Gépi látás alapú robotcella hatékonyságának vizsgálata virtuális beüzemelés alapján

Efficiency test of a machine vision-based robot cell using virtual commissioning

SZABÓ Balázs László[1], KORSOVECZKI Gyula (ORCID: 0000-0003-0072-2639)[2]

[1, 2] Mechatronikai Tanszék, Debreceni Egyetem, Műszaki Kar
Magyarország, 4028 Debrecen, Ótemető utca 2-4.
Honlap: https://mechatronics.unideb.hu/

Abstract – *The aim of the study is to demonstrate the virtual commission of a machine vision-based robotic cell and to evaluate the performances of the simulation. The presented robotic cell model is combined with simulation solutions to perform workpiece sorting based on virtual camera data, presenting a software solution for complex virtual prototyping using RoboDK robot simulation software, RoboDK API, TwinCAT 3 PLC environment and Python programming language. With these tools and the method it is possible to design virtual commission and test production cells, either in terms of debugging, productivity and cycle times.*

Keywords: Virtual Commissioning, Machine Vision, TwinCAT 3, Python, OPC UA

Kivonat – *A tanulmány célja, hogy bemutassa, hogy egy gépi látáson alapuló robotcella virtuális beüzemelését, valamint a szimuláció hatékonyságának felmérését. A bemutatott robotcella-modell szimulációs megoldásokkal ötvözve virtuális kamera adatai alapján munkadarabok szortírozását végzi, feltárva egy komplex virtuális prototípusgyártásra használható szoftveres megoldást a RoboDK robotszimulációs szoftver, RoboDK API, TwinCAT 3 PLC integrált programozási környezet és Python programnyelv használatával. Az említett eszközökkel és a tanulmányban bemutatott módszerrel lehetőségünk van gyártócellák megtervezésére, virtuális beüzemelésére és annak vizsgálatára, akár hibakeresés, akár produktivitás és ciklusidők szempontjából.*

Kulcsszavak: Virtuális beüzemelés, gépi látás, TwinCAT 3, Python, OPC UA

1. BEVEZETÉS

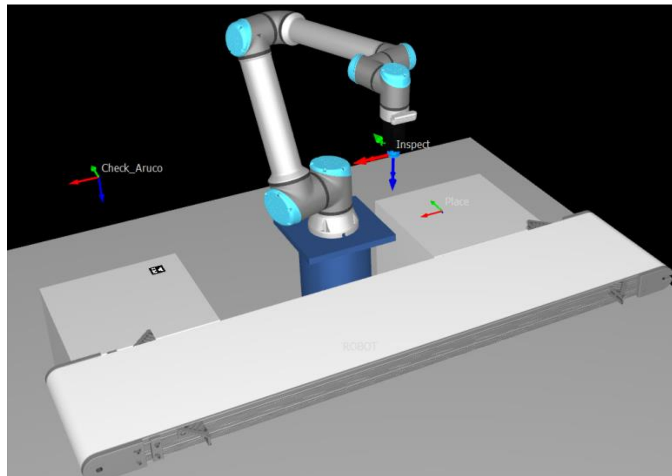
A mai modern gyártástechnológiák nagyon magas szintű érzékelési és kommunikációs képességeket követelnek meg a gyártásban részt vevő gépektől. Az ezen feltételeket kielégítő komplex gyártórendszerek az Ipar 4.0 szerves részét képezik és gyakran hivatkoznak rájuk „okos" gyártórendszerekként. Az effajta mechatronikai rendszereknek nem csak komplexitása növekszik rohamos ütemben, de a tervezésükbe és megvalósításukba fektetett energia és költségek is. Nem meglepő, hogy az iparban is használt számítógépek számítási kapacitásának ugrásszerű növekedésével megkezdődött ezen tervezési folyamatok a virtuális térbe való átköltöztetése [1]. Ez nem csak az üzemi egységek gyorsabb és biztonságosabb üzembe helyezését teszi lehetővé, de anélkül tesztelhetik és optimalizálhatják őket, hogy az nagyobb és váratlan kiadásokba torkollana. Az efféle rendszerek megalkotásához azonban sokszor egyedi szoftveres megoldásokra van szükség, és sok esetben kettő- vagy több ipari applikáció szimultán működésével lehet csak megvalósítani őket [2]. Rogério Adas Pereira Vitalli tanulmányában digitális iker (Digital Twin) alapján közelíti meg a problémát az ipar 4.0 elvei alapján [3]. A robotgyártók is felfigyeltek az igényre és igyekeznek lehetőséget biztosítani erre saját szoftverkomponensükben is. Ilyen példa az ABB Robotstudio, melyet Alaxandru Andrei és társai bemutatnak [4]. További fontos tanulmányok a témában [5], [6] és [7] publikációk. A témának mindazonáltal oktatási relevanciája is van, ahogy azt a [8], [9] tanulmányok is mutatják.

2. A SZIMULÁCIÓS RENDSZER BEMUTATÁSA

A kutatás során a gépelemek kinematikai, mozgástani szimulációja a RoboDK szoftverben történt, míg a vezérlő programkódja a TwinCAT 3 PLC fejlesztőkörnyezetben íródott. A cella összetettsége indokolta a RoboDK API (Application Programming Interface) használatát Python programnyelven. A szoftverek közötti kommunkáció megvalósításához OPC UA (Unified Architecture) kommunikációt használtunk.

2.1. A felhasznált virtuális gépelemek és a munkaállomás előkészítése

A virtuális munkaállomás középpontjában egy Universal Robot U10 hattengelyes robot manipulátor áll. Kollaboratív robot lévén egyszerűen kezelhető, számos robotfunkcióval kompatibilis és a modellhez több megfogó szerszám is használható. Specifikációi közül lényeges az 1300 [mm]-es karkinyúlás és a +/-360 [°]-os csuklótartomány. Ezen felül csuklósebességek terén az 1-2 csuklók sebessége 120 [°/s], míg a 3-6 csuklók sebessége 180 [°/s]. Fontos megemlíteni a teherbírást, ami 10 [kg]. A viszonylag alacsonynak számító karkinyúlást a választott technológiához elegendőnek ítéltük, míg teherbírása reálisan alkalmazkodik a munkadarabokhoz. A robot munkarének fontos részét képezi egy szállítószalag, mely célra a RoboDK általános, egytengelyű mechanizmusára épülő szalagmodelljét alkalmaztuk. Előnye, hogy szabadon választhatók a sebességparaméterek. A technológiából és a munkadarabok tervezett alakjából kifolyólag robotszerszám gyanánt egy vákuumos megfogót választottunk, mely szintjén integráltan elérhető a RoboDK-ban. Ehhez Solid Edge 2022 szoftverben egyéni vákuumtappancs készült a munkadarabok méretének megfelelően. A munkatér előkészítése során 3 db koordináta célpont került definiálásra, melyek egyike a kamerakalibrációhoz szükséges jelölő felett helyezkedik el. A másik célpontot a futószalag felett vettük fel, mely a munkadarabok vizsgálatának alapvető pozícióját jelzi. A harmadik célpont pedig a munkaasztal felett található a lerakás pozícióját reprezentálja. Az 1. ábrán a szimulációs környezet és a célpontok láthatók.



1. ábra. *A szimulációs környezet és a definiált 3 célpont [11]*

2.3. A létrehozott szoftverkönyvtárak

Fontos kritérium volt a rendszer egyszerű indíthatóságát. Bár a program számos alegységet tartalmaz, a forrásnyelv adta lehetőségek megkönnyítették a program architektúra felépítésének. Tekintve, hogy az összes alprogram Python forrásnyelven íródott, könnyen egymásba lehet ezen szubrutinokat ágyazni. Az objektumorientált megoldás során kétféle Python kód keletkezik. Egyrészt az úgynevezett könyvtárak, amelyek futtatható kódot nem tartalmaznak, csupán osztályokba rendezett függvények felsorolását. Másrészt a programok, amelyek valóban futtatók és azok műveleteket hajtanak végre. A program könnyed indíthatósága miatt mindössze egyetlen programot fejlesztettünk, a *Main*, azaz a fő programot. Ez az egyetlen olyan kódrész, amelyet a rendszer üzembehelyezésekor futtatni szükséges. A fő program tehát az egyes könyvtárak függvényeit bizonyos feltételekhez kötve meghívja, így az azok közti sorrendiséget felállítja. Mivel minden, a *Main* program által futtatott *task,* azaz számítási feladat a fő program részeként fut le, nincs szükség arra, hogy egyes, külön-külön futó programrészek közti adatátvitel valamilyen köztes fájlformátumba írva történjen. Ennek kiemelten fontos szerepe van, amikor a képfeldolgozó algoritmus egyes folyamatai által kiadott, minden lefutási ciklusban változó és nagyméretű adatait kell a program többi részének átadni. Az első könyvtár a RoboDK API függvényeit hasznosító, valamint a szoftveren belüli virtuális környezet tartalmára hatással lévő eszközöket magába foglaló *RDK_Tools*. A második az OpenCV

függvényeit használó, képfeldolgozáshoz szükséges függvényeket tároló *CV_Tools*. Végül pedig a szoftverek közti kommunikációt lehetővé tevő OPC UA kliens Pythonos API-ját alkalmazó, függvényeket tároló könyvtár, az *OPC_Tools*.

3. A GÉPI LÁTÁS FUNKCIÓJÁNAK MEGVALÓSÍTÁSA

3.1. A kép beolvasása és a küszöbértékek (threshold) beállítása

A forráskód többi részéhez hasonlóan a képfeldolgozó algoritmus függvényei is annak saját könyvtárában kerültek definiálásra és azokat minden ciklusban a fő programból hívjuk meg. Egyetlen függvény meghívásával kezelhető a teljes képfeldolgozó folyamat, az azon belüli események ugyanis egymásba vannak ágyazva. Az RDK 2D kamerájának képét adott időpillanatokban rögzítettük és fájlként egy adott útvonalra mentettük. Ezt az útvonalat tudja a képfeldolgozó algoritmus elérni, és onnan a fájlt beolvasni. Erre az RDK saját, *Cam2D_Snapshot()* függvényét alkalmaztuk. A *.png* fájlkiterjesztésű képet immáron a Pythonból egyszerűen el lehet érni. A *CV_Tools* könyvtár első függvényében a *.png* kép szürkeárnyalatos módban kerül beolvasásra. Ilyenkor a képet egy változóhoz rendeljük, ami a *read_img()* függvény kimenetével egyenlő. A függvény kimenete szürkeárnyalatos esetben egy 2D mátrix, amiben a kép méretéből adódóan 480 pixelsor, és azokon belül 640 pixeloszlop található. A szürkeárnyalatos képen *thresholding*, azaz küszöbértékelés folyamatot kell elvégezni, hogy a szűrendő alakzatokat elkülönítsük azok hátterétől. Ilyenkor a kép minden pixelét egy fekete vagy fehér színhez rendeljük, így egy bináris képet kapunk. A folyamat egyszerű: az a pixel, ami a küszöbérték alá esik fekete (0), ha felé akkor fehér (255) értékhez rendeljünk. Ez az intervallum a pixelek 8-bites (*unsigned integer*) mivoltából ered.

3.2. Az ArUco jelölő (marker) használata és a kontúrok szűrése

Ahhoz, hogy a képfeldolgozást pozíciómeghatározásra használjuk fel, valamilyen kapcsolatot kell meghatározni a megfigyelt fizikai jelenet és annak kétdimenziós vetülete között. Erre a legegyszerűbb módszer az ArUco jelölő használata. Az ArUco jelölő egy fekete alapon fehérrel jelölt bináris mátrix. A fekete alap gondoskodik a jelölő könnyű beazonosításáról, míg a bináris mátrixszal valamilyen információt képes tárolni. Attól függően, hogy mekkora jelölőt hozunk létre a mátrix, és a benne tárolt adat mérete is változtatható. Ebben az esetben a jelölő mérete bír fontos szereppel. A jelölőt az *OpenCV cv.aruco.detectMarkers()* függvényével lehet beazonosítani. Valós fizikai környezet esetében a jelölőt nyomtatott formában lehet megjeleníteni a képsíkban. Jelen esetben egy kisméretű 3D modellre, mint grafika felhelyeztük, majd grafikával együtt a szimulációs térbe importáltuk. Az imént említett függvény kimenete a jelölő négy sarkának koordinátái pixelekben. Ezen négy pontból ki lehet számítani a jelölő kerületét pixelekben, majd azt elosztva a jelölő valós, "fizikai" kerületével megkapjuk az algoritmus talán legnagyobb jelentőséggel bíró adatát, a pixel-milliméter hányadost. A jelölő beazonosítása után lehetséges más objektumok beazonosítása is a kontúrjának szűrésével. Jelen esetben mivel a képen több objektum, és azokon akár belső kontúrok is megjelennek, akkor a külső körvonalak között hierarchia került felállításra. A folyamathoz a *RETR_CCOMP* metódust alkalmaztuk.

2. ábra. *A munkadarabok a megfelelő kontúrok kiszűrése után [11]*

3.4. A milliméter-pixel konverzió, a munkadarabok szűrése és keresése

Az objektum kontúrjának kinyerését követően a kontúr köré befoglaló négyszöget vettünk fel, amely visszaadja a kontúr elfordulását radiánban, a befoglaló négyszög középpontját pixelekben és a négyszög szélességét, valamint magasságát. Utóbbi adatokat a pixel-milliméter állandóval elosztva megkaptuk a

munkadarab valós méretének közelítő becslését. A négyszög középpontjából (P_{wc} – *'work center point'*) és a kép középpontjából (P_{ic} – *'image center point'*) euklidészi távolságot számítottunk a koordináták kivonásával (D_{icw} , *'image-center – work distance'*), majd az eredményt a pixel-milliméter állandóval elosztva megkaptuk a munkadarab középpontjának távolságát a kép középpontjától milliméterben (D_{mm}). Ezen pont az, amihez a robot TCP-jét vezérelni kell a megfogás során. Az elfordulást radiánból szögbe való átváltással a TCP elfordításának mértékét is lehetséges volt meghatározni (r_w – *'rotation work'*). A függvény eredményeként kapott értéket tehát az RDK_Tools könyvtár *calc_pick_pose()* függvénye használja fel, ami a cellában elhelyezett *PICK_FRAME* koordinátarendszert minden ciklusban, a detektálási sík középpontjához képest D_{mm} -mel tolja, és r_w-vel forgatja el.

$$P_{ic} - P_{wc} = D_{icw} \quad (1)$$

$$D_{icw} / R_{pmm} = D_{mm} \quad (2)$$

A kontúrok pixelben mért területének kinyerésére a *cv.contourArea()* függvényt alkalmaztuk, mely méret alapú szűrést tesz lehetővé. Ez hasznos olyan helyzetekben, mint például a munkadarab látómezőbe érkezése, így ugyanis elkerülhető, hogy már az előtt mérjünk, mielőtt a teljes munkadarab beérkezett volna a vizsgált területre. Az objektumok beazonosítása a *cv.matchShapes()* paranccsal történik. Ennek bemenő paraméterként két kontúrt kell megadni. A referencia kontúr egy külső, a referencia munkadarabról készült képről került beolvasásra, majd ahhoz hasonlítottuk a pillanatnyilag képen látható kontúrt. A függvény a két kép úgynevezett "Hu momentumainak" (Ming-Kuei Hu után elnevezve) kiszámítását veszi alapul. Ezen momentumok olyan, transzlációra, rotációra és méretre invariáns értékek, amelyek megadják a képen lévő pixelek intenzitásának súlyozott átlagát, tehát a vizsgált alakzat tulajdonságait írják le. A *cv.MatchShapes()* parancs ezen hét (a két kontúr miatt 14) paramétert számolja ki, majd azok összehasonlításával meghatározza két alakzat közti különbséget. A kapott érték minél közelebb esik 0-hoz, annál jobban hasonlítanak a vizsgált alakzatok. Az 1. táblázat szemlélteti a programtervezési egységeket.

Programtervezési egységek 1. táblázat [11]

Egységek	Leírás	Kimenet
fb_00_ConvControl(FB)	Feladata a szalagvezérlés. Az egyező munkadarabok esetén *(GVL1.DO_01_MATCH_FILTERED = TRUE)* a szalagot megállítja, majd a tetszőlegesen megszabott szüneteltetési idő után újra indítja.	CONV_RUN (bool) CONV_STOP (bool)
fb_01_FilterMatches(FB)	A találatok szűrését végzi, megszünteti a képfeldolgozó algoritmus által generált prellegő hatást egy időzítő használatával.	MATCH_FILTERED (bool)
fb_02_Visu(FB)	A HMI-n létrehozott gombok és feliratok műveleteiért felelős funkcióblokk.	-

4. A VEZÉRLŐ POGRAM ÉS A KEZELŐFELÜLET FEJLESZTÉSE

Első lépés a rendszer szempontjából a vezérlés megkívánt valósidejűségének biztosítása volt, melyhez, a TwinCAT 3 saját feladatütemezését használtuk. A TwinCAT 3 rendszerbeállításainál a szoftvert futtató számítógép processzorának egy magját izoláltuk, így lehetőség nyílt a valós-idejű feladatütemezésre a Windows környezeten belül is. A PLC feladatok ciklusidejét 10 [ms]-ra határoztuk meg. Az erőforrás optimalizálás során az előre meghatározott alacsony PLC memóriahasználat miatt az előzetes számítás alapján nem okozott gondot a túlnyomórészt globális változók használata. Néhány kivételével az összes PLC programban definiált változó globális, hogy az OPC szerver számára később elérhetővé tett változók könnyen kezelhetők és bárhonnan elérhetőek lehessenek. A cella funkcionális egységeihez külön funkcióblokk programszervezési egységek készültek, melyeket a *Main* fő program POU-ban (Program Organising Unit) hív meg ugyanilyen sorrendben. A POU-k mindegyike létra programnyelven íródott. Mivel valós vezérlő hardver a projektben nem került felhasználásra, így fizikai ki és bemenetei sincsenek, ezért a PLC programban definiált változók memória-címzésére és maszkolására nem volt szükség, a változók közvetlenül a ki és bemeneteknek feleltethetők meg. A futószalag vezérlése a képfeldolgozó algoritmus által visszaadott találati értékre indul. Amennyiben találatot érzékelt a rendszer annak szűrésére van szükség, ugyanis előfordul, hogy rövid időkre a mérés a találati tartományba esik. Ezt a prellegő hatást egy időzítővel küszöböltük ki. Ha a találat egy bizonyos ideig fennáll, akkor a szűrt találatot tároló *bool* típusú változó

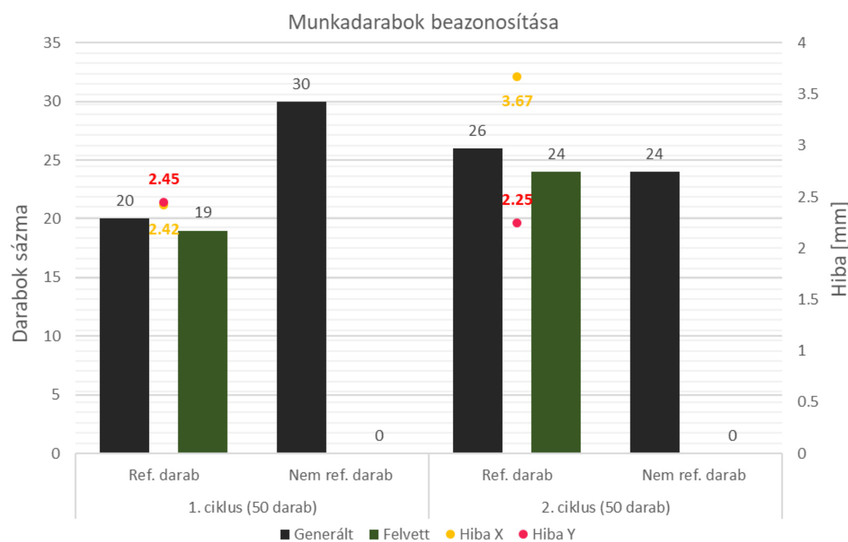
igazra vált, a futószalag megáll, a robot pick & place szubrutinja pedig elindul. Ha a letelik a futószalag megállásakor indult időzítő, a futószalag újraindul és a folyamat újra kezdődik. A HMI megvalósítására a TwinCAT 3 vizualizációs eszközeit alkalmaztuk. A könnyed kezelhetőség és az oldalak közti váltások folyékonyabb megvalósítása érdekében a tervezés fő célja az ablakok számának csökkentése volt. így csak fő ablakot és felugró ablakokat alkalmaztunk. A fő ablakon lévő *START CAM* és *START CONVEYOR* billenő-gombok egy-egy OPC változóra vannak címezve. Ezen kívül helyet kapott egy *EXECUTE COMMAND* gomb, melyből a robothoz fűződő szubrutinok indíthatók. Hasonlóan működik a *CONVEYOR CONTROL* gomb, ami pedig a futószalag beállításait tároló felugró ablakot nyitja meg.

5. AZ OPC UA KOMMUNIKÁCIÓ MEGVALÓSÍTÁSA

Az OPC szervert a TwinCAT 3 *OPC UA Server* kiegészítője futtatja, amely egy olyan interfészt biztosít, amelyen keresztül a szerverre kapcsolódott kliensek elérik a TwinCAT-ben futó PLC, OPC számára elérhetővé tett adatterületeit. Ahhoz, hogy egyes változókat az OPC szerver címterének hozzáférhetővé tegyünk, a változólistán belül az adott változó elé a *{attribute 'OPC.UA.DA' := '1'}* kommentet kell beilleszteni. Ezen módszerrel nemcsak egyszerű regisztereket, de nagyobb adatterületeket foglaló, több regiszterből álló adatstruktúrákat is képes a szerver címterületére beolvastatni. A TwinCAT 3 *Sample Client* kiegészítőjével egy egyszerű klienst modelleztünk és a szerverre felcsatlakozhatunk a benne lévő változók megtekintése céljából. Ezzel a módszerrel a *Main* fő programon belül minden, a PLC-ben deklarált OPC változóra feliratkozásokat készítettünk, így a Python program azonnal értesül az új értékről. A szerver futtatásakor az OPC szerver beolvassa a tetszőlegesen meghatározott felérési úton lévő úgynevezett *Symbol file*-t, amely alapvető információkkal szolgál minden, a PLC projektben található PLC változóról. Ez fájl alapján képes a szerver felépíteni saját címterületét, amelyben az egyes szerveren elérhető ID-k szerepelnek.

6. A RENDSZER TESZTELÉSE, AZ EREDMÉNYEK ÉRTÉKELÉSE

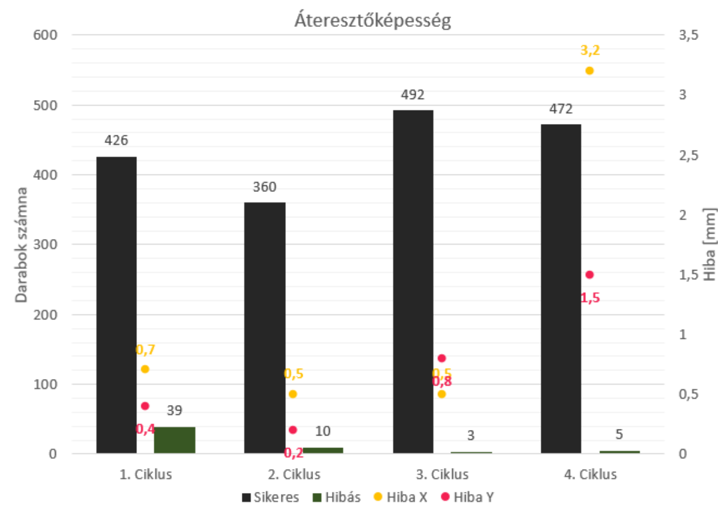
A tesztelést több fázisban történt, melyek különböző szemponttok alapján vizsgálták a rendszer hatékonyságát. Az első teszt célja, hogy számszerűsítse, mekkora arányban képes a rendszer a referencia darabokat felismerni és beazonosítani, valamint azok megfogási pontját bemérni. A fázist 100 munkadarabra vonatkozóan végeztük el. A teszteredmények a 3. ábrán láthatóak.



3. ábra. *Az munkadarabok beazonosítására irányuló teszt eredményei [11]*

A második tesztfázisban azt vizsgáltuk, mi az a legnagyobb hatékonysági szint, amelyen a rendszer működni képes. Ezen hatékonyság növelését a teszt kétféleképpen igyekszik elérni. Elsőként a szalagon érkező termékek gyakoriságának növelésével, valamint a mechanizmusok sebességének növelésével így pedig a ciklusidő csökkentésével. A teszt eredménye megmutatja, hogy mekkora a rendszer maximális áteresztőképessége, tehát az adott idő alatt maximálisan teljesíthető sikeres ciklusok száma. Ezen tesztfázis első feladata az volt, hogy a robot felvétel és lerakás szubrutint a lehető legrövidebb ciklusidőre gyorsítottuk. Következtethetünk, hogy a JMOVE mozgások esetében érdemes a maximális 120 [°/sec] sebességet alkalmazni, míg a lineáris mozgások esetében 600mm/sec-es sebességet meghaladva a ciklusidő

számottevően nem változik. A működésvizsgálat négy, egy órás ciklus formájában játszódott le. A teszt során kapott eredményeket a 4. ábra szemlélteti.



4. ábra. *Az áteresztőképesség meghatározására irányuló teszt eredményei [11]*

A bemutatott eredmények alapján a tervezett szimulációs rendszer olyan igényekre kínál megoldást, mint robotcellák virtuális beüzemelése, megtervezése és hatékonyságának előzetes felmérése. A rendszer rugalmas bővíthetőségéből és komplexitásából adódóan a mechatronikai mérnöki oktatásban is alkalmazható, mely során a végzős hallgatók megismerhetik a valós eszközök digitalizációjának szempontjait és követelményeit, a software-in-the-loop tervezési elveit, valamint a taglalt programnyelvek szintaktikai jellegzetességeit.

IRODALMI HIVATKOZÁSOK

[1] Noga, M.; Juhás, M.; Gulan, M. Hybrid Virtual Commissioning of a Robotic Manipulator with Machine Vision Using a Single Controller. Sensors 2022, *22*, 1621. https://doi.org/10.3390/s22041621

[2] Ye Zhao, Luis Sentis,Distributed impedance control of latency-prone robotic systems with series elastic actuation, Sability, Control and Applivation of Time-delay Systems, 2019, pages 23-51. https://doi.org/10.1016/B978-0-12-814928-7.00002-0.

[3] R. A. P. Vitalli, Digital twin virtual commissioning of robotic cells based on the Industry 4.0 context, International journal of Science Academic Research, Vol. 02, Issue 02, pp.1149-1151, February, 2021.

[4] A. Andrei, A. F. Nicolescu, C. Pupaza; Perpectives of Virtual Commisssioning using ABB Robotstudio and Simatic Robot integrator environments: A review; Proceedings in Manufacturing Systems, Volume 16, Issue 3, 2021, 117-123, ISSN 2067-9238.

[5] S. Makris, G. Michalos, G. Chryssolouris; Virtual Commissioning of an Assembly Cell with Cooperating Robots; Hindawi Publishing Corporation, Advances in Decision Sciences, Volume 2012, Article ID 428060, 11 pages, doi:10.1155/2012/428060.

[6] M. Ugarte, L. Etxeberria, G. Unamuno, J. L. Bellanco, E. Ugalde; Implementation of Digital Twin-based Virtual Commissioning in Machine Tool Manufacturing; Procedia Computer Science, Volume 200, 2022, Pages 527-536.

[7] P. Rueckert, S. Muenkewarf, K. Tracht; Human-in-the-loop simulation for virtual commissioning of human-robot collaboration; Procedia CIRP, Volume 88, 2020, Pages 229-233.

[8] I. A. Fernández, M. A. Eguía and L. E. Echeverría, "Virtual commissioning of a robotic cell: an educational case study," *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Zaragoza, Spain, 2019, pp. 820-825, doi: 10.1109/ETFA.2019.8869373.

[9] R. Raffaeli, P. Bilancia, F. Neri, M. Peruzzini, M. Pellicciari; Engineering Method and Tool for the Complete Virtual Commissioning of Robotic Cells; Applied Sciences, Volume 12, Issue 6, 10.3390/app12063164.

[10] TEch Con, Universal Robot UR10. [Online]. Available: https://universal-robots.tech-con.hu/?gclid=EAIaIQobChMIivrK2f3EgQMVDpGDBx1O6gGsEAAYAiAAEgIe1vD_BwE. (Utolsó letöltés: 2023. 09. 19.).

[11] B. L. Szabó, Gépi látás alapú robotcella hatékonyságának becslése virtuális beüzemelés alapján. Debreceni Egyetem, Műszaki Kar, Mechatronikai Tanszék, 2022.