ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZET
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK

SZAKDOLGOZAT

Mészáros Máté György
T006686/FI12904/K

OE-KVK
2022.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar
Műszertechnikai és Automatizálási Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Mészáros Máté György
Szakdolgozat száma: SZD21060114212650
Törzskönyvi száma: T006686/FI12904/K **Neptun kódja:** [redacted]

Szak: villamosmérnöki (magyar nyelven) (Budapest)
Specializáció: Műszer-automatika specializáció
Számítógépes folyamatautomatizálás modul
Automatizált gyártórendszerek modul
Elektronikai - Orvostechnikai műszerek és tesztelés modul
Felügyeleti informatika és elektronikus vagyonvédelem modul

A dolgozat címe: Arduino alapú, programozható eljárástechnikai szakasz-szimulátor fejlesztése PLC-alapú szabályzások oktatásához.
A dolgozat címe angolul: Plant simulator developement with Arduino for PLC education
A feladat részletezése: Programozható szakasz-szimulátor készítése szabályzások PLC-n történő oktatásának segítésére.

Intézményi konzulens neve: Varga Árpád

A kiadott téma elévülési határideje: 2024. június 1.
Beadási határidő: 2022. 05. 15.

A szakdolgozat: Nem titkos.

Kiadva: Budapest, 2022. 03. 21.

Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar Elektronikai és Kommunikációs Rendszerek Intézet

Intézetigazgató

A dolgozatot beadásra alkalmasnak találom:

Varga Árpád
belső konzulens

külső konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR
ELEKTRONIKAI ÉS KOMMUNIKÁCIÓS RENDSZEREK INTÉZET
MŰSZERTECHNIKAI ÉS AUTOMATIZÁLÁSI TANSZÉK

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja, a titkosításra vonatkozó esetleges megkötések mellett.

Budapest, 2022. 05. 13.

Mészáros Máté

hallgató aláírása

Arduino alapú, programozható eljárástechnikai szakasz-szimulátor fejlesztése PLC-alapú szabályzások oktatásához

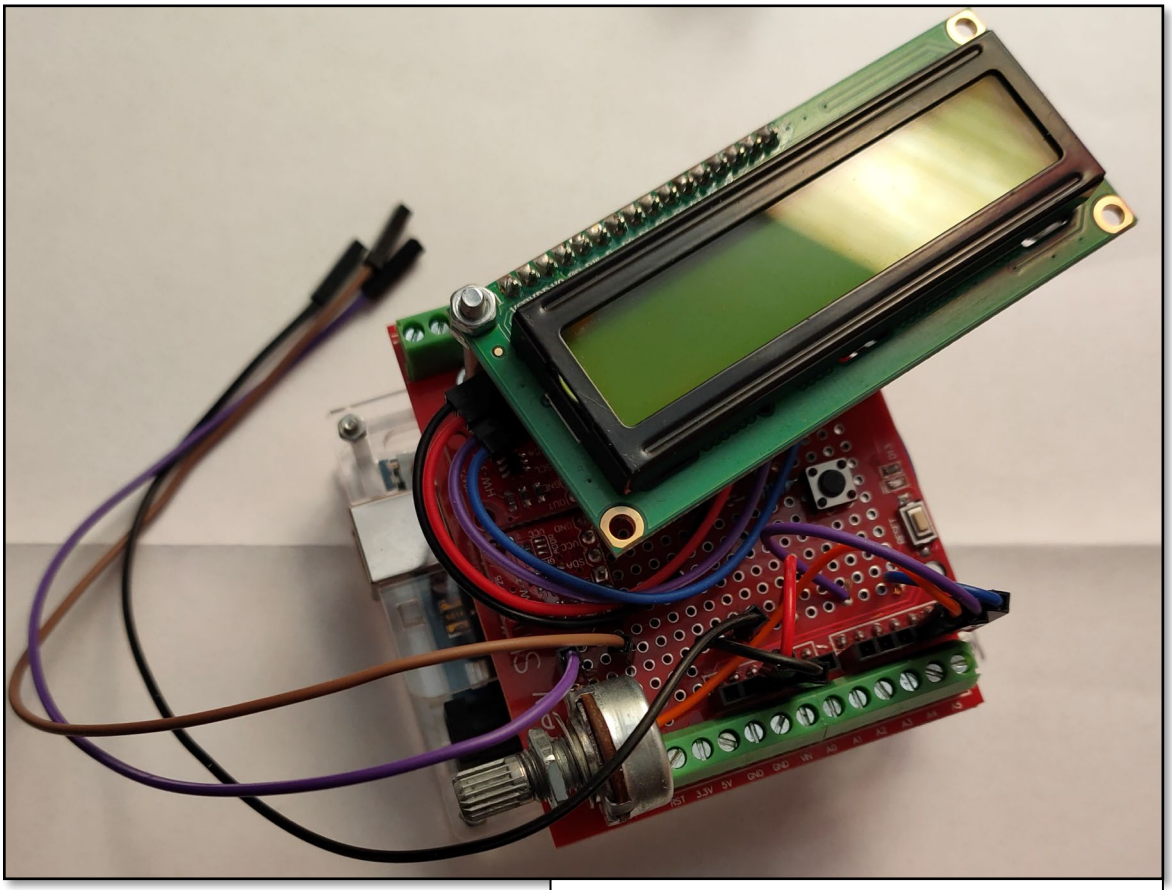
Tartalomjegyzék:

1. BEVEZETŐ

A szakdolgozat célja egy olyan eszköz létrehozása, amely oktatási célra lesz alkalmazható. A használat során a diákok azt tapasztalhatják meg, hogy milyen is az, amikor egy valós külső eszközt szabályoznak PLC (programmable logic controller) segítségével.

Egy korábbi szakdolgozatban Raspberry Pi 3 típusú kártyára fejlesztett szakasz-szimulátor valósult meg Szilágyi Zsombor által. A Raspberry azonban nem annyira tekinthető valós idejű rendszernek, mint egy Arduino. Emiatt nem sikerült mindent kivitelezni a megvalósítás alatt. Én egy Arduino UNO-t (1. ábra) választottam az eljárástechnikai szakasz szimulátor alapjának.

Létrehoztam egy Windows form alkalmazást, ami az Arduino grafikus felhasználói felülete. A GUI (Grafical User Interface) arra hivatott, hogy általa a szimulált szakasz összetételét és paraméterezését kalibrálni lehessen. Ezen kívül visszajelzést ad arról, hogyan viselkedik a szakasz kimenete négyszögjel bemenetre.



1. Ábra: A megvalósított szakasz szimulátor

2. VALÓS IDEJŰ RENDSZER [1]

Egy rendszer akkor valós idejű, ha meghatározott időn belül reagál a környezetből beérkező jelre. Ezt a meghatározott időt nevezik válaszidőnek. A válaszidő a PLC (Programozható Logikai Vezérlő) esetén mikroszekundum nagyságú, míg az elkészült szakasz szimulátor válaszideje ennek a sokszorosa, 15-16 milliszekundum. A valós idejű rendszereknek 2 csoportja van:

- kemény valós idejű rendszer
- lágy valós idejű rendszer

2.1. Hard real-time

A kemény valós idejű (HRT) csoportba tartozó rendszereknek mindenképpen reagálnia kell a külvilágból érkező jelekre, különben katasztrófa következik be.

2.2. Soft real-time

A lágy valós idejű (SRT) csoportba tartozó rendszerek esetén megengedett a válaszidő túllépése, a reakció eredményes lehet az időkorláton kívül is.

3. PROGRAMOZHATÓ LOGIKAI VEZÉRLŐ [2]

Programozható Logikai Vezérlő (2. ábra) egy olyan folyamat irányítási rendszer, ami kapcsolatot létesít a fizikai szakasszal érzékelők és beavatkozók által. A PLC beolvassa a vezérlő motorok, szelepek és egyéb beavatkozók aktuális értékét a folyamatban. Az iparban elengedhetetlen az automatizálás. A PLC erre nyújt valós idejű megoldást. A beérkező jelek alapján kontrollálja a beavatkozókat, ami a folyamat irányítását jelenti.

2. Ábra: Programozható Logikai Vezérlő

4. ARDUINO [3]

Az Arduino egy nyílt forráskódú hardver és szoftver gyártó cég. Az Arduino lap használata előnyös választás a projektekben nyílt forráskódja és kedvező ára miatt. Az IDII (Interaction Design Institute Ivrea) olasz cég adta ki az első lapot még 2005-ben. Ez az első verzió diákok számára készült, a lap fejlesztésénél a programozás könnyed elsajátítására és alacsony költségvetésre törekedtek. 2016-ra már 17 különböző féle hivatalos hardver volt elérhető.

A legtöbb lap Atmel 8-bites AVR mikrokontrollerrel rendelkezik. Ezek a kontrollerek különböző méretű flash memóriával és pin számmal vannak ellátva. A lapok egysoros vagy többsoros kivezetésekkel rendelkeznek, abból a célból, hogy különböző érzékelőkkel, kijelzőkkel vagy más lapokkal kommunikáljon. Ezeket a kapcsolatokat kiegészítő modulokkal, kevesebb pin felhasználásával, shield bekötésével lehet célszerűsíteni. A shieldek I^2C soros busszal külön címezhetők. A legtöbb lap verzió 5 voltos egyenfeszültség szabályozó és 16 MHz kristály oszcillátort tartalmaz. Néhány verzió rendelkezik jelszint erősítővel, amivel a TTL (Transistor–transistor logic) jelet RS-232-re konvertálja. Kiegészítőként is beszerezhető ez az IC. A mostani Arduino modellek USB-n keresztül programozhatóak a lapon beágyazott USB-to-serial adapter chip-nek köszönhetően.

A szimulátorhoz használt Arduino UNO-n 14 digitális ki/be pin található, melyek közül 6 lehet PWM (pulse width modulation) is, további 6 darab analóg inputtal. Ezek az analóg pinek ebben

a formában csak bemenetként használhatóak. A szakasz szimulátor megfelelő működéséhez analóg kimenere van szükség, ezért két darab kiegészítő DAC IC-t (Digital Analog Converter Integrated Circuit) építettem be a kapcsolásba.

5. KOMMUNIKÁCIÓS PROTOKOLLOK ISMERTETÉSE

A szakasz szimulátor működése és felkalibrálása alatt számos kommunikációs folyamatot hajt végre. Az Arduino board digitális kimenetein keresztül kommunikál a LCD (Liquid-crystal display) panellal a DAC kiegészítő lappal és a számítógépen futtatott grafikai felhasználói felülettel. Az analóg bementeken, fogadja a beérkező jeleket a PLC felől, illetve a GUI felől.

5.1. I^2C bus [4]

Az I^2C busz (Inter-Integrated Circuit) rendszert a Philips fejlesztette ki 1982-ben. A cél az volt, hogy kiaknázzák a sok hasonlóságot a hétköznapi elektronikában a távközlésben és az ipari technológiában. Az I^2C egy több mester - több szolga kapcsolatot biztositó rendszer. Csak két buszvezeték szükséges a kommunikáció megvalósításához. Az egyik a soros adatvonal (SDA), a másik a soros órajel (SCL). A buszrendszerhez csatlakoztatott eszközök mindegyike az egyedi címmel és az egyszerű mester-szolga kapcsolat segítségével kommunikál. A mester működhet adóként és vevőként és a mester feladata előállítani az óra jelet. Valós két mesteres busz esetén ütközés detektálás és arbitráció felel az üzenetvesztés elhárításáért. Soros, 8 bites, kétirányú adatforgalom valósítható meg, melynek normál üzenetküldési sebessége 100 kbit/s. Az üzemmód változtatható gyorsra, ebben az esetben 400 kbit/s az adatátvitel.

A megvalósított szakasz szimulátorban az I^2C buszt használom arra, hogy az Arduino board kommunikáljon a LED kijelzővel és a 2 darab digitális analóg átalakítóval. Az előbbiek alapján a buszon van 3 darab szolga és egy mester, ezt a szerepet az Arduino tölti be. A két DAC-nak különböző címet kellett beállítanom a kommunikáció megfelelő működése érdekében. A lap tetején az A0-t összekötöttem a GND-vel, így az alapértelmezett 0x61-es cím helyett 0x62-es címet kapta meg.

5.2. Universal Serial Bus [5]

Az USB (Universal Serial Bus) magyarul univerzális soros busz a számítógépes rendszerek egyik legelterjedtebb csatlakozója. A hat cég által alapított USB Implementers Forum, Inc.

fejlesztette ki 1994-ben. Minden jelenleg elterjedt modern operációs rendszer támogatja ezt a fajta kommunikációs csatornát. Az évek alatt rengeteget fejlődött a sebessége, a legfrissebb generációs szabvány végsebesége fizikai rétegben 10 Gbps, míg valós alkalmazásban 400 Mb/s. Az USB kedvező tulajdonsága, hogy osztható, ami USB hub-al oldható meg.

A megvalósított szakasz szimulátor a PC-vel a felkonfigurálás alatt USB-n keresztül kommunikál. Az Arduinon USB-B fajta csatlakozó van (3. ábra). Az Arduino a kommunikáción kívül itt kapja a működéshez szükséges feszültséget is.

3. Ábra: USB-B fajta csatlakozó

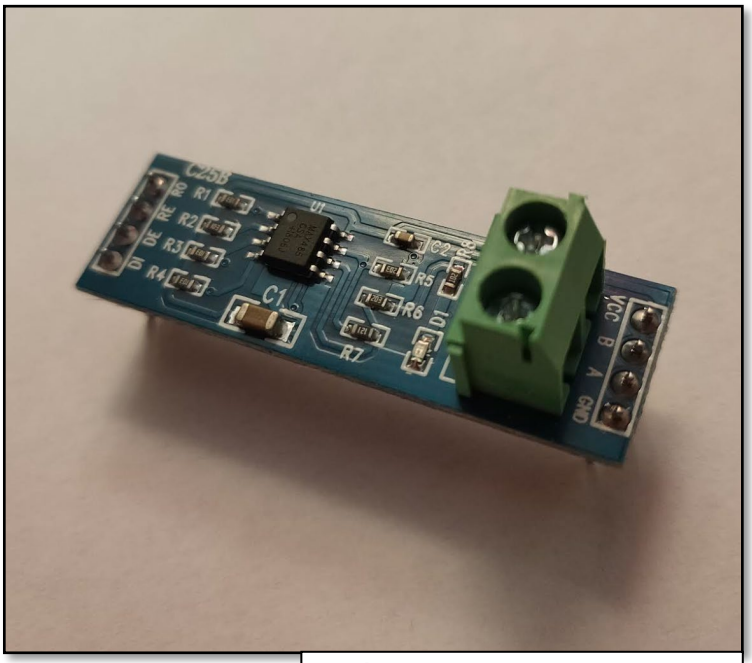
5.3. Serial Peripheral Interface [6]

Az SPI (Serial Peripheral Interface), magyarul szinkron soros kommunikációs interfész, elsősorban rövid távú kommunikációhoz használják. Az interfészt a Motorola fejlesztette ki 1979-ben. Ez egy teljes dupla kommunikáció, mester-szolga architektúra, amely csak egy mestert tartalmazhat. A legfrissebb specifikáció alapján 3,4 Mbps a végsebessége és 8 m a maximális távolsága.

5.4. RS-232 bus [7]

A nemzetközi távközlési bizottság 1960-ban fogalmazta meg először az RS-232 szabvány feladatait és elektromos jellemzőit. Napjainkban az USB helyettesíti az RS-232 szerepét utóbbi elavultsága miatt. Az RS-232C szabványt az EIA adta ki 1969-ben. Ebben a kiadásban már meg lett határozva a jelek karakterisztikája és az univerzális aszinkron adatátviteli jelleg. A jelátvitel maximális távolsága 15 méter, ez ma már elavultnak számít. Ezt a távolságot egy RS-232-RS-485 átalakítóval azonban ki lehet tolni egészen 1200 méterre.

Ahhoz, hogy az Arduino az RS-485-tel kommunikálni tudjon, szükség van a TTL-RS-485 átalakítóra (4. ábra), amely a jelszintek közötti eltérést oldja meg. Az Arduino TTL az 0-5 voltos feszültségű míg az RS-485 feszültségszintje 0-15 volt közötti tartományba esik.



4. Ábra: TTL-RS-485 átalakító

5.5. Profibus [1]

A Profibus (Process Field Bus) egy univerzális kommunikációs protokoll. A BMBF (Bundesministerium für Bildung und Forschung – Képzési és Fejlesztési Minisztérium) fejlesztette ki 1989-ben. Manapság két változattát használjuk. A legelterjedtebb a Profibus-DP (Dezentrale Peripherie) és az applikáció specifikus Profibus PA (Process Automation). Az utóbbi években a Profibust annak fejlesztebb változata a Profinet váltja le.

5.5.1. Profibus DP

A Decentralizált Perifériát (DP) arra arra a célra fejlesztették ki, hogy egy központosított kontroller kommunikáljon az érzékelőkkel és a beavatkozókkal. A legtöbb diagnosztikai lehetőség alapvetően itt végezhető el.

5.5.2. Profibus PA

A Profibusz Folyamat Automatizálási (PA) változatát ara használjuk, hogy monitorozni tudjuk a mérőeszközöket a folyamatirányítási applikációkban. Ezt a változatot veszélyes környezetben is fel lehet használni. A kábelen nem csak információ áramlik, hanem a végpontba

csatlakoztatott eszköz áramforrása is. A minimalizált energiaszint miatt nem történhet semmilyen katasztrófa meghibásodás esetében sem.

Eredetileg az Arduino és PLC közötti kommunikáció Profibusszal valósult volna meg, de mivel az egyetemi laborban egyetlen PLC van, ami rendelkezik a Profibus RS-484 bővítőkártyával ezért így analóg kommunikáció korlátozott jel szintjével valósult meg.

6. FEJLESZTŐ SZOFTVEREK

A szakasz szimulátor megvalósításához és teszteléséhez 3 fő programozó szoftvert használtam. A board programozásához a saját fejlesztő környezetét használtam, ami az Arduino IDE (Integrated Development Environment). A tesztelés során használt PLC programozásához a Siemens TIA (Totally Integrated Automation) szoftverét használtam. A tanári kalibrációs PC szoftver megírásához pedig a Microsoft Visual Studio szoftvert választottam.

6.1. Arduino IDE [3] [8]

Az Arduino IDE egy keresztplatformos fejlesztő környezet, amit Java programnyelven írtak. Kód szerkesztőt tartalmaz, ami támogatja a szöveg kivágást, a formázást, a keresést a felülírást, az automatikus azonosítást, a kapcsos zárójel beazonosítást és a mondatkiemelést. Az egyszerű egy nyomógombos fordítás és feltöltés is biztosítva van az Arduino boardra.

Az Arduino IDE támogatja a C és C++ nyelveket. A fejlesztő környezet szolgáltatja szoftveres könyvtárakat kezdve a huzalozási projekttel, ami lehetővé teszi a gyakori kimeneti és bemeneti folyamatok programozását.

2019-ben kiadták a Pro IDE változatát, amelyet tovább fejlesztve 2021. márciusában kiadták IDE 2.0-ként. Ez a verzió egy felújított modern fejlesztői környezet biztosit. Új board menedzsert és könyvtár menedzsert kapott. Megjelent a Serial Monitor, ez sokat segített a hibakeresésben. Ezen kívül a Dark mode is bekerült ebbe a fejlesztett verzióba.

6.2. Siemens TIA [2]

A Siemens TIA portál az előző szoftverrel ellentétben már nem ingyenes. A TIA több funkciót kínál a felhasználó számára. Egységes platformot nyújt az összes automatizálási feladathoz, ahol PLC, HMI (Human Machine Interface) és SCADA-rendszer (Supervisory control and data acquisition) programozása is lehetséges. A programozási nyelvek széles kínálatából lehet

választani. A két szöveges programozási nyelv az STL (Statement List) és az SCL (Structured Control Language). Használható még a létradiagram és a funkcióblokk diagram.

A tesztelések során a vizualizáció szerkesztőt, programszerkesztőt, szimulációt használtam. Az adatokat a laborban lévő HMI-n kisértem figyelemmel idődiagram segítségével. Az adatokat a PLC ciklus időnként kiírta DB-be (Data Block).

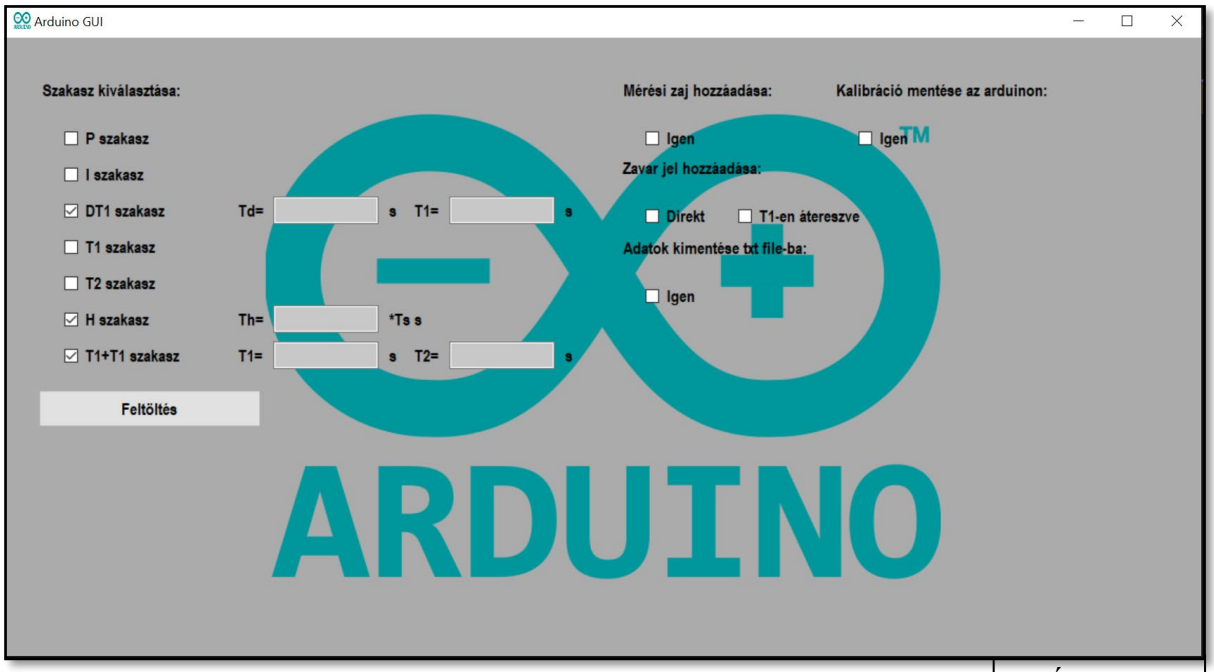
6.3. Microsoft Visual Studio [9]

A Microsoft Visual Studio a Microsoft által kiadott IDE. Ez a fejlesztő környezet asztali program, weboldal, webes applikáció és mobil applikáció fejlesztésére használható. Jelenleg 18 különböző programozási nyelven lehet programozni ebben a fejlesztői környezetben.

A grafikai felhasználói felületet C# nyelven, Windows form applikációként hoztam létre, így a tanári gépen elég, ha fent van az .exe fájl.

7. SZIMULÁLT SZAKASZ ÖSSZETÉTELE

A szimulált szakaszt előzetesen a tanári gépen GUI (5. ábra) segítségével fel lehet konfigurálni, mielőtt a diákok kézhez kapják a szakasz szimulátort. A GUI lehetőséget kínál szinte bármilyen valóságban megtalálható szakasz beállítására. A szakaszok alaptagokból állnak össze. A P (Proporcionális), az I (Integráló), a D (Differenciáló), a T (Egytárolós), a H (Holtidős) és a T2 (Kéttárolós) alaptagok bármely variációjú összetételét be lehet állítani az alaptagok időállandóinak század pontosságú értékével. Választható még T1+T1 két egytárolós összetett tag és DT1 Differenciáló egytárolós összetett tag, így összesen 4 egytárolós tagot lehet berakni a szimulált szakasz összetételébe. A szimulált szakasz állhat egy egyszerű T1-ből, vagy akár HT2T3-ból is összesen 120 db különböző összetételű szakaszt lehet szimulálni.



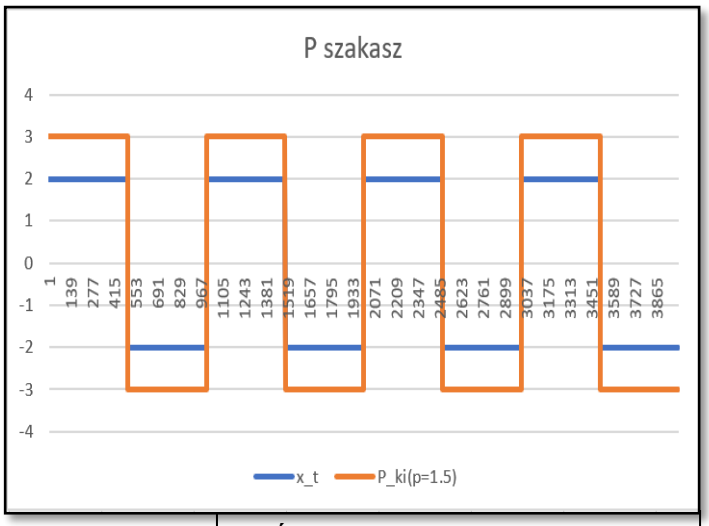
5. Ábra: GUI

7.1. P alaptag

A Proporcionális, avagy arányos alaptag kimenete (6.ábra) arányos a bemenetére érkező jellel. A bemenő jelet K_p-val szorozzuk.

Proporcionális alaptag kimeneti egyenlete: $y(t) = K_\rho \cdot x(t)$

A proporcionális tagra példa az ideális erősítő.



6. Ábra: arányos alaptag kimenete

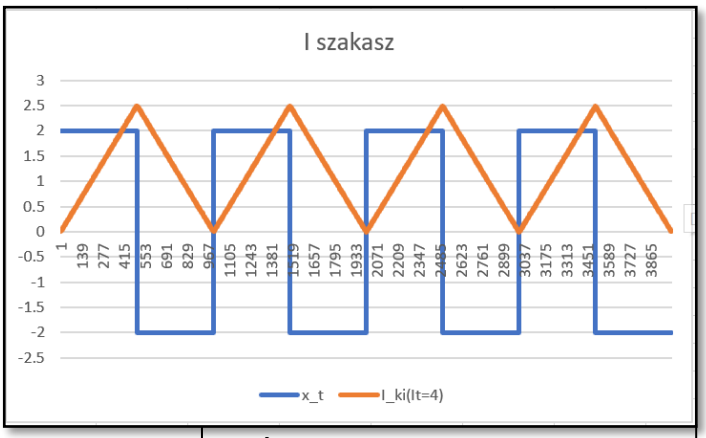
7.2. I alaptag [10]

Az integráló alaptag (7. ábra) a bemenetére érkező jelet integrálja és azt küldi a kimenetére. A programozás során ez úgy érhető el, hogy delta t ciklus időnként téglalap területet számolunk és azt hozzá adjuk az előző összegzett értékhez. A ciklusidőt célszerű minél kisebbre választani, hogy közel nulla legyen az eltérés az integrált érték és az összegzett érték között. Előbbit numerikus integrálásnak nevezzük.

Az integráló alaptag kimeneti egyenlete:

$$y_i(t) = \frac{1}{T_i} * (delta\ t * \sum_{n=0}^{i} x_n)$$

Az integráló tagra példa a tartály.



7. Ábra: integráló alaptag kimenete

7.3. D alaptag [11]

A differenciáló alaptag (8. ábra) a bemenetére érkező jelet deriválja és azt küldi a kimenetére. A differencia egyenlet alapján az egységugrásra adott kimeneten végtelen nagy lenne az érték. Mivel ez nem valós delta t-vel kell osztani.

Differenciáló alaptag differenciál egyenlete:

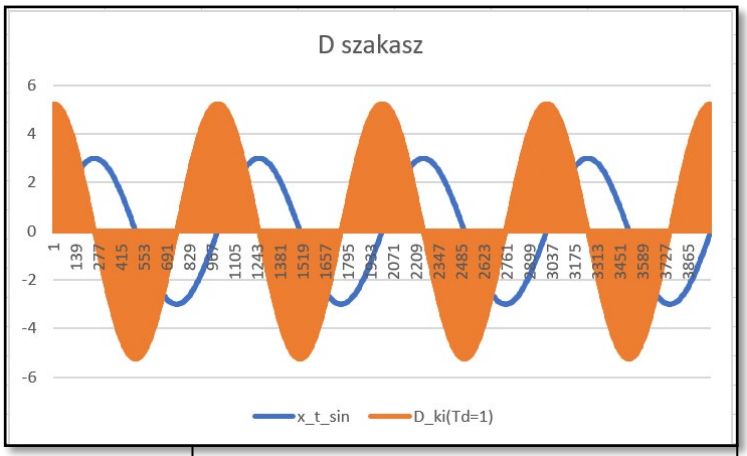
$$y(t) = T_D * \frac{dx(t)}{dt}$$

Differenciáló alaptag differencia egyenlete:

$$y(t) = T_D * \frac{x_i - x_{i-1}}{delta\ t}$$

A differencia egyenletből az következik, hogy a numerikus megvalósításnál a bemenő jel aktuális értékéből ki kell vonni az előző értéket. Ehhez szükséges az előző ciklusbéli bemenő értéket kell elmenteni. A különbséget osztjuk delta t-vel és T_D értékével.

A differenciáló tagra példa a kondenzátor.



8. Ábra: differenciáló alaptag kimenete

7.4. T alaptag [12]

Az egytárolós alaptag (9. ábra) önbeálló jellegű szakaszt eredményez a kimenetén. A beállás sebessége az időállandó nagyságától függ. Ez a függés fordítottan arányos. Minél kisebb az időállandó, annál gyorsabb a bemenő jelre való ráállás.

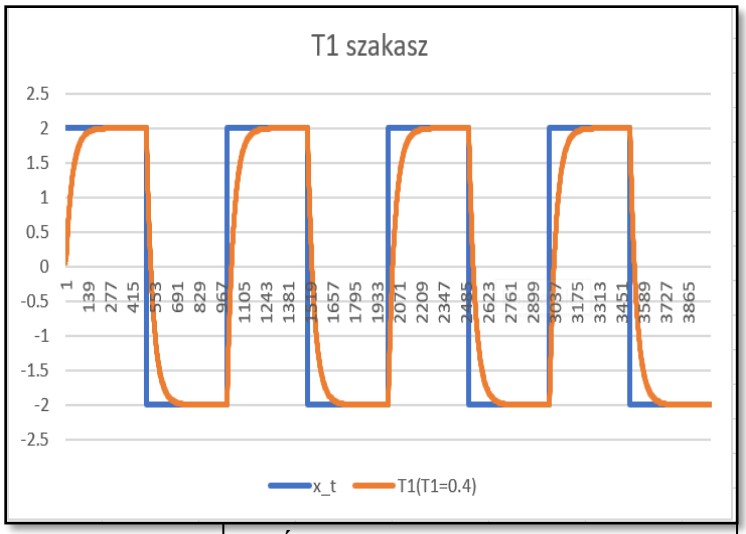
Egytárolós alaptag differenciál egyenlete:

$$T_1 * \frac{dy(t)}{dt} + y(t) = x(t)$$

Ez az egyenlet időben folytonos. Programozás szempontjából csak delta t időben tudjuk megírni a függvényt. Differencia egyenlet felírása szükséges:

$$y_i = y_{i-1} + \frac{delta\ t}{T_1} * (1 - y_{i-1})$$

Az egytárolós alaptagra példa a hőtechnikai melegítés.



9. Ábra: egytárolós alaptag kimenete

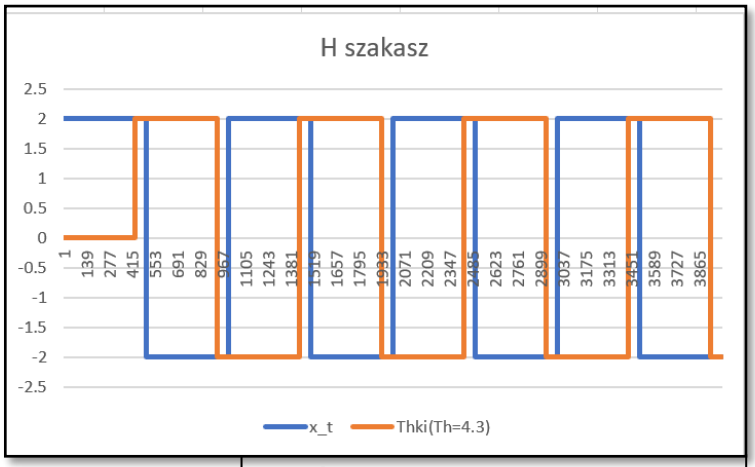
7.5. H alaptag

A holtidős alaptag (10.ábra) a bemenetére érkező jelet késleltetéssel, de változatlanul adja ki a kimenetére. A késleltetés mértékét a holtidős állandó adja meg. Amennyiben a holtidő értéke 2 sec, akkor a bemenő jelet 2 sec késleltetéssel adja ki a kimenetén a szakasz. A megvalósításához szükséges egy tömb használata, amibe folyamatosan betölti és a felülírja azt az értéket a program, ami már nem releváns.

Holtidős alaptag differenciál egyenlete:

$$y(t) = 1 * (t - \tau) * x * (t - \tau)$$

A holtidős alaptagra példa a számítási idő hatásainak modellezése.



10. Ábra: holtidős alaptag kimenete

7.6. T2 alaptag [13]

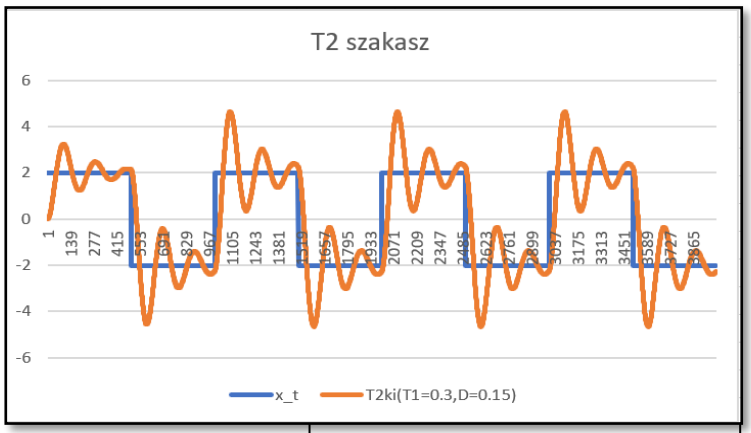
A T2 tag (11. ábra) túllendülésre és lengésre hajlamos. A D csillapítási tényező értékétől függően komplex pólusok lehetségesek, amitől lengés keletkezik a kimeneten. A valódi két tárolós tag nem valósítható meg két darab egytárolós tag összekapcsolásával.

Amennyiben D értéke nagyobb mint 1, akkor nincs lengés és a kimenet azonos, mint két egytárolós tag kimenete. Ha D értéke 0 és 1 között van, akkor előfordulhat lengés, viszont a kimenet csökkenő tendenciával rááll a bemeneti jelre. Az utolsó lehetőség, hogy a D értéke pontosan 0, akkor a kimenet gerjedő lengést eredményez és egyre nagyobb lengést okoz a kimeneten.

Kéttárolós alaptag differenciál egyenlete:

$$T^2 * \frac{d^2y(t)}{dt^2} + 2DT * \frac{dy(t)}{dt} + y(t) = x(t)$$

A T2 alaptagra példa az RLC kör.



11. Ábra: T2 alaptag kimenete

8. GUI RÉSZLETEZÉSE

A GUI, ami a tanári kalibrációs program akkor szükséges, amikor a diák még nem kapja kézhez a szakasz szimulátort. Ezt a programot csak a tanár használja. Lehetőség van 120 különböző összetételű szakaszt beállítani, minden alaptagnak az időálladóját és tényezőjét 2 század pontossággal beállítani. A maxikális érték 999,99 lehet.

8.1. Üzemmódok

12.Ábra: Üzemmód kiválasztása

Két üzemmód van (12. ábra). Amikor a jelölőnégyzet be van pipálva, akkor diák üzemmód, ha nincs bepipálva, akkor az önellenőrző (tanári) üzemmód lesz aktív a kalibráció feltöltése után. Két üzemmódra szükség, mert így lényegesen gyorsabb ciklusidőt lehet elérni, amire a diák kézhez kapja a szakasz szimulátort. Az önellenőrző módban, az Arduino csak a PC-vel kommunikál USB-n keresztűl. Ennek a kommunikációnak a ciklusideje 0.027 sec. A másik üzemmódban ez a kommunikáció nem folytonos, csak a kalibrációs értékek küldése történik a PC felől az Arduino felé, majd ez a kapcsolat lezárul és mentés után csak a PLC-Arduino kommunikál egymással. Ennek a kommunikációnak a ciklusideje sokkal kevesebb, csupán 0,014 sec. Az előbbi értékek ± 0.0005 sec-el tudnak elérni mérések alapján. A kimeneti jel megfigyelése önellenőrző módban a GUI felületén lehetséges. Ebben az üzemmódban a bemeneti jelet is az Arduino állítja elő. A bemeneti jel a ciklusidőtől függ, mint minden más függvény. A program elején és végén mérek egy pontos ezredmásodperces időt és a kettőt kivonva megkapom a delta t-t, azaz a ciklusidőt. A bemeneti jel periódus ideje 1000*delta t négyszögjel. A maximuma 3 V, a minimuma 1 V.

8.2. Adatok kimentése

13. Ábra: Adatok kimentése

Önellenőrző üzemmódban lehetőség van az adatok kimentésére (13. ábra) egy .txt fájlban. A program az adatokat csak abban az esetben menti ki, ha a jelölőnégyzet ki van jelölve. Ezt a fájlt a program az asztalra helyezi el. A kimentés kiválasztásánál beállítható, hogy idő alapján mettől meddig mentsen program. Az adatok fejléccel ellátott, space által tagolt táblázatba

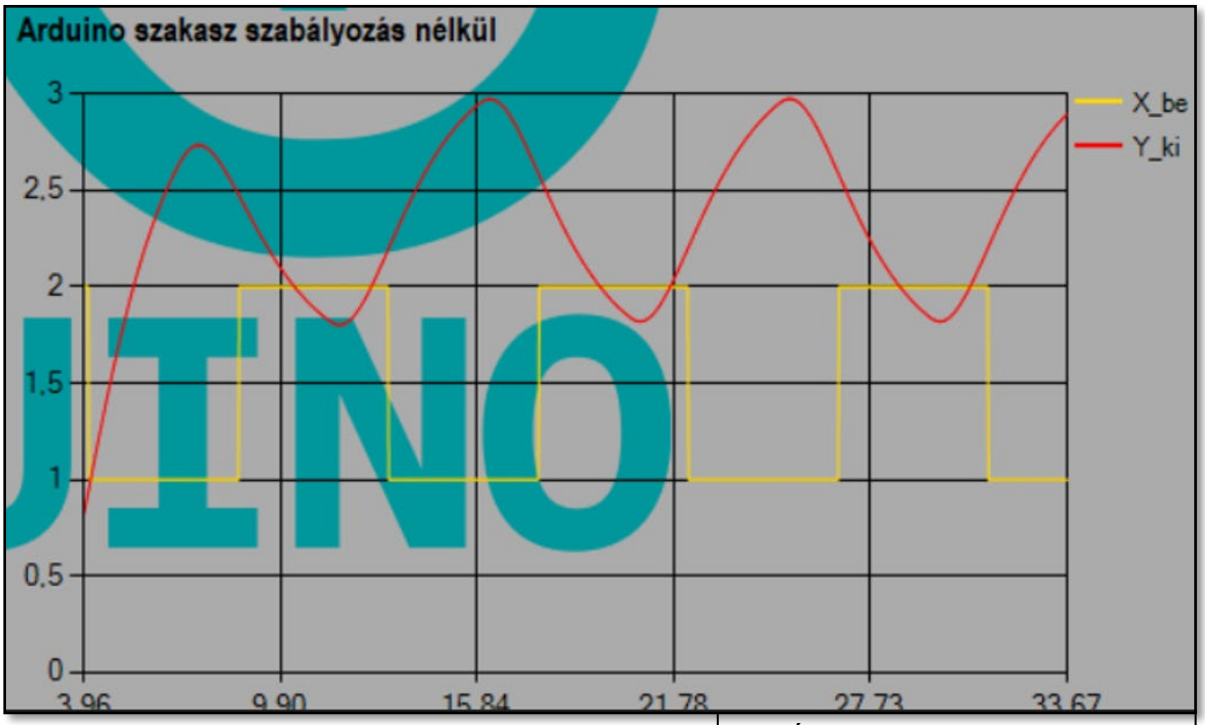
kerülnek kiírásra. A .txt fájl a bemeneti jelet, a kimeneti jelet és az eltelt időt tartalmazza. 14. ábrán látható a kimentés kódrészlete.

```
label4.Text = Convert.ToString(Convert.ToDouble(beolvasott_array[5]) / 1000);
if (kimentes == true && Convert.ToDouble(textBox_AdatTol.Text) <= x && Convert.ToDouble(textBox_AdatIG.Text) >= x)
{
    string txt_data_out = Convert.ToString(x_t_1) + " " + Convert.ToString(x_t_2) + " " + Convert.ToString(x);
        using (StreamWriter outputFile = new StreamWriter(Path.Combine(docPath, "Szakasz_viselkedese.txt"), true))
        {
            outputFile.WriteLine(txt_data_out);
        }
}
```

14. Ábra: Adatok kimentése kódrészlet

8.3. Szakasz viselkedésének ellenőrzése

A szakasz viselkedésének idődiagrammos ellenőrzése csak az első üzemmódban lehetséges. A felkalibrált szakaszt szabályozás nélkül a GUI felületén elhelyezett vizualizációs data chart (15. ábra) jelzi ki időben folytonosan.



15. Ábra: Vizualizációs data chart

A diagram függőleges tengelye automatikusan skálázódik a minimum és a maximum értékek között. A vízszintes tengely, ami az időt mutatja 1100*delta t ideig tölti a diagrammot, utána elkezdi shiftelni jobbról balra, így mindig az eltelt 1100*delta t idő intervallumot láthatjuk.

A vizualizációs data chart a kalibrálás után a Serial Port COM3-as csatornáján fogadott bejövő adatcsomagból jelzi ki az alapjel x(t) és a kimeneti y(t) értékét. Az adatokat csomagokban kapja a USB porton keresztűl. A csomagok úgy néznek ki, hogy egy csomagnyitó és csomagzáró jel

között különleges karakterrel elválasztva vannak az adatrészek. A felépítése <data1#data2#data3#data4>. Az egész csomag string típusú a kommunikáció típusának megfelelően. A típus konverziók miatt negatív számot nem lehet átküldeni, ezért a küldés előtt abszolút értékké alakítom. Azt, hogy a szám eredetileg negatív, vagy pozitív volt, a data1 és a data3 tartalmazza. Amennyiben a data1 értéke 0, akkor negatív, ha 1, akkor pozitív az őt követő data2 valódi értéke. A beolvasott stringet ellenőrzés alá vetem minden esetben, ehhez kell a nyitó és záró karakter, valamint az adat elválasztó karakter. Amennyiben esetleges kommunikációs zavar által okozott space, „\0”, „\r”, „?” és az ABC bármely nagy vagy kis karaktere bekerül a csomagba az megakasztja a folyamatot. Ezért ezeket ellenőriztetem és törlöm, ha előfordul. A beérkezett adatcsomagot feldarabolom az adatelválasztó karakterek segítségével és stringből double formátumú számmá konvertálom. Küldés előtt az Arduino-ban az értékeket beszoroztam 100000-el annak az érdekében, hogy az érték 4 tizedes jegyig legyen pontos. Ezért itt a GUI chart kijelzéshez vissza kell osztani ugyanennyivel.

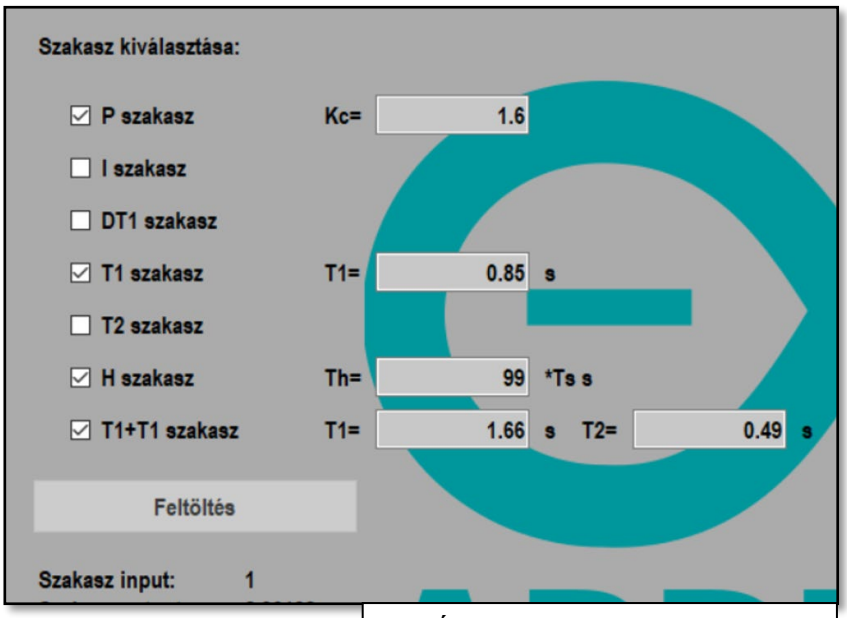
A 16. ábrán a kódolás ide tartozó részlete látható.

```
public void WriteToForm(string beolvasott_string)
{
    //ez a funkció kezelei az arduinotól érkező adatot
    if (beolvasott_string.Contains("#"))
    {
        beolvasott_string = beolvasott_string.Replace(" ", "");

        string[] beolvasott_array = beolvasott_string.Split('#');
        if (beolvasott_array.Length == 7)
        {
            if (beolvasott_array[0] == "<" && beolvasott_array[6] == ">")
            {
                if (beolvasott_array[1] == "-")
                {
                    x_t_1 = Convert.ToDouble(beolvasott_array[2]) / -100000;
                    input_value.Text = Convert.ToString(Convert.ToDouble(beolvasott_array[2]) / -100000);
                }
                else
                {
                    x_t_1 = Convert.ToDouble(beolvasott_array[2]) / 100000;
                    input_value.Text = Convert.ToString(Convert.ToDouble(beolvasott_array[2]) / 100000);
                }
                if (beolvasott_array[3] == "-")
                {
                    x_t_2 = Convert.ToDouble(beolvasott_array[4]) / -100000;
                    output_value.Text = Convert.ToString(Convert.ToDouble(beolvasott_array[4]) / -100000);
                }
                else
                {
                    x_t_2 = Convert.ToDouble(beolvasott_array[4]) / 100000;
                    output_value.Text = Convert.ToString(Convert.ToDouble(beolvasott_array[4]) / 100000);
                }
                Android_szakasz_kimenet.Series[0].Points.AddXY(x, x_t_1);
                Android_szakasz_kimenet.Series[1].Points.AddXY(x, x_t_2);
                if (Android_szakasz_kimenet.Series[0].Points.Count > 1100)
                    Android_szakasz_kimenet.Series[0].Points.RemoveAt(0);
                Android_szakasz_kimenet.ChartAreas[0].AxisX.Minimum = Android_szakasz_kimenet.Series[0].Points[0].XValue;
                Android_szakasz_kimenet.ChartAreas[0].AxisX.Maximum = x;
                x += (Convert.ToDouble(beolvasott_array[5]) / 1000);
                label4.Text = Convert.ToString(Convert.ToDouble(beolvasott_array[5]) / 1000);
```

16. Ábra: A megjelenítés kódrészlete

8.4. Kalibráció beállítása

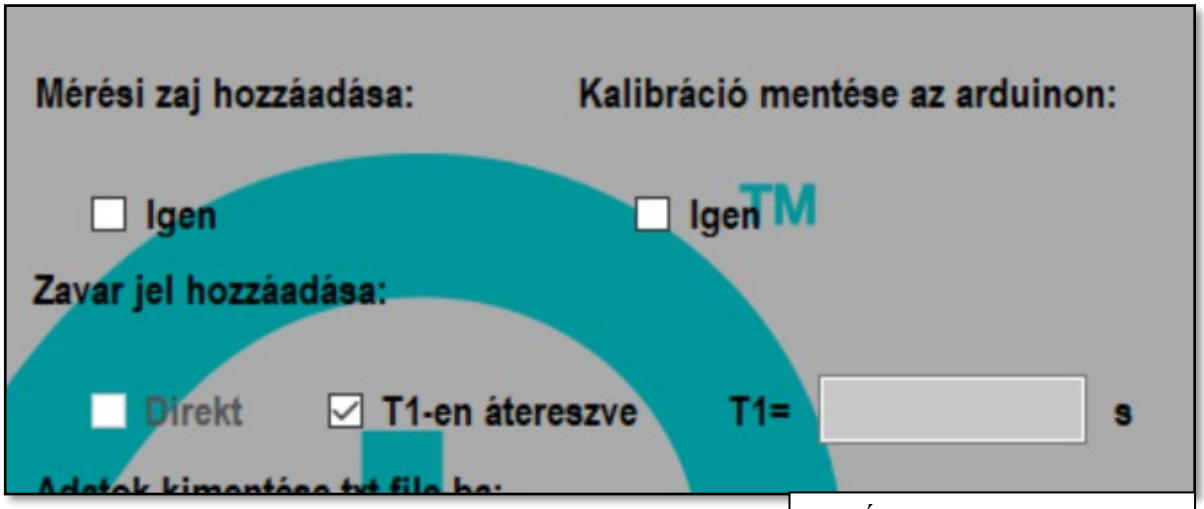
A kalibráció legfontosabb része a szakasz összetétele és a résztvevő tagok időállandóinak beállítása. A 17. ábrán látható az előbbi grafikai megjelenítése.



17. Ábra: HPT3 kalibrációs adatai

Amennyiben a jelölőnégyzet a tag elött ki van választva, megjelenik a taghoz tartozó időállandó vagy tényező. A szövegdobozba csak számot lehet írni 0-999.99 tatrományig. A kétfajta üzemmód miatt a ciklusidő megvátozik a különböző üzemmódokban. Ezért a holtidőt nem sec-ben lehet megadni, hanem ciklus darabszámban. Ennek a maximális értéke elég kicsi, csak 150 db lehet. Ezt egy popup figyelmeztető üzenet is mutatja, ha belekattintunk a szövegdobozba. Azért kellett korlátoznom ezt az értéket, mert az Arduino kifut a memóriából nagyobb tömb esetén. Azt, hogy miért kell tömb, majd a 9. fejezetben részletezem.

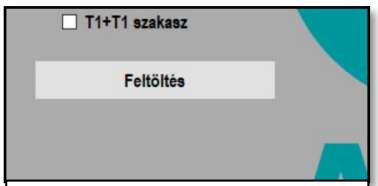
A 18. ábrán a GUI-nak az a részlete látható, ahol a zajt és a zavart lehet beállítani és hozzáadni az szakasz szimulátor kimeneti jeléhez. A zaj egy Arduino által random generált szám ± 0.0500 és 0.0001 közötti érték. A zavar jel alapját pedig az Arduinora csatlakoztatott potenciométer adja, ezt lehet közvetlenül hozzáadni a szakasz szimulátor kimeneti jeléhez, vagy pedig egy T1-en átengedve. Ha a T1-es változatot akarjuk beállítani, akkor megjelenik a T1-hez tartozó időállandó.



18. Ábra: HPT3 kiválasztása

8.5. Kalibrációs adatcsomag küldése

A kalibráció beállítása után a feltöltés gomb (19. ábra) megnyomásával lehet eljuttatni az adatokat az Arduino felé. A GUI egyetlen csomagba rendezi az adatokat és úgy küldi el az USB porton keresztűl. A program létrehoz egy stringet, amiben a különböző adatokat különleges karakterrel választja el. Itt nem füzök a stringhez nyitó és záró karaktereket, mert ez a kommunikáció egyirányú és csak egyszer történik meg, ezért nem fordul elő több adatcsomag, amiket meg kellene különböztetni.



19. Ábra: Feltöltés gomb

A string elején a kiegészítő információkat tárolom. A legelső adatrész azt az információt hordozza, hogy milyen üzemmódban szeretném működtetni a szakasz szimulátort. Amikor ennek értéke 1-es, akkor a diák üzemmód lép érvénybe és az Arduino a kalibrációs adatokat

meg fogja őrizni áramtalanítás esetén. Ellenkező esetben, ha 0 az értéke az első adatrésznek, akkor nem tárol adatokat és belső generált négyszögjel lesz a szakasz bemeneti jele. A következő adatok azt az információt hordozzák, hogy szükség van-e zaj vagy zavarjel, és ha kell zavarjel akkor melyik fajta. A következő adatrészek azt az információt hordozzák, hogy melyik alaptagra van szükség. Amikor egy alaptag aktív, akkor annak mennyi az időállandója vagy a tényezője.

A 20. ábrán ennek a kódrészlete jelenik meg. Amikor, nincs kiválasztva egy szakasz, akkor is kitöltöm egy 0-val a szakaszhoz tartozó mérték(ek) helyét. Így biztosítom azt, hogy minden esetben megegyező adatrészlet mennyiséget küld csomagonként, ami azért fontos, hogy a feldolgozása könnyebb legyen.

```
if (ArduMentes.Checked)
{
    kalibracio = "1";
}
else
{
    kalibracio = "0";
}
if (checkBox_ZavarIgen.Checked)
{
    kalibracio = kalibracio + "#1";
}
else
{
    kalibracio = kalibracio + "#0";
}
if (!DirektZavar.Checked && !T1elZavar.Checked)
{
    kalibracio = kalibracio + "#0#0";
}
if (DirektZavar.Checked)
{
    kalibracio = kalibracio + "#1#0";
}
if (T1elZavar.Checked)
{
    kalibracio = kalibracio + "#2#" + zavarT1.Text;
}
if (P_sz.Checked)
    kalibracio = kalibracio + "#1#" + textBoxKc.Text;
else
    kalibracio = kalibracio + "#0#0";
if (I_sz.Checked)
    kalibracio = kalibracio + "#1#" + textBoxTi.Text;
else
    kalibracio = kalibracio + "#0#0";
if (D_sz.Checked)
    kalibracio = kalibracio + "#1#" + textBoxTd.Text + "#" + textBoxDT1.Text;
else
    kalibracio = kalibracio + "#0#0#0";
if (T1_sz.Checked)
    kalibracio = kalibracio + "#1#" + textBoxT1.Text;
else
    kalibracio = kalibracio + "#0#0";
if (T2_sz.Checked)
    kalibracio = kalibracio + "#1#" + textBoxT2.Text + "#" + textBoxD.Text;
else
    kalibracio = kalibracio + "#0#0#0";
if (H_sz.Checked)
    kalibracio = kalibracio + "#1#" + textBoxTh.Text;
else
    kalibracio = kalibracio + "#0#0";
if (T1T1_sz.Checked)
    kalibracio = kalibracio + "#1#" + textBoxT1T1.Text + "#" + textBoxT1T2.Text;
else
    kalibracio = kalibracio + "#0#0#0";
serialPort1.WriteLine(kalibracio);
```

20. Ábra: A kalibrációs adatok feldolgozása és csomagba rendezése kódrészlet

9. MEGVALÓSÍTÁS AZ ARDUINO-N

Az Arduino szerepe a szakasz szimulálása a szabályozó körben, összesen 120 különböző fajta szakasznak megfelelően tud funkcionálni. A szakaszok mellett képes zajt hozzáadni a kimeneti jelhez, illetve zavar jelet is hozzá tud adni. Aktuális állapot kijelzést mutat LCD panel írásával. Az állapotok a következők:

- „Várakozás a GUI kalibrációjára!”

Itt az Arduino az USB-n beérkező kalibrációs adatcsomagot várja.

- „A kalibráció sikeres volt!”

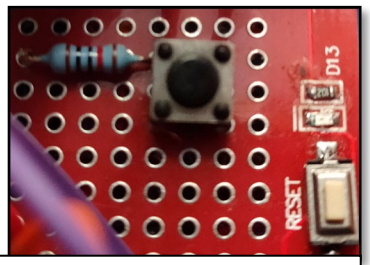
A beérkező adatcsomagot sikeresen beolvasta és feldolgozta, itt már működik a szimuláció a beállított értékeknek megfelelően.

- „EEPROM-ból kiolvasva”

Az Arduino egy áramtalanítást követően a kalibráció egészét kiolvasta az EEPROM-ból, működik a szimuláció.

- „Törlés sikerült!”

Az Arduino törölte a kalibrációs adatokat az EEPROM-ból és 3 sec után visszaáll a „Várakozás a GUI kalibrációjára!” állapotba. Ezt a törlést egy manuálisan megnyomható gombbal (21. ábra) lehet előidézni, ami csatlakoztatva van a Screw shield-en keresztül az Arduino-hoz.



21. Ábra: EEPROM törlőgomb beépítése

A 22. ábrán láthatóak a törlőgomb megnyomására vonatkozó utasitások.

```
if(digitalRead(2) && EEPROM.read(0))//mentett kalibráció törlése A2-re kötött nyomógombbal
{
  EEPROM.write(0, 0);//mentes
  EEPROM.write(1, 0);//zaj
  EEPROM.write(2, 0);//zavar
  EEPROM.write(6, 0);//P_tag
  EEPROM.write(10, 0);//I_tag
  EEPROM.write(14, 0);//DT1_szak
  EEPROM.write(21, 0);//T1_tag
  EEPROM.write(25, 0);//T2_tag
  EEPROM.write(32, 0);//H_tag
  EEPROM.write(26, 0);//T1_T1_szak
  lcd.setCursor(0, 0);
  lcd.print("Torles sikerult!");
  delay(1000);
  setup();
}
```

22. Ábra: EEPROM törlőgomb kódrészlet

9.1. Alaptagok

A 7. fejezetben felsorolt alaptagok kódolt megfelelőjét kellett megvalósítanom az Arduino-val.

A 23. ábrán látható a proporcionális alaptag.

```
float P_szakasz(float x_be)
{
    float x = 0; //return
    x = x_be * Pa;
    return x;
}
```

23. Ábra: Proporcionális alaptag kódrészlet

Az alaptag bemenetére érkező x(t) jel Kc-val való szorzatát küldi a kimenetére.

A 24. ábrán látható az integráló alaptag.

```
float I_szakasz(float x_be)
{
    float x = 0;//return
    int_sum = int_sum + (delta_t * x_be);
    x = int_sum * (1 / Ti);
    return x;
}
```

24. Ábra: Integráló alaptag kódrészlet

Az alaptag bemenetére érkező x(t) jel integrált összeg és az integrálási időállandó hányadosát írja a kimenetre. Az integrált összeg az előző integrált értékének és a delta t, bemeneti jel szorzatának összege.

A 25. ábrán látható a DT1 szakasz kódolása.

```
float DT1_szakasz(float x_be)
{
    if (delta_t < 0.01)
    {
      delta_t=0.026;
    }
    float x = 0;//return
    float D_ki_temp = 0;
    D_ki_temp = Td * (x_be - x_prev) / delta_t;
    x_prev = x_be;
    float inc = 0;
    inc = (delta_t / DT1) * (D_ki_temp - y_curr_DT);
    y_curr_DT = inc + y_curr_DT;
    x = y_curr_DT;
    return x;
}
```

25. Ábra: DT1 szakasz kódrészlet

DT1 szakasz lett megvalósítva a D alaptag helyett, mert a valóságban sem fordul elő olyan szakasz, ami csak D tagkén viselkedik (8. ábra).

Az Arduino ciklusideje a mérések alapján hibát okozott ennél a szakasznál. A kalibrálás feldolgozása 1-2 sec közötti időt vett igénybe és a következő 3 ciklusiő pedig 0.001 sec időt mutatott. Ez a nagyon kis idő bezavart a szakasz helyes működésébe, kiugrasztotta a kimeneti értéket. A feldolgozási időt kiküszöböltem azzal, hogy csak az utána következő ciklustól induljon a tényleges szimuláció, a következő 3 ciklusidőből adódó hibát itt javítottam ki. Ez a hiba csak a tanári (önellenőrző) üzemmódban lép fel, ha a diák üzemmód aktív, a bootolás közben kalibrál be a szakasz szimulátor és ennek ideje nem számít bele a ciklusidőbe.

Az egytárolós tag differencia egyenlete:

$$y_i = y_{i-1} + \frac{delta\ t}{T_1} * (1 - y_{i-1})$$

A differenciáló alaptag differencia egyenletének módosítása:

$$y(t) = T_D * \frac{y_i - x_{i-1}}{delta\ t}$$

A 26. ábrán látható a T1 szakasz kódolása.

```
float T1_szakasz(float x_be)
{
    float x = 0;//return
    float inc = 0;
    inc = (delta_t / T1) * (x_be - y_curr_T1);
    y_curr_T1 = inc + y_curr_T1;
    x = y_curr_T1;
    return x;
}
```

26. Ábra: T1 szakasz kódrészlet

Az aktuális bemenetből kivonjuk az elöző értéket. Ezt megszorozzuk delta t és az egytárolós tag időállandójának hányadosával. Az így kapott értéket hozzáadjuk az előző bemeneti értékhez és ezt íratjuk a kimenetre.

A 27. ábrán látható a zavar jelhez tartozó egytárolós tag kódrészlete.

```
float T1_zavar_szakasz(float x_be)
{
    float x = 0;//return
    float inc = 0;
    inc = (delta_t / T1zavar) * (x_be - y_curr_T1zavar);
    y_curr_T1zavar = inc + y_curr_T1zavar;
    x = y_curr_T1zavar;
    return x;
}
```

27. Ábra: Zavar jel T1 szakasz kódrészlet

A 28. ábrán látható a holtidős alaptag megvalósítása.

```
float H_szakasz(float x_be)
{
    float x = 0;//return
    x = Th_x_t[x_t_seged_h];
    if (x_t_seged_h < Th)//Th max 150 Lehet
    {
        Th_x_t[x_t_seged_h] = x_be;
    }
    else if (Th == x_t_seged_h)
    {
        Th_x_t[x_t_seged_h] = x_be;
        x_t_seged_h = -1;
    }
    x_t_seged_h++;
    return x;
}
```

28. Ábra: Holtidő alaptag kódrészlet

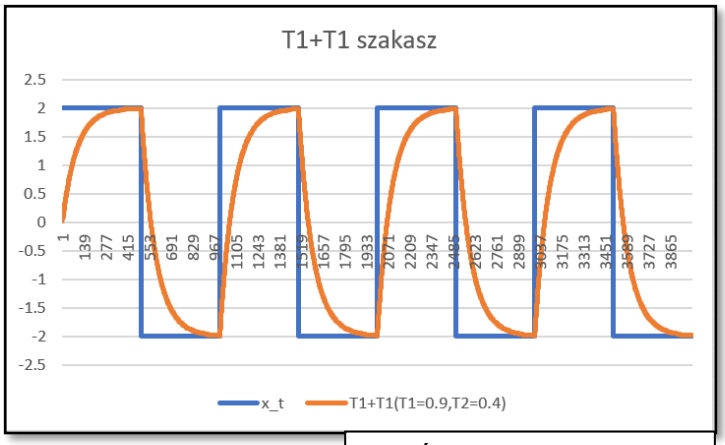
Az aktuális bemenő jel értékét tárolom egy 150 elemes tömbben. Alulról fölfele töltöm a tömböt addig az elemszámig, ami megegyezik a holtidő időállandójával. Ha ezt a számot elérte, akkor felülíródik a legalsó 0-ik elem.

A 29. ábrán látható a két egytárolós szakasz sorba kapcsolása.

```
float T1_T1_szakasz(float x_be)
{
    float x = 0;//return
    float inc = 0;
    inc = (delta_t / T1T1) * (x_be - y_curr_T1T1);
    y_curr_T1T1 = inc + y_curr_T1T1;
    float inc2 = 0;
    inc2 = (delta_t / T1T2) * (y_curr_T1T1 - y_curr_T1T2);
    y_curr_T1T2 = inc2 + y_curr_T1T2;
    x = y_curr_T1T2;
    return x;
}
```

29. Ábra: T1+T1 kódrészlet

Mind a két egytárolós tagnak egyéni időállandója van. A sorba kötött egytárolós tagok kimeneti függvény képe a 30. ábrán látható.



30. Ábra: T1+T1 kimenet

A 31. ábrán látható a valódi kéttárolós tag megvalósítása.

```
float T2_szakasz(float x_be)
{
    float x = 0;//return
    float ddy = 0;
    ddy = 1 / (T2 * T2) * (x_be - y) - (2 * D) / T2 * dy;
    dy = dy + (delta_t * ddy);
    y = y + (delta_t * dy);
    x = y;
    return x;
}
```

31. Ábra: T2 kódrészlet

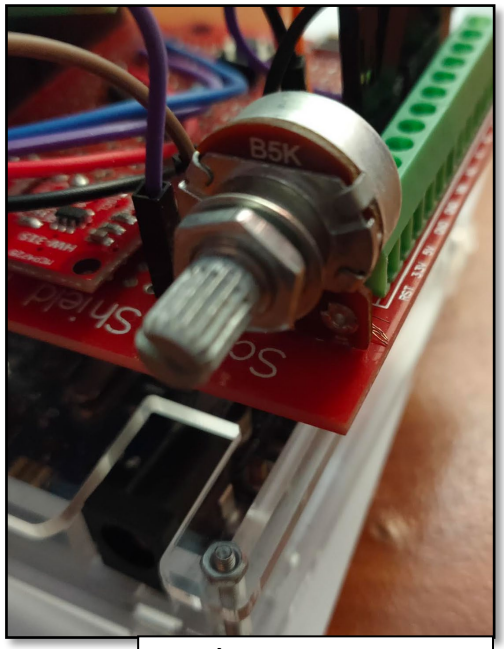
Az 5. sorban a 2. derivált, a következőben az 1. derivált, végül a 0. derivált számolása történik meg.

A kéttárolós tag differencia egyenlete:

$$y(t) = \frac{1}{s^2T_2^2 + s2DT + 1}$$

9.2. Zaj generálása

A valóságban is sokszor előfordul, hogy a szabályozni kívánt szakaszt zaj éri. Például zárt térben mért hőmérséklet, esetében a külső hőmérséklet hirtelen változik. Így a szakasz szimulátor sem maradhatott beépített zaj generálása nélkül. A mértékét egy potenciométer (32. ábra) tekerésével lehet állítani. Azt, hogy direktben, vagy T1-en keresztül adódik hozzá a szabályozott jellemzőhöz a GUI segítségével lehet beállítani.



32. Ábra: Potenciométer

9.3. Kommunikáció a GUI-val

Az Arduino a tanári üzemmódban a felkalibrálás után minden ciklusban az USB serial porton keresztül adatcsomagot küld a GUI felé. Az adatfolyam csak egy irányú és a kis ciklusidő mellett folytonosnak tekinthető. Öt információt rendezek egy adatcsomagba. Az információ küldés az önellenőrzés érdekében történik, a GUI ennek segítségével tudja az idődiagramot kirajzolni. Egy csomagba rakom a bemenő és kimenő jelem értékét, azok előjeleit, valamint a ciklusidőt. A 33. ábrán látható az ide tartozó kódrészlet.

```
if (x_t < 0)
{
    x_t_abs = -1 * x_t;
    dtostrf(x_t_abs * 100000, -4, 0, DataToSend);
    adatToPc = "<#-#" + String(DataToSend) + "#";
}
else if (x_t >= 0)
{
    x_t_abs = x_t;
    dtostrf(x_t_abs * 100000, -4, 0, DataToSend);
    adatToPc = "<#+#" + String(DataToSend) + "#";
}
if (Yki < 0)
{
    Yki_abs = -1 * Yki;
    dtostrf(Yki_abs * 100000, -4, 0, DataToSend);
    int delta = delta_t*1000;
    adatToPc = adatToPc + "-#" + String(DataToSend) + "#" + String(delta) + "#>";
}
else if (Yki >= 0)
{
    Yki_abs = Yki;
    dtostrf(Yki_abs * 100000, -4, 0, DataToSend);
    int delta = delta_t*1000;
    adatToPc = adatToPc + "+#" + String(DataToSend) + "#" + String(delta) + "#>";
}
```

33. Ábra: adatformálás csomagokba

A soros kommunikáció string típusú. Egy stringbe fűzöm fel az aktuális ciklus értékeit. Az adatcsomagokat elkülönítem egymástól egy nyitó (<) és egy befejező (>) karakterrel. Egy csomagon belül pedig a különböző információkat (#) karakterrel választom el. Negatív számot nem tudok string formátumba konvertálni, mert a visszakonvertálásnál elveszti az előjelét. Ennek kiküszöböléséhez az adat értékét abszolút értékké változtatom és az érték elé rakok egy „+” illetve „-” csomagrészt, ami a jel előjelének információját hordozza. Ezt a visszaolvasásnál figyelni kell. Az nem elegendő, hogy külön csomagrészként kell elküldenem azt, hogy a szám pozitív, vagy negatív. A pontosságot megőrizve a bemenő, illetve kimenő jel értékét beszorzom 100000-el, a delta t értékét pedig 1000-rel. Itt, a GUI-hoz beérkező jel fogadásánál a visszaosztásra szintén figyelni kell.

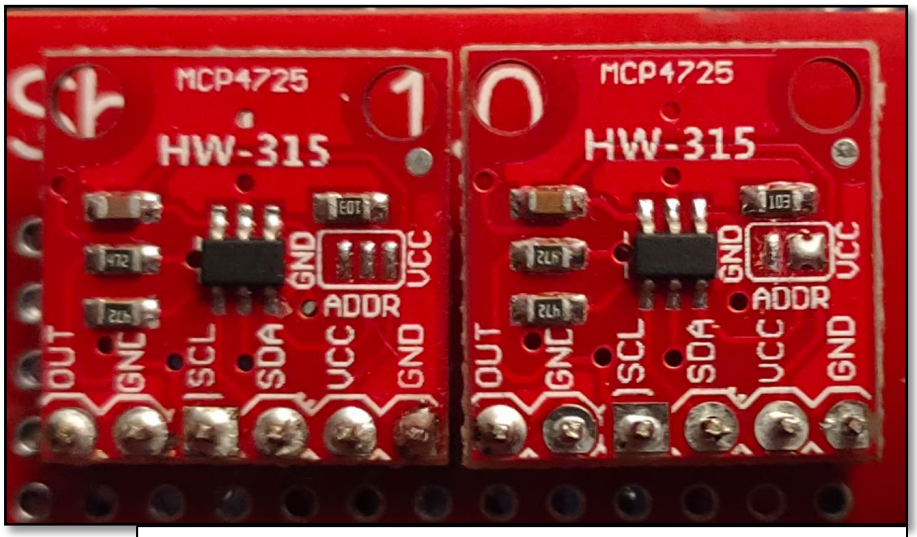
Az előbbi adatfolyamot megelőzi egy egyszeri, fogadott adatcsomag. Ennek a küldését a 8.5. fejezetben részletezem. Ezt az adatcsomagot a küldést ismerve lehet feldarabolni. A beérkező kalibrációs adatokat tartalmazó string felosztása a 34. ábrán látható.

```
String Feloszto_fvg(String data, char delimiter, int sequence)
{
    int stringData = 0;
    String sectionData = "";

    for (int i = 0; i < data.length() - 1; i++)
    {

        if (data[i] == delimiter)
        {
            stringData++;
        }

        else if (stringData == sequence)
        {
            sectionData.concat(data[i]);
        }

        else if (stringData > sequence)
        {
            return sectionData;
            break;
        }
    }

    return sectionData;
}
```

34. Ábra: String darabolása

9.4. Kommunikáció a PLC-vel

Az 5. fejezetben több kommunikációs lehetőséget is részleteztem, amelyekkel meg lehet valósítani a kommunikációt a szakasz szimulátor és a PLC között. Mivel a kis költségvetésre és a kompatibilitásra törekszem az összes laborban megtalálható PLC-vel analóg módon küldöm a kimenő jelet. Analóg kimenete nincs az Arduino-nak, ezért be kell iktatni egy Digital to Analog konvertert a kommunikációba. Esetemben két különböző jelet is szükséges továbbítani, az egyik a szakasz kimenete, a másik pedig a zavar jel erőssége. A 35. ábrán látható a két darab beépített DAC. Az egyik módosított I^2C 0x61 címmel, a másik az alapértelmezett 0x60 címmel.



35. Ábra: Konverterek a baloldali 0x60, a jobboldali 0x61

A TTL maximális jelerőssége 5V feszültségű. Az 5V digitális megfelelője a 4090-es érték. Mivel negatív feszültséget nem tudok előállítani, ezért a szakasz kimenetét korlátozni kellett 0-5V közé. Skálázni se akarjuk az értéket, hogy a diáknak semmilyen plusz információt ne kelljen megadni. Könnyebb így beazonosítani, csak az eszközből kiindulva, hogy milyen összetételre lett konfigurálva szimulátor.

9.5. LCD panel kijelzés

Az 2 soros 16 karakteres LCD panel visszajelzést ad a szimulátor aktuális állapotáról, ezeket a kiírásokat a 9. fejezet bevezetőjében részleteztem. A kiíratáshoz szükséges a LiquidCrystal_I2C.h elnevezésű könyvtár használata. Az alapállapot kiíratása látható a 36. ábrán.

```
if(EEPROM.read(0)==0)
{
lcd.setCursor(0, 0);
lcd.print("Varakozas a GUI");
lcd.setCursor(1, 1);
lcd.print("kalibraciora!");
}
```

36. Ábra: LCD kiírás

9.6. EEPROM mentés

A kalibrációs adatok elmentésére a diák üzemmódban szükség van, hiszen amikor a tanár befejezi a konfigurálást az eszköz áramtalanítás után kerül a diák kezébe. Az adatokat az Arduino beépített EEPROM területére mentem. Az Arduino UNO EEPROM területe 1024 bájt.

Címzése 0-1023 között található és mindegyik címen 1 bájt adatot lehet letárolni. Egy bájt érték binárisan 0-255 közé esik. A menteni kívánt adatok ennél sokkal több helyet foglalnának, ha azokat a bináris formájukban menteném le. Az összes értéket 2 tizedes pontossággal akarom lementeni.

A megoldás az, hogy 6 db digitre rendezem a számot úgy, hogy az első 4 digit az egészek értékeket az 5-6-ik digit pedig a tizedesjel utáni értékeket jelöli. A 37. ábrán látható a formázás.

```
int decHely = felosztott_adat[i].indexOf('.');
int hossz = felosztott_adat[i].length();
int decSzamok = hossz-decHely-1;
String KalibString;
if(decHely!=-1) //Input adat formázása 6 digitre, kalib range: 0.01-9999.01
{
  KalibString = felosztott_adat[i].substring(0,decHely) + felosztott_adat[i].substring(decHely+1,decHely+3);
  switch (decSzamok) {//decimális számok alapján formázás, 2 dec érték kell a 6 digithez.
    case 0:
      KalibString = KalibString + "00";
      break;
    case 1:
      KalibString = KalibString + "0";
      break;
    default:
    break;
  }
  switch (decHely) {//egész számok formázása dec jelig. decből 2 van, 4 egész kell a 6 digithez
    case 1:
      KalibString = "000" + KalibString;
      break;
    case 2:
      KalibString = "00" + KalibString;
      break;
    case 3:
      KalibString = "0" + KalibString;
      break;
    default:
    break;
  }
}
```

37. Ábra: Digitek kezelése

Példának veszek egy 2,2-es értékű Kc erősítést, azt 000220 stringgé alakítom és 3 különböző EEPROM címen tárolom le. Az első címen tárolt érték 0, a második címen tárolt érték 2, a harmadik címen tárolt érték pedig 20. A 38. ábrán látható a mentés kódrészlete.

```
  int digit3 = KalibString.substring(4,6).toFloat();
  EEPROM.write(k+1, digit3);
  int digit2 = KalibString.substring(2,4).toFloat();
  EEPROM.write(k, digit2);
  int digit1 = KalibString.substring(0,2).toFloat();
  EEPROM.write(k-1, digit1);
}
```

38. Ábra: EEPROM mentés

9.7. EEPROM kiolvasása

Az EEPROM kiolvasásáért a 39. ábrán látható kódrészlet felel.

```
String eepromRead(int kezd, int veg)
{
  String beolvasottErtek = "";
  int i=kezd;
  for(i; i<=veg; i++)
  {
  if(String(EEPROM.read(i)).length()==1)
  {
    beolvasottErtek = beolvasottErtek + "0" + String(EEPROM.read(i));
  }
  else if(String(EEPROM.read(i)).length()==2)
  {
    beolvasottErtek = beolvasottErtek + String(EEPROM.read(i));
  }
  }
  return beolvasottErtek;
}
```

39. Ábra: EEPROM kiolvasása

A kiolvasott érték integer típusú szám, amit beolvasó függvényben stringgé alakítok és ha csak egy karakter, akkor elé csatolok egy 0-át. A beolvasott string adatot a megfelelő változónak átadom float típusban, és elosztom 100-zal. Ez egy kódrészletben látható a 40. ábrán.

```
if (EEPROM.read(25)==1)
{
  T2_tag = true;
  T2 = eepromRead(26,28).toFloat()/100;
  D = eepromRead(29,31).toFloat()/100;
}
if (EEPROM.read(32)==1)
{
  H_tag = true;
  T2 = eepromRead(33,35).toFloat()/100;
}
if (EEPROM.read(36)==1)
{
  T1_T1_szak = true;
  T1T1 = eepromRead(37,39).toFloat()/100;
  T1T2 = eepromRead(40,42).toFloat()/100;
}
```

40. Ábra: EEPROM kiolvasása és értékadása

9.8. EEPROM térkép

A kalibrációs adatokat mindig egy térkép alapján rendezve írom. Ez nagyban meghatározza az adatok konstrukcióját.

Az EEPROM térkép:

EEPROM cím	**Érték**	**Magyarázat**
0	0-1	0: nincs mentett kalibráció, azaz a bootolás után a „Várakozás a GUI kalibrációra!” állapot lesz érvényben 1: van mentett kalibráció, azaz a bootolás után a „EEPROM-ból kiolvasva” állapot lesz érvényben
1	0-1	0: nincs mérési zaj 1: van mérési zaj
2	0-3	0: nincs zavar jellemző

		1: direkt zavar jellemző van 2: T1-en áteresztett zavar jellemző van
3-5	000000-999999	A zavar jellemző T1 időállandója megszorozva 100-zal, 3 egymást követő címen 2-2-2 digitre felosztva.
6	0-1	0: nincs proporcionális tag a szimulálandó szakaszban 1: van proporcionális tag a szimulálandó szakaszban
7-9	000000-999999	A proporcionális tag erősítési tényezője megszorozva 100-zal, 3 egymást követő címen 2-2-2 digitre felosztva.
10	0-1	0: nincs integráló tag a szimulálandó szakaszban 1: van integráló tag a szimulálandó szakaszban
11-13	000000-999999	Az integráló tag időállandója megszorozva 100-zal, 3 egymást követő címen 2-2-2 digitre felosztva.
14	0-1	0: nincs DT1 összetett tag a szimulálandó szakaszban 1: van i DT1 összetett tag a szimulálandó szakaszban
15-17	000000-999999	A DT1 összetett tag T_d időállandója megszorozva 100-zal, 3 egymást követő címen 2-2-2 digitre felosztva.
18-20	000000-999999	A DT1 összetett tag T1 időállandója megszorozva 100-zal, 3 egymást követő címen 2-2-2 digitre felosztva.
21	0-1	0: nincs egytárolós tag a szimulálandó szakaszban 1: van egytárolós tag a szimulálandó szakaszban
22-24	000000-999999	Az egytárolós tag T1 időállandója megszorozva 100-zal, 3 egymást követő címen 2-2-2 digitre felosztva.
25	0-1	0: nincs kéttárolós tag a szimulálandó szakaszban 1: van kéttárolós tag a szimulálandó szakaszban

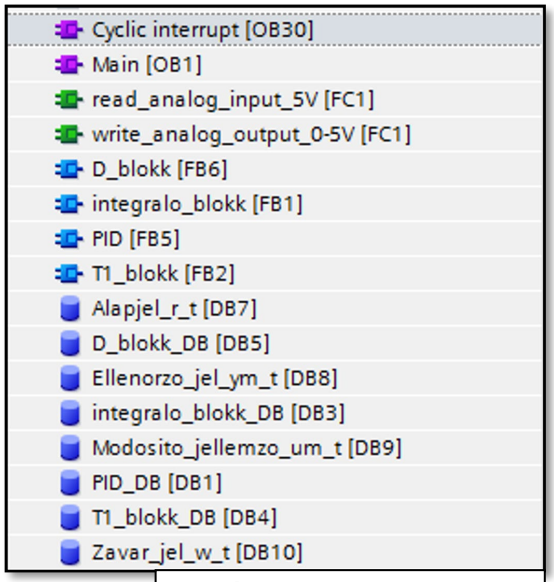
26-28	000000-999999	A kéttárolós tag T2 időállandója megszorozva 100-zal, 3 egymást követő címen 2-2-2 digitre felosztva.
29-31	000000-999999	A kéttárolós tag D tényező értéke megszorozva 100-zal, 3 egymást követő címen 2-2-2 digitre felosztva.
32	0-1	0: nincs holtidős tag a szimulálandó szakaszban 1: van holtidős tag a szimulálandó szakaszban
33-35	000000-999999	A holtidős tag T_h időállandója megszorozva 100-zal, 3 egymást követő címen 2-2-2 digitre felosztva.
36	0-1	0: nincs T1 + T1 összetett tag a szimulálandó szakaszban 1: van T1 + T1 összetett tag a szimulálandó szakaszban
37-39	000000-999999	A T1 + T1 összetett tag T1_1 időállandója megszorozva 100-zal, 3 egymást követő címen 2-2-2 digitre felosztva.
40-42	000000-999999	A T1 + T1 összetett tag T1_2 időállandója megszorozva 100-zal, 3 egymást követő címen 2-2-2 digitre felosztva.

10. MEGVALÓSÍTÁS A PLC-N

A PLC programozására a szabályozási kör teljessé tétele miatt volt szükség. Így lehet letesztelni megfelelően a szakasz szimulátort. A PID (proportional–integral–derivative) szabályozó értékét szaturációval korlátozom. A kielemzéshez szükséges adatokat DB (data block) tárolja egy-egy real típusú adat tömbben. A kimentett adatok:

- Alapjel (r(t))
- Ellenőrző jel (ym(t))
- Módosító jellemző (um(t))
- Zavar jel (w(t))

Tesztelések során kiderült, hogy a közel maximális memóriaterület kihasználásával, real típusú tömbökkel 24000 db adatot tudok elmenteni. A PLC delta t ideje 0.02 sec, így a 6000 db adat 120 sec-es intervallumot fed le. A 41. ábrán látható a PLC-ben létrehozott programblokkok.

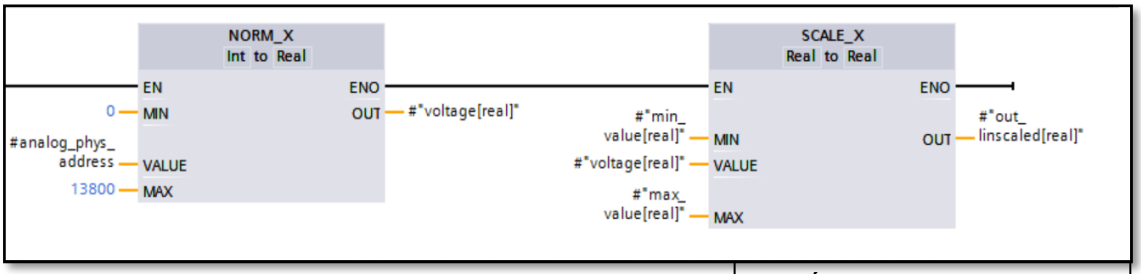


41. Ábra: Programblokkok

10.1. Analóg bemenet olvasása

Az analóg bemeneten az Arduino felől érkező maximum 5V feszültségű jelet olvassuk. Erre a célra egy FC (Function) a megoldás. Az FC el van helyezve a ciklikus interrupt-ban és a kimenete állítja be a bemeneti értéknek létrehozott tag-et. A funkcióban olvassuk az analóg bemenet fizikai címén lévő értéket. NORM_X matematikai átalakítással 0-1 közötti értéket kapunk. és ezt SCALE_X művelettel skálázzuk minimum és maximum érték közé, ez a mi esetünkben 0-5 volt. A TTL jelszintje 5V, míg a PLC jelszintje 10V. A beolvasásnál a maximálisan beolvasható feszültség digitális megfelelőjét, a 27648-at korlátozni kellett 13800-ra.

A 42. ábrán látható az analóg beolvasás funkciója.



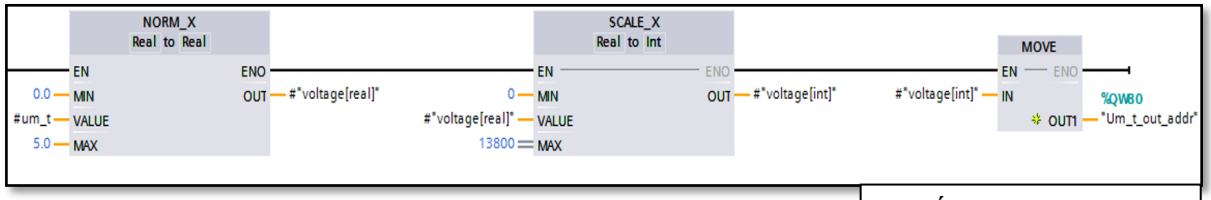
42. Ábra: Analóg beolvasás

Ez a funkció kétszer van meghíva az interrupt-ban: egyszer az Arduino kimenetének értékét, másodszor pedig a zavar jellemző értékét olvassuk be.

10.2. Analóg kimenet írása

Az analóg kimenetnél is, a bemenet olvasásához hasonlóan korlátozni kell a maximum 10V kiadható feszültséget 5V-ra. Itt különösen fontos odafigyelni erre, mert kárt okozhat az Arduino-ban, ha túlfeszültség éri. A funkció bemenetére a módosító jellemző van csatolva, ami szaturálva van 0-5 volt közé. Ezt az értéket 0-1 közé normalizálom, a normalizált értéket skálázom 0-13800 közé. Az így kapott 0-5 voltnak megfelelő digitális értéket move paranccsal kiíratom a PLC analóg kimenetének fizikai címére.

A 43. ábrán látható az analóg kimenet írása.

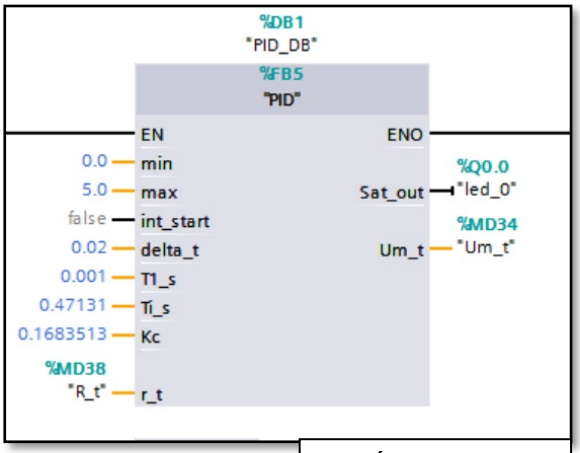


43. Ábra: Analóg kiírás

10.3. Szabályozó

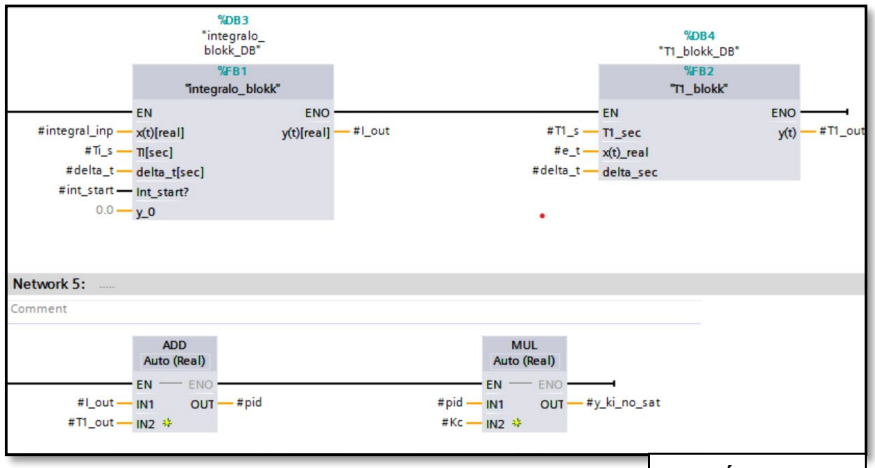
A szabályozót egy ciklikus interrupt-ban helyezem el, melynek a ciklusideje (0.02 sec) nagyobb kell, hogy legyen, mint az Arduino ciklusideje (0.015 sec), különben felesleges mintavételezések történnének és redundancia lenne az értékek között.

A PID kalibrálásának lehetőségét a PID FB (Function Block) bemenetein keresztül valósítottam meg. A PID FB a 44. ábrán látható.



44. Ábra: PID FB

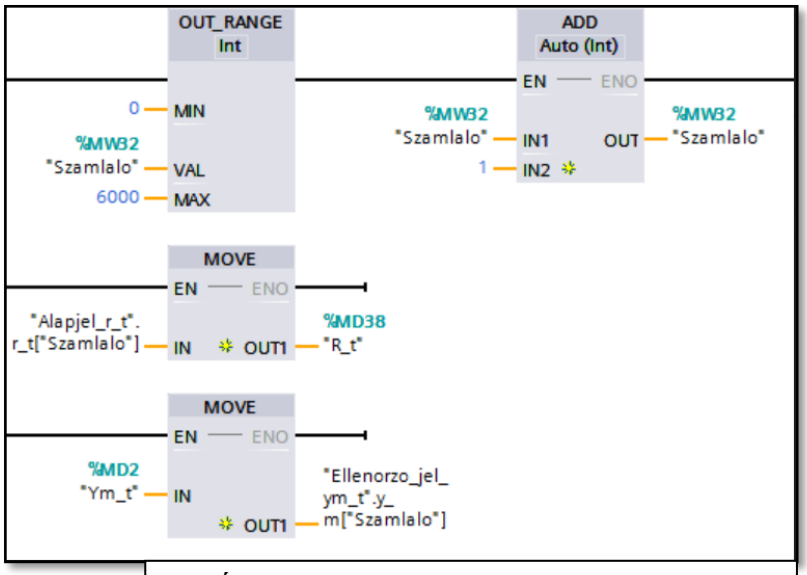
A funkció blokkon belül, találhatóak a PID alkotói: a proporcionális, az integráló és deriváló tagok funkció blokkjai. (45. ábra)



45. Ábra: PIT

10.4. Adatok kimentése DB-be

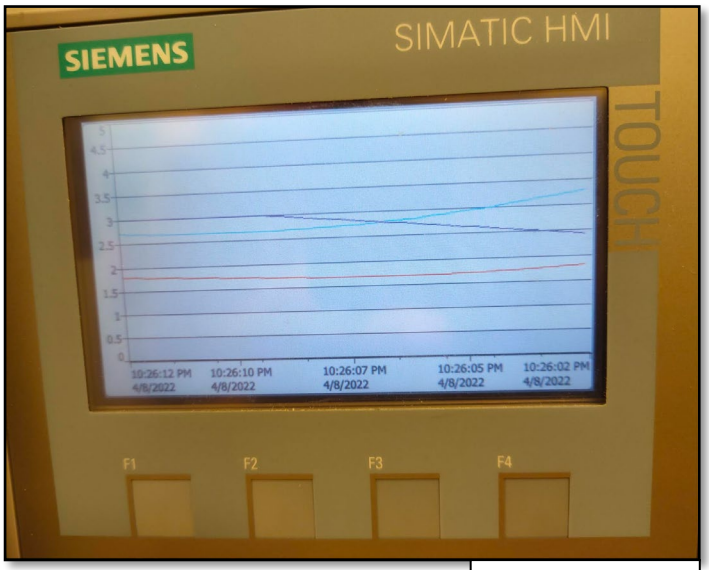
Az adatokat működés közben mentem ki a PLC-re való feltöltés után az első ciklustól a 6000-ik ciklusig. Létrehoztam erre egy változót, ami elszámol 6000-ig, illetve a move paranccsal (46. ábra) az értékeket az adatblokkban tárolt tömbbe írom bele.



46. Ábra: Számolás és az ellenőrző jel feltöltése

10.5. Adatok megjelenítése HMI-n

A négy megfigyelt jel PLC-s tag-jét összekapcsoltam a HMI-ben létrehozott 4 tag-gel. Ezeket a tageket ciklus időnként frissítve idődiagrammal jeleztem ki, hogy a szakasz viselkedését élesben lehessen ellenőrizni. Tesztelés alatt a HMI-ről készített kép látható a 47. ábrán.



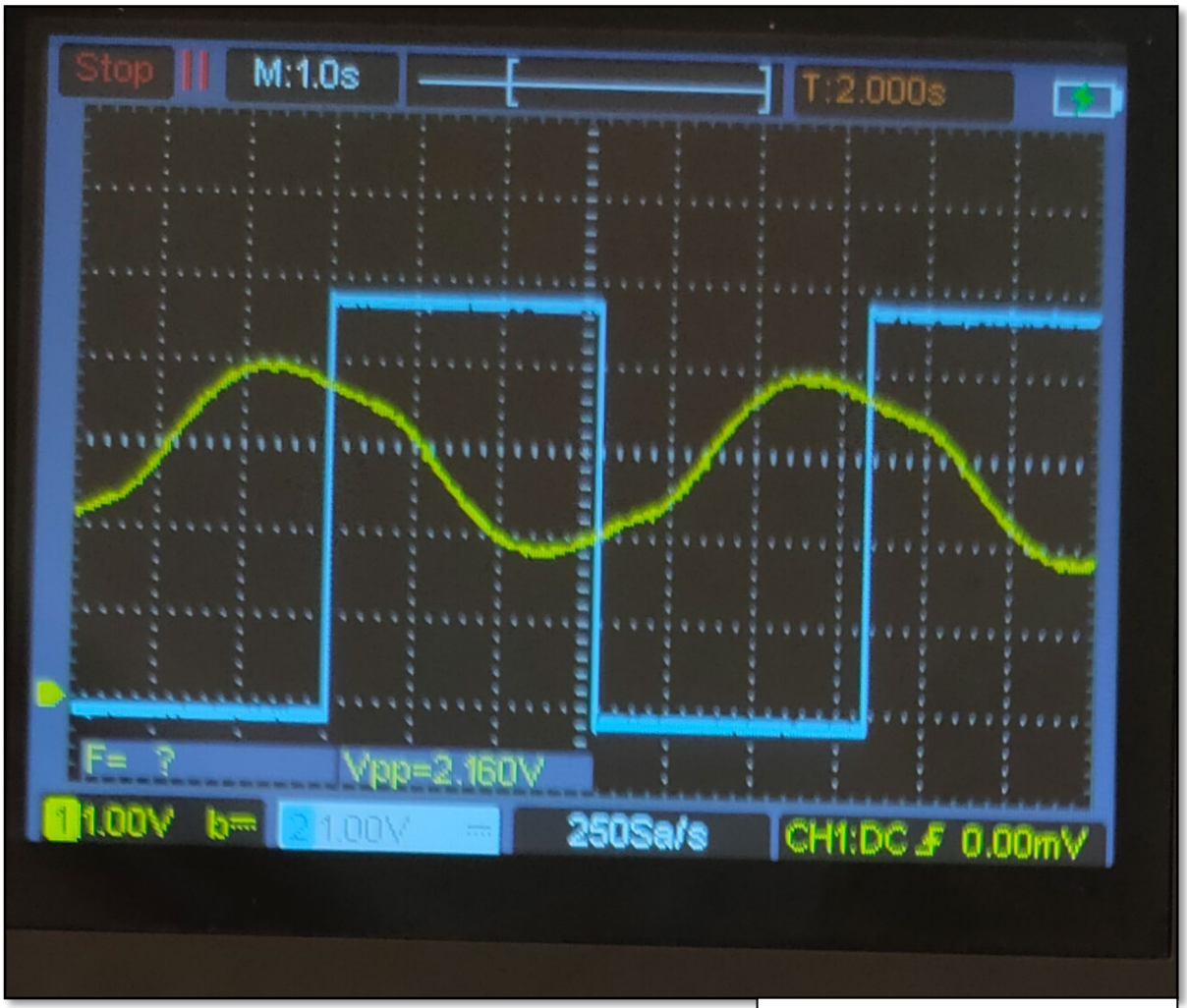
47. Ábra: HMI

11.A SZAKASZ SZIMULÁTOR TESZTELÉSE

A szakasz szimulátor megfelelő működését 2 összetett szakasszal a HPT3-mal és a T2T1-gyel teszteltem le. Két tesztet csináltam, mindkét szakasz esetében. Az első tesztnél zavar jel nélkül a másodiknál azonban zavar jel hozzáadásával. A szakaszokat a Matlab szoftverben identifikáltam és a szabályozó értékeit a Simulink szoftverben PID tuner-rel határoztam meg.

11.1. Ciklusidő ellenőrzése

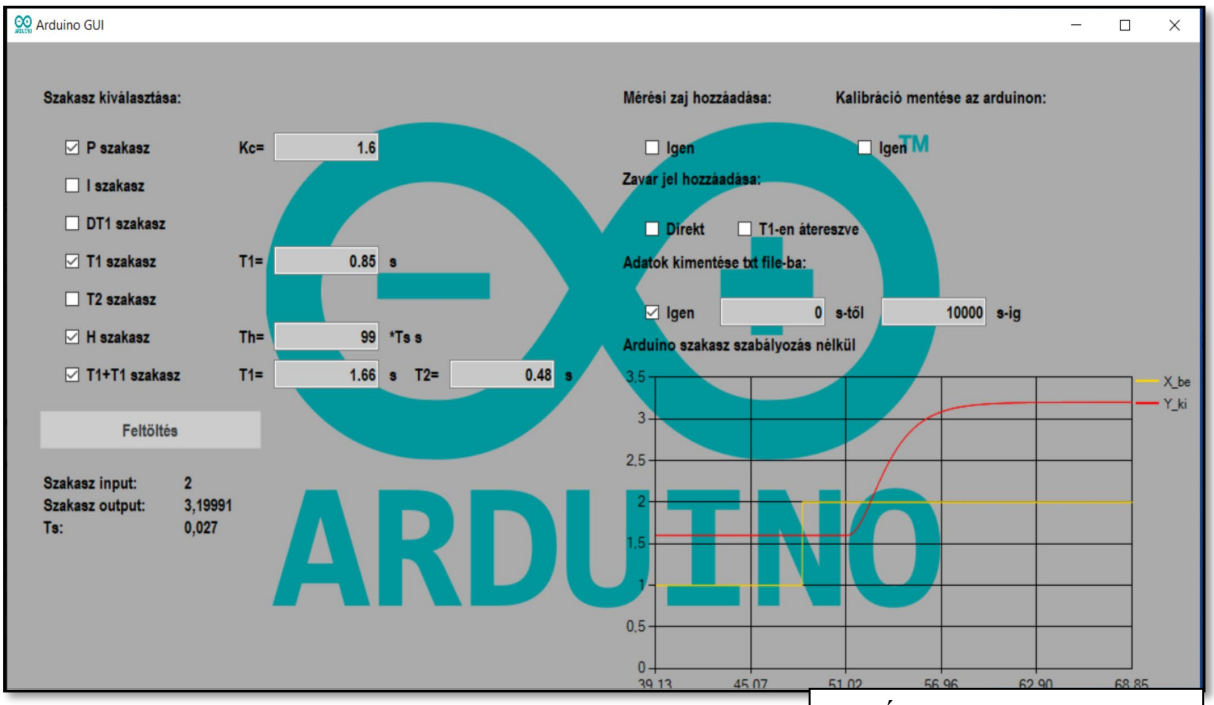
A ciklusidőt egy multiméterrel tudtam ellenőrizni, amire rákötöttem egy digitális kimenetet és a DAC kimenetét. A delta t idejét kiírattam az LCD panelra, ez végig 0,0185 sec volt a tesztelés alatt. Tehát a jeleim periódus ideje 334*delta t, ami 6,179 sec. A 48. ábrán látható az, hogy ez megfelel a valós adatnak, ez kiszámolható a multiméter által mutatott képről. Az időfelosztás 1 sec és a periódus idő vége körülbelül 6,2 DIV (elválasztó) értéknél van.



48. Ábra: Multiméter kijelzője

12.HPT3

Az első tesztelésre összeállított szakasz holtidőt, arányos tagot és három darab egytárolós tagot tartalmaz. A szakasz kimenetéhez még egy zavar jelet is hozzáadok, aminek van egytárolós tag jellemzője. A GUI tanári módban, önellenőrzés céljából a 49. ábrán látható.



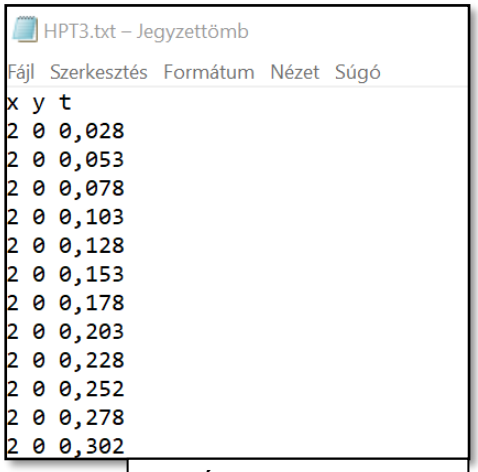
49. Ábra: PT3H kalibációja

A szakasz kimenete a chart alapján nem lépi túl az 5V-ot, ami megfelel a teszteléshez.

Az értékek:

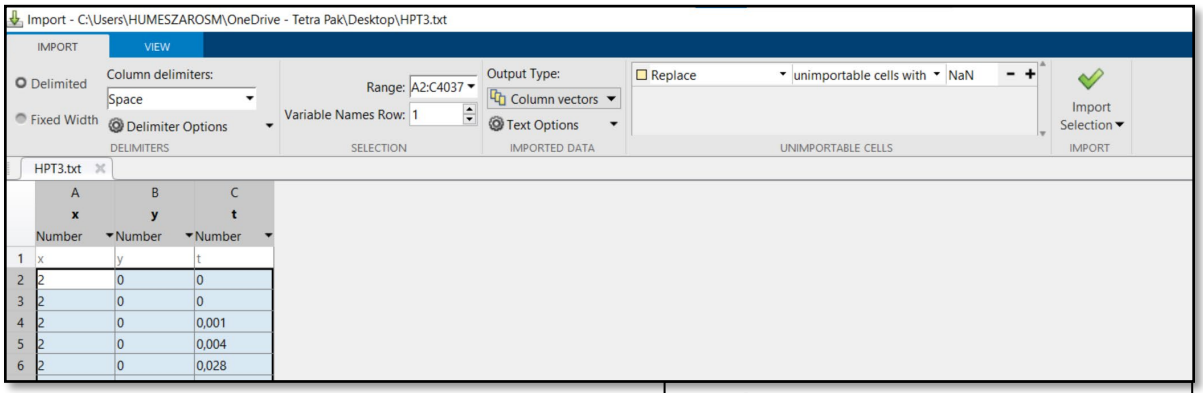
- Proporcionális tag erősítése 1,6
- Első egytárolós tag időállandója 0,48 sec
- Második egytárolós tag időállandója 0,85 sec
- Harmadik egytárolós tag időállandója 1,66 sec
- Holtidő 99*delta t azaz tanár üzemmódban 2,67 sec (99*0,027)

Az eltelt időt, a kimenetet és a bemenetet a GUI segítségével kimentettem egy .txt fájlba (50. ábra).



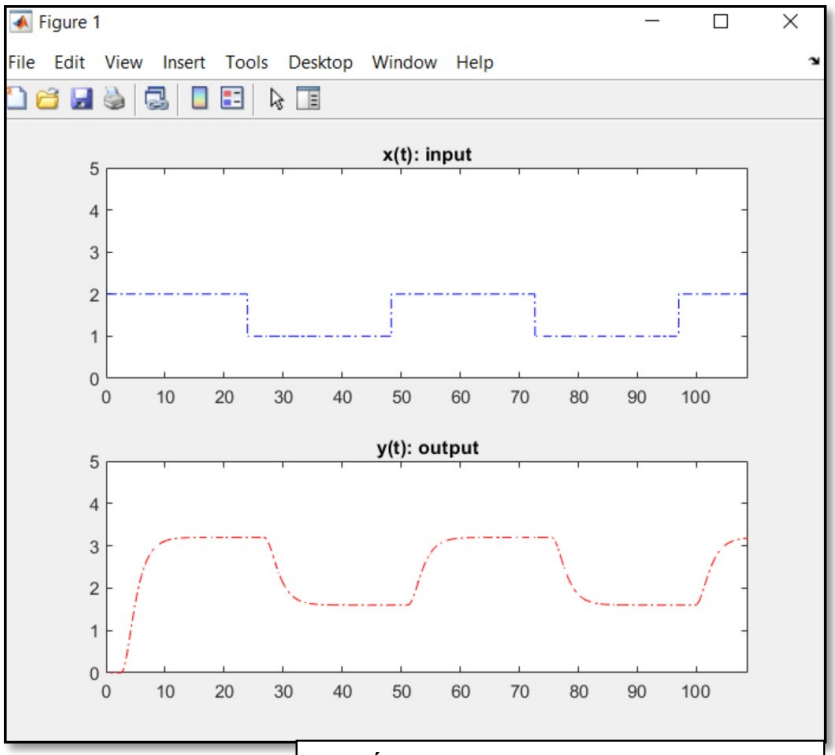
50. Ábra: HPT3.txt fájl

Az így kimentett adatokból identifikálni tudom a szakaszt, ezt a Matlab szoftverével oldom meg. Első lépésben a HPT3.txt fájl tartalmát beolvasom 3 oszlopvektorba (51. ábra).



51. Ábra: Oszlopvektorok importálása

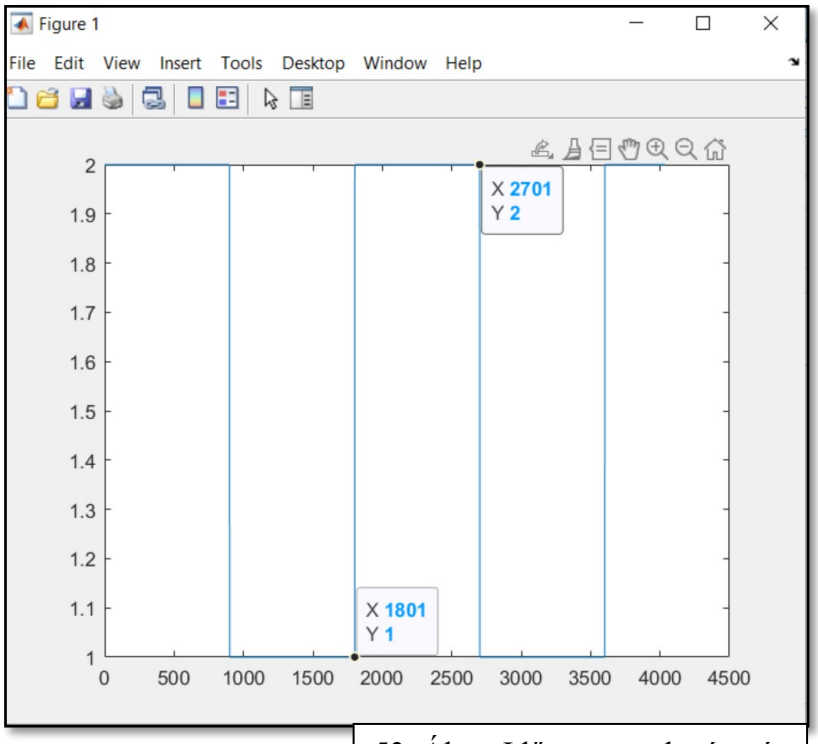
A kapott oszlopvektorokat kirajzoltatom a plot parancs segítségével (52. ábra).



52. Ábra: Beolvasott jelek ellenőrzése

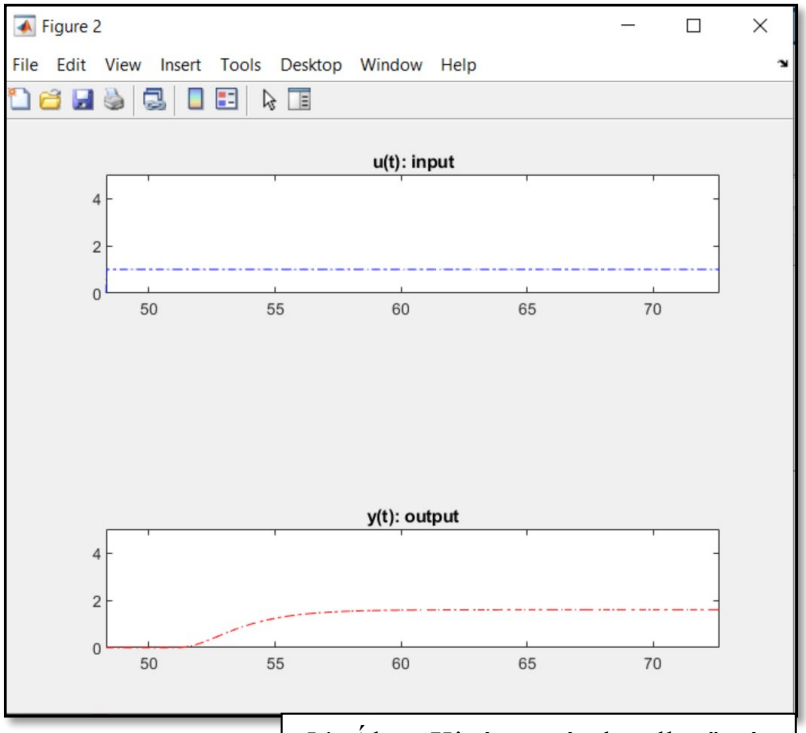
A beolvasott jelalakok megfelelőek, látható a holtidő, az erősítés és a három egytárolós tagra jellemző viselkedés.

Következő lépésben kiválasztok egy felutó szakaszt, ami a második periódus első fele lett ebben az esetben. Ennek a szakasznak meghatározom az X(t) jel felhasználásával a kezdeti időpontját és végérték időpontját (53. ábra).



53. Ábra: Időpont meghatározás

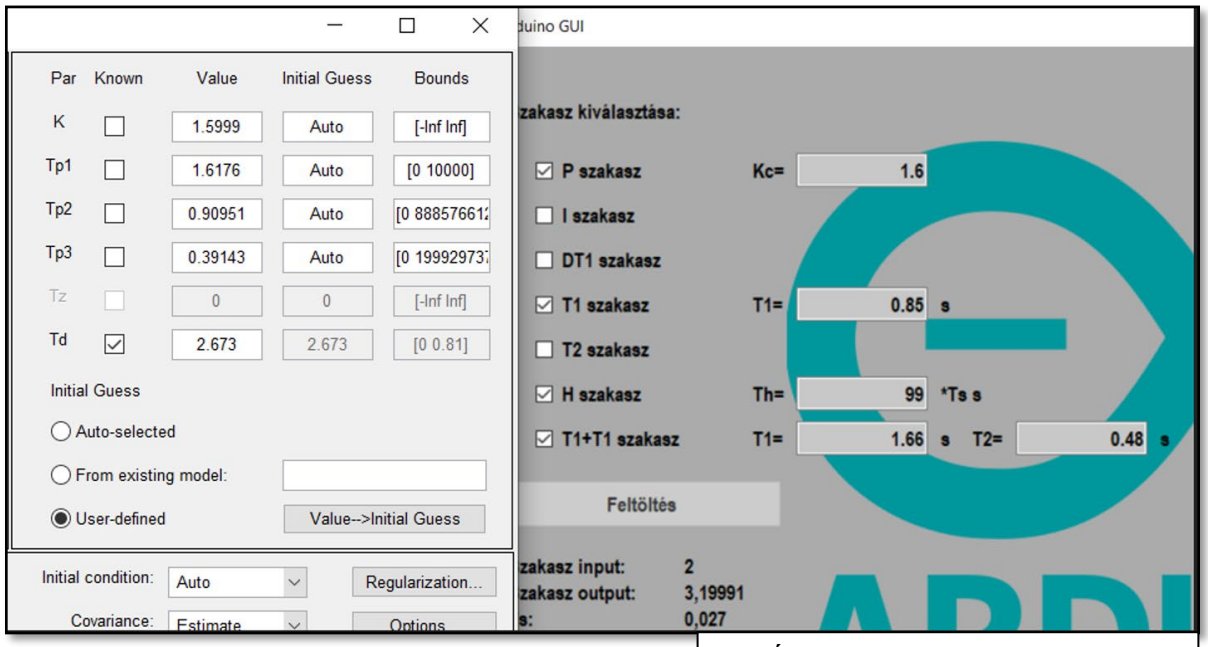
A képen az látható, hogy ez a felfutó szakasz az 1801-ik elemtől tart a 2071-ik elemig. Ennek megfelelően kivágom és eltolom 0 időpontba, valamit az érték alapján letolom 0-ba a jelet. Ezt mutatja be az 54. ábra.



54. Ábra: Kivágott részlet ellenőrzése

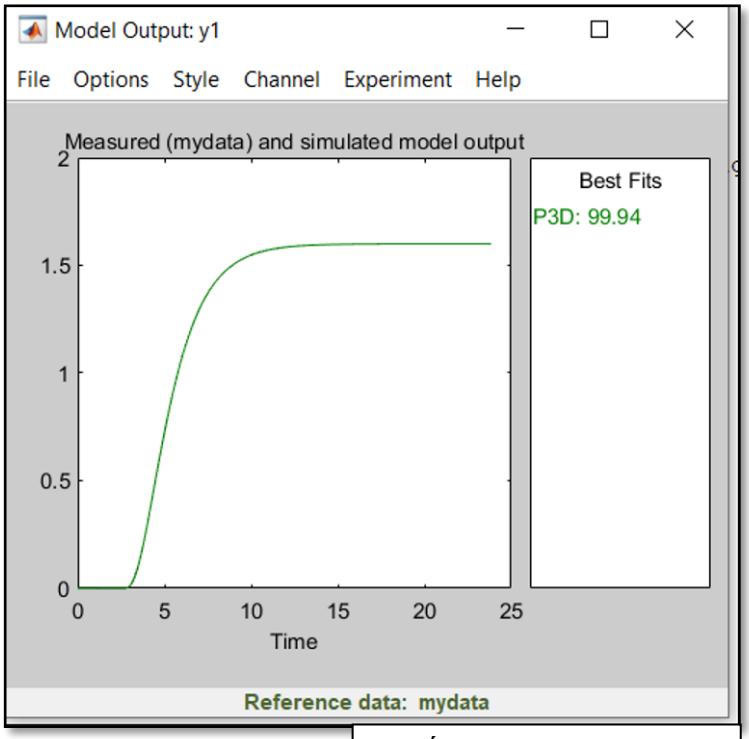
A kirajzolt idődiagram, adja vissza azt amit vártam: felfutó bemenetre 0 időpontban 0 értékkel kezdődő válaszfüggvényt.

Varga Árpád tanárúr segítségével az identifikálás során rá kellett jönnünk, hogyha a szakasz tartalmaz holtidős tagot, akkor a Matlab ezen funkciója nem működik megfelelően. Az identifikálás a holtidőt fixre állítva azonban már megfelelően működik. Az identifikálás eredménye 55. ábrán látható.



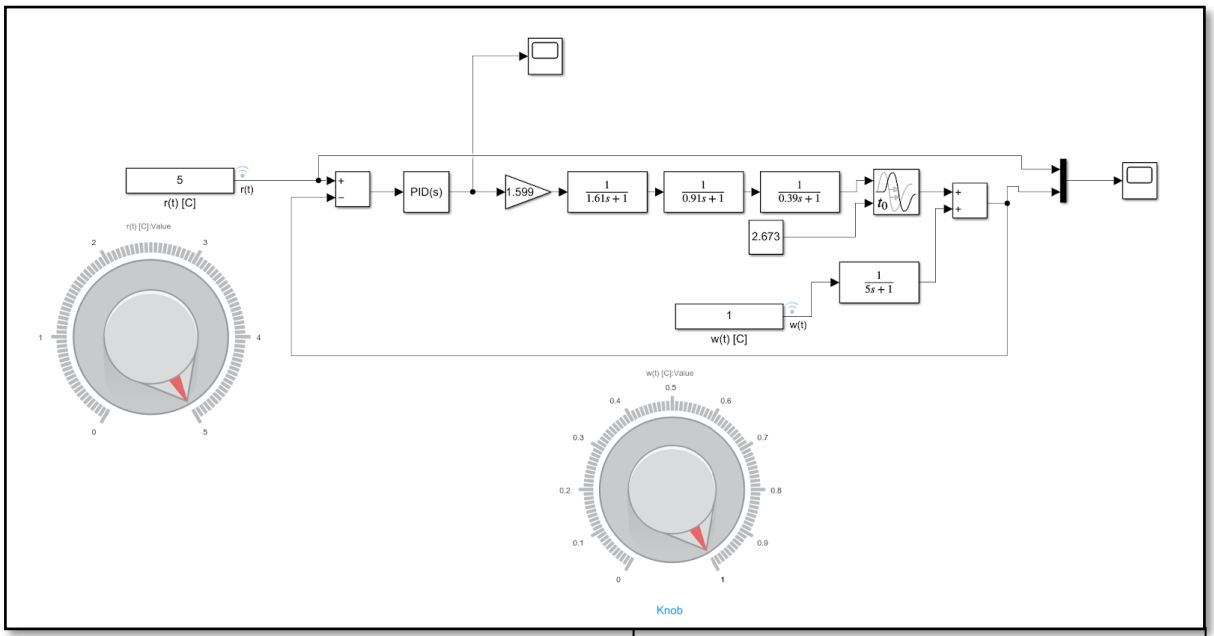
55. Ábra: Identifikálás és ellenőrzés

A kapott eredmény csak kis mértékben tér el a beállítotthoz képest, tehát megfelelően sikerült a beazonosítás. Rajzolva látható a kapott eredmény az 56. ábrán.



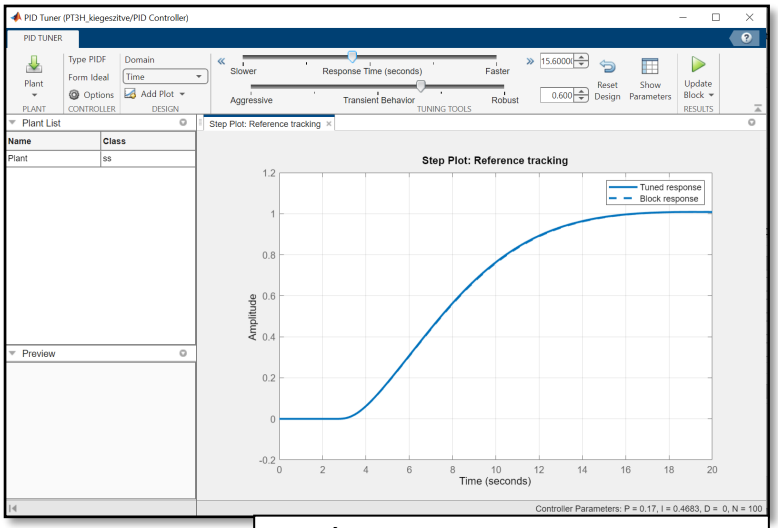
56. Ábra: Szakasz kimenet

A következő lépés a PID szabályozó kalibrációs értékeinek meghatározása. Ehhez a Simulinkben kell létrehozni a kapcsolást (57. ábra) és ott a PID tuner segítségével megkapom a szükséges adatokat.



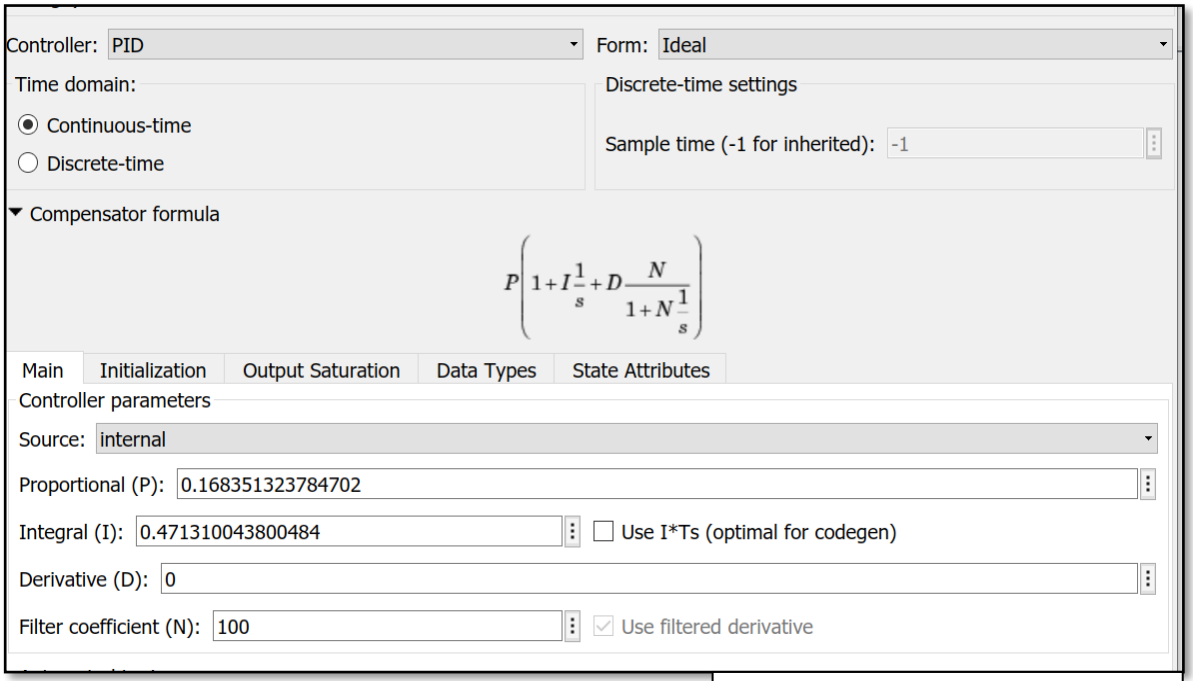
57. Ábra: Szabályozó kör a Simulinkben

A PID Tuner Conroller-t megnyitva finomhangolhatjuk a szabályozó gyorsaságát vagy a túllövés csökkentését (58. ábra).



58. Ábra: PID Tuner finomhangolása

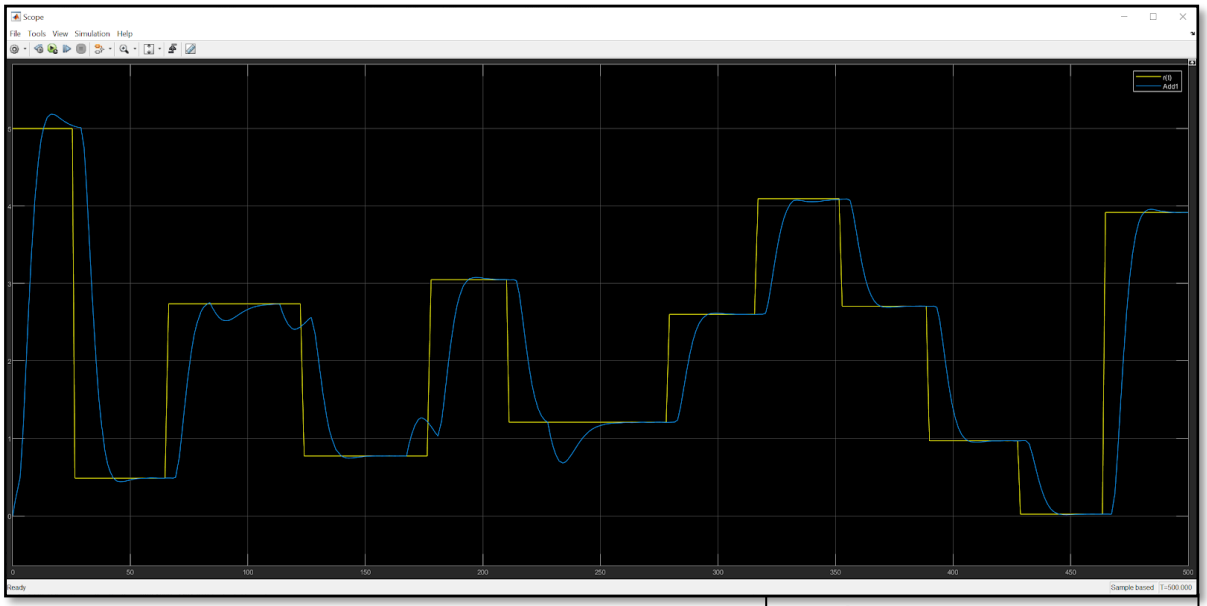
A szabályozott jellemző a holtidő miatt eléggé lassan csupán 18 sec alatt áll be a kívánt értékre. A PID kalibrációjához szükséges információk megtalálhatóak az 59. ábrán.



59. Ábra: PID kalibrációs értékei

- A PID proporcionális értéke 0,16835
- Integrálási időállandója pedig $\frac{1}{0{,}47131} = 2{,}121745\ sec$
- Differenciáló tag pedig nem szükséges a szabályozóhoz ennél a szakasz összetételnél

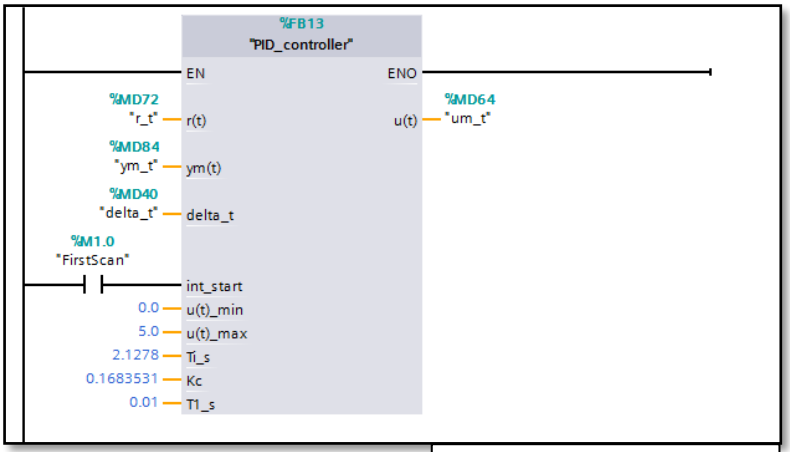
Az 60. ábrán a Simulink szimulációja látható:



60. Ábra: Simulink szimuláció

A szimuláció alapján az első részen látható nagyobb túllövés. A harmadik és ötödik szakaszban látható, hogy a szabályozó szépen kiküszöböli a zavar jel értékét.

A kapott eredmények alapján a PLC programjában beállítottam a PID szabályozót (61. ábra). Az alapjelet egy előre feltöltött tömb adja, mert a PLC mindkét analóg bemenete foglalt és nincs bővítőkártya a Számítógépes Folyamatautomatizálás laborban.



61. Ábra: PID beállítása

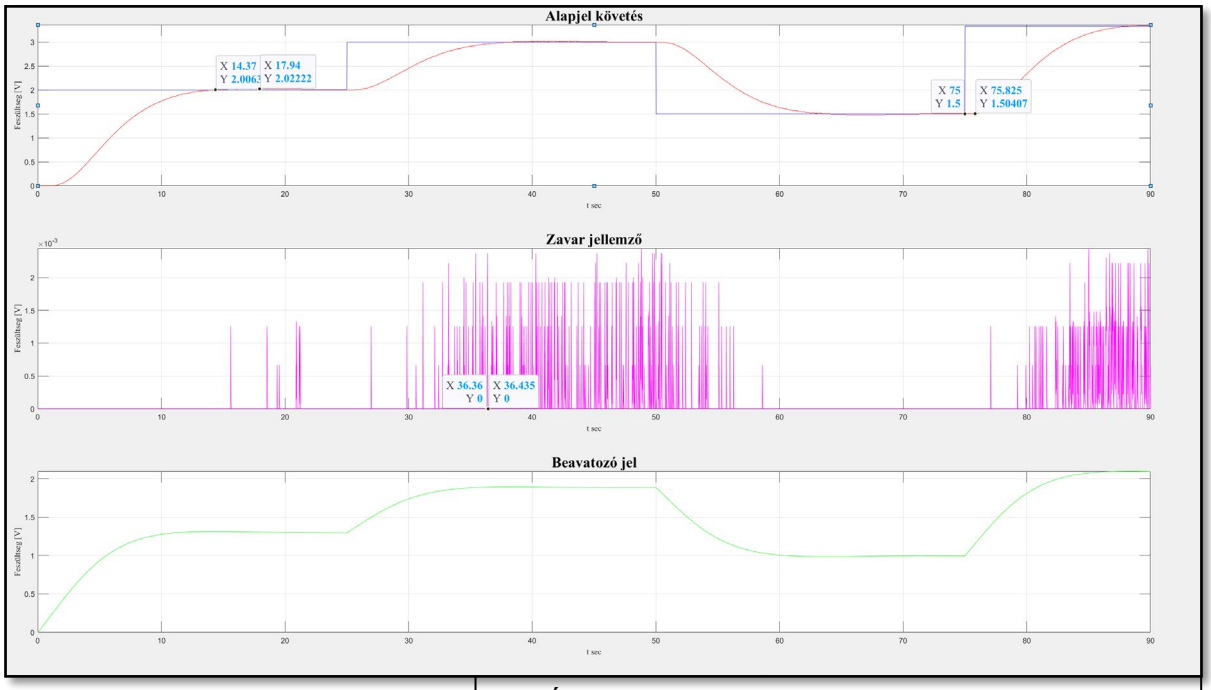
Elindítottam a folyamatot, majd 6000 ciklussal később a datablokkok, amiket létrehoztam feltöltődtek. (62. ábra)

	Name	Data type	Start value	Snapshot	Monitor value
1	Static				
2	Ym_t	Array[0..6000] of R...			
3	Ym_t[0]	Real	0.0	0.0	0.0
4	Ym_t[1]	Real	0.0	0.0	0.0
5	Ym_t[2]	Real	0.0	0.0	0.0
6	Ym_t[3]	Real	0.0	0.0	0.0
7	Ym_t[4]	Real	0.0	0.0	0.0
8	Ym_t[5]	Real	0.0	0.0	0.0
9	Ym_t[6]	Real	0.0	0.0	0.0
10	Ym_t[7]	Real	0.0	0.0	0.0
11	Ym_t[8]	Real	0.0	0.0	0.0
12	Ym_t[9]	Real	0.0	0.0	0.0
13	Ym_t[10]	Real	0.0	0.0	0.0
14	Ym_t[11]	Real	0.0	0.0	0.0
15	Ym_t[12]	Real	0.0	0.0	0.0
16	Ym_t[13]	Real	0.0	0.0	0.0
17	Ym_t[14]	Real	0.0	0.0	0.0
18	Ym_t[15]	Real	0.0	0.0	0.0
19	Ym_t[16]	Real	0.0	0.0	0.0
20	Ym_t[17]	Real	0.0	0.0	0.0
21	Ym_t[18]	Real	0.0	0.0	0.0
22	Ym_t[19]	Real	0.0	0.0	0.0
23	Ym_t[20]	Real	0.0	0.0	0.0
24	Ym_t[21]	Real	0.0	0.0	0.0
25	Ym_t[22]	Real	0.0	0.0	0.0

62. Ábra: Ym(t) értékek

Az adatokról snapshot-ot készítve másolható értékeket kaptam, így át tudtam ezeket tenni egy excel fájlba.

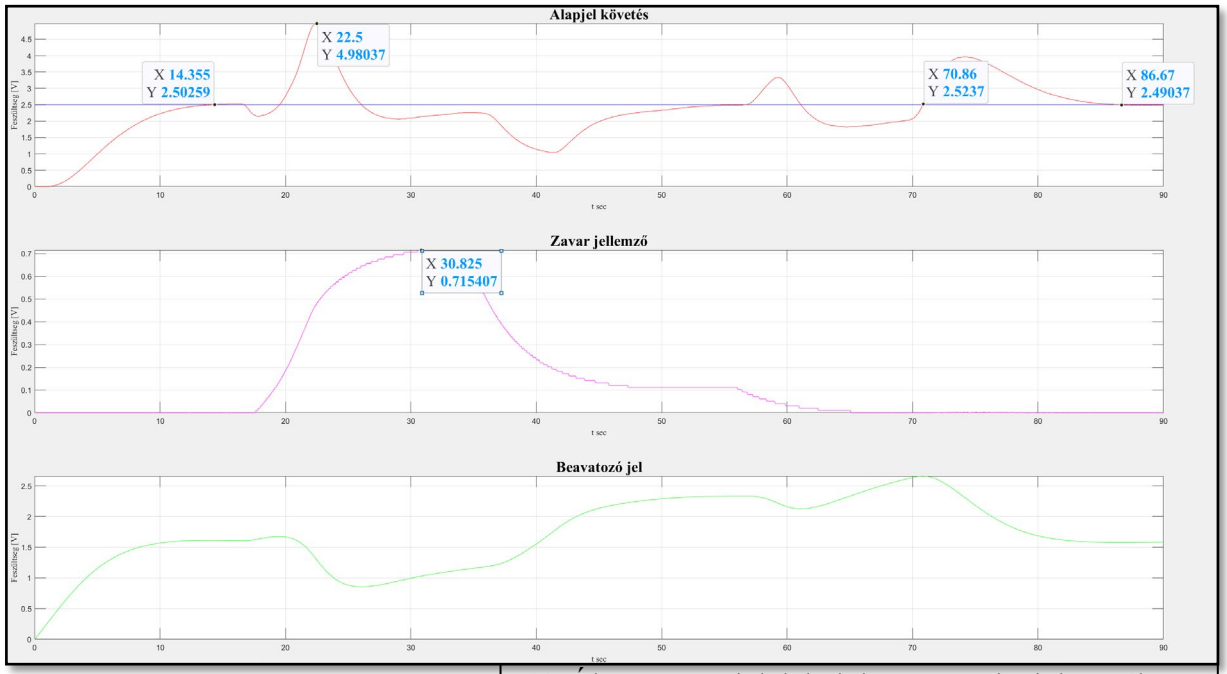
Matlab szoftverben beimprotálom és plotolom az excel fájban lévő oszlopokat. Így kapok egy valós képet arról, hogy a PLC valóban tudja befolyásolni a szakasz szimulátort. Az első teszt változó alapjellel és zavar jel 0 értékével készült. Ez látható az 63. ábrán.



63. Ábra: PT3H jelalakok konstans zavar jel esetében

Az eredmény alapján a zavar jel színtjét zaj érte. A zaj maximális feszültség szintje: $2{,}5 * 10^{-3} = 0{,}0025\,V$. A zavar jellemző ebben az esetben inkább tekinthető zajnak. A zaj jelalakjában a csúcs le- és felfutó éle közötti időtartam $36{,}435 - 36{,}36 = 0{,}075\,sec$. Ez annyira kis idejű kiugrást eredményez, hogy a szabályozozó megfelelően lekezelte. A vártnak megfelelően lassan, de felveszi az alapjel értékét a szakasz szimulátor kimenete. Az alapjel követés diagramot vizsgálva a harmadik felfutó jelnél $75{,}825 - 75 = 0{,}825\,sec$ szükséges a szabályozó számára, hogy reagáljon az alapjel változására. Az első felfutásnál, ahol a legnagyobb túllendülés mérhető, ez az érték $2{,}022 - 2 = 0{,}022\,V$. A 2V-os alapjelhez képest ez az érték +1,1%.

Következő teszt eredménye azt mutatja, hogyan viselkedik a szakasz konstans alapjelre, változó zavar jellemző mellett. Ez látható az 64. ábrán.



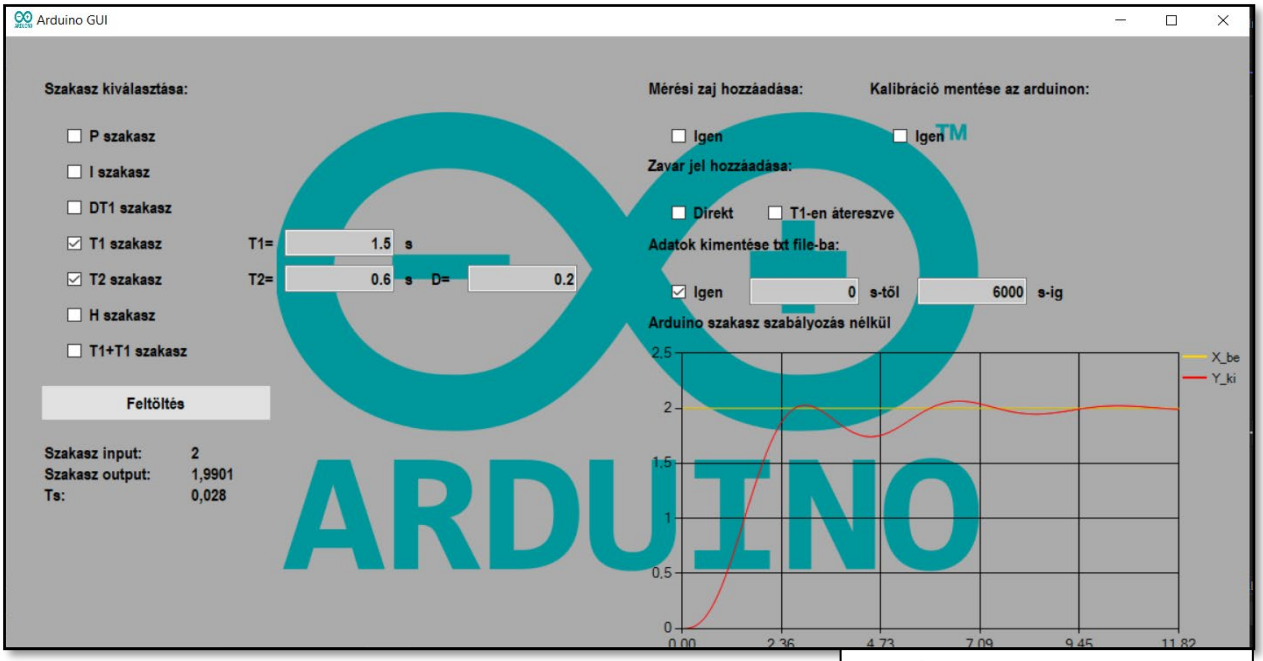
64. Ábra: PT3H jelalakok konstans alapjel esetében

Az ábrán az látható, hogy a szabályozó küzd azért, hogy az alapjel értékére szabályozza a szakasz kimenetét. A szabályozó lassúsága miatt az alapjel elérése ebben az esetben 15 sec-et is igénybe vesz. Ez a időmérték a holtidős tényező miatt ilyen magas. Ha nem lenne holtidő a szabályozandó szakaszban, sokkal gyorsabb lenne a reakció idő. A zavar jellemzőt 0V-ról indítva feltekertem 0,715 voltra. A zavar jellemző meredek emelkedésének hatására a szabályozott jellemző elérte a szaturációs értéket. A szabályozónak köszönhetően a szakasz kimeneti értéke ugyanabban az ütemben, ahogy növekedett, le is csökkent.

13.T1T2

A szakasz szimulátor teszteléséhez a második szakasz összetétele az egytárolós tag és a valódi kéttárolós tag sorba kapcsolása. A lépések ugyanazok, mint az előző HPT3 összetételű szakasznál.

Kalibráció látható a 65. ábrán.



65. Ábra: T1T2 kalibrációja

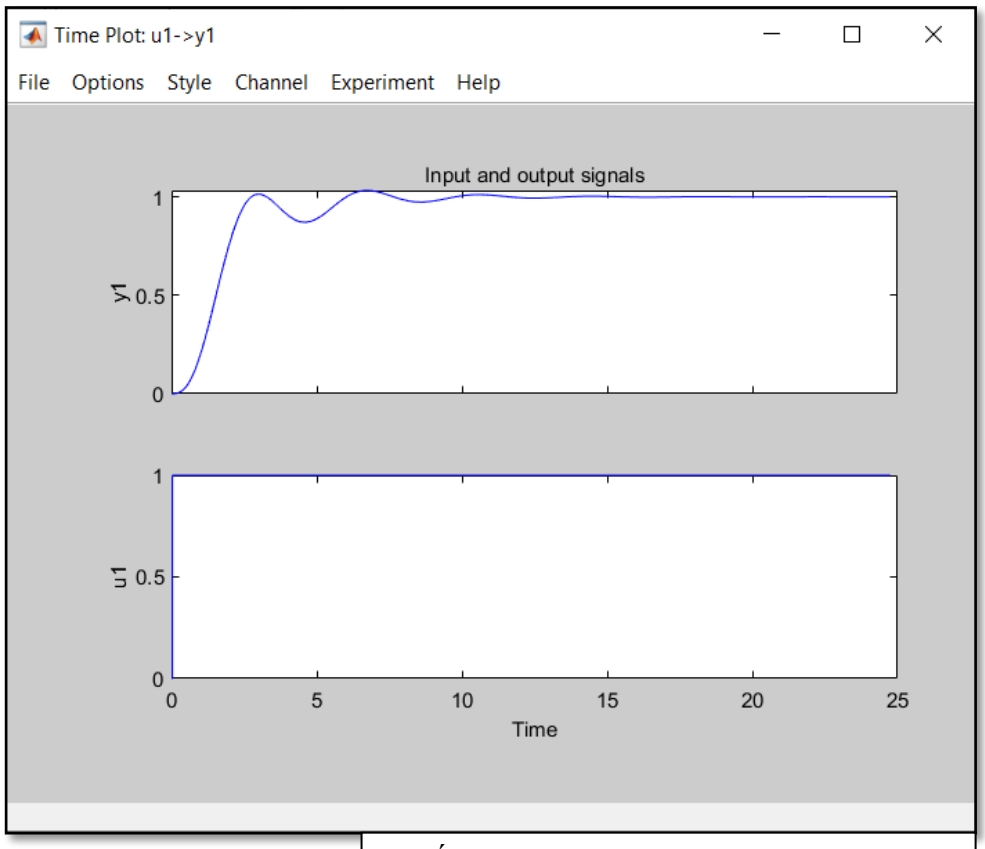
A szakasz kalibrációja:

- T1 időálladója 1,5 sec
- T2 időállandója 0,6 sec
- D tényező 0.2

A tanári üzemmódban a GUI-n elhelyezett Data chart-on látható, hogy a kalibrációs értékek megfelelnek a teszteléshez.

A beolvasott adatokból eltolást és kivágást követően megnyitjuk a szakasz identifikáló applikációt.

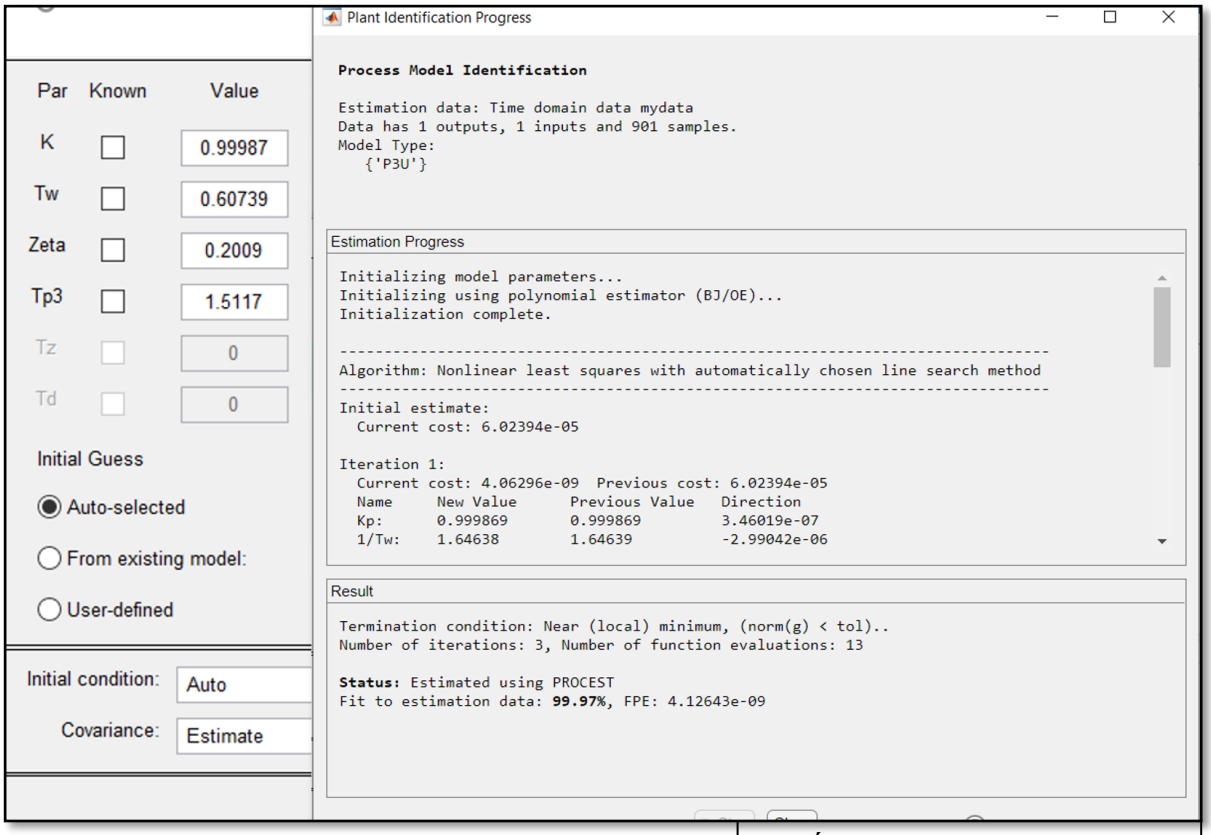
Az identifikáláshoz beolvasott jel alakja a 66. ábrán látható.



66. Ábra: Identifikáláshoz beolvasott jelalak

Az ábrán jól látható, ahogy a T2-re jellemző lengés megjelenik a jelalakon. Ugyanilyen lengést láthattunk a GUI felületén még a kalibrációs adatok megválasztásánál.

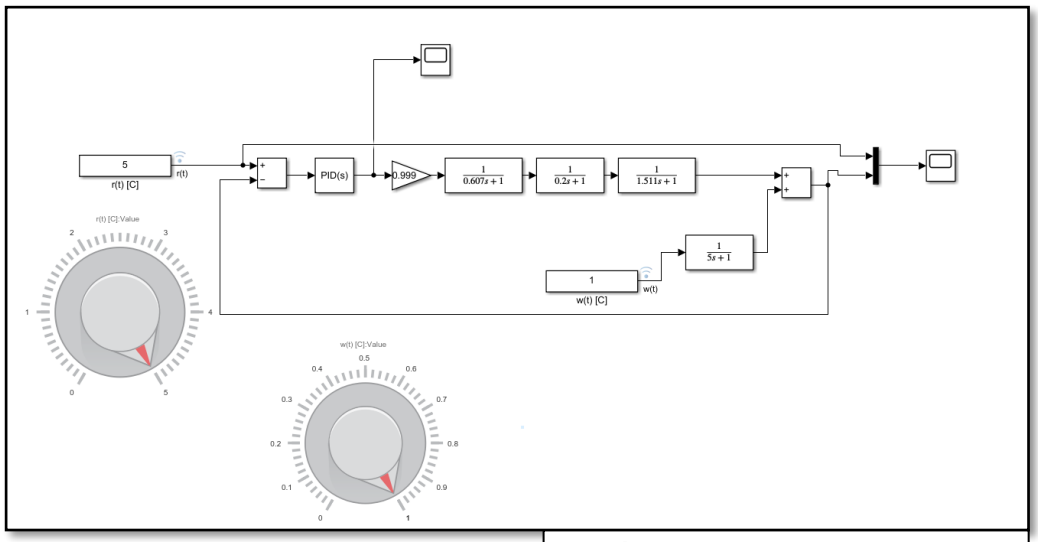
Az identifikálás eredménye 99,97% pontosságú (67. ábra)



67. Ábra: Identifikálás eredménye

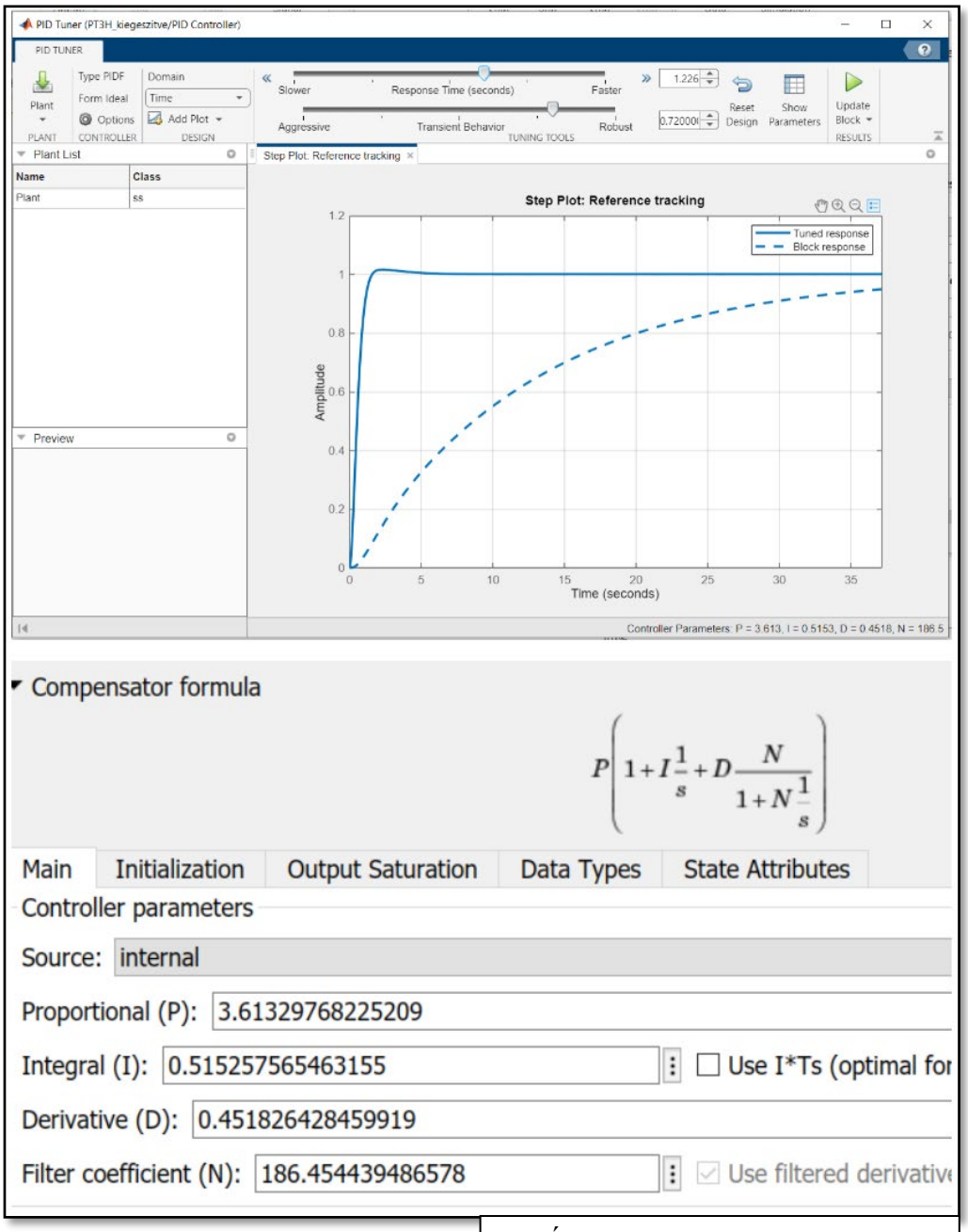
A kapott eredmény ezred pontossággal visszaadta az eredeti kalibrációs értékeket.

A Simulink-ben ezt a kapcsolást is létre kellett hozni ahhoz, hogy a PID Tuner segítségével megkapjam a szükséges szabályozó paramétereket. (68. ábra)



68. Ábra: Simulink T1T2 kapcsolás

A PID Tuner finomhangolása és a kiszámított eredmények a 69. ábrán láthatóak.



69. Ábra: PID Tuner számított adatai

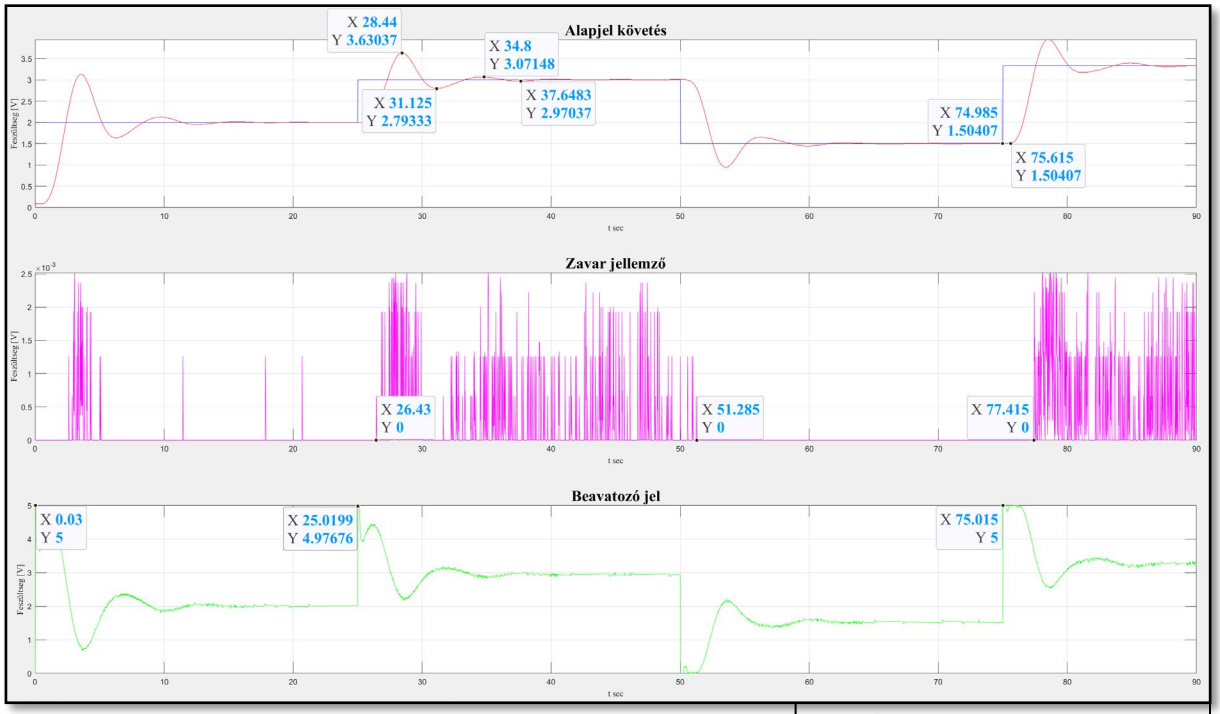
A szabályozó értékei a következők:

- Proporcionális tag erősítése 3,6132
- Integráló tag időállandója $\frac{1}{0{,}515257} = 1{,}753$ sec
- Differenciáló dag időállandója 0,4518 sec
- Egytárolós tag időállandója $\frac{1}{186{,}45} = 0{,}117$ sec

Ezeket az értékeket állítottam be a PLC programjában.

A futtatást követően lementettem az adatblokkok tartalmát.

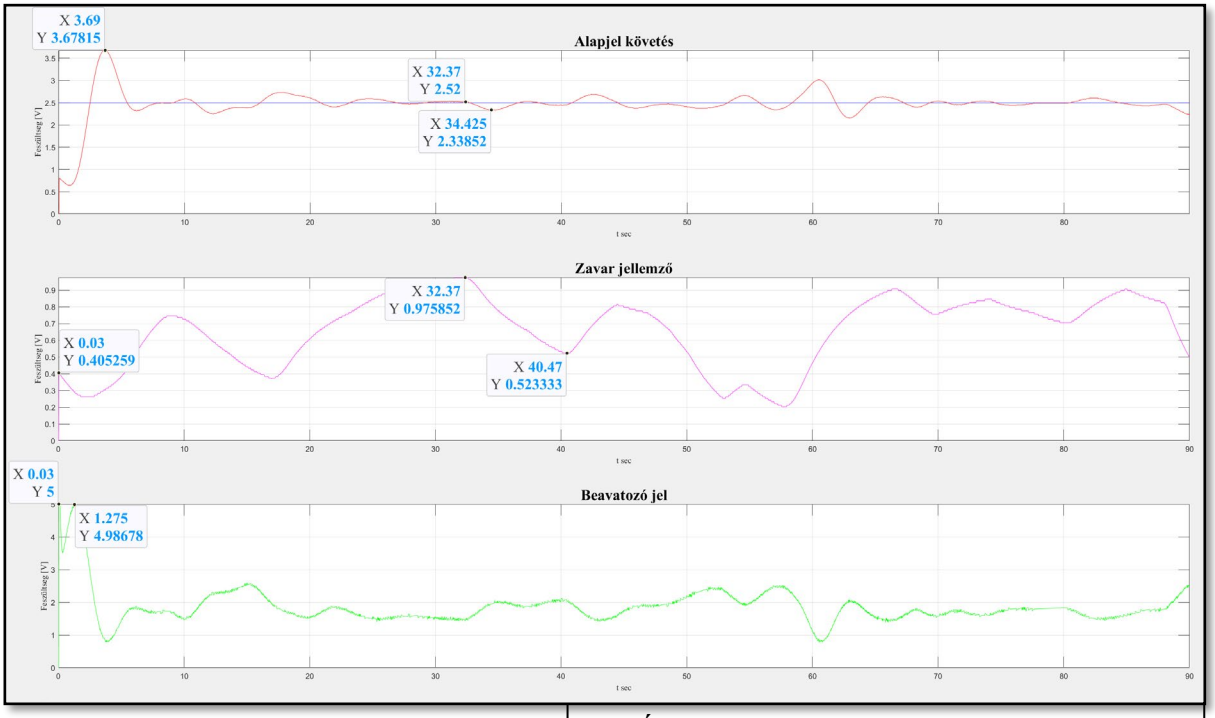
Az első teszt eredménye látható a 70. ábrán. Ez a teszt az volt, amikor nincs zavar jellemző, de változik az alapjel.



70. Ábra: T1T3 szabályozása

Az alapjel követésénél látható a csillapodó tendenciájú lengés, ami végül ráül az alapjelre. A lengés első csúcsértéke 3,63V, a második csúcsértéke 2,79V, a harmadik csúcsértéke 3,07V. A zavar jellemzőnél ugyanazt tapasztalom mint a PT3H szakaszösszetétel esetében, zaj jelenik meg. A zaj viselkedése olyan, mintha a periódus ideje 25 sec lenne. 25 sec és 50 sec között van zaj, 50 sec és 75 sec között nincs zaj. A beavatkozó jelen azt láthatjuk, hogy működik az 5V-os szaturáció. Enélkül 3 időpontban (0,03 sec, 25,01 sec, 75,01 sec) nagyobb feszültséggel szabályozott volna a PID.

A 71. ábrán látható a T1T2 szakasz viselkedése, amely alapjel változás nélküli, zavarjellemző jelenléte mellett.



71. Ábra: T1T3 szabályozása zavar jel mellett

Az ábrán az látható, hogy a szabályozó szépen kompenzálja a zavar jellemző folyamatos behatását, mivel a szakasz szimulátor kimenete tartja az alapjel nagyságának közelségét. A szabályozott jellemző maximális túllendülése az első felfutásnál történik, amelynek értéke 3,69 V. A szaturáció ennél a tesztnél is megfelelően működik. A beavatkozó jel 0,03 sec-nél és 1,275 sec-nél szaturáció nélkül nagyobb lenne mint 5V. A zavar jellemzőt 0,97V-ról csökkentem egészen 0,52V-ra 32,37-40,47 sec között. Ebben az időtartományban az alapjel követés diagramon megfigyelhető, hogy a maximális kitérés $2{,}52 - 2{,}338 = 0{,}182\,V$, a feszültség csökkenés abbamarad 40,47 sec-nél és a feszültségi szint elkezd visszaállni az alapjel értékre.

14.Összefoglalás

Az elkészült szakasz-szimulátor felkalibrálható és identifikálható. A felkalibráláshoz Windows alkalmazást írtam, ami egy .exe fájlal elindítható. Az identifikálás minden esetben visszaadja a kalibrációnak megfelelő értékeket, azaz használható az oktatásban. A beállítható szakasz összetételek 120 féle variációja valósítható meg. A variációkhoz kétfajta zavar jelet és egyfajta zajt is hozzá lehet rakni. A teszteket 0,015 sec-es ciklusidővel végeztem el, ez több mint háromszor gyorsabb, mint az előző félévben elkészült Raspberry Pi-s szakasz-szimulátor.

15.Összefoglalás angol nyelven

The finished plant simulator can work with calibration and identification. I programmed a Windows form application for calibrate the plant simulator. The GUI is startable with an .exe file. The identification is given back the right values in every time, so the plant simulator can be use in education. The teacher can set 120 different combination plant. Two type jam signal and one type of noise signal can be added to these combinations. At the test phase I set the cycle time to 0,015 sec, which is more than three times shorter than the formal year's Raspberry Pi plant simulator.

16.Hivatkozások

[1] T. Sándor, „Programozás II. (ÓE KVK 2149, elektronikus jegyzet)".

[2] V. Petik, „TIA Portal ismertető," 2018..

[3] „Arduino," [Online]. Available: https://www.arduino.cc/.

[4] „wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/I%C2%B2C.

[5] "https://en.wikipedia.org/wiki/USB," [Online].

[6] „wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface.

[7] „wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/RS-232.

[8] „wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Arduino.

[9] „visualstudio," [Online]. Available: https://visualstudio.microsoft.com/.

[10] Á. Varga, „10. Labor videó: összetett irányítástechnikai tagok szimulációja III. rész".

[11] Á. Varga, *7. Labor videó.*

[12] Á. Varga, „8. Labor videó: összetett irányítástechnikai tagok szimulációja I. részURL".

[13] Á. Varga, „9. Labor videó: összetett irányítástechnikai tagok szimulációja II. rész".