# Összehasonlítás a 2D-s és 3D-s tárgyfelismerő technikák között a robot navigációban

# Comparison between 2D and 3D Object Recognition Techniques for Mobile Robot Navigation

MOLNÁR Szilárd[1], TAMÁS Levente[2]

[1]Kolozsvári Műszaki Egyetem hallgatója, Rendszermérnöki kar,
Fejlett mérnöki vezérlés a gyártásban,
Kolozsvár, George Baritiu utca 26-28, 400027, Románia,
+40-(0)721775376, molnarszilard10@gmail.com
[2]Levente.Tamas@aut.utcluj.ro

## Abstract

*This is a short presentation about the comparison between a 2D object recognition technique, YOLO, and a 3D object recognition technique with Kaolin. These techniques are analyzed, and used in navigating a small robot through a hallway. After presenting the used hardware elements, we will conclude the results of the comparison according to the effectiveness of the application.*

## Kivonat

*Ez egy rövid leírás egy projektről, amelyikben egy összehasonlítunk egy 2D-s tárgyfelismerő technikát, a YOLO-t, és egy 3D-s tárgyfelismerő technikát, a Kaolint. Ezeknek a működését analizáljuk, majd felhasználjuk őket egy kisméretű robot folyosón való irányításánál. Ismertetjük a használt eszközöket, majd az alkalmazás hatékonyságát elemezve, levonjuk az összehasonlítás következményét.*

**Kulcsszavak:** Mesterséges intelligencia, tárgyfelismerés, kaolin, YOLO, tér feltérképezés.

## 1. Bevezetés

A mai zsúfolt világban rengeteg autó veszi körül az embert, amiből gyakorta emberi hibából következő baleset történik. Mi történne, ha az amúgy is rohamosan fejlődő számítástechnikát felhasználva alkotni olyan rendszereket, amelyek lényegesen csökkentik a közúti baleseteket? Ez a dolgozat erre a kérdésre próbál válaszolni, azáltal, hogy leír egy alapvetően egyszerű megvalósítását a fent említett problémának. A leírt módszer magában foglalja a használt technológiákat úgy hardware mint software szinten. Ebben a dolgozatban a fókusz csak a beltéri tárgyak felismerésén van, de ebből kiindulva bővíteni lehet azt egészen az önvezető autókban való felhasználásig.

## 2. felhasznált hardware

### 2.1 A kamera

Az első eszköz, amit érdemes meghatározni az a szenzor. Az emberi érzékszervekből kiindulva logikusnak tűnik optikai szenzort használni, mivel az ember a környezetéről felvett információ legnagyobb részét látás útján kapja. A köztudatban a legelterjedtebb optikai szenzor a kamera.

A kameráknak több típusa ismert, ezek közül a 2 dimenziós a legegyszerűbb, sőt annyira elterjedt, hogy jelenleg is rengeteg autóban megtalálható fedélzeti kamerák formájában. Ezáltal felmerül a kérdés, hogy ezeket a kamerákat felhasználva lehetséges-e feltérképezni az adott autó környezetét. Annak ellenére, hogy létezik technológia, ami elfogadhatóan felismeri a tárgyakat 2 dimenziós adatokat felhasználva, létezhetnek problémák, főleg ha egy autót ábrázoló reklámtáblát észlel a rendszer, ami esetleg egy forgalommal szemben közlekedő gépjárműnek tűnhet. Ez a probléma a távolságérzet hiányából ered.

Visszatérve az emberi látáshoz tanulmányozásához, érdemes megjegyezni az elsődleges okot, amiért az embereknek két szemük van. Ezt az okot sztereó látásnak nevezik. Ez azt jelenti, hogy a két szem a különböző elhelyezésükből adódóan két különböző szemszögből látja a környezetet. Ezekből a különbségekből az agy kiszámítja a látott tárgyak helyzetét. A gépjárművekbe szerelt sztereókamerákkal jobb mélységérzetet lehet elérni, viszont ez a megoldás se tökéletes.

Ha mellőzzük az optikát találhatunk egy másik feltérképezési módot, a lézeres feltérképezést. Ez a technika Lidar néven nagy előrelépéseket tett az önvezető gépjárművek terén. A probléma vele az, hogy lényegesebben drágább kivitelezni, mint egy néhány egyszerű kamerát felszerelni a járműre.

Az optimális megoldást a Lidar és a 2 dimenziós kamerák között érdemes keresni. Ezek a kamerák a mélység kamerák. Az általános kamerákhoz hasonlóan képet is felvesznek színekkel együtt, viszont infravörös sugarakkal le is pásztázzák a környezetet, ez által a Lidarhoz hasonló térbeli képet alkotva. Ez a fajta a kamera megfelelő térlátást biztosít megfizethető áron.

Az általunk használt modell az Asus Xtion PRO Live mélység kamera. Ez a kamera képes megszokott 2D-s kép felvétele mellett 3D-s pontfelhőkkel feltérképezni a környezetet.

### 2.2 A jármű

Beltéri használatra egy távirányítós autót választottunk ki, aminek alapvetően meg van oldva az áramellátása, illetve a haladáshoz és kanyarodáshoz szükséges motor. A többi eszközt, erre az autóra szereltünk fel.

### 2.3 A vezérlő egység

Mint sok más képfelismeréses alkalmazáshoz, ide is deep learningre, azaz gépi tanulásra van szükség. A gépi tanuláshoz nagy teljesítményű számítógép szükséges, viszont a mobilitás megőrzéséhez, alacsony fogyasztású és kis helyet elfoglaló egység az ideális, amit egy külső akkumulátorral működtetni lehet.

Elsőre ez nehéz feladatnak minősül, de ismét az Nvidia szolgáltatja a megoldást. Akár a Jetson Nano, akár a Jetson AGX Xavier egységet vesszük, teljesülhetnek a fent említett követelmények.

## 3. Kutatás

Először a 2D-s technológia kiválasztására van szükség. Mivel nagyobb hangsúlyt fektetünk a 3D-s technológiákra, ezért ezen a területen nem végeztünk szétágazó kutatásokat. A korábbi tapasztalatok alapján megfelelőnek találtuk a YOLO v3 [1] (You Look Only Once version 3) használatát.

A mélység kamerák általában pontfelhőként adját át az adatot a vezérlőnek. Ezeket a pontfelhőket kell feldolgozza az eszköz, aminek a kiolvasását és elmentését az ROS keretrendszer segítségével mentjük el.

Léteznek előre megírt gépi tanuló módszerek, amikkel sikeresen lehet pontfelhőkkel dolgozni, viszont ezek hatásossága és felhasználhatósága ingadozik. Első lépésként érdemes ezek közül kiválasztani a nekünk legmegfelelőbbet.

Ezek a módszerek közül sok össze volt hasonlítva, de nem az összes. Ezért mi felhasználva a már összehasonlított módszerek eredményeit, kipróbáljuk a lehetőségeket, és azok közül válasszuk ki a legmegfelelőbbet.

[2] öt általunk kipróbált módszert hasonít össze, amelyek: PointNet [3], PointNet++ [4], PointConv [5], SpiderCNN [6] és PointCNN [7]. Ezek közül a PointConvval érte el a legpontosabb eredményt. Nekünk viszont kevésbé sikerült jól a próba.

Egyéb módszerek a PointPillars [8], SECOND [9], VoteNet [10], PointRCNN [11], ScanObjectNN [12], Frostrum ConvNet [13], TANet [14], VoxNet [15] és Pl-Net3D [16]. Mindegyik módszer lényegében rendben vannak, viszont még mindig nem értük el a megfelelő használhatóságot és integrálhatóságot.

Viszont a Kaolin [17] esetében megtaláltuk a megfelelő módszert a gépi tanuláshoz. Egyszerű a működése, és megfelelő a pontossága. Tartalmaz egy példát [18] is, amit módosítva megkapjuk azt a kódot amire nekünk is szükségünk van.

A Kaolin egy a PyTorch alkalmazására fejlesztett gépi tanuló könyvtár, amelyik többek között sokszög hálóból [19] és pontfelhőből dolgozik. Az adatok PyTorch tenzorban vannak tárolva, továbbá több renderer közül választja ki a legmegfelelőbbet. Ez a módszer a CUDA magokra támaszkodik, amit az Nvidia fejlesztett ki, és ami támogatva van az Nvidia által forgalmazott legtöbb egységben, így használható a Jetson Nanon, vagy a Jetson AGX Xavieren is.

## 4. A modell betanítása

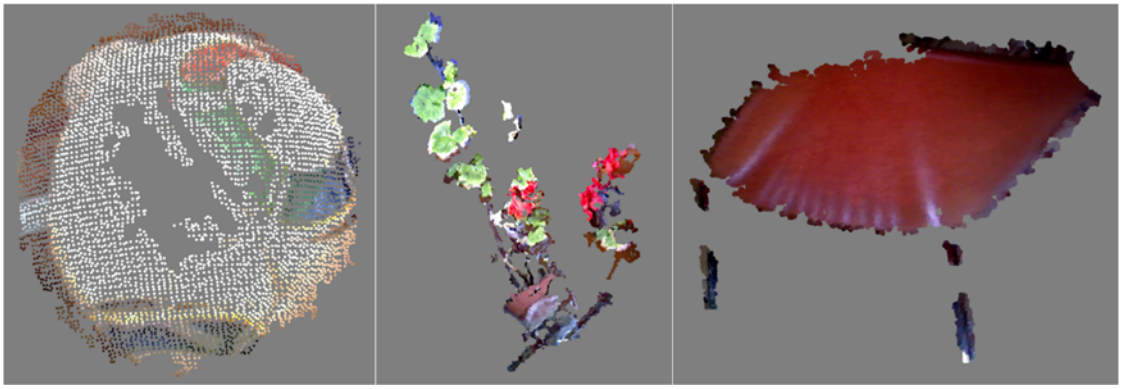
A projekt előkészítése során megfigyeltük, hogy a betanulási folyamatot érdemes egy nagyobb teljesítményű számítógépen elvégezni, a tényleges tárgyfelismerést pedig a használt mikrokontrolleren, ezzel felgyorsítva a fejlesztési folyamatot.

A YOLO módszer egyszerű két dimenziós képeket olvas be, amiken manuálisan előre be volt jelölve a felismert kép illetve a hozzá tartozó címke.

A Kaolin módszer kiválasztásával együtt járt az adatszett kiválasztása is. A módszer leírásban az ajánlott és legegyszerűbben felhasználható adatszett a ModelNet10 [20], amely többnyire háztartásbeli tárgyakról készült sokszög hálókat tartalmaznak. Ezek a sokszög hálók off fájlformátumban [21] találhatóak meg, és a betanítás részhez mi is off formátumba mentjük a saját adat szettünket.

Első körben az adatszett három kategóriában tartalmazott tárgyakat, ezek a következők:

Labda, Szék, Növény (1. ábra).



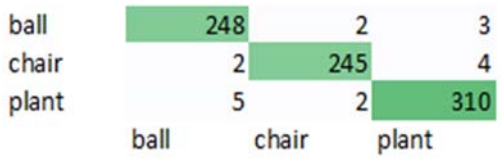
1. ábra

*a)labda* *b) növény* *c)szék*

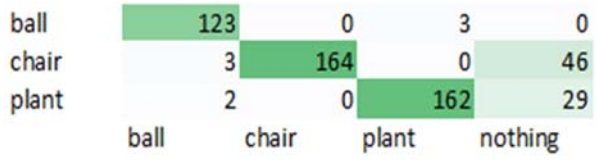
Mindegyik kategóriában két különböző tárgy van. Ezeket a kamera segítségével leszkenneltük, és elkészítettük az őket tartalmazó adatszetteket. Ezekkel a kategóriákkal történik meg a modell betanítása, ezért a létrejött adatszetteket két csoportra osztjuk. A „tanuló” illetve a „teszt” csoportok.

A modell minőségéről a zavar mátrix adhat egy jelentőségteljesebb képet. Egy zavar mátrix egy olyan mátrix, amelyik leírja, hogy a tanítás során előforduló ellenőrzéskor melyik kategóriába hány darab tárgy lett besorolva. A mátrix főátlóján látható értékek jelentik a helyes felismeréseket, a főátlón kívül eső értékek pedig a helytelen felismeréseket.

Két zavar mátrixot érdemes megfigyelni, az elsőben (2. ábra) a kaolin eredményei vannak feltüntetve, a másodikban (3. ábra) a YOLO felismerései láthatóak.




2. ábra *Kaolin három kategóriával*




3. ábra. *YOLO három kategóriával*

Az ábrákat összehasonlítjuk, akkor láthatjuk, hogy a kaolin esetében nagyobb a pontosság, mivel abban az esetben minden esetben behatárolja az adott kategóriát, a YOLO-nál viszont sok esetben nincs meghatározva a kategória.

Ezután következett a tényleges betanítást, amikor egy adott folyosón lévő tárgyakat szeretnénk felismerni. Itt 21 kategóriát illetve ezeknek variációit különböztettünk meg, ezek a következők:

Ajtó (balra, jobbra, előre, előre-akadályként), fal (balra, jobbra, kétoldalt, akadályként), növény (balra, jobbra, előre), tűzoltókészülék (felállítva, ledöntve), szemetes kuka (felállítva, ledöntve, felállítva akadályként, ledöntve akadályként), szekrény (előre, akadályként).

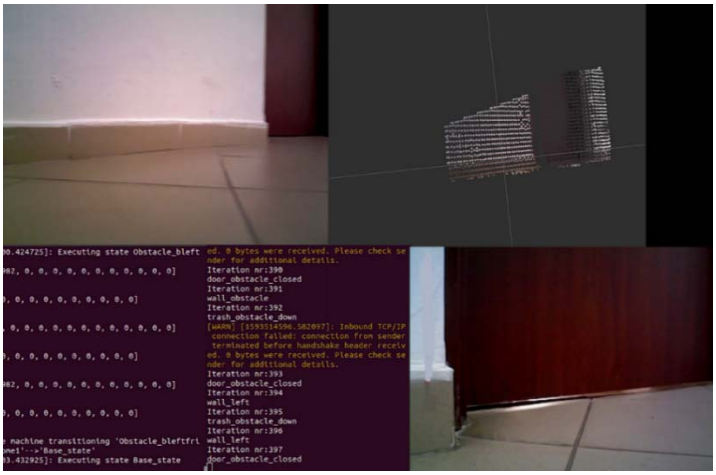
A kaolin esetében a legnagyobb problémát az ajtók alkategóriái adják, ezek nagyon hasonlóak, nehezen tesz különbséget köztük. A YOLO esetében a probléma ismét abból fakad, hogy sok kép van amit nem tud semelyik kategóriába se be bevenni.

Mindkét technika tanításában az elért pontosság 86% körül volt, ami ennyire hasonló kategóriák esetében elsőre nem rossz, de érdemes lehet a jövőben pontosabban meghatározott kategóriákra tanítani az algoritmust, és ezzel elérni legalább 95%-ot.

## 5. Tesztelés

A modulok ROS (Robotic Operating System) csatornákon keresztül kommunikáltak. Az élő képet a megfelelő képfelismerő módszer megkapta, majd visszatérített egy valószínű kategóriát. A visszatérített kategória határozta meg, hogy a robot előre menjen, kanyarodjon, vagy tolasson.

A 4. ábrán látható a kaolin módszer működés közben. Az online kép a bal felső sarokban van, a jobb felső sarok a pontfelhőt mutatja, a balalsó sarokban a felismert kategória illetve az elvégzendő parancs. A jobb alsó sarok a felismert kategóriáról mutat egy képet.




4. ábra. *Kaolin működésben*

A két metódust egymás után alkalmaztuk az autó irányításában. Először a YOLO metódus volt tesztelve. Az autó működött bizonyos mértékig, de voltak problémák. Az első abból adódott, hogy néhány esetben nem ismert fel kategóriát, ami miatt kézzel el kellett mozdítani a robotot. A másik probléma az volt, hogy a tárgyak pozícióját néhány esetben nem találta el, illetve túl sok energiát húzott le a felismerés, a Jetson néha kikapcsolt emiatt.

A kaolin esetében a probléma okozója az volt, hogy a pontfelhő nem lát fél méternél közelebb, ami miatt ha egy tárgy túl közel van, a pontfelhő ritka lesz, ami hasonlít ahhoz az esethez, amikor nincs semmi a kamera előtt.

## 6. Végső értékelés

Az alkalmazás segített behatárolni a lehetséges felhasználási és továbbfejlesztési lehetőségei mindkét kategóriában. Egyik módszer se volt tökéletes, de megfelelő munkával mindkettő járható utat jelent.

A kaolint és más pontfelhővel dolgozó alkalmazások nem tanácsoltak kis tér esetében. Ilyen esetekre akár telefonon futó 2 dimenziós tárgyfelismerő alkalmazások is megfelelőek.

Az autóiparban talán a két dimenziós YOLO és a pontfelhőket használó kaolin kombinációja lenne a legmegfelelőbb megoldás. Annak ellenére, hogy a feldolgozó számítógép erősebb kellene legyen, érdemes ezen a téren dolgozni, mivel megfelelő optimalizálással egy autó képes lehet ellátni egy ilyen számítógép szükségleteit.

## Könyvészet


[1] M. Bjelonic, "{YOLO ROS}: Real-Time Object Detection for {ROS}," 2016. [Online]. Available: https://github.com/leggedrobotics/darknet_ros}. [Accessed 25 5 2020].

[2] Y. a. W. H. a. H. Q. a. L. H. a. L. L. a. B. M. Guo, "Deep Learning for 3D Point Clouds: A Survey," *arXiv preprint arXiv:1912.12033,* 2019.

[3] C. R. a. S. H. a. M. K. a. G. L. J. Qi, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," *arXiv preprint arXiv:1612.00593,* 2016.

[4] C. R. a. Y. L. a. S. H. a. G. L. J. Qi, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," *arXiv preprint arXiv:1706.02413,* 2017.

[5] W. a. Q. Z. a. F. L. Wu, "PointConv: Deep Convolutional Networks on 3D Point Clouds," *arXiv preprint arXiv:1811.07246,* 2018.

[6] Y. a. F. T. a. X. M. a. Z. L. a. Q. Y. Xu, "SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters," *arXiv preprint arXiv:1803.11527,* 2018.

[7] W. a. Q. Z. a. F. L. Wu, "PointConv: Deep Convolutional Networks on 3D Point Clouds," *arXiv preprint arXiv:1811.07246,* 2018.

[8] Y. a. T. O. Zhou, "PointPillars," *arXiv,* no. 10.1109/CVPR.2018.00472.

[9] Y. a. M. Y. a. L. B. Yan, "SECOND : Sparsely Embedded," no. 10.3390/s18103337, pp. 1-17, 2018.

[10] C. R. a. L. O. a. H. K. a. G. L. J. Qi, "Deep Hough Voting for 3D Object Detection in Point Clouds," 2019.

[11] S. a. W. X. a. L. H. Shi, "PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud," 2019.

[12] A. a. C. A. X. a. S. M. a. H. M. a. F. T. a. N. M. Dai, "ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes," 2017.

[13] Z. a. J. K. Wang, "Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal 3D Object Detection," 2019.

[14] T. R. 3. O. D. f. P. C. w. T. Attention, "Zhe Liu and Xin Zhao and Tengteng Huang and Ruolan Hu and Yu Zhou and Xiang Bai," *AAAI,* no. 1912.05163, 2020.

[15] D. a. S. S. Maturana, "VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition," 2015.

[16] A. {. a. R. {. a. C. Y. {. a. R. {. a. A. {Bab-Hadiashar}, "PL-Net3d: Robust 3D Object Class Recognition Using Geometric Models," *IEEE Access,* vol. 7, no. 10.1109/ACCESS.2019.2952638, pp. 163757-163766, 2019.

[17] {. M. a. S. E. a. L. J.-F. a. {. T. C. a. R. A. a. C. W. a. X. T. a. L. R. a. F. S. J., "Kaolin: A PyTorch Library for Accelerating 3D Deep Learning Research," *arXiv:1911.05063,* 2019.

[18] "Kaolin example Classification," [Online]. Available: https://github.com/NVIDIAGameWorks/kaolin/tree/master/examples/Classification. [Accessed 03 05 2020].

[19] "Description about PolygonMesh message type," [Online]. Available: http://docs.ros.org/fuerte/api/pcl/html/msg/PolygonMesh.html. [Accessed 03 05 2020].

[20] "ModelNet10 Dataset source," [Online]. Available: https://modelnet.cs.princeton.edu/. [Accessed 03 05 2020].

[21] "Description about the .off file format," [Online]. Available: https://shape.cs.princeton.edu/benchmark/documentation/off_format.html. [Accessed 03 05 2020].

[22] "Description about the .pcd file format," [Online]. Available: https://www.online-convert.com/file-format/pcd. [Accessed 03 05 2020].

[23] "Description about the .vtk file format," [Online]. Available: https://vtk.org/wp-content/uploads/2015/04/file-formats.pdf. [Accessed 03 05 2020].

[24] "Description about the .obj file format," [Online]. Available: http://paulbourke.net/dataformats/obj/. [Accessed 03 05 2020].

[25] "Description about Triangle Mesh," [Online]. Available: https://www.cs.cmu.edu/~quake/triangle.html. [Accessed 03 05 2020].

[26] "Example of reading point cloud and converting it into triangle mesh," [Online]. Available: https://pointcloudlibrary.github.io/documentation/documentation/tutorials/greedy_projection.php. [Accessed 03 03 2020].