Surveyor SVS robot navigáció sztereó rekonstrukció segítségével

Mészáros Ádám[1], Megyesi Zoltán[2]

[1] Kecskeméti Főiskola Gépipari és Automatizálási Műszaki Főiskolai Kar
elrood@gmail.com
[2] Kecskeméti Főiskola Gépipari és Automatizálási Műszaki Főiskolai Kar
megyesi.zoltan@gamf.kefo.hu

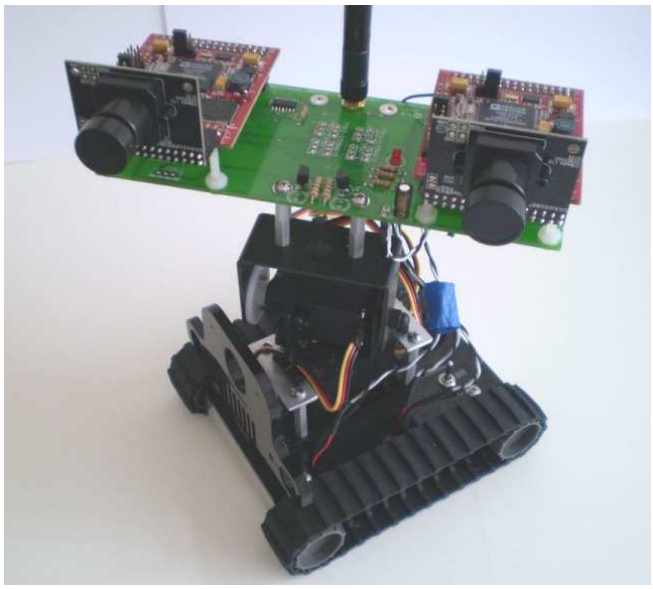
Absztrakt. Bemutatunk egy, OpenCV-ben írt, 3D rekonstrukción alapuló valós idejű robot navigációs alkalmazást, amely lehetővé teszi, hogy a Surveyor SVS robot sík terepen akadályokat tudjon kikerülni. A robot sztereó kameráinak képeit vezeték nélküli hálózaton egy személyi számítógépre továbbítottuk, ahol valós idejű sztereó rekonstrukciós módszert alkalmaztunk. A kapott 3D színtér elemzése után a robot a felállított szabályok alapján dönt, hogy melyik irányba kerülje ki az akadályokat, miközben folyamatosan frissíti a látott színteret. Mindehhez pontos kamera kalibrációra és megbízható sűrű illesztési algoritmus megvalósítására volt szükség. Az alkalmazott módszerek leírása után berendezett akadálypályán végzett kísérletek eredményeit mutatjuk be.

1. Bevezetés

A 3D színtér rekonstrukció alkalmazási területei között fontos helyet foglal el a robot navigáció. Az általunk kitűzött cél jelen esetben egy Surveyor SVS robot (1. ábra) sík terepen történő navigálása volt. Ezeknél a feladatoknál a klasszikus keskeny bázistávolságú rekonstrukció [6] alkalmazását nehezíti, hogy a robotokra szerelt kamerák sokszor gyenge optikát és alacsony felbontású szenzorokat használnak, továbbá, hogy a robotra integrált processzorok számítási kapacitása nem elég a valós idejű sűrű illesztés megvalósításához. A sebességi problémák megoldására egy lehetőség a ritka 3D rekonstrukció alkalmazása. Ilyenkor csak jól megfeleltethető jellemző pontok vagy sarokpontok illesztésére kerül sor, azonban ebben az esetben kapott ritka pontfelhőből álló színtér elemzése (elsősorban annak eldöntése, hogy van-e akadály a robot előtt, vagy sem) kevésbé megbízható. Hogy a sűrű illesztés számításigényeit leküzdjük, külső számítási kapacitást használtunk, és hatékony illesztési algoritmust kerestünk.

Az illesztési eredményként kapott sűrű pontfelhőt elemeznünk kell, hogy megtudjuk, hol helyezkednek el akadályok a robot útjában. Megalkottunk egy egyszerű döntési logikát, amely alapján a robot képes a pontfelhő alapján kikerülni az akadályokat.

Jelen cikkben először áttekintjük az alkalmazott 3D rekonstrukciós módszert, majd leírjuk a használt navigációs szabályokat. Ezek után berendezett akadálypályákon végzett kísérletek eredményeit mutatjuk be.



1. ábra: A Surveyor SVS (stereo vision system) robot elforgatott fejjel

1.1. Eszközök bemutatása

Az általunk használt Surveyor SVS roboton (1. ábra) megtalálható két darab Omnivision OV9655 1,3 megapixeles kamera, amik egymással párhuzamosan helyezkednek el 107,5 mm-es távolságra, két darab szervomotor a fej mozgatására és további két motor a gumilánctalpak hajtására. A kameramodulok nincsenek lerögzítve, az egyik oldalon a foglalat tartja, a másik oldalon csak távtartók vannak, amiken nem lehet állítani. A PC-vel a kommunikáció WLAN 802.11g-n történik. További információk az SVS-ről megtalálhatóak a gyártó honlapján [1, 2].

A képek feldolgozása és a robot irányítása a PC-ről történik. Az adatok feldolgozásához az ingyenes és nyílt forráskódú OpenCV 2.1-es verzióját [3, 5] használtuk.

2. Sztereó rekonstrukció

A megfeleltetés megállapításához 320×240-es felbontású képeket használtunk. Ennél nagyobb felbontásnál nem teljesülnek a valósidejű navigációhoz a sebesség feltételek.

A sűrű illesztés során az egyik képen szereplő összes pontnak megkeressük a párját úgy, hogy minden egyes pontot összehasonlítunk a másik kép összes pontjával, a kapott eredményt elmozdulás térképpel szoktuk ábrázolni. Ez 320×240-es felbontású képek esetén közel $6 \cdot 10^9$ összehasonlítás. Mivel elsődleges

szempont a sebesség, ezért szükség volt arra, hogy csökkenjen a keresési tér. Egy szintbe kell hozni a képeket (rektifikáció), hogy az azonos képpárok egy sorba essenek, így egy pont párját csak a vele azonos sorban lévő pontok között kell megtalálni, jelen esetben 320 közül, az összehasonlítások száma így körülbelül $24{,}5{\cdot}10^6$-ra csökken.
Ebben a fejezetben áttekintjük a 3D rekonstrukcióhoz szükséges lépéseket:

1. kamera középpontok, és képsíkok helyzetének meghatározása (kalibráció)
2. egymásnak megfelelő pontpárok keresése (illesztés)
3. kamera középpont összekötése a megfeleltetett vetülettel (trianguláció)

2.1. Kamera kalibráció

A 3D rekonstrukcióhoz szükségesek a kamerák belső és külső paraméterei, ezeket a kalibráció során kapjuk meg.

Az OpenCV-ben a `cvStereoCalibrate` függvény használatával tudjuk egyszerre kalibrálni mindkét kamerát. A függvénynek ehhez objektumpontokra van szüksége. Ezeket a pontokat egy ún. kalibrációs objektummal gyűjthetjük össze, ami egy sík, fekete-fehér négyzethálós sakktábla, melynek sarokpontjai jól meghatározhatóak. Az objektumpontok mozgását követve lehet következtetni a belső és külső paraméterekre. Ez az eljárás Zhangtól [7, 8] és Sturmtól [9] származik.

A sakktábla sarkai képenként a `cvFindChessboardCorners` függvénnyel gyűjthetőek össze, amely Harris sarokdetektálót használ. A táblát 3 szabadságfok mentén kell forgatni, hogy tudjunk 3D-ben kalibrálni. A pontosabb kalibráció érdekében szükséges minél több felvételt csinálni. Fontos, hogy a képpár mindkét képén látszódnia kell a teljes sakktáblának, hogy képpáronként azonos objektumpontok legyenek (lásd 2. ábra). A függvény bemenete 8 bites szürkeárnyala-

2. ábra: A robot bal és jobb kamerájával készített eredeti, legjobb minőségű, 320×240-es képek. Az OpenCV berajzolta a talált sarkokat.

tos kép, amin szerepel a sakktábla. A robottól 3 csatornás, csatornánként 8 bites színmélységű kép érkezik, ezt kell átalakítani a megfelelő formátumra. A

`cvCvtColor` az átalakításhoz a lumát (Y) használja, ami az R, G és B súlyozott átlagaként számolható:

$$\text{Gray}: \quad Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

A Nemzetközi Távközlési Egyesület Rádiótávközlési Szektor (ITU-R) 601-es ajánlása is tartalmazza ezt az összefüggést a monokróm kép megjelenítéséhez [4].

A `cvFindChessboardCorners` által visszaadott sarkok csak hozzávetőlegesek. Gyakorlatban ez azt jelenti, hogy csak a kamera fizikai határain belül találja meg a sarkot, ami egy pixel. Egy külön függvény szükséges ahhoz, hogy finomítsa a sarkok pozícióját subpixel pontossággal. Ehhez az eredeti szürkeárnyalatos képet és a sarkok előbb megkapott hozzávetőleges elhelyezkedését kell megadni. Az eljárás áttekintéséhez és újabb technikákért lásd Lucchese [10] és Chen [11].

A több felvételből kapott objektumpontokból `cvStereoCalibrate` meghatározza a kamerák belső paramétereit külön-külön (kamera mátrix: M, torzítási paraméter: D) valamint a két kamera kapcsolatát jellemző külső paramétereket, a forgatási mátrixot (R) és az eltolás vektort (T), opcionális esetben az esszenciális (E) és fundamentális mátrixot (F). A belső paraméterek meghatározására a függvény Zhang [7] eljárásán alapuló algoritmust, a torzítási paraméterekhez pedig Brown [12] módszerén alapuló algoritmust használ. A kalibrálás eredményét lásd a 3. ábrán.

Ha a kamerák nem mozdíthatóak és a belső paramétereik változatlanok, akkor a kalibrálást elég egyszer elvégezni, így a kapott mátrixok és vektorok később is felhasználhatóak.

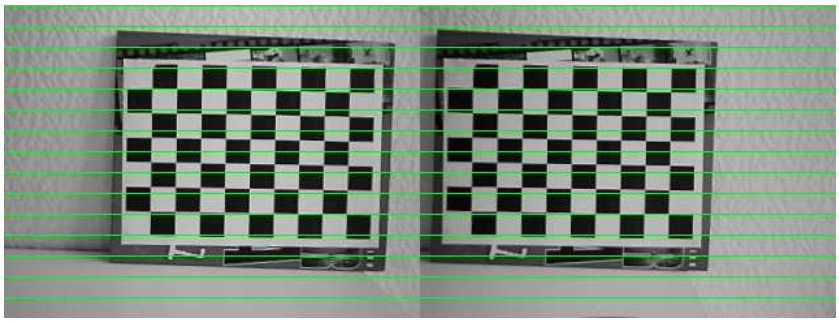
3. ábra: A kamera kalibrációja után a 2. ábrán látható képpár rektilineáris lett

2.2. Rektifikáció

Cél az, hogy a képpár egymásnak megfelelő sorai a rektifikálás után egybe essenek, így a sztereó megfeleltetés megbízható és megfelelően gyors lesz. Az OpenCV lehetőségei közül Bouguet algoritmuson alapuló függvényét használtuk

(`cvStereoRectify`), mert a rotációs mátrix (R) és az eltolási vektor (T) már rendelkezésre áll a kalibrációból. A függvénytől megkapjuk a bal és a jobb képre a rektifikált rotációs (R1, R2), vetítési mátrixokat (P1, P2) és a projektív mátrixot (Q), amivel majd a 3D-s koordinátákat ki tudjuk számolni. A paraméterben meg kell adni, hogy a képen lévő nem használt részeket ne vegye figyelembe, különben a kapott Q mátrixszal nem lehet a valós távolságokat kiszámolni. A Bouguet algoritmus az először publikáló Tsai [13] és Zhang [7, 8] eljárásának egy befejezése és egyszerűsítése.

Megvannak a képpárok, az átalakításához szükséges mátrixok, most már le kell képezni a régi képeket a rektifikáltra. A képek átalakítására a `cvRemap` szolgál, ami leképezi a régi kép pontjait egy újra (lásd 4. ábra), így megkapjuk a rektifikált képeket. A leképezéshez szükséges a képpel megegyező méretű mátrixokat a `cvInitUndistortRectifyMap`-ból nyerhetjük.



4. ábra: A 3. ábra a rektifikáció után, ugyanazon pontok egy sorba kerültek

2.3. Sztereó megfeleltetés, BM algoritmus

Az OpenCV implementált egy gyors és eredményes mintaillesztő sztereó algoritmust, a `cvFindStereoCorrespondenceBM`-t, ami a Kurt Konolige [14] eljárásán alapul.

Az illesztés egy „hibák abszolút összege” ablakot használ arra, hogy megtalálja az egyező pontokat a bal és a jobb rektifikált képen:

$$\text{SAD}: \quad \sum_{(i,j)\in W} |I_1(i,j) - I_2(x+i, y+j)| \tag{1}$$

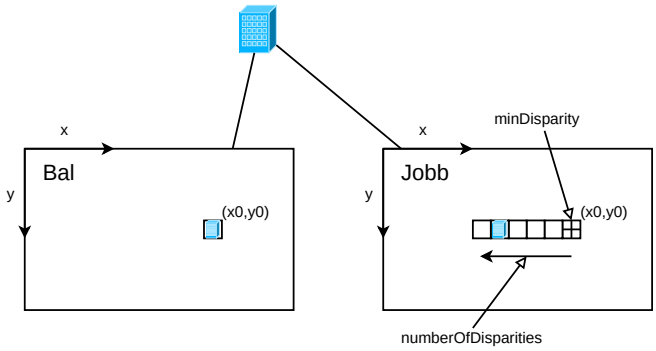
Az algoritmus csak az erősen illeszkedő (erősen textúrált) pontokat találja meg a két kép között. Így a gyengébben textúrált helyszínen, például egy folyosón kevesebb párosítást fog megtalálni.

A sztereó megfeleltetési mintaillesztő algoritmus végrehajtása három lépésben történik:

1. Előszűrés, normalizálja a kép fényességét és kiemeli a texturáltságát.
2. Egy SAD ablakkal megfeleltetéseket keres az egymásnak megfelelő sorok mentén.
3. Utószűrés, ahol törli a rosszul megfeleltetett találatokat.

Az első lépésben normalizálja a képeket úgy, hogy csökkenti a fényességbeli különbségeket és fokozza a kép textúráltságát. Ezt úgy teszi, hogy végigmegy a képen egy 5×5-ös, 7×7-es (ez az alapértelmezett) vagy akár 21×21-es (ez a maximum) ablakkal a képen. Az ablak közepén található pontot (I_c) lecseréli a (2) képletből kiszámított értékre, ahol a $\bar{I}$ az ablakban szereplő értékek átlaga, az I_{cap} pedig a megadott pozitív szám, aminek az alapértelmezett értéke 30.

$$I_c = \min\left[\max\left(I_c - \bar{I}, -I_{cap}\right), I_{cap}\right] \tag{2}$$



5. ábra: A BM algoritmus 2. lépése

Az 5. ábrán látható, mi történik a második lépésben. Egy csúszó SAD ablak segítségével számoljuk a megfeleltetéseket. Minden egyes sajátossághoz a bal képen megkeressük a legjobb egyezőséget a jobb képen ugyanabban a sorban.

A `minDisparity` paraméterrel állítható, hogy a jobb képen hol kezdje el a keresést. Segítségével a nem szükséges távoli tárgyak kiszűrhetőek. Alapértelmezetten ez az érték 0, ilyenkor a bal képen szereplő (x_0, y_0) pontot a jobb képen is ugyanabban a (x_0, y_0)-ban kezdi el keresni. Ha ez a szám negatív, akkor a jobbra, ha pozitív akkor balra tolódik a jobb képen ez a pont.

A másik paraméter a `numberOfDisparities` pedig az adja meg, hogy a keresés során maximum hány pixelt mozdulhat el a SAD ablak. A paraméternek pozitívnak és 16-tal oszthatónak kell lennie, alapértelmezett értéke 64. Kamera paramétereitől függ, de ha túl alacsonyra állítjuk be ezt az értéket, akkor a függvény a közeli tárgyakat nem találja meg.

Minél kisebb a `numberOfDisparities` és a `minDisparity` különbsége, annál kevesebb összehasonlítás szükséges, így gyorsabban jutunk eredményhez.

A megfeleltetés után a harmadik lépésben az utószűrés következik. Gyakran előfordulhat, hogy egy zajos képen a függvény egy erős központi csúcsot körülvevő részeken talál párosításokat. Ezeknek a hibás pároknak kiszűrésére az egyik lehetőség az `uniquenessRatio` paraméter. Ez egy százalékban kifejezett érték, ami meghatározza a legjobb érték (min_match) és az aktuális érték (match_val) közti maximális különbséget. Ha egy érték a (3) összefüggés alapján a paramétertől nagyobb százalékban tér el a legjobb értéktől, az törlődik.

$$\texttt{uniquenessRatio} > \frac{\text{match\_val} - \text{min\_match}}{\text{min\_match}} \tag{3}$$

A második lehetőség a `textureThreshold`, ami egy határt szab meg a SAD ablaknak, hogy a nagy csúcsokat jelentő zajos részeket ne vegye figyelembe.

A harmadik lehetőség pedig az úgynevezett folt (speckle) használata. Minden mintaillesztést használó algoritmusnál probléma lép fel, ha egy objektum határához ér. Egy lokális terület egyik részén kicsik a másik részén pedig magasak az elmozdulási értékek. A határvonal menti párok megelőzésére használhatjuk a folt detektálót.

A BM függvény végeredménye a bal képre elkészített sűrű elmozdulás térkép, az egyes pixelekhez az x-ben mért elmozdulás értéke kerül: $x_l - x_r$. Ahol nem sikerült párosítást találni, ott a pixel értéke `minDisparity` $- 1$ lesz.

2.4. Bal-jobb konzisztencia

Előzőekben megismerhettük, hogyan készül el a bal képre az elmozdulás térkép, de a megfelelő beállítások ellenére is előfordul, hogy olyan párosításokat talál, ami nem létezik az eredeti képeken. Ezeknek a hibáknak egy kiküszöbölési lehetősége a bal-jobb konzisztencia (left-right consistency, LRC), amihez ki kell számolni a bal majd a jobb képre is az elmozdulás térképet.

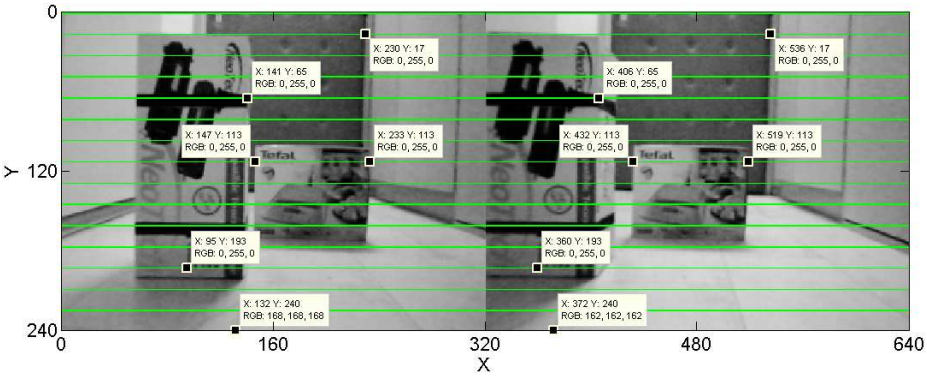
A BM függvény csak bal képre tudja megcsinálni az elmozdulás térképet, így szükséges az alapbeállítások megváltoztatása. A 6. ábrán láthatóak a kézzel mért elmozdulások, amik megegyeznek a 7. ábrán látható számolt elmozdulásokkal.

A bal elmozdulás térkép számításához a `numberOfDisparities` értékét 96-ra vettük, a `minDisparity` értéket pedig hagytuk 0-n, minden más az alapértelmezett maradt.

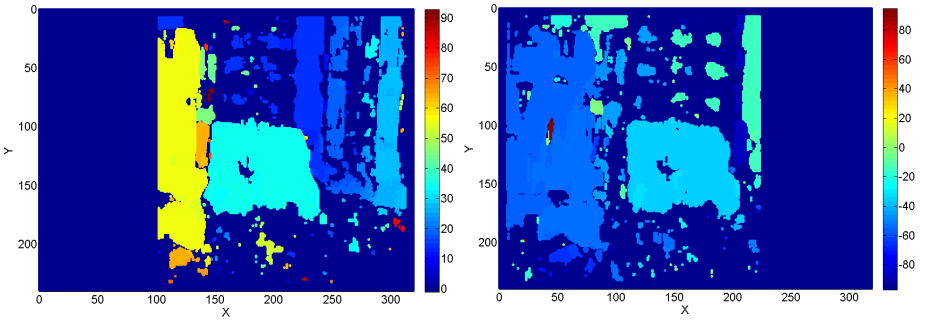
A jobbhoz a BM paramétereknél felcseréltük a jobb és a bal képet, a `numberOfDisparities` érték maradt 96, a `minDisparitity` értéket pedig -96-ra állítottuk, így az eredeti ponttól a keresést 96 pixellel jobbra kezdi el. A 7. ábrán látható, milyen elmozdulásokat számolt.

A bal és jobb elmozdulás térképek csak negatív előjelben térhetnek el egymástól. Minden egyes pixelt meg kell vizsgálni, ha a (4) egyenlőtlenség teljesül, akkor a talált párosítás jó (8. ábra).

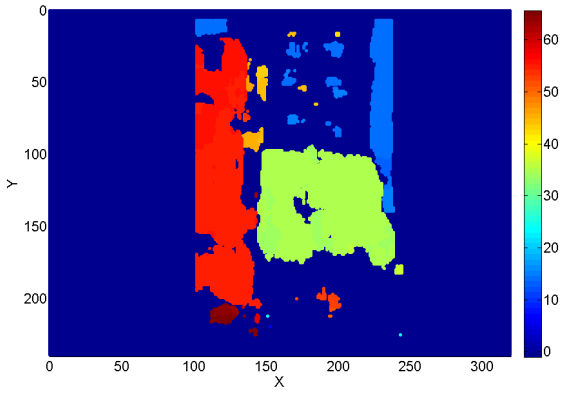
$$\text{disp\_left}(x, y) + \text{disp\_right}(x - \text{disp\_left}(x, y), y) < threshold \tag{4}$$



6. ábra: A kézzel megmért pixeltávolságok értékei. A jobb oldali képen lévő X-ek ből 320-at ki kell vonni.



7. ábra: A 6. ábra alapján a bal és jobb képekre készített elmozdulás térképek



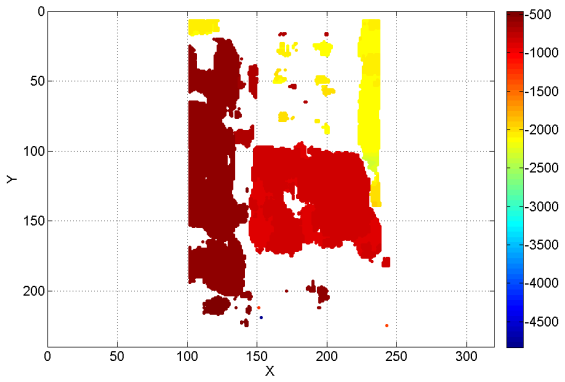
8. ábra: A 7. ábrán látható elmozdulás térképek felhasználásával a bal elmozdulás térkép LRC-vel megszűrt eredménye

2.5. 3D rekonstrukció

Miután megvan a normalizált elmozdulás térkép, 3D-s rekonstrukcióra, vagy mélységtérképre van szükség. A `cvReprojectImageTo3D` az elmozdulás térkép és a Q mátrix segítségével kiszámolja minden egyes pixelhez (x, y) az X, Y és Z koordinátákat:

$$[X\,Y\,Z\,W]^T = Q \cdot [x\,y\,\text{disparity}(x,y)\,1]^T$$
$$\text{3DImage}(x,y) = (X/W, Y/W, Z/W)$$

A kapott koordinátákból (9. ábra) csak a Z-t használtuk fel, mert egy akadály megállapításához ez is elegendő.



9. ábra: A 8. ábrán látható párosított pontok elmozdulás térképből készített 3D-s rekonstrukciónak a Z értékei mm-ben

3. Navigáció

Az előző fejezetben ismertetett, megfelelően paraméterezett eljárásokat felhasználva egy megbízható algoritmust kellett kidolgozni, amivel a robot elkerülheti az ütközést.

3.1. Mérések

A cél az volt, hogy a robot egyenletes sebességgel haladjon előre, és mikor akadályt észlel maga előtt, akkor álljon meg, válassza ki a megfelelő irányt, forduljon arra, ami megfelelő és haladjon tovább.

A robot különböző sebességekkel tud haladni, de ha túl gyorsan megy, akkor a fejnehéz robot csak mosódott képeket tud továbbítani az erős rázkódás miatt, amit nem lehet felhasználni a feladathoz. A helyes sebesség az alapértelmezettől 6 egységnyivel lassabb volt.

Teszteket végeztünk ahhoz, hogy megállapítsuk, hogy a bal képre készített 3D-s képből a Z értékeket, amik mm-ben vannak megadva, hogyan értelmezze.

Az elmozdulás térképekből megállapítható volt, hogy a robot, a távoli kamerák miatt, körülbelül fél méternél távolabb lévő tárgyaknak a képét tudja párosítani, ezért a fél méteren történő fordulás lett a cél. A -500-nál kisebb Z értékkel rendelkező pixeleket fogja számolni a program.

Meg kellett továbbá állapítani, hogy a képen szereplő pixelek közül melyek tartoznak a robottal szemben lévő térre. A méréseink szerint (lásd pl. 6. ábra) az X értékének 130 és 210 közé kell esnie, az Y-ra nem tettünk feltétel. Több tereptárggyal történő kísérlet után, ha 3500 darab -500-nál kisebb Z értéket talál a megadott képtartományban, akkor az akadálynak minősül.

Ha a `numberOfDisparities` értéke 96, akkor 0–102 közötti X-en nem lesz párosítás, 128 érték esetén pedig 0–134 pixelen.

A 8. ábrán észrevehető, hogy az LRC számolás után a kép bal és jobb szélén szinte alig található párosítás. Ahhoz, hogy a robot megállapítsa, balra vagy jobbra induljon el, el kell forgatnia a fejét. Balra és jobbra maximum 45°–45°-s szögbe tudja elfordítani a kamerákat, ezért erre a szögre kellett megállapítani azokat a motorutasítás értéket, amelyre pont ennyit tud fordulni. Ezeket az értékeket különböző padlóknál mindig újra ki kell mérni. A fejforgatás parancs kiadása után egy kis szünetet kell beiktatni, mert a mozgatás is időbe telik, ha túl hamar hív le képet a program, akkor az eredmény ugyancsak elmosódott kép lesz, amin alig található párosítás.

3.2. Algoritmus

A navigálás úgy történik, hogy miután meglettek a 3D-s adatok, a program a tartományban megszámolja, hogy mennyi Z érték túl kicsi. Ha ez a szám meghaladja a küszöböt, akkor előtte akadály van, el kell fordulni.

```
Ciklus
    Közeli pontok számának meghatározása
    Ha közeli pontok száma túl sok
        Állj
        Fejet balra, közeli pontok számának meghatározása
        Fejet jobbra, közeli pontok számának meghatározása
        Fejet egyenesbe fordítani
        Ha balra megfelelőbb, fordulj balra
        Ha jobbra megfelelőbb, fordulj jobbra
    Egyébként
        Előre
Ciklus vége
```

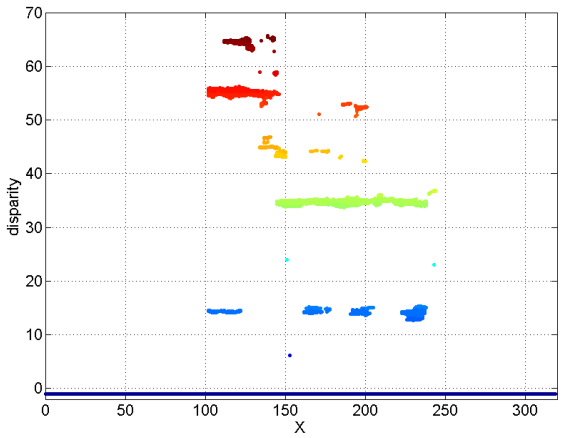
4. Eredmények

Sikerült egy olyan programot írnunk, amivel a robot ki tudja kerülni az elé került akadályokat.

4.1. Elmozdulások

A BM algoritmus segítségével sikerült megfelelő számú elmozdulást találni, így lehetségessé vált a navigáció. A közelebb lévő tárgyak elmozdulása nagyobb lett, mint a távolabb lévő tárgyaknak (10. ábra), de nem csak a távolságuk, hanem az alakjuk is kivehető (11. ábra).

A használt 96 értékű `numberOfDisparities` esetén 35 cm-re lévő tárgyat is képes volt észlelni, 128 érték esetén ez a távolság 25 cm-re csökkent.



10. ábra: A 8. ábrán látható elmozdulás térkép „fentről”, az elmozdulás felől nézve. Jól kivehető a robotra merőleges két sík doboz oldala és köztük lévő távolság

Ellenőrzésként ráhelyeztük az elmozdulás térképet az eredeti képre, hogy mennyire találta meg pontosan a dobozokat (12. ábra).

4.2. Sebességtesztek

Mivel a robot folyamatosan halad előre, ezért szükséges, hogy az akadály számolása real-time legyen. Sebességteszteket végeztünk egy szálon, majd több szálon futó programmal, hogy a robot milyen sebességgel dolgozza fel a képpárokat. A fekete képnél a robot csak feketeséget látott, így a küldött kép a lehető legkisebb volt, statikusnál egy tárgyra irányultak a kamerák. Az 1. táblázatban a másodpercenként feldolgozott képpárok száma szerepel.

4.3. Z értékek

A 2. táblázatban látható a számított, a valódi távolságok és a közöttük lévő eltérés.

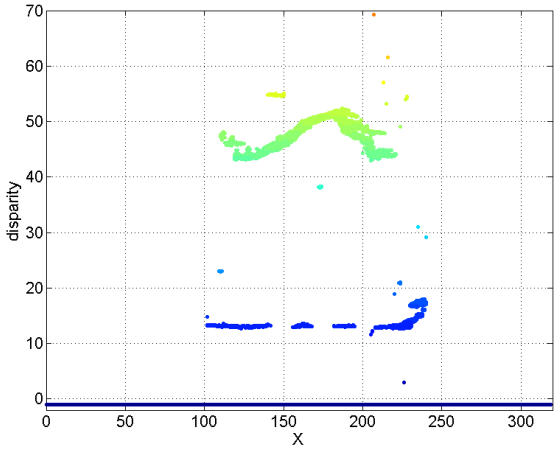
1. táblázat: Az akadályszámolás sebességtesztje képpár/másodpercben. A továbbított kép: 320 × 240, legjobb minőség.

1,6 GHz-es, egy magos CPU				
	Egy szál		Több szál	
	Fekete kép	Élő kép	Fekete kép	Élő kép
Csak kép küldés	11 FPS	3,9–4 FPS	11 FPS	3,9–4 FPS
Navigáció	6,2–6,3 FPS	2,9–3 FPS	9,9–10 FPS	3,8–3,9 FPS
3,1 GHz-es, két magos CPU				
	Egy szál		Több szál	
	Fekete kép	Élő kép	Fekete kép	Élő kép
Csak kép küldés	10–10,2 FPS	3,8 FPS	10–10,2 FPS	3,8 FPS
Navigáció	6,8–6,9 FPS	3,2–3,3 FPS	10–10,2 FPS	3,7–3,8 FPS

2. táblázat: A mért és a tényleges távolságértékek, a Z tengely ellenkező irányba mutat, ezért negatívat az értékek. A 13. mérésnél a `numberOfDisparities` értéke 128 volt, a többinél 96. 2 méter után jelentősen nő az eltérés a mért és a valós távolságok között.

Mérés	Távolság (mm)	Z értéke (mm)	Eltérés (mm)
1.	1590	−1670	80
2.	1400	−1480	80
3.	1290	−1335	45
4.	1210	−1250	40
5.	1100	−1140	40
6.	1020	−1070	50
7.	910	−940	30
8.	830	−860	30
9.	720	−745	25
10.	640	−664	24
11.	530	−543	13
12.	450	−470	20
13.	250	−275	25

bal kép



LRC-vel tisztított bal elmozdulás térkép

11. ábra: A bal oldalon látható képre készített elmozdulás térképen jól kivehető a merőleges sarok

bal jobb LRC-s bal

12. ábra: A megtalált elmozdulások ráhelyezve a 6. ábrán látható képekre

4.4. Pályabejárás

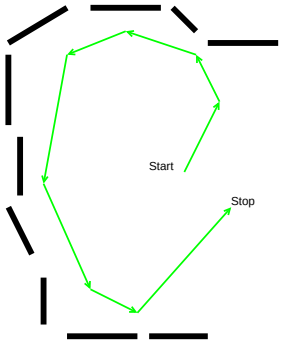
A program tudását két akadálypályán próbáltuk ki. Az egyik esetben egy félkör alakú (14. ábra) pályát kellett követnie, a másik esetben pedig egy L alakú csatornán kellett végighaladnia ütközés nélkül. Mindkét pályát teljesítette, a kihelyezett akadályokat fél méteren belül észlelte. A tesztek során kiderült, hogy a robotnak a hagyományos lámpák fénye sötétedés után nem mindig elegendő. A homogén felületű tárgyakat rendszerint nem látta meg, a pályát jól textúrált elemekből építettük fel, pl. könyvekből, mintás dobozokból.

5. Konklúzió és további lépések

Bemutattunk egy sztereó rekonstrukción alapuló robot navigációs alkalmazást, amely jól hozzáférhető elemekre épül, de számos módosítást tartalmaz. A megbízhatóság érdekében alkalmaztuk a bal-jobb konzisztencia megszorítást, és a

13. ábra: A 9. ábrát ráhelyezve az eredeti képre, a nem párosított értékek nem látszanak

Fotó a pályáról



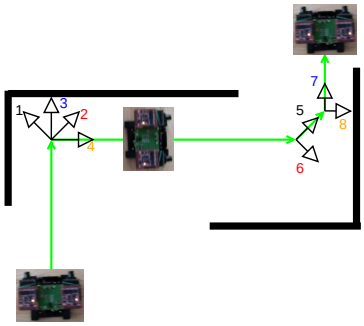
A pályán bejárt út

14. ábra: Az O alakú pálya bejárása

párhuzamos implementáció segítségével valós idejű feldolgozást értünk el. Megmutattuk, hogy a robot képes a megfelelően textúrált tárgyak határát felismerni és elnavigálni A pontból B pontba. Az elmozdulások és a 3D-s adatok nem voltak teljesen pontosak, de a biztos navigáláshoz ez is megfelelő volt.

További lépések lehetnek az erősen textúrált padlók kiszűrése, hogy minél több helyen képes legyen navigálni. Szeretnénk, ha fel tudná majd ismerni, hogy milyen akadályon tud még áthaladni. Tervezzük még, hogy mozgás közben letároljuk a 3D-s adatokat és a hozzájuk tartozó távolságokat, majd ezekből rekonstruáljuk és megjelenítjük a bejárt pályát. Másik célunk az, hogy kihasználjuk a robot mozgékonyságát arra, hogy egy helyben, minden lehetséges irányból felvett képből rekonstruáljuk a színteret.

Fotó a pályáról



A pályán bejárt út

15. ábra: Az L alakú pálya bejárása. Először 1-es és 2-es irányba néz szét, majd a 2-es irány felé fordul. Ez is túl közel van, 3-as és 4-es irányba néz szét, majd 4-es irányba fordul. Továbbhalad, itt a falat távolabbról észreveszi, így 5-ös irányba is halad egy kicsit, mielőtt újra érzékelné a falat.

Köszönetnyilvánítás

A cikk végén szeretnénk megköszönni Dr. Kovács Tamás és Pásztor Attila oktatóknak, hogy a Surveyor SVS robototot a munka idejére a rendelkezésünkre bocsájtották.

Irodalom

1. `http://www.surveyor.com/stereo/stereo_info.html`
2. `http://www.surveyor.com/blackfin/OV9655-datasheet.pdf`
3. `http://opencv.willowgarage.com/`
4. `http://inst.eecs.berkeley.edu/~cs150/Documents/ITU601.PDF`
5. Gary Bradski and Adrian Kaehler: *Learning OpenCV: Computer Vision with the OpenCV Library.* O'Reilly Media, Inc., 1st edition, October 2008.
6. Milan Sonka, Vaclav Hlavac, and Roger Boyle: *Image Processing, Analysis, and Machine Vision.* Cengage-Engineering, March 2007.
7. Z. Zhang: *A flexible new technique for camera calibration.* IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (2000): 1330–1334
8. Z. Zhang: *Flexible camera calibration by viewing a plane from unknown orientations.* Proceedings of the 7th International Conference on Computer Vision (pp. 666–673), Corfu, September 1999.
9. P. F. Sturm and S. J. Maybank: *On plane-based camera calibration: A general algorithm, singularities, applications.* IEEE Conference on Computer Vision and Pattern Recognition, 1999.
10. L. Lucchese and S. K. Mitra: *Using saddle points for subpixel feature detection in camera calibration targets.* Proceedings of the 2002 Asia Pacific Conference on Circuits and Systems (pp. 191–195), December 2002.
11. D. Chen and G. Zhang: *A new sub-pixel detector for x-corners in camera calibration targets.* WSCG Short Papers (2005): 97–100.

12. D. C. Brown: *Close-range camera calibration.* Photogrammetric Engineering 37 (1971): 855–866.
13. R. Y. Tsai: *A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses.* IEEE Journal of Robotics and Automation 3 (1987): 323–344.
14. K. Konolige: *Small vision system: Hardware and implementation.* Proceedings of the International Symposium on Robotics Research (pp. 111–116), Hayama, Japan, 1997.