Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Ütközésvizsgáló és pályatervező algoritmusok ipari robotok számára

TDK dolgozat

Készítette:

Zahorán László

Konzulens:

Dr. Kovács András
Dr. Hullám Gábor

2019

Tartalomjegyzék

Kivonat

A robotika és digitalizáció terjedésével egyre nő azoknak a feladatoknak a köre, amelyeket szeretnénk a lehető legmagasabb automatizáltság mellett elvégezni. Egyre nagyobb teret nyernek a rugalmasan alakítható gyártósorok. Ehhez nagy mennyiségben van szükség automatikusan, akár valós időben generált ütközésmentes robotprogramokra.

Az említett ipari trendek közös pontja, a folyamatban résztvevő ipari robotok mozgásának automatikus tervezése, legyen szó szerelési, megmunkálási és anyagmozgatási feladatról. A pályatervező algoritmus teljesítményét leginkább befolyásoló művelet a cellában elhelyezett geometriák ütközésvizsgálata, ami valós idejű vagy kollaboratív alkalmazásoknál kiemelt fontosságú. Az ütközésvizsgálatot nem csak a robot és a cella egy-egy statikus konfigurációjában kell tudni elvégezni, hanem a robot lehetséges folytonos mozgásaira is. Jelenleg erre a célra leginkább hagyományos mintavételezés használt, ennél azonban ismertek hatékonyabb módszer is, ún. *Conservatice Advancement* eljárások, amelyek képesek formális garanciát adni az ütközésmentes mozdulatokra.

Ipari robotos pályatervezés nehézsége, hogy egy általános hatcsuklós ipari robot esetén az állapottér is hat dimenziós. Ezzel a kihívással leginkább randomizált útkereső algoritmusok képesek megbirkózni. Ezek közül a *Rapidly-exploring Random Trees* (RRT) és a *Probabilistic Roadmaps* (PRM) eljárások a legjelentősebbek. Előbbi minden pályatervezési kérést előfeldolgozás nélkül hajt végre az aktuális cellaállapotnak megfelelően (ún. *single-query* eljárás), míg utóbbi egy előre elkészített ütközésmentes térkép segítségével teszi ezt (*multi-query* megközelítés).

Jelen dolgozatban az említett ütközésvizsgálati és pályatervező algoritmusok irodalma, implementálása, továbbgondolása és teljesítménymérése kerül áttekintésre. Az algoritmusok implementálásához egy, az alapjaitól saját kézben tartott környezet készül el, amely tartalmazza az ehhez szükséges geometriai és kinematikai modelleket, ütközésvizsgáló és pályatervező algoritmusokat, valamint a tervezéshez használt cella és a kiszámított robotmozgások vizualizációját. Az elkészült programkönyvtár felhasználásával tervezett pályákat szimulált környezetben és a valóságban is *UR5, UR10* kollaboratív ipari robotok segítségével teszteljük és az így gyűjtött tapasztalatok szintén megjelennek. Esettanulmányként két valós ipari alkalmazás, egy kamerás anyagmozgatási (*pick-and-place*), valamint egy összeszerelési művelet automatizálása kerül bemutatásra, amelyek a megvalósított eljárásokat teljeskörűen felhasználják.

Abstract

With the spread of robotic applications, the range of tasks we want to accomplish with the highest degree of automation is growing. Flexible production lines are gaining ground. This requires large amounts of automatically generated collision-free robot programs, even in real time.

The common point of these industrial trends is the automated planning of the movement of the industrial robots involved in the process, whether assembly, machining or material handling. The operation that most influences the performance of the path planning algorithm is the collision detection of geometries in the work cell, which has paramount importance in real-time or collaborative applications. The collision test must be performed not only in the static configuration of the robot and the cell, but also has to be capable of examining the possible continuous movements of the robot. Currently, sampling is the most commonly used method for this purpose, however, more effective methods are known, such as *Conservative Advancement* procedures that can provide a formal guarantee of collision-free movements.

The difficulty of industrial robot path planning is that for a generic six-degrees-of-freedom industrial robot, the state space is also six-dimensional. Mostly randomized path finding algorithms can cope with this challenge. These include *Rapidly-exploring Random Trees* (RRT) and *Probabilistic Roadmaps* (PRM). In case of the former method it executes all path planning requests without preprocessing according to the current cell state (so-called *single-query* procedure), while the latter does so using a precomputed collision-free *roadmap* (*multi-query* approach).

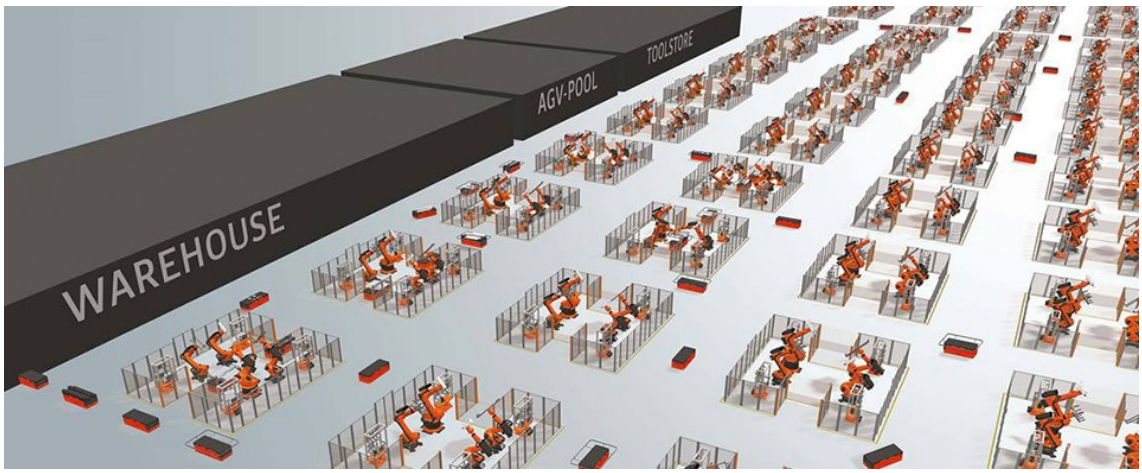
In this paper, the methods, literature, implementation, refinement and performance measurement of the mentioned collision detection and path planning algorithms are reviewed. To implement the algorithms, a self-contained environment is created which includes the necessary geometric and kinematic models, collision detection and path planning algorithms, as well as visualization of the work cell and the computed robot movements. Paths designed using the implemented library are tested in simulated environments and in reality with *UR5, UR10* collaborative industrial robots, and the learned experiences are also discussed. As a case study, a camera-based *pick-and-place* application is presented, which makes full use of the implemented procedures.

1. fejezet

Bevezetés

1.1. Motiváció

A robotika és szenzortechnológia fejlődésével egyre nő azoknak a feladatoknak a köre, amelyeket szeretnénk a lehető legmagasabb automatizáltság mellett elvégezni. Egyre fontosabb a gyártás során a folyamatok mérése, pontos ütemezése, így a hatékonyság és termelékenység növelése. Egyre nagyobb teret nyernek azok a gyártósorok, amelyek rugalmasan alakíthatóak, a lineáris gyártósorok mellett megjelennek a mátrixba rendezett gyártócellák [5]. Ezek közt a termékek megmunkálása még rugalmasabb mind sorrend és ütemezés terén, mind a felhasznált gyártási metódusok palettáját tekintve, valamint a karbantartási és cella beállítások is kényelmesebben végezhetőek. Az így gyártott szériák egyre kisebbek lehetnek, egyre több konfigurálási lehetőség van a termékek gyártása során. Mindez maga után vonja, hogy a tervezést és optimalizálást a lehető legnagyobb részben szeretnénk automatizált vagy kevés beavatkozást igénylő szoftveres eszközökre hagyni.

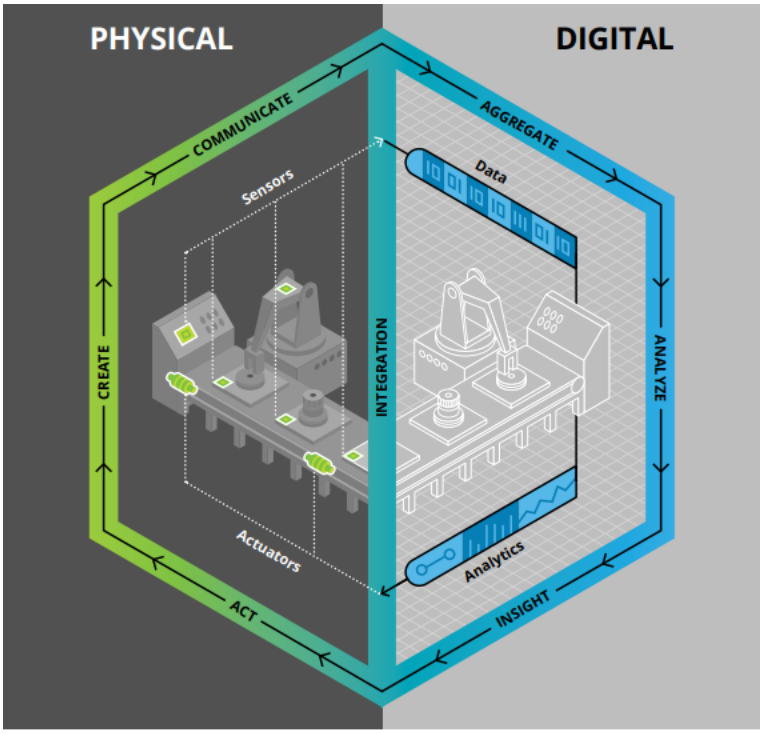


1.1. ábra. Mátrixba rendezett gyártócellák [4]

A korábbi évtizedekhez képest nagyobb hangsúly kerül ember és gép közötti munkakapcsolatra. Eddigi gyártási folyamatok élesen elváltak az alapján, hogy ember vagy egy célgép munkája árán végezzük el a feladatot és nagyrészt kizárólag egy vezérlő jelentette köztük a kapcsolatot. A kollaboratív robotika ezt az éles határt a munkatér megosztásával [11] változtatja meg. A kollaboratív robotok megjelenésével az ember által befektetett energia kényelmesebben és hatékonyabban használható fel, míg a robotok a számukra nehezen megoldható feladatokban, például finom-motoros mozgások elvégzésében szorulnak

emberi segítségre. Mindez új szempontokat teremt a munkafolyamatok tervezése [16] és végrehajtása során, valamint a munkabiztonság is még kiemeltebb tényezővé vált.

A digitális iker [15, 3] szemlélete összefoglalja elképzeléseinket arról, hogy a jövőben hogyan fog zajlani a gyártás. A gondolat lényege, hogy szeretnénk a fizikai valóságban létező vagy létrehozni kívánt gyártási folyamat teljes képét digitálisan is megvalósítani. A valós fizikai gyártósor folyamatos adatokat szolgáltat: ciklusidők, szenzoradatok, kihozatal és selejt-szám a vállalatirányítási és gyártás optimalizáló rendszerek számára. Az így kinyert adatokat a legkülönbözőbb módszerekkel tudjuk értékes gyártási információvá konvertálni, majd ezek mentén változtatni a valós gyártási folyamaton, aminek eredményességéről a visszacsatolásnak köszönhetően újra valós adatokat kapunk. A szemlélet részeként a gyártási folyamat, gyártócella összes szükséges eleme rendelkezésre áll, mint a felhasznált eszközök, munkadarabok, gépek geometriai és kinematikai modellje, így a folyamat ideje, hatékonysága, egyéb paraméterei a korábbiaknál pontosabban felmérhetőek, tervezhetőek. Ezzel a fizikai gyártósoron való hagyományos tesztelés, robotok pályájának "bemozgatással" való tanítása, optimalizálása minimalizálható, hiszen ez költséges művelet a gyártó számára.



1.2. ábra. Digitális ikrek [15]

Az említett ipari trendek közös pontja a folyamatban részt vevő ipari robotok mozgásának autonóm tervezése, legyen szó szerelési, megmunkálási vagy anyagmozgatási feladatról. Jelen dolgozatban ütközésvizsgálati és pályatervezési módszerek tanulmányozása, implementáció elkészítése, hatékonyságuk mérése és hiányosságaik alapján új vagy kibővített módszerek kidolgozása a célom. Az algoritmusok implementálásához egy, az alapjaitól saját kézben tartott környezet készül el, amely tartalmazza az ehhez szükséges geometriai és kinematikai modelleket, ütközésvizsgáló és pályatervező algoritmusokat, valamint a tervezéshez használt cella és a kiszámított robotmozgások vizualizációját. Az implementáció utóélete szempontjából fontos elvárás volt, hogy a létrehozott könyvtár a MTA-SZTAKI-ban [8] újrafelhasználható legyen, további kutatások, alkalmazások során. A labor munkatársai, leginkább mechatronikus- és gépészmérnök kollégák igénye és a már meg-

lévő Microsoft világban (*.NET, C#*) létrejött projektek miatt indokolt, hogy a könyvtár platformja első sorban Windows legyen. Offline tervezési feladatok mellett akár telepített ipari megoldásban önállóan vagy szoftver komponensként legyen képes működni. Az elkészült programkönyvtár felhasználásával tervezett pályákat szimulált környezetben és a valóságban is *UR5, UR10* kollaboratív ipari robotok segítségével teszteljük és az így gyűjtött tapasztalatok szintén megjelennek. Esettanulmányként két valós ipari alkalmazás, egy kamerás anyagmozgatási (*pick-and-place*), valamint egy összeszerelési művelet automatizálása kerül bemutatásra, amelyek a megvalósított eljárásokat teljeskörűen felhasználják.

1.2. Jelenlegi megoldások

Jelenleg számos szoftveres megoldás érhető el, leginkább szimulációs képességekkel amelyek tartalmaznak pályatervezési funkcionalitást. Ezek az alkalmazások magas funkcionalitással rendelkeznek, viszont több szempontból megvizsgálva mindegyikük olyan megkötéseket tartalmazott, ami az algoritmusok implementálása, mérése vagy későbbi használata során akadályt jelentett volna. Az alábbi eszközökkel szemben egy teljes mértékben saját kézben tartott pályatervezési könyvtár elkészítése és mérési környezet létrehozása lett a kitűzött cél, amennyiben lehet, vizualizációval kiegészítve.

Általános tervezési és szimulációs célokra elérhető kész szoftverek, például *RoboDK*[1], *Optopuz*[2] kényelmes és magasszintű megoldásokat nyújtanak mint támogatott robotok könyvtára, ütközésmentes pályák létrehozása, vizualizációja és gyakran a robotok közvetlen vezérlése is megoldott. A legtöbb esetben az alkalmazások rendelkeznek valamilyen szkript nyelvű API-val, amellyel saját igényünknek megfelelően készíthetünk alkalmazásokat. Ezek a kész szoftverek kereskedelmi termék révén, leginkább egy interfészt szolgáltatnak. Belső működésük zártsága miatt az algoritmusok nem vizsgálhatóak vagy használhatóak tetszőlegesen. A laboratóriumban jelenleg napi használatban a *RoboDK* nevű eszköz van, amely *Pyton* API-n keresztül vezérelhető, viszont rendelkezik az említett korlátozásokkal. Emellett, a használat során felmerültek teljesítménybeli problémák amelyek a később szereplő mérések során összevetési alapként szolgáltak. Ezek az alkalmazások korlátozottan használhatóak online feladatok megoldására, egy beágyazott környezetben, például a bemutatott kamerás esettanulmány során, más komponensekkel való integrálhatóságuk rugalmatlan.

A fent említett megoldások mellett a *Robotic Operation System*[3] (ROS) és annak leginkább *ROS-Industrial*[4] (ROS-I) csomagja tekinthető alapvetőnek a felhasználható technológiák között. A ROS széles körben használt nyílt forráskódú, közösség által fejlesztett rendszer, amelyben a robotikai alkalmazásokhoz szükséges funkcionalitás szinte teljeskörűen rendelkezésre áll, mint kommunikáció, geometriák és modellezési eszközök, szenzor adatok kezelése és emulációja, integráció ipari megoldásokkal, diagnosztikai és navigációs szolgáltatások. Emellett a vizualizációhoz is hasznos eszközök ingyenesen állnak rendelkezésre, például *Gazebo, rviz*. A *ROS Core* illetve ROS-I funkcionalitásának megfelelően nagy méretű kódbázissal rendelkezik, ez kiegészül a közösség által fejlesztett egyéb szükséges bővítményekkel, amelyek különböző programozási nyelveken és konvenciókkal készültek. Így a rendszer rugalmas használata komoly járatosságot igényel, ami a jelenlegi projekt méretein túlmutat. Az alapvető funkcionalitás biztonsággal használható, viszont a komponensek nem rendelkeznek gyártói és a közösség által vállalt garanciával, ami esetleges kritikus ipari rendszerekben szintén nem elfogadható. A ROS alapvetően Linuxos környezetben támogatott, azonban a *Windows Subsystem for Linux* megjelenésével van

[1] https://robodk.com/
[2] https://octopuz.com/
[3] https://www.ros.org
[4] https://rosindustrial.org

lehetőség Windows fölött futtatni a rendszert, azonban a tapasztalatok szerint ez még nem minden szempontból kiforrott megoldás. Jelen esetben a könyvtárral kapcsolatban a *.NET*-es *C#*-os komponensekkel való integrálhatóság, valamint *.dll* szerelvényként való alkalmazás (akár Wolfram Mathematica, LinkageDesigner tervezés során) elvárás volt. Így a ROS a jelenlegi feladat során a Microsoft világgal való integrációja, mérete, komplexitása, hibatűrése miatt nem célszerű választás.

Egy harmadik az iparban kevésbé szokványos lehetőség játékmotorok, mint *Unity, Unreal Engine* használata. Szintén rendelkeznek kedvező tulajdonságokkal a jelenlegi és későbbi alkalmazás szempontjából, például fizikai motorok, virtuális, illetve kiterjesztett valósággal való integráció, platform-függetlenség, geometria és kinematika építés, megjelenítés teljes körű személyre szabása. Leginkább szimulációra, vizualizációra használható szinte korlátozások nélkül, viszont konzolos alkalmazásként vagy más programok moduljaként nem a játékmotorok tipikus felhasználása. Jelen esetben is az elkészült könyvtár újrafelhasználhatósága, külvilággal, szenzorhálózattal más komponensekkel való integráció lehetőségei korlátozottak.

1.3. Dolgozat áttekintése

2. **Robotos gyártócellák modellezése:** Robotok általános felépítése, tulajdonságai különös tekintettel *Universal Robots UR5*, *UR10* kollaboratív robotokra, általános és mechanikus megfogók tulajdonságai (*RobotiQ*), működésük a fizikai valóságban és modellezésük. Denavit - Hartemberg módszer alkalmazása robotok kinematikai láncához, direkt és inverz kinematikai feladat meghatározása. Egyéb cellaelemek szabadformájú geometriái a gyártócella összeállítása transzformációkkal.

3. **Ütközésvizsgáló algoritmusok:** Ütközésvizsgálathoz szükséges objektumok, szabályrendszer meghatározása és általános ütközésvizsgáló könyvtár lekérdezése. Statikus konfiguráció ütközésvizsgálata. Ütközésvizsgáló algoritmusok mintavételezéssel, *Conservative Advancement* eljárásokkal, irodalmi áttekintésük, megvalósítási megfontolások. Javított becslő eljárás Conservative Advancement módszeréhez és az alapgondolat kiterjesztése megfogott objektumra.

4. **Pályatervező algoritmusok:** Pályatervező algoritmusok (*Rapidly-exploring Random Trees*, *Probabilistic Roadmaps*) irodalmi áttekintése, megvalósítási lehetőségek, alkalmazhatóságuk. Keresőfa helyett általános gráf használata PRM térképként, RRT algoritmus felhasználása lokális tervezőként.

5. **Implementáció:** Az elkészített könyvtár felépítése, moduljai, kialakított interfészek. Felhasznált külső könyvtárak, bővítési lehetőségei és implementált funkcionalitás összefoglalása. Integrálási lehetőségek és grafikai megjelenítő bemutatása.

6. **Kísérleti eredmények:** Az ütközésvizsgálati és pályatervező algoritmusok mérési környezetének bemutatása, teljesítmény mérése. Az esettanulmányként bemutatott alkalmazások ismertetése, kísérleti környezet és a kapott eredmények bemutatása, megvalósított algoritmusok használata a feladatok megoldására.

2. fejezet

Robotos gyártócellák modellezése

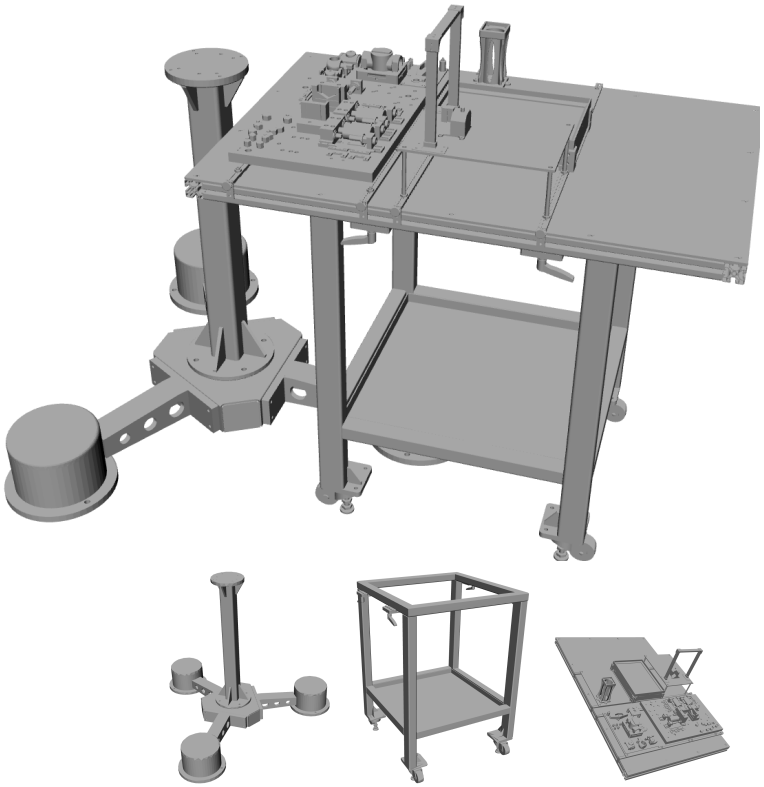
2.1. Ipari robotok

Az ipari robotok és kiegészítőik piaca az elmúlt évtizedben nagy fellendülésen ment keresztül, az elvégzendő feladatnak megfelelően a legspeciálisabb követelményekhez is léteznek kereskedelemben kapható termékek. Az általános ipari robotok a robot tagjaiból (szegmens, *link*) azokat összekötő transzlációs (adott egyenes mentén mozgó) vagy rotációs (adott tengely körül elforduló) csuklókból (*joint*) és a végberendezésből épülnek fel. Legfőbb paramétereik a csuklók száma, a maximális teherbírás és bejárható munkaterület. Az általános csuklószámnak a hat tekinthető, mert ez biztosítja, hogy a munkatér tetszőleges pozíciója tetszőleges orientációval elérhető legyen. Amellett, hogy a mai napig megvan a három vagy kevesebb szabadsági fokkal rendelkező manipulátorok, delta robotok felhasználási területe. A hagyományos ipari robotokkal szemben kollaboratív társaik számos ponton eltérnek. Munkaterületük és terhelhetőségük kisebb, a csuklók gyorsulási és sebesség értékei kollaboratív használat során kötelezően limitálandók. A csuklókban erőszenzorok vannak elhelyezve, amiknek köszönhetően az esetleges ütközések érzékelését vagy a bemozgatással való tanítást meg lehet valósítani. Felhasználásuk leginkább anyagmozgatás, egyszerűbb megmunkálás, összeszerelés, palettázás és csomagolás. A végberendezéseket tekintve rengeteg speciális termék kapható. Legtöbb alkalmazásban mechanikus, vákuumos vagy elektromágneses elven működő végberendezések használatosak. A feladat tervezése során maximálisan hét rotációs csukló kezelése a kitűzött cél, mivel ez a legtöbb esetben elegendő. A megfogókat tekintve a mechanikus megfogók kezelése a végcél, mivel ennek a feladatnak a megoldása már magában foglalja az ilyen szempontból könnyebben modellezhető statikus végberendezéseket (például: festékszóró, 3D szkenner, vákuumos vagy mágneses megfogók).

A későbbi esettanulmányok egy UR5-ös és UR10-es robotokat használnak, így ezek modellezésre kerülnek. Mind a két robot hat rotációs csuklóból és hét tagból áll, munkaterületük 0,85/1,3 méter és 5/10 kg teherbírás jellemzi őket. A felhasznált RobotiQ megfogó releváns paramétere a maximális pofatávolsága, ami 85 mm és a tervezett maximális teher 5 kg.

2.2. Geometriai modellek

A cellában felhasznált elemek modelljei képzik az ütközésvizsgálat és a megjelenítés alapját. Szabadformájú geometriák kezeléséhez szükségünk van a geometriát leíró ponthalmazokra, háromszöghálókra és a pontos helyzetüket meghatározó homogén lineáris transzformációkra. A testek háromszöghálójának tárolására több fájltípus létezik, amelyek rögzítik a háromszögek csúcsait, normálvektorukat, valamint egyéb szín és textúra információkat. A *CAD* tervező programok rengeteg ilyen kiterjesztést támogatnak, jelen esetben az *.slt* formátum kerül felhasználásra a geometriák feldolgozásának bemeneteként, ami talán az egyik legegyszerűbb ezek közül. A implementált ütközésvizsgáló könyvtár külső függőségeinek növelésével a támogatott formátumok köre könnyen bővíthető például az *Open Asset Import Library*[1] felhasználásával.



2.1. ábra. Gömbcsap szerelésére és kábelvégek rendezésére felállított moduláris munkaállomás robot állvánnyal

A Descartes-féle koordináta-rendszerben való leíráshoz egy x, y, z tengely szerinti $\vec{t} = (t_x, t_y, t_z)$ eltolás vektorra van szükségünk, valamint az orientációt meghatározható három tengely körüli, α, β, γ elforgatást leíró szögekre. Skálázásra alapesetben nincs szükség, mivel a modellezett geometriák egymással és a fizikai tárgyal is méretarányosak, de a teljesség igénye miatt egy $\vec{s} = (s_x, s_y, s_z)$ vektor a három egymásra merőleges tengely szerinti skálázásához bevezetésre kerül. A felparaméterezett 2.1, 2.2, 2.3 transzformációs mátrixok $R, T, S \in \mathbb{R}^{4\times4}$, ahol $R = R_x R_y R_z$. A mátrixok ebben a sorrendben való szorzásával kaphatóak meg a cellaelemek végleges transzformációi. Így a geometriák beolvasásával és transzformálásával tetszőleges gyártócella modellje felépíthető (2.1 ábra). Ilyen módon

[1] https://github.com/assimp/assimp

a robotok, megfogók összeállított modellje (2.4. ábra) is tovább transzformálható, ezzel a cellán belül tetszőlegesen elhelyezhető.

$$R_x R_y R_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & \text{-}\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ \text{-}\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\gamma) & \text{-}\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

$$T = \begin{pmatrix} 1 & 0 & 0 & \mathrm{t}_x \\ 0 & 1 & 0 & \mathrm{t}_y \\ 0 & 0 & 1 & \mathrm{t}_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

$$S = \begin{pmatrix} \mathrm{s}_x & 0 & 0 & 0 \\ 0 & \mathrm{s}_y & 0 & 0 \\ 0 & 0 & \mathrm{s}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

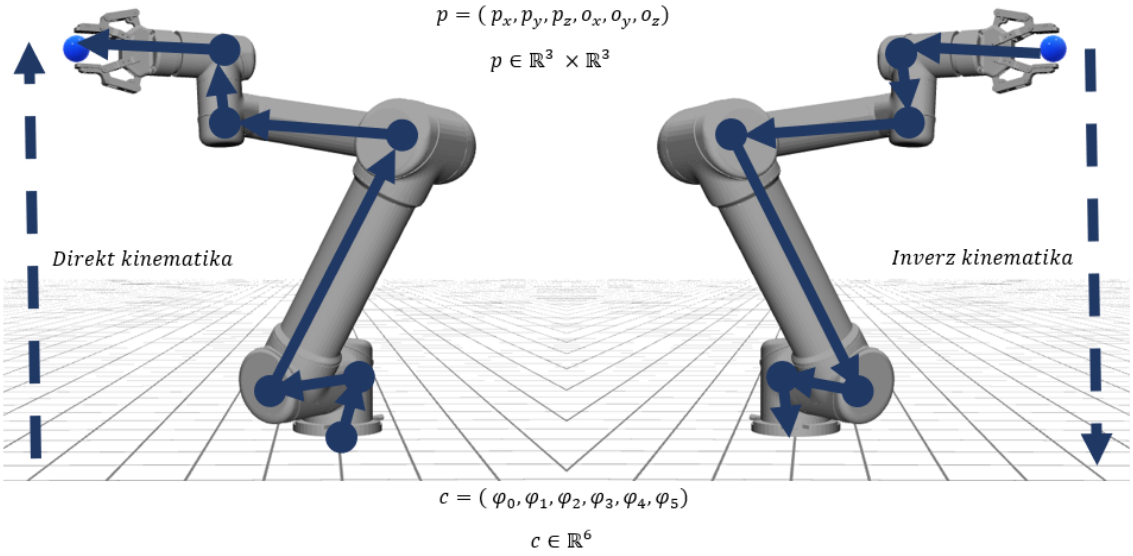
2.3. Robotok kinematikai modellezése

A létező szoftveres megoldásoknak alapvető zárt részét képzi a robotok megvalósított mozgása. A saját megvalósítást többek közt az is indokolta, hogy a geometriák pontos ismeretében tudunk az ütközésvizsgálat során kedvezőbb eredményt elérni. A robotok geometriái rendelkezésre állnak, olyan transzformációkat kell keresnünk amelyek ezeket az elemeket a fizikai valóságnak megfelelően tartják össze. A következőkben a minimálisan szükséges fogalmak, helyenként eltérő jelölésekkel kerültek felhasználásra a [19, 18] könyvekből. A robot pontos állapotát a csuklószögek írják le, egy statikus konfiguráció $c = (\varphi_0, ..., \varphi_{m-1}) \in C$, ahol C az m csuklós robot konfigurációs tere. A rotációs csuklókkal rendelkező robotok kinematikai láncának leírására a Denavit - Hartenberg módszer alkalmazható. Alapvető feltétel, hogy a modellezett geometriák, robot alkatrészek forgáspontja (amennyiben két forgásponttal rendelkezik, az alsó) az origóba essen. A módszer alkalmazásához a konfiguráció szögeire és a forgáspontok adott tengely menti eltolásának mértékére van szükség, az alábbi formában.

- d_i - Kijelölt csuklók távolságát jelöli, azaz a két forgáspont közötti, az i csukló z_i tengelye mentén mért távolság.
- a_i - Távolság az i és $i+1$ csuklótengelyek közös normálisának a hossza, amelyet az x_i tengely mentén mérünk.
- φ_i^x - Rotációs csukló esetén az x_{i-1} tengely és az x_i tengely között bezárt szög nagysága jobbsodrású rendszerben.
- φ_i^z - Az i. csukló z_{i-1} tengelye, valamint az z_i tengelye között mért szög, az φ_i^z paraméterre merőleges síkban jobbsodrású rendszerben.

Ilyen módon minden elem orientációja és elhelyezkedése pontosan megadható az előzőhöz képest (2.4 egyenlet). A kinematikai lánc tagjain a robot talpazatától felfelé haladva a Denavit-Hartemberg mátrixok szorzásával minden tagra pontosan kiszámítható a szükséges homogén transzformációs mátrix, ami magában foglalja az előző elemeket (2.5 egyenlet). Az így végig számított lánccal megkaptuk a robotunk digitális mását c konfigurációban. Ezzel megoldásra kerül egy ún. *direkt kinematikai* feladat, amely c konfiguráció

ismeretében meghatározva $p \in \mathbb{R}^3 \times \mathbb{R}^3$ pontot, ami a végberendezés vagy a robot végpontja a munkatérben, Descartes koordináta-rendszerben. Ennek a műveletnek fordított irányú elvégzése az *inverz kinematika*, ami az előzővel ellentétben komplex és egyértelműen nem oldható meg. Mivel egy p ponthoz több c konfiguráció is tartozhat.



2.2. ábra. Direkt és inverz kinematika

$$
{}^{i-1}D_i(\varphi_i^x, d_i, a_i, \varphi_i^z) =
$$

$$
\begin{pmatrix} \cos(\varphi_i^x) & \text{-}\sin(\varphi_i^x) & 0 & 0 \\ \sin(\varphi_i^x) & \cos(\varphi_i^x) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi_i^z) & \text{-}\sin(\varphi_i^z) & 0 \\ 0 & \sin(\varphi_i^z) & \cos(\varphi_i^z) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.4}
$$

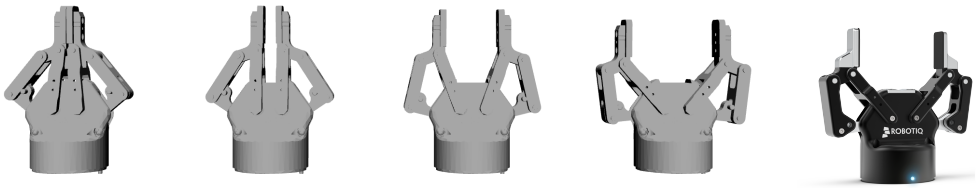
$$
D_m = {}^0 D_1 *^1 D_2 * \cdots *^{m-1} D_m \tag{2.5}
$$

Mivel a robotok mozgását szeretnénk tervezni, a továbbiakban a mozgások definiálása és a vizsgálat módszerei kerülnek kifejtésre, felhasználva a fentieket. Egy mozgás több módon definiálható, (1) két konfigurációtérbeli pont ($c_a, c_b \in C$) vagy (2) két munkatérbeli pont megadásával ($p_a, p_b \in \mathbb{R}^3 \times \mathbb{R}^3$). Ettől függetlenül az így definiált mozdulat értelmezhető további két módon, (A) csuklótérben lineáris illetve, (B) munkatérben lineáris mozgásként. A munkatérben lineáris mozdulat (B) megvalósítására leginkább olyan esetekben van szükség, mikor a végberendezés helyzete az út során folyamatosan számít, például ív- és lézerhegesztés vagy amikor egy felületet folyamatosan kell érinteni. Ehhez a mozdulat megalkotása során az *inverz kinematikai* megoldások közül a lehető legfolytonosabbat kell kiválasztani megoldásként minden pontban. Tehát az eszköztérben való lineáris mozgás nehezen kivitelezhető és az esetek kis részében szükséges. Így az alapvető mozdulat értelmezése csuklótérben lineáris mozgás (A) ami két pont közt egyértelműen definiált. A végpontok megadása esetén a (1) térbeli pontok két alkalmas inverz megol-

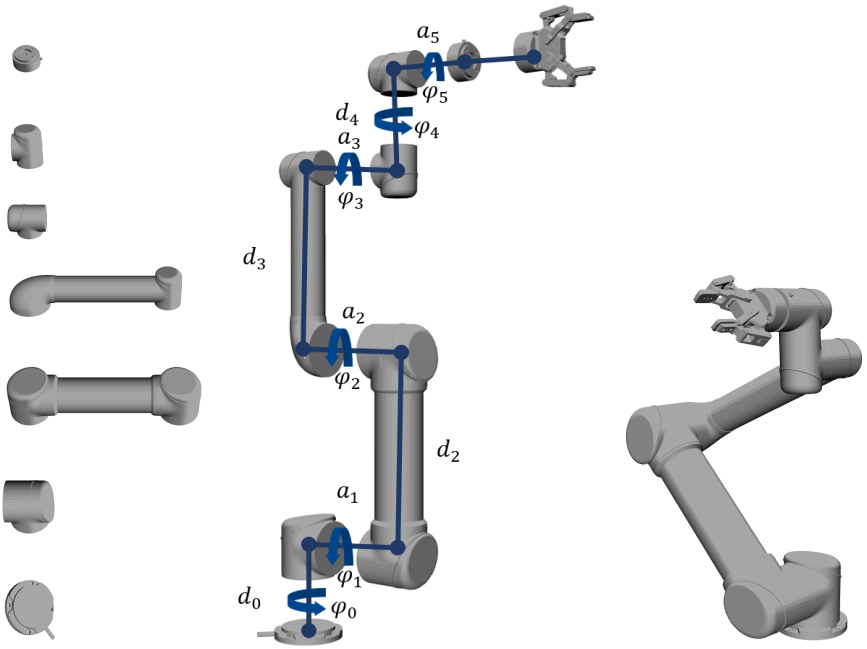
dással átválthatóak konfiguráció térbeli (2) definícióra. Így az alapértelmezett mozgás a két konfigurációval megadott csuklótérben értelmezett egyenes (2A).

2.4. Megfogók modellezése

A megfogók terén már említésre került változatosságuk. Sok közülük geometriáját modellezve állandónak tekinthető, például a ragasztót adagoló vagy a lézerhegesztő végberendezések. Ezzel ellentétben a mechanikus megfogók saját kinematikával rendelkeznek, ami a robothoz hasonlóan szintén implementálandó. A kis alkatrészek külön mozgatása, transzformációinak előállítása és kezelése jelen alkalmazásban elhanyagolható nyereséget hozna, így a következő egyszerűsítést tesszük. Az említett megfogóknak egyetlen paraméterük van, ami befolyásolja aktuális térbeli kiterjedésüket, ez pedig a befogó pofák távolsága. Ennek az értéknek a változtatásával több állapotában készítünk statikus geometriákat (2.3 ábra) és ahelyett, hogy a megfogó pontos kinematikájának megfelelően transzformálnánk, ezeket a statikus geometriákat cserélgetjük. A pofatávolságot tartományokra osztva egy-egy ilyen statikus háromszög hálót rendelünk hozzájuk, így tetszőleges apró változásokat modellezhetünk. A megfogó transzformációját a robottól kapja miután a teljes kinematikai lánc kiszámításra került.



2.3. ábra. RobotiQ megfogó geometriái különböző pofa távolságokkal



2.4. ábra. UR5 kollaboratív robot geometriája és kinematikai lánca

3. fejezet

Ütközésvizsgáló algoritmusok

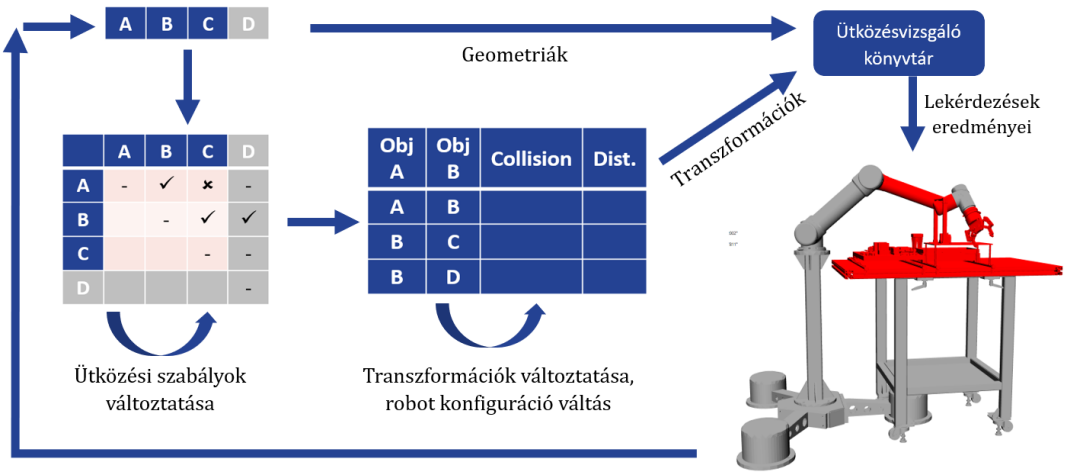
3.1. Statikus konfiguráció ütközésvizsgálata

A gyártócella teljes modellje az eddigiek alapján összeállítható egy tetszőleges c konfigurációban a robottal, megfogójával. Minden olyan geometria, ami különállónak tekinthető ütközésvizsgálati szempontból, mint a robot alkatrészei, végberendezés, munkakörnyezet és a mozgatott vagy megmunkált részegységek egy-egy ütközési objektumba kerülnek becsomagolásra az implementáció során. Ezek az objektumok többek közt rendelkeznek egyedi azonosítóval, geometriával és transzformációval. Ezen objektumokból összeállított párokon futtathatóak egy általános ütközésvizsgáló könyvtár (Proximity Query Package (PQP) [9] / Flexible Collision Library (FCL) [14]) metódusai, amelyben hatékonyan van leimplementálva transzformált geometriák távolságmérése, ütközésvizsgálata. Ütközésvizsgálat esetén egyetlen igaz/hamis értéket várunk, hogy a két objektum ütközik vagy sem. Távolságmérés esetén pedig a pár két geometriájának legközelebbi pontjai közt mért távolságot kaphatjuk meg.

Az általános ütközésvizsgáló könyvtárhoz lekérdéseket intézhetünk, ami két objektumot tartalmaz, eredményként pedig az említett visszatérési értékek valamelyikét. A teljes celláról szeretnénk eldönteni, hogy tartalmaz-e ütközést, ehhez azonban szükség van egy szabályrendszerre, ami definiálja milyen lekérdezésekre van szükség, mely párokat kell vizsgálnunk és melyeket nem. Például az egymást követő robot tagok közt, a megfogó és megfogott tárgyak vagy a robot talpazata és konzolja közt nincs szükség ütközésvizsgálatra, mivel ezek mindenképpen érintkeznek. Ezen a ponton a megépített cellában, a szabályok megadása után képesek vagyunk egy c robot konfigurációban a lekédezések futtatásával megállapítani tartalmaz-e ütközést (3.1 ábra).

3.2. Robotmozgások ütközésvizsgálata

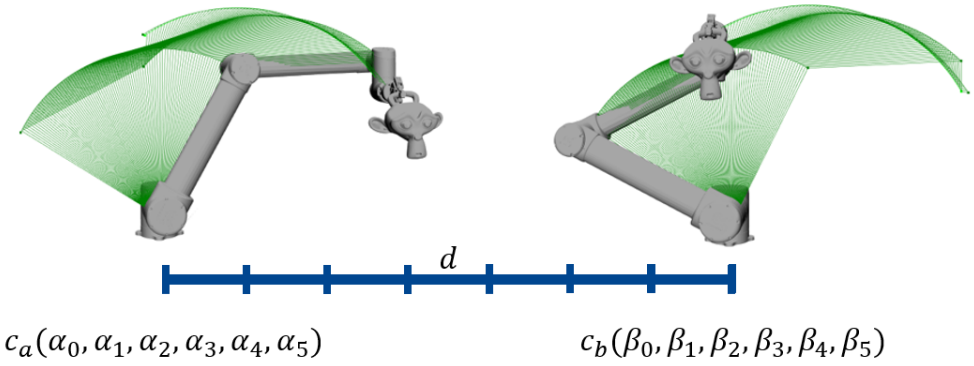
Egy mozdulat alapvető értelmezése c_a és c_b közti csuklótérben lineáris egyenes. Mozgások ütközésvizsgálata során egy fontos szempont a *biztonsági távolság* megléte. Gyakran nem elegendő az ütközésmentesség a geometriák között, hiszen ez magában foglalja azt, hogy a vizsgált testek éppen nem érintik egymást. Azonban a valós alkalmazások során problémát jelenthet a kalibaráció, vagyis a robot és a cellában lévő elemek relatív pozíciójának pontatlansága. Valamint a robot gyártási, modellezési pontossága változó és a tagokra rögzített vezetékek is nehezen vehetők számításba.



3.1. ábra. Statikus konfiguráció ütközésvizsgálata

3.3. Mintavételezés

A statikus ütközésvizsgálat kiterjesztése mozdulatokra egy kézenfekvő megoldás. A bevezetésben említett eszközök a két konfiguráció közti utat egy megadott d mintavételezési frekvenciával minden konfigurációban megvizsgálják (3.2 ábra). Ez a módszer gyorsan implementálható, de hatékonysága nagyban függ a megválasztott maximális szögkülönbségtől (mintavételezési frekvencia), amellyel az adott szakaszt felbontjuk. Túl nagy konstans választása esetén a konfigurációk között van esély az ütközés detektálásának elmulasztására, ellenkező esetben a túl sok ütközésvizsgálat miatt nagyban megnő a szakasz ellenőrzésének ideje.



3.2. ábra. Mintavételezett mozgás vizsgálata

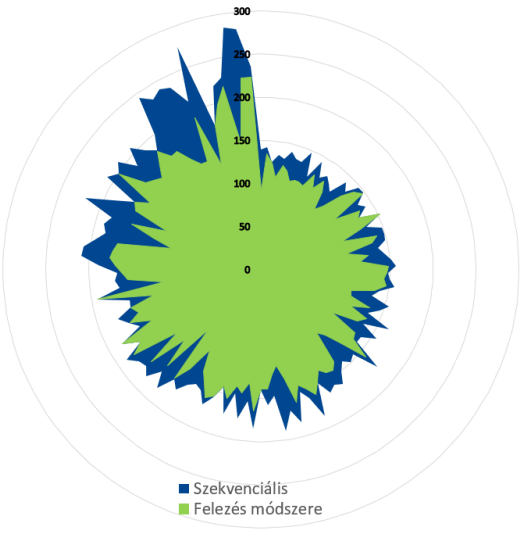
Az általános ütközésvizsgáló könyvtárak által biztosított lekérdezések esetén lehetőségünk van egyszerű bináris értéket kérni egy objektumpár ütközéséről vagy azok pontos távolságát meghatározni. Utóbbi egy nagyságrenddel több időt vesz igénybe. Mintavételezés esetén csak úgy lenne ellenőrizhető a biztonsági távolság betartása, ha minden tesztelt konfigurációban a költségesebb távolság alapú lekérdezést használjuk. A teljes szakaszon vett minimális távolság ekkor tekinthető biztonsági távolságnak (amennyiben eltekintünk a mintavételezések közti szakaszon mért távolságtól), de a felhasznált távolság lekérdezés nagy költségtöbbletet jelent az egyszerű bináris visszatérési értékű módszerrel szemben, így az implementáció során az utóbbi módon került megvalósításra.

Mozdulatok ütközésvizsgálata során a legköltségesebb művelet a lekérdezés, ezért szeretnénk egy mozdulat ellenőrzésekor a lehető legkevesebbszer végrehajtani. A mintavéte-

lezés során a kezdő és végpont ($c_a, c_b \in C$) által definiált egyenest egy megadott d frekvenciával vizsgáljuk n konfigurációkban $c_1, ..., c_i, ..., c_n \in \overline{c_a c_b}$ és futtatjuk az ütközési szabályokból leképzett k lekérdezést $(q_0, ..., q_k)$. A mintavételezési frekvencia és a cellában lévő elemek száma meghatározza n és k értékét, mely így az alkalmazástól függ. Amennyiben a vizsgált szakasz ütközésmentesnek bizonyul, a vizsgálat során mind az $n * k$ vizsgálatra szükség van. Egy pálya generálása során azonban sok ütköző mozdulat is megvizsgálásra kerül, így érdemes az ütközést a lehető legkorábban megállapítani, hogy fölösleges lekérdezéseket ne indítsunk el. Ezt a konfigurációk vizsgálatának és a lekérdezések sorrendjének változtatásával érhetjük el.

A konfigurációk alapértelmezett vizsgálata szekvenciálisan történik $(c_1, c_2, ..., c_n)$. Ebben az esetben az egymás után vizsgált konfigurációk nagyon közeliek, tehát ha c_i konfiguráció ütközésmentes, akkor kisebb valószínűséggel lesz a következő $(c_{i+1}.)$ konfigurációban ütközés, mint egy jóval távolabbi pontban. Így érdemes minden konfiguráció vizsgálata után egy olyan ponton folytatni, ami a korábbiaktól lehetőleg messzebb van, erre használható a szakaszfelezés módszere $(c_1, c_{[n]}, c_{[\frac{n}{2}]}, c_{[\frac{n}{4}]}, c_{[\frac{3}{4}n]}, ...)$. A 3.3 ábra bemutatja 5000 véletlenszerű mozdulat vizsgálatához tartozó lekérdezésszámot egy egyszerű szerelőcellában. Minden pont origótól való távolsága 40 mozdulat átlagos lekérdezésszámát mutatja, első esetben szekvenciális sorrendben, majd a szakaszfelezés módszerével vizsgálva. Látható, hogy az esetek nagy részében utóbbi módszerrel hamarabb értesülünk az ütközésről, így jobb döntés először nagyobb lépésekben bejárni a szakaszt és legrosszabb esetben is a szekvenciális végrehajtással megegyező lépésszámot kapunk, mivel ütközésmentes szakaszok esetén mindenképpen n konfigurációt kell megvizsgálnunk.

A minden konfigurációban futtatott k páros lekérdezése esetén is kihasználhatjuk azt a heurisztikát, hogy a gyakran ütköző elemeket a lekérdezési lista elejére soroljuk. Erre alkalmas egy egyszerű számláló az ütközési objektumban, ami minden esetben növelésre kerül, ha ütközésben bármikor szerepe van. Ennek megfelelően időnként rendezve a párokat (párban szereplő objektumok számlálójának összege szerint csökkenő sorrendbe) növelhető a hatékonyság. A fentebb említett 5000 mozdulat során rögzítésre került a cellában lévő geometriák ütközéseinek száma amit a 3.1 táblázat mutat be. Látható, hogy a különböző objektumok nem azonos valószínűséggel fognak ütközni, ez az egyszerű statisztika nagyban függ a cella geometriájától és a korábban vizsgált konfigurációk elhelyezkedésétől, azonban sorrendezéséhez mindenképpen jó heurisztikát szolgáltat.



3.3. ábra. Ütközésvizsgálat lekédezéseinek száma

Cella elem	Ütközések száma
Link1	234
Link4	307
Link6	3 336
Link0	3 540
RobotiQ	8 471
Link5	12 585
Link2	20 536
Workcell	20 889
Link3	25 226

3.1. táblázat. Cellaelemek ütközéseinek száma 5000 szakasz vizsgálata után

3.4. Conservative Advancement

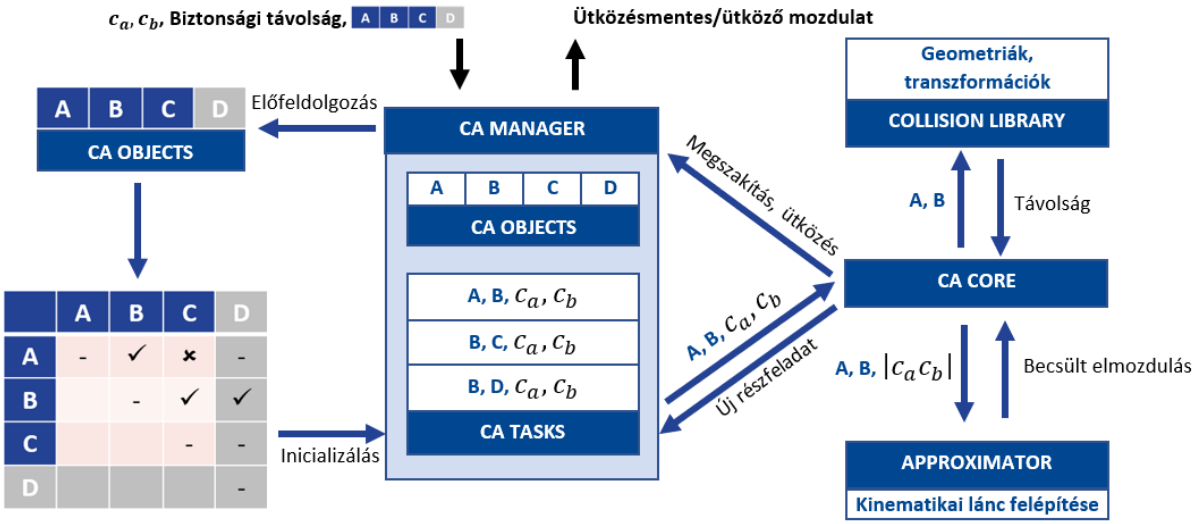
Ahogy a mintavételezést használó algoritmusok általában, a mintavételezési-frekvencia megválasztása a legnehezebb feladat, valamint ennek következményeként az ütközésmentes mozdulat és biztonsági távolság nem garantálható. Így egy olyan folytonos eljárásra lenne szükség, amely ezeket a garanciákat megadja és futásidőben is képes a mintavételezéses módszert elérni, megközelíteni. Erre a célra az ún. *Conservative Advancement (CA)* eljárások alkalmasak.

A jelenlegi implementáció és áttekintés irodalmi alapjának jelentős részét a [17] cikk adta, ami bemutatja a teljes ütközésvizsgálati folyamatot. Az eljárás alapötlete, hogy a fent említett problémák miatt, mintavételezés helyett, az objektumok távolságát és a robot kinematikai láncának sajátosságait igyekszik kihasználni annak meghatározására, hogy mekkora mozdulatot vagyunk képesek megtenni biztonságosan az ütközésmentes térben. Így ebben az esetben a költségesebb, távolságmérésen alapuló lekérdezést használjuk és azt reméljük, hogy a nagy méretű ütközésmentes térben elegendő ritkán alkalmaznunk, míg akadályokkal teli szűk helyeken az átlagosnál többet.

A kulcs gondolat, hogy egy adott konfigurációban lekérdezzük az objektumpárok távolságait és ezeket felhasználásával egy olyan következő konfigurációt próbálunk keresni a mozdulat egyenesén, amely a lehető legtávolabb van az előző ponttól, de a mért távolságot még nem tette meg. Ilyen módon képesek vagyunk vizsgálat nélkül a két konfiguráció közi szakaszt ütközéstől mentesnek nyilvánítani, formális garanciát vállalva az ütközésmentességre (ellenben a mintavételezéssel), valamint biztonsági távolság megtartására is ígéretet tudunk tenni, ami a valós alkalmazásokra való tervezés során elengedhetetlennek bizonyult. Az áthidalandó probléma, hogy az ellenőrizendő szakasz csuklótérben lineáris egyenes, amelyen a távolságot szögben értelmezzük, a lekérdezések viszont munkatérben mért távolságot szolgáltatnak. Így nem tudjuk, mekkora (szögben mért) lépést tehetünk a csuklótérben értelmezett egyenesen anélkül, hogy a robot egyetlen része is átlépné a maximális munkatérben mért távolságot. Erre egyszerű összefüggés nem adható, mivel kis szögelfordulások is eredményezhetnek összességében nagy elmozdulást a robot különböző pontjain és fordítva. Viszont elegendő egy felső becslés, amely adott konfigurációváltozás esetén a robot kinematikai láncának és a tagjainak fizikai méretéből adódóan, meghatározza a munkatérben megtett maximális távolságot.

3.4.1. Általános működés

Mindezek keretbe foglalásához a statikus vizsgálat során használt egymás után futtatott lekérdezés halmazt szét kell bontanunk és más környezetbe helyezni. Az alapvetően használt ütközési objektumokon előfeldolgozást kell végeznünk és ezek eredményét, egy Conservative Advancement Objektumokba (CA Object) csomagolni. Ez a cellamodell építésekor a lekérdezéstől függetlenül előre elvégezhető. Ezt követen érkezik a kérés a mozdulat ellenőrzésére, ami tartalmazza a szakasz leírását ($c_a, c_b \in C$) valamint a kért biztonsági távolságot (ϵ). Ezek után a statikus és mintavételezéses vizsgálathoz hasonlóan a vizsgálandó párokat, az ütközési szabályok alapján a vizsgált szakasz végpontjaival együtt feladatok formájában (CA Task) egy sorban tároljuk. Innen vehet feladatot az algoritmus ütközésvizsgáló magja (CA Core), ami a két objektumon illetve az adott szakaszon a később bemutatott módon végzi el az ütközésvizsgálatot a becslő és ütközésvizsgáló motor felhasználással. Ezek eredménye (1) ütközés, (2) részleges ütközésmentesség vagy (3) teljes ütközésmentesség lehet. Ütközés esetén (1) a további futás megszakad, következő esetben (2) a fennmaradó biztosítatlan szakasz a vizsgált objektumokkal visszakerül a tárolóba új feladat formájában és végül (3) sikerül a szakaszt ütközésmentesnek jelölni ekkor nincs további teendő, kivehető a következő feladat. A szakasz akkor tekinthető ütközésmentesnek, ha a feladattároló kiürül és nem történt megszakítás.



3.4. ábra. Conservative Advancement folyamatábrája

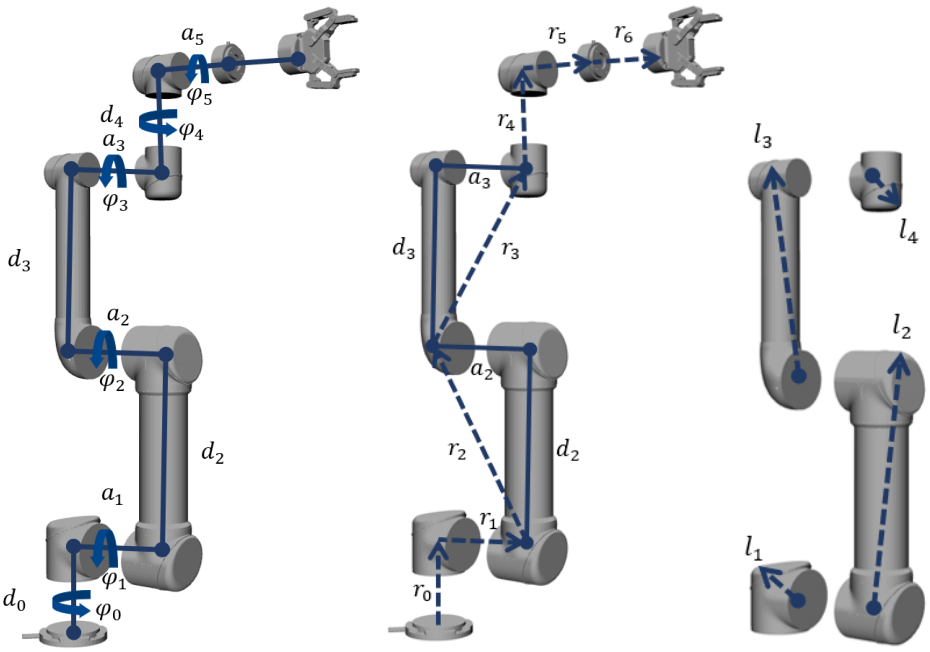
3.4.2. Előfeldolgozás, becslés

A korábban létrehozott ütközési objektumok kiegészítése szükséges az előfeldolgozás során kapott információkkal, ezekkel kerül újracsomagolásra CA Object formájában. A cella és robot alkatrészeinél egyaránt szükségünk van a geometrák méreteire a becslés elvégzéséhez. Ezek a következő paraméterek (3.5. ábra):

- d_i, a_i - Korábban említett Denavit-Hartemberg paraméterek.
- l_i - Az i. tag alsó csuklópontjának és legtávolabbi pontjának távolsága. A csúcsok bejárásával számítható.
- r_i - Az i. és $i+1$. tag alsó forgáspontjának távolsága. A d_i, a_i pontjai által leírt háromszög átfogójának hossza.
- φ_i - Vizsgált mozdulat ($c_a, c_b \in C$) i. csuklószögeinek abszolút különbsége radiánban.
- ϵ - Biztonsági távolság.

A paraméterek a csuklószögek kivételével előre kiszámíthatóak, a CA objektumokhoz ezentúl hozzátartoznak. Így a bemeneten érkező két konfiguráció tudatában a teljes kinematikai lánc maximális elmozdulására adhatunk felső becslést az alábbi (3.1) összefüggést felhasználva. Ez tartalmazza a kinematikai lánc tagjainak és a csuklópontok közti eltolásoknak a hosszát összegeként és ezek adott szöggel való elforgatása során húzott ívhosszt. A csuklók elfordulása minden esetben tartalmazza az előző csuklók elfordulását, mivel egy alsó tag mozgatása hatással van az összes utána következőre.

$$\delta = \sum_{x=0}^{m}((r_x + l_x)\sum_{y=0}^{x}(\varphi_y)) + \epsilon \tag{3.1}$$



3.5. ábra. Conservative Advancement paraméterek

3.4.3. Ütközésvizsgálat

Az ellenőrzés során az ütközésvizsgálatok lényegi részét a már említett CA Core végzi el, a tárolóból vett CA Task-ban foglaltak szerint. A feladat tehát tartalmazza a két objektumot (a szükséges paraméterekkel), amelyek ütközésvizsgálatát el kell végezni és a vizsgált szakaszt leíró két pontot. A művelet során továbbá szükséges a becslések elvégzése (Approximator) és a korábban már bemutatott általános célú ütközésvizsgáló könyvtárak lekérdezése. Adott konfigurációban c_i két objektum (i, j) távolságának lekérdezését $dist_{i,j}(c_i)$ jelöli.

```
PROCEDURE CATaskSolver(CATask(c_a, c_b, CAObject i, CAobject j), ε)
1   dist_a :=dist_{i,j}(c_a)
2   dist_b :=dist_{i,j}(c_b)
3   IF (dist_a ≤ ε OR dist_b ≤ ε)
4     RETURN collision
5   c_a^+ := c_a + FindMaxStep(c_a, c_b, dist_a)
6   c_b^- := c_b + FindMaxStep(c_b, c_a, dist_b)
7   IF |c_a − c_b^-| <= |c_a − c_a^+|
8     RETURN collision-free
9   ELSE
10    RETURN new CATask(c_a^+, c_b^- , CAObject i, CAobject j)
```

3.6. ábra. Algoritmus CA Task megoldására

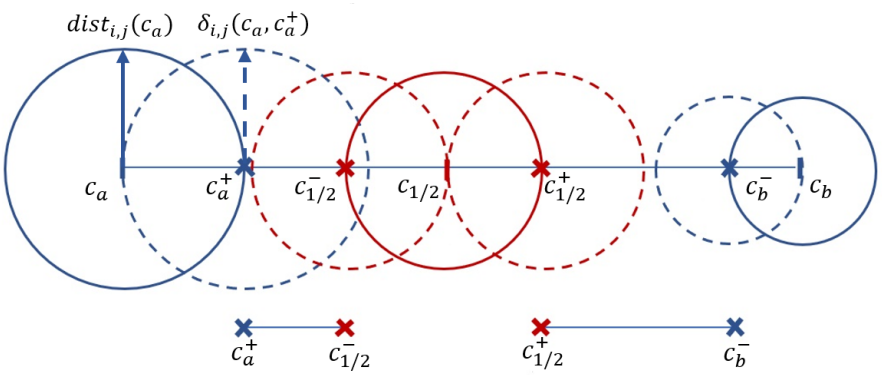
```
PROCEDURE FindMaxStep(FROM c_a , TO c_b , FROM_DISTANCE MaxDistance)
1   SafeStepConfiguration := [0, ..., 0]
2   ApproxDistance := 0
3   WHILE (ApproxDistance < MaxDistance)
4     ApproxDistance := δ_{i,j}(SafeStepConfiguration)
5     SafeStepConfiguration := Move on c_a → c_b linear line
6   RETURN SafeStepConfiguration
```

3.7. ábra. Maximális ütközésmentes lépés meghatározása

Az első (3.6) algoritmus tehát a két végpontban megméri az objektumok távolságát $(dist_a, dist_b)$ amennyiben ezek az elvárt biztonságitávolságot nem teljesítik vagy ütköznek a mozdulatunk hibásnak bizonyult, nincs több teendőnk. Ellenkező esetben keresnünk kell a két konfigurációból (c_a, c_b) egy maximáslis léptéket egymáshoz közeledve (c_a^+, c_b^-). Ezt a második (3.7) eljárás képes meghatározni a kiinduló konfiguráció, az irányt meghatározó másik pont, valamint a kiinduláskor mért távolság ismeretében. Itt a becslés felhasználásával meghatározásra kerül a maximális szögben mért lépték ami biztosan ütközésmentes, az átadott távolságnak megfelelően. Ezt követően ismerjük c_a^+, c_b^- konfigurációkat és amennyiben a két pont az ellenőrzés során átlépett egymáson a szakasz ütközésmentes, tehát a feladat megoldott, ellenkező esetben új feladatot kell készítenünk a fennmaradó c_a^+, c_b^- szakaszból és a két vizsgált objektumból.

A fenti algoritmusok (3.7, 3.7) használatával tehát megoldhatunk egy feladatot aminek kimenete ütköző, ütközésmentes szakasz, illetve újabb feladat lehet. A pszeudo kódokban szereplő algoritmusoknak megfelelően 3.9 ábrán a kékkel jelölt pontok kerültek

megvizsgálásra. Az ábrán szereplő megoldás egy köztes felező pontot (piros) is használ és a megadott algoritmusokban leírt műveleteket alkalmazza. Így az egy köztes távolság lekérdezés eredményét kétszer tudjuk hasznosítani, valamint a mintavételezés optimalizálásánál bemutatottak alapján a felezéseknek köszönhetően jobb esélyekkel találunk egymástól távoli konfigurációkban ütközést. A CA eljárás esetén is van lehetőség a sorrendezéssel optimalizálni. A CA Taskok listája folyamatosan változik be/ki kerülnek a feladatok, a mintavételezés során használtak alapján itt is érdemes a taskok listáját olyan szempontok alapján rendezetten tartani, hogy az első feladat mindig a legnagyobb valószínűséggel ütköző legyen. Ehhez felhasználható a már összegyűjtött statisztika az objektumokról és a szakasz hossza amelyet a feladat tartalmaz. A sorrend fenntartása az így megspórolt ütközésvizsgálati lekérdezésekhez képest elhanyagolható.



3.8. ábra. Mozdulat ellenőrzése CA eljárással

3.5. Fejlesztett Conservative Advancement

A bemutatott CA algoritmus már tartalmaz a megvalósításhoz szükséges részleteket az eredeti cikkek [17] tartalmához képest. Viszont a futásidő és valós alkalmazásokban felhasználhatósághoz a további kiegészítések szükségesek. Ezek közé tartozik egy továbbfejlesztett becslő eljárás [21], amellyel pontosabb felső becslés adható, ezzel jobban kihasználva az ütközésmentes tereket, kevesebb távolság lekérdezésével gyorsabban ellenőrizhetjük a mozdulatokat, valamint az eljárás megfogott geometriákra való kiterjesztése.

3.5.1. Továbbfejlesztett becslő eljárás

A már bemutatott eljárás során a feladatokban (CA Task) az objektumok párokban szerepelnek, azonban a becslő eljárás minden esetben a teljes kinematikai lánc elmozdulását használja és nem veszi figyelembe, hogy az objektumok a cellákban milyen szerepet töltenek be, a láncban hol szerepelnek. Ennek érdekében az összes objektum (CA Obejct) kap egy számot, ami a kinematikai láncban elfoglalt helyét jelöli. A statikus objektumok -1, a robot tagjai a talptól felfelé 0-tól $m-1$-ig, a megfogó m valamint a megfogott objektumok mind $m+1$ értéket kapnak. Így a két objektum vizsgálatakor elégséges az elhelyezkedésüket leíró rétegek közti kinematikai lánc szegmensre alkalmazni a becslést. Ez a kiegészítés (3.3 egyenlet) magában foglalja, hogy két *statikus ütközési objektum* mint cellaelem vagy munkadarab esetén nincs elmozdulás (mivel -1.-től -1. rétegig becsülünk), két *dinamikus ütközési objektum* esetén a két réteg közti lánc szakasz elmozdulása, és statikus-dinamikus pár esetén pedig lánc elejétől a dinamikus elem rétegéig kerül figyelembevételre. Arról, hogy a becslésünk valójában mennyivel lett ügyesebb, nehezen készíthető önmagában értékelés,

erre a kérdésre a futásidőket megjelenítő 6.1 táblázat ad választ a *Kísérleti eredmények* fejezetben.

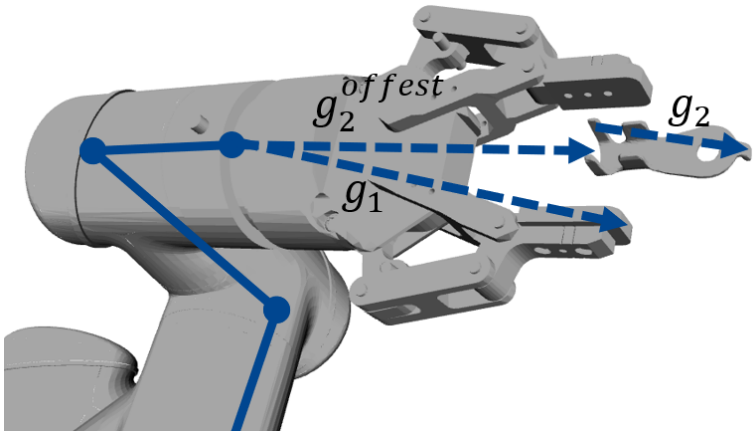
$$\delta_{i,j} = \sum_{x=layer_i}^{layer_j}((r_x + l_x)\sum_{y=layer_i}^{x}(\varphi_y)) + \epsilon \tag{3.2}$$

3.5.2. Megfogott objektumok kezelése

A robotok munkavégzése során a legtöbb esetben szükségünk van a cellában lévő elemekkel történő interakcióra. Ezt a mintavételezésnél külön nem kellett kezelni mivel, a mozgatott munkadarabok a konfigurációváltással együtt változtatták helyzetüket és a mintavétel idején adott pozícióban kerültek megvizsgálásra. Ez a CA esetben nincs így, mivel a becslés során nincsen kezelve a megfogott tárgy geometriája miatt meghosszabbodott kinematikai lánc. A megfogás után statikus pozíció helyett a lánc végével összhangban dinamikusan mozog a konfigurációk váltásával. Emellett az egyszerű objektumok nem rendelkeznek a becslésben használt Denavit-Hartenberg paraméterekkel, csuklópontokkal.

Így az előfeldolgozás során minden nem robot alkatrész, munkadarab maga a megfogó és egyéb cellaelemek esetén szükségünk van egy új paraméterekre ami, a g_i - geometria befoglaló téglatestének testátlója és g_i^{offset} ami egy lekérdezett távolság érték a kinematikai lánc vége és a megfogott objektum legközelebbi pontja között. Így több megfogott elem esetén is rendelkezésre áll a végponttól való távolság és a test átmérőjének mérete, ezek közül azt kell kiválasztanunk amely elem esetén ezek összege a legnagyobb, $max(g_i + g_i^{offset})$.

$$\delta^*_{i,m+1} = \delta_{i,m-1} + \sum_{y=i}^{m}(\varphi_y) max(g_i + g_i^{offset}) \tag{3.3}$$



3.9. ábra. CA eljárás kiterjesztésének paraméterei

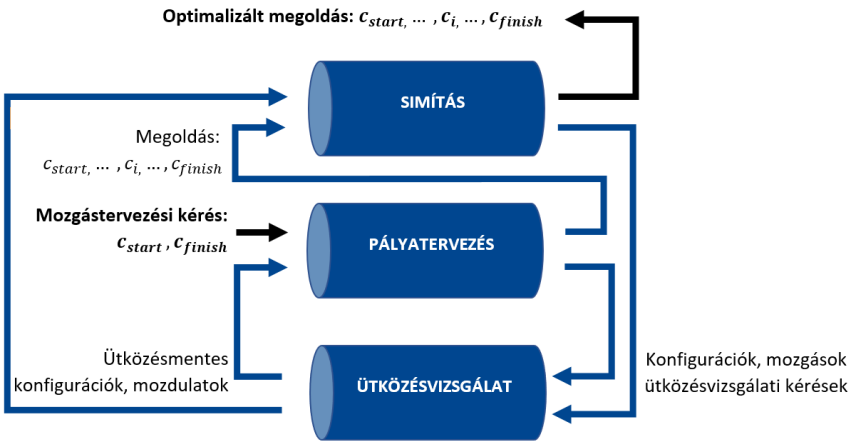
4. fejezet

Pályatervező algoritmusok

Az előző fejezetek szerint létrejött a teljes cellamodell, lehetőség van statikus konfiguráció és mozdulat ütközésvizsgálatára, ezek működése testreszabható ütközési szabályok segítségével, valamint a megfogó és a megfogott objektumok is kezelhetőek. Felhasználásukkal rugalmasan le lehet írni a környezetet, amelyben pályát szeretnénk tervezni, valamint a rendelkezésre álló hatékony ütközésvizsgálati metódusok szintén elengedhetetlenek a feladat megoldásához.

Az útvonalat alkotó szakaszok vizsgálatának száma nagyban függ a pályatervező algoritmustól és a használt cellamodellben lévő akadályok sűrűségétől. Az ipari robotok mozgásának tervezésében a nehézséget a magas szabadsági fok adja, jelenleg az UR5/UR10 esetén ez 6 csuklót jelent. A magas dimenzióval rendelkező konfigurációs tér szisztematikus bejárása nem lehetséges, ezért randomizált algoritmusokkal igyekszünk minél hatékonyabban szabad pályát tervezni. A mozgástervezés bemenete jelen esetben két konfiguráció lesz és kimenetként egy konfiguráció listát várunk el, amely pontjainak bejárásával ütközésmentesen vihető át a robot kiindulási állapotából a céljába. Jelen esetben is a *Kinematikai modellezés*nél mozdulatként definiált konfigurációs térben lineáris mozdulatok felhasználásával, tehát csuklótérben tervezett pályák elkészítése a kitűzött cél. Ebben az esetben is megvan a lehetőségünk a kiinduló- és végpontként munkatérben leírt pozíciók konverzióját két inverz kinematikai feladat megoldásával elvégezni, így a munkatérben kijelölt pontok közti pálya is tervezhető csuklótérben. Továbbiakban két pályatervező algoritmus a *Rapidly-exploring Random Trees* (RRT) [10] és *Probabilistic Roadmaps* (PRM) [7] áttekintése szerepel és azok kiegészítései, felkészítve őket valós ipari problémák megoldására.

Az RRT felhasználási területe leginkább a magas dimenziószám miatt rács alapú módszerrel nem felderíthető, előfeldolgozást nem igénylő, kevés pályát használó feladatok megoldása, vagy ahol előre nem definiálható módon változó környezetben szeretnének tervezni, különös tekintettek a kollaboratív robotok környezetére, emberi beavatkozásra. Azonban sok feladat esetén a gyártócellában ilyen változásokra nem kell számítanunk, a cella állapotai jól definiáltak, az esetleges környező gépek kinematikája számítható és emberi beavatkozás is gyakran a cella teljes áramtalanításához kötött, illetve sok pályát szeretnénk készíteni. Ebben az esetben az RRT futása során készített gráfhoz hasonló offline térkép igen hasznos lenne előfeldolgozásként, így amikor tényleges pályatervezésre kerül a sor az előre felderített biztonságos mozdulatokat felhasználhatunk. Ezt az igény szolgálja ki a PRM tervező eljárása.

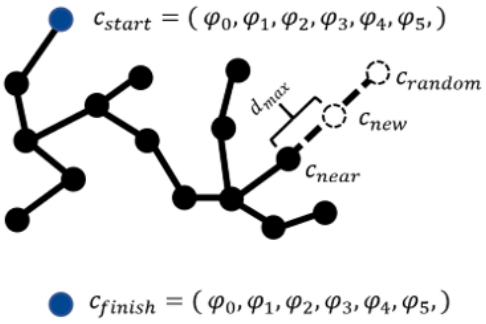


4.1. ábra. Pályatervezés folyamatábrája

4.1. Rapidly-exploring Random Trees (RRT)

Az algoritmus leírásának sok variánsa született az évek során, ezek közül jelenleg az eredeti [10] cikkeben publikált legáltalánosabb formája kerül felhasználásra. Az RRT egy randomizált single-query algoritmus, amely futásához előzetes tervezés nem szükséges, ennek megfelelően olyan esetekben használt, amikor a tér felfedezése és változásainak definiálása problémát jelent. Általánosan autonóm robotok, drótok pályájának mozgásának tervezésére alkalmazzák.

Az eljárás bemenete a két konfiguráció melyek közt utat szeretnénk találni, ezzel elvégezve a csomagolási, anyagmozgatási, egyéb feladatunkat. Amennyiben a két pont nem köthető össze ütközésmentesen egyetlen lineáris mozdulattal, egy olyan összetett mozdulatot szeretnénk keresni konfigurációk listájaként, amely mentes minden ilyen problémától. Tehát a kezdő és végső ütközésmentes konfiguráció ($c_{start}, c_{finish} \in C_{free}$) valamint a keresés során épített fa csomópontjainak maximális távolsága (d_{max}) és a már bemutatott ütközésvizsgálati képességek elegendőek az algoritmus futtatásához.



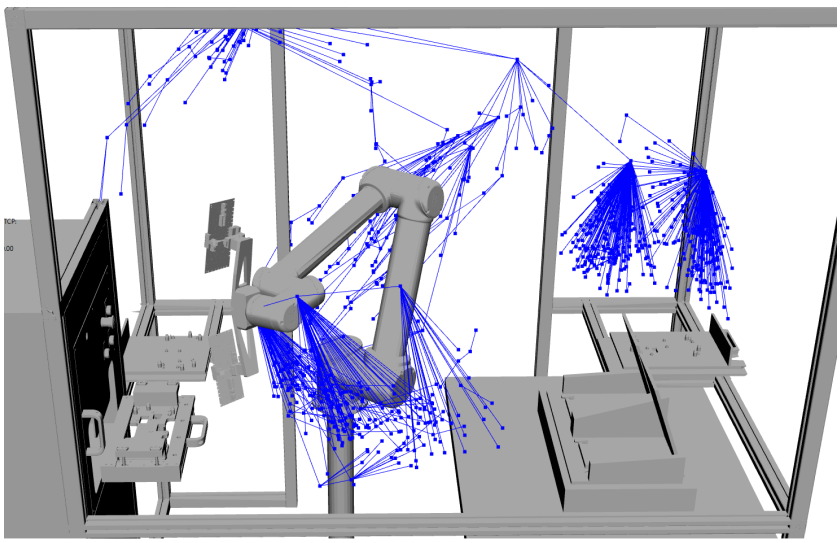
4.2. ábra. RRT gráf építése

A fa gyökér csomópontja a c_{start} konfiguráció és az eljárás akkor ér véget, ha a fa tartalmazza a befejező c_{finish} konfigurációt. Ehhez a következő lépéseket ismételjük (4.3 algoritmus, 4.2 ábra). Generálunk egy véletlenszerű konfigurációt (c_{rand}) a robot csuklótartományain belül, majd a már meglévő fában megkeressük a hozzá legközelebb álló csomópontot (c_{near}). Ez különböző mérési módszerekkel történhet, mint a szakasz megtételének ideje, a konfigurációban szereplő szögek különbségének valamilyen súlyozott összege vagy átlagos eltérése. Majd megvizsgáljuk, hogy az előbbi módszerek valamelyikének megfelelően megadott d_{max} távolság határain belül esik-e a generált pontunk. Amennyiben

ezt a feltételt nem teljesíti, a két pont közti lineáris egyenesen közelebb kell vinnünk az új konfigurációnkat. Így előállt egy maximális távolságon belüli új csomópont (c_{new}), ezzel egy lehetséges mozdulat amelyet felvehetünk a fába amennyiben az ütközéstől mentes. Minden új csomópont esetén érdemes megvizsgálnunk hogy nem kerültünk-e a befejező csomóponthoz elég közel. Amennyiben igen, a pálya utolsó szegmenseként felvesszük a fába az ütközésvizsgálatot követően. Ezeknek a lépéseknek az ismétlésével elérjük, hogy a fába levélként bekerül az utolsó csomópont is és a gyökér felé haladva kiolvasható a megoldásunk. A 4.4 ábra egy ilyen tervezési folyamat végére előállított gráfot mutat be. Az RRT algoritmusok kutatása és vizsgálata ma is aktív terület, léteznek két irányú variánsaik [20], valamint megközelíthető célorientált mintavételezést [6] használó eljárások amelyek segítségével a nehezen bejárható célok is az eddigieknél gyorsabban megtalálhatóak.

```
PROCEDURE RRT(c_start, c_finish, d_max)
1   Graph.AddVertex(c_start)
2   WHILE c_finish ∉ Graph
3     c_random := setRandomConfig();
4     c_near := findNearest(c_random, Graph);
5     IF DistanceOf(c_random, c_near) > d_max
6       c_new := moveCloser(c_random, c_near);
7     ELSE
8       c_new := c_random
9     IF IsMotionCollides(c_near, c_new))
10      BREAK;
11    ELSE
12      Graph.AddVertex(c_new)
13      Graph.AddEdge(c_near, c_new)
14      IF DistanceOf(c_new, c_finish) > d_max AND IsMotionCollides(c_new, c_finish)
15        Graph.AddVertex(c_finish)
16        Graph.AddEdge(c_new, c_finish)
```

4.3. ábra. RRT algoritmus



4.4. ábra. RRT pályatervező használata egyedi cellában

4.2. Probabilistic Roadmaps (PRM)

A Probabilistic Roadmaps (PRM) [7] kétlépéses pályatervező algoritmus, amely szintén képes magas dimenziójú terekben is tervezni. *Multi-query* algoritmusként első lépésben egy *ütközésmentes térkép (Roadmap)* készül, ami később újrafelhasználható. Ehhez először csúcsokat kell generálnunk, majd ellenőriznünk a létrehozott konfigurációkat ütközések szempontjából. Ezután egy *lokális tervező (local planner)* valamilyen megfontolás mentén megpróbálja összekötni ezeket ütközésmentes élekkel, mozdulatokkal. Ezt követően második, online lépésként a kialakított gráfban már gyorsan tervezhető a mozgás gráfkereső algoritmusok segítségével.

Tehát első lépésben a térkép építéséhez (4.5 algoritmus, 4.8 ábra) a csomópontok generálására kerül sor. Az ütközésmentes csomópontok közt egy lokális tervező segítségével, egy megadott $edge_{max}$ számú kapcsolatot igyekeznek kialakítani minden csomópont esetén. A PRM változatok nagy százalékában a legközelebbi $edge_{max}$ számú csomópont közti éleket tesztelik ütközésvizsgálati szempontból és amennyiben ütközésmentesnek bizonyul két pont kapcsolata, az így létrejött mozdulat, él bekerül a gráfba. Így kialakításra kerül egy kívánt méretű megfelelő sűrűségű gráf.

```
PROCEDURE RoadmapBuilding(node_max, edge_max, Graph)
1   numberOfNodes := 0
2   WHILE nubmerOfNodes < node_max
3     c_random := setRandomConfig()
4     IF NOT ConfigCollides(c_random)
5       Graph.AddVertex(c_random)
6       numberOfNodes++
7   FOR each v_a ∈ Graph
8     FOR each v_b nearestNeighbour(v_a, edge_max)
9       IF NOT MotionCollides(v_a, v_b)
10        Graph.AddEdge(v_a, v_b)
```

4.5. ábra. PRM Roadmap építése

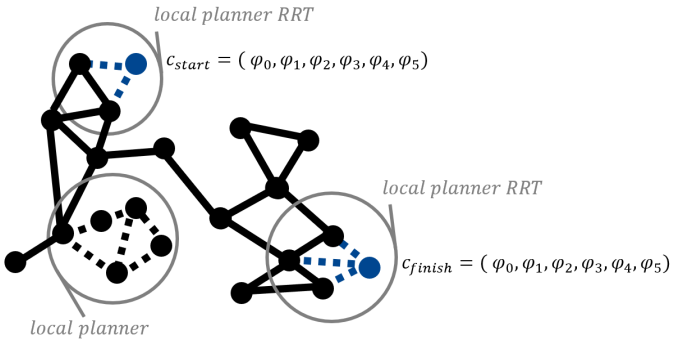
Ezt követően a pályatervezési kérés érkezésekor a kért kezdő- és végpontot (c_{start}, c_{finish}) lehetséges, hogy a gráf nem tartalmazza. Ebben az esetben egy már használt lokális tervező segítséget nyújthat, mivel a legközelebbi csomópontokkal képes kapcsolatot kialakítani. Erre azonban csak abban az esetben képes ha egyetlen lineáris mozdulattal ez megoldható. Lehetséges, hogy a csomópont félreeső, nehezen elérhető helyen van így, egy olyan lokális tervező futtatása lenne célszerű, amely ennél komplexebb kapcsolatokat is ki tud alakítani. Erre alkalmas a már bemutatott RRT algoritmus és ezt felhasználó 4.6 algoritmussal leírt lokális tervező. Így a c_{start}, c_{finish} pontokkal kiegészített térképben alkalmas gráfalgoritmus segítségével kereshetünk utat.

A felépített gráf tulajdonságait tekintve az eredeti cikk [7] keresőfát említ, ebben azonban a létrejött pályák minősége a tapasztalatok szerint gyakran nem kielégítő. Így ehhez képest az általános gráf alkalmazása jobb technikának bizonyult. Ebben az alkalmazott keresőalgoritmus lehet egyszerű *szélességi keresés (BFS)* vagy *Dijkstra-algoritmus* is. Valamint a kezdő- és végpontok esetén említésre került az RRT felhasználása. A *4.1. Rapidly-exploring Random Trees (RRT)* alfejezetben bemutatott verzió két pont között keres utat, viszont a jelenlegi lokális tervező (4.6) több csomóponton keresztül is megkísérelné a pont bekötését, így ez a helyzet igényelne egy olyan RRT implementációt,

amely több levelet is képes számon tartani célként. Valamint felmerül az elzárt, nehezen hozzáférhető pontok bekötésekor a két irányból növesztett fák [20] alkalmazása.

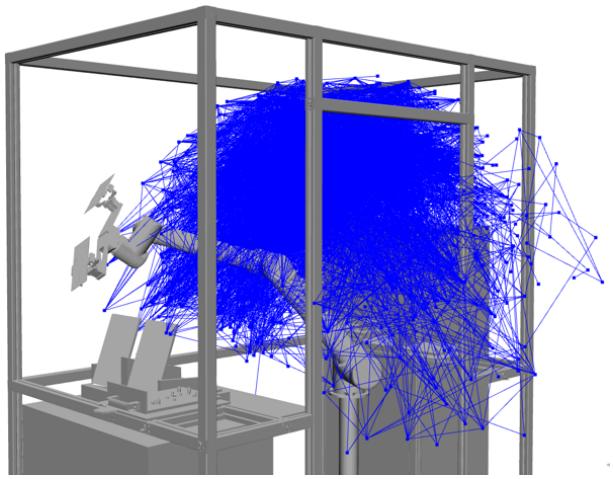
```
PROCEDURE LocalPlannerRRT(Graph, c_target, d_max)
1   FOR each c_near ∈ nearestNeighbour(Graph, edge_max, c_target)
2     IF NOT MotionCollides(c_near, c_target)
3       Graph.AddEdge(c_near, c_target)
4     ELSE
5       TryRRT(c_near, c_target, k_maxiteration)
```

4.6. ábra. Local Planner RRT



4.7. ábra. PRM gráf építése, lokális tervezők felhasználása

A PRM eljárások és azok felhasználási területei, az alapvetőnek számító [7] cikk óta is folyamatos fejlődésben vannak [1]. Például a szűk átjárók és terek kezelése különböző mintavételezési [2] eljárásokkal egy aktív terület. Valamint ezt az is jelzi hogy az algoritmus első verziójának kidolgozása közben, a kutatólaborban általunk is használt RoboDK tervező szoftver is egy ilyen implementációval bővült [13]. A PRM áttekintésének lezárásaként kiemelném, hogy az algoritmus implementációja és a használt eljárások finomítása vendégkollégánk Németh Balázs és konzulensem Kovács András munkáját dicséri [12].

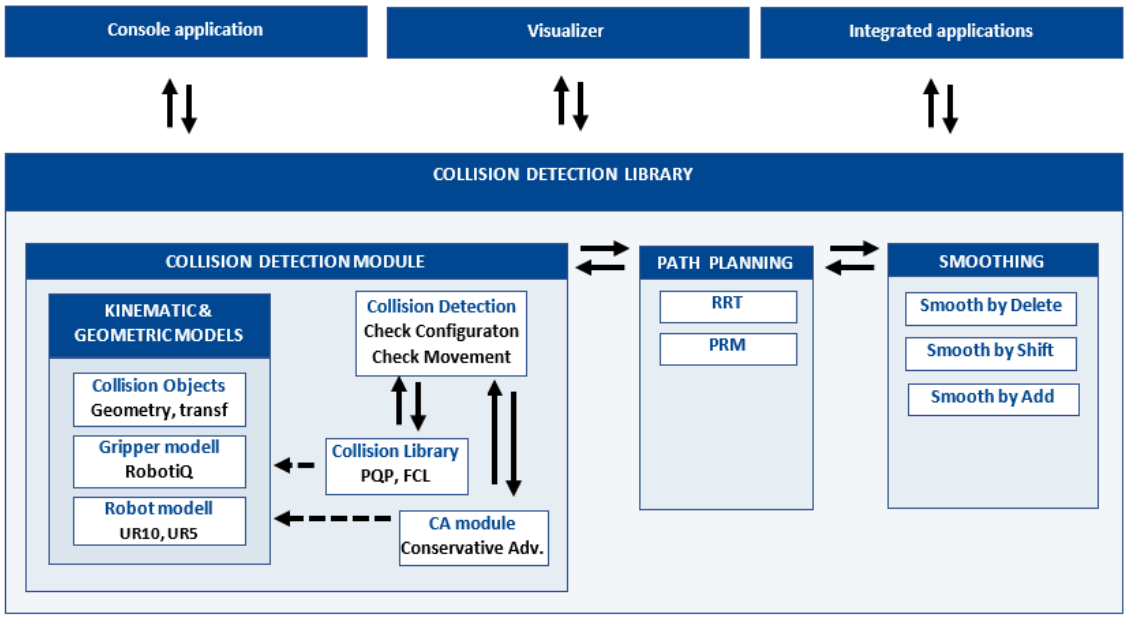


4.8. ábra. Felépített PRM gráf nyomtatott áramkör gyártó cellában, UR10 és egyedi dupla megfogó alkalmazásával

5. fejezet

Implementáció

A bemutatott módszerek vizsgálatához, későbbi felhasználásukhoz elengedhetetlen volt az algoritmusok implementálása, melyeket egy Visual Studio Solution fog össze .NET keretrendszerben C# nyelven. Jelen dolgozatban áttekintésként szerepel a létrehozott implementáció váza, a teljes átláthatóság érdekében, viszont a fejlesztés során hozott tervezői döntések és sémák nem képzik részét, mivel a jelenlegi implementáció 14 000 sort és több mint 70 osztályt és interfészt tartalmaz, amelyek felszínes áttekintése az 5.4 ábrán és az alábbi leírásban követhető végig.



5.1. ábra. Implementált könyvtár áttekintése

Ütközési objektumok tárolója (Kinematic Geometric Models)**:** Az implementáció alapegységei a bemutatott geometriai modelleket csomagoló ütközési objektumok (Collision Object) valamint ezekből építkező magasabb szintű modellek. Ilyen a robot és megfogók modelljének absztrakt osztálya (Gripper modell, Robot modell) amelyek megvalósítják az alapvető működést, például robot konfigurációk beállítása, transzformációk elvégzése, megfogó csukása/nyitása. Így a konkrét megfogó és robot típusok könnyen bővíthetőek, a jelenlegi implementáció UR5, UR10-es kollaboratív robotokat, valamint a RobotiQ és egyedi gyártású megfogókat tartalmaz. A cella konfigurálásánál ezek könnyen cserélhetőek, így lehetővé téve több eszköz kipróbálását egy adott feladatra ezzel segítve

akár beszerzést, akár a munkafolyamat automatizálásakor felmerülő kérdések megválaszolását.

Általános célú ütközésvizsgáló könyvtárak (Collision Library)**:** A fejlesztés során egyetlen felhasznált külső függőség az ütközést és távolságot vizsgáló könyvtár, amely hatékonyan végzi a transzformált geometriák legközelebbi pontjainak távolságmérését. Több ilyen könyvtár is rendelkezésre áll, ebben az esetben a Proximity Query Package (PQP) [9] került felhasználásra, de kezdeti lépésben felmerült a Flexible Collision Library (FCL) [14] integrálása is, ami egy újabb megoldás, viszont a dokumentáció hiánya miatt nem sikerült alkalmazásba venni. A saját könyvtár ennek figyelembevételével készült, egy egységes interfész került kialakításra, amihez a későbbiekben igény szerint kapcsolható másik erre a célra készült könyvtár.

A PQP egy az általunk használt C# nyelvnél alacsonyabb szintű natív C++-ban implementált könyvtár. Hatékony módszerrel képes egy pár transzformált geometrián a két legközelebbi pontot megkeresni, azok távolságát jelezni, valamint az ütközésüket detektálni. A lekérdezések futtatásához az alkalmazás indulásakor fel kell dolgoznia a használt geometriákat, ekkor egy speciális doboz hierarchiát (*bounding volume hierarchy*) alakít ki. Ezt követően a betöltött objektumok azonosítókat kapnak és a tárolóval, ütközésvizsgálatokat végző egységekkel kommunikálva a szükséges lekérdezések futtathatóak.

Conservative Advancement modul (CA Module)**:** A bemutatott Conservative Advancement eljárások alkalmazásához szükséges előfeldolgozás, becslés és ütközésvizsgálat külön egységbe zárva szerepel, így a saját logikájához szükséges többlettudás nem vegyül a többi komponens által használt alapinformációkkal. Amennyiben nincs rá szükség, nem terheli a többi módszert a saját futási feltételeivel. Az algoritmusok bemutatása során elkerülhetetlen volt az implementációban használt fogalmak bevezetése, mivel az eredeti cikk [17] leginkább elméleti síkon tárgyalja az eljárást, jelen esetben pedig a hiányzó konkrétumokat is szerettük volna megmutatni. Az eredeti és fejlesztett becslő eljárás is megvalósításra került a mérések során való vizsgálathoz.

Ütközésvizsgálat (Collision Detection)**:** Az eddig részletezett egységek implementálását követően, a megfelelő cellamodell, robot mozgatási műveletek és általánoscélú ütközésvizsgálat, rendelkezésre állnak. Így a bemutatott statikus konfigurációra és mozdulatokra vonatkozó algoritmusok implementálhatók a kiegészítő logikát követően, mint ütközésvizsgálati szabályok, lekérdezések és segédfüggvények és konfigurációk kezelése.

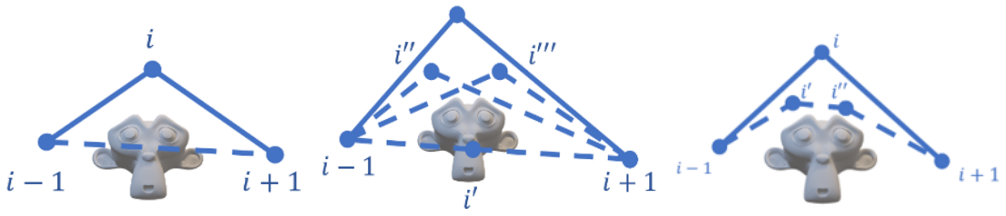
Pályatervezés (Path Planning)**:** A pályatervezés során az algoritmusok ütközésvizsgálati igénye leginkább arra terjed ki, hogy a robot által megtett mozdulat vagy felvett pozíció ütközésmentes vagy sem, esetleg a biztonsági távolság és a használt módszer beállítása szükséges. Emellett alapvető cellamanipulációs műveletek, mint munkadarabok megfogása, elengedése, robot mozgatás, egyéb cella objektumok helyzetének és orientációjának változtatása. Így a pályatervező algoritmusok és a ütközésvizsgálat között egy olyan interfész került kialakításra ami a bonyolultabb, nem mindennapi használat logikát igyekszik elrejteni. A összes funkció töredékét biztosító metódusok és tagváltozók az interfészen keresztül könnyen áttekinthetők és bármely más alkalmazásban, új algoritmusok fejlesztése során gyorsan használatba vehetőek. A pályatervező algoritmusok bemenetei, kimenetei és metódusai könnyen formalizálhatók, mivel általában két állapot, konfiguráció közt szeretnénk mozgást tervezni, valamint a megoldás is feltehetően ezeknek a konfigurációknak egy listája lesz. Ennek megfelelően a már említett single-query és multi-query algoritmusok külön-külön interfészbe terelhetőek, így egymással típusonként csereszabatosak lesznek, és minden már meglévő vagy új algoritmusnak elégséges a saját funkcionalitásával törőd-

nie. Így ebben az esetben is további algoritmusok könnyen implementálhatóak, mérhetőek az interfészek megvalósításával. Mivel a legtöbb ilyen algoritmus futása során leginkább általános vagy fa típusú gráfokat épít és azokat keresésre, állapotok tárolására használja, egy általános gráf könyvtár implementációja külső könyvtár[1] felhasználásával került a megvalósításba, ezzel egységesítve a pályatervező algoritmusok belső működését.

Simító algoritmusok (Smoothing)**:** A pályatervezés kimenetén megjelent ütközésmentes pályák minősége nagyban függ a használt eljárásoktól, azok paraméterezésétől és a randomizált működés miatt az éppen aktuális szerencsétől is. A simító eljárások a meglévő pálya pontjain igyekeznek olyan módon változtatni, hogy a pálya megtételének ideje vagy hossza rövidebb legyen. Ehhez azonban továbbra is ütközésvizsgálati módszerekre van szükségünk hiszen egy változtatást csak úgy tudunk megtenni, ha azzal a pálya ütközésmentes marad. Ezeknél az algoritmusoknál is fontos szempont volt az egymással helyettesíthetőség és egymás után fűzhetőség, így minden ilyen algoritmusnak ki- és bemeneti oldalon egyaránt egy konfigurációlista szerepel valamint egyetlen a folyamatot indító metódus. A simító algoritmusok indítását az algoritmus paramétereinek konfigurálása előzheti meg, de képesnek kell lenniük külön beállítások nélkül, egy általánosan jónak gondolt alapértelmezett módban futni, a könnyű használat érdekében.

A simító algoritmusok a dolgozatban több helyen megjelennek, azonban pontos működésük nem képzi részét. Alapvető kérdés felhasználásuk során, hogy mennyi időnk van a pályatervezés kapott megoldás javítására. Leglátványosabb eredményre a törlést használó algoritmusok képesek, itt minden három csomópontot megvizsgálunk és amennyiben az általuk leírt mozdulat a középső elvételével is ütközésmentes marad, a vizsgált pont elhagyható. Ennek egy időigényesebb variánsa mikor ezeket a lehetséges törléseket nem hajtjuk végre szekvenciálisan amint lehetséges, hanem több eshetőséget megvizsgálva a legkedvezőbbet választjuk, hiszen lehet, hogy egy törölhető pont meghagyásával később egy még jelentősebb pontot hagyhatnánk el. Ezt követően érdemes megvizsgálnunk, hogy az egymást követő pontok vihetőek egymáshoz konfigurációs térben közelebb, ezzel a legkedvezőbb egyenes utat közelítve. Egy harmadik eshetőség, hogy érdemesebb egy konfigurációt, kettővel helyettesíteni, ilyen módon egy akadály kerülése során, annak határait jobban lekövetve gyorsabban kivitelezhető mozgást találni.

Az egyszerűbb törléssel, csomópont eltolással, valamint hozzáadással való javítást alábbi (5.2) ábra mutatja be. A *Kísérleti eredmények* fejezetben a simítás minden esetben az első két módszer alkalmazását jelenti.



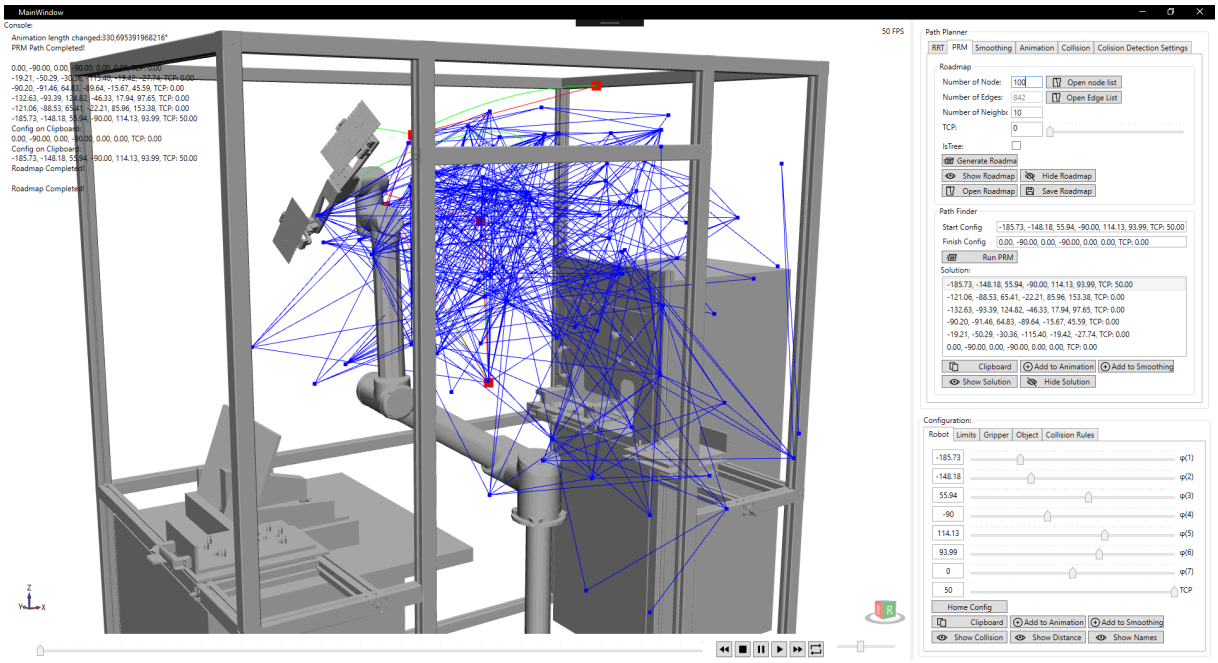
5.2. ábra. Törléssel, eltolással és hozzáadással simított pálya

Újrafelhasználhatóság (Console Application, Visualizer, Integrated applications)**:** A bemutatott részegységek tehát rugalmasan variálhatóak, sorrendezhetőek és alkalmazástól függően konfigurálhatóak a sebesség, minőség és megbízhatóság szempontjai mentén. A kevés külső függőségnek köszönhetően a komponensek egyetlen .dll szerelvénybe .NET Standard Class Library-ként, vagy önálló futtatható fájlba csomagolhatók a használt geo-

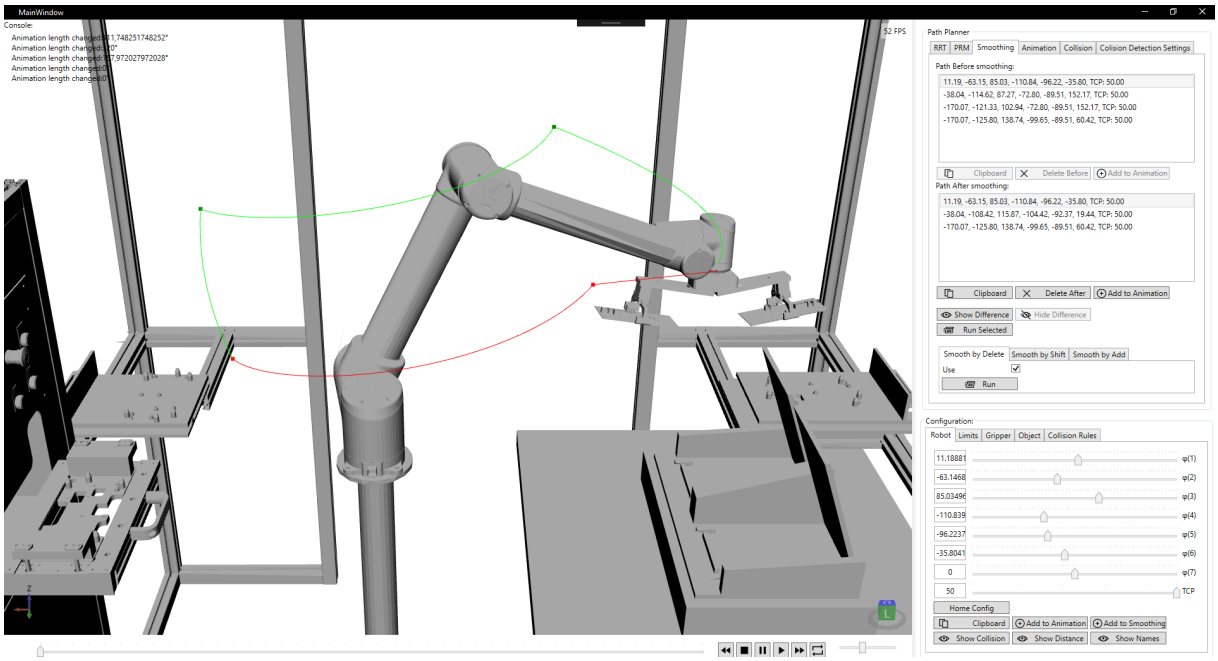
[1] https://github.com/pelikhan/quickgraph

metriák mellékelésével. Így az itt bemutatott funkcionalitás könnyen felhasználható más C# alapú projektben, RaspberryPi és Windows 10 IoT felhasználásával beágyazott eszköz méretben, alkalmazás szervereken, Azure felhőszolgáltatásokban és Docker konténer alapú részegységként. Az elkészült csomag jelenleg is használatban van a kutatócsoportnál Wolfram mathematica alkalmazással, dll szerelvényként importálva.

Vizualizáció: Külön projekt formájában a konzolos felület mellett elkészült egy vizuális megjelenítésre alkalmas eszköz is. Az alkalmazás .NET WPF alapokra épül, HelixToolkit megjelentő segítségével. Sok esetben a fejlesztés során hasznos eszköznek bizonyult ütközések vizsgálatának, transzformációk helyes elvégzésének ellenőrzésére. Emellett a dolgozatban szereplő ábrák is ennek a megjelenítőnek a segítségével készültek. Az alkalmazás segítségével könnyen használhatóak a pályatervező algoritmusok, a futás során készült gráfok megjeleníthetőek, a kapott pályák simítása, megjelenítése, módosítása és animáción való megtekintése is lehetséges. Valamint a cella megépítését követően, alapvető tervezési műveletek ablakos alkalmazással is elvégezhetőek.



5.3. ábra. Cella vizualizációja, PRM ütközésmentes gráffal



5.4. ábra. Cella vizualizációja, simított és simítatlan pályák megjelenítésével

6. fejezet

Kísérleti eredmények

A jelenlegi fejezet a már bemutatott eljárásokat továbbfejlesztett, optimalizált formájukban elemezi. Minden bemutatott mérés az implementált Visual Studio Solution segítségével kerül vizsgálatra a laborban elkészített cellamodellek és kiegészítő folyamatok felhasználásával. A elért eredmények egy átlagos teljesítményű Intel Core i5-4200M és 8 GB RAM-ot használó hordozható számítógépen kerültek kimutatásra.

6.1. Esettanulmányok

Esettanulmányként két, már a korábbi ábrákon látott cellamodell kerül bemutatásra, amelyek valós ipari igényeket elégítenek ki. Ezeken keresztül az elkészített algoritmusok teljesítménye vizsgálható, valamit az elvárt feladatokhoz mérten betöltött szerepük mérlegelhető, így az alkalmazásokhoz tartozó megfigyelések a kísérleti eredmények után kerülnek magyarázatra.

6.1.1. Nyomtatott áramkörök gyártásának automatizálhatósága

Egy nyomtatott áramkörök gyártásával foglalkozó cég számára megvalósíthatósági tanulmány készült el azzal a céllal, hogy a jelenleg emberi munka befektetésével végzett folyamat hogyan automatizálható. Így került megtervezésre a 6.1.ábrán szereplő gyártócella, amely a valóságban nem létezik. A cellában elhelyezett gépek és megmunkáló állomások között történik az alaplapok mozgatása a középpontban elhelyezett UR10-es kollaboratív robot segítségével. A cella középpontjában viszonylag nagy tér áll rendelkezésre, azonban az elérni kívánt pozíciók a cellának igen sűrű, speciális pontjain helyezkednek el. A tartótálcákon kialakított távtartókra és a gépekben kialakított befogatásokba való illesztés, valamint a speciális dupla megfogó nagy térbeli kiterjedése pályatervezési szempontból kihívást jelent. A pályatervezés célja 7 konfiguráció bejárása, ott az alaplapok elhelyezése vagy felvétele.

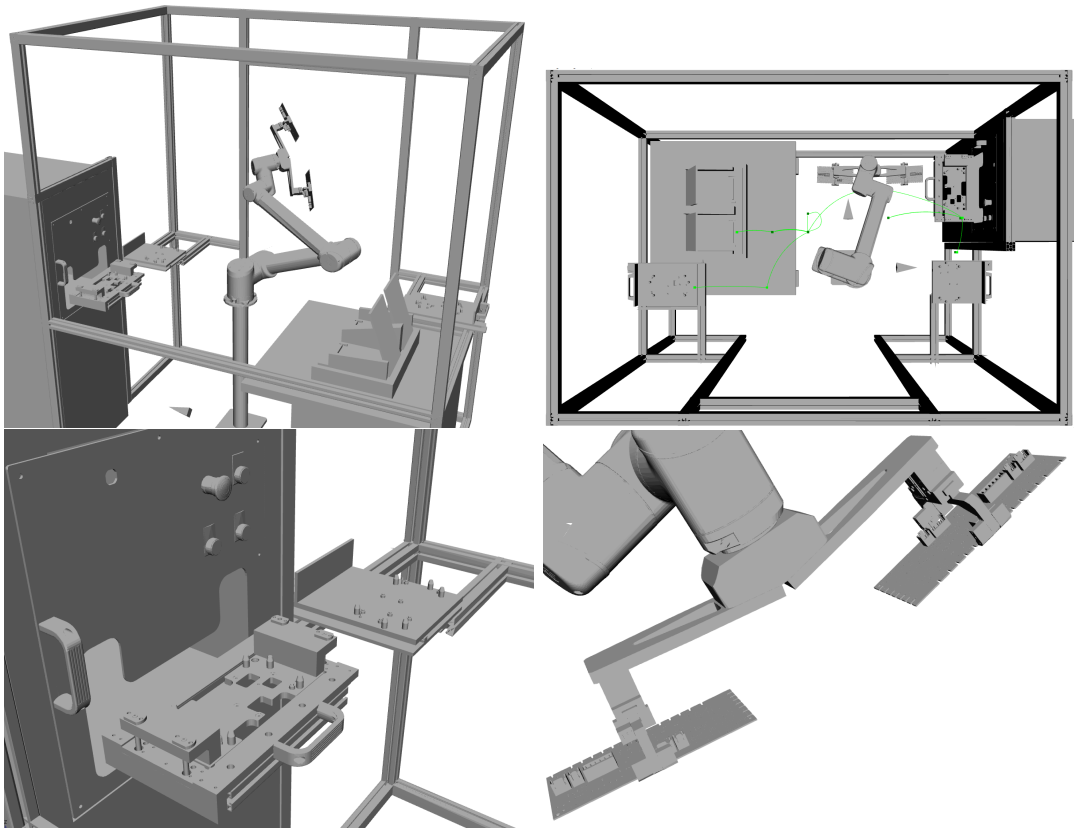
A digitális modell elkészítésével, az első csavar megvásárlása előtt a tervezett cellában mérhetőek a ciklusidők, a várható kihozatal és egyéb paraméterek. Emellett eldönthető, hogy elegendő esetleg az UR10-es helyett a kisebb, olcsóbb UR5-ös beszerzése vagy a jelenleg is használt dupla megfogó egyedi elkészítése milyen előnyökkel jár szimpla verziójához képest.

Egyetlen, a feladatot kielégítő pálya bemozgatással való elkészítése a már megépített cellában nem jelent problémát, a robotok gyártói által kidolgozott oktatóanyagoknak köszönhetően. Viszont amennyiben a cella megépítetlen állapotában kell előzetes mérőszámokat mondanunk a feladatok komplexitásától függően sok mérnöki munkaórát vehet igénybe. Az elkészült vizualizációs pályatervező eszköz már ebben is segítséget nyújthat, azonban erre a célra alkalmasak a *Bevezetésben* bemutatott szoftverek is, de használatuk

még mindig sok erőforrást és tudást feltételez. A létrehozott könyvtár segítségével azonban a mozgástervezés magas szintű automatizáltság mellett elvégezhető.

Nem beszélve egy olyan esetről amikor egyetlen pálya ismétlése nem elegendő, szeretnénk ennél komplexebb funkcionalitást elhelyezni a gyártócellában. Feltehetjük, hogy a gyártó egy adott cellában több különböző terméket szeretne gyártani. Így a cellába be- és kikerülő részegységek sok szempontból azonosak, de valamilyen tulajdonságuk szerint más helyről kell az alapanyagokat vennünk és az elkészült termékeket szintén típus szerint differenciálnunk kell. Elképzelhető az is, hogy bizonyos munkafolyamatok kimaradnak vagy rövidebb/hosszabb ideig tartanak a különböző munkadarabok esetén. Ekkor már nagyszámú ütközésmentes pályára van szükségünk, például a 7 rögzített pontunk esetén bármely pontból bármelyikbe való eljutás 42 pályát feltételez és amennyibe a dupla megfogó átforgatása is szükséges ez máris 168 pályát jelent, ami hagyományos tervezési eszközökkel nem lehetséges. Amennyiben a megfelelő válaszidőket és az ütközésmentességet garantálni tudjuk egy ütemező, sorrendtervező vezénylésével, akár a műveletvégzésekkel egyidőben tervezhetőek a robot mozdulatai.

A cella geometriája 70 000, az UR10-es robot a megfogóval és két munkadarabjával 195 000, tehát a modell összesen 265 000 háromszöget tartalmaz.



6.1. ábra. Alaplap gyártáshoz készült cellamodell speciális megfogóval, UR10-es kollaboratív robottal, saját vizualizációs eszköz segítségével megjelenítve.

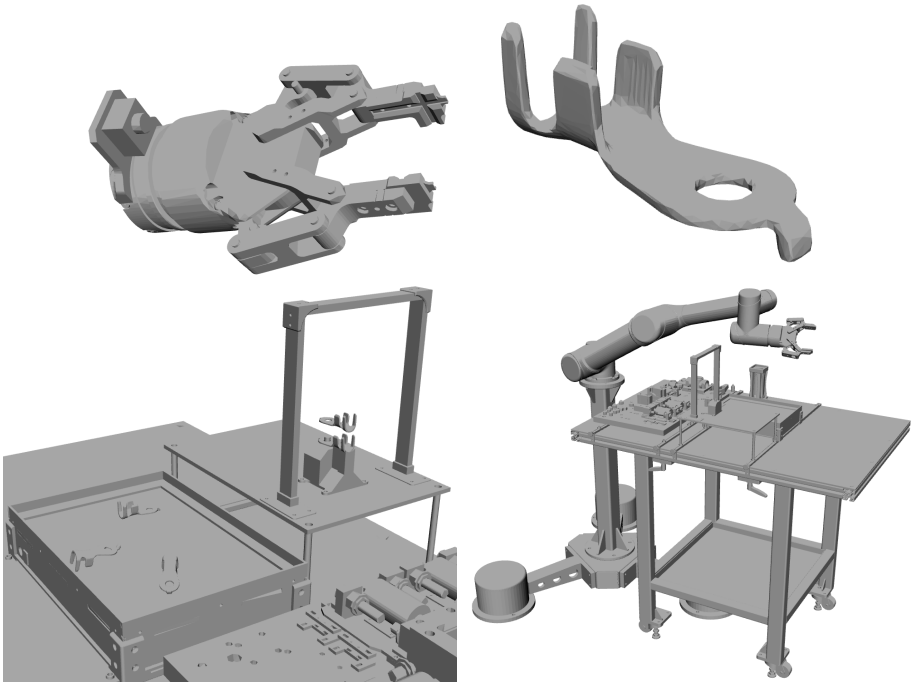
6.1.2. Ömlesztett kábelvégek rendezése műveletvégzéshez

Egy másik napjainkban gyakran előkerülő feladat az ömlesztett alkatrészek rendezése. Az automatizált gyártás, legyen szó gépiesített vagy robotokkal végzett munkáról, előfeltételként kezeli, hogy az alkatrészek pozícióját ismerjük. Ez az állapot azonban nem minden alkatrész esetén tartható fenn folyamatosan, hiszen gyakran dobozokban ömlesztett formában kerülnek szállításra, beszerzésre a felhasználandó részegységek.

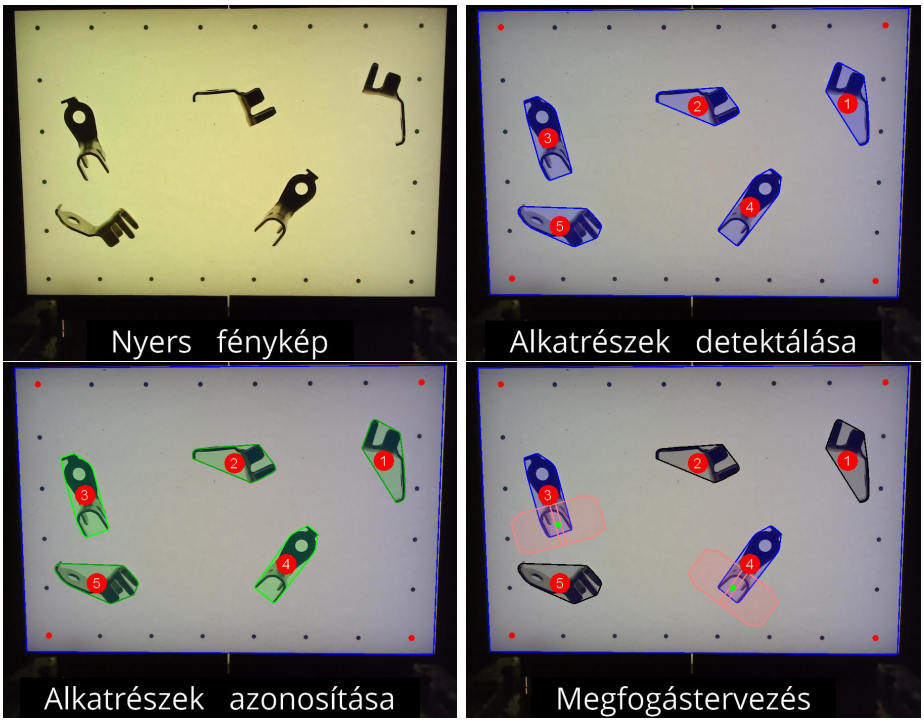
Egy kábelkorbácsokat gyártó cég igénye az volt, hogy az ömlesztve érkező kábelvégeket előre megadott helyre, egy présgépbe kell egyesével adagolni. Az ilyen típusú feladatok hagyományos megoldása, általában rezgő illetve lépcsős adagolók segítségével történik, azonban ezeket az eszközöket adott alkatrészekre egyedileg kell tervezni és gyártani, valamint több nyugalmi helyzettel rendelkező szabályos testek esetén, például furatok helye nem garantálható ilyen módon. Ezért erre a feladatra egy képfeldolgozás alapú robottal végzett rendezés merült fel lehetséges megoldásként.

A folyamat tehát a dobozban érkező kábelvégekkel kezdődik, ezek egy rezgő levilágítóasztalra kerülnek kiszórásra, a robot a kijelölt terület fölé emelkedik és a rá rögzített kamera segítségével képet készít az elhelyezett tárgyakról. A kép készítésekor a levilágítóasztal felvillan, így a kapott kép kontrasztosabb lesz, az alkatrészek kontúrja jobban felismerhető. Képfeldolgozó eljárás segítségével a levilágítóasztal széleihez képest meghatározható az elemek elhelyezkedése és orientációja. Kijelölésre kerülnek a megfelelő pozícióban lévő alkatrészek, amelyek hozzáférhetőek a megfogóval. Ezt követően kiszámításra kerül a felvenni kívánt geometriához tartozó megfogási konfiguráció többek közt inverz kinematika segítségével. Így a kapott konfiguráció és a kijelölt lerakási pont közt alkalmazhatóak a pályatervezési algoritmusok.

A cella geometriája 737 000, az UR5-es robot a RobotiQ megfogóval 44 000, tehát a modell összesen 785 000 háromszöget tartalmaz.



6.2. ábra. Kamerával felszerelt megfogó, munkadarab és a feladat modellezéséhez kialakított levilágító- és rázóasztal geometriája.



6.3. ábra. A roboton lévő kamera által készített kép a feldolgozás során.

6.2. Ütközésvizsgálati módszerek mérése

Az ütközésvizsgálati módszerek az utóbb bemutatott munkaasztal segítségével kerülnek összevetésre amelyet a 2.1 és 6.2 ábra is bemutat. A cella kiválasztása során fontos szempont volt a részletes, lehetőleg nagy számú háromszöggel rendelkező munkatér. Jelenlegi mérés során 785 000 háromszöget tartalmaz és egy statikus konfiguráció vizsgálatát 27 aktív ütközési szabály alapján azonos számú lekérdezéssel dönthető el a teljes cella ütközési állapota. A mérés során 5000 véletlenszerű mozdulat (az összehasonlított eljárások során teljes mértékben megegyező mozdulatok) kerül megvizsgálásra. Ezek a már említett megfontolások miatt 80%-ban ütközéstől mentesek, valamint csuklónként átlagosan 45-50 fokos eltérés van a mozdulat c_a és c_b konfigurációja között. Az összevetett eszközök között szerepel a már említett RoboDK által kínált Python alapú, mintavételezéses ütközésvizsgálat. Ennek saját könyvtárban implementált finomított megközelítése, valamint a Conservative Advancement eredeti és továbbfejlesztett verziója. Az így kapott eredményeket és az eljárások tulajdonságait a 6.1. táblázat mutatja be.

	Mintavételezés (RoboDK)	Mintavételezés (saját)	CA (eredeti)	CA (fejlesztett)
Mintavételezési frekvencia	1°	1°	-	-
Biztonsági távolság	-	-	0 mm	0 mm
Formális garancia	-	-	✓	✓
Hibás detektálás	15	2	0	0
Idő [mm:ss]	38:08	01:41	05:47	02:00

6.1. táblázat. Ütközésvizsgálati módszerek mérése

A két mintavételezéses eljárásban a mintavételezési frekvencia 1-1 fok volt és biztonsági távolság, formális garancia a *3.3. Mintavételezés* fejezetben leírtak alapján a hozott döntésre nem adható. Jól látható, hogy a RoboDK eljárásának hatékonysága, futási ideje a nagy mennyiségű ütközésvizsgálati kérést tartalmazó tervező algoritmusok fogadására kevésbé alkalmas, leginkább a mérnöki tervezés során előkerülő szakasz ellenőrzésére használatos. A zárt interfész miatt azonban a RoboDK által használt módszerről több nem mondható el. A saját könyvtárban elkészített mintavételezési eljárás a mozdulatban szereplő konfigurációk és azok statisztika alapú rendezésével optimalizált formában jelentősen jobb eredményt ért el futás idő tekintetében. A teszthalmaz futtatása után a CA eljárásoktól való eltéréseket megvizsgálva észrevehető volt, hogy a mintavételezéses algoritmusok valóban képesek mintavételezésből adodó hibákat elkövetni. Mivel ezeknek a hibáknak a feltétele, hogy a mérés során például 1-1 fokon belül legyen, illetve ne legyen ütközés, így az asztalon, vastagabb eszközökön való átnyúlás nem lehetséges. Tehát leggyakrabban alig érintett, súrolt, kisebb jellegű ütközések a tipikusak, különös tekintettel a megfogó, mint gyakran ütköző elem vékony pofáinak szerkezetére. A hibás detektálás sorában álló értékek miatt a mintavételezéses eljárással ellenőrzött mozdulatok és pályák emberi vagy más módszer felülvizsgálata nélkül automatikusan nem hajthatóak végre a roboton, mivel megvan a lehetőség az ütközésre.

A mintavételezéses algoritmusok egy másik problémája a simító eljárásokban való felhasználás, amelyek futtatása során már a meglévő konfigurációkat próbálják minél közelebb vinni egymáshoz, a legrövidebb egyenes mozgás közelítését erőltetni. Ezzel együtt a került akadályokat minél jobban közelíteni, így ez esetben nagy számban olyan szakaszokat kell ellenőrizni, amelyek éppen érintik/nem érintik vagy nagyon közel haladnak el az akadályok mellet. Ez azt eredményezi, hogy ezekkel a próbálkozásokkal a fent említett kis ütközéseket produkálja, tehát a mintavételezés problémáját igyekeznek kihasználni. Többek között ezért is hasznos lenne számunkra a biztonsági távolság és formális garanciát vállaló eljárás, mint a Conservative Advancement módszerei.

Áttérve a táblázat másik felében szereplő CA eljárásokra, láthatjuk, hogy folytonos eljárásokként nincs szó mintavételezési frekvenciáról. A biztonsági távolság megadható, jelen esetben 0 mm, mivel ez felel meg a mintavételezési eljárások nem létező értékével és a hibás detektálások csak így figyelhetőek meg. De lehetőségünk és képességünk megvan a használatára, a pályasimító eljárásokkal kapcsolatban tett megjegyzés és a kalibarációs, gyártási pontatlanságok miatt fontos szerepe van gyakorlati használhatóság szempontjából. Továbbá a döntéseinkre formális garanciát tudunk vállalni, mivel az eljárást alapelveinek figyelembevételével bővítettük a megfogott objektumok kezelésével és javítottuk a kinematikai lánc mozgás jobb becslésével. Így a CA eljárás esetén, a vizsgált szakaszok és létrehozott pályák minősége sokkal magasabb lehet, mint mintavételezéssel készített társaiké. Ennek tükrében a CA eljárások megérhetik azt a többlet időt amely a futásukhoz szükséges, különös tekintettel a továbbfejlesztett változatra amely ezen az időkülönbségen hivatott tompítani. Ezzel az eredeti algoritmushoz képest kevesebb mint felére csökkent az időszükséglet, azonos eredmények mellett.

További kiegészítés az elért eredményekkel kapcsolatban, hogy a mintavételezéses eljárások egyszerű bináris értékű ütközésvizsgálati lekérdezéseket, míg a Conservative Advancement módszerek a költségesebb távolságalapú lekérdezéseket használják. Itt a felhasznált PQP nevű csomag lehetőséget ad a távolság lekérdezések pontosságának hangolására. Ezeknek a paramétereknek, kiegészítve a biztonsági távolsággal való hangolása az alábbi 6.2 táblázatban tekinthetőek meg.

A távolság alapú lekérdezések használata során a kiinduló álláspont a teljes pontosság elvárása volt. Az ütközésvizsgáló könyvtár lehetőséget ad relatív, illetve abszolút hibahatárok definiálására, amelyek alapján a lekérdezésünk pontossága és válaszideje közti egyenesen egyensúlyozhatunk. Tehát amennyiben a relatív hibát használunk egy százalékos

PQP Relatív hiba [%]	PQP Abszolút hiba [mm]	Biztonsági távolság [mm]	Szakasz vizsgálat Ütköző/ nem ütköző	Hibás detektálás	Ideje [sec]	Átlagos szakasz lekérdezése [sec]
0	0	0	105 / 95	-	105.16	0.53
0	0	5	115 / 85	-	84.95	0.42
-	2	0	106 / 94	1	56.86	0.28
-	5	0	112 / 88	7	34.79	0.17
-	8	0	123 / 77	18	24.87	0.12
-	2	5	122 / 78	7	43.47	0.22
-	5	5	130 / 70	15	25.13	0.13
-	8	5	198 / 2	83	6.89	0.03
0.1	-	0	105 / 95	0	27.79	0.14
0.5	-	0	105 / 95	0	15.30	0.08
0.8	-	0	105 / 95	0	14.25	0.07
0.9	-	0	105 / 95	0	14.13	0.07
1	-	0	105 / 95	0	16.58	0.08
0.1	-	5	117 / 83	2	19.14	0.10
0.5	-	5	123 / 77	8	9.00	0.04
0.8	-	5	127 / 73	12	11.14	0.06
0.9	-	5	126 / 74	11	7.77	0.04
1	-	5	129 / 71	7	7.39	0.04

6.2. táblázat. Lekérdezésk futásának ideje és pontossága ütközésvizsgáló könyvtár paramétereinek és biztonsági távolság alkalmazásának figyelembevételével

érték adható meg, abszolút esetén pedig egy milliméter érték, amely a visszaadott távolságérték függvényében pozitív irányban értelmezettek, tehát így a PQP által visszaadott érték minden esetben nagyobb lesz.

A mérés során távolság lekérdezéseket használó szakaszellenőrzések kerülnek futtatásra, amelyek során azok futásidejét és a hozott döntéseket vizsgáljuk, hogy egy adott paraméterhármas esetén milyen arányban detektálunk ütköző, illetve nem ütköző szakaszokat. Az alapvető kiindulási érték a teljes pontossággal és 0 mm, illetve 5 mm biztonsági távolsággal futtatott lekérdezések időbeli és döntési eredménye.

Mind a két típusú hiba használata esetén a kapott távolságokon lehetőségünk van korrekcióval élni. Abszolút hiba esetén a távolság értékből levonható a használt maximális hiba értéke, tehát $dist' = dist - PQP_{abs}$. Ez azért tehető meg, mivel a mért távolságok esetén a hiba minden esetben pozitív irányú és az előjele helyes. Relatív hiba esetén, a távolság értékből levonható a kapott lekérdezés értékének függvényében relatív hiba mértéke, tehát $dist' = \frac{dist}{1+PQP_{rel}}$, ahol szintén kihasználjuk a tévedés pozitív és előjel helyes tulajdonságát.

Így az abszolút hibát tekintve előfordulhat, hogy a korrigált érték ennek köszönhetően amennyiben a vizsgálat során a könyvtár nem használja ki az abszolút hiba mértékét, egy ütközésmentes szakasz ütközőnek lesz titulálva. Ez a hiba azonban az esetek kis részét érinti, futásidőben kedvező eredményt hoz és fontos, hogy csak negatív irányú tehát helyes megoldások vesztésével jár. Relatív értékek esetében hasonlóképpen juthatunk futásidőbeli előnyhöz, minimális minőségbeli romlás árán. A relatív hiba korrekciója a táblázat alapján hatékonyan képes pontosságunkat fenntartani, mivel ebben az esetben ütközés közeli, kis távolság esetén a hiba is biztosan kicsi, tehát nem képes az ütközés tartományába vinni a kapott távolságértéket. Mind a két esetben a biztonsági távolság bevezetése ront a hibás detektálások arányán, mivel az eddigi 0 mm felhasználásával vizsgált távolságok esetén az

előjel helyesség tulajdonsága egybe esett azzal a határral amely alapján egy távolságértéket ütközőnek titulálunk vagy sem. A biztonsági távolság bevezetésével ez az egybeesés megszűnik és több érték fog a hiba és a korrekció miatt egyaránt ütközőnek minősülni.

Ezek alapján a relatív hiba használata előnyösebb számunkra, hiszen biztonsági távolság használata nélkül képes szinte teljes pontosságot tartani, ellenkező esetben az ütközés közeli szakaszok 15%-át veszítjük el, ami a 8-10 szeres lekérdezési idő különbözet figyelembevételével elhanyagolható.

6.3. Pályatervezési módszerek mérése

6.3.1. Mozgástervezés Probabilistic Roadmap segítségével

Elsőként a bemutatott PRM eljárás kerül vizsgálatra, amelynek során elsősorban az általános gráf generálási idejére, általános pályatervezései válaszidőre, a válaszban foglalt pálya minőségére és a rajta alkalmazott simítási folyamat részleteire vagyunk kíváncsiak. Az első esettanulmányként bemutatott nyomtatott áramkör gyártásra használt cella erre kiváló lehetőséget biztosít, mivel egy a speciális nagyméretű megfogó, valamint a sűrű helyek bejárása kihívás pályatervezési szempontból. A vizsgálat során különböző méretű gráfokban vizsgáljuk a mozgástervezés részleteit, egy 1 000 mozdulatból álló teszthalmaz segítségével, amely a már említett 7 bejárandó pontot valamint véletlenszerű, köztük félreeső, általában nem használt végpontokkal. A gráfok generálásának, illetve a tervezés legfontosabb részleteit a 6.3 táblázat mutatja be. A 6.3 és 6.4 táblázatban szereplő pályahossza másodpercben vannak kifejezve, amelyek a robot megadott gyorsulási és maximális sebesség profiljának megfelelő útidőt jelölik, így azok aránya a kifejező és leginkább minőségi viszonyokat hivatott leírni.

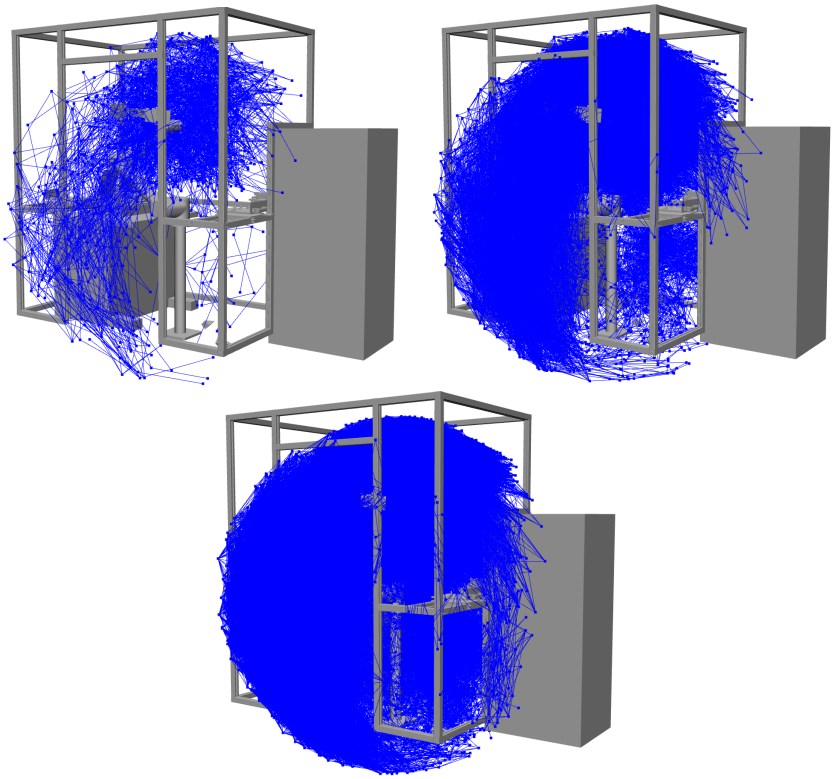
Csomópontok száma	Vizsgált szomszédok	Élek száma	Generálási idő [min]	Átlagos lekérdezés [sec]	Pálya hossza [sec]
1 000	5	4 506	00:35	0.80	4.80
1 000	10	8 474	01:21	0.83	4.46
1 000	20	14 170	02:22	0.75	4.35
5 000	5	14 423	02:38	0.88	5.02
5 000	10	26 169	05:20	0.81	4.21
5 000	20	47 841	12:48	0.79	4.17
10 000	5	31 322	05:36	0.84	4.88
10 000	10	58 994	11:00	0.87	4.50
10 000	20	107 819	23:06	0.87	3.39

6.3. táblázat. PRM gráfok generálási részletei, átlagos teljesítményük

A PRM futtatása előtt az ütközésmentes térkép generálásakor megadható az elvárt csomópont szám (1. oszlop) és a kapcsolatok száma (2) amelyet a lokális tervező minden csomópont esetén a legközelebbi pontokkal megpróbál kialakítani. Az adott cella sűrűségének megfelelően a felvett pontok számától és a célzott élszámtól függően jön létre az adott kapcsolattal (3) rendelkező általános összefüggő gráf az eredeti cikkben [7] említett keresőfához képest. Az feltérképezés során az említett szakaszellenőrzési módszerek közül bármelyik felhasználható, jelen esetben a gyakorlatban is kivitelezhető pályák érdekében a saját Conservative Advancement eljárás kerül felhasználásra. A gráfok generálásának ideje nem sok esetben tervezési szempont mivel csak egyszer kell megtenni és nagyban függ a robot környezetétől. Több, jól definiált állapotú cellában akár több ilyen térkép

is létrehozható, így ezek cserélgetésével például CNC megmunkáló állomás esetén a gép nyitott és csukott ajtajának megfelelően tervezhetőek a mozdulatok.

Az átlagos lekérdezési idő (5) magába foglalja a tervezni kívánt kiinduló és befejező konfiguráció bekötési idejét, köztük a gráfkeresést és a pálya simításának idejét. A pálya hossza (6) a mozgástervezési kérésre érkezett megoldás robottal való megtételének ideje. Előbbi (5) értéke futásidő az alkalmazás szempontjából jónak mondható közel állandó érték ami önmagában vizsgálva függetlennek mondható a gráf méretétől. Az így kapott pályák hosszával (6) ellentétben, itt leginkább a csomópontok fokszáma határozza meg milyen minőségű pályákat kapunk. Utóbbi két érték összetételét a 6.4 táblázat részletezi, valamint a generált térképek sűrűségét és a megfogó által elérhető térbeli koordinátákat a 6.4 mutatja be.



6.4. ábra. Különböző méretű PRM gráfok a cellában.

A lekérdezés során amennyiben a kiinduló és végpont nem szerepel a gráfban azok bekötése szükséges, ezt vagy az egyszerű lokális tervező a két mozdulat vizsgálatát követően a csuklótérben legközelebbi csomópontokhoz meg tudja tenni vagy ha, ez egy nehezebb feladat, a másik bemutatott lokális tervezővel, az RRT algoritmus lépéskorlátos változatával lehet próbálkozni. Ez azonban csak az esetek töredékében szükséges (2. oszlop), ezért a végrehajtási időben nem mutatkozik meg valamint nem is túl kifizetődő (3) és szintén a nagyobb, illetve gazdagabb kapcsolatokkal rendelkező gráfok esetén kedvez, azzal együtt hogy az így futtatott RRT-nek már valószínűleg nagyon nehéz vagy lehetséges hogy lehetetlen dolga lesz. Az így bekötött pontok között szélességi keresést végzünk (4), aminek az aránya a lekérdezési időben más okból, de szintén nem számottevő, valamint az élek számával költsége növekszik. Az eddigi lépések megtétele után 0.5 másodpercen belül, és a kirívó esetekben 1.5 másodperc alatt adott minőségű (5) ütközésmentes pálya áll rendelkezésünkre. A létrejött mozgássorozat viszont a simítás utáni pályahossz (5,7) alapján nem a legrövidebb módja a pálya bejárásának. Ezen simító algoritmusok segítségével javíthatunk. Illetve látható, hogy az előző táblázatban bemutatott átlagos lekérdezési idők

tetemes részét teszik ki (6) amellett, hogy esetenként fele olyan hosszúra csökken a megtett út ideje. Ebben az esetben is elmondható hogy a sűrűbb gráfokban az eredetileg megtalált utak már kiinduláskor hatékonyabbak, azonban több csomópontból is állnak, így közel azonos ideig tart a a simításuk.

A bemutatott adatokon is látható, hogy nem minden esetben lehet egyértelműen kijelenteni növekvő vagy csökkenő tendenciákat, mivel a vizsgált algoritmus alapvetően randomizált eljárás, így a térképek generálása, az RRT futtatására felső korlát vagy időbeli garancia nehezen adható. Általánosságban azonban elmondható, hogy a kapcsolatok, élek száma prioritást élvez a csomópontok számával szemben. A bemutatott eljárással nagy általánosságban másodpercen, de kedvezőtlen esetben is 5 másodpercen belül nagyon nagy valószínűséggel kapható értékelhető optimalizált pálya, így az említett alaplap pakolási feladat során a végzett műveletek hosszának figyelembevételével akár "*just in time*" módon alkalmazható.

PRM	RRT használat	RRT ideje [sec]	Gráfkeresés ideje [sec]	Szakasz hossza [sec]	Simítás ideje [sec]	Simítás utáni hossza [sec]
1 000 * 5	1.4%	0.47	0.0010	4.80	0.80	2.33 (-2.47)
1 000 * 10	1.6%	0.59	0.0013	4.46	0.82	2.38 (-2.08)
1 000 * 20	0.4%	0.46	0.0017	4.35	0.86	2.47 (-1.88)
5 000 * 5	1.4%	0.31	0.0030	5.02	0.87	2.23 (-2.79)
5 000 * 10	0.6%	0.54	0.0042	4.21	0.81	2.34 (-1.87)
5 000 * 20	0.4%	0.13	0.0061	4.17	0.78	2.37 (-1.80)
10 000 * 5	0.4%	0.31	0.0066	4.88	0.82	2.29 (-2.59)
10 000 * 10	0.4%	0.63	0.0100	4.50	0.85	2.38 (-2.12)
10 000 * 20	0.3%	1.40	0.0142	3.99	0.85	2.36 (-1.63)

6.4. táblázat. PRM lekérdezésének részletei, a gráfok méretének függvényében

6.3.2. Mozgástervezés Rapidly-exploring Random Trees segítségével

Az RRT implementáció tesztelésére a bemutatott kamerás kábelvég mozgatási feladat alkalmas, mivel a kábelvégek a levilágító asztalon minden esetben máshogy helyezkednek el, ezáltal az asztalon lévő akadályok is változnak. Ezt az alkatrészek számára kijelölt terület környékére generált felvételi és alkalmas lerakási konfigurációk párosításával létrejött tesztalmazon tudjuk elvégezni. Az így keletkezett 114 feladatot előfeldolgozás nélkül 785 000 háromszöggel rendelkező cellában szeretnénk megoldani. A 6.5 táblázatban ezeknek a feladatoknak különböző paraméterekkel futtatott RRT példányok eredményei láthatóak. A paraméterek (1) két gráfba bekerülő konfiguráció szögtávolságát, az utolsó csomópont maximális bekötési szögtávolságát, valamint azt, hogy ez alulról számítva hány csuklón kerüljön betartásra a gráfba bekerülő csomópontok közt. Ez a lépéstávolság próbálja a fa szomszédos csomópontjait közel tartani egymáshoz, mert amennyiben távoli csomópontokból építkezünk az élek ellenőrzési ideje megnő és a létrejött utak is hosszú kerülőt fognak tartalmazni. Ellenben a csomópontok bekötése könnyíthető és gyorsabban vagyunk képesek nagy konfigurációs távolságot megtenni. Jelen esetben az algoritmus iteráció száma korlátozott, amennyiben az utat nem sikerül 2000 csomópont bekötésével megtalálni, a pályatervezési kérés sikertelen.

RRT	Sikeres/ sikertelen lekérdezés	Átlagos tervezés			Átlagos simítás		Teljes lekérdezés [sec]
		Iteráció	Ideje [sec]	Pálya hossza [sec]	Ideje [sec]	Pálya hossza [sec]	
100/100/1	114 / -	159	0.73	8.80	4.65	4.94 (-3.86)	5.38
100/100/2	114 / -	169	0.55	8.34	3.71	4.07 (-4.27)	4.26
100/100/3	114 / -	174	0.48	8.18	3.53	4.07 (-4.11)	4.02
100/100/4	110 / 4	569	1.25	8.18	3.47	3.99 (-4.19)	4.71
50/100/2	105 / 9	645	0.87	7.99	3.23	3.90 (-4.09)	4.10
50/100/3	103 / 11	851	1.25	8.22	3.11	3.91 (-4.31)	4.36
50/100/4	40 / 74	1 680	1.57	6.21	2.83	3.56 (-2.65)	4.40

6.5. táblázat. RRT pályatervezsi kérések részletei

Látható, hogy a nagy megengedett távolságok esetén alacsony iteráció szám (3) mellett gyorsan (4) található megoldás, azonban az út hosszán(5), a simítással töltött időn (6) és egy bizonyos lépték alatt a simított pálya minőségén (7) is meglátszik. Ellenkező esetben a magunkkal szemben támasztott iterációs korlátba ütközünk és a lekérdezés ideje is megnő. A simítás a jelenlegi alkalmazás során is indokolt, a pálya hossza gyakran a felére csökken. A vizsgált RRT példányokat tekintve 100/100/3-as felhasználásával az összes kábelvég elhelyezhető lenne, valamint a legjobb átlagos lekérdezésidőt garantálja a vizsgáltak közül. Jelen esetben ez az eljárás elég gyors ahhoz, hogy a feladatát ilyen formában ellássa, így a képeknek megfelelően szintén alkalmazható.

Jelen esetben is elmondható, hogy a definiált feladat részleteinek kihasználásával, a bemutatott módszerek felhasználásával ebben az esetben is készíthető lenne hatékonyabb eljárás a présgép kiszolgálására. Az RRT mellett bizonyos megkötésekkel a PRM eljárása is használható lenne és ebben az esetben a pályatervezés gyorsabban és biztosabban végezhető volna.

Összefoglalás

Jelen dolgozat témája az ipari robotos cellákban történő ütközésvizsgálat, pályatervezés áttekintése és megvalósítása, valamint esetenként ezek továbbgondolása rövidebb futásidő vagy végeredményben jobb minőségű pályák érdekében.

Mindez magában foglalja *szabadformájú geometriák* kezelését és feldolgozását. A robotok (*UR5, UR10*[1]) *kinematikai lánc*ának felépítését *Denavit - Hartenberg* transzformációk segítségével, valamint mechanikus végberendezések (*RobotiQ*[2]) kezelését. *Ütközési objektumok* és *ütközési szabályok* definiálásával a cella rugalmas konfigurálását, valamint *általánoscélú ütközésvizsgáló könyvtár PQP*[3], *FCL*[4] felhasználásával azok ütközésének ellenőrzését.

Pályatervezés során kiemelt fontosságú a felhasznált mozdulatok ütközésvizsgálati eljárása, így ezek közül a *mintavételezés alapú* valamint, folytonos, formális garanciát nyújtó *Conservative Advancement (CA)* módszerek áttekintése, megvalósítása és teljesítményük mérése, ezzel kapcsolatban biztonsági távolság szükségessége és alkalmazása is leírásra került. Ezt követően a mai nap is aktívan kutatott randomizált pályatervező algoritmusok, köztük a *Probabilistic Roadmaps (PRM)*, *multi-query* és Rapidly-exploring Random Trees (RRT), mint *single-query* algoritmus egy-egy alapvető változata került áttekintésre és implementálásra.

Ezeken felül az ipari alkalmazásokban való használathoz az alábbi fejlesztések voltak szükségesek a megfelelő minőségű pályák, elvárt időn belüli megtervezéséhez. Ezek közé tartozik a (1) mozdulatok bejárásának, illetve lekérdezésének sorrend optimalizálása. (2) Conservative Advancement gyakorlati megvalósítása, új hatékonyabb becslő eljárás kidolgozása és a használt feladatsor sorrend optimalizálása. (3) CA módszerének kiterjesztése megfogott objektumokra. (4) PQP könyvtár relatív, abszolút hibahatárainak és a biztonsági távolság kapcsolatának vizsgálata, korrekciója, ez alapján teljesítmény javítása.

Ezekkel a változtatásokkal az implementálásra került könyvtár képes valós ipari igényekre megoldást nyújtani, ahogyan azt a nyomtatott áramkörök megmunkálása vagy a kamerás képfeldolgozás alapján működő kábelvég rendezés bemutatta. Az elkészül programkönyvtár alapot nyújt robotos cellákban való tervezésre, pályák létrehozására, simítására, műveletek megvalósíthatóságának vizsgálatára. A már meglévő funkcionalitás könnyen bővíthető új, akár kutatási munkák során felmerülő eljárásokkal, robot, cella és megfogó megvalósításokkal.

[1] https://www.universal-robots.com/hu/termékek
[2] https://robotiq.com/products/2f85-140-adaptive-robot-gripper
[3] https://github.com/GammaUNC/PQP
[4] https://github.com/flexible-collision-library/fcl

Jövőbeli munka:

- Valós tervezési feladatok megoldása.
- A dolgozatban említett kezdetleges formában már létező kétirányú RRT több cél csomóponttal és célorientált konfiguráció generálást használó RRT kidolgozása és tesztelése.
- Olyan alkalmazások megismerése amelyeknek feltétele az inverz kinematika alkalmazása.
- Az UR vezérlő által használt lekerekített mozgások vizsgálata, megvalósítása.
- A cellában elhelyezett elemek állapotváltozásának kezelése a tervezés során. Cellában lévő elemek optimális elhelyezése robot ciklusidejének szempontjából.

Köszönetnyilvánítás

Ezúton is szeretném megköszönni Kovács András témavezetőm munkáját, folyamatos támogatása nélkül ez a dolgozat nem jött volna létre, valamint, hogy a SZTAKI-ban végzett munkámat minden téren készségesen segítette.

Továbbá szeretném megköszönni Hullám Gábor tanszéki konzulensem munkáját, Váncza József laborvezetőnek és a Mérnöki és Üzleti Intelligencia Kutatócsoport munkatársainak a lehetőséget és hasznos tanácsokat, Németh Balázsnak a könyvtár fejlesztésében végzett munkáját, valamint Tipary Bencének a modellezéssel, esettanulmánnyal és az elkészült alkalmazással kapcsolatos észrevételeit, segítségét.

A kutatást az "Ipar 4.0 kutatási és innovációs kiválósági központ" című GINOP-2.3.2-15-2016-00002 és a "Kutatások az ipari digitalizáció által nyújtott potenciál minőségi kiaknázására" című ED_18-2-2018-0006 támogatások tették lehetővé.

Ábrák jegyzéke

Táblázatok jegyzéke

Irodalomjegyzék

[1] R. Bohlin – L. E. Kavraki: Path planning using lazy PRM. In *Proceedings of ICRA 2001, IEEE International Conference on Robotics and Automation* (konferenciaanyag), 1. köt. 2000, 521–528. p.

[2] K. Cao – Q. Cheng – S. Gao – Y. Chen – C. Chen: Improved prm for path planning in narrow passages. In *2019 IEEE International Conference on Mechatronics and Automation (ICMA)* (konferenciaanyag). 2019. 08, 45–50. p.

[3] J. Eyre – C. Freeman: Immersive applications of industrial digital twins. In *IEEE Transactions on Robotics and Automation, London* (konferenciaanyag). 2018.

[4] KUKA Systems GmbH: Matrix production: an example for industrie 4.0. `https://www.kuka.com/en-de/industries/solutions-database/2016/10/matrix-production`, 2016. 10.

[5] KUKA Systems GmbH: Experience matrix production live in the kuka smartproduction center. `https://www.kuka.com/en-de/press/news/2018/02/smartproductioncenter_opening`, 2018. 02.

[6] Gitae Kang – Yong Bum Kim – Young Hun Lee – Hyun Seok Oh – Won Suk You – Hyouk Ryeol Choi: Sampling-based motion planning of manipulator with goal-oriented sampling. *Intelligent Service Robotics*, 12. évf. (2019. 07) 3. sz., 265–273. p.

[7] L. E. Kavraki – P. Svestka – J. C. Latombe – M. H. Overmars: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12. évf. (1996) 4. sz., 566–580. p.

[8] Zs. Kemény – R. Beregi – J. Nacsa – Cs. Kardos – D. Horváth: Human–robot collaboration in the MTA SZTAKI learning factory facility at győr. *Procedia Manufacturing*, 23. évf. (2018), 105 – 110. p. ISSN 2351-9789. "Advanced Engineering Education Training for Manufacturing Innovation" 8th CIRP Sponsored Conference on Learning Factories (CLF 2018).

[9] E. Larsen – S. Gottschalk – M. C. Lin – D. Manocha: Fast proximity queries with swept sphere volumes. In *Proc. IEEE Int. Conf. Robot. Autom.* (konferenciaanyag). 2000, 3719–3726. p.

[10] Steven M. Lavalle – James J. Kuffner: Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions* (konferenciaanyag). 2000, 293–308. p.

[11] Nikolaos Nikolakis – Vasilis Maratos – Sotiris Makris: A cyber physical system (CPS) approach for safe human-robot collaboration in a shared workplace. *Robotics and Computer-Integrated Manufacturing*, 56. évf. (2019), 233 – 243. p.

[12] B. Németh – A. Kovács: Robot path planning over a sequence of points specified in task space. In *ICAPS Workshop on Planning and Robotics (PlanRob2020), Nancy, France, working paper* (konferenciaanyag). 2020.

[13] Alex Owen-Hill: How it works: RoboDK's New PRM motion planner. `https://robodk.com/blog/prm-motion-planner/`, 2019. 05.

[14] J. Pan – S. Chitta – D. Manocha: FCL: A general purpose library for collision and proximity queries. In *IEEE International Conference on Robotics and Automation* (konferenciaanyag). 2012, 3859–3866. p.

[15] Aaron Parrott – Dr. Lane Warshaw: Industry 4.0 and the digital twin, manufacturing meets its match. Jelentés, 2017, Deloitte University Press. `https://www2.deloitte.com/content/dam/insights/us/articles/3833_Industry4-0_digital-twin-technology/DUP_Industry-4.0_digital-twin-technology.pdf`.

[16] S. Pellegrinelli – A. Orlandini – N. Pedrocchi – A. Umbrico – T. Tolio: Motion planning and scheduling for human and industrial-robot collaboration. *CIRP Annals*, 66. évf. (2017) 1. sz., 1 – 4. p.

[17] F. Schwarzer – M. Saha – J. C. Latombe: Exact collision checking of robot paths. In *Algorithmic Foundations of Robotics V*. 2004, Springer, 25–41. p.

[18] B. Siciliano – O. Khatib: *Springer Handbook of Robotics*. 2016, Springer International Publishing, 122 – 126, 28 – 32. p. ISBN 978-3-319-32550-7.

[19] Zs. Szabó – Cs. Budai – L. Kovács – Gy. Lipovszki: *Robotmechanizmusok*. 2014, BME MOGI. ISBN 978-963-313-170-1.

[20] W. Xinyu – L. Xiaojuan – G. Yong – S. Jiadong – W. Rui: Bidirectional potential guided RRT* for motion planning. *IEEE Access*, 7. évf. (2019), 95046–95057. p.

[21] L. Zahorán – A. Kovács: Efficient collision detection for path planning for industrial robots. In *6th Student Computer Science Research Conference (StuCoSReC 2019), Koper, Slovenia* (konferenciaanyag). 2019.