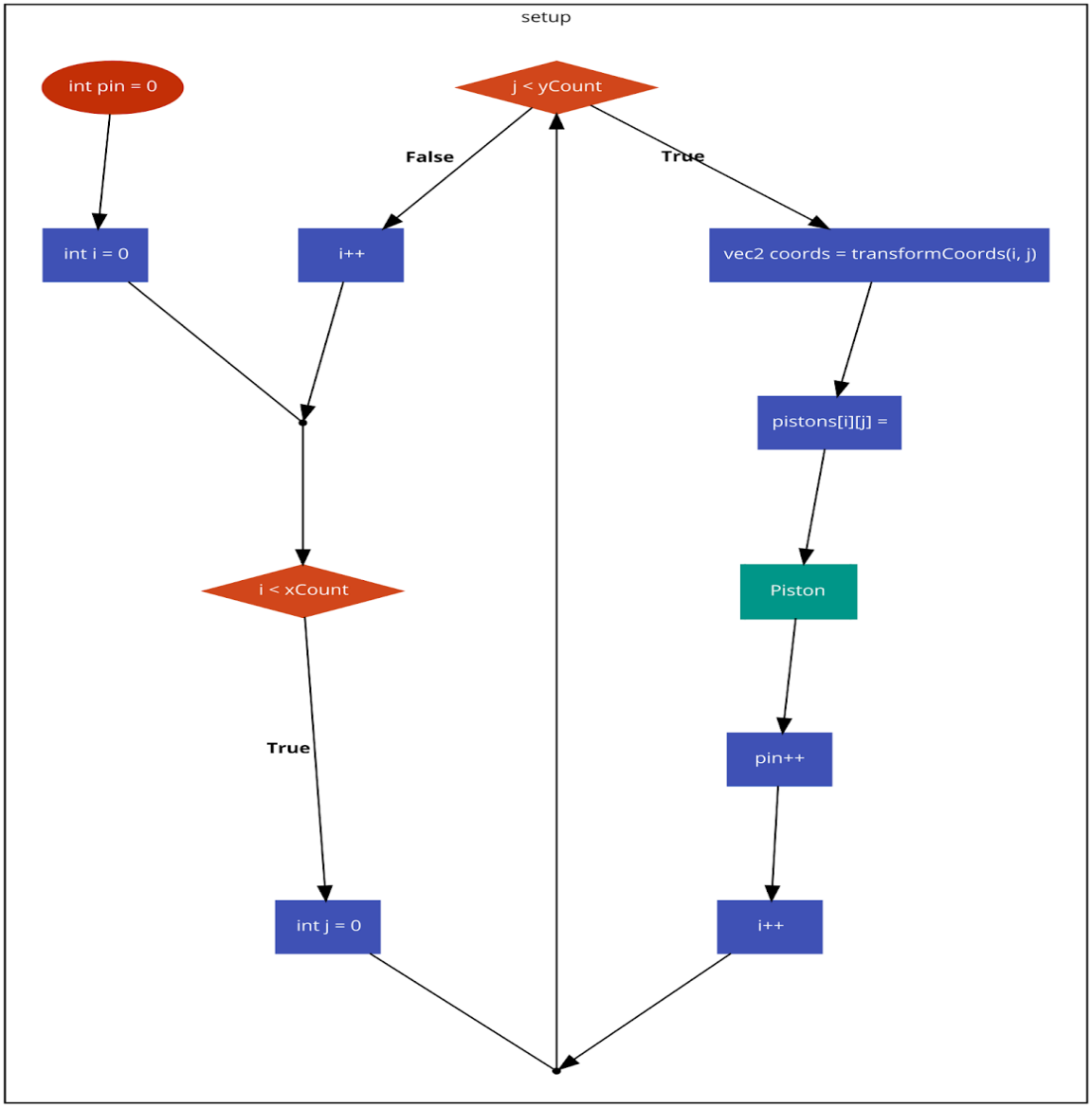
3.1 Блок схема на софтуера



3.1.1 Function setup

(Фигура 5.Д)

Основната функция на setup е да инициализира всички Piston-и със съответните им пинове, координати и функция за определяне на височината.



3.1.2 Class Piston

Класът Piston представлява абстракция за едно бутало и съхранява информация за координатите на буталото, съответно пиновете му за посока и стъпка, моментното му състояние (накъде се движи, и на каква височина е в момента), и указател към функция, която да се използва за определяне на целевата позиция.

3.1.2.1 `Piston` (Constructor)

(Фигура 5.Б)

Конструкторът на класа инициализира обектът като приема следните параметри:

- `int stepPin - пин за стъпка на този мотор`
- `int dirPin - пин за посока на този мотор`
- `float x - математически координати на буталото(в setup се...`
- `float y - ...изчисляват чрез функцията...`

`...toCoord(float x, float y)`(**Фигура 5.г**)

- `float(*getTargetElevation)(float, float, int) -`

Добре е да се обърне специално внимание на `getTargetElevation` параметъра - той представлява указател към функция, която приема три параметъра:

- `float x - координати на буталото`
- `float y - координати на буталото`
- `int time - изминалото време`

В тази функция се намира логиката за вертикалите позиции на буталата. Може функцията да е проста математическа функция или нещо много по-сложно - например да се свърже по мрежата със някакви сензори и по тях да определя как да се издигат буталата. Функцията може да се замени по всяко време без проблем. Примерни функции могат да се видят на **Фигура 5.Ж**

3.1.2.1 `updateTarget`

(Фигура 5.А)

Функцията **`updateTarget`** служи за обновяване на целевата позиция на едно бутало. Прави това по следния начин: Първо извиква `getElevation` и подава координатите на буталото и текущото време за да получи новата целева позиция. След това изчислява колко стъпки трябва да са направени от стъпковия мотор за да се намира на тази позиция и проверява дали позицията е в диапазона на мотора и ако не я слага в него. След това се проверява в коя посока трябва да се завърти мотора, спрямо текущото положение на буталото, за да се достигне целевата позиция и се подава съответното напрежение на dir пина и вътрешната променлива за следене на посоката се задава. Накрая се задава стойността на целевата позиция в класа, която ще бъде използвана, за да се достигне.

3.1.2.1 `updateVoltagesBegin` и `updateVoltagesBegin` (Фигура 5.B)

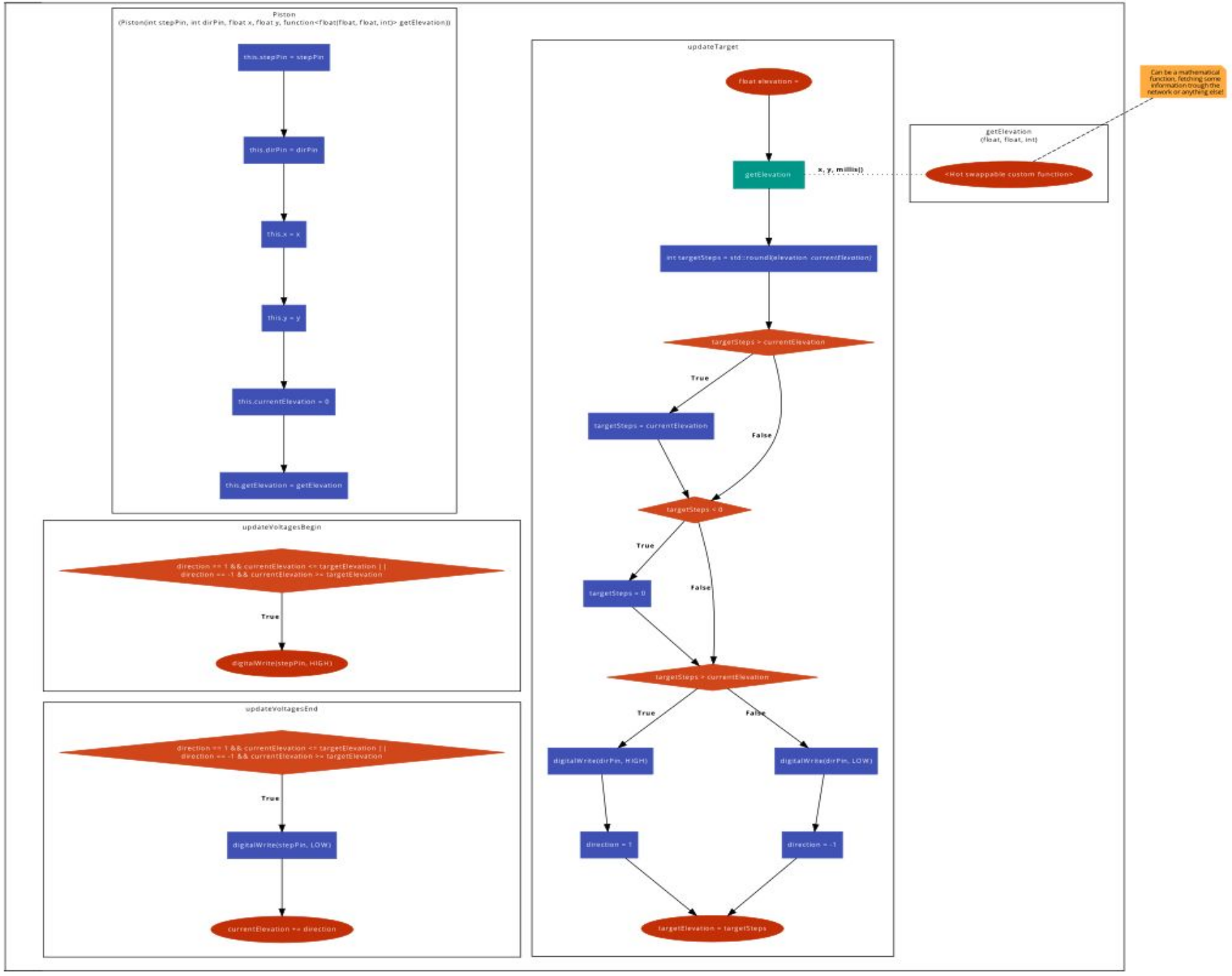
Функциите `updateVoltagesBegin` и `updateVoltagesEnd` служат за задаване на напрежението на импулса, като `updateVoltagesBegin` е за първата част на импулса, а `updateVoltagesEnd` е за последната. Налага се да има две функции, за да може първо да се зададат напреженията на всички мотори и след това да се изчака достатъчно време за да може драйверът да регистрира сигналът и след това да се върнат на предишната стойност и да се повтори процедурата. Тези функции също така проверяват дали целевата позиция е достигната, и ако е - не подава импулс.

3.1.2.1 `~Piston` (Destructor)

Деструкторът служи най-вече за да се постави моторът във занулено положение (тоест на най-ниско ниво). Това е нужно, защото няма как да се провери докъде е завъртян моторът и ако не се ресетне, когато се пусне програмата отново, позицията ще е грешна. Този проблем може да се реши и със някакъв вид сериализация, но така е по-стабилно.

3.2 Симулатор на проекта

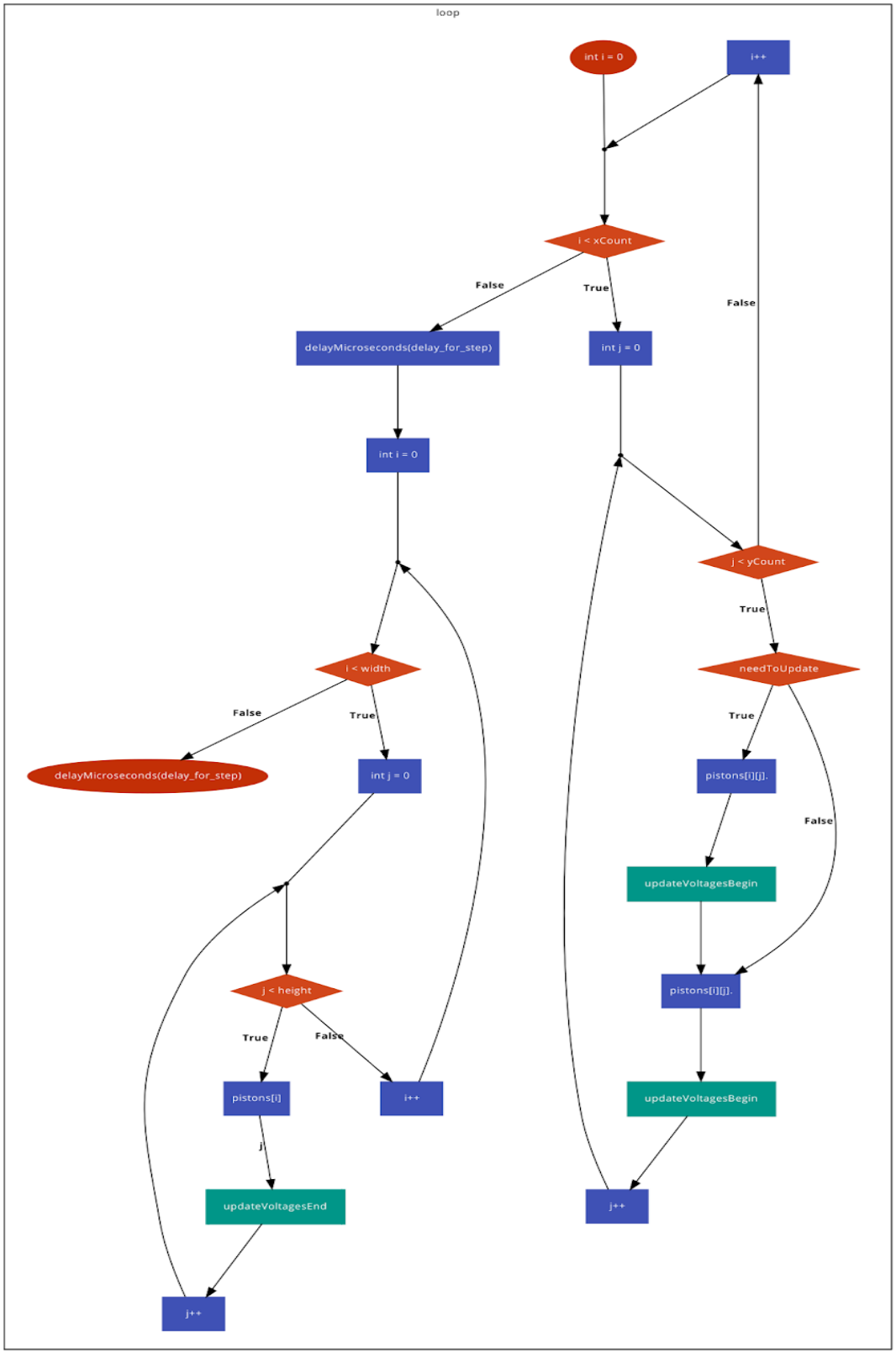
За по лесна разработка и тестване сме разработили симулатор на проекта, в който могат да се задават множество различни параметри на моторите, да се променя конфигурацията и да се задава функцията за определяне положението на мотора (било то математическа функция, или например функция която следи модел и кара моторите да репликират движенията).


Piston
(Piston(int stepPin, int dirPin, float x, float y, function<float(float, float, int)> getElevation))
this.stepPin = stepPin
this.dirPin = dirPin
this.x = x
this.y = y
this.currentElevation = 0
this.getElevation = getElevation
updateVoltagesBegin
direction == 1 && currentElevation <= targetElevation || direction == -1 && currentElevation >= targetElevation
True
digitalWrite(stepPin, HIGH)
updateVoltagesEnd
direction == 1 && currentElevation <= targetElevation || direction == -1 && currentElevation >= targetElevation
True
digitalWrite(stepPin, LOW)
currentElevation += direction
updateTarget
float elevation =
getElevation
x, y, millis()
int targetSteps = std::round(elevation - currentElevation)
targetSteps > currentElevation
True
False
targetSteps = currentElevation
targetSteps < 0
True
False
targetSteps = 0
targetSteps > currentElevation
True
False
digitalWrite(dirPin, HIGH)
digitalWrite(dirPin, LOW)
direction = 1
direction = -1
targetElevation = targetSteps
getElevation
(float, float, int)
<Hot swappable custom function>
Can be a mathematical function, fetching some information trough the network or anything else!

3.1.3 Function loop

(Фигура 5.E)

В loop се извикват методите за актуализиране на напреженията на всяко бутало, изчаква се достатъчно време за да се отчете step сигналът и ако има нужда се изчисляват нови целеви позиции.



3.3 Разглеждане на кода

```
#include <cmath>

class Piston
{
 private:
   int stepPin;
   int dirPin;
   int currentElevation;
   int direction;
   int targetElevation;
   float x;
   float y;
   float (*getTargetElevation)(float, float, int);
 public:
    void updateTarget()
    {
        float elevation = getTargetElevation(x, y, millis());
        int targetSteps = std::roundl(elevation * total_steps);
        if (targetSteps > total_steps)
        {
            targetSteps = currentElevation;
        }
        if (targetSteps < 0)
        {
            targetSteps = 0;
        }
        if (targetSteps > currentElevation)
        {
            digitalWrite(dirPin, HIGH);
            direction = 1;
        }
        else
        {
            digitalWrite(dirPin, LOW);
            direction = -1;
        }
        targetElevation = targetSteps;
    }
```

Фигура 5.А

```
Piston(int stepPin, int dirPin, float x, float y, float
(*getTargetElevation)(float, float, int))
                  : stepPin(stepPin), dirPin(dirPin), x(x), y(y),
currentElevation(0), getTargetElevation(getTargetElevation)
        {
        }
```

Фигура 5.Б

```
        void updateVoltagesBegin()
        {
                    if (direction == 1 && currentElevation <=
targetElevation || direction == -1 && currentElevation >=
targetElevation)
            {
                digitalWrite(stepPin, HIGH);
            }
        }
        void updateVoltagesEnd()
        {
                    if (direction == 1 && currentElevation <=
targetElevation || direction == -1 && currentElevation >=
targetElevation)
            {
                digitalWrite(stepPin, LOW);
                currentElevation++;
            }
        }
```

Фигура 5.В

```
         void setElevationFunction(float (*getTargetElevation)(float,
float, int))
        {
            this->getTargetElevation = getTargetElevation;
        }
    };
```

```
struct vec2
{
   float x;
   float y;
};

constexpr int xCount = 5;
constexpr int yCount = 5;

Piston pistons[xCount][yCount];

float xScale = 1;
float yScale = 1;
float scale = 1;
float timeScale = 1;
```

```
vec2 transformCoords(int i, int j)
{
   return {xCount / 2 - i, yCount / 2 - j};
}
```

Фигура 5.Г

```
float (*const elevationFunction)(float, float, int) = sineWave;
```

```
void setup()
{
   int pinCounter = 0;
   for (int i = 0; i < xCount; i++)
   {
       for (int j = 0; j < yCount; j++)
       {
           vec2 coords = transformCoords(i, j);
             pistons[i][j] = Piston(pinCounter, pinCounter + 1,
coords.x, coords.y, elevationFunction);
           pinCounter += 2;
       }
   }
}
```

Фигура 5.Д

```
void loop()
{
   for (int i = 0; i < xCount; i++)
   {
       for (int j = 0; j < yCount; j++)
       {
           if(need_to_retarget)
           {
               pistons[i][j].updateTarget();
           }
           pistons[i][j].updateVoltagesBegin();
       }
   }
   delayMicroseconds(delay_for_step);
   for (int i = 0; i < xCount; i++)
   {
       for (int j = 0; j < yCount; j++)
       {
           pistons[i][j].updateVoltagesEnd();
       }
   }
   delayMicroseconds(delay_for_step);
}
```

Фигура 5.Е

```
/*
   Moving 3D sine wave
*/
float sineWave(float x, float y, int millis)
{
    return std::sin(10 * (x * x + y * y) + millis / timeScale) *
scale;
}
```

Фигура 5.Ж

ГЛАВА ЧЕТИРИ: РАЗХОДИ

4.1 Цена на компонентите.

2-Phase 4-Wire Micro Stepper Motor - 30бр. - US $1.01 / piece (**Фигура 6.А**)

Фигура 6.А **Фигура 6.Б**

Stepper Motor Driver A4988 - 30бр. - US $1.14 / piece (**Фигура 6.Б**)

Arduino mega - 1бр. - US $7.45 / piece

Breadboard - 1бр. - BG 10lv / piece

Проводници - 20бр. - BG 10lv / total

PCB - 30бр. - BG 1.5lv / piece (rough estimate)
Батерии и захранвания - BG 6lv / total
Изработена по поръчка резба - ?

ОБЩО: 191.7+лв.