

S Z A K D O L G O Z A T

Egervári Bence

2021.



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
MÉRNÖKI KAR

***INTELLIGENS ÚTVONALTERVEZÉS ÉS KONTÉNER
RAKODÁS MEGVALÓSÍTÁSA VASÚTI ÁRUMOZGATÁS
ESETÉN***

SZAKDOLGOZAT

Egervári Bence

IV. Mechatronikai mérnök BSc

hallgató

Témavezető:

Dr. Sarcevic Péter

Főiskolai docens

SZEGED

2021.

ÖSSZEFOGLALÁS

A dolgozatomban egy elképzelt szituációt modellezek, ami egy automatizált rakodó berendezés, egy vasútállomáson. Azon ötletből alakult ki a feladat, hogy az áram hajtotta járművek fejlesztésén manapság elég nagy hangsúly van, mégis úgy érzem, ebből a vasutak kimaradnak, így próbáltam egy ötlettel effektívebbé tenni a vasúti áruszállítást. A munkámhoz a LabVIEW programozói környezetet választottam, ebben történik képfeldolgozás, azon belül QR kód olvasás, SQL adatbázis kezelés és ipari robot vezérlés. Az iskolában elérhető hardver egy Universal Robot UR5 volt, ami kiváló lehetőséget adott a reprezentáláshoz, ehhez alkalmaztam egy pneumatikus megfogót, és egy PLC vezérelt lineáris hajtást. Végeredményben egy működő rendszert sikerült kialakítanom, amely a kezdeti elképzelésimnek teljesen eleget tesz.

ABSTRACT

As my thesis work, I am willing to make a model of an automated cargo placer system in a railway station. The idea derives from that, in my opinion, the development of electric vehicles is much more emphasized nowadays than trains'. So I aimed to come up with a solution that makes rail freights significantly more effective and efficient through the use of cargo transit. I used LabVIEW development enviroment in my work, that handles image processing, QR code reading, SQL database managment and industrial robot control. The hardware available at the school was an Universal Robot UR5 with a pneumatic tool, and a PLC controlled linear drive. In effect, I was able to develop a well-functioning system which completely fulfilled my expectations (from the beginning).

TARTALOMJEGYZÉK

1	Bevezetés	3
2	Irodalmi áttekintés.....	5
2.1	Vonattal és kamionnal való szállítás összehasonlítása	5
2.1.1	Civil életre gyakorolt hatások	5
2.1.2	Légellenállás összehasonlítása	5
2.2	Modellezés.....	8
2.2.1	Konténer makett	10
2.3	Alkalmazott robot.....	12
2.4	QR kód	13
2.5	Fejlesztői környezet.....	15
2.6	Adatbázis kezelés	16
3	A modell megalkotása.....	18
3.1	Program felépítés.....	18
3.1.1	Kamera	20
3.1.2	Figyelmeztető Pop UP	22
3.1.3	Számláló.....	23
3.1.4	Feladatmeghatározás	24
3.2.	Feladatok megvalósítása.....	24
3.2.1	Visszarakás a vonatra.....	24
3.2.2	Lerakás tárolóba	25
3.2.3	Lerakás átvételre	25
3.2.4	Lepakolt állapot.....	26
3.3	Manuális mód	27
3.4	SQL megoldások	28
3.4.1	Meghívás	28
3.4.2	Táblába helyezés	29
3.4.3	Érték frissítése	30
3.5	Robot kommunikáció	31
3.6	Lineár hajtás	32
3.7	Megfogó fej	33
3.8	Robot program.....	34
3.8.1	Futás előtt.....	34
3.8.2	Program törzs	35
3.8.3	Home	36

3.8.4	Vonat alprogram.....	36
3.8.5	Kamera alprogram.....	37
3.8.6	Tároló alprogramok.....	37
3.9	Kezelőfelület	38
3.9.1	Vezérlő	38
3.9.2	Manuális vezérlés.....	39
3.9.3	Kamera és táblák	40
3.10	Tesztelés	41
4.	Összefoglalás.....	42
6.	Irodalomjegyzék.....	44

1 BEVEZETÉS

A témám alapötlete, az elektromos hajtású járművek fejlesztése kapcsán merült fel. Mindig is érdekelt a téma, nyomon követem az elektromos autók, járművek fejlődését már több éve.

A villanyautosok.hu-n olvastam Simon Zsolt „Magyar fejlesztésű elektromos kamion szállít az Audinak Győrben” című cikkjét 2020 őszén, mikor a témaválasztás aktuális volt, ahol egy egyenlőre egyedülálló elektromos kamionról van szó, amit szolgálatba helyeztek, és hogy fejlesztés alatt van az iparág folyamatosan, ami először nagyon jó ötletnek tűnt, hiszen rengeteg kamion megy az utakon, rengeteg gázolajat elhasználva. Napokkal később jutott eszembe újra ez a cikk és kezdtem gondolkodni azon, hogy hiszen már van árammal működő teherszállítási eszközünk, mégpedig a vonat. Kicsit utánajárva megtudtam, hogy manapság lényegesen több áru van szállítva kamionnal, mint vonattal.

Ennek oka, hogy a kamion „háztól-házig” jár, míg a vonat csak állomástól állomásig., illetve a megszervezése is sokkal egyszerűbb, csak felhívunk egy fuvarozó társaságot és egy bizonyos összeg ellenében, jönnek és elviszik a rakományt a megbeszélte időpontban. Egy vonat esetében sokkal körülményesebb ezt megvalósítani, először is, ha csak egy kamionnyi rakományunk van azért nem igazán éri meg elindítani egy darab szerelvénnel egy mozdonyt, másrészt mivel a vasúti pályák szinte teljes egésze egy nyomvonalas csak, így pontos megszervezése kell, ahhoz melyik vonat éppen mikor használ egy bizonyos szakaszt. Na de akkor miben tudnánk segíteni, hogy népszerűbbek legyenek a vonatok? – merült fel bennem a kérdés.

Első gondolatom egy menetrend létrehozása, felmérni mekkora és milyen frekvenciát transzport áru közlekedik Európa városai között, illetve kisebb cégek esetében országokon belül, és e szerint létrehozni egy lehetőleg sűrű menetrendet. A lényeg, hogy egy cégnek csak annyi dolga legyen, hogy kiválaszt egy menetrend szerint közlekedő vonatot a hozzá legközelebbi csomóponton/állomáson, és bérel rajta annyi vagonnyi helyet amennyi neki szükséges, és mintha postára adta volna onnantól az általam elképzelt rendszer generál neki egy útvonalat, átszállásokkal együtt. Itt jön egy újabb probléma, még hozzá, hogy egy ilyen komplex és sűrű rendszer borzasztóan túlterhelne minden állomást. Dolgozatomban modellezni szeretnék egy szituációt, miszerint egy állomáson található egy robot (esetemben egy Universal UR5 lineár

hajtással), ami egy beérkező vonat konténereit vizsgálja és dönti el, hogy mi a teendő az adott konténerrel.

Az elképzelt munkafolyamat, hogy először is érzékeljük, hogy érkezett egy vonat, majd a robot sorrendben elkezd leolvasni a konténerekről a QR kódokat, minden olvasás után az eredmény eljut egy adatbázisba ahol, vizsgáljuk hogy mi a teendő vele, négy variáció lehet:

- a konténer tovább halad a vonaton – nem a jelenlegi állomás a célja, illetve nem ezen a helyen van átszállása – ez esetben nincs teendőnk vele
- a konténernek átszállása van az állomásunkon – tehát le kell venni a vonatról és el kell helyezni egy erre kijelölt helyre, az adatbázisnak elküldeni, hogy a konténer ezen az állomáson van és megadni a várakozóhely címkéjét
- a konténernek ez a célállomása, de az átvevő nem érkezett meg érte –ekkor is lekerül a vonatról az áru egy kijelölt várakozó részre, szintén el kell küldeni az adatbázisnak a csomag érkezését és a lerakás pontos helyét
- a konténernek ez a célállomása és az átvevő megérkezett érte – ekkor levesszük és megnézzük, hogy az átvevő hova kapott várakozási engedélyt, majd oda helyezzük a konténert

Az előbb említett várakozási engedély, egy jegyrendszer lenne, az adott állomás a megérkezett átvevőnek egy jegyet ad ki, hogy például A1-es helyen várakozhat vagy veheti át a konténerét. Amennyiben még nem érkezett be a vonat, akkor az elfoglalt helyen várakozva a robot egyből a szerelvényről a várakozó helyre teheti a konténert, ha már a robot levette és félrerakta a szállítmányt, akkor az átvevőnek várakoznia kell addig míg a robot üresjáratba nem kerül, amíg az állomáson vonat van azzal fog foglalkozni, hogy az minél előbb tovább haladhasson.

2 IRODALMI ÁTTEKINTÉS

2.1 Vonattal és kamionnal való szállítás összehasonlítása

2.1.1 *Civil életre gyakorolt hatások*

Amennyiben a teherszállításunk nagyrésze vonatokkal történne, az infrastrukturális életben azt vennénk észre, hogy az autópályákon lényegesen kevesebb kamiont láthatnánk, civil utazóként igaz nem zavaróak annyira, de a baleseti kockázatuk illetve a nagy tömegüknek köszönhető relatív gyors útrongálásuk megszűnne. Mivel az elképzelésem nem ad megoldást az állomás és telephely közti áruszállításra ezért ott továbbra is kamionok közlekednének, amik közutakon és városokban sok baleseti forrást hordoznak magukkal illetve a város környéki környezetszennyezést se csökkentené.

2.1.2 *Légellenállás összehasonlítása*

Ebben a fejezetben számításokkal szeretném vizsgálni, hogy a két szállítási forma között mekkora a légellenállási különbség.

Mielőtt a számításokba bele kezdenék, szeretnék elméleti síkon felvetni egy gondolatot, mégpedig hogy miért is valószínűsítem azt, hogy a bekezdés végén a vonat fog jobban produkálni. Egy mozdony átlagosan 20-30 szerelvényt tud húzni rakománnyal együtt, ennek nagyon nagy előnye, hogy egymás után sorba vannak és mind ugyan abban az irányban haladnak, tehát egymásnak szélárnyékot tartanak. A számításom abból a megközelítésből fogja vizsgálni a dolgot, hogy létezik a bevezetésben megálmódott rendszer és 25 kamiont fel tudunk váltani 1 vonattal, tehát ha 25 kamionnal indulnánk útnak, azoknak mind-mind lekéne győzniük a légellenállást, sajnos nem ajánlott egymáshoz nagyon közel menniük, mivel a féktávolságra vannak kötelezve a saját biztonságuk érdekében, amennyiben ezt betartják az előttük haladónak a szélárnyékából csak töredék részt tudnak hasznosítani.

Az összehasonlításhoz választott járművem talán kérdéseket vetnek fel. Vonatnak egy MÁV-V43-ast választottam, mégpedig azért mert formai kialakítása nagyon hasonlít egy téglatesthez, tehát homlokfalát közelíthetem egy négyzethez, pont mint az európai kamionok legnagyobb hányadának. Ezzel kicsit a kamionnak kedvezem mivel korszerűbb vonatok esetében már ügyeltek a légellenállás csökkenésére de mivel még mindig nagyon gyakoriak a V43-ashoz hasonló vonatok, ezért nem tartom valótlannak a vele való számolást. Mivel a kamionokról elmondható, hogy nagyjából ugyan azt a formát képviselik nagy vonalakban, ezért választásom a

megbízható forrás vezérelte, ezért a Scania oldalán talált R580LA brossurájából szeretném most idézni az adatokat (1. táblázat).



1.ábra
MÁV V43

Forrás: <http://vasutmania.blogspot.com/2011/11/mav-v43-sorozat.html>



2.ábra
Scania R580LA
Forrás: kleyntrucks.com

1.táblázat Az összehasonlításhoz szükséges adatok

Forrás: mozdonyfoto.5mp.eu és Scania Chassis Specification R580LA

	MÁV V43	Scania R580LA
Magasság (h)	4540 mm	3042 mm
Szélesség (w)	3106 mm	2600 mm
Homlokfelület nagysága(h*w=A)	14101240 mm ² ≈ 14,101m ²	7909200 mm ² ≈ 7,909mm ²

A légellenálláshoz használt képlet (Szűcs, 2016):

1.

$$F_l = \frac{\rho}{2} * A * c_w * v^2$$

F_l: légellenállási tényező(N)

c_w: formatényező (légellenállási tényező)

A: homlokfelület nagysága(m²)

ρ: a levegő sűrűsége(kg/m³)

v: sebesség (m/s)

A következő képletből pedig kiszámolható lesz a légellenállás legyőzéséhez szükséges motorteljesítmény:

2.

$$P_l = \frac{\rho}{2} * A * c_w * v^3 = F_l * v$$

P_l: motorteljesítmény (W)

Az összehasonlítás során 90 km/h-ás sebességgel számolok ami 25 m/s-nak felel meg és a levegő sűrűségét 20 °C-on 1,2045 kg/m³-nek veszem, illetve a formátényezőt(c_w) a hasonló forma miatt mindkét esetben 1-nek.

Így a légellenállás a vonat esetében:

3.

$$F_{lv} = \frac{\rho}{2} * A_v * c_w * v^2 = \frac{1,2045 \frac{kg}{m^3}}{2} * 14,101 m^2 * 1 * \left(25 \frac{m}{s}\right)^2 = 5307,7 N$$

4.

$$P_{lv} = F_l * v$$

5.

$$P_{lv} = F_{lv} * v = 5307,7 N * 25 \frac{m}{s} = 132692,5 W$$

A kamionnak pedig a következő:

6.

$$F_{lk} = \frac{\rho}{2} * A_k * c_w * v^2 = \frac{1,2045 \frac{kg}{m^3}}{2} * 7,909 m^2 * 1 * \left(25 \frac{m}{s}\right)^2 = 2976 N$$

7.

$$P_{lk} = F_{lk} * v = 2976 N * 25 \frac{m}{s} = 74400 W$$

Tehát a kettő különbségéből megtudható, hogy egy vonatnak mennyivel több motorteljesítményt kell alkalmazni a légellenállás legyőzése érdekében.

8.

$$P_{lv} - P_{lk} = 132692,5 W - 74400 W = 58292,5 W$$

Ez nem kis teljesítmény különbséget jelent, de a mi esetünkben a már fent említett szituációt elemezzük, amikor is 25 konténernyi rakomány elszállításáról van szó, ez esetben 25 kamiont és 1 vonatot kell számításba venni.

9.

$$25 * P_{lk} - P_{lv} = 25 * 74400W - 132692,5W = 1727307,5W$$

A számítás alapján a 25 kamionnak 1,7 millióval több Joule-t kell elhasználniuk másodpercenként erre a célra. A példa sarkított esetet vesz, hisz mint említettem már a kamionoknak is lehet minimális szélárnyéka, illetve a vonat szerelvények között is van hely, nem fedik egymást tökéletesen a konténerek tehát mindegyiknek van minimális légellenállása, de ha még a differencia felét is vesszük akkor is említésre méltó a különbség. Illetve azt is látni, hogy ebből a megközelítésből már 2-3 konténer esetén érdemesebb vonatot indítani.

2.2 Modellezés

Modellezésnek azt hívjuk, amikor a valóság egy részletének egyszerűsített mását hozzuk létre, lényeges szempontjai, hogy mit modellezünk, milyen szisztéma szerint, és hogy mennyire egyszerűsítjük az adott részletet. Célja az adott jelenség vagy rendszer megismerése (Geda, 2011). Ennek elengedhetetlen része, hogy minél több információt gyűjtsünk be, tehát méréseket kell végeznünk a modellezni kívánt környezeten Modellezést a következő dolgokra alkalmazunk :

- kommunikáció, dokumentáció
- analízis
- származtatás
- szimuláció

Hasznos dolog mivel kisebb, jobban áttekinthető, könnyebben elmondható és finomítható (Gönczy, 2010). Az én esetemben szimulációs modellezést fogok alkalmazni egy környezeten, kiragadom a számomra fontos elemeket az elképzelt



3.ábra

Kikötőkben alkalmazott konténerrakodó

Forrás:

<https://sbm.news/articles/6135dc7bc4e8f4abd2b0141d>

állomásról, és méréseket tudok végezni, tulajdonságokat elemezni, illetve észre vehetek olyan hibákat, melyeket elméleti síkon nem sikerült meglátni. Bár az én munkámnak inkább a szemléltetés a célja, arról hogyan is nézne ki egy ilyen rendszer, mégis elengedhetetlen része lenne, ez a modell egy valós tervnek mely szerint ez megépül életnagyságban.

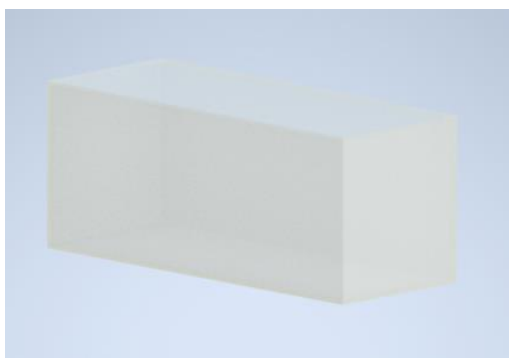
A modellben törekedni kell, hogy minél jobban reprezentálja a valós környezetet, a lehető legtöbb részletnek hasonlítania kell a valós elképzeléshez, hogy miket kell ilyen szempontból előtérbe helyezni függ a modellezés céljától. Esetemben ilyen sarkalatos pontja a modellnek a robot, egy erre ideális rakodó, a hajóiparban nagyon gyakori és a 3. ábrán látható is, ezt a kikötőkben alkalmazzak, fel és lehelyezésére a konténereknek. Lényege, hogy két támasztási pontja van amik síneken tudnak elmozdulni(ezt nevezzük most „x” iránynak), melyeket áthidal egy merevítés, ezen tud mozogni az effektor „y” irányban, és végül a merevítésen sodronyokkal állítható a megfogó magassága, vagyis „z” tengely szerinti elmozdulása, tehát ebben a robotban nincs rotációs csukló, minden tengely transzlációsan mozog. Ennek eredménye, hogy munkatere egy téglatest, ezt általánosságban nem, de ebben az esetben hasznos, a rakodótér magas lehet, jól kihasználható, a konténereket a robot könnyen megközelíti felülről, és a kialakításnak köszönhetően jól megtámaszthatók a tartó lábak, nagy tömegeket is biztonságosan tud szállítani.

Sajnos ilyen megvalósításhoz nincs lehetőségem, de a szemléltetés céljából a rendelkezésre álló robot is tökéletes lesz, hiszen a modell konténerek súlya kicsi, nem kell megpakolni őket áruval, hiszen ez a modell szempontjából majdnem lényegtelen. Abból a szemponttól viszont nem lehet eltekinteni, hogy ügyelni kell programozás közben, hogy a robot ne végezzen oldal irányú forgatást rajtuk, például ha gépjármű van az adott konténerben. Mivel nem tudjuk mi van egyes konténerekben, ezért a biztonság miatt minden egyes darabra azt kell feltételeznünk, hogy forgatás a két vízszintes tengely mentén árthat a benne lévő árunak. A függőleges tengely menti elforgatásnak nincs ekkora jelentősége mivel a gravitációs erővel mindig párhuzamos marad a konténer eredeti magasság tengelyével, tehát a rakományra változatlanul hat, csak a centripetális erő és gyorsulás tehetne kárt, ami pedig a lassú forgatási sebességgel elkerülhető.

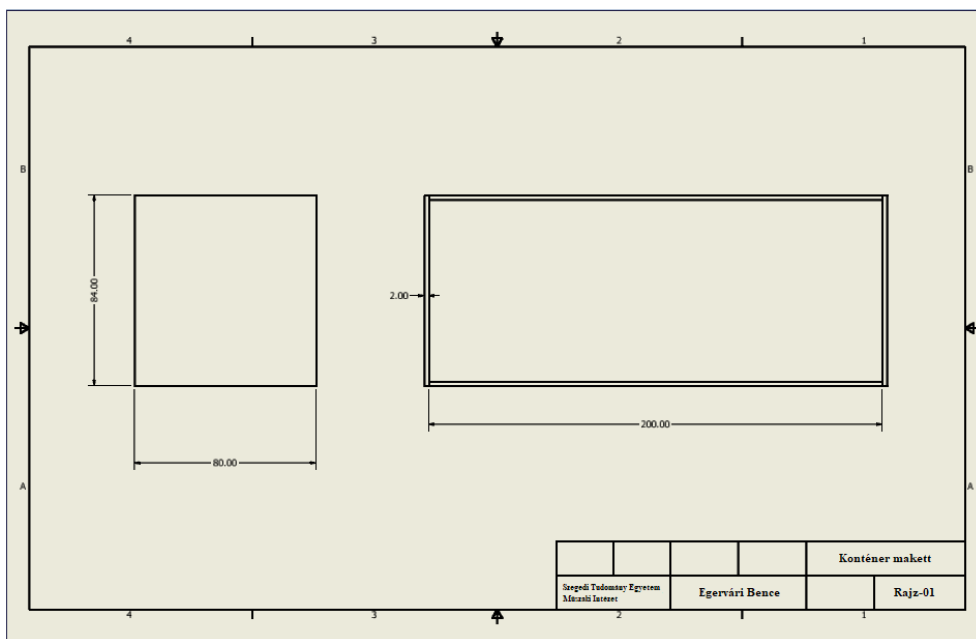
2.2.1 *Konténer makett*

Az általam elképzelt környezet modellezésében nagyon fontos, hogy legyenek kézzel fogható, egészen pontosan robottal fogható konténer makettek. Ezt otthon készítettem el, eredetileg színezett műanyagból szerettem volna, de nem állt rendelkezésre a környékemen, csak átlátszó plexit tudtam beszerezni. Azért kifejezettem műanyaggal szerettem volna dolgozni, mert könnyen megmunkálható, könnyű makettet tudunk belőle csinálni, és nem utolsó sorban a robotkar vákuumos megfogást fog alkalmazni, amihez jó tapadás szükséges.

Első lépésként csináltam róla egy 3D-s modellt (4.ábra), a gyors megtervezés és jó meglátás érdekében, illetve a demonstráció miatt egy rajzot.



4.ábra
Konténer 3D modell
Forrás: saját



5.ábra
Konténer makett rajz
Forrás: saját

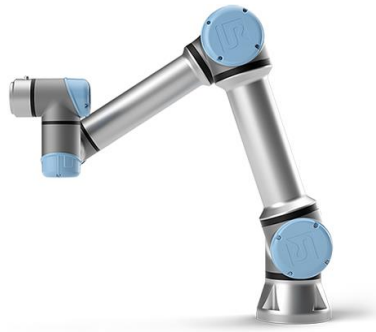
A plexit 1000mm*500mm-es táblában tudtam megvenni, a darabolást előkarcolás mentén törve oldottam meg, majd ez után ragasztottam össze őket. Összesen 6 darabot készítettem, hogy minden robotfeladatot be tudjak mutatni emberi közbeavatkozás nélkül.



6.ábra
Saját készítésű konténer makett
Forrás: saját

2.3 Alkalmazott robot

Az egész munkám fő pontja a robot ami végezni fogja a konténerek rakodását. Erre az Szegedi Tudomány Egyetem Mérnöki Karán lévő Universal UR5e típusú 6 rotációs csuklóval rendelkező robotot választottam, az iskolában ez egy lineáris



7.ábra

Universal UR5e

Forrás: universal-robots.tech-con.hu

hajtással van ellátva, tehát a robot platformja vízszintes irányban el tud mozdulni egy irányba. Ez nagy segítség lesz nekem, így könnyebben végig tudok haladni a vonat mellett párhuzamosan és a konténereket ugyan azzal a mozdulattal meg tudom fogni. Ez a robotkar az „e” szériába tartozik, ezek a robotok félépítésükben nagyon hasonlóak, a különbség a terhelhetőség, ami az én robotom esetében 5 kg, erre utal a nevében is a szám. Ennél vannak kisebb és nagyon teherbírásúak is a szériában, amihez ezzel arányos hatókör is társul.

A gyártó műszaki adatlapjáról szeretnék néhány szerintem fontos adatot felsorolni az UR5e-ről:

- Terhelhetőség: 5 kg
- Hatósugár: 850 mm
- Szabadságfokok: 6 forgó csatlakozás
- Áram fogyasztás maximális átlaga: 570 W
- Pozíció ismételhetség: $\pm 0,03$ mm
- Tengelymozgás: minden csukló $\pm 360^\circ$
- I/O portok
 - Digitális bemenet: 2
 - Digitális kimenet: 2
 - Analóg bemenet: 2

- Alapterület: Ø 149 mm
- Tömeg: 20,6 kg

(EN ISO 10218-1:2011)

Programozása egy 12 colos érintőképernyőn történik, ami grafikus interfésszel is rendelkezik. Ezt nekem is használnom kell, de a program fő logikáját és a ki-és bemenetek, illetve az adatbázis kommunikációját LabVIEW-ban valósítom meg, erre későbbi fejezetekben fogok kitérni bővebben.

2.4 QR kód

A QR kód egy két dimenziós szimbólum neve a „Quick Response” angol kifejezésből van rövidítve, ami annyit tesz „gyors válasz”. 1994-ben találta fel egy Denso nevű cég ami a Toyota csoport tagja, ezért is történt, hogy a kezdetekben Japánban volt nagy sikere. Ez annak az érdekességnek is köszönhető, hogy támogatja a kandzsi és kínai írásjeleket, illetve a kana japán szótagírást is. Szabványosítása 2000 júniusában történt, ISO/IEC 18004 szám alatt, későbbiekben két kiegészítést is kapott.

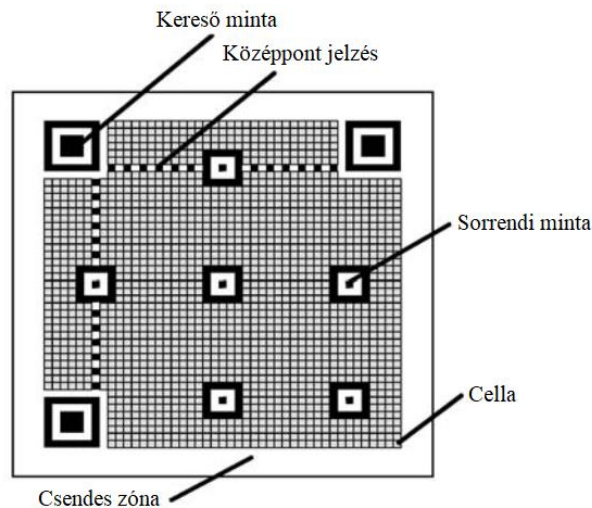
Mérete a benne tárolt adat mennyiségétől függ, a legkisebb méret a 21x21 cella, növelés csak 4 cella bővítéssel lehetséges egyszerre horizontális és vertikális irányban is. A legnagyobb előállítható méret pedig 177x177, ami a minimális mérethez képest 39 bővítést jelent, tehát elmondható, hogy sok méretvariáció létezik, e szerint verziókba sorolják őket 1-től 40-ig.

Előnyei:

- A legnagyobb előnye és ezért is előszeretett a használata, hogy három nagyon fontos dolgot ötvöz egybe, és eddig ezt más kódok csak külön-külön tudták. Az első a nagy adattárolási képesség, mint például a PDF417-nek, a második a relatív kicsi helyigénye, méretre és formára ezért is hasonlít a „Data Matrix” kód típushoz, végül pedig a könnyű és gyors olvasás.
- Bármilyen irányból készülhet róla fénykép, nem kell gondolnunk a tájolásra fotózáskor. Ez annak köszönhető, hogy a kód három sarkában el van helyezve egy kereső minta ami minden kódban ugyan olyan (megtekinthető a 8. ábrán), ezek segítségével tájoló a dekóder program és kezdi meg az értelmezést.

- Hibajavítási képesség, ennek szintjeit betűjelekkel jelölik, van L, M, Q, és H. Ezek a felsorolt sorrend szerint 7%, 15%, 25% és 30% hibatűrést engednek meg, ezt a kód generálásánál tudja megadni a készítő, minél jobb hibatűrést választ annál jobban veszt a kódunk kompaktságából. A kód beolvasásakor több tényező is bezavarhat, hogy ne tudjuk tökéletesen lefotózni. Ilyen például a hiányos megvilágítás, a túl kicsire nyomtatott kód, vagy esetleg ha szennyeződés vagy sérülés érte a kódbélyeget. Ezen problémák ellenére is hibátlanul ki lehet olvasni a tartalmat, ezt a kódon belül lévő kiegészítő jelzések teszik lehetővé, melyek a 8. ábrán megtekinthetők.

(Anil, 2016)



8.ábra

QR kód kiegészítő jelzései

Forrás: Tan Jin Soon: QR Code

Középpont jelzés (Timing Pattern): Ez a sorminta definiálja a cellák középpontjának a koordinátáit. A dekóder megkeresi ezt a mintát és e szerint tájolja a többi a cellát melyekből a kívánt adat kinyerhető. Így elkerülhető, hogy esetleges torzulás vagy sérülés miatt egyes cellákra hibát adjon a dekóder, ez a korrigálás természetesen csak bizonyos mértékig lehetséges.

Sorrendi minta (Alignment Pattern): Szintén egy korrigálási minta, legfőképp a nemlineáris torzulásokat küszöböli ki, például ha egy kör alapú oszlopra van ragasztva a QR kód. Ez egy vastag fekete keretű négyzet, ezeknek a torzulásából és egymáshoz viszonyított helyzetéből képes korrigálásra következtetni a dekóder, benne

elhelyezett négyzet a középpontjában pontos tájolásához szükséges. Csendes zóna (Quiet Zone): Egy üres mezőnek tűnik, de mégis elengedhetetlen része a szimbólum olvasásának, segít a dekódernek egy képen megtalálni a QR kódot, négy vagy több cellának kell a csendes zónát kialakítania.

Adat tér(Data Area): A 3.ábrán látható szürke háttérű rész, vagy másképp írva a kiegészítő jelzéseken kívüli részek, az úgynevezett adat tér, a kódgenerátor bináris rendszerben konvertálja az adatokat, majd ezután rendezi el őket a kódban, a cellák feketék vagy fehérek lehetnek, ezzel jelezve, hogy 1 vagy 0 az értékük. Az adat egyenesen van Reed-Solomon kóddal mely hibaelhárítási céllal van belevegyítve. A maximális tárolt adat mennyiség(tárolási hatékonysággal):

- Számokból: 7089 darab (3,3 cella/karakter)
- Betűkből és jelekből: 4296 darab (5,5 cella/karakter)
- Bináris adatok 8-bites rendezésben: 2953 darab (8 cella/karakter)
- Kandzsi jelek: 1817 darab (13 cella/karakter)

(Tam, 2008)

2.5 Fejlesztői környezet

A program megírásához a LabVIEW-t választottam, amit iskolai tanulmányaim alatt sikerült megszeretnem.

A 1980-as évekre tehető az az időszak, amikor fellendült az elektronikai műszerek használata, egyre komplexebb és bonyolultabb mérőrendszereket kezdtek el alkalmazni a felhasználók. Emiatt ezzel együtt fejlődnie kellett a számítógépek alkalmazásainak is, ebben két fő irány bontakozott ki.

Az első a beágyazott rendszerek, amikor is a mérőműszerbe beágyazva került egy erre specializált számítógép, ami végezte a jelfeldolgozást és az adatkezelést, ezek mellett tudta kezelni a műszerre kötött perifériákat is, mint például kijelző és gombok.

A másik ágon mindeközben, az a gondolatmenet bontakozott ki, hogy a mérőműszer marad a standard funkciójánál, és nem kap plusz kiegészítőket, hanem egy sima személyi számítógépen futó szoftver fogja nekünk elvégezni a jelfeldolgozást, illetve meghatározni a műszer funkcióját. Ehhez alkotta meg a National Instruments a LabVIEW-t melynek első verzióját 1986-ban adták ki. Ezt a kivitelezést egyébként virtuális műszernek nevezzük, vagyis angolul „virtual instrumentation”-nek, ennek rövidítéséből ered a „vi” fájlkiterjesztés.

A LabVIEW nem egyedülálló, de sajátosságának tekinthető, hogy grafikus programozási nyelv, emiatt más gondolkodásmódot is kíván, mint a leggyakrabban emlegetett írásos, procedurális nyelvek. A fejlesztők fő célja, hogy a szoftver adatfolyam-programozásra legyen alkalmas, és a műszerekre jellemző „adatbeolvasás-feldolgozás-kijelzés” feladatsorra emlékeztessen, és ennek minél egyszerűbb leírhatóságát tegye lehetővé (Hainzmann et al., 2011).

Emiatt egy kicsit specifikusnak mondható az alkalmazása, könnyű példát találni olyan feladatokra, amelyeket grafikusán pár kattintás alatt meg lehet oldani de például C nyelvben hosszú órák kellenének hozzá, illetve ez fordítva is igaz, hogy vannak olyan szituációk melyekre a LabVIEW-ban csak nagyon körülményesen lehet megoldást kialakítani. Az én programom úgy érzem talán ennek a két állításnak a határán van, programozás közben többször is éreztem, hogy egy-egy rész csak pár sor lenne C nyelvben, itt pedig rengeteg blokk és kötözés után jött létre a funkció amit szeretnék. De maga az alapja a dolognak, a lépésről-lépésre történő robotvezérlést pedig kifejezetten kényelmes tekintettem.

2.6 Adatbázis kezelés

A programomban szükségem van adatok tárolására és kiolvasására, amire először a LabVIEW-ban tömbök (arrays) használatát gondoltam ki. Neki is kezdtem ezekkel dolgozni, de hamar rájöttem, hogy nem egészen erre a felhasználásra van kitalálva és tudomást szereztem róla, hogy a LabVIEW-hoz letölthető egy SQL Tool kit ami lényegesen megkönnyíti a dolgom.

Az SQL (structured query language), mint a neve is elárulja egy strukturált lekérdező nyelv, ez relációs adatbázisok programozására alkalmas, angol nyelvű lekérdező parancsokkal tudunk végrehajtani utasításokat egy adatbázisban. Ezt az IBM-nél kezdték el fejleszteni a hetvenes években, ahol az általuk használt nagy adatmennyiséget akarták kezelhetőbbé alakítani.

Jellemzői:

- kevés parancs
- nincsenek vezérlő elemek
- nincs input-, output kezelés
- nincs grafikus kezelőfelület

(Pedro, Pam, 2019)

Sokáig előnye volt, hogy szabvány utasítások léteznek hozzá, emiatt bármilyen kezelőfelületről használtuk teljes kompatibilitása volt. Későbbiekben létrejött helyi nyelvek új elemekkel egészítették ki, ennek eredménye az inkompatibilitás lett, és már több SQL nyelvjárásról beszélünk (Kovács, 2004). Az én esetemben csak a legalapvetőbb parancsokra volt szükségem melyek benne vannak az 1986-os SQL szabványban (ISO 9075), emiatt a programom adatbázis kezelés szempontjából nagyrészt univerzálisnak tekinthető. Ettől függetlenül megemlítem, hogy MySQL architektúra van a programomban, mivel ez volt ingyenesen elérhető.

Az SQL használata alapvetően úgy működik, hogy az utasításokat egy parancssorba begépeljük, majd lefuttatjuk. Ezeket a parancsokat négy csoportra oszthatjuk. Az első az adatdefiníció, ilyen például a CREATE mellyel létrehozni tudunk, ennek ellentétje a DROP, ami megszüntetésre alkalmas. Második csoport az adatkezelők, INSERT, amivel beszúrni tudunk sorokat táblákba, vagy adott esetben a DELETE paranccsal törölni belőlük, illetve ebben a csoportban az utolsó az UPDATE, amivel egy feltételt megadva módosíthatunk bizonyos adatokat. A harmadik és legkisebb csoport a lekérdező, ide csak a SELECT utasítás tartozik, amivel kiválasztani tudunk, majd listázni. Végül az adatvezérlő parancsok, például GRANT/REVOKE ami bizonyos jogokat ad illetve elvesz, ezek mellett van a COMMIT/ROLLBACK ami jóváhagy vagy visszagörget. A nyelvben az utasításokat pontosvessző választja el egymástól.

3 A MODELL MEGALKOTÁSA

3.1 Program felépítés

Ebben a fejezetben szeretném részletesen leírni a program megírásának menetét, illetve a logikájának magyarázatát.

Sorban szeretnék haladni, és mindig a soron következő munkafolyamatot részletezni. Először is, a program indulásának feltétele, hogy a robot és a szoftver is el van indítva, ekkor a programban lehetőség van a robot manuális, illetve automatikus kezelésére. A számunkra érdekesebb az automatikus verzió. Mindenekelőtt el szeretnék magyarázni egy alapvető logikai dolgot a robot kezelésével kapcsolatban, mivel ez a továbbiakban gyakran lesz emlegetve. Ahhoz, hogy a robot és a program is lépésekben haladjon, úgy építettem fel a programot, hogy a robottól mindig lekérdezem, hogy éppen egy munkafolyamatot végez-e, vagy már befejezte azt, és készen áll a következő utasításra. Ezt úgy oldottam meg, hogy amint a robot elkezd egy programot, az elején átír egy változót nullára, és amikor végez, ezt egyre változtatja. Így feltételt tudtam alkotni a LabVIEW-ban arra, hogy akkor lépjen tovább a programom, amikor ez a változó egyes értéket vesz fel.

Az egész program egy „While loop”-ban (ameddig ciklus) van megírva. A megnevezés arra utal, hogy ez a ciklus mindaddig fut amíg egy feltétel be nem teljesül. Ezt a feltételt alapból a LabVIEW-ban a While ciklus jobb alsó sarkában megtalálható piros gombként ábrázolt dobozba kell bekötni, én ide egy VÉSZ-STOP gombot kötöttem be. A While ciklus elengedhetetlen része a program működésének, mivel ez teszi lehetővé a folyamatos futást, e nélkül az indításkor csak minden képernyőn látható elem egyszer lefutna, és az én programom esetében nem történne semmi. Alternatív megoldásom lehetett volna egy „Control and Simulation loop”, ami szintén hasonló szerepet tölt be, és bizonyos paraméterei módosíthatók, de ehhez külön bővítményt kell a LabVIEW-hoz telepíteni, másrészt nekem nem volt szükségem a kínált beállítások használatára.

A program szerkezetének egy alappillére a LabVIEW-ban „Case Structure”-nek nevezett ablak. Ez grafikusán úgy néz ki, hogy a képernyőn kijelölhetünk egy területet, ami onnantól az alá a Case Structure alá tartozik. Ez az elem arra szolgál, hogy több lehetőséget meg lehet neki adni, és egy feltétel alapján dönti el, hogy melyik eset valósuljon meg. Ezeket a megvalósításokat úgy tudjuk programozni, hogy az egész keret tetején van egy lenyitható fül ahol kiválasztjuk melyik esetet szeretnék

A Case Structure feltétele, hogy melyik legyen a következő esete amibe lép, az önmagában van megvalósítva. Ez csak „Shift Register” segítségével oldható meg, ami egy a While loop szélein elhelyezhető elem, lényege, hogy egyes ciklusokból átvihető legyen egy-egy adat a következő ciklusba. Én például ebben az esetben minden ciklusba tovább viszem a Case struktúráim állását, és amikor a robottól kapom a jelet, hogy végzett egy „Select” blokk egy másik állást fog beletenni a register-be ami az új ciklus következtével léptetni fogja a Case Structure-t.

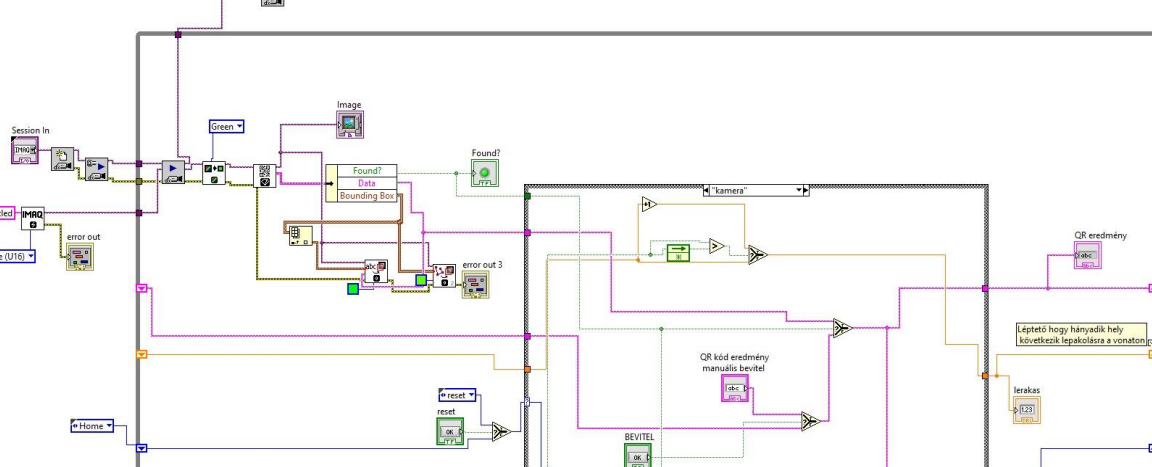
19

Alaphelyzetben „Home” pozícióban van a robot és várja, hogy megérkezzen egy vonat amin tud dolgozni. Itt következik egy AND (és) logikai kapu, ami arra vonatkozik, hogy igaznak kell lennie annak is, hogy van vonat, amin lehet dolgozni, illetve a robot is azt a jelet küldi, hogy készen áll egy utasításra. Ha mindkét feltétel igaz, akkor még egy dolognak meg kell felelnie, még pedig, hogy az adott vonat még nincs-e lepakolva, amennyiben ez igaz nem fog neki állni lepakolni azt, hanem más folyamatnak áll neki amit később fogok részletezni.

A képzeletbeli vonatom hat vagonnal rendelkezik és sorba pakolja le azt. Induláskor elmegy az első pozícióra, ebben a Case-ben csak annyi, van, hogy amennyiben a robot jelzett, hogy végzett a felvétellel, mehet és megmutathatja a konténert a kamerának, vagyis a Case structure a „kamera” nevű fülre ugrik. Mielőtt ezt a fület részletezném, még elmondanám, hogy az összes felvételi pont ennyiben merül ki, itt nincs más dolga a programnak csak felvetetni a konténereket.

3.1.1 Kamera

A „kamera” fül egy viszonylag fontos része a szoftvernek és több dolog is játszódik le itt. Először is, ami a robottal kapcsolatos, hogy egy fix pozícióba küldi azt, ahol egy web kamerának megmutatja a konténerre ragasztott QR kódot. Ezt a valóságban elegánsabb és takarékosabb lenne megoldani úgy, hogy a robotra van erősítve a kamera. Ezt én kihagytam, és ezt a megoldást választottam mivel az utóbbinál kábelezési problémáim is lehettek volna, illetve ahhoz, hogy teljesen fixen rögzíteni tudjam a kamerát a robotkar végéhez, valószínűleg túl nagy módosításokat kellett volna végre hajtanom, amit iskolai tulajdonnal természetesen nem tehetek meg. Rátérve a programra, a programozási részét a kamera használatnak nem magában ebben a Case-ben csináltam meg, hogy manuális mód esetén is lehessen használni, illetve ha ezen belül csinálom, akkor mindig újra és újra el kell indítania a kamerát, ami az én esetemben sok plusz idővel járt volna, mivel csak egy nem túl modern webkamera állt rendelkezésemre és több másodpercbe telik neki felállnia, megtalálni



10.ábra

Kamera fül és a fizikai kamera használata QR kód olvasásra

Forrás: Saját

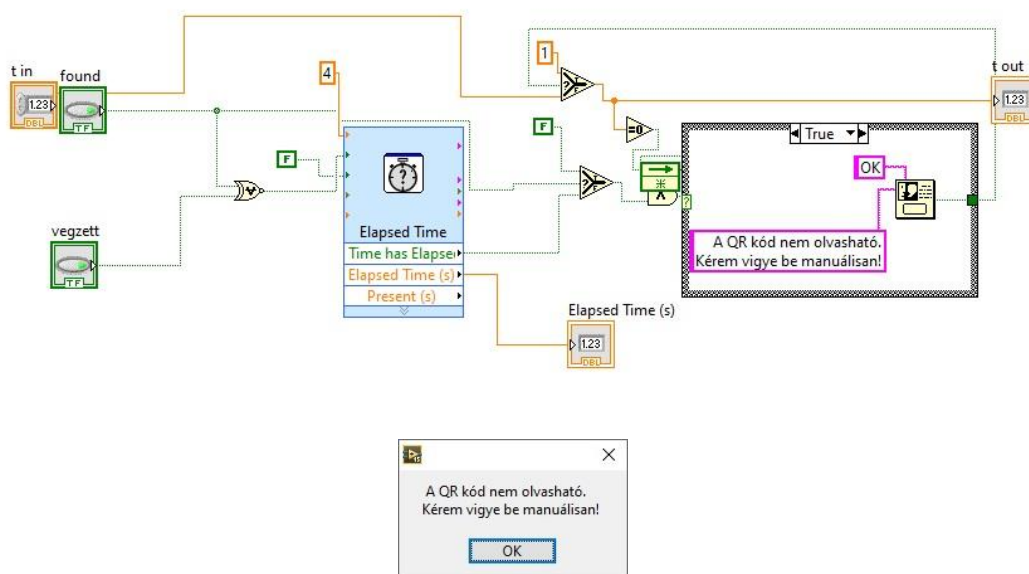
A kamera adatok bevitele a 10. ábra bal felső sarkában kezdődik. Először is egy Control megy bele egy „kamera megnyitó” blokkba. Erre azért volt szükségem mert a laptopom is rendelkezik webkamerával és azt használta alapértelmezettnek, de mint írtam már én egy külön kihelyezett USB-s kamerát használok. Ezután két konfiguráló „Grab” blokkot kellett beillesztenem, ami beolvassa a kamera kép adatait, itt meg lehet adni a színbeállításait a kamerának, fekete-fehér mellett döntöttem mivel csak QR kód olvasásra van használva, és így számítási igényt csökkenthetek. A második Grab igényel egy „Camera Close” kimenetet, ami a használat végén befejezi annak meghívását a programba. Majd pedig a QR kód kereső, és egyben olvasó elem következik. Ebből vezetem ki végül a képet, egy „Image” nevű kép megjelenítő indikátorba, illetve a blokk másik kimenete a „QR Code Reports”, amit kibontva különböző adatokat tudhatok meg a kódról. Számomra szükségesek voltak, természetesen a kód eredménye, ezután hogy talált-e kódot, amit egy boolean értékben kapok meg, és még esztétikát figyelembe véve használtam a „Bounding Box” kimenetet, ami értékekkel szolgál a QR kód hollétéről a képen belül, így megoldhattam, hogy amikor a szoftver talál egy kódot, akkor azt zölden bekeretezi és fölé írja annak eredményét.

A QR kód adata és a „Found?” boolean értékem egy Select elembe megy, az előbbi az igaz ágra, az utóbbi pedig a feltétel lesz. A logika itt, hogy amennyiben talált

kódot az igaz ág lesz aktív, tehát a kimeneten ez fog kimenni, amit pedig egy Shift register-be kötöttem. Ezzel azt értem el, hogy az utoljára megtalált kódot eltárolom, és későbbiekben még tudok vele dolgozni, amire szükségem is lesz. Egyébként ennek a Select-nek a hamis ágon való bemenete, egy másik Select kiemenete, egy manuális beviteli lehetőséget is építettem a programba, ez teszt üzem közben nagyon hasznos. Ez úgy néz ki, hogy ennek a második Select-nek a hamis ágába megy a Shift Register kiemenete, az igazba pedig az általunk bevinni kívánt kód. Az eldöntést egy „Bevitel” nevű gomb határozza meg, ha ezt megnyomjuk a beírt értékünk bekerül a register-be.

3.1.2 Figyelmeztető Pop UP

Mellékeltem mind ezek mellé egy felugró jelzést amennyiben a kamera fülön van a program, de már négy másodperce nem talált QR kódot. Ami a valóságban jelzést ad nekünk arról, hogy esetleg sérült a kód, vagy rosszabb esetben a kamerával van valamilyen gond. Az én megvalósításom szerint a QR kód alá írtam annak adatát, tehát ha ilyen merülne fel, a konténerről kell leolvasni és bevinni kézzel a kódját. Ezt a Pop Up részt egy külön SubVI-ban készítettem el, sajnos elég bonyolult logikai rendszert kellett alkalmaznom, nem vagyok benne biztos, hogy ez a legtisztább megoldás, de a célom az volt, hogy egy figyelmeztetés után, az „OK” gombra kattintva



11.ábra
A Pop Up SubVI belseje és a felugró ablak
Forrás: Saját

eltűnjön, és többet ne figyelmeztessen. A talált erre alkalmas blokkokkal így sikerült megcsinálnom működő képesre.

Az „Elapsed Time” blokk az úgymond időmérőnk, aminek egy reset bemenet segítségével lehet állítani, hogy mikor induljon el. Ez egy Boolean értéket kér, mely ha folyamatosan igazat kap értelem szerűen folyamatosan reset-elődik, így nem megy a számláló. Ezen kívül egy számértéket kér még, hogy mennyi idő elteltével adjon egyes értéket a kimenetén, erre a számra én egy konstans négy másodpercet adtam meg. Mint látható a reset-re a „Found?” és a robot végzett van rá kötve egy XNOR logikai kapun keresztül. Ez a kapu azért szükséges, mert így amíg a robot nem jelzett, hogy befejezte a folyamatot, vagyis nincs a kamera előtt a konténer addig nem indul a számláló, tehát XNOR esetén ha mindkét bemenet nulla akkor a kimenet egy, vagyis reset-eli a számlálót. Amennyiben az egyik bemenet egyes a másik nulla, vagyis a robot végzett, de nem talált kódot a program, akkor a kapu kimenete nulla lesz, elindul a számláló. Végül amikor a kapu mindkét bemenete igaz akkor nulla a kimenet, vagyis talált kódot és végzett is a robot akkor nincs szükség figyelmeztetésre, vagyis nem kell a számlálónak mennie. Ezek után már csak annyi történik, hogy ha letelt a négy másodperc akkor a kimeneten érkezik egy egyes, ami átvált egy Case Structure-t ahova elhelyeztem a blokkot ami előhívja az ablakot. Ebbe két bemenet kell, az egyik a szöveget jeleníti meg, a másik a gomb feliratát határozza meg. Ezek után a gomb megnyomása eredményez nekünk egy igaz jelet, amit kiveztem a cellából és visszavezettem egy feltételbe, ami azt hivatott végezni, hogy az ablak ne tudjon ismét felugrani.

3.1.3 Számláló

Mielőtt rátérnék a QR adatainak feldolgozására, megemlíteném, hogy ezen a fülön található egy számláló. Ennek célja, hogy tudjuk a sorban melyik konténer lett már felvéve. Ezt a gyakorlatban jobb lenne megoldani úgy, hogy az állomáson a sínek mellett lenne minden vagonra egy szenzor ami nézi, hogy melyiken van konténer illetve melyiken nincs, és aszerint küldeni sorba a robotot felvételre. Sajnos ezt nem tudtam megvalósítani, nem állt rendelkezésemre hat darab szenzor erre a célra.

A számláló minden alkalommal lép egyet amikor a kamera fül lesz aktív. Ezt azért ide építettem be mert minden felvétel után a kamera következik. Mikor elkerült a konténer a megfelelő helyre, utána lesz vizsgálva ennek a számnak az értéke és a szerint fog menni a robot egy új konténerhez a vonaton, de erről később is írni fogok.

3.1.4 Feladatmeghatározás

Visszatérve a QR eredményére, ez minden esetben egy hatjegyű számkód, ezt elemzem egy táblázatban. A tábla, amiben keresek két oszlopos, az első oszlopa a vonaton lévő konténerek kódjait tartalmazza, a második oszlop pedig a konténerhez tartozó feladat számát. Ezt a táblát SQL segítségével töltöm be a LabVIEW-ba, de erről részletesebben később számolok be.

A logika lényege, hogy a betöltött táblánk egy 2D Array a LabVIEW-ban ezért nehéz vele dolgozni, mivel ez csak két oszlop, könnyű belőle 1D-t csinálni. Ezt úgy oldom meg hogy egy-egy oszlop törlés blokkba vezetem az Array-t. Így külön-külön rendelkezem a két oszloppal, amely a kódot és a feladatsorszámot tartalmazza, a kódot bevezetem egy kereső blokkba. Ez két bemenetet kér, az egyik az 1D tábla amiben keresünk, a másik pedig a keresni kívánt elem. Kimenete pedig az indexe lesz a megtalálta adatnak. Eközben a feladat kódokat tartalmazó oszlop be van vezetve egy „Index Array” blokkba, ami kimenetként megadja azt az egy elemet, amely indexet kértük. Ebbe természetesen az előző keresésünk eredményét kötöm be, így kapjuk meg, hogy a sorszám kódhoz, milyen feladatkód tartozik.

A feladatkód egyes, kettes, illetve hármas szám lehet, a keresés után ezt vizsgálom, és egy feltételt állítok, hogy ez a szám határozza meg a következő lépést.

3.2. Feladatok megvalósítása

Mint írtam, megkaptuk egy számkód formájában, hogy mi a teendő az éppen leolvasott konténerrel. A számok a következőket jelentik:

- 1 = Visszarakás a vonatra – erre csak azért van szükségünk, mert nem tudtam a robotfejre erősíteni a kamerát, ahogy azt a valóságban a legeffektívebb lenne, egyébként csak annyi dolga lenne, hogy a leolvasott konténer marad a vonaton, mehet a következőhöz.
- 2 = Lerakás tárolóba
- 3 = Lerakás átvevőhelyre

3.2.1 Visszarakás a vonatra

Amennyiben az SQL táblában egyes szerepel a QR kódunk mellett, azt vissza kell raknunk a vonatra, mivel a képzeletbeli állomás nem is végállomása vagy átszállási hely. Itt ismét az előbb említett számlálót olvasom ki, és aszerint tudom meg, hogy éppen melyik volt az utolsó felvett konténer, és vissza is küldöm oda. Ezt is egy „al Case Structure”-ben oldottam meg, és a feltétel ezt fogja beállítani és ennek a

füleiben van beírva a parancs a robotnak. Amint a visszarakás megtörtént, ugyan azt a felvételi keresést végzem el szintén a számlálón, és a szerint küldöm a robotot a következő konténerért.

3.2.2 *Lerakás tárolóba*

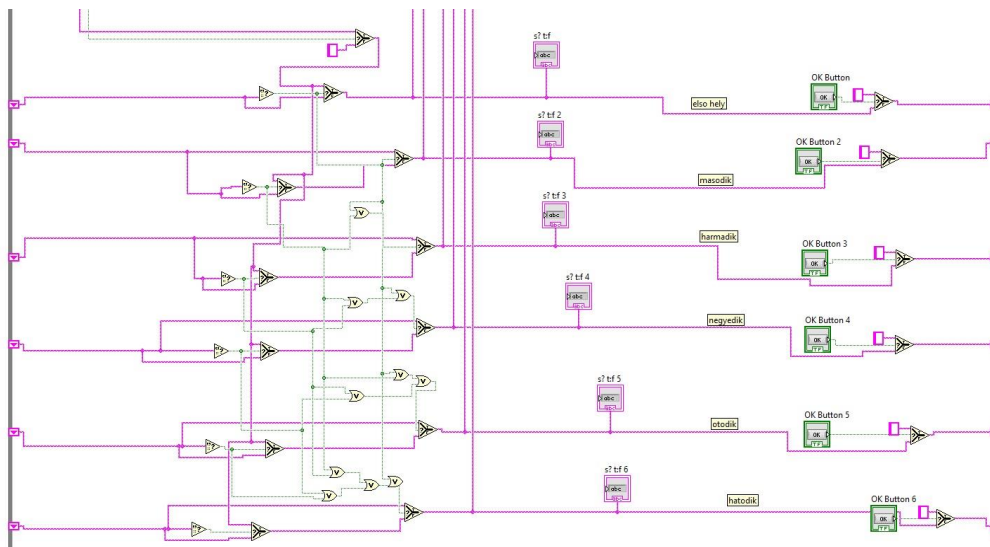
A kamera fülön kettes értéket kaptunk, és a robot végzett visszajelzést küld, ez esetben a „lerakas taroloba” fülre lép a program.

A tárolóm hat hellyel rendelkezik, minden helyhez tartozik egy boolean shift register amiben eltárolom, hogy van-e már ott konténer. Tehát mikor erre a fülre lépünk elindul egy vizsgálat az első helyen, ahol azt nézi a program, hogy a shift register értéke igaz vagy hamis, amennyiben igaz, tehát már van ott valami így továbbmegy a második helyet vizsgálni és így tovább. Amennyiben valahol hamis értéket talál, oda fogja lerakni az éppen megfogott konténert, a találat után a keresés megáll, és nem vizsgálja a többi helyet. Itt szintén egy Case Structure-t fogunk vezérelni ezzel a feltétellel, ahol elkészítettem minden helyhez egy külön esetet, ami tartalmazza a robotutasítást a hely pozíciójához. Ezen kívül természetesen ad egy igaz értéket a helyhez tartozó shift register-nek. A későbbi munkálatokhoz szükséges volt egy SQL táblába is mentenem, hogy melyik helyen milyen kódú konténer van, az SQL-ről később összefoglalóan fogok írni. Mikor befejeződött minden folyamat, elindul egy vizsgálat, ami a „kamera” fülön említett számláló állását nézi. Mivel ez mindig azt a számot mutatja a sorban, amelyik már le van pakolva a vonatról, ezért mindig a plusz első pozícióba megy. Például ha a számláló hármast ad akkor a feltétel szerint a negyedik helyre fog menni a robot felvenni az új konténert.

3.2.3 *Lerakás átvételre*

Az alapesetünk, hogy az állomáson még nincs egy várakozó kamion se a konténerére, amikor megérkezik valaki, akkor meg kell mutatni, hogy mely kódú szállítmányért jött. Ezt nálam manuális beírással lehet megtenni, de érdemesebb lenne ha itt is QR kód olvasás lenne, csak sajnos nem állt rendelkezésemre egy második webkamera. Tehát beírtuk a kódot és a „Bevitel” nyomógommbal be is került a rendszerbe. Itt minden helyhez társítottam egy Shift Register-t amely eltárolja, hogy az adott helyen melyik kódú konténerre várnak, ami a lepakolásnál nyer értelmet, hogy tudjuk hova melyiket kell rakni. Ezekbe a register-ekbe való adat elhelyezés egy lépcsős rendszeren alapul. A program mindig vizsgálja, hogy éppen melyik a legközelebbi üres hely és oda fogja rendelni a legutóbb bevitt kódot, ehhez egy „String

Empty?” logikai blokkot használtam, ami Boolean értékben adja meg nekem az eredményt.



12.ábra
SQL tábla meghívás LabVIEW-ban
Forrás: Saját

A 12 ábrán látható ennek kivitelezése, illetve a jobb oldalon lévő „OK Button”-ök lényege, hogy amennyiben valaki átvette és elhagyta a helyet akkor az onnantól ismét kiadható lesz másnak, tehát üressé kell azt tegyük, hogy a program ismét annak ismerje fel és feltöltse új érkezővel.

Amikor a 3-as feladat kódot olvasunk ki egy QR kódból, akkor azt ezekkel a helyekkel vizsgálom, hogy melyikkel egyenlő, amelyik helynél egyenlőséget talál, azt az utasítást küldi a robotnak.

3.2.4 Lepakolt állapot

Mint írtam a folyamat úgy kezdődik, hogy egy érzékelő jelet küldd amint az állomásra érkezett egy vonat, vagyis ez fogja megadni a kezdő jelet a folyamat indításának. Mivel amikor lepakolta a robot a hatodik konténert is, és felveszi a „Home” pozícióját újra indulna a folyamat. Ezért az a hatodik konténer egy kész értéket fog adni egy változónaak, aminek negatív állapota szintén feltétel a lepakolás indításához.

Ahogy a dolgozat elején írtam, szerettem volna azt az esetet is megvalósítani a modellben amikor egy konténer a tárolóban várja az adott vonatunk megérkezését, vagyis átszállása volt az állomáson, akkor azt a robot automatikusan felhelyezze a vonatra. Itt egy olyan problémába ütköztem, hogy a legideálisabb megoldás erre az

lenne, ha minden vagon mellett lenne egy érzékelő az állomáson, hogy van-e rajta konténer, avagy nincs. Ez a lepakolás esetében is hasznos lenne mivel a modellemben azt az ideális esetet vizsgálom, hogy minden vagonon van valami. Tehát ezek az érzékelők azért lennének jók az elsőként említett szituációra, mert ezzel vizsgálhatnám, hogy melyik vagon üres és lehet rá pakolni. Ezek az érzékelők ha valóságban képzeljük el a projektet akkor csekély összegűek lennének és egyértelmű lenne beépítésük, de az én esetemben anyagi megfontolásból sajnos ki kellett hagynom ezt a részt.

Rátérve a tényleges problémára, sokat gondolkoztam, hogy megvalósítható lenne így is a dolog. Egy táblába menteném, hogy mely konténereknek nem kellett lekerülniük a vonatról, vagy fordítva, megnézem, hogy melyik QR kódokhoz volt levétel parancs, és utána vizsgálom, hogy az hányadik helyről lett le véve. De végül arra jutottam, hogy ez fölösleges munkának bizonyulna a modellezés szempontjából, és inkább konklúzióként fogom fel, hogy egyes esetekben nem kell minden áron működőnek látszó, de sok hibalehetőséget rejtő megoldást találni modellezés esetében. Elegendő, ha átgondoljuk, hogyan is lehetne programozni az ideális esetre.

Ez az én esetemben kifejezetten egyszerű, a programot a felvétel részéig el is készítettem. Ami annyiból áll, hogy mint előzőekben említettem a tároló tartalmát egy SQL táblában vezetem, és emellett a vonatnak is van egy táblája ahol a konténerek kódja van melyekre neki szüksége lesz. Ebben a táblában csak a vagon kódja van, és hogy melyik állomásról várja azt. Tehát futtatok egy SQL lekérdezést, hogy sorolja fel azokat a kódokat melyeknél az én állomásom szerepel, utána ebben a listában egyesével elkezdek haladni, az aktuálisan kiragadott kóddal indítok egy keresést a tárolóban és kiolvasom annak helyét. Végül csak elküldöm a robotnak, ami elindul és felveszi azt. Ezután következni egy egyszerű lépés, mégpedig, hogy sorban elkezdeném vizsgálni, hogy ellőről melyik az első üres vagon, amint az első megtalálja a keresés le is állna és elindulna a robot letenni rá a konténert. Lényegében ugyan az a helykeresés játszódna le, mint amit a tároló esetében bemutatam.

3.3 Manuális mód

Létrehoztam a programban a lehetőséget a robot manuális mozgatására, erről a képet a kezelőfelület fejezetnél lehet megtekinteni.

Annak érdekében, hogy az automatizált rész amit az eddigiekben részleteztem ne legyen aktív, bekerült egy Case Structure hamis részébe, és az igaz részébe

alakítottam ki a manuális részt, ennek a struktúrának az állapotát egy csúszó kapcsolóval tudjuk állítani.

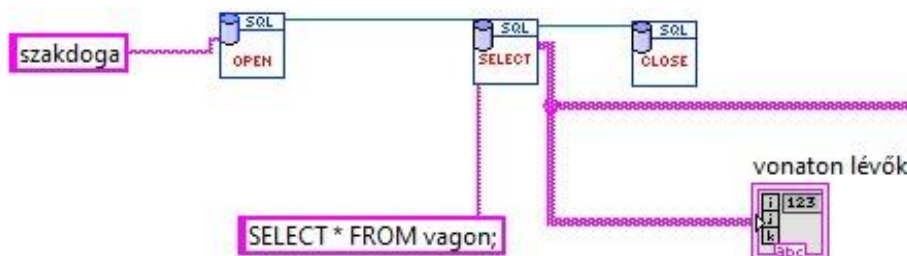
A működtetés úgy történik, hogy LED lámpák kapcsolásával tudjuk megadni a pozíciót ahova elszeretnék küldeni a robotot, illetve egy kapcsolóval megadjuk, hogy a megfogó nyitott vagy zárt állapotát szeretnénk. A programozási részét mondhatni „fejjele a falnak” módszerrel oldottam meg, ami ilyen kis tároló hellyel még nem volt sok idő, de a valóságban sok hely kezelésével nem opció az egyesével hozzárendelés, mint amit én csináltam. Mivel annyit tettem, hogy a leküldendő három értéket, vagyis a feladatot, a távolságot és a megfogó helyzetét minden pozícióhoz egy Select feltételhez kötöttem, így a programrész egy átláthatatlan vonalköteg lett. Utólagos belegondolás közben jöttem rá, hogy tömbösíteni kellett volna a jeleket valamilyen rendszer szerint, de a meglévő működő szerkezetet nem szerettem volna már kidobni.

3.4 SQL megoldások

Az SQL használatához egy „Wamp Server” nevű programot használtam, melyben MySQL táblák és lekérdezések használatára van lehetőség. Emellé kellett egy „ODBC SQL Server Driver” ami megoldotta, hogy LabVIEW-ból elérhetőek és használhatóak legyenek a tábláim. Végül a magához a LabVIEW-hoz kellett egy „SQL tool kit”-et letöltenem, amivel elérhetőek lettek olyan program blokkok, amik SQL utasításokra használhatók.

3.4.1 Meghívás

Elsőként a táblákat kell meghívnom a programban. Először is egy „Open SQL” blokkal utalok, az adatbázis nevére mellyel szeretnék dolgozni. Ezt a blokkba kötött string constans-al tudom megtenni. Majd ezután két opcióm van, „Select” vagy „Execute” program blokk következik (13. ábra). Előbbi táblák tartalmának

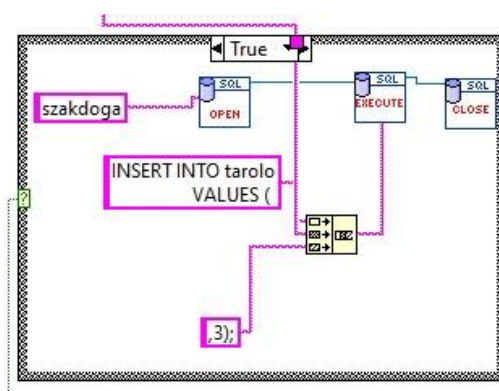


13.ábra
SQL tábla meghívás LabVIEW-ban
Forrás: Saját

lekérdezésre alkalmas, ez lehet teljes tábla tartalom, de feltétellel szűkíthetjük is az eredményt. Ebben az esetben a blokkból lesz egy 2D Array kimenetünk (13. ábra). Az utóbbi pedig valamilyen SQL utasítás végrehajtására alkalmas, például amiket én is használtam az INSERT INTO, UPDATE és DELETE. Ezek után pedig egy „Close SQL” blokk fogja nekünk lezárni a folyamatot

3.4.2 Táblába helyezés

A tároló rész tartalmát szintén SQL táblában jegyzem. Két oszlopa van a táblának, az első, hogy milyen kódú konténer került be, a második pedig, hogy melyik helyre került az.



14.ábra
SQL táblába illesztés, változó alapján
Forrás: Saját

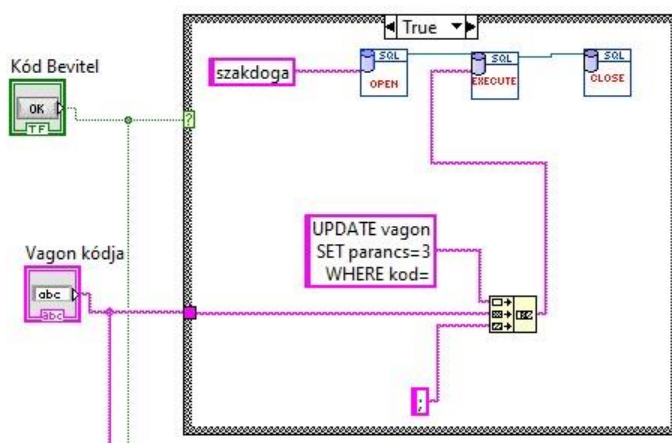
Itt az „Execute” blokkot kell használni mivel utasítást hajtunk végre. A bevitt utasítás kinézete a 14. ábrán tekinthető meg.

A képen látható, hogy az INSERT INTO parancsot használva illeszték a táblába. Használok egy string összerakó blokkot annak érdekében, hogy mindig az aktuális kód kerüljön be, ezt mint írtam egy shift register-ben tárolom, innen tudom előhívni. A képen látható eset, amikor az első két hely már foglalt a tárolóban, látható hogy a harmadik része a string-nek hármast tartalmaz, tehát ez a harmadik helyre fog kerülni. Ez azért nem változó, mert minden pozíciónak külön case-e van, így mindegyikben csak egy konstanssal elég megadnom azt. Ezen kívül egy olyan problémába ütköztem, hogy mivel az egész program egy While Loop-ban foglal helyet így mindaddig, amíg az adott Case van előtérben addig a program folyamatosan tölti fel a táblát ezzel az adattal minden ciklusban. Először az SQL kódot próbáltam megvariálni, és feltételt adni neki, hogy csak akkor szűrje be az elemet, ha ilyen még nem létezik a táblában, de sok fórum olvasás és több ismerős megkérdezése után

kiderült, hogy erre nem igazán van egyszerű megoldás. Ezért úgy oldottam meg a dolgot, hogy az egészet egy Case Structure igaz esetébe raktam. Ennek egy feltételt adtam, hogy egy számláló értéke nulla legyen. Ez a számláló mindig nulláról indul, amikor ez a fül lép életbe és minden ciklusban növekszik eggyel. Tehát az első ciklusban nulla az értéke tehát igaz lesz a feltételem eredménye, majd a következő ciklustól már egy és így tovább, tehát a beillesztés Case Structure-je többet nem lesz igaz, így csak egyszer fogja beilleszteni az elemet.

3.4.3 Érték frissítése

Amennyiben egy konténerért még nem érkeztek meg, akkor az SQL táblában kettes fog mellette szerepelni feladatként, tehát megy a tárolóba várakozni. Amint valaki megérkezett érte változtatni kell ezt az értéket hármásra. Ez a következő képpen néz ki a programban.



15.ábra
SQL tábla frissítése
Forrás: Saját

Az UPDATE parancsot használom erre a célra, ehhez meg kell adnom a frissíteni kívánt tábla nevét, majd hogy melyik oszlopban változtasson milyen értékre, és végül hogy mi a feltétele a változtatásnak. Nálam ez, hogy a kód oszlopban egyezzen az érték azzal, amit kézzel viszek be, egy String Control-on keresztül. A folyamat úgy hajtodik végre, ha lenyugtázzuk az egészet egy OK gombbal. Ez fogja a Case Structure igazra állítani a felfutó él erejéig, tehát csak egy „pillanatra”, így az utasítás csak egyszer fut le.

(Simon, 2015)

3.5 Robot kommunikáció

A robot és a LabVIEW közötti kommunikációt Modbus protokoll segítségével oldottam meg, ami az Universal Robot esetében alapból támogatott, a LabVIEW-ban a DSC modul telepítése után lesz elérhető.

A Modbus egy kommunikációs protokoll ami eredetileg PLC-khez készült a hetvenes évek végén, majd később terjedt el nagyobb körben, de fő felhasználási területe változatlan maradt a mai napig.

Az alapfelállítás szerint egy Modbus rendszerben az egyik fél a Server (Slave) a másik pedig a Client(Master), ahol a kliens tesz egy kérést, amire a szerver fog adni választ. Nálam a robot volt a szerver, a LabVIEW pedig a kliens (Halsorven, 2010).

A robot esetében elérhető az Universal Robots honlapján egy táblázat mely tartalmazza, hogy mely Modbus portok milyen paraméterekhez használhatók. Nekem elsőként egy Integer érték küldése volt a cél a robotnak. Erre a 128-256-os portok vannak kijelölve. Ezen kívül Boolean értékek vizsgálatát szerettem volna a hardver és a szoftver között melyre a 0 és 1-es portok voltak megjelölve, de sajnos ezeket nem tudtam megoldani, és időszűke miatt itt is integer értékekben használok egyes vagy nullás formájában.

A robot program felületén a `write_port_register('port száma','értéke')` script kóddal lehet egy portnak értéket adni, amit majd a szoftverrel lehet olvasni. Ezt arra használtam, hogy minden alprogramom a robotnál azzal kezdődik, hogy egy porton nullát írok, ezzel jelezve a szoftvernek, hogy a robot most mozgásban van, és a végén átirom ugyan azt a portot egyesre, ezzel jelezve, hogy végzett és kész az új utasításra.

A `read_port_register('port száma')` utasítással tudjuk kiolvasni a robottal egy port értékét, én így olvastam ki, hogy mi lesz a száma a végrehajtandó feladatnak. Ezt a következő fejezetben részletezem is.

Az általam használt portok a 2.táblázatban tekinthető meg.

2.táblázat Modbus portok

Felhasználás	LabVIEW port érték	Robot port érték	LabVIEW olvasás	LabVIEW írás	Adat típus
Feladat meghatározás	131	130		x	UInt16
Robot végzett	133	132	x		UInt16
Távolság megadás	135	134		x	UInt16
Megfogó feladata	137	136		x	UInt16
QR kód észlelés	139	138		x	UInt16

3.6 Lineár hajtás

A robot egy lineáris vonalon mozgó szervomotoros hajtáson helyezkedik el. Ez nagyban segítette a feladat megoldását mivel ezzel tudtam pozicionálni a robotot és az egyenes vonalban elhelyezkedő konténer makettek mellé így kényelmesen be tudtam állítani. A hajtás PLC-vel vezérelt, a robot küld neki egy abszolút értéket ahova szeretne mozogni és egy jelet, hogy indulhat a hajtás. Ezek után a PLC elindítja azt, majd ad vissza egy jelet, hogy befejeződött a mozgás, a robot folytathatja a

programját. Ez a rendszer már kész volt az iskolában, nekem nem kellett vele foglalkoznom, csak megértenem a működését, hogy használni tudjam.



16.ábra
UR5 a lineáris hajtáson
Forrás: Saját

3.7 Megfogó fej

A konténerek mozgatásához szükségem volt egy megfogóra, szerencsére az iskolában található egy AirTAC HFT 20x60-as pneumatikus munkahenger kifejezetten erre a célra, amihez engedélyt is kaptam, hogy használjam. A munkahenger magában még nem volt elég, még szükségem volt két felszerelhető pofára melyek érintkeznek a konténerrel. Ezeket 3D-ben megterveztem Inventor-ban, majd Mészáros Attila tanár úr segítségével (amit innen is szeretnék megköszönni neki) az iskolában lévő 3D nyomtatóval hoztunk fizikailag is létre. Mivel a konténer makettek átlátszó műanyaglapokból készítettem és a pofák is műanyagból voltak, ezért a tapadás érdekében a felszerelés előtt még ragasztottam rájuk nyílászáró szigetelésre alkalmazható szivacs csíkokat, melyek nagyon jól látták el ezt a feladatot.

A munkahengert egy 5/2-es elektropneumatikus szelep állítja, amit a robot fej részénél csatlakoztattunk. Ez a bemenete a robotnak direkt a megfogó részére lett kialakítva, és a Teach Pendant-en is Tool Input/Output néven található meg. Két bittel lehet vezérelni, amennyiben mindkettő alacsony, akkor a szelep alaphelyzete él, vagyis zárt. Amennyiben az első bit magas és a második alacsony akkor a munkahenger szétnyit, fordított bit állások esetén pedig összezár.

Egy regulátor is beépítésre került a rendszerbe, amit szintén Mészáros Attila tanár úrnak köszönhetek. Így tudtuk szabályozni a záraskor kilépő levegő

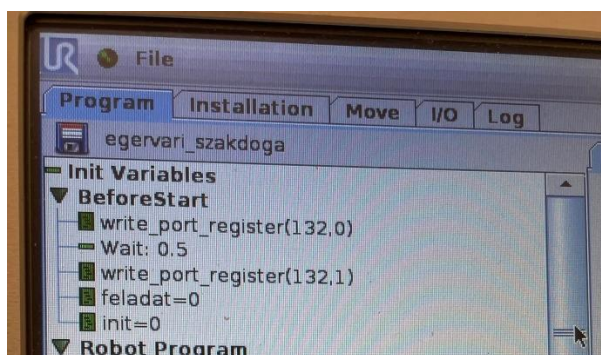
menyiségét, ezzel annak erősségét is, ami szükséges volt mivel féltettem a maketteket, hogy nagy nyomás esetén a szorítóerő roncsolja őket.

3.8 Robot program

Ez előtt is volt szerencsém gyakorlataim során robotokat programozni saját kezelőfelületükről, de most találkoztam először Universal Robottal. Számomra nagyon logikusan felépített és könnyen kezelhető felülete van. Sajnos a kezelőfelületen nem lehet képernyőképet készíteni, ezért külső eszközzel kellett lefotóznom.

3.8.1 Futás előtt

A program írásakor lehetőség van egy „Before Start” részt programozni, ez minden indításkor lefut, még mielőtt a valódi programrész kezdődne, itt leggyakrabban inicializálni tudjuk a változóinkat.



17.ábra

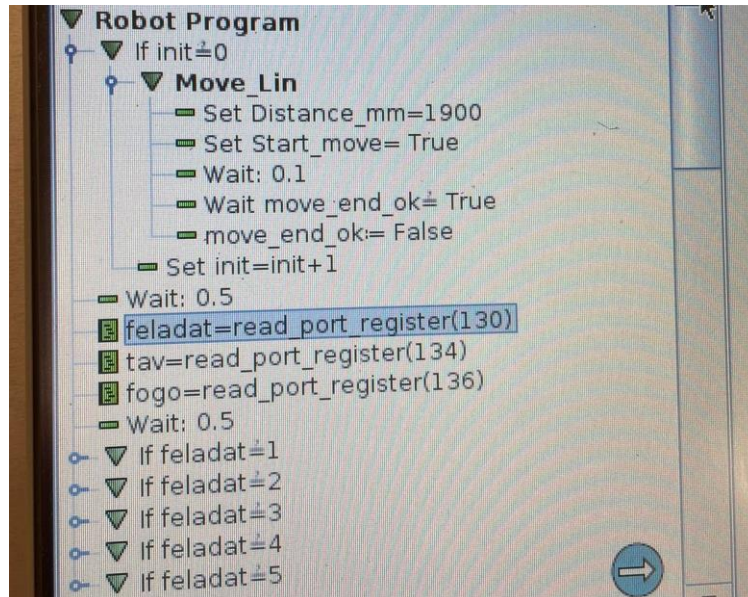
Before Start rész a robot programomból

Forrás: Saját

Mint látható ez az én esetemben is csak ennyiben merül ki, először is azt a portot kell újra írnom amellyel jelzek a LabVIEW-nak a robot állapotáról. Itt furcsának tűnhet, hogy egymás után írom újra, de a szoftveremben felfutó él vizsgálat van, így az újra küldött egyes felfutó éle jelzi a számítógépemnek, hogy elindult a robot program. Ez után még két változónak adok nulla értéket, az első a végrehajtandó feladat. A robot megtartja az utolsó felvett értéket a változókhoz, még kikapcsolás esetén is, tehát ha ezt nem nulláznám, egyből elkezdené azt a feladatot amit utoljára csinált. A második változó pedig a lineáris hajtás Home pozíciójához szükséges, ami a következő részben fog értelmet nyerni.

3.8.2 Program törzs

Itt került megírásra a mondhatni legfontosabb része a programnak. Először is a már említett lineáris hajtás Home pozíciójának felvétele, ami az „init” változó nulla értékénél történik, vagyis minden indításkor. Itt jól látható a hajtás vezérlése, először megadok neki egy abszolút értéket, ez esetben fixet és nem változót, majd kiadom, hogy indulhat a hajtás és várok mindaddig, amíg a PLC nem jelez vissza, hogy végzett a mozgás. Ez után annak értékét újra hamissá teszem, hogy a következő ciklusnál ne ugorja át a várakozást, legvégül pedig az „init” változót növelem eggyel, hogy ne nulla legyen az értéke és ne álljon többet Home pozícióba. A feladatok kezelésére egy olyan megoldást választottam, hogy az egyes feladatokat alprogramokban hívtam meg, ezért a törzs nekem csak annyiból áll, hogy három változó értékét olvasom folyamatosan és ezek közül a feladat változót vizsgálom, „IF” feltételekkel. Amint egy olyan feladatkódot olvas ki a program melyhez tartozik alprogram, akkor átugrik arra. Itt jön képbe az előzőekben említett változó inicializálás. Mivel programfutás előtt nullás értéket kapott, amelyhez nincs alprogram rendelve ezért nem fog meghívni semmit így folytatódik előlről az olvasás és összehasonlítás. Az alprogramok meghívása a „Call” paranccsal történik.



18.ábra

A robot program törzs része

Forrás: Saját

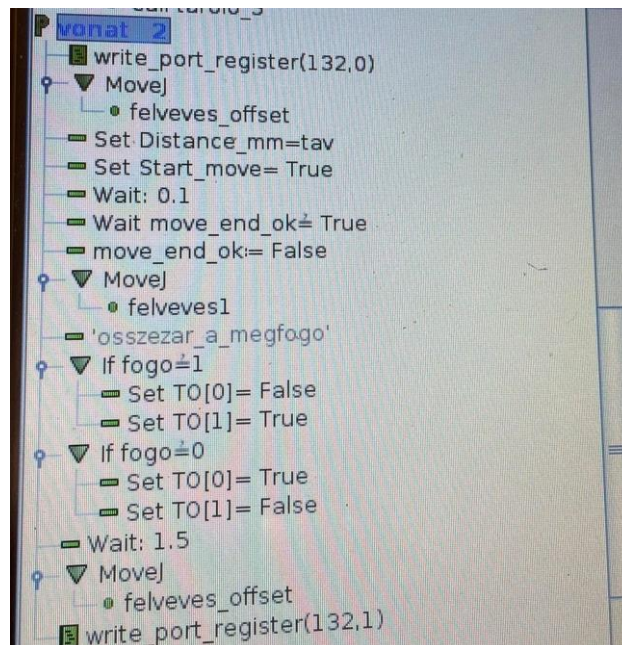
A feladat kódján kívül még két változót olvasok ki a robottal, mint ahogy a 18-as ábrán is látható. A „tav” nevű fogja meghatározni a hajtás pozícióját, a „fogo” pedig, hogy a folyamat során megfogni vagy elengedni kell-e az adott konténert.

3.8.3 Home

Egyszerűsége miatt nem helyeztem alprogramba, a feladat változó egyes értékével lép életbe, és csak egy MOVE utasítást hajt végre mellyel a robot egy semleges pozíciót vesz fel melyet Home-nak nevezünk.

3.8.4 Vonat alprogram

Ez az alprogram akkor indul el amennyiben a robot kettes értéket olvasott ki feladatként. Mint már említettem minden alprogram azzal indul, hogy jelzem a LabVIEW-nak, hogy programfutás van, ezt az első sorban meg is teszem 132-es portra nullát írva. Ezután egy mozgást hajt végre robot, a vonat fölé hajol előkészítve a felvevést vagy a lerakást. A már bemutatott lineáris hajtás következik, ami a „tav” változó értékéhez mozgatja azt. Ezzel gyakorlatilag azt határozom meg, hogy hányadik vagonon történik az adott munka. Ismét egy mozgás következik, amivel már ténylegesen „ráhajol” a robot a konténerre, vagy lerakás esetén a helye fölé, majd következik a vizsgálat, hogy megfognia vagy elengedni kell-e a megfogónak. Itt a jel kiadása után használtam egy relatíve hosszabb várakozást, hogy a munkahengernek legyen ideje a mozgásra. Végül kiemelem a robot egy újabb MOVE paranccsal és jelzek a LabVIEW felé, hogy végrehajtódott a művelet.



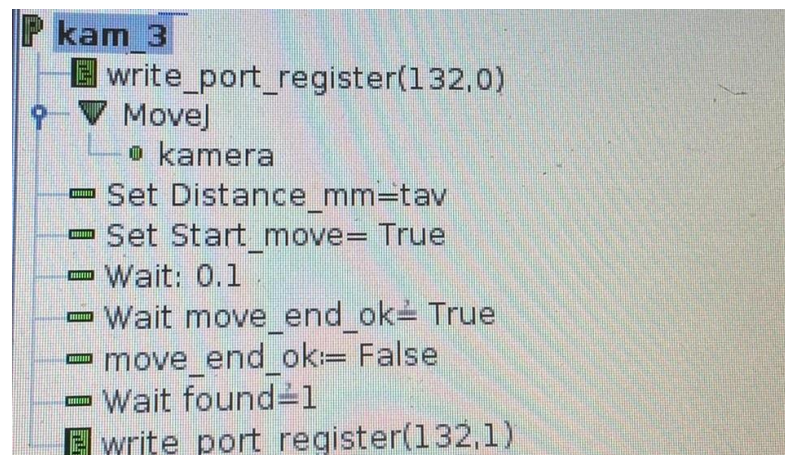
19.ábra

Vonat fel és lepakolás alprogramja

Forrás: Saját

3.8.5 Kamera alprogram

Ez a rész a „feladat=3” esetben lép életbe. Lényegében csak egy robot pozícióváltás és egy lineáris hajtásmozgatás történik, illetve ezek után még vár a program egy egyes értéket a LabVIEW-tól, hogy van-e kód. Ezt kétféle képpen kaphatja meg, vagy a QR kód programblokk küldi a saját „Found?” kimenetén, vagy abban az esetben, ha a kamera nem talált semmit, a kézi kódbevitel esetén a nyugtázó gomb fogja megadni. Ezt az értéket egy külön szálon (Thread), olvasom folyamatosan, erre azért van szükség, mert ha programom belül lenne akkor csak egyszer olvasná ki a port értékét. Amint ezt megkapta, küldi a jelet, hogy készen áll új feladatra,



20.ábra

Kamera alprogram

Forrás: Saját

3.8.6 Tároló alprogramok

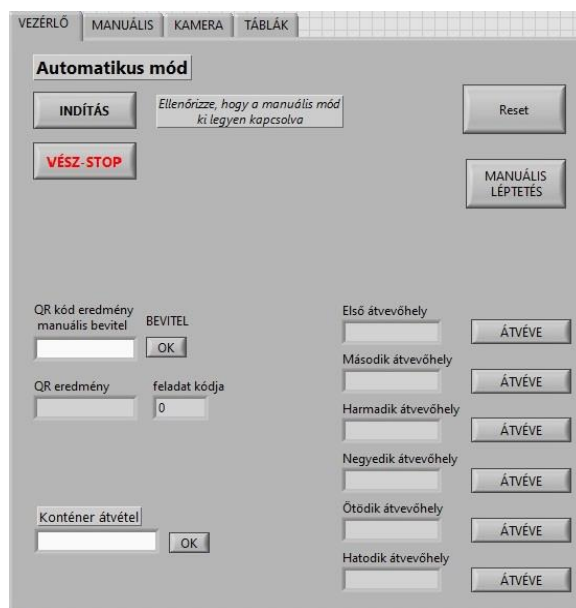
A tárolóban történő műveletekhez két alprogramot is készítettem, mégpedig azért mert két sorban képzeltem el a konténerek helyeit, és sajnos változó értéket nem tudtam megadni a robotpozíció koordinátaiban. A lineár hajtással könnyen meg lehet oldani az egymás után sorba helyezést, de ebben az esetben egy új alprogrammal egy új sort is kellett csinálnom. Ezeket a négyes és ötös értékekkel lehet meghívni. Ez az alprogram lényegében ugyan azt tartalmazza, mint a „Vonat”, csak más robotpozíciók vannak megadva. Ezen kívül mivel az átvétel is ugyan ezen az oldalán történik a robotnak a valóságban, és az első sorral egyvonalban lett megjelölve, ezért abban az esetben is ezt hívom meg csak távolibb hajtás távolsággal.

3.9 Kezelőfelület

A program kezelőfelületét a LabVIEW-ban úgynevezett „Tab Control”-al lehet külön részekre bontani a felületet, és ezzel külön fülekre helyeztem az egymáshoz tartozó dolgokat. Ezeket szeretném bemutatni ebben a fejezetben.

3.9.1 Vezérlő

Sajnos a következő bekezdésben látható manuális mód miatt nagy alapterületet igényelt blokk, emiatt nehezen töltöttem ki a teret a többi fülön, de szerencsére tényleg csak ennyi minden kell a rendszer automatikus futásához.



21.ábra
Kezelőfelület vezérlő oldala
Forrás: Saját

Bal felül helyet kapott az indítás gomb, ami értelem szerűen elindítja a folyamatot, mellette egy emlékeztető, hogy csak a manuális mód kikapcsolt állapotában fog elindulni a program, alatta pedig egy VÉSZ-STOP gomb, amit megnyomva leállítjuk a While loop-ot amiben az egész program helyet kapott. Így lényegében megállítjuk a program teljes futását, sajnos ezt nem tudtam a robot VÉSZ-STOP funkciójával össze egyeztetni, tehát mondhatni lényegi értelmét elveszti a biztonság szempontjából, mivel ez csak újabb utasítás leküldését akadályozza meg. Mindenesetre nekem hasznos volt a tesztelés során megállítani a programot egyes helyeken elemezni a történéseket.

Jobb oldalt felül egy Reset gombot raktam, ami teszt közben szintén sokszor volt használva. A robotot HOME pozícióba küldi és mindent lenulláz a LabVIEW programban, vagyis kezdődhet az egész folyamat előlről.

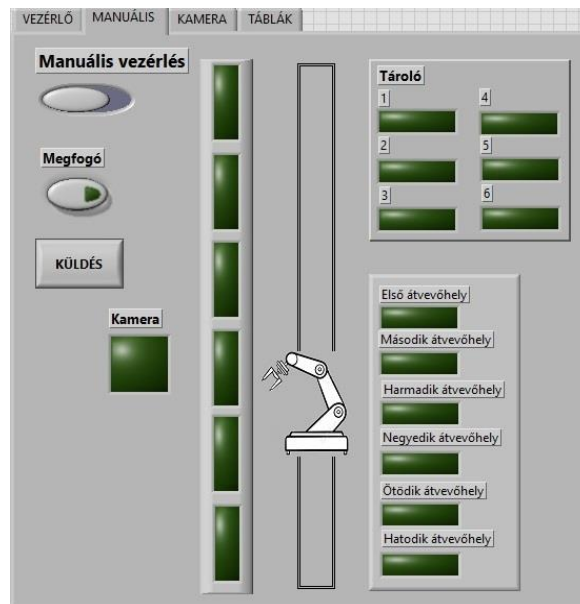
Ez alatt egy manuális léptetés gomb látható, ezt abban az esetben kell alkalmazni amennyiben a robot nem hajtja végre a következő feladatot, ekkor ezzel újra tudjuk neki küldeni az utasítást. Szerencsémre ezt csak néhány alkalommal kellett kipróbálnom a valóságban, viszont a virtuális tesztelés közben végig ezt használtam, hogy a lépéseken végig haladjak.

Bal alul a QR kód bevitele van, abban az esetben ha nem talált a kamera, illetve alatta egy visszajelzőn amin monitorozhatjuk éppen milyen kód van használatban. Legalul az átvételhez szükséges kódbeviteli mező látható, erről is írtam már korábbiakban.

Jobb alul pedig láthatjuk, hogy az átvételre várók melyik helyen melyik kódot várják, és amint az átvétel megtörtént a mellette lévő gombbal jelezhetjük, hogy ismét felszabadult a hely.

3.9.2 Manuális vezérlés

Ezen az oldalon szerettem volna egy kicsit vizuálisan bemutatni az általam használt elrendezést. Bal oldalt látható függőlegesen a vonat, felette a kamera, jobb oldalt pedig felül a tároló két sorban, alatta az átvevőhelyek.



22.ábra

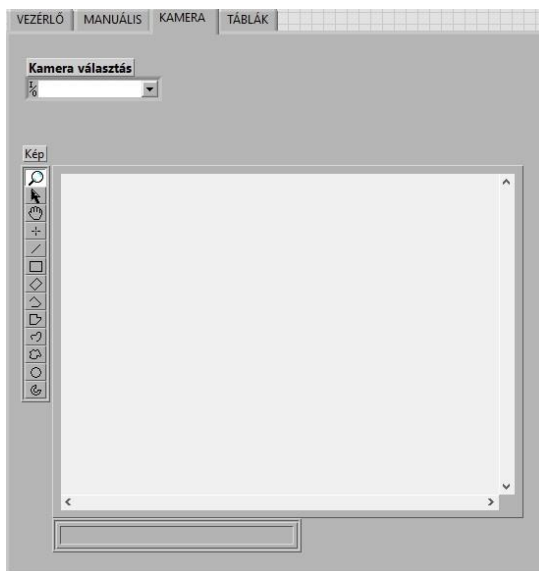
Kezelőfelület manuális kezelőfelülete

Forrás: Saját

A manuális módot a bal felső sarokban lehet aktiválni. Teendők csupán annyi, hogy egy LED-re nyomva kiválasszjuk az úticélt, majd eldöntjük, hogy megfogás vagy elengedés fog történni, ezt bal oldalt a „Megfogó” nyomógommbal szabályozhatjuk, majd alatta a „KÜLDÉS” gombbal elküldjük a parancsot a robotnak.

3.9.3 Kamera és táblák

A következő két fület egyben mutatom be egyszerűségük végett.



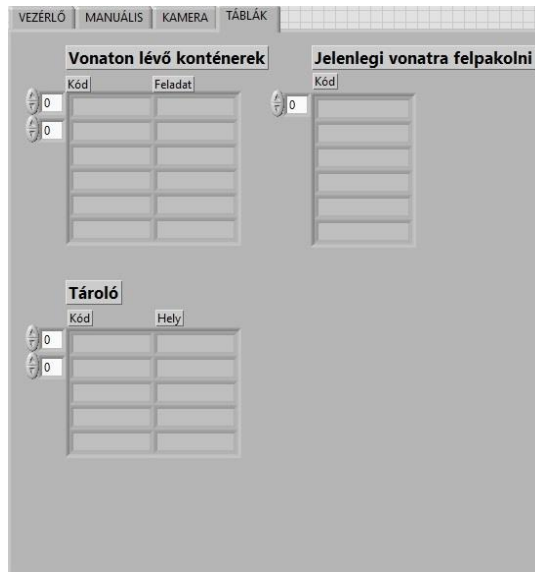
23.ábra

Kamera oldal

Forrás: Saját

A kamera oldalon egy lenyíló fülön lehetőségünk van kiválasztani, hogy melyik kamerát szeretnénk alkalmazni, értelemszerűen ideális esetben csak egy kameránk van, de a laptopom rendelkezik egy beépítettel is és hiába adtam meg neki a külsőt alapértelmezettnek, ha megnyitott LabVIEW közben csatlakoztattam, azt néha felcserélte őket, ezért raktam be ezt a választó menüt pluszban. Alatta látható a megjelenítő felület a kameraképhez.

A „TÁBLÁK” névű ablakban csupán elhelyeztem az SQL táblák tartalmát, melyek valós időben frissülnek. A 23. ábrán a feliratozáson látható melyik mire szolgál, és melyik oszlop mit tartalmaz.



24.ábra

Táblák oldal

Forrás: Saját

3.10 Tesztelés

Az iskolában lehetőségem volt tesztelni a program működését a roboton, így felismertem, és kijavítottam benne kisebb hibákat. Főként időzízési problémák fordultak elő, így kerültek bele várakozások, és szabályozás, mivel volt rá eset, hogy a robot kétszer olvasta ki ugyan azt feladatot, így elhelyeztem egy időzítőt ami csak 3 másodpercig engedi a feladat értéket küldeni a robotnak. Ezen kívül a robot néha indokolatlanul küldött egyes értéket azon a porton amin azt jelzi, hogy végzett. Ez a program váratlan lépését eredményezte, ami felborította a feladat sorrendet. Ezt a portok számának egymástól távolabbi elkülönítése javította meg. Az elrendezéshez csupán három asztalt használtam. Kettő amin hosszában elhelyeztem a hat vonaton lévő konténert, és a robot túl oldalára pedig egy amin a tároló és az átvevő hely lett kialakítva. Végeredményben egy stabil, működő rendszert sikerült elérnem.

4. ÖSSZEFOGLALÁS

Dolgozatomban egy automatizált vonatrakodó modellt alakítottam ki, melyet egy LabVIEW vezérelte Universal Robots UR5 és QR kóddal ellátott műanyagból készült makett konténerekkel készítettem el, ezek mellett szükségem volt még egy megfogó megtervezésére a robothoz. Az állomásra érkező konténereket a robot felveszi, majd egy kamerának mutatja és QR kódban tárolt sorszám alapján egy SQL táblázat segítségével döntés hoz, hogy mi a teendő a megfogott konténerrel, majd azt elhelyezi a megfelelő területre. A tároló területek tartalmáról is automatikusan táblázatot vezet a szoftver.

Az ötlet az elektromos járműfejlesztés fellendülése miatt alakult ki bennem, és ezen belül is a teherszállítási oldalról. Vizsgálatot és számítást végeztem a már létező elektromos hajtású teherszállítási eszköz, vagyis a vonat és a kamionok között, melyek elektromossá tétele jelenkorunkban fejlesztés alatt van. Eredményeim alapján a vonat gazdaságosabb, és a civil életre gyakorolt hatásai elenyészőek a kamionhoz képest. Ezért megpróbáltam egy ötlettel előállni, ami megkönnyíti és felgyorsítja a vonatokon történő áruszállítást.

Ehhez alkotnom kellett egy LabVIEW programot, ami kezeli a webkamera képét és azon QR kódot keres, illetve SQL adatbázisokból kérdez le adatokat, vagy adott esetben beszúr és tárol. Megvalósítottam a robot és a LabVIEW közötti kommunikációt Modbus protokoll segítségével, és kialakítottam egy rendszert, amely döntésmeghozatalok után megfelelő feladatsorszámot küldd a robotnak. Ehhez írnom kellett egy robotprogramot, amely kezeli ezeket, és a megfelelő utasítást hajtja végre. A robotkar egy lineáris hajtáson helyezkedik el, ami a feladat jellegéhez nagyon hasznos volt és folyamatosan használt elem. Ezt egy PLC vezérli, ami a robotprogramból kap indító jelet, illetve abszolút értéket a felvenni kívánt távolságról.

Valós idejű tesztelést is végeztem az iskolában elérhető UR5-ös robotkarral, az általam tervezett megfogóval és konténermakettekkel. Sikeres működést értem el.

Főbb tapasztalataim az elméleti résznél szeretném kezdeni, mégpedig hogy az egész ötlet egy makett, vagyis egy elképzelt vasútállomás kicsinyített mása volt. Arra sikerült jutnom, hogy megéri egy kicsinyített verziót megvalósítani, ha ekkora volumenű dologba akarunk belevágni, de nem ekkora részletességgel. Az általam behatárolt részletgazdagság pont azt hozta magával, hogy sajátossága miatt olyan

problémák is felmerültek, amik a valóságban nem feltétlenül. Gondolok itt például a QR kód kamerának mutatása, ez is annak az eredménye volt, hogy kihagytam azt a részletet, hogy a kamerát a robotra kell erősíteni. Ez hozott magával több felesleges program részt is. Ellenben a makett rávilágított arra, hogyan is lenne érdemes kialakítani a tárolók és egyébek helyét, azokat hogyan kellene adatbázisokban eltárolni, a programfelépítésnek hogyan kellene kinéznie, amik mind-mind nagyon hasznosak. Tehát azt tudom ajánlani, hogy ilyen esetben vagy teljesen el kell veszni a részletekben, vagy csak vázlatosan készíteni a dolgokat, mert ez a középút több gondot is hoz magával.

Következő konklúzióm, a fejlesztő környezettel kapcsolatos. Kezdetben minden áron LabVIEW-ban szerettem volna létrehozni a projektet, mert számomra ez preferált, de be kell ismernem, hogy nem egészen a robot vezérlés a fő profilja, legalábbis nem ebben a formában, amiben én próbálkoztam. Sok esetben úgy éreztem az egész sokkal egyszerűbb lenne, ha egy PLC vezérelné, és a LabVIEW csak a kamerát kezelné, illetve az SQL táblákat. Amennyiben jelenlegi tudásommal állnék neki ennek a feladatnak, biztos emellett döntenék.

Az ilyen feladatok fontos része a tesztelés, lehetőleg minél hosszabb időtartamban. Az én tesztjeim során is több időzítési probléma bukkant fel, melyekre megoldást találva, egyre stabilabban működő rendszert tudtam megvalósítani.

A QR kódok kapcsán azt vettem észre, hogy tényleg nagyon effektív, és nagyon rossz körülmények között is olvasható kód formátum. Érdemes alkalmazni őket ilyen területeken.

A későbbiekben, ha lesz rá lehetőségem szeretnék még visszatérni a feladathoz, sok bővítést rejt még magában.

6. IRODALOMJEGYZÉK

Szűcs Tamás (2016): A járművek menetellenállása, Budapest, 2-4

Geda Gábor (2011): Modellezés és szimuláció az oktatásban, Eger, 3-5

Dr. Gönczy László (2010): Rendszermodellezés, Budapesti Műszaki és Gazdaságtudományi Egyetem, Villamosmérnöki és Informatikai Kar, Méréstechnika és Információs Rendszerek Tanszék, Budapest, 4-5

Tam Jin Soon (2008): QR Code, EPCglobal Singapore, Szingapúr, 2-10

Hainzmann János, Molnár Károly, Scherer Balázs, Tóth Csaba (2011): LabVIEW alapismeretek, Budapesti Műszaki és Gazdaságtudományi Egyetem, Villamosmérnöki és Informatikai Kar, Méréstechnika és Információs Rendszerek Tanszék, Budapest, 2-6

dr. Kovács László (2004): Adatbázisok tervezésének és kezelésének módszertana, Miskolci Egyetem, Gépészmérnöki és Informatikai Kar, Általános Informatikai Intézeti Tanszék, Miskolc, 11-15

Dr. Simon János (2015): LabVIEW 2015 Data exchange and MSSQL DBMS, Szegedi Tudomány Egyetem Mérnöki kar, Szeged, 4-7

Hans-Petter Halsorven (2010): Modbus with practical LabVIEW example, Oslo, 3-10

Anil Kumar Jharotia (2016): Use of Quick Response (QR) Code In Libraries:, Delhi, 3-4, 7-8

Pedro Lopes, Pam Lahoud (2019): T-SQL Querying – Pedro Lopes, Pam Lahoud, Birmingham, 20-22

Szabványok

ISO 9075 (1986): Information technology - Database languages – SQL

EN ISO 10218-1:2011 (2011): Universal Robots A/S

Gyártói brossúra

Universal Robots - Brochure_UR_e_series

<https://www.universal-robots.com/media/1802432/e-series-brochure.pdf>

(Letöltve: 2021. 05. 11.)

Universal Robots: e-series-tech_specs_hun

<https://www.universal-robots.com/media/1803712/hu-e-series-brochure.pdf>

(Letöltve: 2021. 05. 11.)

KÖSZÖNETNYILVÁNÍTÁS

Elsősorban szeretném megköszönni Dr. Sarcevic Péter tanár úrnak, hogy koordinálta a dolgozatom készülését, és hogy segítette megfogalmazni a valódi témát és feladatot, mivel kezdetben csak egy elég céltalan gondolatmenettel álltam elé, amit a bevezetésben is részletezek.

Köszönöm minden segítségét konzulensemnek, Csík Dominiknak, aki felügyelte a gyakorlati munkám, és mindig lehetővé tette, hogy az iskolában gondtalanul haladhassak.

Köszönettel tartozom Mészáros Attila tanár úrnak a 3D nyomtatás megvalósításában, illetve a megfogó pneumatikájának finomhangolásában.

Kötelező megemlítenem Bálint Ádám tanár urat, aki a robotprogramozás résznél kivétel nélkül teljes odaadással próbált segíteni, és szívesen fektetett időt egy-egy elakadásomnál, hogy közösen agyaljunk ki egy megoldást.

Végül pedig az iskolavezetésnek, hogy rendelkezéseimre bocsájtották ezeket a nagyértékű berendezéseket tanulás céljából.

NYILATKOZAT

Alulírott Egervári Bence kijelentem, hogy a szakdolgozat a saját munkám,
minden felhasznált forrásmunkát az irodalomjegyzékben megneveztem.

Szeged, 2021.12.1.



Aláírás