Parkování v garáži

Vojtěch Přibáň
České vysoké učení technické v Praze
pribavoj@fel.cvut.cz

Richard Randák
České vysoké učení technické v Praze
randaric@fel.cvut.cz

Frederik David Albl
České vysoké učení technické v Praze
alblfred@fel.cvut.cz

OBSAH

I. ÚVOD

Cílem naší práce bylo naprogramovat poskytnutého robota. Úkolem robota bylo dojet do barevné kartonové brány bez nárazu do žádné překážky. Naše řešení nejprve kamerou mapuje okolí, poté za pomoci Dijkstrova algoritmu nalezne nejkratší cestu bez překážek a následně po ní robot dorazí do cíle.

II. ROZBOR PROBLÉMU

A. Specifikace zadání

Robot musí dojet do barevné kartonové brány. V žádném momentě se nesmí dotknout žádné překážky, ani samotné brány. Na dokončení cesty má robot neomezeně času. V případě, že robot do čehokoliv narazí a dojde ke stisku nárazníku, musí okamžitě zastavit. Do brány musí zajet tak, aby nevyčuhoval a na konci musí zahrát nějaký tón. Brána je žlutá krabice ve tvaru ⊔ s fialovými pruhy u vjezdu. Překážky jsou kartonové trubice o výšce 35 cm a průměru 5 cm a mají barvu červenou, modrou nebo zelenou.

B. Poskytnutý Hardware

Ke splnění úlohy nám byla poskytnut robot Turtlebot 2 zobrazen na obrázku 1:

Fig. 1. Turtlebot 2

Vybavení robota:

- Výpočetní jednotka Intel NUC8i7BEH
- Intel RealSense D435 hloubková kamera
- Bumper senzor
- tlačítka
- obrazovka

Dále je robot vybaven senzorem lidar a interním výpočtem odometrie, která byla využita pro pohyb robota.

C. Návrh řešení

Problém jsme nejprve rozdělili na následující části:

- Počítačová vize, která má za úkol zjištění poloh brány a překážek vůči robotu.
- Zmapování okolí robota na základě výstupu z počítačové vize. Dále nalezení vhodné cesty na základě této mapy.
- Ovládání pohybu robota.
- Vysokoúrovňové chování robota v různých situacích.

Nejprve jsme se rozhodli vypracovat samotné mapování a hledání nejkratší cesty, jelikož jsme usoudili, že se jedná o stěžejní problém dané úlohy, a v pozdější fázi by bylo obtížné tuto část dodatečně vytvářet a integrovat s ostatními prvky.

Následně jsme vypracovávali počítačové vidění a pohyb robota již se znalostmi požadavků pro mapování a tvorba projektu tak mohla proběhnout bez zbytečných zpětných úprav.

III. ŘEŠENÍ PROBLÉMU

A. Počítačové vidění

V této části problému máme k dispozici barevnou fotku a takzvaný point cloud. Point cloud lze vnímat jako fotku, na které jednotlivé složky pixelu nepředstavují barvu, ale polohu v kartézských souřadnicích relativní vůči kameře.

Cílem je zjistit relativní polohu objektů, jako jsou překážky a brána, vůči robotovi.

Nejprve z fotky zjistíme, kde se na fotce jednotlivé objekty nacházejí. Využili jsme předpokladu, že všechny objekty mají

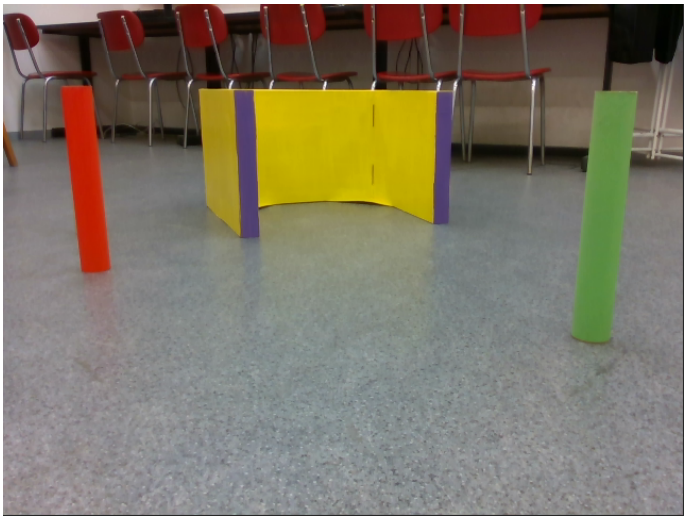

Fig. 2. Příklad výstupu barevné kamery.

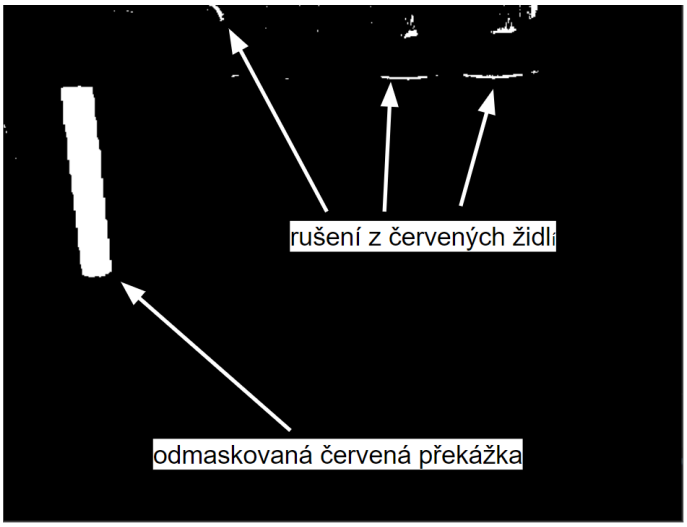

Fig. 3. Maska aplikovaná na červenou barvu u obrázku 2 s přidanými popisky.

jednoznačně dané barvy (žlutá a fialová pro garáž, červená, zelená a modrá pro překážky). Proto jsme vždy u každé fotky nejprve pro každou takovouto barvu vytvořili masku, neboli 2D pole, obsahující čísla ”1” a ”0”. Hodnota byla ”1”, pokud se v daném místě nacházela naše barva. Pixel byl považován tak, že nabývá dané barvu tehdy, když jeho jednotlivé HSV barevné složky ležely v námi určeném intervalu.

Barva	HSV1	HSV2
Modrá	[95, 110, 70]	[102, 255, 255]
Zelená	[42, 70, 70]	[75, 255, 255]
Žlutá	[22, 135, 70]	[32, 255, 255]
Fialová	[113, 50, 70]	[146, 255, 255]
Červená1	[180, 105, 122]	[180, 255, 255]
Červená2	[0, 110, 120]	[10, 255, 255]

TABLE I
HSV INTERVALY PRO JEDNOTLIVÉ BARVY

Dále jsme v této masce nalezli kontury, neboli uzavřené

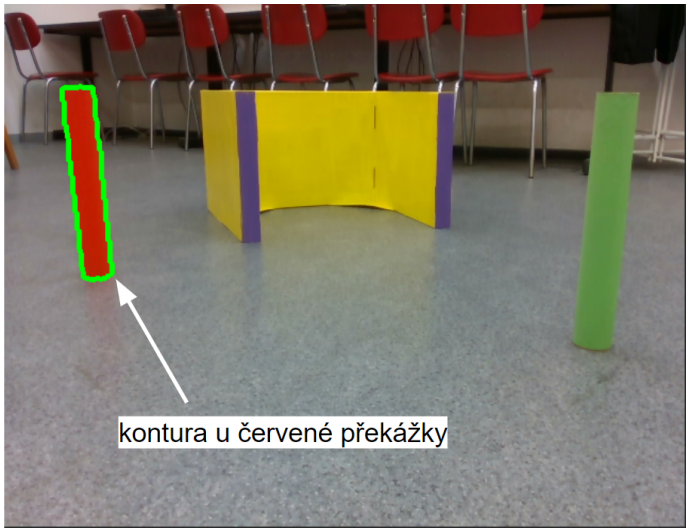

Fig. 4. Nalezená kontura okolo dostatečně velké oblasti červené barvy s popisem.

křivky, ohraničující spojité oblasti o stejné barvě. Příliš malé kontury jsme vyřadili.

Pro každý bod, který ležel v nějaké ze zbylých kontur, jsme vzali odpovídající bod v point cloudu. Z těchto bodů jsme následně počítali výslednou pozici objektu tak, že každá souřadnice objektu odpovídala prvnímu tercilu souřadnic bodů v kontuře (Volba užití tercilu vzešla z různých testů, kde tento postup získával výsledky, které podle nás nejlépe odpovídaly skutečnosti).

Po provedení tohoto postupu pro každou z barev jsme znali polohu všech překážek na fotce a mohli jsme je tak umístit do mapy.

Při hledání žlutých objektů jsme dodatečně před nalezením kontur rozdělili masku do několika vertikálních segmentů. Díky tomu se vytvoří pro jedinou žlutou oblast více objektů. Stejné segmentování bylo použito při hledání objektů o nespecifikované barvě (šedé objekty). Na ty je dále ještě aplikována maska, která zabraňuje nalezení šedého objektu pod určitou výškou. Filtruje se tak podlaha. Rozdělení do vertikálních segmentu lze vidět na obrázku 5.

B. Mapování

Mapování, neboli počítačovou reprezentaci prostoru, jsme se snažili co nejvíce zjednodušit, ale zároveň zachovat nezbytné vlastnosti pro možnost nalezení cesty. Rozhodli jsme se reprezentovat všechny překážky jako kružnice ve 2D prostoru (Mapě). Tato mapa dále obsahovala rozprostřené body pospojované úsečkami, které dohromady znamenaly možné cesty, kterými se robot mohl vydat. Samotný robot byl reprezentován nekonečně malým bodem.

Tato myšlenka vycházela ze skutečnosti, že je snadné vypočítat, zda-li úsečka protíná kružnici. Jsme tak schopni snadno přidat body a pak každé takové 2 body spojit jen tehdy, když mezi nimi neleží překážka. Získáme tak vlastně neorientovaný ohodnocený graf, ve kterém body představují

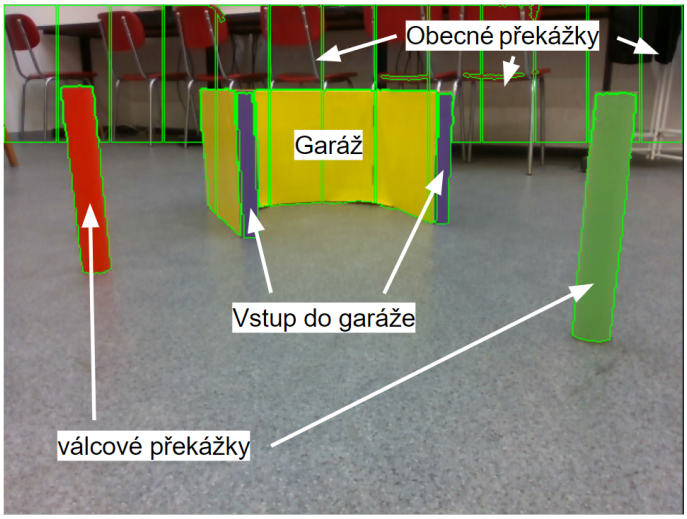

Fig. 5. Zobrazení všech relevantních nalezených kontur s popisem jejich významu.

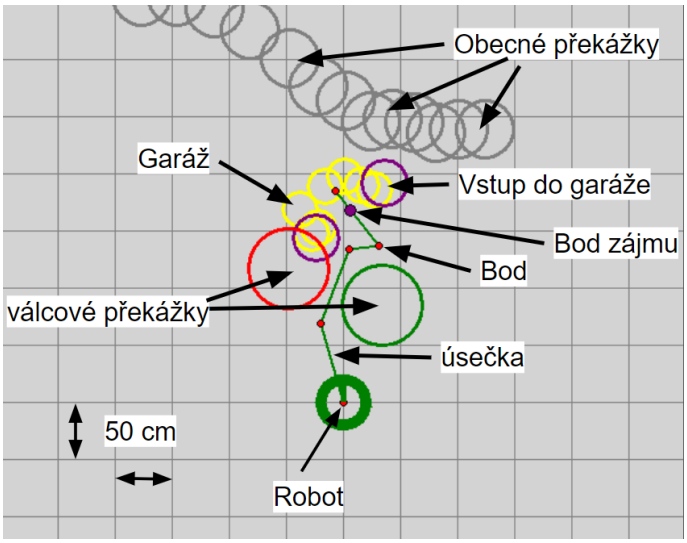

Fig. 6. Popsané zobrazení mapy vygenerované ze vstupů z kamer. Odpovídá obrázku 5.

vrcholy a úsečky propojující body představují hrany. Hrany tohoto grafu jsme ohodnotili podle euklidovské vzdálenosti spojených bodů.

Na tento graf jsme následně aplikovali Dijkstrův algoritmus [Dij59] a nalezli tak nejkratší možnou cestu v grafu mezi pozicí robota a zvoleným cílovým bodem.

C. Pohyb Robota

Poskytnutý robot byl schopen jezdit rovně dopředu a také se otáčet. Jízda tohoto robota však nebyla příliš přesná a nedalo se tak na ni příliš spoléhat. Částečného zpřesnění jsme dosáhli využitím další robotovy funkce zvané odometrie, která umožňovala odhadovat uraženou vzdálenost jak pro rovný, tak pro rotační pohyb.

1) Metoda bez využití odometrie: Robotovi lze periodicky nastavovat rychlost pohybu v nějakých časových intervalech.

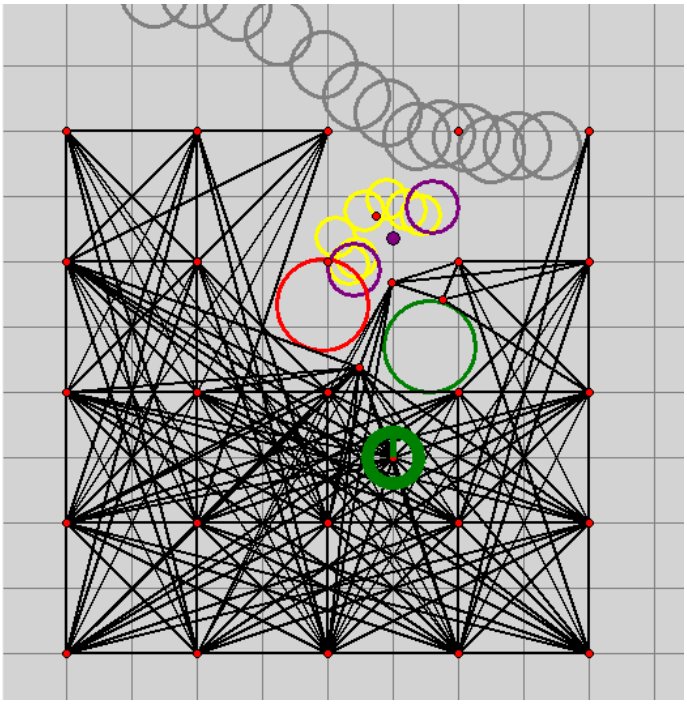

Fig. 7. Stejná mapa jako v obrázku 6 ovšem se zobrazenými všemi body a úsečkami.

Této funkcionality jsme pro uražení zvolené vzdálenosti d využili tak, že jsme nastavili konstantní rychlost v a tu robot udržoval po vypočtenou dobu t.

$$t = \frac{d}{v} \tag{1}$$

Jako maximální rychlost pohybu jsme zvolili 30 [cm/s] a rychlost otáčení 28,6 [deg/s].

2) Metoda s využitím odometrie: V této metodě, kterou jsme pak i využili ve finálním řešení, robot jede, dokud odometrií odhadovaná uražená vzdálenost nepřekročí vzdálenost žádanou. Robotova odometrie vrací odhad vzdálenosti se zpožděním, což jsme řešili přičtením konstanty k žádané vzdálenosti. Pro přesnější měření odometrie bylo zapotřebí zařídit, aby robot nezrychlil okamžitě na požadovanou rychlost, nýbrž aby plynule akceleroval. Takto nedocházelo k proklouznutí koleček při prvotním zrychlení a naměřená hodnota odometrie nebyla tímto zkreslena.

D. Vysokoúrovňové chování robota

Tato část měla za účel propojit všechny ostatní části řešení. Popisuje způsob, jakým se robot zachová v různých situacích.

Robot se vždy nejprve podívá a vytvoří mapu. V této mapě zjistí počty jednotlivých nalezených objektů, na základě kterých se dále rozhodne.

- Pokud před sebou vidí 2 fialové objekty, vytvoří v mapě vhodnou cestu vedoucí přibližně metr před tuto garáž a vydá se po ni. Poté se znovu zjistí polohu garáže a zaparkuje.
- Pokud vidí pouze 1 fialový objekt, snaží se popojet dál od tohoto objektu.

- Pokud nevidí žádný fialový objekt ale vidí alespoň 1 žlutý objekt, snaží se dojet na pozici napravo od průměrné pozice žlutých objektu. Cílem je objet bránu.
- Pokud nevidí ani fialový, ani žlutý objekt, udělá manévr, při kterém se postupně otáčí o 360°, po každých 30° se znovu podívá a vytvoří novou mapu na které vypočítá počet objektů. Nakonec se otočí směrem, kterým viděl ty nejžádanější (žádanost viděných objektů odpovídá pořadí tohoto seznamu).

Robot si také v mapě umístí bod, který představuje místo, na kterém nejvíce očekává žádané objekty. Za tímto bodem se otočí vždy, když ujede určenou vzdálenost. Díky tomu si periodicky opravuje mapu. Řeší se tím chyba vzniklá nepřesností pohybu robota.

IV. Implementace

Veškerý kód byl psán v jazyce Python 3 [VD09]. Projekt byl rozdělen do několika souborů. Každý soubor obsahuje jednu třídu. Pro počítačovou vizi jsme využili převážně knihovny numpy [Har+20] a opencv-python [Bra00]. Pro Implementaci Dijkstrovu algoritmu v hledání nejkratší cesty jsme použili knihovnu Dijkstar [Wya21]. (Tuto knihovnu se nám nepodařilo nainstalovat do robota, a proto jsme její soubory museli přidat přímo do projektu).

Pro tvorbu funkce zjišťující, zda-li úsečka protíná kružnici, jsme využili ChatGPT-3.5 Turbo (Feb 13) [Ope23]. Tuto funkci bylo potřeba pouze drobně upravit, aby detekovala jako protnutí také, když se celá úsečka nachází uvnitř kružnice (a tím pádem ji technicky neprotíná).

V. Popis experimentů

A. Kalibrace pohybu

Abychom se ubezpečili, že se pohyb robota v naší implementaci vykonává s přijatelnou přesností, použili jsme metr na měření uražené vzdálenosti. Tyto experimenty byly spíše orientační a tak jsme nevedli podrobný záznam výsledků. Iteračně jsme upravovali výpočet kalibrační hodnoty, která se přičítala k žádané vzdálenosti k uražení. Po kalibraci nedocházelo k chybě větší než 3 centimetry. Testovány byly vzdálenosti od 5 centimetrů do 4 metrů.

Pro měření úhlu otáčení robota jsme použili kompas zabudovaný v chytrém telefonu umístěný na horní část robota mimo dosah magnetického rušení elektromotorů. Opět jsme při těchto experimentech nevedli žádný záznam výsledků. Dále jsme postupovali stejně jako v předchozí části. Úhel natočení robota se po kalibraci od žádaného úhlu natočení nelišil více než o 3°. Testovali jsme různé úhly natočení v intervalu od 10° do 360°.

B. Ladění počítačového vidění

Pro ladění počítačového vidění jsme usoudili, že nebude potřeba přístup k robotovi, budeme-li mít uložená data z výstupů z kamer. Proto jsme vytvořili jednoduchý program, který nám umožnil použít robota jako fotoaparát. Robot vždy po stisku předního nárazníku vyfotil a uložil výstupy z barevné kamery a hloubkové kamery.

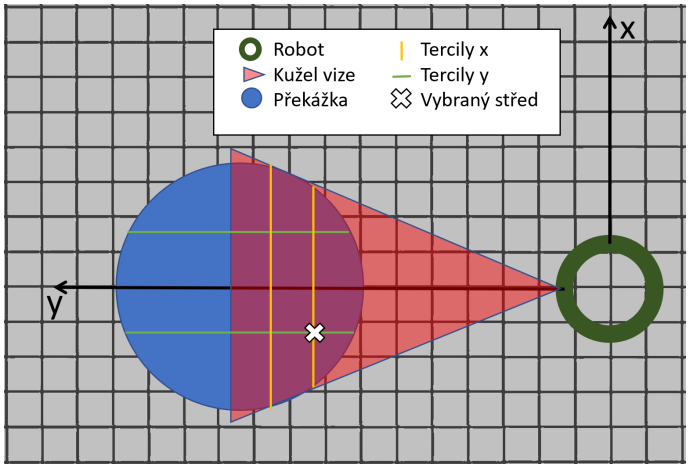

Fig. 8. Schématické znázornění vybrání středu objektu za pomoci tercilů

Tyto fotky byly ukládány objektovou serializací, abychom později měli přístup k datům ve stejném formátu, jaký vracely kamery. Fotky byly ve formátu Numpy pole. Nebyla na nich provedena žádná komprese. Z tohoto důvodu zabíraly poměrně hodně místa v paměti. Vytvořili jsme přibližně 60 fotek v různých místech, s různými objekty a v různém osvětlení. Fotky v námi zvoleném formátu uložení dohromady zabíraly v paměti přibližně 500 megabytů.

Z těchto fotek bylo potřeba zjistit vhodné intervaly hodnot pro jednotlivé barvy důležitých objektů.

Barvy jsme reprezentovali v HSV reprezentaci [Wik23]. Z důvodu limitace HSV reprezentace bylo potřeba pro červenou barvu vytvořit 2 intervaly.

Po vytvoření masek bylo generování kontur zajištěno knihovní funkcí findContours() z knihovny opencv-python [Bra00].

Dále bylo potřeba vykonat ladění určování pozic objektů za pomoci množiny bodů v kontuře a point cloudu. Metodou ”pokus omyl” jsme dospěli k tomu, že je nejlepší hledat pozici jako první tercil ze všech pozic v kontuře. Po pozdější úvaze jsme pak usoudili, že pro horizontální souřadnici by bylo lepší brát průměrnou pozici, jelikož první tercil způsoboval drobnou asymetrii v nalezených pozicích. Lépe zobrazeno na obrázku 8. Ve výsledku tak byly všechny nalezené objekty o několik centimetrů posunuty doprava od skutečné pozice. Toto jsme si dříve neuvědomili a řešili to dodatečným posunutím všech nalezených objektu o 5 centimetru doleva (z pohledu robota).

C. Ladění mapy

Každý objekt je v mapě reprezentován kružnicí s daným poloměrem a pozicí. Jelikož jsme znali poloměr robota a překážek (změřením), mohli jsme napevno určit poloměr těchto překážek. Poloměr červených, modrých a zelených překážek jsme nejprve určili jako součet poloměru robota a poloměru překážky (minimální vzdálenost středu robota a středu překážky taková, že se nedotýkají). Později jsme k tomuto rozměru ovšem přičetli několik centimetrů pro zajištění většího odstupu od překážek. Zvolený poloměr byl tedy 35

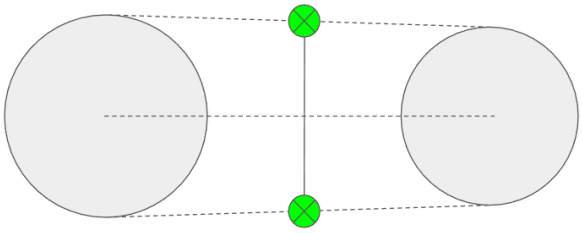

Fig. 9. Schématické znázornění nalezení 2 bodů pro 2 objekty.

centimetrů. Velikost žlutých, fialových a šedých objektů jsme nastavili spíše orientačně postupně 15 centimetrů, 20 centimetrů, 25 centimetrů. Pro volbu těchto hodnot jsme neměli speciální odůvodnění.

Dále bylo potřeba vhodně umístit body po kterých robot cestoval. Zvolili jsme 2 metody. První umístí několik bodů pravidelně do čtvercové mřížky okolo robota. Druhá metoda umístí vždy 2 body mezi 2 překážky tak, že tyto body leží na průniku osy úsečky spojující střed objektů a společnými tečnami objektů (takovými tečnami, které se neprotínají mezi objekty). Nalezení 2 takových bodů je znázorněno na obrázku 9, kde 2 šedivé kružnice představují objekty a 2 zelené kružnice body.

Pokud víme, že se tyto objekty neprotínají v žádném bodě, pak lze po těchto bodech mezi objekty bezpečně cestovat. Z tohoto důvodu jsme tuto metodu zvolili. Velmi se osvědčila při orientaci mezi překážkami. Ve výsledku jsme použili sjednocení těchto 2 metod. Dále se vytváření bodů dá vidět znovu na obrázku 7, kde jsou naše body zobrazeny červeně.

D. Zobrazování

Pro testovaní funkčnosti počítačového vidění a mapy jsme vytvořili třídu, která umožňuje zobrazit vytvořenou mapu. Tento program využívá knihovny Tkinter [Lun99]. Grafická podoba tohoto zobrazení je vidět na obrázcích 6 a 7. Zhotovení tohoto nástroje velmi usnadnilo veškeré testování.

Pro testování počítačové vize jsme si také zhotovili několik funkcí, které zobrazují jednotlivé fáze hledání objektů. K tomuto účelu jsme použili knihovnu opencv-python [Bra00]. Výstup tohoto zobrazení je vidět na obrázcích 3, 4 a 5.

Dále by stálo za zmínku, že jsme si pro pomoc s hledáním vhodných HSV intervalů přidali k tomuto zobrazování funkci, která vypsala parametry pixelu, na který uživatel kliknul. Tato funkce také výrazně urychlila celý proces.

E. Testování celku

Při testování celku jsme byli opakovaně schopni demonstrovat, že je robot schopen úspěšně z fotky zmapovat okolí a nalézt vhodnou cestu do garáže, pokud vjezd do garáže v této fotce vidí. Při této cestě se také umí vyhnout překážkám. Po delší uražené trase se ovšem začne projevovat nepřesnost pohybu robota a proto jsme na základě těchto testů implementovali korekční chování popsané v kapitole III-D.

VI. Diskuse

Velká výhoda našeho přístupu je taková, že po malých úpravách bychom měli mýt zhotovené i 3. zadání úlohy. Tedy to, kde se parkuje s překážkami.

Jako největší nedostatek robota vnímáme návrh vysokoúrovňového chování robota. Domníváme se, že existuje mnoho různých situací, při kterých by se robot nezachoval podle našich představ.

Implementované počítačové vidění častokrát nezaznamená přítomnost židlí. Pohyb robota by šel podle našeho názoru učinit jak rychlejším, tak i přesnějším než co umožňuje naše implementace.

Časová složitost algoritmu na hledání nejkratší cesty rychle narůstá s počtem přidaných bodů. Proto by vhodné vytvořit efektivnější způsob umisťování těchto bodů do mapy.

VII. Závěr

Seznámili jsme se s funkčností robota a navrhli jsme program na detekci garáže a překážek. Dále jsme vytvořili nástroje na hledání nejkratší cesty do garáže. Nakonec jsme implementovali pohyb robota. Robot je v mnoha situacích schopen úspěšně dorazit do garáže a splnit tak zadání. Jsme si však vědomi mnoha nedostatků v našem řešení, ze kterých se poučíme.

Zdroje

[Bra00] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[Dij59] Edsger W Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[Har+20] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[Lun99] Fredrik Lundh. “An introduction to tkinter”. In: *URL: www. pythonware. com/library/tkinter/introduction/index. htm* (1999).

[Ope23] OpenAI. *ChatGPT-3.5 Turbo (Feb 13 version)*. https://chat.openai.com/chat. [Online; accessed 27-February-2023]. 2023.

[VD09] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[Wik23] Wikipedia. *HSL and HSV — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=HSL\%20and\%20HSV&oldid=1149339709. [Online; accessed 20-April-2023]. 2023.

[Wya21] Seth Junot Wyatt. *Dijkstar*. https://github.com/wylee/Dijkstar. 2021.