SZÉCHENYI EGYETEM
UNIVERSITY OF GYŐR
GÉPÉSZMÉRNÖKI, INFORMATIKAI ÉS VILLAMOSMÉRNÖKI KAR

Projektlabor II.

Redőny vezérlésének mikrokontrolleres megvalósítása

Zsámpár Bálint

Villamosmérnöki BSc szak

Automatizálási szakirány

2023.

Tartalomjegyzék

1. Bevezetés

Napjainkban a technológiai fejlődés az élet szinte minden területén fellelhető, szinte napról- napra történnek a változások. Az ipar elképesztő sebességű fejlődése mellett azonban a mindennapi életünkben is hatalmas változások állnak be az okosotthonos megoldások elterjedésének köszönhetően. Az automatizálás ebben a szegmensben is soha nem látott méreteket kezd ölteni.

Mára már képesek vagyunk távolról figyelni az otthonunkban történő eseményeket a telefonunkról, képesek vagyunk pár kattintással beállítani a benti hőmérsékletet, vagy akár a napsütéshez idomulva automatikusan beállítani az árnyékolókat. A közeljövőben pedig még ennél is nagyobb újítások jöhetnek, amelyek még sokkal könnyebbé tehetik a mindennapi életünket.

Dolgozatomban ennek a technológiai iránynak az egyik ágát, az árnyékolók automatizálását szeretném bemutatni, és modellezni. Bemutatásra kerül, hogyan is lehet fizikailag megépíteni egy ilyen eszközhöz a megfelelő vezérlést, valamint ehhez a rendszerhez elkészíteni a megfelelő szoftvert is. A végeredmény pedig egy olyan eszköz lesz, amely képes a telefonunkról kiadott utasítás alapján működni. Ez magában foglalja a kézi fel és lefelé irányítást, illetve a kívánt fényviszonyok megadását, amit követően az eszköz önműködően képes megtalálni a kívánt beállítást.

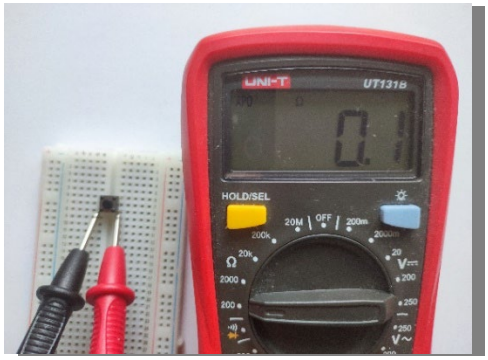
2. Kézi működtetés

Az eszköz elkészítése három nagy folyamatra bontható. Az első folyamatot hivatott bemutatni ez a fejezet, melynek során elkészítem a működés alapvető logikáját, felépítését, és kézi működtetéssel, nyomógombokkal már teljesen működőképes állapotban lesz az eszköz.

2.1 Kapcsolás tervezése

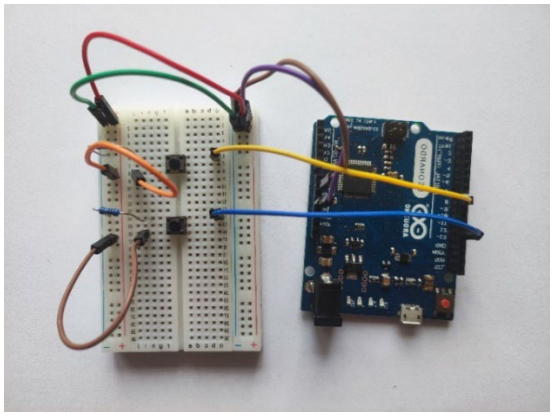
Ebben a szakaszban tehát a kézi működtetéshez tervezem meg a megfelelő hardveres kialakítást. A kialakítás első körben próbapanelen történik, majd miután elkészült a projekt összes eleme, azt követően kerül majd fizikailag használható formájában megépítésre. A projekt megvalósításának alapját egy Arduino Leonardo lapka biztosítja, amely kellően elegendő számú bemenettel és kimenettel rendelkezik (20 darab) ahhoz, hogy a projekt megvalósítható legyen.

A működéshez első körben szükség lesz két darab nyomógombra, mindkettő az egyik forgásirányért lesz felelős. A nyomógomb nyitott állapotban igen kicsi ellenállásként viselkedik, ahogy az *1. ábra* alapján is látható, az itt felhasznált nyomógombok ellenállása nyitott állapotban 0,1Ω.



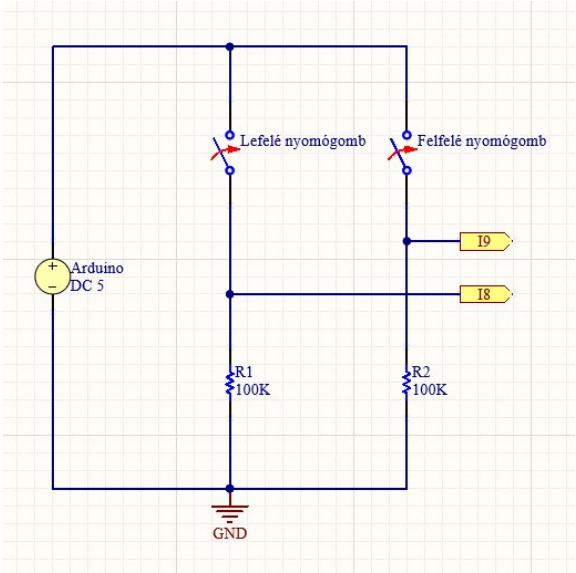
1. ábra. Nyomógomb ellenállása.

Emiatt a tulajdonsága miatt nem kapcsolható közvetlenül a bementi lábakra, hiszen ez a feszültségszint ingadozását okozhatja, ami miatt bizonytalan működés állna elő. A megoldást egy ellenállás sorba kötése jelenti a nyomógombbal, amely kellően nagy kell legyen, ebben az esetben 100kΩ. Ennek két féle megoldása létezik, felhúzó és lehúzó ellenállásokról beszélhetünk, attól függően, hogy a feszültséget minimum vagy maximum értékre állítja. Ebben az alkalmazásban lehúzó ellenállásra van szükségünk a *2. ábra* alapján látható bekötés szerint, ezért az ellenállást a nyomógomb és a nulla közé kötöm be, innen veszi le a jelet az alaplap a bemenetre. A bemeneti jelet innentől kezdve a szoftver kezeli. Mivel képről nehezen követhető az egyes folyamatok bekötése, ezért Altium Designer szoftverrel minden esetben, amikor azt szükségesnek találom elkészítem a kapcsolási rajzot is. Az első ilyen a *3. ábrán* látható.



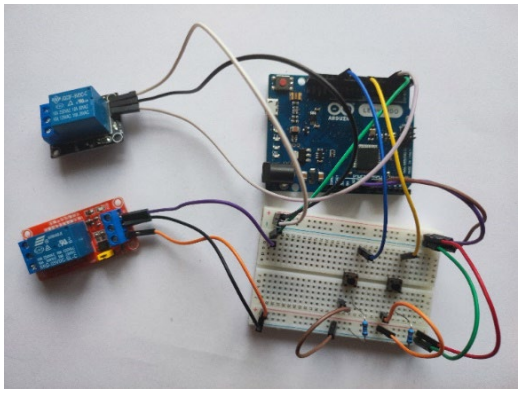
2. ábra. Bemeneti jelet adó kapcsolás.

A következő lépés a bemeneti jel alapján kimenetet szolgáltatni a relék felé. Mindkét relé hat csatlakozási ponttal rendelkezik, ezek közül ötre lesz szükség. A vezérlő bemenetekből három található, az 5 voltot igénylő, a 0 voltot igénylő, valamint a vezérlőjel, ezeket mind fel kell használni.



3. ábra. Nyomógombok bekötése.

A vezérelt kimenetek három része a közös pont, az alaphelyzetben zárt (NC) kapcsoló, valamint az alaphelyzetben nyitott kapcsoló (NO). Mivel a megvalósítás során akkor szükséges kimeneti jelet adni a motor felé, mikor a nyomógomb lenyomásra kerül, ezért az alaphelyzetben nyitott kapcsolót érdemes felhasználni, az elkészült kapcsolást az *4. ábra* szemlélteti.



4. ábra. Rendszer a relék rákötése után.

2.2 Szoftver

Az eszköz fizikai megépítése után ennek szoftvere kerül megírásra. Mivel a mikrokontroller szerepét egy Arduino lapka tölti be, ezért a kódot C nyelven írom, amihez az Arduino saját IDE programját fogom felhasználni. Elsőként az alaplap típusát és port számát érdemes kiválasztani, mivel bár a programot enélkül is meg lehet írni, tesztelésre csak ezt követően van lehetőség. A felhasznált alaplap Arduino gyártmány, ezért a listából könnyen ki lehet keresni. Más gyártók esetén előbb megfelelő könyvtárak letöltésére lesz szükség.

Bár a program C nyelven íródik, a programnak megvannak az Arduinohoz köthető sajátosságai.

Két nagy blokkból épül fel, az egyik a *setup*, a másik pedig a *loop.* A *setup* blokk indításkor kerül lefuttatásra, ezt követően nem foglalkozik vele többet a hardver. Főként a program számára fontos változók deklarálása történik ebben a blokkban. Ezt követi a *loop*, amely egy folyamatosan ismétlődő ciklus, ami egészen a leállításig tart. Ebben a blokkban történik a folyamatok érdemi része.

Ezen felül saját parancsokkal is rendelkezik az Arduino rendszere, ezek a teljesség igénye nélkül az *1. táblázatban* találhatóak[1].

1. táblázat. Beépített parancsok.

pinMode(láb, INPUT/OUTPUT)	Adott lábnak meghatározza, hogy kimenetként vagy bemenetként viselkedjen e.
digitalRead(láb)	Adott lábról digitális bemenetet olvas.
digitalWrite(láb, érték)	Adott lábra digitális kimenetet ír.
analogRead(láb)	Adott lábról analóg bemenetet olvas.
analogWrite(láb, érték)	Adott lábra analóg kimenetet ír.
Serial.println(változó)	Soros portra írás.
delay(ms)	Késleltetés adott milliszekundummal

A korábban leírtak szerint a program elején a különböző változók deklarálása található az *5. ábrán* látható módon.

```
int BUTTON_UP=12;
int BUTTON_DOWN=8;
int UP=3;
int DOWN=4;


void setup() {
  pinMode(BUTTON_UP, INPUT);
  pinMode(BUTTON_DOWN, INPUT);
  pinMode(UP, OUTPUT);
  pinMode(DOWN, OUTPUT);
}
```

5. ábra. Kezdeti változók.

A *setup* előtti deklarációkkal létrehozzuk a változókat, amelyekben a felhasznált lábak sorszámait tároljuk. Erre nem feltétlenül lenne szükség, azonban a későbbi könnyebb módosíthatóság érdekében érdemes ezeknek külön változókat létrehozni, hiszen később egy módosítással megváltoztatható több előfordulás értéke, valamint könnyebben megtalálhatóak ezek az értékek.

Ezt követi a *setup* függvény, ahol a kívánt lábakra beállítjuk, hogy bemenetként, vagy kimenetként szeretnénk használni őket. Ehhez a *pinMode* függvényt tudjuk segítségül hívni, megadva a kívánt lábat, és üzemmódot.

Később ezeket a beállított lábakat felhasználva fogunk tudni a *loop* függvényben adatokat

küldeni, vagy beolvasni a mikrokontrollerről. Itt a *digitalWrite* és *digitalRead* függvényeket fogjuk felhasználni.

A program alapvető működésének kulcsfontosságú része a motor részére kiadott jel biztosítása, amit utána a relék kezelnek. Annak biztosítására, hogy egyszerre ne kaphasson mindkét irány jelet, a XOR (kizáró vagy) logikát kell segítségül hívni, programbeli megvalósítása a *6. ábrán*, igazságtáblázata a *2. táblázatban* látható.

```
int pressed_up=digitalRead(BUTTON_UP);
int pressed_down=digitalRead(BUTTON_DOWN);
if (((pressed_up == LOW) || (pressed_down == LOW)) && !((pressed_up == LOW) && (pressed_down == LOW)))
{
  if(pressed_up) digitalWrite(UP, LOW);
  if(pressed_down) digitalWrite(DOWN, LOW);
}
else {
  digitalWrite(UP, HIGH);
  digitalWrite(DOWN, HIGH);
}
```

6. ábra. XOR logika megvalósítása C programnyelvben.

2. táblázat. XOR logika igazságtáblázata.

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Ahogy az igazságtáblázatból is láthatjuk, ezt a logikát alkalmazva csak abban az esetben jut vezérlés a kimenetre, amennyiben pontosan egy változó igaz állapotot vesz fel. A feltételen belül pedig már meg tudjuk határozni, hogy melyik érték esetén milyen kimeneti jel érvényesüljön. Amennyiben pedig egyik változó sem igaz vagy mindkettő igaz, abban az esetben a kimenetre nem juthat jel, a kimenetek 0 értéket vesznek fel. Az elkészült programot a *7. ábra* szemlélteti.

```
int BUTTON_UP=12;
int BUTTON_DOWN=8;
int UP=3;
int DOWN=4;


void setup() {
  pinMode(BUTTON_UP, INPUT);
  pinMode(BUTTON_DOWN, INPUT);
  pinMode(UP, OUTPUT);
  pinMode(DOWN, OUTPUT);
}

void loop() {
  int pressed_up=digitalRead(BUTTON_UP);
  int pressed_down=digitalRead(BUTTON_DOWN);
  if (((pressed_up == LOW) || (pressed_down == LOW)) && !((pressed_up == LOW) && (pressed_down == LOW))
  {
    if(pressed_up) digitalWrite(UP, HIGH);
    if(pressed_down) digitalWrite(DOWN, HIGH);
  }
  else {
    digitalWrite(UP, LOW);
    digitalWrite(DOWN, LOW);
  }
}
```

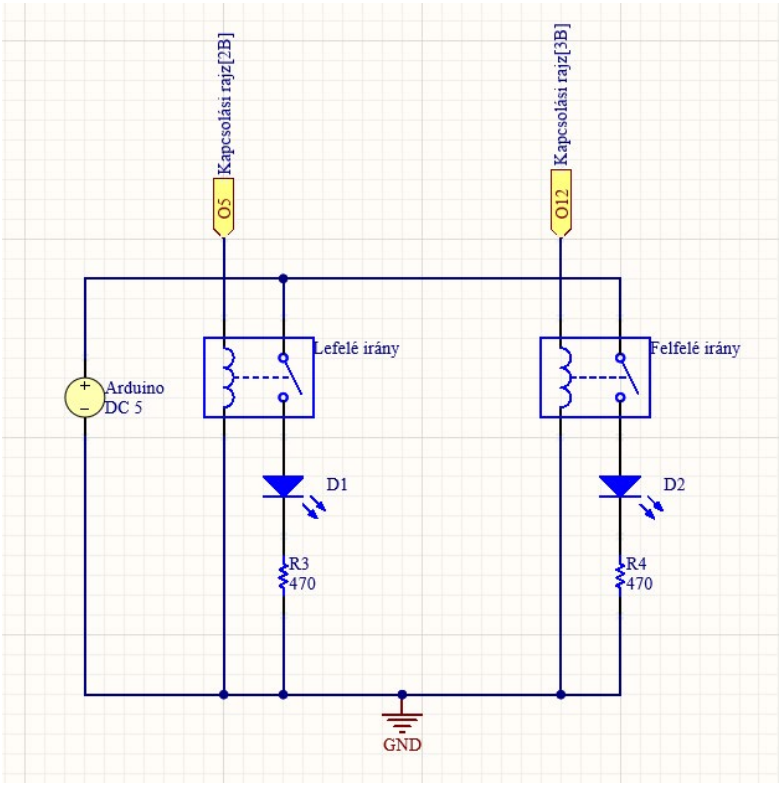
7. ábra. Teljes program.

2.3 Teszt

Miután elkészült a hardver és a szoftver is a tényleges felhasználás előtt érdemes egy tesztet lefuttatni, hogy a rendszer valóban a kívánalmaknak megfelelően működik e. Ehhez LED-eket fogunk segítségül hívni. Mindkét forgásirányra egy-egy LED-et rákötve jól szimulálhatjuk, hogy a kívánt jelszint jut-e a kimenetekre.

Az Arduino kimeneti lábai 5 voltot szolgáltatnak, 10mA-es áramhoz tehát 500Ω értékű előtétellenállást kell választanunk, így a LED már megfelelő áram és feszültség szintet kap ahhoz, hogy megfelelően működjön. Az ellenállások szabványértékű gyártása miatt ez a gyakorlatban 470Ω értékű ellenállást jelent.

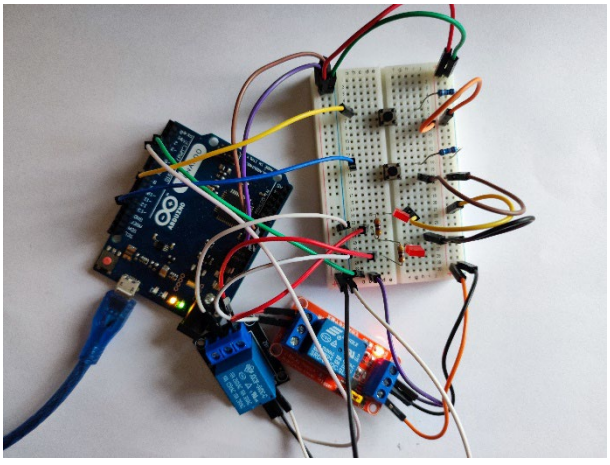
Mivel a LED polaritásérzékeny eszköz ezért figyelni kell a megfelelő irányú beültetésre is, az eszköz anódja (hosszabb láb) kell, hogy a nullapont felé mutasson. Így a beépítés során a LED katódjára pozitív tápfeszültséget kapcsolunk. Az anódra rákötjük a kiválasztott ellenállást, majd tovább vezetjük a relé alaphelyzetben nyitott lábára, a COM közös pontot pedig tovább visszük a nullapont felé. A kapcsolási rajzot a *8. ábra* szemlélteti.



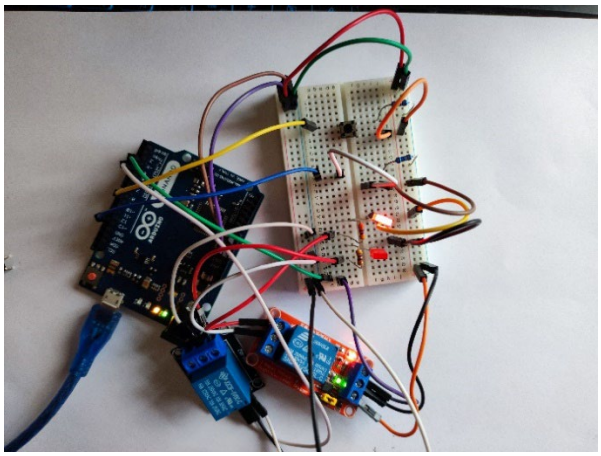
8. ábra. LED bekötési rajza.

Az előzetes elvárás a kapcsolástól az, hogy egyik nyomógombot lenyomva az egyik, másik nyomógombot lenyomva a másik LED világítson. Mindkét nyomógombot lenyomva egyik LED sem világíthat, ahogy alaphelyzet sem. A könnyebb ellenőrizhetőség érdekében a lenyomást rövidzárral helyettesítjük.

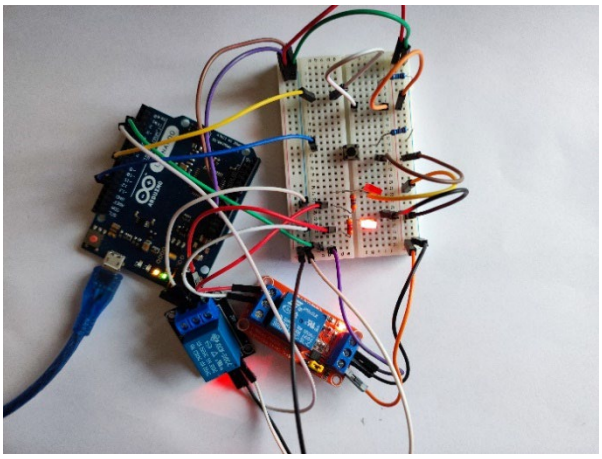
Az eredménye a tesztnek a 9- *10- 11- 12 ábrákon* látható.



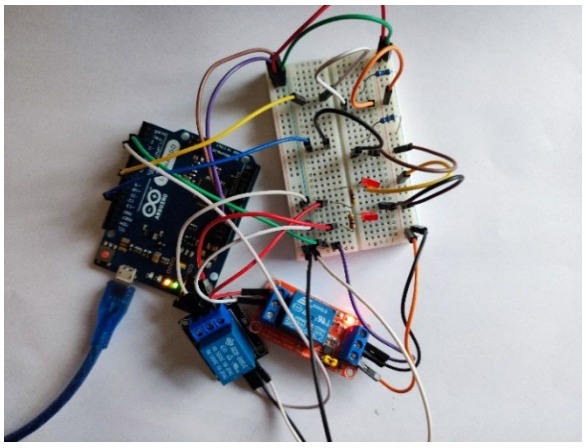
9. ábra. Alaphelyzet (nyomás nélkül).



10. ábra. Egy nyomógomb lenyomva.



11. ábra. Másik nyomógomb lenyomva.



12. ábra. Mindkét nyomógomb lenyomva.

A képekből jól látható, hogy valóban egyszerre maximum egy LED lehet aktív, ami gyakorlati szempontból azt jelenti, hogy egyszerre egy forgásirány lehet aktív.

2.4 Véghelyzet érzékelés

Miután az eszköz képes már a forgásirányok kezelésére, a következő lépés határt szabni ennek a folyamatnak. A valós működés során a teljesen felhúzott, és teljesen leengedett állapot elérésekor a motornak abba kell hagynia a forgatást. Valamint előfordulhatnak hibák, amik miatt a motor nem képes ezt a pozíciót elérni, ebben az esetben egy időkorlátot célszerű megadni, amennyi időn belül, ha az eszköz nem éri el valamelyik véghelyzetet, akkor a motor automatikus leáll. Ezen problémák megoldásához szoftveres és hardveres megoldásokat is alkalmaznunk kell, ezeket hivatott bemutatni ezen fejezet.

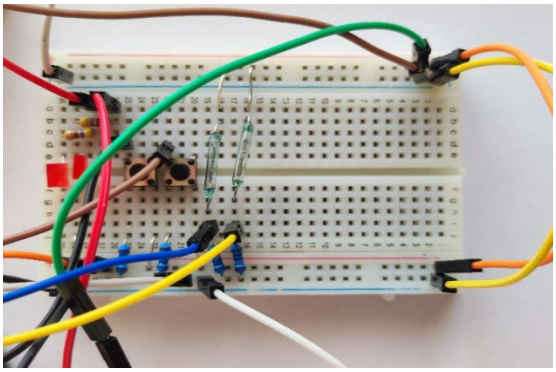
Elsőként a véghelyzetek érzékelése kerül ismertetésre. Ehhez szükségünk lesz két darab érzékelőre, amely érzékelni képes, ha az eszköz elérte a véghelyzetet. Esetünkben két darab a *13. ábrán* látható Reed- relé fogja betölteni ezt a szerepet.

13. ábra. Reed relé.

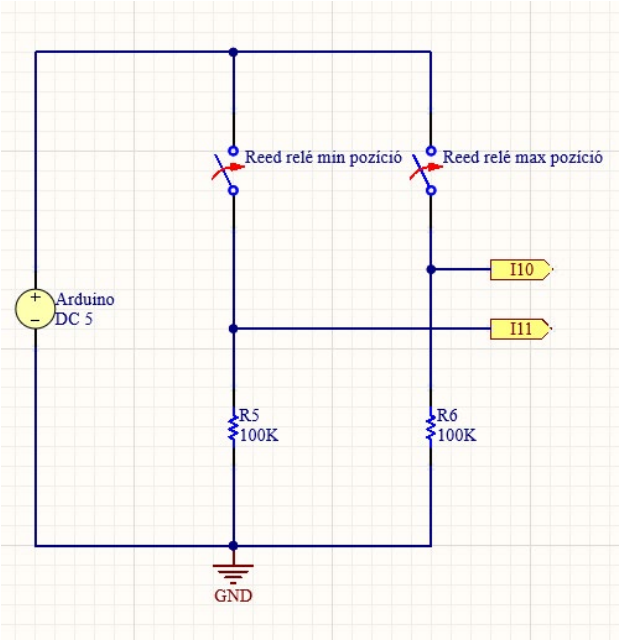
A Reed- relék kisméretű üvegcsövek, amelyek védőgázzal töltöttek. Ezen belül helyezkedik el két ferromágneses érintkező, egymástól igen kis távolságra, de még érintkezés nélkül. Mágneses tér közelében ezek az érintkezők ellentétes polaritású válnak, így vonzani kezdik egymást, míg végül összeérnek. Ezt kihasználva az eszköz mágneses térrel vezérelhető kapcsolóként működik.

Mivel a redőnyök általában műanyagból készülnek, így a gyakorlatban ezekre egy apró mágneses anyagot szerelve könnyedén érzékelhető a két véghelyzet ezen érzékelőkkel.

A nyomógombhoz hasonlóan zárt állapotban ezen eszközök ellenállása is igen kis értékű, ezért itt is hasonlóan eljárva egy 100kΩ értékű ellenállás kerül sorba kötve vele, lehúzó ellenállásként. Az elkészült kapcsolást a *14.és 15. ábra* szemlélteti.



14. ábra. Reed- relé bekötése.



15. ábra. Reed relé kapcsolási rajza.

A bekötés a programban minimális változást okoz csak. Létre kell hozni két változót a két új bemeneti láb jegyzésére, valamint újabb kettőt a lábakon lévő érték olvasására (*max_position, min_position*). Ezt a két változót pedig felhasználhatjuk feltételként az irányok bekapcsolásához, amely csak abban az esetben aktiválódik, ha az adott irányhoz kapcsolódó nyomógomb aktív, míg a hozzá tartozó szenzor nem aktív. A programrészlet a *16. ábrán* látható.

```
int pressed_up=digitalRead(BUTTON_UP);
int pressed_down=digitalRead(BUTTON_DOWN);
int max_position=digitalRead(MAX);
int min_position=digitalRead(MIN);
if (((pressed_up == LOW) || (pressed_down == LOW)) && !((pressed_up == LOW) && (pressed_down == LOW)))
{
  if(pressed_up && !max_position) digitalWrite(UP, HIGH);
  if(pressed_down && !min_position) digitalWrite(DOWN, HIGH);
}
else {
  digitalWrite(UP, LOW);
  digitalWrite(DOWN, LOW);
}
```

1. ábra. Program a véghelyzetérzékelők felhasználásával.

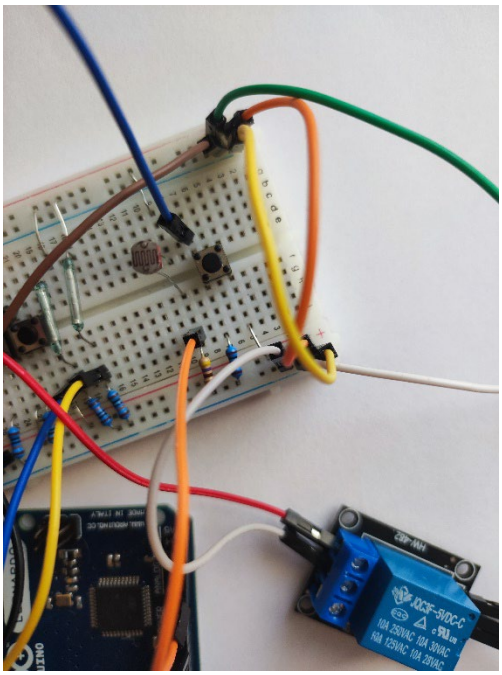
3. Önműködő üzemmód

Miután az eszköz kézi működtetéssel már képes a kívánalmaknak megfelelően működni, a következő lépés az automatikus működtetés létrehozása. Ahogy korábban is, ez a folyamat is lépésenként fog felépülni, először a hardveres, majd a szoftveres részt megalkotva kerül bemutatásra.

3.1 Kapcsolás tervezése

A kézi működtetés létrehozásához már elkészült a hardveres felépítés jelentős része, ezt már csak egy érzékelővel kell kiegészíteni, ami képes az aktuális fényviszonyok érzékelésére. Ehhez tökéletes választás egy fotorezisztor alkalmazása. A bekötése hasonló módon épül fel, mint a korábbi alkatrészeknek, egyik lábát a tápfeszültségre kötjük, másik lábát egy ellenálláson keresztül a 0V-ra. Az eltérést mindössze az ellenállás nagysága adja, korábban, digitális jelek olvasására 100kΩ nagyságú ellenállás is megfelel, mivel azonban itt analóg jelet fog az eszköz figyelni, ezért a minél nagyobb értékváltozás a kívánatos, emiatt egy sokkal kisebb, 470Ω értékű ellenállás kerül beépítésre. A jelet a fotorezisztor, és az ellenállás lába közül vesszük le, és az eddigiektől eltérően az analóg jeleket kezelő egyik lábra kötjük rá.

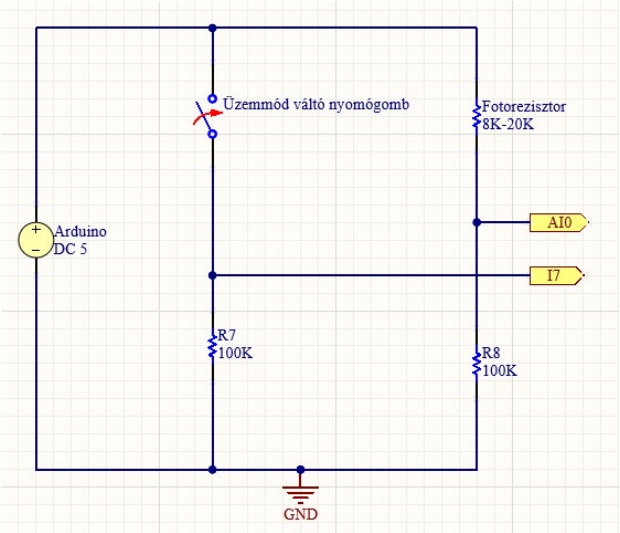
Befejezésként egy nyomógombot kell beépítenünk, hogy a kézi és automatikus üzemmódok között léptetni tudjuk. Ennek bekötése a már a korábbi fejezetben ismertetett módon történik, az elkészült kapcsolást a *17. és 18. ábra* szemlélteti.



2. ábra. Fotorezisztor és nyomógomb bekötése.

3.2 Szoftver

A szoftveres részben először is két új változót kell megadnunk, melyek a kezelni kívánt lábakat adják meg. Eltérés, hogy az analóg láb jelölésére a szám elé egy A betű kerül, például A0. A *setu*p és *loop* függvényekben az alap teendők is megegyeznek a korábbiakkal, *pinMod*e beállítása, és *digitalRead*, *analogRead* függvényenek hívása. *analogRead* függvénynél érdemes figyelni rá, hogy int típus



18. ábra. Fotorezisztor és nyomógomb bekötés kapcsolási rajza.

helyett *float* típust adjunk meg, mivel nem egész szám is előfordulhat az olvasott értékek között, esetünkben azonban ennek nincs jelentősége. Egy „belső” változóra is szükség lesz a folyamathoz, amit érdemes globális változóként létrehozni, egyrészt, hogy bárhonnan elérhető legyen, másrészt mivel, ha a *loop* függvényeben belül inicializálnák, akkor minden ciklusban is újra 0 értéket venne fel, ezzel nem kívánt működést eredményezve.

A változók létrehozása után az első lépés az üzemmód meghatározása kell legyen. Ehhez a programnak meg kell vizsgálni, hogy az üzemmód váltó nyomógomb aktív-e, ha igen, ellentétére változtatja az aktuális üzemmódot, majd vár fél másodpercet, időt adva a gomb elengedésére.

A 0 üzemmód a kézi üzemmódot indítja, ennek programja a korábbi fejezetben megírt kódrészlet. Az önműködő üzemmód 1-es értéknél indul el. Tapasztalati úton meghatározható, hogy az analóg érték körülbelül 0 és 800 között mozog a választott ellenállás érték miatt, minél nagyobb a fény, annál nagyobb értékkel. Csak összehasonlításképpen, ha az ellenállás értéke a többször alkalmazott 100kΩ lett volna, az érték 1000 és 1020 között mozgott volna, ami sokkal kisebb variációs lehetőséget biztosít. Ezt követően egyéni preferencia meghatározni milyen értékek esetén próbálja növelni (redőny fel), vagy csökkenteni (redőny le) a beeső fény mennyiségét. Az indítás természetesen csak akkor történik meg, ha lefelé mozgáskor még nincs minimum helyzetben, vagy felfelé mozgáskor maximum helyzetben már indítás előtt.

Az értéktartományok mivel nem esnek egybe, ezért egyszerre a két forgásirány nem lehet aktív, ennek elkerülését így nem is kell biztosítani. Egészében a program a *19. ábrán* látható.

```
int mode=digitalRead(SWITCH);
float light=analogRead(LIGHT);
if(mode)
{
  if(!mode_change) mode_change=1;
  else mode_change=0;
  delay(500);
}
if(!mode_change)
{
      if (((pressed_up == LOW) || (pressed_down == LOW)) && !((pressed_up == LOW) && (pressed_down == LOW)))
      {
        if(pressed_up && !max_position) digitalWrite(UP, HIGH);
        if(pressed_down && !min_position) digitalWrite(DOWN, HIGH);
      }
      else
      {
        digitalWrite(UP, LOW);
        digitalWrite(DOWN, LOW);
       }
}
else
{
  Serial.println(light);
  if(light>500 && !min_position)
  {
    digitalWrite(UP, HIGH);
  }
  else digitalWrite(UP, LOW);
  if(light<200 && !max_position)
  {
    digitalWrite(DOWN, HIGH);
  }
  else digitalWrite(DOWN, LOW);
```

19 ábra. Önműködő üzemmód.

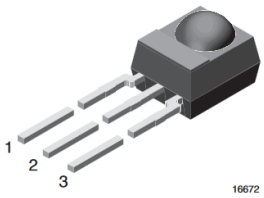
4. Vezetéknélküli működtetés I.

Miután az eszköz teljesen működőképes állapotba került, az utolsó folyamat a vezérlést vezetéknélküli üzemmódra is kiterjeszteni, és ennek köszönhetően akár telefonról vezérelni az eszközt. Ehhez két üzemmód is megalkotásra kerül, az egyik az infrán keresztüli vezérlést teszi lehetővé, a másik pedig wifin keresztül fog történni.

4.1 Kapcsolás tervezése

Ahhoz, hogy infra jelet tudjunk küldeni a panel felé mindössze egy infra vevőre lesz szükség. A *20. ábrán* látható termék adatlap részlet alapján a bekötés:

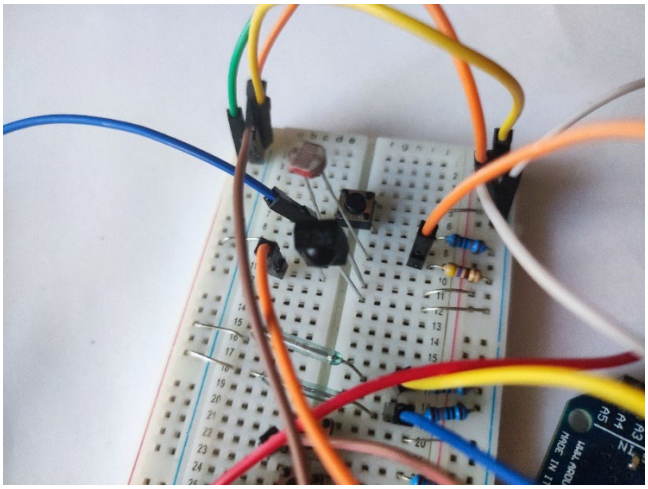
1. 1-es láb az adatláb
2. 2-es láb a 0V
3. 3-as láb a tápfeszültség



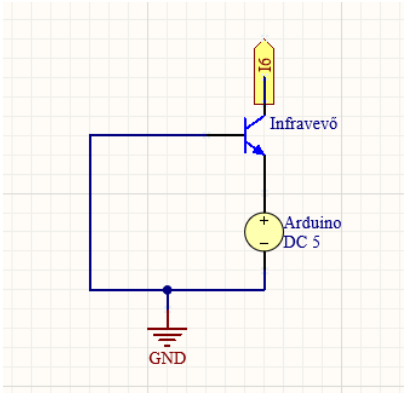
Mechanical Data
Pinning:
1 = OUT, 2 = GND, 3 = V_S

20. ábra. Infravevő adatlapja (forrás [A1]).

Bekötve az alkatrészt azonnal képes is az adatok fogadására. Összeépítve a *21. és 22. ábrán* látható.



21. ábra. Infravevő bekötése.



22. ábra. Infravevő kapcsolási rajza.

4.2 Szoftver

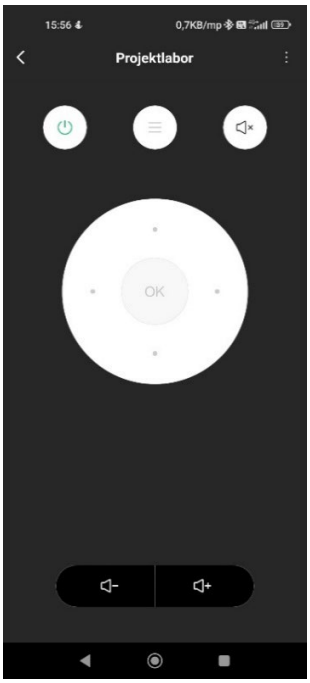
Mielőtt a vevő jelei felhasználásra kerülnének egy tesztprogramot érdemes írni, amellyel meghatározhatóak az értékek, amelyeket az vevő az alaplap felé küld. Ehhez először is szükség van egy könyvtárra, amelyet az Eszközök lista, Könyvtárak kezelése menüpontjában lehet elérni, *IRremote* néven.

Letöltés után elsőként ezt érdemes importálni, valamint a lábkiosztást megadni. Ezt követik a könyvtár specifikus függvények, ahol megadhatjuk honnan várjuk az adatokat, és hova kérjük az eredményeket. A *setu*p függvényen belül jóváhagyjuk az adatok fogadását, majd a *loop* függvényben kezeljük a kapott adatokat. Ennek egy lehetséges módját a *23. ábra* szemlélteti.

```
#include <IRremote.h>

int INFRA = 6;
IRrecv irrecv(INFRA);
decode_results results;

void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();
  irrecv.blink13(true);
}

void loop(){
  if (irrecv.decode(&results)){
        Serial.println(results.value, DEC);
        irrecv.resume();
  }
}
```

23. ábra. Adatok beolvasása tesztre.

Ezután egy infraadóra lesz szükség, amivel utasításokat küldhetünk a vevő felé, lehet ez távirányító, vagy akár egy infra képes telefon, ahogy a *24. ábrán* is látható.



24. ábra. Alkalmazás infra jelek küldésére.

A soros portot figyelve a gombok lenyomása után meg kell, hogy jelenjenek a kódok, mindegyik gomb egy saját azonosítóval rendelkezik, erre példa a *25. ábra*.

```
1095653798
1095653798
1095653798
1095653798
1095653798
17758470
1095651758
1095653798
```

25. ábra. Gombok egyéni azonosítói.

Ezek közül kettőre lesz szükségünk a fel és le irányokhoz, az összes többi gomb nem kerül felvételre, így azok közül bármelyik lenyomása fog „elengedésként” funkcionálni.

A programba illesztés első lépése a szükséges könyvtár importálása, valamint az új kezdeti változok felvétele, a lábkiosztáshoz, illetve a venni kívánt adatokhoz.

A *loop* függvényben először is ki kell egészítenünk az üzemmód kapcsoló feltételünket, hiszen már nem binárisan két állapot közül választhatunk, hanem három üzemmód is rendelkezésünkre áll. Az infra üzemmódban csak jel beolvasáskor történik állapot változás, ezért biztosítanunk kell már itt, hogy egyik üzemmód sem aktív, mielőtt utasítást kapna a rendszer, ennek módját a *26. ábra* foglalja össze.

```
if(mode)
{
  if(mode_change==0) mode_change=1;
  else if(mode_change==1) mode_change=2;
  else mode_change=0;
  digitalWrite(UP, LOW);
  digitalWrite(DOWN, LOW);
  delay(500);
}
```

26. ábra. Üzemmód kiválasztása.

Az infrán belüli üzemmód maga a kézi működtetéshez lesz hasonlatos, egyik gombot lenyomva az egyik irány, másik gombot lenyomva a másik irány lesz aktív. Bármelyik másik gomb lenyomása esetén mindkét irány kikapcsol, valamint véghelyzet elérve is megáll a rendszer.

Ahhoz, hogy két irány ne lehessen aktív egyszerre a korábbiaktól eltérő megoldást kell alkalmazni. A gombok lenyomása egyszerre nem lehetséges, hiszem egyszerre egy jelet fogad a vevő, a problémát itt az jelenti, hogy a leállítást nem a gombok elengedése jelenti, hanem egy újabb utasításnál frissül a vezérlés állapota. Emiatt az egyik irány bekapcsolásával egyidőben a másikat kikapcsolásra kell kényszerítenünk. Összefoglalva a program a *27. ábrán* látható.

```
else
{
    if (irrecv.decode(&results)){
      Serial.println(results.value, DEC);
      if(results.value==1095653798 && !max_position)
      {
        digitalWrite(UP, HIGH);
        digitalWrite(DOWN, LOW);
      }
      else if(results.value==1095651758 && !min_position)
      {
        digitalWrite(DOWN, HIGH);
        digitalWrite(UP, LOW);
      }
      else
      {
        digitalWrite(UP, LOW);
        digitalWrite(DOWN, LOW);
       }
      irrecv.resume();
    }
}
```

27. ábra. Infra üzemmód.

5. Vezeték nélküli működtetés II.

Miután elkészült az infrán keresztüli vezérlés lehetősége, már csak egy utolsó lépés van hátra az eszköz fizikai megépítése előtt, ez pedig a hálózatra kapcsolás, hogy távolról is vezérelhetővé váljon az eszköz. Ehhez szükség lesz egy újabb panelra, amely támogatja a vezetéknélküli kommunikációt, esetünkben ez a már a dolgozat elején is említett Arduino Nano IoT lesz. A végső felépítésben ez az üzemmód fog szerepet kapni, hiszen napjainkban már sokkal inkább ez a vezeték nélküli vezérlési forma kap szerepet.

5.1 Kapcsolás tervezése

A könnyebb kezelhetőség és a későbbi bővíthetőség érdekében szükségessé vált két panel használata a projekt teljesítése érdekében. Ezért tehát az eredeti panel utasításokat fog kapni az új paneltől, amely pedig hálózaton keresztül várja az utasításokat. Ahhoz, hogy két alaplap kommunikálni tudjon egymással össze kell kapcsolni őket valamilyen módon. A projekt során az I2C kommunikációs protokollt fogom használni, és ennek megfelelően kerülnek összekapcsolásra a panelok.

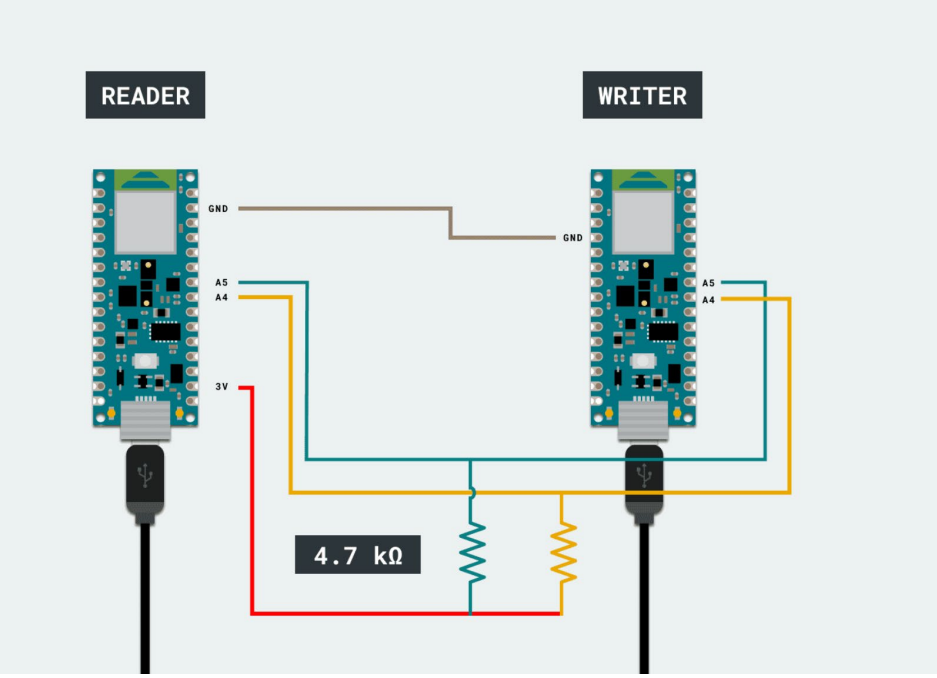
Mielőtt elkezdeném azonban egy kis összefoglalót szeretnék adni a protokollról. Az ilyen jellegű projektekhez többféle protokoll is rendelkezésre áll: UART, CAN, SPI és az I2C. A kommunikáció egyszerűsége miatt nem érdemes túlbonyolítani a kapcsolatot, ezért a legegyszerűbb ezek közül az I2C használata. Az összes közül ez a legegyszerűbb, gyakorlatilag „plug and play” módon használható. A működés során egy master és több slave komponensből állhat a rendszer.

Esetünkben ez egy master és egy slave komponenst jelent. Az összekapcsoláshoz három vezetékre lesz szükségünk. Értelemszerűen egy földvezeték a kezdő lépés. Minden panelen található két dedikált pin, amelyek SCL és SDA névre hallgatnak. Ezek egymással kerülnek összekötésre (SDA SDA-val, SCL SCL-el), úgy hogy ezen felül becsatlakozik hozzájuk egy tápvezeték, természetesen egy megfelelő méretű ellenállás kíséretében, esetünkben ez 4,7kΩ. A két pin szerepe az angol nevükből már könnyen kitalálható, a Serial Clock (SCL) az órajelet biztosítja a két panel között, míg a Serial Data az adatmozgás csatornája. Az összekötött panelokról mutat példát a *28. ábra*.

Fizikálisan tehát készen áll az eszköz a wifin keresztül történő működtetéshez.

5.2 Szoftver

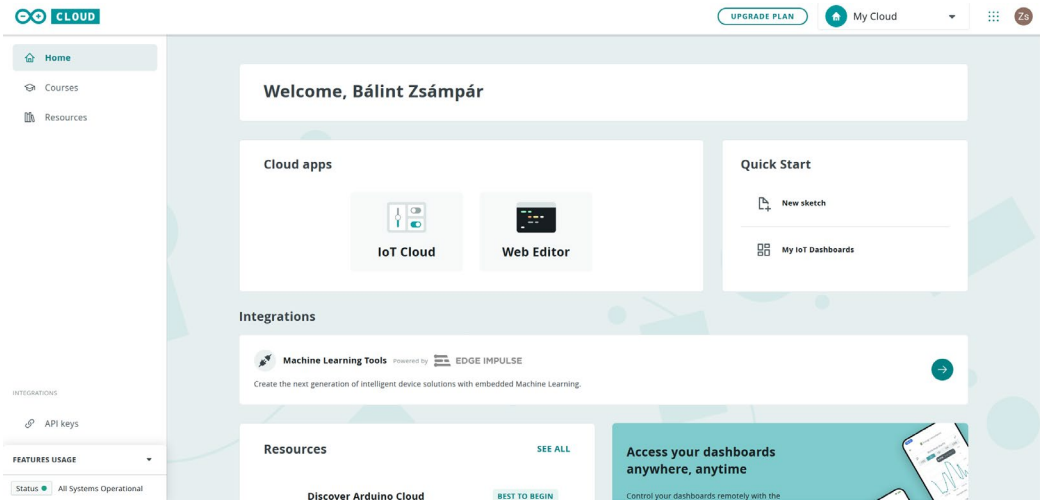
A folytatásban két irány közül lehet választani, az egyszerűbb eset az lenne, ha akkor tudnánk elérni az eszközt, ha ugyanarra a wifi hálózatra vele felcsatlakozva, ez azonban nem igazán felel már meg a modern eszközökkel szemben támasztott követelményeknek. A másik lehetőség tehát az, hogy az eszköz miután felcsatlakozik egy wifi hálózatra, megkeresi a szervert, aminek az adatait megkapta a programban, és azon keresztül távolról is képesek vagyunk elérni és utasításokat adni neki, gyakorlatilag a világ bármelyik pontjáról. Ehhez már rengeteg elérhető szolgáltatást találhatunk az interneten, ilyen például a Blynk, vagy az Arduino Cloud. Mivel Arduino alaplapokon történik a fejlesztés, ezért én az utóbbit választottam.



28. ábra. I2C protokoll. [A2]

Mielőtt a program tényleges megírása történne, fontos végig menni azon, hogyan is kell előkészülni a használatára, így először következzen ez. Az oldal a https://cloud.arduino.cc/home/ linken érhető el. Ahhoz, hogy használni tudjuk, először regisztrálni szükséges. Ennek lépéseit nem írnám le, napjainkban ez már magától értetődő. Az induló képernyő a *29. ábrán* látható. Először a képernyő közepén látható IoT Cloud gombra kell kattintani, majd az almenük közül a Devices menüpontot kiválasztani. Itt lehetséges eszközök megadni, amelyekkel dolgozni szeretnénk, az Add gomb megnyomása után. Első indításkor egy segéd szoftvert szükséges letölteni, ennek segítségével

képes az oldal felismerni a csatlakoztatott panelt. Miután felismerte önállóan konfigurálja a panelt, amely ezt követően használatra kész.

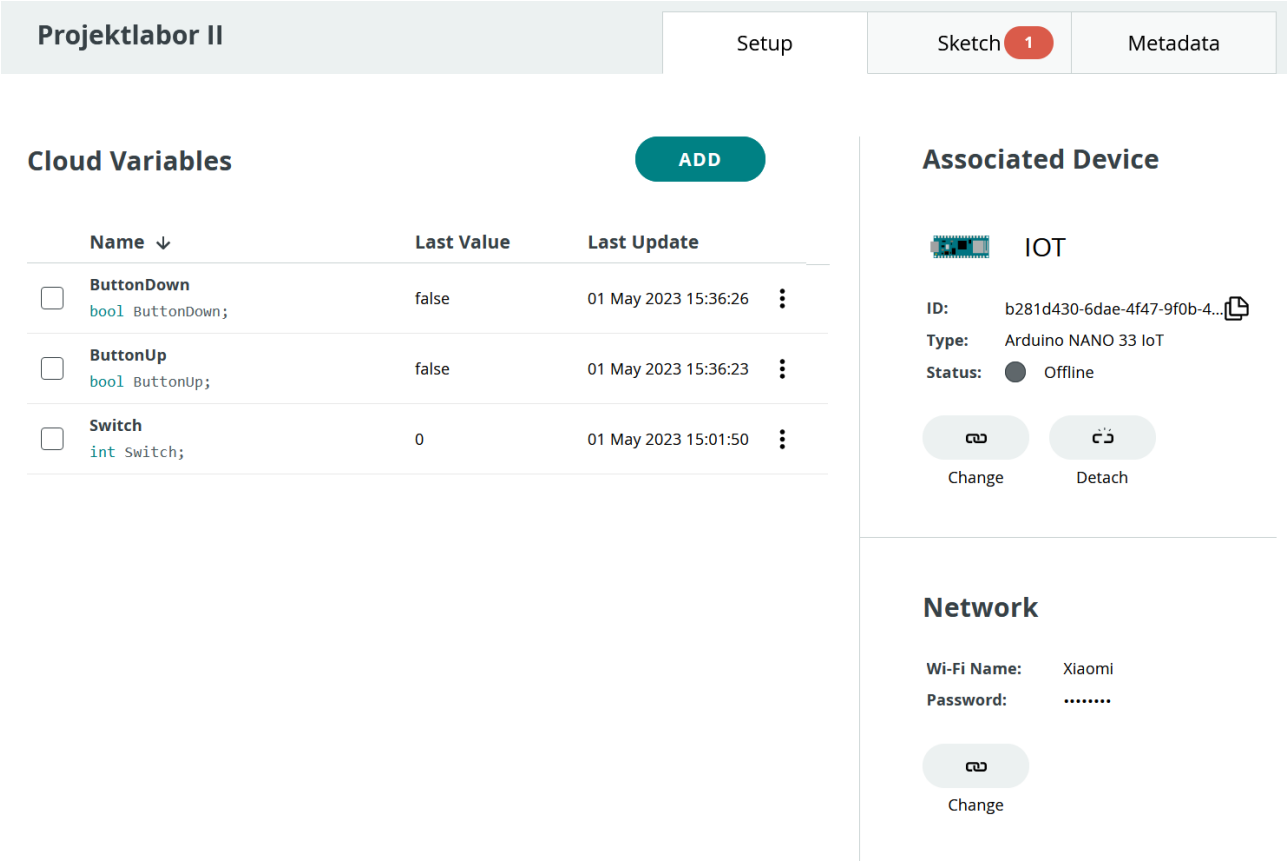


29. ábra. Arduino Cloud.

A következő lépés létrehozni egy projektet, ezt a Things menüpont alatt lehet megtenni a Create gomb megnyomása után. Az oldalon innentől kezdve létrehozható a projekt, amelyet meg szeretnénk alkotni. Megadhatóak változók, amelyeken keresztül kommunikálni szeretnénk az alaplappal, erre ennél a projektnél négyre lesz szükség. A változók tulajdonságai gyakorlatilag megegyeznek a normál fejlesztői környezetben létrehozott változókéival, nevet és típust kell kapjanak. Meg kell adni további az eszközt, amelyet használni szeretnénk a projekthez. Ezen felül megadható a hálózat, amelyre szeretnénk, ha csatlakozna az eszközünk, ez azonban később is megadható, itt nyugodtan elhagyható. Egy összefoglaló látható erről a *30. ábrán*. Utolsó lépés a grafikus felület létrehozása. Ezt a Dashboard menüpont alatt lehet megtenni. Rengeteg különböző kimenet közül választhatunk, ebből a projekthez háromra lesz szükség, amelyet a *31. ábra* mutat. Miután a kimenetek le lettek helyezve, fontos, hogy össze legyenek kapcsolva a hozzájuk tartozó virtuális változókkal, hiszen enélkül nem kerülnek kiküldésre az adatok az alaplaphoz.

Minden készen áll tehát a program megírásához minden, át lehet lépni a Web Editor oldalra, amelyet még a *29. ábrán* lehetett látni. Amennyiben minden jól sikerült létre kellett, hogy jöjjön egy új program a projekt nevével, amely négy fájlból áll. A program megírható ezen a webes felületen keresztül is, azonban mivel eddig az asztali verzióban készült a program, ezt is ott fogom folytatni. A fő programon kívül találunk egyrészt egy általános leírást, ezt hozzátehetjük a programhoz, ha publikálni szeretnénk azt valahol, például Github-on. Ezen felül találunk egy a Cloud által automatikusan generált fájlt, ez a thingProperties.h. Ennek tartalma a *32. ábrán* látható. Ahogy látható a minimális információkat tartalmazza csak, létrehozza a virtuális változókat, és csaklakoztatja a

Cloudhoz az eszközt. A másik generált fájl csak rejtekként szolgál, ide tudjuk beírni a saját wifi hálózatunk nevét és jelszavát, ez látható a *33. ábrán.*



30. ábra. Projekt létrehozása.

31. ábra. A projekt felülete.

Az utolsó lépés a program írása előtt egyrészt a Wire.h könyvtár letöltése, ha alapértelmezetten nem elérhető. Ez az Eszközök, Könyvtárak kezelése menüpont alatt intézhető el. A másik teendő

pedig az alaplap fájlok letöltése. Ezt az Eszközök, Alaplap, Alaplap kezelő menüpont alatt tehetjük meg, a letöltése után már csak ki kell választani az alaplapot, és a portot amelyre csatlakoztatjuk, ennek mikéntje korábban már ismertetésre került.

```
// Code generated by Arduino IoT Cloud, DO NOT EDIT.

#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>

const char SSID[]     = SECRET_SSID;     // Network SSID (name)
const char PASS[]     = SECRET_OPTIONAL_PASS;     // Network password (use for WPA, or use as key for WEP)

void onColorChange();
void onSwitchChange();
void onButtonDownChange();
void onButtonUpChange();

CloudColor color;
int Switch;
bool ButtonDown;
bool ButtonUp;

void initProperties(){

  ArduinoCloud.addProperty(color, READWRITE, ON_CHANGE, onColorChange);
  ArduinoCloud.addProperty(Switch, READWRITE, ON_CHANGE, onSwitchChange);
  ArduinoCloud.addProperty(ButtonDown, READWRITE, ON_CHANGE, onButtonDownChange, 1);
  ArduinoCloud.addProperty(ButtonUp, READWRITE, ON_CHANGE, onButtonUpChange, 1);

}

WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);
```

32. ábra. Arduino segéd fájl.

```
#define SECRET_SSID ""
#define SECRET_OPTIONAL_PASS ""
```

33. ábra. Secret.h fájl.

A program megírása két eszközre szükséges, a slave program a már korábban megírt programon belül fog helyet kapni. A master program egy új program lesz, először ez kerül ismertetésre. Az első lépés a szükséges könyvtárak importálása. Az egyik természetesen a thingProperties, amely a Cloud-dal történő kommunikációhoz szükséges. A másik a Wire, amely az I2C protokoll miatt kell. A setup függvény indítja a folyamatokat, initProperties() inicializálja a virtuális változókat, az ArduinoCloud.begin() indítja a kommunikációt a hálózaton, a debug függvények a folyamat üzeneteit írják ki a soros portra. Az utolsó sor pedig a két alaplap közötti kommunikációt indítja.

A loop függvénynek itt mindössze annyi szerepe van, hogy figyeli az új adatok érkezését. Ezen felül minden változónak van egy saját függvénye, mikor az adott változóban változás áll be, ezek a

függvények kezelik a változással járó feladatokat. Ebben az esetben egy küldést indítanak az új változó állapottal, majd zárják azt. A küldés során valamilyen módon meg kell különböztetni, hogy melyik változó értéke is kerül éppen küldésre. Ennek talán legegyszerűbb módja, ha a változó nevének információját a tízes helyi érték hordozza. Tehát 0 és 1 esetén a felfelé irányra, 10 és 11 esetén a lefelé irányra, 20, 21, 22 esetén az üzemmódra vonatkozik az üzenet tartalma. A teljes program a *34. ábrán* látható.

```
#include "thingProperties.h" //Cloud változók kezeléséhez
#include <Wire.h> //I2C prtokollhoz

void setup() {
  initProperties();
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
  Wire.begin(); //I2C indítás, masterként
}

void loop() {
  ArduinoCloud.update();

}


void onButtonUpChange()  {
  Wire.beginTransmission(0); //Adat küldése
  Wire.write(ButtonUp);
  Wire.endTransmission(); //Küldés zárása
}

void onButtonDownChange()  {
  Wire.beginTransmission(0); //Adat küldése
  Wire.write(ButtonDown+10);
  Wire.endTransmission(); //Küldés zárása
}

void onSwitchChange()  {
  Wire.beginTransmission(0); //Adat küldése
  Wire.write(Switch+20);
  Wire.endTransmission(); //Küldés zárása
}
```

34. ábra. Master program.

A slave vezérlő logikája gyakorlatilag a korábban alkalmazott XOR logika kibővítése lesz, hiszen már nem csak a kézi, hanem a vezetéknélküli jelekre is figyelni kell. Ehhez mindössze annyira van szükség, hogy *vagy* logikával egybe kapcsoljuk a kézi és vezeték nélküli változókat, a korábbi önálló változók helyett. Ezzel lehetővé válik a két üzemmód egyszerre történő változtatása. A programrészlet a *35. ábrán* látható.

```
if(mode_change==0) //Egyes üzemmód, kézi működtetés
{
      if (((pressed_up == HIGH || wifi_up == HIGH) || (pressed_down == HIGH || wifi_down == HIGH)) && !((pressed_up == HIGH || wifi_up == HIGH) && (pressed_down == HIGH || wifi_down == HIGH)))
      {
        if((pressed_up == HIGH || wifi_up == HIGH)&& !max_position) digitalWrite(UP, HIGH);
        if((pressed_down == HIGH || wifi_down == HIGH) && !min_position) digitalWrite(DOWN, HIGH);
      }
      else
      {
        digitalWrite(UP, LOW);
        digitalWrite(DOWN, LOW);
       }
}
```

35. ábra. XOR kibővítve.

A szemantikai hibák felkutatásának és kiküszöbölésének szemléltetése érdekében mutatom be a következő problémát a programmal, az egyszerű kijavítása helyett. A program ebben a formában látszólag jó működést mutat, azonban akad egy probléma, amely a tesztesetek szimulálásakor kerül elő. Ebben az állapotban, amennyiben a program valamelyik irány indítója megnyomásra kerül, az eszköz elindul az egyik irányba. Ezt követően a véghelyzet elérését követően a program kikerül a feltételes állapotából, azonban a leállító feltétel csak a fő feltételen (csakis egy irány aktív) kívül található meg, amely emiatt nem is szakítja meg az eszköz mozgását. A probléma könnyen javítható, azonban jól szemlélteti, hogy még egy látszólag jól működő program is könnyen tartalmazhat kezeletlen eseteket. Hiszen a teszteset során a véghelyzetben lévő eszköz csak a másik véghelyzet irányába volt képes elindult, ez azonban nem kezelte azt a helyzetet, amikor az eszköz egy köztes állapotból érkezik. A javított programot a *36. ábra* tartalmazza.

```
  if((pressed_up == HIGH || wifi_up == HIGH)&& !max_position) digitalWrite(UP, HIGH);
  else if((pressed_down == HIGH || wifi_down == HIGH) && !min_position) digitalWrite(DOWN, HIGH);
  else
  {
    digitalWrite(UP, LOW);
    digitalWrite(DOWN, LOW);
  }
}
else
{
  digitalWrite(UP, LOW);
  digitalWrite(DOWN, LOW);
 }
```

36. ábra. Javított program.

Ezen felül csak annyival szükséges kiegészíteni a programot, hogy egy függvény a master paneltől érkező adatokat képes legyen kezelni, majd azokat továbbítani vezérlésként. Ehhez be kell olvasni az érkező adatokat, majd a megfelelő változó felé továbbítani azt. Ezt foglalja össze a *37. ábra.*

```
void dataArrived(int howMany) {
   int c = Wire.read();
   if(c==0 || c==1) {wifi_up=c;}
   else if(c==10 || c==11) {wifi_down=c-10;}
   else { mode_change=c-20;}
}
```

37. ábra. Adatok kezelése.

6. Fizikai felépítés

Miután elkészült a szükséges vezérlés az eszközhöz, az utolsó lépés megvalósítani azt egy funkcióját betöltő rendszerrel is. Ehhez a *37. ábrán* látható redőnyök vezérlésére használt motort fogom alkalmazni. A fizikai eszköz mivel a hagyományos redőny felépítésén alapszik, ezért csak röviden mutatom be annak elkészítését.

37. ábra. Redőny vezérlő motor.

Az első lépés a motor beépítése a redőny belsejébe. Ehhez a *38. ábrán* látható tengely belsejébe kell helyzeni a motort. Ezután a motor, másik oldalról pedig a tengely vége már behelyezhető a tengely tokjába, majd rászerelhető a redőny, így összeszerelésben nem tér el az eszköz a hagyományos verzió összeszerelésétől.

A kábelezés elvezetése a feladat során nehezen megoldható, hiszen az nem kerül beültetésre, de a valós alkalmazás során ezt a falban szokás elvezetni egy kapcsolóig. Ebben az esetben ez az egyenáramnál is használt relékre csatlakozik, átvéve a LED-ek helyét.

Az utolsó lépés az érzékelők elhelyezése. Ehhez két lehetőség áll rendelkezésre. Egyik lehetőség elegendő hely esetén a sínben elhelyezni az érzékelőket, majd a redőnyre helyezni apró mágneseket. A több hely és jobb hozzáférhetőség miatt azonban érdemes ezeket az érzékelőket inkább

a redőny tokjában elhelyezni és ott rögzíteni őket. A mágneseket ebben az esetben is a redőnyön érdemes elhelyezni.

38. ábra. Redőny tengely.

A fényerősség érzékelését ezzel szemben máshol érdemes elhelyezni, így adja magát a lehetőség, hogy ez a kapcsolókkal egy helyen kerüljön elhelyezésre. A kapcsolókat szintén érdemes lecserélni, majd a teljes kapcsolást egy tokba helyezni, egyrészt zavarvédelmi, másrészt érintésvédelmi szempontok miatt is.

7. Összegzés

A dolgozat tehát összefoglalta egy a hétköznapokban jól hasznosítható vezérlési feladat megvalósítását. A tervezési folyamat kezdetben próbapanelon került összeépítésre, így könnyítva meg a tervezést az első lépésekben. Ezután több funkcióval is kiegészítésre került, melyektől még praktikusabban használhatóbbá vált az elkészült projekt

Ezen felül ismertetésre került a vezetéknélküli kommunikáció megvalósításának egy-egy formája, infrás és wifis formában is. Hasznos ismeret továbbá az alaplapok kommunikációjának ismerete, ezzel is szélesítve a megvalósítható feladatok körét.

Ebben a formában már jól kihasználható redőnyt kapunk eredményűl, azonban ez esetleg még tovább bővíthető, például beállított időre történő mozgással, esőérzékeléssel, vagy esetleg az ablakok mozgatásával, amelyekkel már kifejezetten komplex rendszer készíthető.

A projekt azonban így is bemutatta, hogy néhány alap eszköz segítségével milyen, a hétköznapokban is jól hasznosítható rendszer készíthető.

Irodalomjegyzék

[1] *Tandon School of Engineering:*

http://engineering.nyu.edu/gk12/amps-cbri/pdf/ArduinoBooks/Arduino%20Programming%20Notebook.pdf, 2022. 09. 19.

Ábrajegyzék

[A1] *https://www.hestore.hu/prod_getfile.php?id=10611* (utolsó látogatás időpontja: 2022. 11. 25.)

[A2] https://docs.arduino.cc/static/91f13065803647ec2ff99746de37ddd6/29114/nano33BS_06_illustration.png (utolsó látogatás időpontja: 2023. 02. 21.)

[A3] https://docs.arduino.cc/static/c18e027f826663ba9f16ffd94b60500f/2f891/pinout.png

[A4] https://docs.arduino.cc/static/e62e15c564ec48ac82db2d311d20fb1a/A000057-pinout.png

Mellékletek

1. számú melléklet, alapszoftver infra vezérléssel

```
#include <IRremote.h> //Könyvtár az infra kezeléséhez

int BUTTON_UP=9; //Felfelé nyomógomb állapot érzékelése
int BUTTON_DOWN=8; //Lefelé nyomógomb állapot érzékelése
int UP=5; //Felfelé utasítás
int DOWN=12; //Lefelé utasítás
int MAX=10; //Max pozíció elérés figyelése
int MIN=11; //Min pozíció elérés figyelése
int SWITCH=7; //Üzemmód váltás
int INFRA = 6; //Infra jelek fogadása
int LIGHT=A0; //Analóg láb a fotorezisztív szenzorhoz
int mode_change=0; //Aktuális üzemmód

IRrecv irrecv(INFRA);
decode_results results;


void setup() {
  pinMode(BUTTON_UP, INPUT); //Felfelé nyomógomb állapot érzékelése
  pinMode(BUTTON_DOWN, INPUT); //Lefelé nyomógomb állapot érzékelése
  pinMode(UP, OUTPUT); //Felfelé utasítás
  pinMode(DOWN, OUTPUT); //Lefelé utasítás
  pinMode(MAX, INPUT); //Max pozíció elérés figyelése
  pinMode(MIN, INPUT); //Min pozíció elérés figyelése
  pinMode(SWITCH,INPUT); //Üzemmód váltás
  pinMode(LIGHT,INPUT); //Analóg láb a fotorezisztív szenzorhoz
  pinMode(INFRA,INPUT); //Infra jelek fogadása
  int pressed_up=0; //Felfelé nyomógomb állapot érzékelése
  int pressed_down=0; //Lefelé nyomógomb állapot érzékelése
  int max_position=0; //Max pozíció elérés figyelése
  int min_position=0; //Min pozíció elérés figyelése
  irrecv.enableIRIn();
  irrecv.blink13(true);
}

void loop() {
  int pressed_up=digitalRead(BUTTON_UP); //Felfelé utasítás olvasása
  int pressed_down=digitalRead(BUTTON_DOWN); //Lefelé utasítás olvasása
  int max_position=digitalRead(MAX); //Max pozíció elérve olvasása
  int min_position=digitalRead(MIN); //Min pozíció elérve olvasása
  int mode=digitalRead(SWITCH); //Üzemmód váltás olvasása
  float light=analogRead(LIGHT); //Fénymennyiség olvasása
  if(mode) //Ha mode gomb nyomva egyet lép előre, kimeneteket kikapcsolja
  {
    if(mode_change==0) mode_change=1;
```

```
  else if(mode_change==1) mode_change=2;
  else mode_change=0;
  digitalWrite(UP, LOW);
  digitalWrite(DOWN, LOW);
  delay(500);
 }
 if(mode_change==0) //Egyes üzemmód, kézi működtetés
 {
     if (((pressed_up == LOW) || (pressed_down == LOW)) && !((pressed_up == LOW)
&& (pressed_down == LOW))) //XOR kettő kimenet bekapcsolásának kizárásához
     {
      if(pressed_up && !max_position) digitalWrite(UP, HIGH);
      if(pressed_down && !min_position) digitalWrite(DOWN, HIGH);
     }
     else
     {
      digitalWrite(UP, LOW);
      digitalWrite(DOWN, LOW);
     }
 }
 else if(mode_change==1) //Kettes üzemmód, automatikus működtetés
 {
  if(light>500 && !min_position) //Érték felett megpróbálja csökkenteni a fénymennyiséget
  {
   digitalWrite(UP, HIGH);
  }
  else digitalWrite(UP, LOW);
  if(light<100 && !max_position) //Érték alatt megpróbálja növelni a fénymennyiséget
  {
   digitalWrite(DOWN, HIGH);
  }
  else digitalWrite(DOWN, LOW);
 }
 else //Hármas üzemmód, infrás működtetés
 {
    if (irrecv.decode(&results)){
     if(results.value==1095653798 && !max_position) //Felfelé mozgatás
     {
      digitalWrite(UP, HIGH);
      digitalWrite(DOWN, LOW);
     }
     else if(results.value==1095651758 && !min_position) //Lefelé mozgatás
     {
      digitalWrite(DOWN, HIGH);
      digitalWrite(UP, LOW);
     }
     else //Más gombnál kikapcsolás
     {
      digitalWrite(UP, LOW);
```

```
        digitalWrite(DOWN, LOW);
       }
      irrecv.resume();
    }
  }
}
```

2. számú melléklet, tesztszoftver infra vezérléssel

```
#include <IRremote.h>

int INFRA = 6;
IRrecv irrecv(INFRA);
decode_results results;

void setup(){
  Serial.begin(9600);
  irrecv.enableIRIn();
  irrecv.blink13(true);
}

void loop(){
  if (irrecv.decode(&results)){
        Serial.println(results.value, DEC);
        irrecv.resume();
  }
}
```

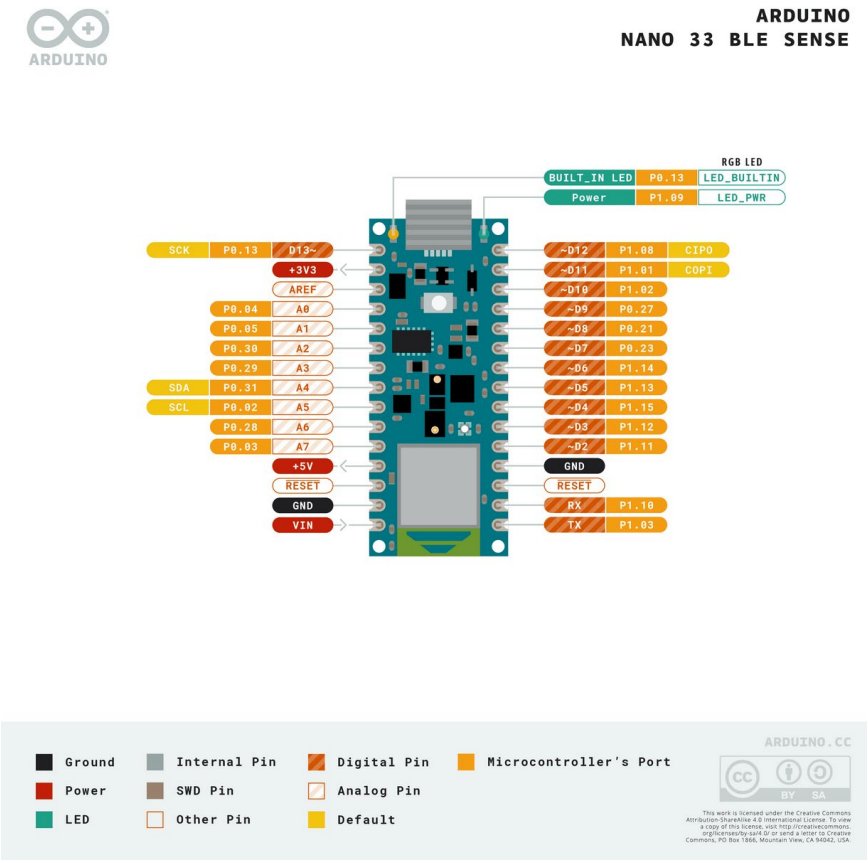
3. számú melléklet, alapszoftver infra vezérléssel

```
#include <Wire.h> //I2C protokoll használatához

int BUTTON_UP=9; //Felfelé nyomógomb állapot érzékelése
int BUTTON_DOWN=8; //Lefelé nyomógomb állapot érzékelése
int UP=5; //Felfelé utasítás
int DOWN=12; //Lefelé utasítás
int MAX=10; //Max pozíció elérés figyelése
int MIN=11; //Min pozíció elérés figyelése
int SWITCH=7; //Üzemmód váltás
int LIGHT=A0; //Analóg láb a fotorezisztív szenzorhoz
int mode_change=0; //Aktuális üzemmód
int wifi_up=0;
int wifi_down=0;

void setup() {
  pinMode(BUTTON_UP, INPUT); //Felfelé nyomógomb állapot érzékelése
  pinMode(BUTTON_DOWN, INPUT); //Lefelé nyomógomb állapot érzékelése
  pinMode(UP, OUTPUT); //Felfelé utasítás
  pinMode(DOWN, OUTPUT); //Lefelé utasítás
  pinMode(MAX, INPUT); //Max pozíció elérés figyelése
  pinMode(MIN, INPUT); //Min pozíció elérés figyelése
  pinMode(SWITCH,INPUT); //Üzemmód váltás
  pinMode(LIGHT,INPUT); //Analóg láb a fotorezisztív szenzorhoz
  int pressed_up=0; //Felfelé nyomógomb állapot érzékelése
  int pressed_down=0; //Lefelé nyomógomb állapot érzékelése
  int max_position=0; //Max pozíció elérés figyelése
  int min_position=0; //Min pozíció elérés figyelése
  Wire.begin(0);  // I2C indítás, slave 0 néven
```

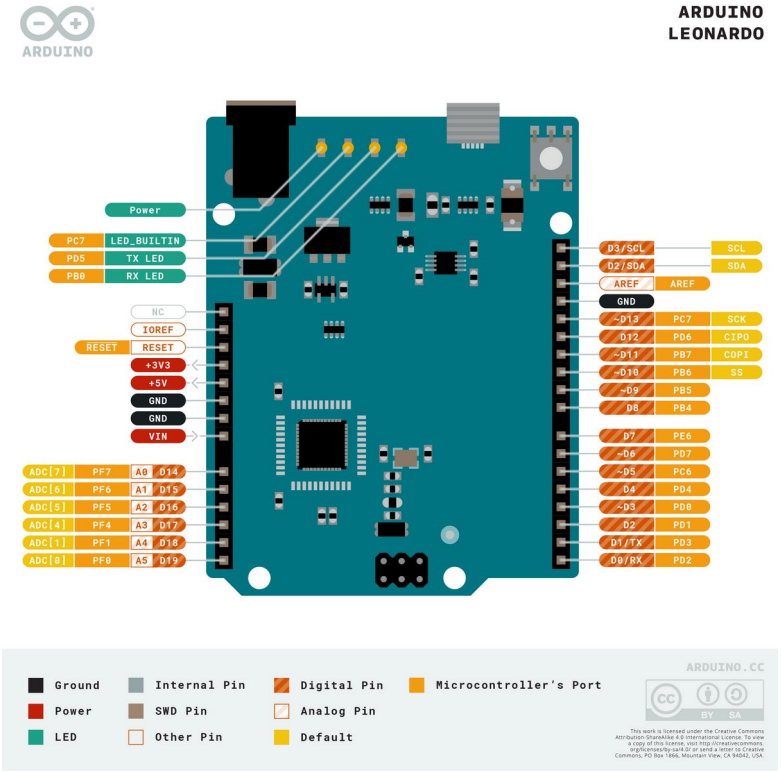
```
}

void loop() {
 int pressed_up=digitalRead(BUTTON_UP); //Felfelé utasítás olvasása
 int pressed_down=digitalRead(BUTTON_DOWN); //Lefelé utasítás olvasása
 int max_position=digitalRead(MAX); //Max pozíció elérve olvasása
 int min_position=digitalRead(MIN); //Min pozíció elérve olvasása
 int mode=digitalRead(SWITCH); //Üzemmód váltás olvasása
 float light=analogRead(LIGHT); //Fénymennyiség olvasása
 Wire.onReceive(dataArrived); //Adat érkezésre adott függvényt hívja
 if(mode_change==0) //Egyes üzemmód, kézi működtetés
 {
     if (((pressed_up == HIGH || wifi_up == HIGH) || (pressed_down == HIGH ||
wifi_down == HIGH)) && !((pressed_up == HIGH || wifi_up == HIGH) &&
(pressed_down == HIGH || wifi_down == HIGH))) //XOR kettő kimenet bekapcsolásának
kizárásához
     {
      if((pressed_up == HIGH || wifi_up == HIGH)&& !max_position) digitalWrite(UP,
HIGH);
      else if((pressed_down == HIGH || wifi_down == HIGH) && !min_position)
digitalWrite(DOWN, HIGH);
      else
      {
       digitalWrite(UP, LOW);
       digitalWrite(DOWN, LOW);
      }
     }
     else
     {
      digitalWrite(UP, LOW);
      digitalWrite(DOWN, LOW);
     }
```

```
  }
  else if(mode_change==1) //Kettes üzemmód, automatikus működtetés
  {
   if(light>500 && !min_position) //Érték felett megpróbálja csökkenteni a fénymennyiséget
   {
    digitalWrite(UP, HIGH);
   }
   else digitalWrite(UP, LOW);
   if(light<100 && !max_position) //Érték alatt megpróbálja növelni a fénymennyiséget
   {
    digitalWrite(DOWN, HIGH);
   }
   else digitalWrite(DOWN, LOW);
  }
}

void dataArrived(int howMany) {
  int c = Wire.read();
  if(c==0 || c==1) {wifi_up=c;}
  else if(c==10 || c==11) {wifi_down=c-10;}
  else { mode_change=c-20;}
}
```

4. számú melléklet, lábkiosztások



Arduino Nano IoT lábkiosztása [A3]



Arduino Leonardo lábkiosztása [A4]