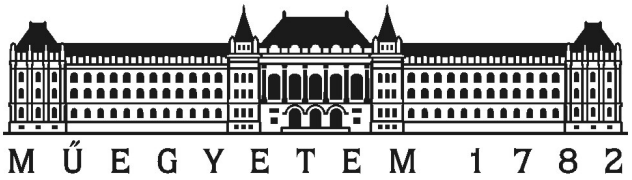


Mért pontfelhők alapján rekonstruált, 3D-s számítógépes modellek tökéletesítése

TDK dolgozat

Kovács István
BME TTK, Matematika MSc, I. évfolyam

Témavezető:
Dr. Várady Tamás
tudományos főmunkatárs
BME IIT

2013

Tartalomjegyzék

Kivonat

A digitális alakzatrekonstrukció (vagy mérnöki visszafejtés) célja, hogy mért adatok alapján elkészítsük egy tárgy számítógépes modelljét. Ezt a technológiát számos területen alkalmazzák, többek között régi alkatrészek újragyártására, a gyártási minőség ellenőrzésére, személyre szabott gyógyászati segédeszközök (pl. protézisek, fogsorok, hallókészülékek) előállítására, illetve a kulturális örökség (pl. épületek, szobrok) digitális megőrzésére.

A legtöbb 3D-s rekonstrukciós eljárás egy ideális modell jó közelítését állítja elő. A pontatlanság részben a nagyméretű zajos adatokból, másrészt a határoló felületek létrehozásakor alkalmazott numerikus módszerek pontatlanságából adódik. Jelen dolgozat célja, hogy ezeket kiküszöböljük, és egy mérnöki értelemben „tökéletes" modellt készítsünk CAD/CAM rendszerek számára.

A külön-külön megillesztett felületek általában „jól" közelítik az adatpontokat, de egy tökéletes modell létrehozása megköveteli, hogy felismerjük a leginkább valószínűsíthető geometriai kényszereket, majd ezek figyelembevételével illesszük újra az érintett felületcsoportokat. Geometriai kényszerek alkalmazása nélkül a lapok nem lesznek tökéletesen párhuzamosak, a tengelyek nem lesznek merőlegesek, a körök nem lesznek koncentrikusak, a numerikus értékek pontatlanok lesznek, és így tovább.

A mérnöki kényszer-egyenletek figyelembevételével történő csoportos felületillesztés matematikailag jól megfogalmazható. Jelen projekt egy korábban publikált eljárásra épít [2], amelyet egy kereskedelmi forgalomban kapható szoftver rendszerben (Geomagic Studio [1]) is alkalmaznak. A feladat igen nehéz a zajos adatok és a kényszerekből álló rendszer komplexitása miatt. Nagyon sok és sokféle felület fordulhat elő, és a kényszerekből álló nagy nemlineáris egyenletrendszert hatékonyan és megbízhatóan kell megoldani. Előfordulhat, hogy a kényszerek ellentmondanak egymásnak, és egy prioritási sorrendet felállítva el kell döntenünk, hogy melyeket kívánjuk érvényesíteni.

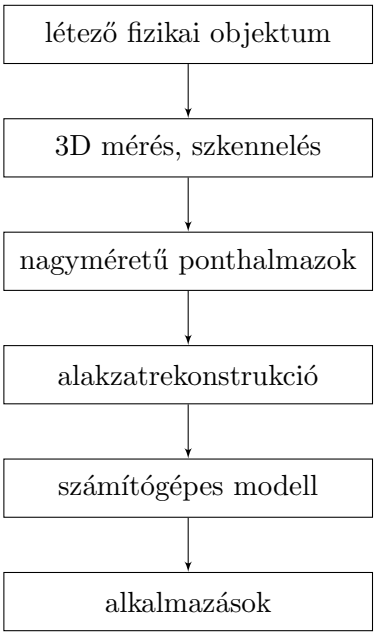
Jelen kutatási projekt két új vonatkozásban bővíti a meglévő technológiát. Egyrészt a valószínűsíthető kényszereket nem a felhasználóra bízza, hanem automatikusan állít fel hipotéziseket. Másrészt a szomszédos felületekre, vagy felületcsoportokra vonatkozó „lokális" kényszereken túl, „globális" - az egész alkatrészre vonatkozó - kényszerek felállítását is megcélozza. Például, meghatározzuk egy teljes alkatrészre legjobban illeszkedő, méretező rács koordináta tengelyeit és rácsállandóját, vagy meghatározzuk egy alkatrész legjobb - teljes vagy részleges - szimmetria tengelyeit.

A dolgozatban bemutatjuk az általános kényszeres illesztés alapjait, majd részletesen ismertetjük a hipotézisek felállításának és az optimális, globális tulajdonságok meghatározásának algoritmusait. Ezeket az algoritmusokat implementáltuk egy fejlesztés alatt álló tesztkörnyezetben, egyelőre sík-kontúrok kényszerekkel történő illesztésére, azonban a matematikai apparátus könnyen kiterjeszthető 3D-re. A vonatkozó elméleti és numerikus problémákat számos tesztpélda segítségével illusztráljuk.

1. Bevezetés

1.1. Digitális alakzatrekonstrukció

A ma már egyre szélesebb körben elterjedt 3D-s szkennerek segítségével elérhetővé vált, hogy fizikai tárgyakat lemérve elkészítsük azok számítógépes modelljét. Ezt a folyamatot digitális alakzatrekonstrukciónak, vagy más néven mérnöki visszafejtésnek nevezzük [15]. A módszernek számos mérnöki alkalmazása ismert többek között régi alkatrészek újragyártása, gyártás minőség-ellenőrzés. Orvosi alkalmazások is ismertek, például személyre szabott gyógyászati segédeszközök (protézisek, hallókészülékek stb.) gyártásában. Szintén fontos alkalmazás a kulturális örökség (épületek, szobrok stb.) digitális megőrzése.



1. ábra. A digitális alakzatrekonstrukció folyamata

Mérnöki alkalmazások esetén a szkennelésnek, és a rekonstrukciónak nagyon pontosnak kell lennie. Ebben az esetben digitális alakzatrekonstrukció folyamata a következő technikai fázisokra tagolható:

1. 3D-s mérés (szkennelés)
2. ponthalmazok szűrése és egyesítése
3. háromszögháló létrehozása
4. háromszögháló egyszerűsítése és javítása
5. szegmentálás (tartományokra bontás)
6. tartományok osztályozása
7. felületek illesztése
8. összekötő felületek (pl. lekerekítések) illesztése
9. *felületek tökéletesítése (kényszerek alkalmazása, simítás)*
10. exportálás CAD/CAM rendszerekbe

1.1.1. Szegmentálás

A szegmentálás célja, hogy a háromszöghálót — különböző lokális geometriai jellemzők alapján — különálló tartományokra osszuk. Először le kell választanunk az erős görbülettel bíró *elválasztó* tartományokat, a megmaradó részeket *elsődleges* tartományoknak hívjuk. Az elsődleges tartományok struktúrája megfelel a lapok struktúrájának a végső CAD modellben. A szakirodalom a szegmentációt az alakzatrekonstrukció legkritikusabb, a végső modell minőségét alapvetően meghatározó fázisnak tekinti [15].

1.1.2. Klasszifikáció

A tartományok *osztályozása* szintén fontos lépés, ugyanis célunk az eredeti alkatrész felületeinek lehető legtökéletesebb rekonstrukciója. Ez nem mindig egyszerű, hiszen zajos adataink vannak és csak annyit tudunk, hogy egy ismeretlen matematikai felület bizonyos részhalmaza jól közelíti az adatpontjainkat. Az osztályozás általában több lépésben történik: hipotéziseket állítunk fel, és az osztályozást addig folytatjuk, amíg a tartomány ki nem elégíti a hipotézisben kijelölt felülettípus geometriai tulajdonságait. Egyszerű felülettípusokkal indulunk, majd egyre összetettebb felületekkel próbálkozunk:

1. Sík
2a. Kihúzott felület (henger; általános profil) [extrusion]
2b. Ferdén kihúzott felület (kúp; általános profil) [drafted extrusion]
3. Forgásfelület (gömb; tórusz; általános profil)
4. Állandó profilú söpört felület [sweeping]
5. Változó profilú söpört felület [lofting]
6. Szabadformájú felület

1.1.3. Felületek illesztése

A klasszifikáció után minden egyes régióhoz meghatároztuk, hogy az milyen típusú matematikai felülettel közelíthető a legjobban. Célunk ezekhez a régiókhoz tartozó pontfelhőkhöz megtalálni a legjobban illeszkedő felületet. Ezt gyakran már a klasszifikáció során megkapjuk, hiszen az egyik lehetséges módszer, hogy megpróbáljuk approximálni a felületet egy adott típussal, és az illesztés végén döntünk, hogy az megfelelő-e.

Általában nem egy, hanem több felületet kell illesztenünk egyidejűleg. Ha a felületeket külön-külön illesztjük, az illesztés nem lesz pontos (lásd 1.3. fejezet). Legyenek az illesztendő felületek $\{s_i\}$, és a hozzájuk tartozó ponthalmazok $\{p_{ij}\}$. A feladatunk minimalizálni az egyes illesztendő felületcsoport négyzetes távolságát a hozzájuk tartozó ponthalmazoktól, amit egy minimalizáló f_i függvénnyel adunk meg. Az $\mathbf{x}$ vektor tartalmazza az illesztendő s_i objektumok paramétereit. A kényszerek nélküli (külön-külön) illesztés tehát az

$$f_i(\mathbf{x}) = \sum_j d(p_{ij}, s_i)^2$$

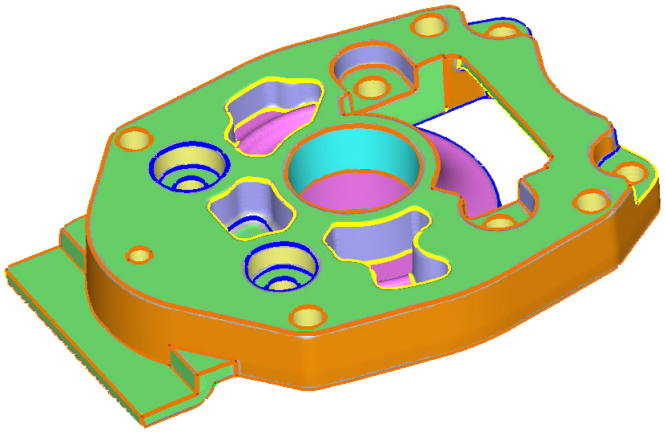
$$f_i(\mathbf{x}) \to \min$$

formában írható. A feladat egyszerűbb felületek – például sík, gömb, henger – esetén egyszerű sajátértékproblémához vezet [5], [14], de bonyolultabb felülettípusokra – mint például B-Spline felületek – is kidolgoztak hatékony módszereket [13].

A továbbiakban feltételezzük, hogy a szegmentáció és klasszifikáció már lezajlott, és a továbbiakban adottak elsődleges tartományok és a hozzájuk tartozó felülettípusok.

1.2. Geometriai kényszerek

A *geometriai kényszerek (constraints)* megadják, hogy különböző határoló és szerkesztőelemek hogyan kapcsolódnak egymáshoz. A kényszerek kulcsfontosságú szerepet töltenek be a mérnöki tervezés során, hiszen ezek határozzák meg a párhuzamosságot és merőlegességet, forgástengelyek illeszkedését, számszerű értékek pontosságát, az alakzatok szimmetriáit, és még sok minden mást.



2. ábra. Egy mérnöki alkatrészt számos kényszer jellemez

A tervezés során leggyakrabban előforduló, egyszerű 2D-s és 3D-s objektumok (egyenesek, körök, síkok, hengerek, kúpok, eltolásos és forgásfelületek) közötti *lokális* kényszerek:

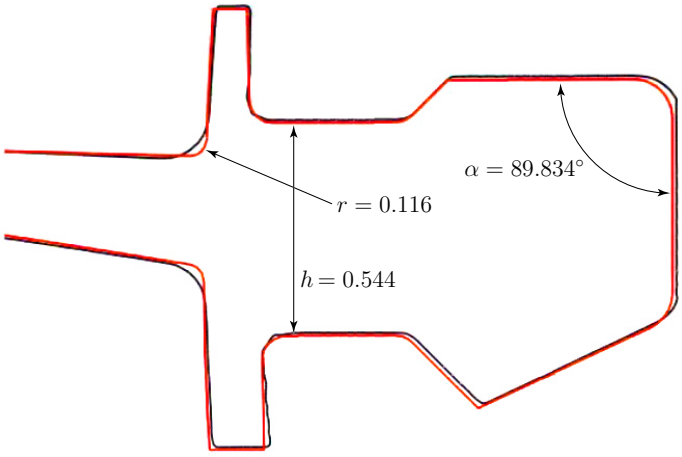
- merőleges/párhuzamos görbék és felületek
- koncentrikus görbék és felületek
- egymást érintő görbék és felületek
- adott ponton áthaladó görbék és felületek
- kerekített értékek
- rögzített értékek

Kényszerek érvényesülhetnek akár nagyobb felületcsoportok között is. Ezeket *globális* kényszereknek hívjuk, a legfontosabbak a következők:

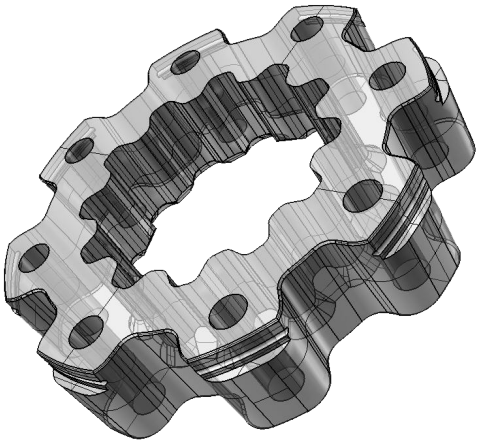
- közös eltolásos irány
- közös forgástengely
- globális szerkezetű rácsra való illeszkedés
- globális tengelyes szimmetria
- globális forgásszimmetria

A kényszerrendszerek legtöbbször nagyon összetettek, és gyakran nem csak szomszédos felületekre, hanem a tárgy számos felületére vonatkoznak, mint például rácsra illeszkedés vagy szimmetriák.

A kényszerekről a szkennelés során nem feltétlenül rendelkezünk információkkal, így a rekonstrukció során magunknak kell ezeket megadni, vagy valamilyen automatikus eljárás során kell ezeket felismerni. A fizikai objektum, és a mérés pontatlanságai miatt ez egy nehéz feladat, hiszen a kényszerek csak valamilyen toleranción belül teljesülnek, pontosan



3. ábra. A mért pontokból származtatott geometriai elemek eltérnek az absztrakt modelltől – pl. közelítő párhuzamos, illetve merőleges élek, pontatlan sugár, stb.



4. ábra. Gyakran találkozhatunk globális forgásszimmetriákkal is

nem. Amennyiben ezeket sikeresen felismertük, vagy explicite megadtuk, a ponthalmazokhoz a matematikai felületeket a kényszerek teljesülése mellett kell egyszerre illesztenünk, így a felületeket és a kényszereket egy rendszerként kell kezelnünk. Ezt a folyamatot *kényszeres illesztésnek* nevezzük.

A kényszeres illesztés matematikailag könnyen formalizálható. Legyen a minimalizálandó függvény az előző részben használt jelölésekkel

$$f(\mathbf{x}) = \sum_i \alpha_i \sum_j d(p_{ij}, s_i)^2,$$

ahol α_i az egyes felületekhez rendelt súly, ami függhet például a felszíntől. A cél tehát f minimalizálása a megadott kényszerek teljesülése mellett. Fontos hogy a rendszer kezelni tudja azt, ha a kényszerek egymásnak ellentmondanak, vagy a rendszer túlhatározott. Sok numerikus módszer ezekben az esetekben csődöt mond, így külön figyelmet kell fordítanunk ezekre az esetekre is.

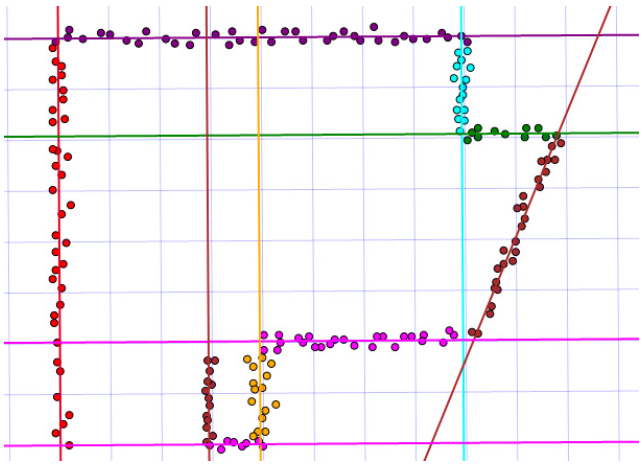
A feladatra több módszert is kidolgoztak [17], [12], a dolgozatban ismertetett módsze-

rek egy módosított Newton-iterációt használó algoritmuson alapulnak, amit a 2. fejezetben részletezünk [2].

1.3. Kényszerek felismerése - a feladat ismertetése

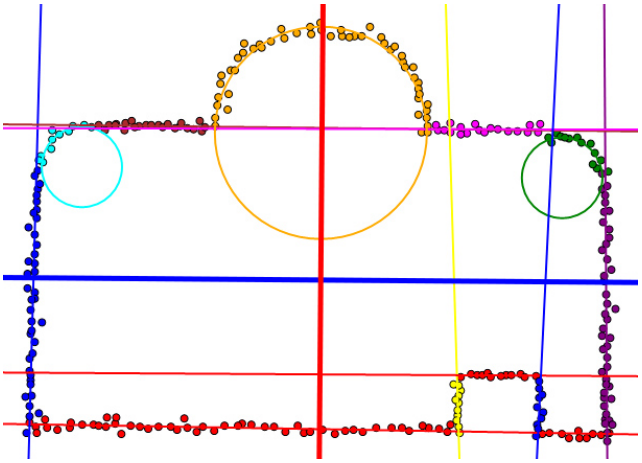
A kényszerek kulcsfontosságú részét képzik a CAD/CAM modelleknek. Ezeket a jelenlegi digitális alakzatrekonstrukciós rendszerekben általában a felhasználóknak kézzel kell megadniuk. Valójában ezek közül nagyon sok automatikusan is felismerhető, azaz felállíthatunk hipotéziseket a lehetséges kényszerekről és minősíthetjük azokat. A 3. fejezetben egy megoldást adunk a kényszerek automatikus felismerésére, osztályozására, amihez az előző részben említett eljárást használjuk fel.

Az 5. fejezetben a „lokális", egyszerűbb felületek közötti kényszereken túl, „globális" kényszerek automatikus felismeréséről lesz szó. Fontos globális tulajdonság a rácsra illeszkedés. Ennek lehet oka egyszerűen, hogy az objektumok mérőszámai (valamilyen mértékegységben) egészek, de előfordulhat, hogy a tervezés során az alkatrész fő pontjai egy nagyobb előre megadott szerkesztő rácshoz illeszkednek. A feladat fő nehézsége, hogy az objektum nagy része illeszkedik a rácsra, de bizonyos lokális részek nem. Ez utóbbiakat fel kell ismernünk, és figyelmen kívül kell hagynunk. Az 4. fejezetben megoldást adunk arra, hogyan lehet rácsra illeszkedést felismerni, ami magában foglalja a rács orientációjának, rácsállandójának meghatározását, és a rácshoz való kényszeres illesztést, amire az 5. ábrán láthatunk példát. Természetesen előfordulhat, hogy több (más orientációjú, rácsállandójú) rács is megtalálható a modellen. Ekkor a rendszer több hipotézist állít fel, azokat minősíti és rangsorolja.



5. ábra. Felismert rács

A mérnöki alkatrészekben nagyon gyakoriak a globális szimmetriák, mint például tengelyes szimmetria vagy forgásszimmetria. Gyakran az is előfordul, hogy az alkatrész csupán részleges szimmetriát tartalmaz (lásd 2., 16. ábra). A jelen kutatásunk tengelyes szimmetriák vizsgálatára koncentrál. Algoritmust adunk arra, hogyan lehet az objektum tengelyes szimmetriáit felismerni, a felismert szimmetriatengelyeket osztályozni (lásd 6. ábra).



6. ábra. Felismert főbb szimmetriatengelyek

A módszerek demonstrálásához készült egy 2D-s tesztkörnyezet (6. fejezet), amiben a kidolgozott algoritmusokat implementáltuk; a program az internetről is elérhető. Az algoritmusokat használni lehet digitális alakzatrekonstrukcióra, elsősorban síkbeli profilgörbékre és összetett kontúrokra, de ezek a matematikai módszerek könnyen kiterjeszthetők 3D-s objektumokra is.

1.4. A kutatás célja és a dolgozat felépítése

A külön-külön megillesztett felületek általában „jól” közelítik az adatpontokat, de egy tökéletes modell létrehozása megköveteli, hogy felismerjük a leginkább valószínűsíthető geometriai kényszereket, majd ezek figyelembevételével illesszük újra az érintett felületcsoportokat. Geometriai kényszerek alkalmazása nélkül a lapok nem lesznek tökéletesen párhuzamosak, a tengelyek nem lesznek merőlegesek, a körök nem lesznek koncentrikusak, a numerikus értékek pontatlanok lesznek, és így tovább. A kutatás célja, hogy a rekonstruált modellen hipotéziseket állítsunk fel a lehetséges kényszerekről automatikusan.

Mielőtt rátérnénk a kifejlesztett új algoritmusokra, a 2. fejezetben ismertetünk egy korábban publikált eljárást. Ez képezi jelen projekt alapját.

A 3. fejezetben algoritmust adunk egyszerűbb, lokális geometriai kényszerek felismerésére, hipotézisek felállítására és – ezekkel összhangban – felületcsoportok illesztésére. A 4. fejezetben algoritmust adunk optimális globális rácsok meghatározására és a vonatkozó kényszerek érvényesítésére. Az 5. fejezetben pedig megoldást adunk arra, hogy felismerjünk és osztályozzunk globális szimmetriákat.

Az algoritmusok szemléltetésére készítettem egy Silverlight alapú tesztkörnyezetet, amit a 6. fejezetben ismertetünk.

2. A kényszeres illesztés alapjai

Miután az elsődleges felületek típusát meghatároztuk, és azokat külön-külön megillesztettük, szükségünk van az alakzatok közti kényszerekre, hogy egy pontos (mérnöki alkalmazásokra használható) CAD modellt kapjunk. Mivel mért adatokkal dolgozunk, a kényszerek csak bizonyos tolerancián belül teljesülnek, így ezeket vagy explicite meg kell adnunk, vagy egy rendszernek automatikusan fel kell ismernie. Ezután a felületeket újrailleszjtjük, immár a kényszerek érvényesítése mellett. Erre a problémára már több megoldás is született, főként iteratív vagy genetikus algoritmusokkal [17], [12]. Ebben a fejezetben egy, az iparban is jól bevált módszert mutatunk be [2], amire a dolgozat későbbi részeiben is támaszkodni fogunk.

A kényszerek igen sokfélék lehetnek, hathatnak egy (pl. kör sugara fix) vagy több objektumra (pl. két egyenes merőleges) is. Gondolnunk kell arra is, hogy az előre megadott kényszerek *ellentmondásosak* lehetnek, például egyszerre akarjuk, hogy két egyenes merőleges és párhuzamos legyen. Ennél persze sokkal összetettebb problémák is előfordulhatnak, ahol számos kényszer ellentmond egymásnak. Ekkor ezt az algoritmusnak fel kell ismernie, és ki kell választania (valamilyen prioritási sorrend szerint), hogy melyik kényszerek teljesüljenek. Hasonlóan fontos, hogy az algoritmus felismerje, ha egyes kényszerek már *következnek* az előzőekből, így azok teljesítésével már ne foglalkozzon, és erről tájékoztassa a felhasználót is. A bemutatott algoritmus nagyszerű megoldást nyújt ezekre a feladatokra is.

A kényszeres illesztés során kétféle objektumtípust különböztetünk meg. Az elsők az ún. *elsődleges* (primary) objektumok, azaz görbék és felületek, amelyek ténylegesen illeszkednek a mért adatpontokhoz, és a *segéd* (auxiliary) objektumok amelyek nem illeszkednek a mért adatpontokhoz, de kényszerek hathatnak köztük és az elsődleges objektumok között.A segédobjektumok egyszerűsítik és stabilizálják az egyenletrendszert, és lehetővé teszik bonyolultabb kényszerek definiálását is. Az objektumok n paramétere egy n dimenziós vektorral van megadva, a kényszerek ezen vektorok koordinátái között felírt egyenletek, vagy egyenletrendszerek.

2.1. Egy példa

Vizsgáljuk meg a feladatot először egy példán keresztül [16]. Nézzük a 7(a). ábrán látható forgásfelület profilját. Ez egymás érintő körökből áll, ahogy azt a 7(b). ábrán láthatjuk. Tegyük fel, hogy felismertük a forgásfelületet, és megkaptuk a profilgörbe diszkrét pontjait szegmentálva, azaz minden egyes c_i körhöz tartozik egy $\{p_{ij}\}, j \in \{1, ..., n_i\}$ ponthalmaz. Feladatunk tehát minimalizálni a körök pontoktól vett négyzetes távolságát, azon kényszerek mellett, hogy a körök érintik egymást.

A kör egyenlete felírható az alábbi alakban

$$F(x, y) = A(x^2 + y^2) + Bx + Cy + D = 0.$$

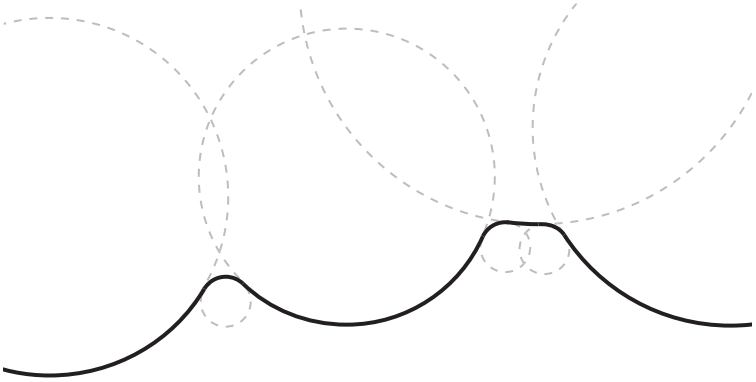
Ezekkel a paraméterekkel szükségünk van még egy normalizálási feltételre is, miszerint a gradiens egységnyi a körön. Azaz

$$\left| \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y} \right) \right| = 1$$

amit kifejtve

$$(2Ax + B)^2 + (2Ay + C)^2 = 1$$

(a)



(b)

7. ábra. (a) Forgásfelület; (b) alsó profilgörbe.

amiből kivonva $4AF(x,y) = 0$-t, a

$$B^2 + C^2 - 4AD = 1$$

normalizálási kényszert kapjuk.

Legyenek c_i paraméterei (A_i, B_i, C_i, D_i). Ekkor a minimalizálandó négyzetes távolság

$$f(\mathbf{x}) = \sum_{i,j} d_{i,j}^2 = \frac{1}{n_i} \sum_j (A_i(x_{ij}^2 + y_{ij}^2) + B_i x_{ij} + C_i y_{ij} + D_i)^2.$$

A tangenciális kényszerek a körök közt (pl. 1. és 2. körre):

$$2A_1x + B_1 = -(2A_2x + B_2)$$

$$2A_1y + C_1 = -(2A_2y + C_2)$$

ebből

$$(2A_1x + B_1 + 2A_2x + B_2)^2 + (2A_1y + C_1 + 2A_2y + C_2)^2 = 0$$

az egyenlet tovább egyszerűsíthető a

$$2A_2D_1 + 2A_1D_2 - B_1B_2 - C_1C_2 \pm 1 = 0$$

alakra.

A kényszereink tehát:

- normalizálási kényszerek ($B_i^2 + C_i^2 - 4A_iD_i = 1$)
- tangenciális kényszerek (szomszédos körökre $2A_jD_i + 2A_iD_j - B_iB_j - C_iC_j \pm 1 = 0$)

Azaz a feladatunk $f(\mathbf{x})$ minimalizálása a kényszeregyenletek fennállása mellett. A következő fejezetben ismertetjük, hogyan kezelhető mindez általános esetben.

2.2. A feladat numerikus megoldása

Ebben az alfejezetben ismertetjük a feladatot megoldó numerikus módszert, amiről részletes leírást találhatunk a szakirodalomban [2].

Legyenek az illesztendő objektumok (felületek, egyenesek, körök stb.) $\{s_i\}$, és a hozzájuk tartozó ponthalmazok $\{p_{ij}\}$. A feladatunk minimalizálni az egyes illesztendő felületek négyzetes távolságát a hozzájuk tartozó ponthalmazoktól, amit egy globális minimalizáló f függvénnyel adunk meg. Az $\mathbf{x}$ vektor tartalmazza az összes felület összes meghatározandó paramétereit, ez az ismeretlen.

$$f(\mathbf{x}) = \sum_i \alpha_i \sum_j d(p_{ij}, s_i)^2$$

$d(p, s)$ jelölje a p pont s-től való távolságát, α_i pedig az objektumokhoz rendelt súlyt, ami függhet például pontok számától, felületek területétől, vagy a felhasználó által megadott paraméterektől is.

Adottak továbbá a kényszeregyenletek $\{c_i\}$, amelyek segítségével felírható egy globális egyenletrendszer

$$\mathbf{c}(\mathbf{x}) = 0.$$

Így tehát a feladatunk f minimalizálása a $\mathbf{c} = 0$ feltétel mellett.

Az ilyen jellegű minimalizációs feladatokat tipikusan Lagrange-multiplikátorokkal szokták megoldani, viszont ez a módszer nem használható, amennyiben a kényszerek nem függetlenek. Később genetikus algoritmusokkal próbálták kiküszöbölni ki ezt a problémát, ezeknél azonban a számításigény jelentős [17]. Most egy olyan iterációs módszert ismertetünk, amelyben ha a kényszerek ellentmondanak egymásnak, prioritási sorrenddel dönthetünk a teljesülésükről, és a folyamat során az is kiderül, ha a rendszer túlhatározott.

Legyen $\mathbf{c}(\mathbf{x}) = (c_1(\mathbf{x}), ..., c_k(\mathbf{x}))$ ahol a c_1 a legnagyobb c_k a legkisebb prioritású kényszer egyenlete. Feladatunk $\mathbf{c}(\mathbf{x}) = 0$ megoldása $f(\mathbf{x})$ minimalizálása mellett. Tegyük fel, hogy f és $\mathbf{c}$ elegendően sima. A megoldás módosított Newton-iterációval történik. Minden lépésben $\mathbf{c}$-t lineárisan, f-et másodfokúan közelítjük, és kiszámolunk egy $\mathbf{d}$-t amivel az éppen adott x_i paramétervektorokból továbblépünk. Ehhez először vizsgáljuk meg f és $\mathbf{c}$ Taylor sorát $\mathbf{x}_0$ körül.

$$\mathbf{c}(\mathbf{x}_0 + \mathbf{d}) \approx \mathbf{c}(\mathbf{x}_0) + \mathbf{c}'(\mathbf{x}_0)\mathbf{d} \tag{1}$$

$$f(\mathbf{x}_0 + \mathbf{d}) \approx f(\mathbf{x}_0) + f'(\mathbf{x}_0)\mathbf{d} + \frac{1}{2}\mathbf{d}^T f''(\mathbf{x}_0)\mathbf{d} \tag{2}$$

Ezek segítségével a feladat közelítőleg a következő formában írható:

$$C\tilde{\mathbf{d}} = 0 \tag{3}$$

$$\tilde{\mathbf{d}}^T A\tilde{\mathbf{d}} \text{ minimális,} \tag{4}$$

ahol $\tilde{\mathbf{d}} = (d_1, ..., d_n, 1)$, $C = [\mathbf{c}'(\mathbf{x}_0)|\mathbf{c}(\mathbf{x}_0)]$ és A a következő $(n+1)\times(n+1)$ méretű mátrix:

$$A = \left[\begin{array}{c|c} f''(\mathbf{x}_0) & f'(\mathbf{x}_0) \\ \hline f'(\mathbf{x}_0)^T & 0 \end{array}\right].$$

Annak érdekében, hogy $\tilde{\mathbf{d}}$-ot kiszámítsuk, először redukáljuk egy alacsonyabb dimenziós $\mathbf{d}^*$ vektorrá (3) segítségével úgy, hogy $\mathbf{d}^*$ koordinátái már függetlenek legyenek. Ehhez kiszámítunk egy M mátrixot, amivel $\tilde{\mathbf{d}} = M\mathbf{d}^*$, és $CM = 0$. Ekkor $\mathbf{d}^*$ koordinátáinak száma megadja hány független változóból áll a rendszer. Ekkor már csak a $\mathbf{d}^{*T}A^*\mathbf{d}^*$ kifejezést kell minimalizálnunk (kényszerek nélkül), ahol $A^* = M^TAM$. A kényszer nélküli minimalizáció egy egyszerű lineáris egyenletrendszer megoldására vezet.

Most pedig nézzük meg, hogyan számíthatjuk ki M-et. A módszer nagyon hasonlít a Gauss-eliminációra. Először tegyük fel, hogy a kényszerek egymástól függetlenek, nincs ellentmondás, és nem következik egyik sem a többiből. Később megnézzük, mi történik azokban az esetekben, amikor ez nem teljesül.

A megoldás során a C mátrixban sorról sorra haladunk. Nézzük meg, mi történik az első lépésben. Legyen C_l a legnagyobb abszolút értékű együttható (C_{n+1}-en kívül). Valójában bármelyiket választhatnánk, de numerikus stabilitási okokból érdemes a legnagyobbat. A $C\tilde{\mathbf{d}} = 0$ egyenlet első sora a következő alakban írható:

$$C_1d_1 + C_2d_2 + ... + C_nd_n + C_{n+1} = 0.$$

Ebből átrendezve kapjuk, hogy

$$d_l = -\frac{C_1d_1}{C_l} - ... - \frac{C_{l-1}d_{l-1}}{C_l} - \frac{C_{l+1}d_{l+1}}{C_l} - ... - \frac{C_{n+1}}{C_l}.$$

Ebből megkaphatjuk a $\tilde{\mathbf{d}} = M_1\mathbf{d}_1$-et, ahol

$$M_1 = \left[\begin{array}{ccc|ccc} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ \hline -\frac{C_1}{C_l} & \cdots & -\frac{C_{l-1}}{C_l} & -\frac{C_{l+1}}{C_l} & \cdots & -\frac{C_{n+1}}{C_l} \\ \hline 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 \end{array}\right]$$

ahol a kiemelt sor az l., és

$$\mathbf{d}_1 = (d_1, ..., d_{l-1}, d_{l+1}, ..., d_n, 1)$$

ami $\tilde{\mathbf{d}}$-nál eggyel kisebb dimenziójú vektor. Ezek után ezt a lépést iteráljuk, a második lépésben CM_1-mátrixszal és $\mathbf{d}_1$-el, ahol

$$(CM_1)\mathbf{d}_1 \;=\; 0.$$

Végeredményképpen $M = M_1M_2....M_j$, ahol j a független változók száma.

A lépések során előfordulhat, hogy az eliminációs lépés nem végezhető el, azaz hogy az m-edik lépésben $C_l = 0$. Mivel C_l a legnagyobb abszolút értékű, a többi együttható is 0, C_{n+1}-et kivéve. Ha C_{n+1} is 0, akkor az éppen aktuális kényszer *következik* az előzőekből, így ezt a sort átugorhatjuk, és M_m-et választhatjuk identitásnak. Ha $C_l = 0$, de $C_{n+1} \neq 0$,

akkor ez azt jelenti, hogy az m-edik kényszeregyenlet *ellentmond* az eddigieknek, ezért ekkor is figyelmen kívül hagyjuk ezt a sort. Megjegyezzük, hogy vannak kényszerek, amelyek több egyenletből állnak, így ebben az esetben az összes aktuális kényszerhez tartozó lépést vissza kell vonnunk, és az összes hozzá tartozó egyenletet figyelmen kívül kell hagynunk.

2.3. Hatékony reprezentáció

Ahogy az előző részben láthattuk a minimalizáláshoz szükségünk van minden egyes iterációs lépésben a minimalizálandó f, azaz a távolságfüggvények súlyozott átlagának első és második deriváltjára. Ez normál Euklideszi távolságfüggvénnyel nehéz, és túlságosan számításigényes feladat. Ehelyett a távolságfüggvényt közelíteni lehet, a távolságfüggvény ún. *hű (faithful)* approximációjával [10]. A távolságfüggvény approximációja hű, ha (i) pontosan akkor nulla, amikor az eredeti távolságfüggvény is az (azaz csak az objektumon), és (ii) a deriváltja az objektumon megegyezik az eredeti távolságfüggvény deriváltjával.

Célszerű olyan hű távolságfüggvényt választani, amivel a számítások oly módon egyszerűsödnek, hogy az iteráció előtt csak egyszer legyen szükség a pontok előfeldolgozására. Ehhez olyan P és S vektorértékű függvényeket választunk, amivel egy p pontra

$$d(p, \mathbf{x}) = S(\mathbf{x})^T P(p) = P(p)^T S(\mathbf{x})$$

adódik az előjeles távolságfüggvényre. Ha ez sikerül, akkor egy objektumra a távolságfüggvény

$$e(\mathbf{x}) = \sum_p d(p, \mathbf{x})^2 = S(\mathbf{x})^T \left(\sum_p P(p) P(p)^T \right) S(\mathbf{x}).$$

Ez azért jó, mert ekkor elegendő az iterációs lépéssorozat előtt az $M = \sum_p P(p)P(p)^T$ mátrixot csak egyszer kiszámolni ami csak az adatpontoktól függ. Az egyes iterációs lépések során csak az objektumok paramétereitől függő $S(\mathbf{x})$-et kell kiszámolnunk. Ebben az esetben a deriváltak számítása a következő:

$$e' = 2S^T M S,$$

$$e'' = 2(S'^T M S' + S^T M S'').$$

Még egyszerűbb a helyzet, amennyiben olyan paraméterekkel jellemezzük az objektumainkat, hogy S-et választhatjuk identitásnak, tehát $S(\mathbf{x}) = (x_1, x_2, ...)$. Ez esetben a deriváltak

$$e' = 2M\mathbf{x},$$

$$e'' = 2M$$

alakot öltik.

Lássunk egy példát. Legyen adott egy egyenes $l_0 x + l_1 y + l_2 = 0$ egyenlettel. Ekkor az előjeles távolságfüggvény hű approximációja a $d(p, \mathbf{x}) = l_0 x_0 + l_1 y_0 + l_2$, tehát $S(\mathbf{x}) = (l_0, l_1, l_2)$ és $P(p) = (x_0, y_0, 1)$.

2.4. Objektumok reprezentációja

Ahogy azt láthattuk az objektumok hatékony reprezentációja fontos feladat. A projekt jelen állásában 2D-s *köröket* és *egyeneseket* illesztünk kényszerek figyelembe vételével. A későbbiekben 3D-s objektumok is részt vesznek majd a kényszeres illesztésben.

Az egyenesnél a paraméterek (l_0, l_1, l_2), ahol az egyenes egyenlete

$$l_0 x + l_1 y + l_2 = 0.$$

A kör reprezentációja már bonyolultabb. Egy p pont körtől való távolsága $||p-o|| - r$. Ez a távolságfüggvény nem túl jó, hiszen egy négyzetgyökös kifejezést tartalmaz, ezért egy hű approximációjával helyettesítjük, azaz

$$\left((p-o)2 - r^2\right)/2r = \left(p^2 - 2\langle p, o\rangle + o^2 - r^2\right)/2r.$$

Ekkor $S(\mathbf{x}) = S(o_x, o_y, r) = (1/2r, -o_x/r, -o_y/r, (o^2 - r^2)/2r)$, $P(p) = (p^2, p_x, p_y, 1)$. Ez még mindig nem megfelelő ahhoz, hogy az egyenletet az előző fejezetbeli formában írhassuk (hiszen $S(\mathbf{x})$ még nem identitás). Ezért a kört inkább a

$$\mathbf{x}' = S(\mathbf{x}') = (\kappa, o'_x, o'_y, A)$$

paraméterekkel jellemezzük, ahol $o' = -2\kappa o$ és $A = (o^2 - r^2)/2r$. Ezekkel a paraméterekkel a deriválás leegyszerűsödik a kívánt alakra.

Hasonló értelmezésű jó reprezentáció felírható 3D-s felületekre is, további példákat a szakirodalomban találhatunk [2].

2.5. Megvalósított kényszerek

A geometriai kényszerek az objektumok paraméterei között felírt egyenletek vagy egyenlet rendszerek, és $c_i(\mathbf{x}) = 0$ formában adottak. Fontos, hogy az egyenletek azt is jellemezni tudják, hogy nem teljesülés esetén mekkora a hiba, azaz például merőlegességi kényszernél $c_i(\mathbf{x})$ monoton függvénye legyen a kilencven foktól való eltérésének. Ahogy a numerikus módszer részleteinél láthattuk, c első deriváltjára is szükség van. Ezt megadhatjuk explicit módon is, de gyakran dolgozunk a derivált másodfokú numerikus közelítésével.

Jelen projektben elsősorban 2D-s kényszerekre összpontosítottunk, és ezeket implementáltuk. Ezek közül néhány:

- két egyenes hajlásszöge fix,
- kör sugara fix,
- kör középpontja fix,
- két kör koncentrikus,
- két kör érinti egymást,
- két kör középpontja azonos,
- egyenes érint egy kört,
- egyenes átmegy a kör középpontján.

2.6. Segédobjektumok

Az itt felsorolt kényszereknél gyakran sokkal bonyolultabbakra is szükségünk lehet, amit nem, vagy csak nagyon nehezen tudnánk egyenletként felírni. Például egy c_1 kört érint egy l egyenes, és ugyanebben a pontban l egy másik c_2 kört is érint. Ez a kényszer megfogalmazható úgy is, hogy l érinti c_1-et egy p pontban, és érinti c_2-t is p-ben. Ekkor p-t, mint két koordinátával megadott pont objektumot nem rendeltük mért pontfelhőhöz, csak azért van a rendszerben, hogy a rendszer egyszerűsödjön. A bonyolultabb kényszereket így egyszerűbb építőkockákból tudjuk összerakni. Az ilyen objektumokat *segéd- (auxiliary) objektumnak* hívjuk. Az általunk használt segédobjektumok:

- egyenes,
- kör,
- *szám*,
- *pont*,
- *pont-vektor*.

Az utóbbi három csak segédobjektumként használható, ezeket nem illesztjük ponthalmazokhoz, a kör és egyenes mind elsődleges, mind segédobjektumként is használható. A segédobjektumok segítségével több bonyolultabb kényszer is felírható. Ezeket alap építőkövekből, egyszerűbb segédobjektumokat is használó kényszerből építjük fel:

- pontok négyzetes távolsága egy szám
- egyenesek szöge egy szám,
- kör sugara egy szám,
- egy egyenes átmegy egy ponton,
- egy kör átmegy egy ponton,
- kör középpontja egy pont,
- egy pont-vektor merőleges egy egyenesre,
- egy pont-vektor merőleges egy körre,
- egy pont-vektor illeszkedik egy pontra,
- pont illeszkedik rácsra.

Lássunk néhány példát. Egy egyszerűbb kényszer, hogy három egyenes átmegy egy közös ponton (lásd 9. ábra). Ez gyakran előfordul, gondoljunk csak egy téglatest sarokpontjaira. Zajos oldalak esetén a külön-külön illesztett egyenesek páronként három eltérő metszéspontot szolgáltatnak. Ezt a kényszert megadhatnánk egyenlettel is, de sokkal egyszerűbb, ha felveszünk egy pontot mint segédobjektumot, és mindhárom egyenesnek megadjuk kényszerként, hogy menjenek át a ponton.

Az előzőnél jóval bonyolultabb példák is hozhatóak. Például három kör egy közös pontban érinti egymást, és ugyanezen a ponton áthalad egy egyenes, és egy negyedik kör. Ekkor a feladat segédobjektumokkal úgy fogalmazható meg, hogy a három kör mind érint egy pont-vektor típusú objektumot (azaz átmennek a ponton, és az orientációs vektor érintővektor), és az egyenes és a kör áthalad a pont-vektor pont részén.

A dolgozat későbbi részében bemutatott összetettebb kényszerekhez (rács, szimmetria) szintén szükségünk lesz segédobjektumokra, mivel ott a kényszerrendszerek nagyon összetettek is lehetnek.

3. Kényszerek automatikus felismerése

A kényszereket a legtöbb digitális alakzatrekonstrukciós rendszerben kézzel kell megadnunk. Valójában nagyon sok egyszerűbb lokális kényszer felismerhető a rekonstrukció során. Amellett, hogy ezzel a felhasználónak segítséget nyújtunk, az automatikusan felállított, egyszerűbb hipotézisek alapján következtetni tudunk bonyolultabb globális kényszerekre is. Ebben a fejezetben bemutatjuk, hogyan ismerhetőek fel, és rangsorolhatók egyszerűbb geometriai kényszerek. Az összetettebb segédobjektumokat is használó kényszerek felismerése nehezebb feladat, a fejezetben erre is megoldást adunk. A 4. és 5. fejezetekben bemutatott módszerek szintén az itt található algoritmusokon alapulnak.

A kényszerek általában csak valamilyen toleranción belül teljesülnek mért adatok zajosságából eredően, például merőleges síkok hajlásszöge $89,834°$ (lásd 3. ábra), azonos sugarú körök sugara egy picit eltérhet, vagy több egyenes nem halad át közös ponton, holott az eredeti CAD objektum tervezésekor ezek feltehetően fennálltak. Ahhoz tehát, hogy hipotéziseket állíthassunk fel a lehetséges kényszerekre, toleranciaszinteket kell beállítanunk. Fontos, hogy a toleranciaszintek (ha nem szögeket jellemeznek) függnek az éppen vizsgált adathalmaz méretétől, ezért gyakran a toleranciák az objektum határoló keretének méretétől való aránnyal vannak megadva. A kutatás során két módszert is megvizsgáltunk a feladat megoldására aszerint, hogy a toleranciaszintek vizsgálata mikor történik.

1. a kényszerek toleranciavizsgálata az iteráció előtt, majd kényszeres illesztés
2. toleranciavizsgálat minden iterációs lépés során, ekkor a kényszereket dinamikusan vesszük fel

Az első módszer esetében a felismert kényszerekkel való illesztés után lehetséges, hogy újabb kényszerek kerülnek toleranción belül, így érdemes lehet a kényszereket újra megvizsgálni és újakat felvenni. A második módszer tehát kezeli ezeket az eseteket is, a hátránya viszont, hogy a rendszert el tudja vinni rossz irányba is, ahogyan arra a későbbiekben láthatunk példákat.

A rendszer kezelésére egy olyan numerikus megoldást adunk, amelyben a 2. fejezetben ismertetett rendszer maga kezeli a toleranciák vizsgálatát, kényszerek felvételét. A módszer alapötlete, hogy az automatikus felismeréshez *az eredeti kényszeregyenleteknek egy módosított változatát használjuk*, ami csak akkor érvényesül, ha az adott kényszer függvénye egy felhasználó által megadott toleranción belül van.

3.1. Módosított kényszeregyenletek

Legyen $c(\mathbf{x}) = 0$ egy egyszerű, két objektum között ható kényszeregyenlet. Legyen ε egy előre megadott toleranciaszint. A kényszer akkor van toleranción belül, ha $|c(\mathbf{x})| < \varepsilon$, ekkor ezt a kényszert felvesszük és a kényszeres illesztés során használjuk, egyébként pedig figyelmen kívül hagyjuk. Ezt a következőképpen valósítjuk meg: legyen

$$s_\varepsilon(x) := \begin{cases} x & \text{ha } |x| < \varepsilon \\ 0 & \text{egyébként.} \end{cases}$$

Ekkor az $x \mapsto s_\varepsilon(c(x))$ függvényt nevezzük a módosított kényszeregyenletnek. Figyeljük meg, hogy ha $c(x)$ toleranción kívül esik, $s_\varepsilon(c(x)) = 0$, és ahogy a 2.2. fejezetben láthattuk, ekkor a rendszer ezt a kényszert figyelmen kívül hagyja (rendszer azt jelzi, hogy a kényszer következik az előzőekből). Minden más általános esetben a függvény megegyezik $c(x)$-el, tehát a rendszer ugyan úgy viselkedik, mintha a $c(x)$ kényszert explicite felvettük volna.

Itt több probléma is felvetődik. Az egyik gond, az együtt illesztés miatt, az iteráció során először minden koordináta kicsit megváltozik, majd egyszerre stabilizálódnak. Ez pedig azt okozza, hogy az egyes kényszerek teljesülésének mértékét maga a kényszeregyenlet nem feltétlenül tudja jól jellemezni, így nincs értelme toleranciáról sem beszélni. Erre megoldás, hogy minden kényszeregyenletet úgy kell megadni, hogy azok jól jellemezzék a valós mennyiséget. Lássunk egy példát.

Legyen c_p a 'pont rajta van az egyenesen' kényszer. Legyenek az egyenes $Ax+By+C = 0$, és a pont (x_0, y_0). Ahogyan a 2.5. fejezetben láthattuk a *pont rajta van az egyenesen* kényszer egyenlete

$$c_{pl}(\mathbf{x}) = Ax_0 + By_0 + C = 0.$$

Ez tényleg pontosan akkor teljesül ha a pont az egyenesen van, viszont a pont-egyenes előjeles távolságának képlete

$$c'_{pl}(\mathbf{x}) = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}.$$

Bár a kényszer fel van véve, miszerint az (A, B) vektor egység normájú, erre nem hagyatkozhatunk a már említett iterációs problémák miatt, tehát a toleranciavizsgálathoz, így az automatikus felismeréshez a c'_{pl} egyenletet kell használnunk.

További probléma amire figyelnünk kell, hogy a numerikus deriválás során gond lehet a nem folytonos egyenletekkel, ezért figyelmet kell fordítanunk arra, hogy a numerikus deriváláskor tolerancián belüli esetekben az eredeti függvény értékeit használjuk a numerikus becslésre.

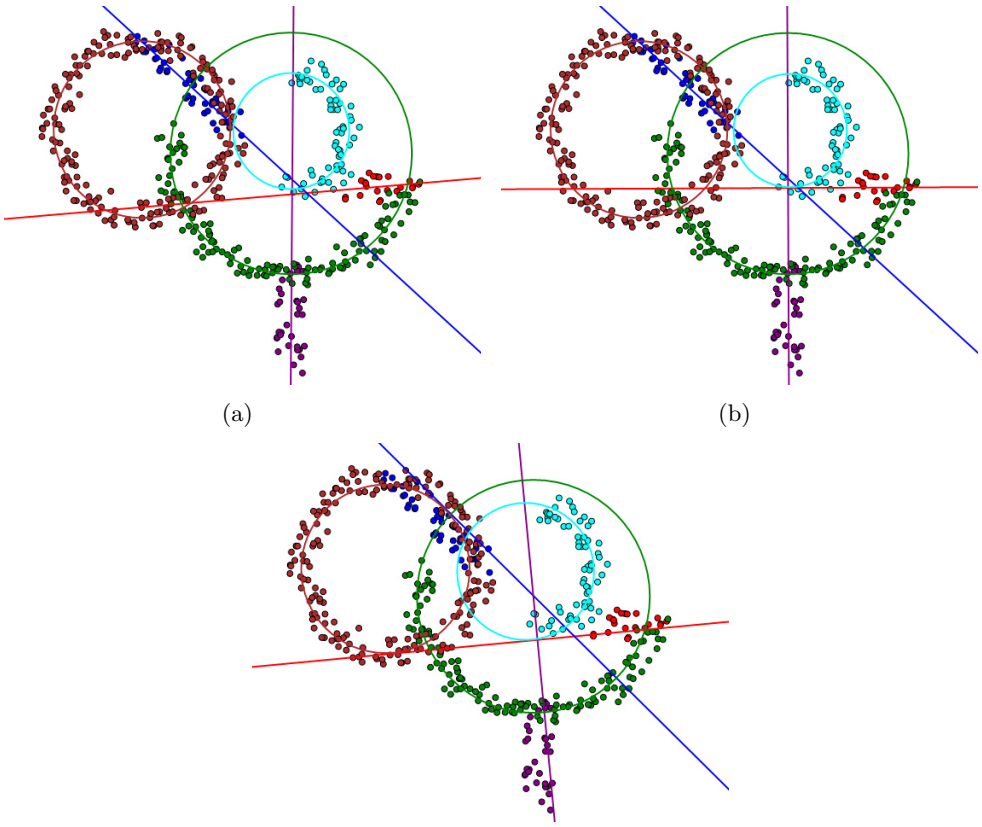
3.2. Automatikus felismerés

Egy naiv módszer a kényszerek felismerésére, hogy a megadott toleranciákkal minden egyes kényszertípushoz felvesszük a módosított változatát a rendszerbe, és elvégezzük a kényszeres illesztést. Ennek hátránya lehet, hogy nagyon sok kényszerből fog állni a rendszer, ami a futásidő kárára megy. Később láthatjuk, hogy bizonyos kényszerek esetén nincs szükség felvenni a kényszereket minden objektumpár esetén.

Legyen az objektumok halmaza $S = \{s_i\}$ és a különböző kényszertípusok c_j, ahol c_j csak a sémáját írja le a kényszeregyenletnek, azt nem, hogy milyen objektumok között hatnak. Minden kényszerhez meg van határozva, hogy az milyen objektumok között hat. Jelöljük $c(s_1, s_2)$-vel azt a konkrét kényszert ahol c az s_1 és s_2 között hat. Így tehát minden egyes c_j-hez felvesszük a $C_j = \{s_{\varepsilon_j}(c_j)(s_1, s_2) : s_1, s_2 \in S, s_1, s_2$ megfelelő típusú$\}$ kényszerhalmazt a rendszerbe. Amennyiben így hagyjuk a rendszert, és elindítjuk az iterációt, a 2. módszert kapjuk, hiszen a kényszerek minden iterációs lépésben újra kiértékelődnek, így a módosított egyenletek esetén a toleranciavizsgálat is újra megtörténik. Az 1. módszerhez a toleranciavizsgálat után a teljesülő kényszereket hozzáadjuk normál kényszeregyenlettel a rendszerhez.

3.3. Összetett kényszerek

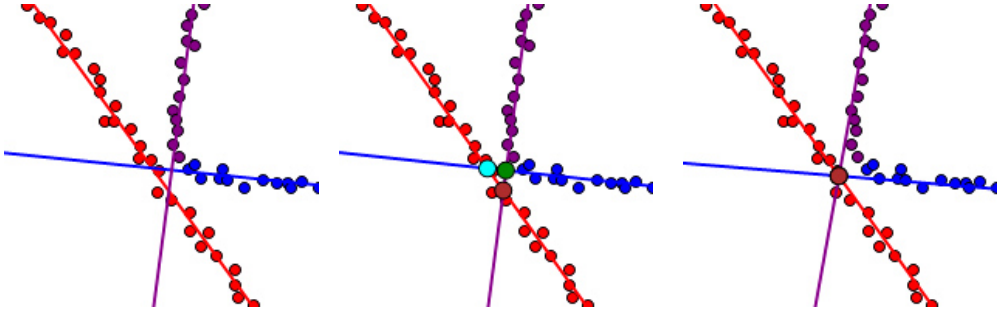
A 2.6. fejezetben bemutattuk, hogy segédobjektumok segítségével összetettebb kényszerek is létrehozhatók. Ezek felismerése nehezebb feladat, hiszen toleranciát vizsgálni csak egyszerű két objektum között ható kényszerek között tudunk, és amíg nem vesszük fel az összetettebb kényszereket, segédobjektumaink sincsenek. Sok összetettebb kényszer felismeréséhez komolyabb apparátus kell (lásd 4. és 5. fejezet), de ezek közül néhányat így is felismerhetünk.



(a) (b)

(c)

8. ábra. Automatikusan felismert kényszerek: (a) kiindulóállapot; toleranciák a (b) illetve (c) esetben, a küszöb: 10°/10° *egyenesek merőlegesek*; küszöb: 10/10 egység *egyenes átmegy a kör középpontján*; küszöb: 15/25 egység *egyenes érinti a kört* kényszerre.



(a) (b) (c)

9. ábra. Három egyenes átmegy egy közös ponton: (a) kiindulóállapot; (b) metszéspontok felvétele segédobjektumos kényszerekkel; és, (c) *pontok egybeesnek* kényszer felismerése és érvényesítése.

Térjünk vissza a már korábban ismertetett *három egyenes egy ponton megy keresztül* kényszerre. Természetesen a közös pont definíciója is egy toleranciától függ. Ehhez a kényszerhez szükség van egy pontra, amihez megvan az a három kényszer, hogy minden egyes egyenes áthalad rajta. A megoldás, hogy bármely két egyenes metszéspontját felvesszük mint segédobjektumot a megfelelő egyenesre illeszkedési kényszerekkel. Ezek után a toleranciát arra adjuk meg, milyen toleranciaszint mellett esik egybe két pont. Ezek után, ha a három metszéspont közel van egymáshoz, az automatikus illesztés egyberántja őket, így a három egyenes egy ponton halad majd át (lásd 9(c). ábra).

Egy másik egyszerű példa, az egybeeső egyenesek esete. A párhuzamossági kényszer felismerése egyszerű, hiszen csak a hajlásszöget kell megvizsgálni tolerancia szempontjából. Egybeesés esetén ez nem mérvadó, hiszen bármely két nem párhuzamos egyenes metszi egymást, így a koordináták távolsága sem lehet mérvadó szempont. A megoldás, hogy ezekben az esetekben kétlépéses felismerést alkalmazunk, azaz először felismerjük a párhuzamosságokat, majd az illesztett párhuzamos egyenesek között már megvizsgálhatjuk, hogy a távolságuk toleranción belül van-e. Ha igen, a két egyenes azonosnak mondhatjuk.

4. Globális rácsstruktúra meghatározása

Az egyszerűbb „lokális” kényszerek felismerése mellett szükségünk lehet „globális” kényszerek érvényesíteni is. Ilyenek lehetnek például rácsra való illesztés és különböző szimmetriák felismerése (tengelyes, forgatásos, eltolásos).

A rácsok fontos szerepet töltenek be a kényszeres illesztés témakörében, hiszen általában a mérnöki tervezés során már a pontok, egyenesek, síkok egy rácsra illeszkednek, illetve általában az objektum mérőszámai is kvantáltak valamilyen mértékegységrendszer szerint, ami szintén egy rács-struktúrát ad. További érdekes probléma, hogy bár létezhet egy jó globális rács, előfordulhat, hogy az objektum egyes lokális részei nem illeszkednek rá, így azokat fel kell ismernünk és figyelmen kívül kell hagynunk.

Ebben a fejezetben megoldást mutatunk arra, hogy hogyan lehet egy globális rácsra illeszkedést felismerni. Bemutatjuk, hogyan reprezentálható a ’rács’ objektum a 2. fejezetben bemutatott egyenletrendszerben, hogyan tudunk a ’rács’-csal kapcsolatos kényszereket felvenni, azokat automatikusan felismerni, és hogyan tudjuk meghatározni az optimális orientációt és rácsállandót. Az algoritmus a következő alapvető lépésekből áll:

1. orientáció meghatározása
2. megfelelő orientációjú egyenesek orientációhoz való kényszeres illesztése
3. optimális rácsállandó meghatározása
4. megfelelő egyenesek rácshoz illesztése

4.1. Rács objektum

Annak érdekében, hogy a ráccsal kényszereket készíthessünk, és fel tudjuk használni kényszeres illesztés során, szükségünk van az objektum egy megfelelő reprezentációjára, ahogyan azt más objektumok esetében is láthattuk a 2.4. fejezetben. A rácsot nem illesztjük ponthalmazokhoz, hanem csak *segédobjektumnak* tekintjük a kényszeres illesztés során.

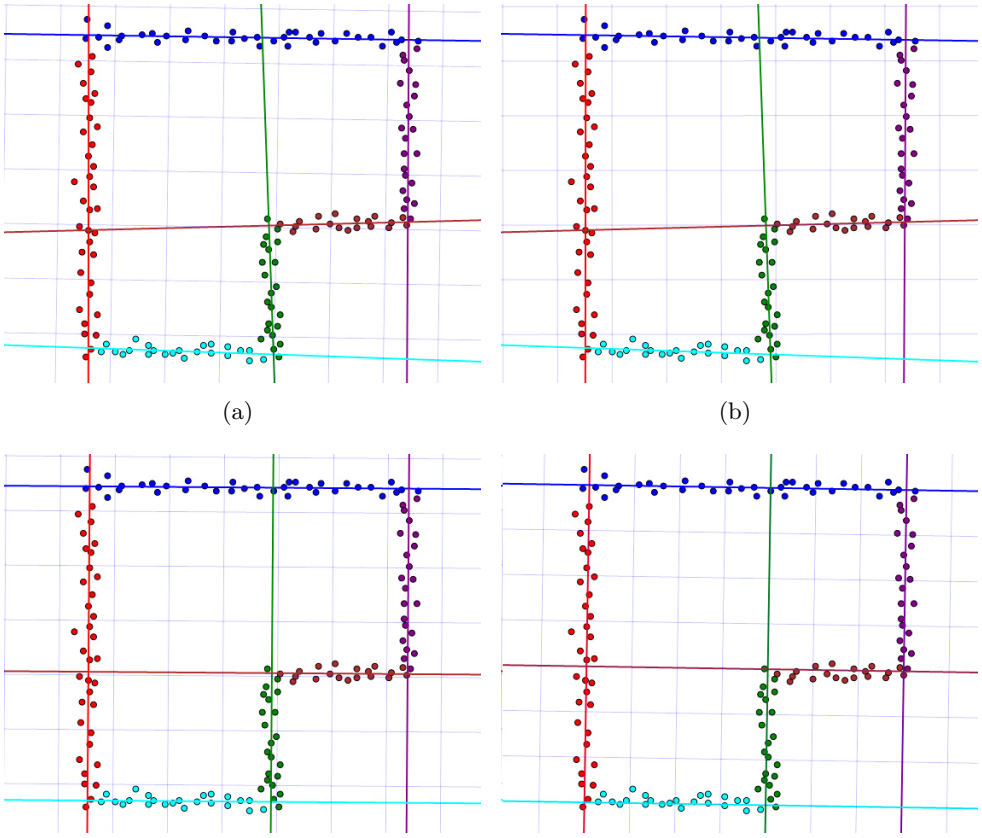
Egy rácsot az *orientációja* (n egységvektora), *origója* (egy p rácspont pozíciója) és a *rácsállandója* (c pozitív valós szám) határozza meg, azaz a rács egy 5 dimenziós vektorral megadott objektum, azzal a normalizálási kényszerrel, hogy az orientáció egységvektor. Megjegyezzük, hogy ugyanazt a rácsot többféleképpen is lehet definiálni, hiszen bármely rácspontot megadhatjuk origónak, és az orientációs vektort is négyféleképpen lehet definiálni.

4.2. Kényszerek

Fontos, hogy a ’rács’ segítségével is megadhassunk (esetleg felismerhessünk) kényszereket. Ebből vannak egyszerűbbek, mint például, hogy egy egyenes párhuzamos/merőleges a rácsra. Ennek a kényszernek az egyenlete $c(\mathbf{x}) = \min(|An_1 - Bn_2|, |An_2 + Bn_1|) = 0$ azaz a rács orientációs vektorára merőleges, vagy párhuzamos az egyenes normálvektora. A pont rácsra illeszkedésének kényszere, lényegében azt mondja ki, hogy a rács koordinátarendszerében az adott pont koordinátái egészek. Legyen n a rács orientációs vektora, $n^\perp$ egy rá merőleges egységvektor, p_0 egy rácspont pozíciója, és c a rácsállandó. Ekkor a kényszerek, hogy $\langle p - p_0, n\rangle / c$ és $\langle p - p_0, n^\perp\rangle / c$ egészek. A többi felhasznált kényszer ehhez hasonlóan került megvalósításra.

A legfontosabb ráccsal kapcsolatos kényszerek:

- pont illeszkedik a rácsra
- egyenes párhuzamos/merőleges a rácsra
- egyenes illeszkedik a rácsra



(a) (b) (c) (d)

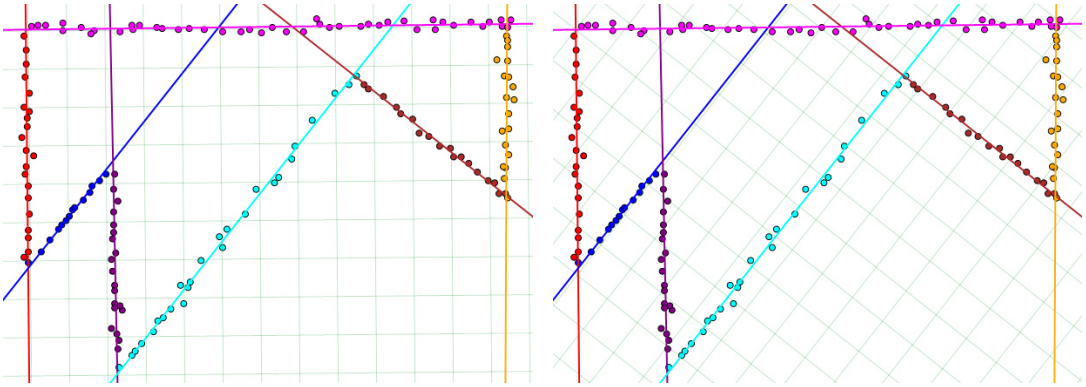
10. ábra. Rács felismerése: (a) kiindulóállapot (független illesztések); (b) optimális orientáció; (c) újra-illesztett orientált egyenesek; és (d) újra-illesztés a meghatározott rácsállandóval.

- a rácsállandó egy szám
- a rács orientációja fix
- a rács pozíciója fix

Általános esetben nem alkalmazhatjuk egyszerűen a 3.2. fejezetben ismertetett módszereket rácsokra. Ennek oka, hogy csak akkor kapunk a ráccsal jó illesztést, ha a rácsot megfelelő vektor és rácsállandóval inicializáltuk, mert egyéb esetben az első iteráció után „ragadnak” a pontok/egyenesek a megfelelő rácsponthoz/egyeneshez, és a további iterációs lépésekben végig hibás lesz az illesztés. Feladatunk tehát megtalálni a rács egy jó inicializálását, amivel már elvégezhető az automatikus felismerés.

4.3. Orientáció

Egy objektum koordinátatengelyeinek meghatározása nemcsak a rácsok esetén fontos. Nehézséget jelenthet, hogy gyakran több fő orientációs iránya is lehet az objektumnak, illetve hogy előfordulhatnak olyan részletek, amelyek lokálisan a fő orientációtól eltérő rendszerben vannak definiálva. Az itt bemutatott algoritmussal több hipotézist állíthatunk fel, majd rangsorolhatjuk azokat. Az algoritmus azt használja ki, hogy az egyes felületek adnak orientációs irányokat, például a síkok normálvektora, a hengerek, a forgásfelületek tengelye, az egyenesek iránya és így tovább. Ezeket az irányokat súlyozzuk a hozzájuk tartozó objektum kerületével (vagy 3D-ben kerületével), majd ezek között végezzük el a



(a) (b)

11. ábra. Felismert orientációk: (a) a legerősebb irány; (b) a második legerősebb irány.

klaszterezést.

Az algoritmust 2D-s esetben mutatjuk be. Feltételezzük, hogy az illesztett egyenesek mindig egy szakaszhoz tartoznak; legyen az l_i egyenes hossza $h(l_i)$, az egyenes normálvektorának argumentuma radiánban $\arg(l_i)$. Mivel két merőleges egyenes azonos orientációhoz tartozik, így az argumentumokat modulo $\pi/2$ vizsgáljuk. A megoldás, hogy klaszterezést végzünk az egyenesek argumentumai között modulo $\pi/2$, és a klasztereket súlyozzuk a benne levő egyenesek $h(l_i)$ hosszával. A klaszterek mérete egy előre megadott toleranciaszinttől függ. Az így megkapott legjobb klaszter a modell egy jó orientációját adja. A második legjobbat úgy kaphatjuk, hogy kivesszük az elsőhöz tartozó egyeneseket, majd ismét megkeressük a legjobb klasztert.

Az eljárás 3D-re is könnyen kiterjeszthető, ott a Gauss gömbön kell klaszterezni a normálvektorokat, majd kiválaszthatjuk a legjobb klasztert és a rá merőleges síkhoz közeli vektorok közt klaszterezve kapjuk meg a másik két irányt.

4.4. Rácsállandó

Az optimális orientáció megtalálása után elkezdhetjük megkeresni az optimális rácsállandót is (10(b). ábra). A rácsállandó megkereséséhez először a megfelelő orientációjú egyeneseket kényszeresen illesztjük a rácshoz (10(c). ábra), majd az ezek közötti távolságok felhasználásával keressük meg a rácsállandót (10(d). ábra).

Jelöljük az orientációhoz illesztett egyeneseket $\{l_i\}$-vel és keressük a rácsállandót egy előre megadott intervallumban. A következő lépés, hogy vesszük ezek között az összes párhuzamos egyenespárt, és a köztük lévő abszolút távolságokat $\{d_{ij}\}$. Ha létezik optimális egyenesekre illeszkedő rácsállandó, akkor ez tolerancián belül a távolságok legjobb közös osztója, azaz egy olyan szám aminek az átlagos maradéka a számoktól kicsi.

Legyen d egy közelítő közös osztó, és $\{d_{ij}\} = \{n_l\}$ az összes párok távolságainak halmaza, ekkor

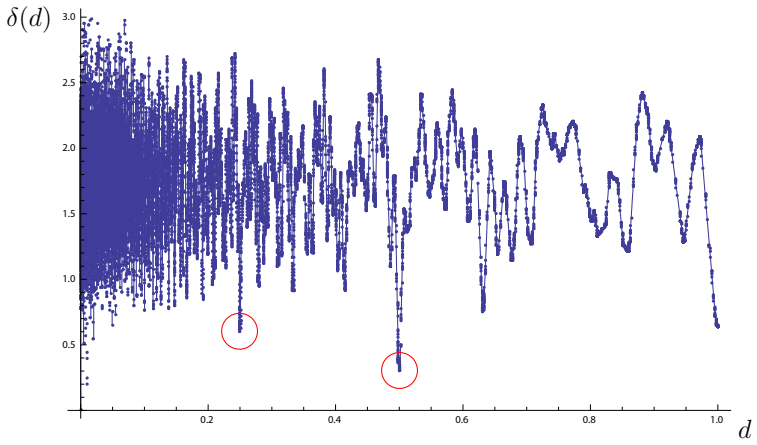
$$\delta(d) = \sum_l \min(n_l \mod (d), d - (n_l \mod (d)))$$

az összes eltérés. Ennek a minimalizálása nem a legjobb, hiszen $\delta(d)$-nél $\delta(d/k)$ elég nagy k-ra biztosan jobb eredményt ad. A megoldás, hogy az eltérést d arányában nézzük, így a

$$\delta(d) = \sum_l \frac{\min(n_l \mod (d), d - (n_l \mod (d)))}{d} = \sum_l \min\left(\left\{\frac{n_l}{d}\right\}, 1 - \left\{\frac{n_l}{d}\right\}\right)$$

képletet fogjuk alkalmazni, ahol $\{x\}$ az x törtrésze.

Tehát a célunk δ minimumhelyének megkeresése egy adott $[d_{min}, d_{max}]$ intervallumban. Az 12. ábrán láthatóan δ-nak nagyon nagy a variációja, így a hagyományos numerikus minimumkeresési módszerek megbuknak, és ezért más eszközöket kell alkalmaznunk.



12. ábra. $\delta(d)$ függvény a $\{12.0302, 4.9804, 3.9998, 2.5101, 17.0301, 18.9502, 2.0201\}$ számokkal; minimum a $d = 0.50081$-ben

4.1. Állítás. *δ a minimumát valamely n_l/k értékben veszi fel valamely n_l-re és k pozitív egésszel.*

Bizonyítás. $\min\left(\left\{\frac{n_l}{d}\right\}, 1 - \left\{\frac{n_l}{d}\right\}\right)$ lokálisan egy adott d kis környezetében minden egyes n_l-re a $\frac{n_l}{d} - m_l$ vagy $1 - (\frac{n_l}{d} - m_l)$ alakban írható. Figyeljük meg, hogy az ilyen lokális részeken a függvény d-ben monoton, ami azt jelenti, hogy δ ott veszi fel a minimumát, ahol a képlet már nem érvényes, azaz valamely m_l megváltozik. Ez pontosan akkor következik be, ha valamely n_l d egész számú többszöröse, azaz $d = n_l/k$. □

Tehát a minimumot elég előre meghatározott helyeken keresni. Ezt olyan n_l-re és hozzátartozó k-ra kell elvégezni, ahol $n_l/d_{max} \leq k \leq n_l/d_{min}$, ami összesen $O(n \max_l n_l/d_{min})$ darab szám, δ kiszámítása pedig $O(n)$ lépés, tehát a minimumkeresés nagyságrendileg elvégezhető $O(n^2 \max_l n_l/d_{min})$ lépésben.

5. Szimmetriák felismerése és meghatározása

A szimmetriák szintén nagyon fontos szerepet töltenek be a globális kényszerek között. Gyakran fordulnak elő forgásszimmetriák, tengelyes szimmetriák, közös eltolásos irányok, és így tovább. A jelen kutatás a tengelyes szimmetriák felismerésére koncentrál. Fontos ismertetnünk Langbein és mások [9] munkáját, ahol ponthalmazokban ismertek föl szimmetriákat. Másik fontos eredmény [11], ahol modellek lokális görbületi jellemzői alapján keresnek részleges szimmetriákat. A mi kutatásunk mérnöki objektumok pontos szimmetriatengelyeinek meghatározása, nem diszkrét pontok vagy geometriai jellemzők alapján, hanem illesztett görbékből és felületekből kiindulva.

A módszer lényege, hogy felvesszük az összes lehetséges szimmetriatengelyt (segédegyenesek), amelyek lokálisan két objektumhoz (pl. egyenespár, pontpár, körök) tartozhatnak. Ezen segédegyenesek között ezután klaszterezést végzünk, és a legjobb klasztereket kiválasztjuk szimmetriatengely jelölteknek, amelyeket ezek után minősítünk. A kiválasztott legjobb tengelyhez fel lehet venni kényszereket, így kényszeres illesztéssel a modell tökéletesíthető. Az algoritmus lépései:

1. összes pontpár közötti szakaszfelező merőlegesek és az összes egyenespár közötti mindkét szögfelező kigyűjtése: *segédegyenesek*
2. segédegyenesek között klaszterezés
3. klaszterek egyeneseinek átlagolása: *szimmetriatengely jelöltek*
4. szimmetriatengely jelöltek minősítése
5. legjobb tengely kiválasztása, és a kényszerek meghatározása

Az algoritmus működését a 13. ábra szemlélteti.

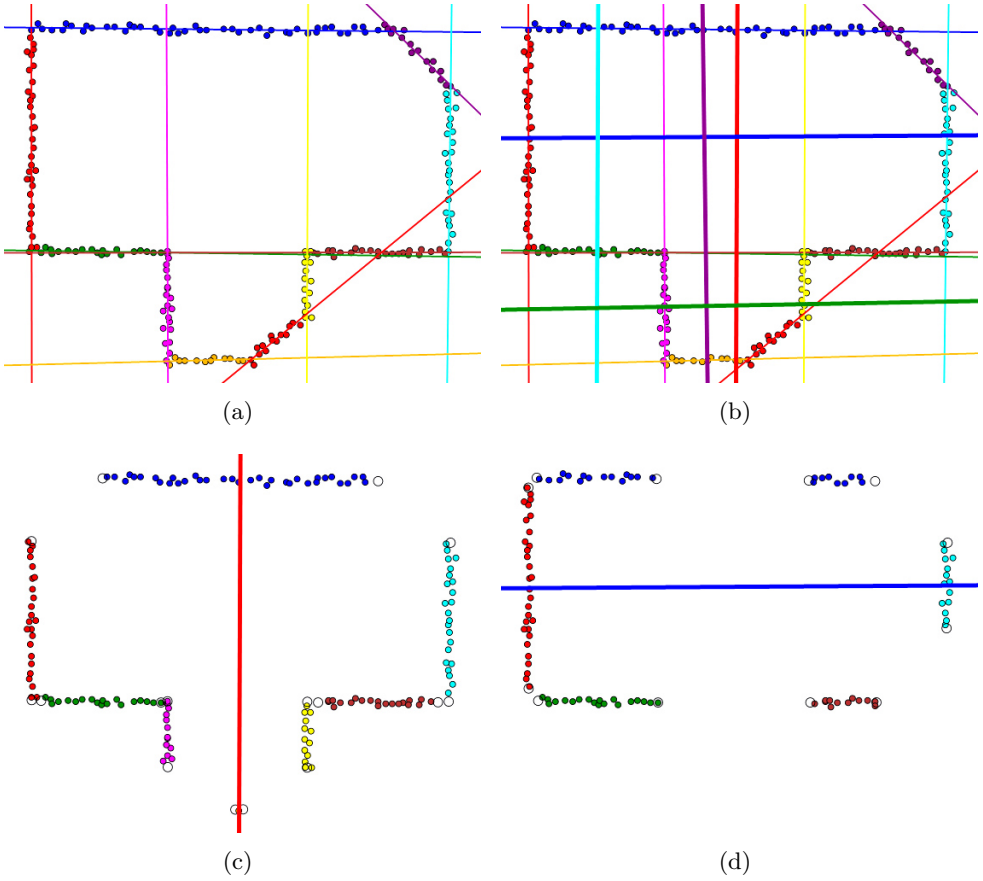
5.1. Segédegyenesek, klaszterezés

Feltételezzük, hogy az összes egyenes egy pontsorozathoz van illesztve, amely egy véges szakaszt reprezentál. Az illesztés után meghatározhatjuk a szakaszok végpontjait, legyenek ezek $\{p_i\}$, az egyenesek $\{l_i\}$. Legyen $L_1 = \{Szakaszfelezo(p_i, p_j) : i < j\}$ és $L_2 = \{Szogfelezo_{1,2}(l_i, l_j) : i < j\}$ egyeneshalmazok. A segédegyenesek pedig $L = L_1 \cup L_2$.

A klaszterezés a segédegyenesek között történik. Először az egyenesek orientációja között végzünk klaszterezést, majd az azonos orientációjú egyenesek origótól való távolságát vizsgáljuk. A klaszterezés közben mindig egy előre megadott ε hosszú intervallumot keresünk, amibe a lehető legtöbb szög, illetve távolság esik. Ezt az intervallumot diszkretizált minimumkereséssel határozzuk meg. A így megkapott $K \subset L$ klaszterekben lévő egyeneseket átlagoljuk, így megkapjuk az l_K egyeneseket. Ezeket hívjuk szimmetriatengely jelölteknek. Az egyenesek minősítése klaszterek alapján nem elégséges, számos példa mutatja, hogy nagyobb klaszterekhez sokszor gyengébb globális szimmetriatengely tartozik.

5.2. Szimmetriatengely jelöltek minősítése

A szimmetriatengelyek minősítése a szimmetrikus részek ívhosszának és az összkerület arányával történik. Legyen egy szimmetriatengely jelölt l_K. Ekkor K segédegyenesei alapján információkat kapunk arról, milyen egyenesek/körök vesznek rész a szimmetriában. Azokkal az egyenesekkel is foglalkoznunk kell, amelyek merőlegesek a tengelyre, hiszen a szakasz önmagában szimmetrikus lehet az adott tengelyre, hasonlóan azokat a köröket is, amelyek középpontján átmegy a tengely. Természetesen e két utóbbi tulajdonságot is csak valamilyen toleranción belül tudjuk eldönteni. Ezek után meghatározzuk az adott egyenesekhez



13. ábra. Szimmetriatengely felismerése: (a) kiindulóállapot; (b) szimmetriatengely jelöltek; (c) a legerősebb szimmetriatengely (70%); és (d) egy második szimmetriatengely (41%).

vagy egyenespárhoz, körhöz vagy körpárhoz tartozó szimmetrikus részek ívhosszát. Az ívhosszak összege és az összkerület aránya adja a szimmetriatengely minőségét.

5.3. Szimmetria kényszerek

Amennyiben egy szimmetriatengely jelöltet elfogadtunk, fel kell vennünk a hozzá tartozó kényszereket, és elvégezhetjük az illesztést. A szimmetriatengelyt egy segédobjektumként felvett egyenes reprezentálja. Természetesen már használhatjuk az *egyenes átmegy a kör középpontján* és az *egyenes merőleges a tengelyre* kényszereket. A szimmetriákhoz a következő új kényszerekre van szükségünk:

- a szimmetriatengely két egyenes szögfelezője (ebből két típus van mivel két szögfelező van)
- a szimmetriatengely egy szakasz (két pont) szögfelezője

Mindkét kényszer több egyszerűbb kényszerből áll, segédobjektumokkal megvalósítva.

6. Keretrendszer

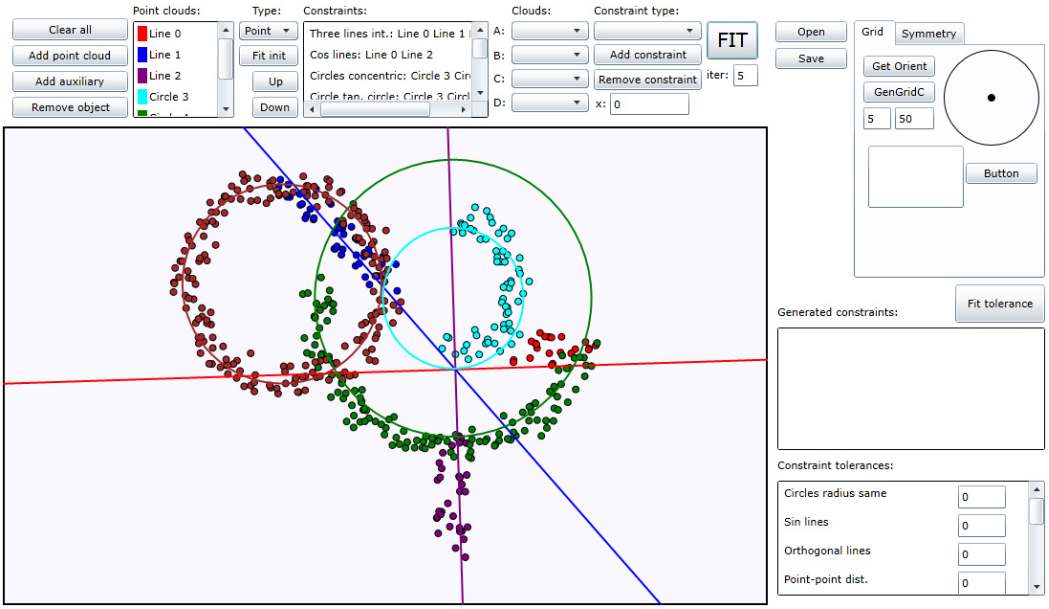
Az algoritmusok szemléltetésére készítettem egy 2D-s keretrendszert, amivel a dolgozatban bemutatott ábrák nagy része is készült. A program *C#*-ban íródott Silverlight környezetben. A program ún. *.pcf* formátumú file-okat (lást [2]) képes írni/olvasni, amelyek tartalmazzák a mért adathalmazokat, a hozzá tartozó illesztett objektumokat, és a kényszereket. Az alkalmazás internetről is elérhető. Egy 3D-s keretrendszer fejlesztése folyamatban van. A bemutatott ábrák nagy része szintén a program segítségével készült.

A program 2D-s szegmentált pontfelhőket kezel. Az illeszthető görbék lehetnek egyenesek vagy körök. Megvalósításra került a projekt alapját képező algoritmus kényszeres illesztésekre, aminek működését a 14. ábra szemlélteti. Ahogyan az algoritmus futása során kiderül, a program kijelzi, ha az egyes kényszerek egymásnak ellentmondanak, vagy egymásból következnek. A kényszerek között megadható prioritási sorrend. Szintén felvehetőek segédobjektumok is, amikkel számos kényszer megadható. Ezt több ízben is kiegészítettük, ahogy az a dolgozatban bemutatásra került.

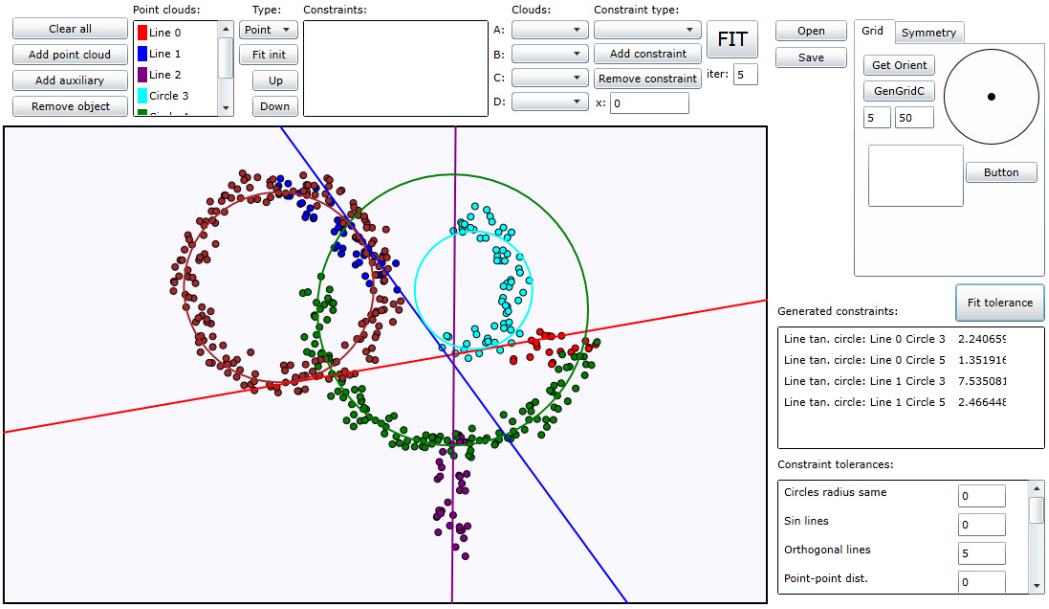
A kényszerek automatikus felismeréséhez szükségünk van a toleranciák megadására. Ezek megadása után a program végrehajtja a kényszeres illesztést, és megvizsgálható, hogy melyek azok a kényszerek amiket érvényesítettünk (lásd 15. ábra).

A rácsok felismerésénél egy paraméter, hogy milyen értékek között keressük a rácsállandót, és melyik orientációt fogadjuk el. A program ezek után elvégzi a kényszeres illesztést a megfelelő paraméterekkel.

Szimmetriák esetén hipotéziseket kapunk az egyes szimmetria tengelyekre, és megtekinthetjük, hogy mennyire elfogadhatóak ezek a hipotézisek. Ezek után a kiválasztott tengelyhez a program felveszi a kényszereket, és elvégzi az illesztést is.



14. ábra. Kényszeres illesztés explicit módon megadott kényszerekkel

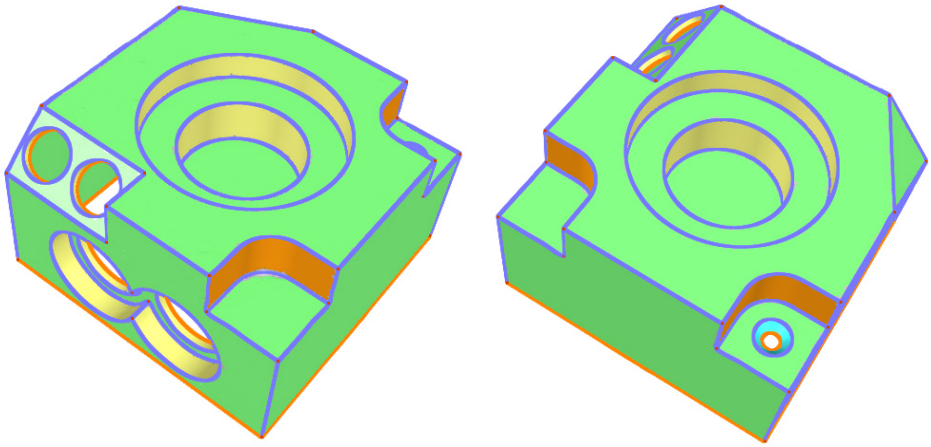


15. ábra. Kényszeres illesztés automatikusan felismert kényszerekkel

7. Összefoglalás

Jelen projektben 3D-s mért adatok tökéletesítésével foglalkoztunk. Ha az adott szegmentált pontfelhőket külön-külön illesztjük felületekkel, egy „pontatlan” CAD modellt kapunk. Kutatásunk célja ezen pontatlanságok kiküszöbölése volt oly módon, hogy különböző geometriai kényszereket vezettünk be, és felületcsoportokat illesztettünk egységesen kényszerek figyelembevételével.

A dolgozatban bemutattunk egy robusztus módszert kényszeres illesztésre. Kidolgoztunk egy algoritmust, ahol a kényszerek megadása nem a felhasználó feladata, hanem ezeket automatikusan fel lehet deríteni. Megoldást adtunk egyszerűbb „lokális” kényszerektől kezdve az összetettebb, felületcsoportok között ható „globális” kényszerek kezelésére.



16. ábra. Egy mért adatokból származó szegmentált objektum

A 16. ábra jól szemlélteti a jövőbeli célokat. Az itt látható 3D-s objektumnak számos lényegi szimmetriája van, ezek nem teljes szimmetriák, egyes részek ezektől eltérhetnek. A további tervek között szerepel, hogy az algoritmusainkat 3D-s környezetben általánosítjuk. Fontos cél, hogy kiterjesszük az algoritmust szabadformájú görbékre és felületekre. Szintén fontos lehet további összetettebb kényszerek vizsgálata és felismerése, mint például forgásszimmetriák, pontmintázatok stb.

Köszönetnyilvánítás

Köszönöm témavezetőmnek Várady Tamásnak a munkám irányítását, a felmerülő problémákra adott kreatív ötleteket és a sok konzultációt. Jelen kutatás az OTKA (101845) támogatásával folyik.

Hivatkozások

[1] Geomagic, Inc., `http://www.geomagic.com`.

[2] P. Benkő, G. Kós, T. Várady, L. Andor, and R. R. Martin. Constrained fitting in reverse engineering. *Computer Aided Geometric Design*, 19(3):173–205, 2002.

[3] P. Benkő, R. R. Martin, and T. Várady. Algorithms for reverse engineering boundary representation models. *Computer-Aided Design*, 33(11):839–851, 2001.

[4] P. Benkő and T. Várady. Segmentation methods for smooth point regions of conventional engineering objects. *Computer-Aided Design*, 2004.

[5] I. Coope. Circle fitting by linear and nonlinear least squares. *Journal of Optimization Theory and Applications*, 76(2):381–388, 1993.

[6] J. Jiang, Z. Chen, and K. He. A feature-based method of rapidly detecting global exact symmetries in CAD models. *Computer-Aided Design*, 45(8–9):1081 – 1094, 2013.

[7] Y. Ke, W. Zhu, F. Liu, and X. Shi. Constrained fitting for 2D profile-based reverse modeling. *Computer-Aided Design*, 38(2):101–114, 2006.

[8] F. C. Langbein. *Beautification of reverse engineered geometric models*. PhD thesis, Cardiff University, 2003.

[9] M. Li, F. C. Langbein, and R. R. Martin. Detecting approximate incomplete symmetries in discrete point sets. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pp. 335–340. ACM, 2007.

[10] G. Lukács, R. R. Martin, and D. Marshall. Faithful least-squares fitting of spheres, cylinders, cones and tori for reliable segmentation. In *Computer Vision—ECCV'98*, pp. 671–686. Springer, 1998.

[11] N. J. Mitra, L. J. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3D geometry. *ACM Transactions on Graphics (TOG)*, 25(3):560–568, 2006.

[12] J. Porrill. Optimal combination and constraints for geometrical sensor data. *The International Journal of Robotics Research*, 7(6):66–77, 1988.

[13] H. Pottmann, S. Leopoldseder, and M. Hofer. Approximation with active B-spline curves and surfaces. In *Computer Graphics and Applications, 2002. Proceedings. 10th Pacific Conference on*, pp. 8–25. IEEE, 2002.

[14] V. Schomaker, J. Waser, R. t. Marsh, and G. Bergman. To fit a plane or a line to a set of points by least squares. *Acta crystallographica*, 12(8):600–604, 1959.

[15] T. Várady and R. R. Martin. Reverse engineering. *G. Farin, J. Hoschek, M. S. Kim, Handbook of Computer Aided Geometric Design, Chapter 26*, Elsevier, 2002.

[16] T. Várady, P. Salvi, *3D-s számítógépes geometria és alakzatrekonstrukció tárgy diasorok*, BME IIT (2013)

[17] N. Werghi, R. Fisher, C. Robertson, and A. Ashbrook. Modelling objects having quadric surfaces incorporating geometric constraints. In *Computer Vision—ECCV'98*, pp. 185–201. Springer, 1998.