ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

BAKALÁŘSKÁ PRÁCE

Systém na ovládání miniaturních robotů

Autor: Jiří Šedivec **2020**

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická

Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jiří ŠEDIVEC**
Osobní číslo:	**E17B0098P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a telekomunikace**
Téma práce:	**Systém na ovládání miniaturních robotů**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování

1. Zpracujte rešerši na plánování trajektorie pohybu objektu v ploše.
2. Navrhněte program na komunikaci se zařízením pro manipulaci minirobotů.
3. Vytvořte základní algoritmus plánování trajektorie pohybu minirobotů.
4. Vytvořte uživatelské rozhraní pro ovládání vytvořeného systému.
5. Ověřte navržené algoritmy experimentem.

Rozsah bakalářské práce:	**30 – 40 stran**
Rozsah grafických prací:	**podle doporučení vedoucího**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam doporučené literatury:

1. M. Jurík, J. Kuthan, J. Vlček and F. Mach.: Positioning Uncertainty Reduction of Magnetically Guided Actuation on Planar Surfaces. IEEE International Conference on Robotics and Automation (ICRA), 2019.
2. J. Kuthan, M. Jurík and F. Mach.: Magnetic Actuation of Multiple Robots by the Coplanar Coils System. International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS), 2019.
3. M. Jurík, V. Šmídl, J. Kuthan and F. Mach.: Trade-off Between Resolution and Frame Rate for Visual Tracking of Mini-robots on Planar Surfaces. International Conference on Manipulation, Automation and Robotics at Small Scales (MARSS), 2019.
4. O. Takahashi, R. J. Schilling: Motion planning in a plane using generalized Voronoi diagrams. IEEE Transactions on Robotics and Automation, 1989.
5. J. Bruce, M. Veloso: Real-Time Randomized Path Planning for Robot Navigation. International Conference on Inteligent Robots and Systems (IROS), 2002.

Vedoucí bakalářské práce:	**Ing. Martin Jurík** Katedra teoretické elektrotechniky
Datum zadání bakalářské práce:	**4. října 2019**
Termín odevzdání bakalářské práce:	**11. června 2020**

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 4. října 2019

Abstrakt

Cílem práce bylo prostudovat možnosti plánování trajektorie pohybu objektu v ploše, vytvořit uživatelské rozhraní pro ovládání již vytvořených modulů a naprogramovat základní algoritmy hledající optimální trajektorii v ploše. V době zadání této bakalářské práce bylo ovládání modulů limitované. Řízení bylo prováděno pomocí počítačového terminálu.

Pro vytvoření algoritmů, řešící problematiku hledání cesty, bylo použito objektově orientované programování ve skriptovacím jazyce MATLAB. Grafické uživatelské rozhraní bylo vytvořeno v prostředí App Designer nástroje MATLAB, které je určeno pro tvorbu webových a desktopových aplikací. Výsledkem řešení je okenní aplikace využívající několik druhů algoritmů hledajících optimální cestu v ploše. Aplikace dále umožňuje uživateli řídit miniaturní roboty a několik podpůrných prvků ovládající celou platformu (webkamera, zdroj elektrického napájení, LED osvětlení).

Klíčová slova

Plánování trajektorie objektů v ploše, řízení miniaturních robotů, Dijkstrův algoritmus, A star algoritmus

Abstract

Objectives of this bachelor thesis were to study options of object path planning in the area, develop a user interface for miniature robot control and program path planning algorithms. At the time when the thesis was assigned, the robot control was very limited. All the tasks were performed using a computer terminal.

Object oriented programming in the MATLAB scripting language was used to program the path planning algorithms. The graphical user interface was developed using App designer, which is a development tool of MATLAB, used for web and desktop applications development. Result of this thesis is a desktop application, which uses several types of path planning algorithms. The application allows user to control miniature robots and other supporting elements (webcam, power supply, LED lights).

Key words

Planning the trajectory of objects in the area, miniature robots control, Dijkstra algorithm, A star algorithm

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této bakalářské práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

.....

podpis

V Plzni dne 16.6.2020 Jiří Šedivec

Poděkování

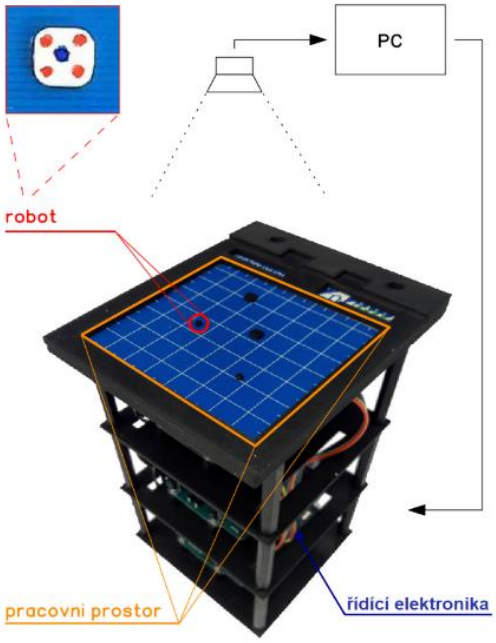
Tímto bych rád poděkoval vedoucímu bakalářské práce Ing. Martinovi Juříkovi za cenné profesionální rady, metodické vedení práce a možnost častých osobních konzultací.

Obsah

Úvod

Motivací pro vznik této práce byly nedostatečné možnosti ovládání stávajícího systému. Tento systém obsahuje několik již vytvořených modulů (viz Obr. 1) a další podpůrné prvky (webkamera, elektrický zdroj a LED osvětlení). Další motivací bylo neexistující uživatelsky přívětivé rozhraní, které by usnadnilo komunikaci se zmíněnými moduly. Na tomto základě byla vyvinuta okenní aplikace *MagNet* (2.2). Aplikace umožňuje automatickou detekci (2.4) a separátní ovládání modulů, a to několika způsoby (2.2.1). Do aplikace byly implementovány tři algoritmy sloužící pro hledání trajektorie v ploše (1.2). Tyto algoritmy lze ovládat několika hardwarovými zařízeními (2.5). MagNet také poskytuje možnost ovládání zmíněných podpůrných prvků celého systému.

Text je rozdělen do dvou částí. První část se zabývá teoretickou rešerší dostupných algoritmů hledajících cestu v ploše (1.2) a možným způsobům ovládání stávajícího systému jak z hardwarového, tak ze softwarového hlediska (1.3). Druhá část práce uvádí praktickou realizaci. V této části lze najít podrobný popis vytvořené aplikace (2.2). Dále jsou zde uvedena řešení problematiky optimalizace trajektorie objektů v ploše (2.3), kde implementované algoritmy byly prověřeny několika testy. Následuje popis principu komunikace s jednotlivými moduly. Poslední dvě podkapitoly této části se věnují – 1) možnostem ovládání aplikace pomocí hardwarových prvků, 2) řízením podpůrných zařízení pro správnou funkci platformy. Závěr práce je věnován možnostem dalšího rozšíření.



Obr. 1 – Schéma již vytvořeného modulu pro manipulaci s miniroboty

1 Teoretický úvod

Práce je zaměřena na problematiku plánování trajektorie objektu v ploše. Tuto problematiku nejčastěji řeší výpočetní algoritmy, se kterými se lze setkat jak v běžném životě (např. navigační systémy, herní průmysl, směrování paketů v počítačových sítích), tak ve speciálních aplikacích (robotické vozítko). S plánováním trajektorie je úzce spjatý problém hledání nejkratší cesty. Tento problém se řadí do grafový úloh, jejichž cílem je nalézt spojitou cestu v zadaném grafu mezi uzly *A* a *B*.

Jako nejkratší cestu považujeme tu, která má nejkratší možnou délkou. Velmi často jsou ale důležité i další parametry cesty (rychlost nalezení, rychlost vykonání, bezpečnost nebo energie potřebná k jejímu vykonání – často označována jako cena). Při výběru algoritmu se musí často zohlednit jaké parametry jsou pro danou aplikaci stěžejní. Většina algoritmů hledajících trajektorii je řešena teorií grafů. V této problematice platí, že mezi dvěma libovolnými vrcholy lze nalézt alespoň jednu cestu, která je tvořena konečným (potažmo nekonečným) počtem hran.

1.1 Dělení algoritmů

Algoritmy lze dělit různými způsoby. Jde jen o jiný pohled na danou problematiku. Jeden algoritmus se tedy nachází v několika kategoriích. V této kapitole jsou vybrány kategorie, které byly zohledněny v další části práce.

1.1.1 Počet dimenzí

Většina algoritmů je obecných, a proto je lze implementovat v libovolném prostoru. Nejčastěji používaným prostorem je dvoudimenzionální (2D). V některých případech, kde je zapotřebí pracovat ve větších prostorech (nejčastěji trojrozměrný 3D), je možné provést aproximaci do 2D zavedením rozdílných cen za stejnou vzdálenost trasy. Výhody použití algoritmů ve 2D jsou nižší výpočetní nároky (často tedy vyšší rychlost algoritmů) a snadnější implementace. V některých aplikacích je ovšem práce ve 3D prostoru vyžadována, např. robotické vozítko v členitém terénu.

Jak již bylo řečeno v úvodu, tato práce se zaměřuje primárně na pohyb v ploše. Proto zde jsou popsány principy algoritmů pracujících ve 2D prostoru. V praktické části práce je následně ukázána jejich tvorba a implementace v aplikaci.

1.1.2 Využití heuristiky

I přesto, že většina algoritmů využívá heuristiku, nemusí být tento způsob rozhodování vždy výhodou. Heuristické rozhodování chápeme tak, že algoritmus bere v potaz nově získané, resp. vypočítané, informace (nejčastěji vzdálenost do cíle z právě kontrolovaného uzlu) již při hledání cesty a na jejich základě může uzpůsobit další postup hledání. Časová složitost algoritmu je závislá na použité heuristice. Nejhorší možný případ by znamenal exponenciální počet prozkoumaných uzlů k délce řešení. V optimálním případě je složitost polynomiální. Algoritmy, nevyužívající toto rozhodování, vykonají menší počet výpočtů (o heuristickou složku) avšak ta může hrát klíčovou roli při hledání cesty a dramaticky snížit počet uzlů, které musí být zkontrolovány algoritmem. Některé algoritmy se řídí jen na základě informace získané při každém kroku (např. Greedy algoritmus). Heuristické rozhodování algoritmu nemusí být vždy výhodou.

1.1.3 Upřednostňované kritériu

V různých aplikacích požadujeme různá kritéria pro nalezenou cestu. V některých aplikacích je třeba upřednostňovat více kritérií. V problematice mobilních robotů má nejvyšší prioritu bezpečnost cesty. To znamená, dodržení minimálního odstupu od jiných objektů, které by způsobily kolizi. Další požadavky jsou, aby byla cesta, pokud možno nejkratší a zároveň nejplynulejší. Prakticky není možné splnit všechna kritéria maximálně. Vždy se jedná o jejich kompromis.

1. **Nejkratší cesta** – Výpočet nejkratší délky cesty ve 2D prostoru je založen na Eukleidovské metrice (označováno také jako *L1 norma).* Vypočteme vzdálenost vždy dvou, po sobě jdoucích, bodů. Tím získáme délku daného segmentu. Výsledná délka cesty se rovná součtu všech vypočtených segmentů.

   Výpočet vzdálenosti mezi dvěma body:

$$dis(p_{i,}\, p_{i+1}) = \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \qquad (1)$$

   , kde p_i je první bod a p_{i+1} je bod následující. Jejich souřadnice jsou poté $[x_i, y_i]$, resp. $[x_{i+1}, y_{i+1}]$.

Výpočet celkové délky cesty je dán vztahem:

$$Length\ (p) = \sum_{i=0}^{n} dis(p_i, p_{i+1}) \qquad (2)$$

2. **Bezpečnost** – Jak již bylo zmíněno výše, bezpečnost cesty je dána vzdáleností od překážek. Máme-li překážky P = {P_1, P_2, …, P_n}. Potom minimální vzdálenost mezi segmenty (spojení dvou bodů) s_i, s_{i+1} a překážkou P_j označíme jako dis($s_i s_{i+1}$, P_j) (1). Bezpečnost cesty se zvyšuje se zvyšující se touto vzdáleností.

3. **Plynulost cesty** – Tato problematika spočívá v úhlech ohybů výsledné cesty. Pro plynulou jízdu, chceme tyto úhly minimalizovat. Nejplynulejší možná cesta, má úhel ohybu roven 0°, jedná se o přímočarou cestu. Mezi dvěma segmenty, resp. třemi body p_1 p_2 a p_2 p_3, $p_i = (x_i, y_i)$, $i \in \{1, 2, 3\}$, můžeme vypočítat úhel odchylky pomocí následující rovnice:

$$Angle = \pi - cos^{-1} \frac{(x_2 - x_1)(x_3 - x_2) + (y_2 - y_1)(y_3 - y_2)}{dis(p_1, p_2) \times dis(p_2, p_3)} \qquad (3)$$

Následně můžeme vypočítat celkovou plynulost cesty, kterou získáme aritmetickým průměrem jednotlivých úhlů:

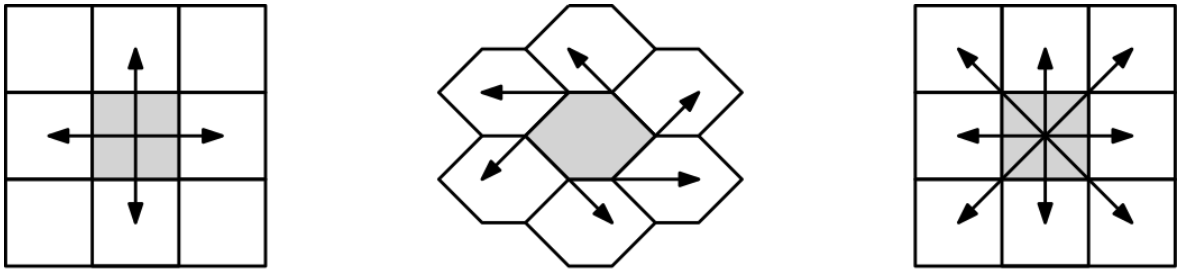
$$Smoothness\ (p) = \frac{1}{n} \sum_{i=0}^{n-1} \{Angle\ [p_i, p_{i+1}, p_{i+2}]\} \qquad (4)$$

Plynulejší cesta často vede na delší výslednou trajektorii. Pokud je třeba objet čtvercový objekt (úhel ohybu 90°), cesta je ve výsledku delší, ale kulatá. V mnoha aplikacích je však upřednostněna právě plynulost cesty (např. jízda dopravními prostředky).

1.1.4 Povolené směry

Počet povolených směrů pohybu má zásadní vliv na výslednou cestu včetně její délky. S narůstajícím počtem těchto směrů klesá výsledná délka cesty. Realizovatelný počet směrů může být závislý na konstrukci daného grafu. Graf je nejčastěji tvořen čtvercovou nebo šestiúhelníkovou mřížkou. Pro dosažení jakéhokoli místa v ploše musí mít pohyblivé

objekty možnost pohybu minimálně čtyřmi směry (2 vodorovné a 2 svislé). Pro mnohé aplikace jsou ale pouze 4 směry nedostatečné. Lehkého navýšení počtu směrů lze dosáhnout rozdělením grafu do šestiúhelníků. Zde se využívá 6 směrů pohybu. Často se lze setkat s aplikacemi umožňujícími 8 směrů pohybu, zde ke 4 základním směrům přibyly 4 diagonální. V aplikacích, kde je požadován přesný a plynulý pohyb objektů je třeba využít algoritmy umožňující pohyb teoreticky neomezeným počtem směrů. Zde jsme často limitováni hardwarovými možnostmi. Častým omezením je nutný minimální krok v daném směru (např. CNC stroje). Pohyb strojů je tedy velmi často diskrétní, i když je to pro lidské oko téměř nepozorovatelné.



Obr. 2 – Časté možnosti pohybu objektů, zleva 4 směrný, 6 směrný, 8 směrný

Pokud se mobilní objekt umí hýbat pouze 4 směry, pak výsledná délka cesty v ploše bez překážek je tzv. Manhattanova vzdálenost, kterou lze získat následujícím výpočtem,

$$d_x = |x_1 - x_n|$$

$$d_y = |y_1 - y_n|$$

$$L_{Manhatt} = D \cdot (dx + dy) \qquad (5)$$

kde x_1, y_1 jsou souřadnice startovního uzlu a x_n, y_n souřadnice cílového uzlu. Promměnná D označuje cenu za jeden krok.

Pokud je aplikace navržena pro osm směrů (viz. 4 předchozí + 4 diagonální) pohybu, pak se celková vzdálenost cesty vypočte jako tzv. *diagonální vzdálenost*. Tu lze vypočítat nasledovným způsobem:

$$L_{diagonal} = L_{Manhatt} + (D_2 - 2 \cdot D) \cdot min(d_x, d_y) \qquad (6)$$

, kde min(dx, dy) reprezentuje prvek s menší hodnotou z proměnných d_x, d_y a D_2 označuje cenu za vykonání diagonálního kroku. Tento druh se navíc dělí do dvou skupin,

které jsou dány velikostí proměnných D a D_2 z daného vzorce. Pokud $D = D_2 = 1$, pak se tato vzdálenost nazývá *Chebyshova vzdálenost*. O *oktilní vzdálenosti* mluvíme pokud $D = 1,\ D_2 = \sqrt{2}$.

Jak bylo zmíněno u předchozího kritéria, pokud aplikace umožňuje pohyb všemi směry, výsledná vzdálenost je dána Euklidovskou vzdáleností

$$L_{Euclid} = D * \sqrt{{d_x}^2 + {d_y}^2} \qquad (7)$$

1.1.5 Dynamika algoritmu

Jako dynamické algoritmy považujeme takové, které při vykonávání cesty reagují na změny prostředí, a tak mohou svou cestu změnit. Statické algoritmy vypočtou cestu při jejím zadání a pak už ji nemění. V praktické části práce byly použity pouze statické algoritmy, protože pro danou aplikaci jsou vyhovující.

Jako dva základní přístupy dynamických algoritmů se nejčastěji uvádějí: 1) *Path and Roadmap Modification* (úprava cesty, cestovní mapy), 2) *Replanning* (přeplánování). První metoda po vytvoření cesty neustále kontroluje, zda nedojde ke kolizi s pohyblivou překážkou. Pokud překážka překříží cestu robota, je okamžitě přepočítána celá cesta. V této metodě se používá například *Roadmap-Path Reshaping Algorithm* [11]. Druhá metoda po vytvoření cesty také kontroluje možnou kolizi. Pokud by k nějaké mohlo dojít, není ale přepočítána celá cesta. Pouze je přepočtena oblast, kde se překážka objevila. V této metodě se velmi často používá *D** algoritmus, který je blíže popsán v článku [12].

1.2 Popis algoritmů

V této kapitole je nastíněn princip několika algoritmů. První tři zmíněné jsou následně použity i v praktické části práce. Čtvrtý zmíněný přidává jiný náhled na problematiku. Všechny popsané algoritmy vyjma Dijkstrovo algoritmu využívají při hledání cesty heuristiku (1.1.2). Společnou vlastností pro všechny algoritmy je možnost implementace jak ve 2D, tak ve 3D prostoru.

1.2.1 Dijkstrův algoritmus

V problematice hledání cesty v grafu vychází velká část algoritmů právě z Dijkstrovo algoritmu [13]. Jedná se o jeden z prvních algoritmů pro hledání cesty v grafu. Edsger

Dijkstra byl nizozemský informatik, který roku 1959 publikoval tento algoritmus a následně (nejen za něj) získal Turingovu cenu.

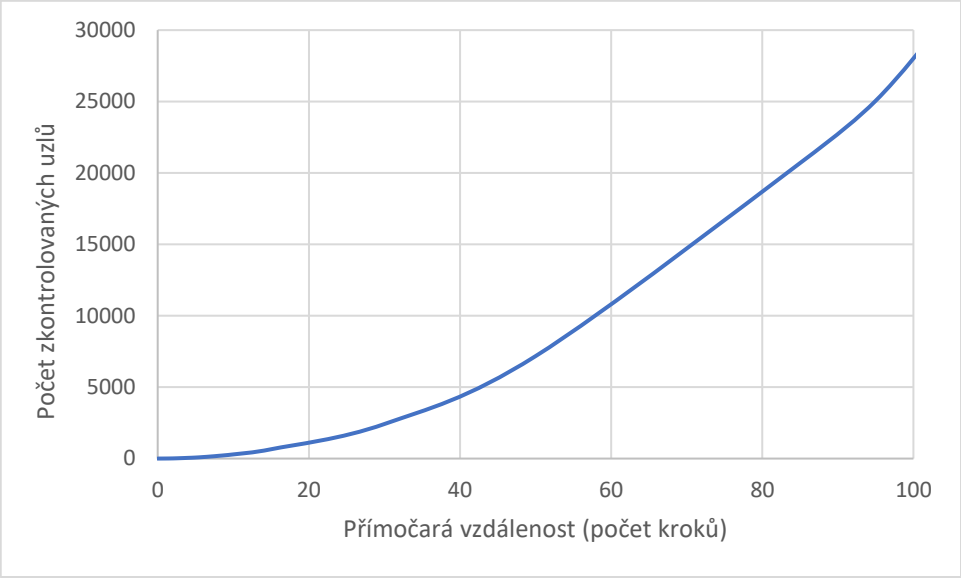
Algoritmus startuje z počátečního bodu. „Hledač“ zkontroluje všechny přímo sousedící uzly a přiřadí jim cenu často označovanou písmenem „g“ nebo spojením „gCost“. Tato cena reprezentuje např. čas, energii nebo vzdálenost potřebnou pro dosažení sousedního uzlu (v homogenním 2D prostředí je tato cena pro všechny tyto uzly stejná). Následuje uzamčení počátečního uzlu a přesunutí hledače na uzel s nejnižší cenou, pokud existuje vícero uzlů s minimální cenou, vybere se jeden (nejčastěji náhodně). Opět se zkontrolují přímo sousedící uzly s právě vybraným uzlem (kromě uzamčeného) a je jim vypočítána cena. Tento cyklus se opakuje, dokud: a) nejsou všechny uzly uzamčeny (cesta neexistuje), nebo b) byl nalezen cíl. Následující obrázek znázorňuje prohledanou oblast Dijkstrovým algoritmem. Každý uzel (v tomto případě) čtvercové mřížky má přiřazenou hodnotu gCost, která roste se vzdáleností od startovního uzlu. Pro jednoduchost jsou algoritmu povoleny pouze 4 základní směry pohybu (1.1.4).

50	40	30	20	30	40	50	60
40	30	20	10	20	30	40	50
30	20	10	Start 0	10	20	Cíl 30	40
40	30	20	10	20	30	40	50
50	40	30	20	30	40	50	60

Obr. 3 - Vizualizace přiřazování ceny uzlům čtyř směrným Dijkstrovo algoritmem (bez implementace early exit)

Mezi kladné vlastnosti tohoto algoritmu patří, že je vždy nalezena nejkratší možná cesta. Prohledání jednoho uzlu je rychlé, protože je mu přiřazována pouze jedna cena (není zde heuristický aspekt). Naopak nevýhodou je velké množství uzlů, které musí být zkontrolováno a s tím často spojený delší čas potřebný k nalezení cíle. Pokud není implementován tzv. *early exit* (algoritmus se zastaví po nalezení cíle), tak je vždy zkontrolováno n^2 uzlů, kde n reprezentuje počet uzlů jedné strany čtvercového grafu. Brzké

ukončení algoritmu ušetří velké množství nutně zkontrolovaných uzlů, toto množství je závislé na přímočaré vzdálenosti mezi startovním a cílovým bodem. Tato závislost je znázorněna v následujícím grafu.



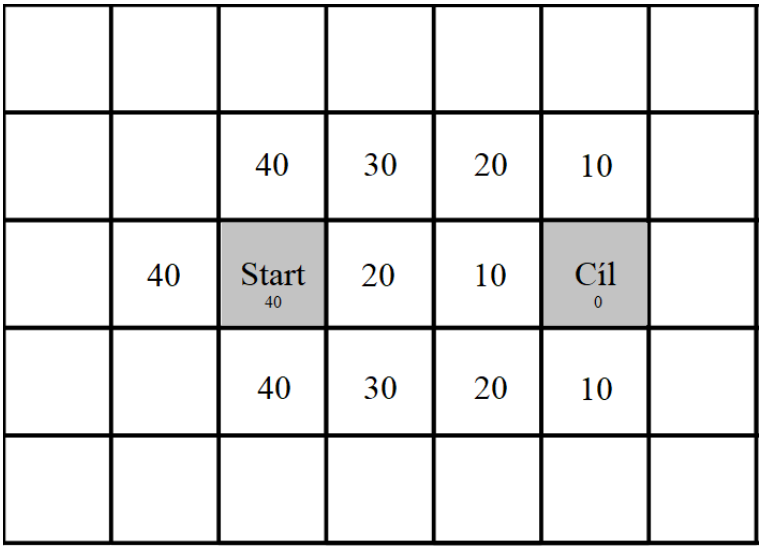
Obr. 4 - Počet zkontrolovaných uzlů závislý na přímočaré vzdálenosti startu a cíle

Z grafu lze vidět, že již při relativně krátkých vzdálenostech Dijkstrův algoritmus musí prohledat řádově tisíce, resp. desetitisíce uzlů. V tomto konkrétním případě byl použit Dijkstrův algoritmus hledající cestu v osmi směrech pohybu (1.1.4). Prohledaná oblast v tomto případě má tvar pravidelného osmiúhelníku (viz. Obr. 12)

1.2.2 Greedy algoritmus

Vychází z principu Dijkstrovo algoritmu [17]. Také probíhá kontrola všech sousedních uzlů. Rozdíl je v tom, že algoritmus nehledí na cenu, kolik ho již cesta stála, ale pouze na cenu, která zbývá k dosažení cíle, často označována písmenem „h“ nebo spojením „hCost“. Výpočet této ceny úzce souvisí s možnostmi pohybu robotů, proto je výpočet zmíněn v kapitole 1.1.4. Rozhodování algoritmu tedy probíhá v každém, právě kontrolovaném, uzlu, který z jeho sousedních uzlů je nejblíže k cíli. Jedná se tedy pouze o heuristické rozhodování. Algoritmus se ukončí jedním ze dvou možných způsobů, jaké byly zmíněny i u Dijkstrovo algoritmu – nalezen cíl nebo neexistence cesty. V grafu bez překážek je tento algoritmus velmi účinný a rychlý, protože kontroluje minimální množství uzlů. Množství prohledaných uzlů a přiřazení jejich ceny je ilustrováno na Obr. 5. Zkomplikujeme-li cestu překážkami, zjistíme, že nalezená cesta mnohdy není nejkratším

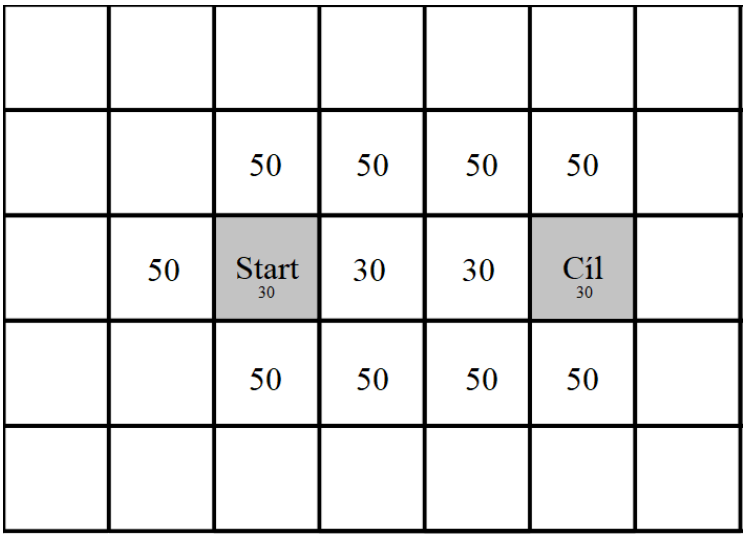
možným řešením. Ve většině případů jsme schopni nalézt cestu tímto algoritmem rychleji než Dijkstrovým algoritmem, protože není zapotřebí zkontrolovat tak velké množství uzlů.



Obr. 5 – Vizualizace přiřazování ceny uzlům Greedy algoritmem

1.2.3 A* algoritmus

Mezi velmi známý a využívaný algoritmus patří právě tento. Byl publikován roku 1968 třemi vědci ze standfordského výzkumného institutu [16]. Funguje na principu obou předešlých algoritmů, resp. se jedná o jejich kombinaci. Při rozhodování se řídí jak heuristickou složkou z Greedy algoritmu, tak na základě ceny již uražené vzdálenosti. Prohledávaným uzlům je přiřazena jedna cena označována písmenem „f“ nebo také „fCost“, která se rovná součtu již vykonané ceny a zbývající ceny do cíle. Na následujícím obrázku se lze přesvědčit, že hodnoty přiřazené jednotlivým uzlům jsou rovny součtu obou předešlých vizualizací. V grafu bez překážek je tento algoritmus opět velmi rychlý, resp. zkontroluje stejné množství uzlů jako Greedy algoritmus. Pokud graf obsahuje překážky, pak je tento algoritmus stále velmi úsporný jak v počtu zkontrolovaných uzlů v porovnání s Dijkstrovým algoritmem, tak v délce nalezené cesty v porovnání s Greedy algoritmem.

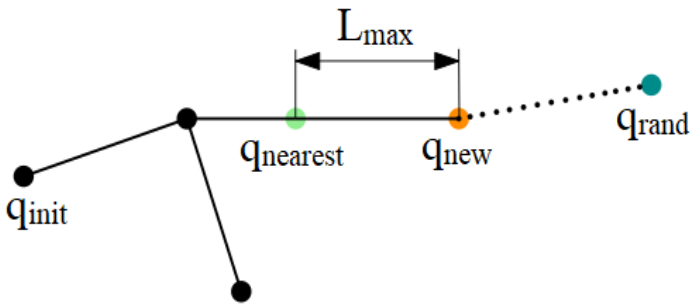


Obr. 6 - Vizualizace přiřazování ceny uzlům A algoritmem*

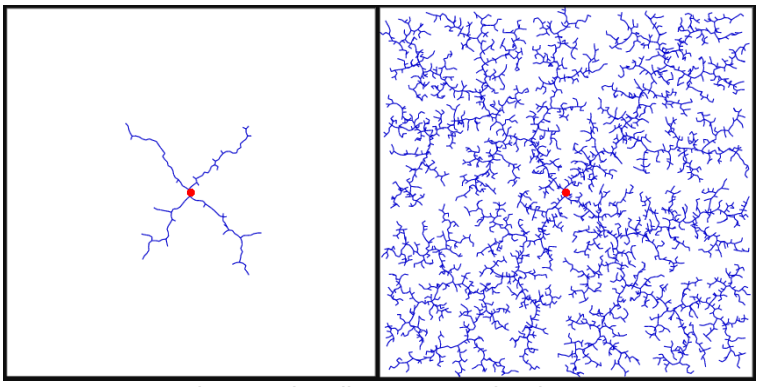
1.2.4 Rapidly-exploring random tree algoritmus

Jak již bylo zmíněno v úvodu této kapitoly, Rapidly-exploring random tree (dále jen RRT) algoritmus řeší problematiku hledání cesty naprosto jiným způsobem, než dosud bylo zmíněno [7]. Jedná se o iterační algoritmus, jehož výsledkem může být i cesta vedoucí pouze dostatečně blízko k cílovému bodu. Tento algoritmus lze najít v různých modifikacích, které jsou podrobněji popsány v [15]. Zde je popsána základní technika algoritmu.

Ze startovního bodu, často označovaným q_{init}, se vybere nejbližší bod směrem k náhodně vybranému uzlu grafu (q_{rand}). Tento nejbližší bod, označovaný jako $q_{nearest}$, je dán nastavenou maximální délkou úseku (L_{max}). Mezi startem a $q_{nearest}$ dojde k propojení (vytvoření možné cesty), pokud se mezi těmito body nenachází překážka, pak je zvolen nový $q_{nearest}$. Následuje opět vybrání náhodného bodu grafu. Je-li nový q_{rand} blíže k bodu $q_{nearest}$ než k q_{init}, tak se vytvoří možná cesta mezi $q_{nearest}$ a nově zvoleným bodem, označovaným jako q_{new}, délka tohoto segmentu je opět dána maximální délkou úseku. S přibýváním iterací se snižuje maximální délka segmentu cesty. Algoritmus se ukončí po dosažení maximálního počtu iterací, nebo po dostatečném přiblížení k cíli, případně po jeho přímém nalezení. Při zadání, kde neexistuje cesta mezi q_{init} a cílovým bodem je algoritmus ukončen po dosažení maximálního počtu iterací. Nevýhodou je, že algoritmus nikdy nezjistí informaci o neexistenci cesty. Následující obrázek naznačuje princip algoritmu v prostoru s předdefinovanými cestami (lze si představit jako již vytvořené cesty na mapě).



Obr. 7 - Princip RRT algoritmu
Obrázek převzat z [7]



Obr. 8 - Vizualizace RRT algoritmu
Obrázek převzat z [7]

1.3 Možnosti ovládání

Dalším úkolem této práce bylo promyslet a následně vybrat hardwarové a softwarové řešení ovládání algoritmů, resp. celé aplikace. Ze softwarového hlediska je kromě samotné tvorby a implementace algoritmů také důležitá jejich vizualizace uživateli. Z hardwarového pohledu je důležité, aby uživatel mohl systém ovládat jak běžným hardwarem (počítačová myš, klávesnice), tak specializovaným pro danou aplikaci (gamepad).

1.3.1 Software

První a nejjednodušší možností pro ovládání byl počítačový terminál. Tato možnost však není uživatelsky přívětivá. Algoritmy pro hledání cesty jsou realizovatelné, ale výběr startovní, cílové pozice je velmi komplikovaný stejně tak možnost tvorby překážek

Dalším řešením může být webová stránka. V tomto řešení by už byla možná implementace algoritmů hledající cestu např. v programovacím jazyce JavaScript. Výhodou této možnosti jsou téměř neomezené možnosti tvorby uživatelského rozhraní. Zároveň

uživatel nemusí vlastnit speciální software, stačí internetový prohlížeč. Díky této vlastnosti, je možný přístup i z mobilních zařízení. Nevýhodou je však nutnost nepřetržitého internetového připojení a těžko realizovatelná komunikace s hardwarovými zařízeními. Tyto nevýhody v tomto případě převažují výhody, proto bylo zvoleno jiné řešení.

Třetí alternativou byla okenní aplikace. Zde se nabízí více programovacích jazyků, ve kterých by byla možná realizace řešení. Nižší programovací jazyky, jako je např. C nebo C++, nabízí snazší komunikaci s hardwarem, avšak pro tvorbu okenní aplikace a vizualizaci algoritmů nejsou příliš vhodné. Vyšší programovací jazyky typu C# nebo Java nabízejí snadnou tvorbu uživatelského rozhraní. Složitější je práce s hardwarem (např. s webkamerou) a implementace algoritmů. Skriptovací jazyky jako jsou např. MATLAB nebo Python se nabízejí jako další možnost. Tyto jazyky nabízejí jak snadnou tvorbu okenní aplikace, tak implementaci algoritmů (využití maticového počtu v programovacím jazyce MATLAB) i dostatečnou komunikaci s hardwarem. Drobnou nevýhodou je nižší rychlost běhu programu oproti jazykům nižší úrovně, avšak v této aplikaci je tato nevýhoda zanedbatelná.

1.3.2 Hardware

Již vytvořené moduly jsou řízené programovatelnými mikrokontroléry. Ty jsou naprogramovány tak, aby podle doručených příkazů po sériové lince řídily moduly. Za tímto účelem mohou být využita různá zařízení.

Mezi hardwarové možnosti patří většina polohovacích zařízení – počítačová myš, touchpad, trackball, obrazovky citlivé na dotyk atp. Dále připadají v úvahu různá herní zařízení (joystick, gamepad) nebo textová vstupní zařízení (klávesnice).

Z každé zmíněné skupiny bylo zvoleno, do praktické části, právě jedno. Z polohovatelných zařízení byla zvolena myš pro svoji jednoduchost a dostupnost. Ze stejných důvodů byla zvolena klávesnice. Z herních zařízení byl vybrán gamepad pro přiměřené množství programovatelných tlačítek a uživatelsky přívětivé ovládání.

2 Praktická realizace

2.1 Původní stav projektu

Při zadávání této práce již bylo vytvořeno několik modulů pro řízení miniaturních robotů (viz Obr. 1). Princip ovládání modulů spočívá v silových účincích magnetického pole. Na miniaturní roboty, které mají ve své konstrukci neodymové permanentní magnety, působí magnetické pole z cívek, které se nacházejí v podložce, určené pro pohyb robotů. Tyto cívky jsou zapojeny tak, aby je bylo možné napájet nezávisle na sobě, tím pádem řídit polohu a velikost vytvořeného magnetického pole. Podrobný princip fungování je vysvětlen v článcích [18], [19].

Softwarové možnosti projektu však byly velmi omezené. Řízení robotů probíhalo přes počítačový terminál, stejně tak ovládání osvětlení a externího zdroje napájení. Za účelem vytvoření uživatelsky příjemného prostředí s možností ovládání a řízení všech prvků projektu vznikla aplikace MagNet.

2.2 Aplikace MagNet

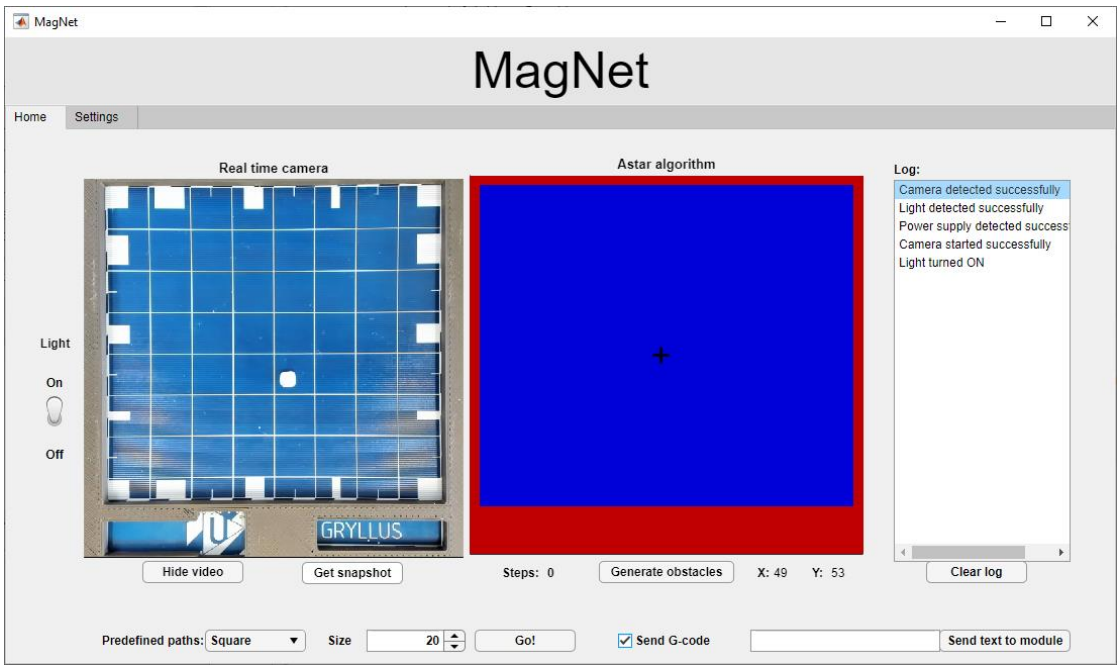
Mezi hlavní cíle této práce patřilo vytvoření uživatelského rozhraní pro ovládání vytvořeného systému (modulů). Z důvodu výhodných vlastností programovacího prostředí MATLAB byl pro vývoj aplikace využit právě tento software. Byla vytvořena okenní aplikace pomocí prostředí App Designer s názvem MagNet a zároveň naprogramovány zmiňované algoritmy hledající cestu ve 2D prostoru. V těchto algoritmech jsou velmi časté maticové operace, ve kterých je MATLAB dominantní mezi ostatními programovacími jazyky. Jednalo se také o vhodnou volbu pro sériovou komunikaci s ostatními hardwarovými zařízeními, které jsou nezbytnou součástí práce.

Vytvořená aplikace umožňuje:

- Řízení modulů
- Ovládání zdroje napětí a proudu
- Ovládání externího osvětlení
- Hledání trajektorie v ploše
- Konverzi nalezené cesty na G-kód

- Automatickou detekci připojených modulů
- Připojení a nastavení webkamery

Celé grafické rozhraní je vytvořeno v anglickém jazyce. Po spuštění aplikace se uživateli zobrazí okno o rozlišení 1100x620px. Uživatel může velikost tohoto okna měnit. Při zvětšování okna se jednotlivé prvky aplikace velikostně přizpůsobí. Problém může nastat při zmenšování, protože není možné, aby se všechny prvky vešly do jednoho okna. V horní části okna lze vidět název aplikace, pod kterým jsou dvě záložky (viz Obr. 9).



Obr. 9 - Aplikace MagNet

2.2.1 Záložka Home

Ve výchozí záložce „Home“ nalezneme hlavní komponenty aplikace. Na pravé straně této záložky se nachází log, ve kterém se hned po spuštění aplikace vypíše, zda byla detekována webkamera (viz 2.6.1). Záznamy lze jednoduše smazat tlačítkem, umístěným pod samotným logem.

Blíže ke středu okna od záznamníku lze vidět část aplikace věnující se pohybu objektů v ploše, resp. problematice hledaní cesty v ploše, která je detailně přiblížená v samostatné kapitole 2.3. Titulkem této části je aktuálně zvolený typ algoritmu. Tlačítko s nápisem „Clear obstacles“ nabízí uživateli smazat všechny vytvořené překážky. Po smazání překážek je text tlačítka změněn na „Generate obstacles“, což nabízí uživateli

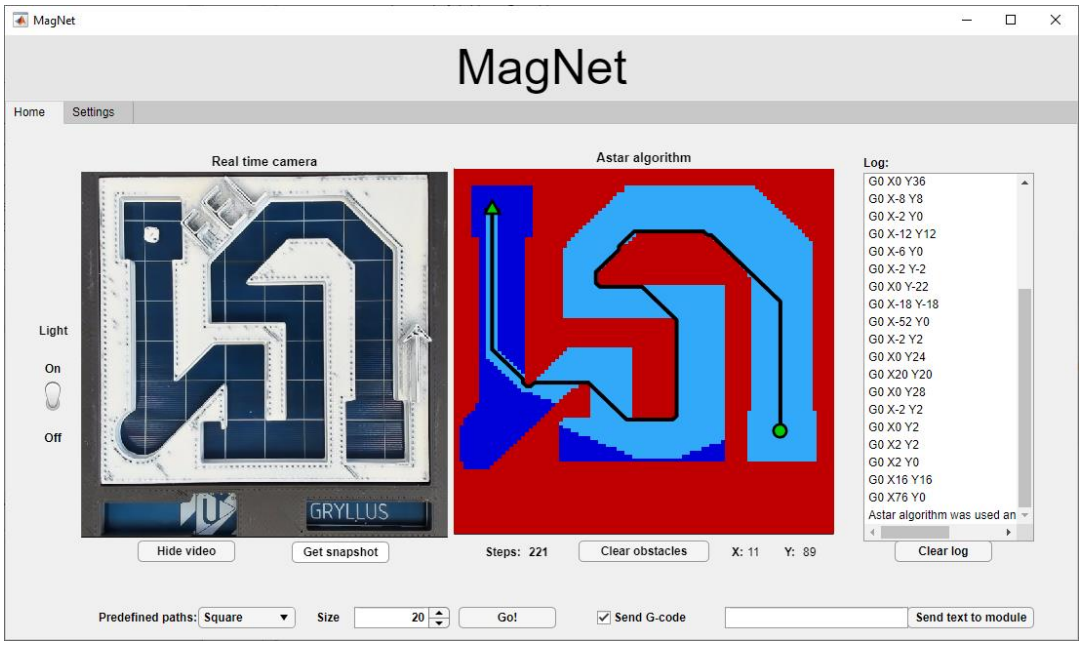
možnost automatické tvorby překážek (2.3.2). Po stranách tohoto tlačítka se nachází tři pomocné popisky. První signalizuje počet kroků po nalezení cesty. Zbylé dva udávají souřadnice kurzoru.

Část aplikace, věnující se webkameře, je umístěna blíže k levé straně aplikace. Po stisku tlačítka „Show video“ a úspěšném spuštění webkamery je zobrazeno video v místech nad tlačítkem. Zároveň je změněn text zmíněného tlačítka na „Hide video“. Po opětovném kliknutí na toto tlačítko je video zastaveno. Po spuštění kamery je odemčeno další tlačítko s nápisem „Get Snapshot“, které umožňuje uložení snímku obrazu z videa na pevný disk. Bližší informace o obsluze webkamery lze naleznout v kapitole 2.6.1.

Nejblíže k levému okraji je umístěn přepínač s označením *Light*. Tento přepínač slouží k ovládání osvětlení modulů (viz 2.6.3).

Dolní část aplikace je věnována řízení miniaturních robotů. Nejdůležitějším komponentem této části je zaškrtávací políčko „Send G-code“, které odemkne zbylá tlačítka a další komponenty. Toto políčko lze zaškrtnout pouze pokud je detekován vhodný modul (blíže popsáno v kapitole 2.4). Při jeho zaškrtnutí je spuštěn převod pohybu pravého joysticku gamepadu (2.5.2) na G-kód. Tím je umožněno přímé řízení robotů. Stejně tak je převedena nalezená cesta z předešlé části aplikace.

Dále se zde nachází tlačítko „Go!“. Po stisku tohoto tlačítka je robotem vykonána zvolená předdefinovaná cesta. Předdefinované cesty se posílají přímo v podobě G-kódu vybranému modulu. Robot tak vykoná trajektorii ve tvaru např. čtverce („Square“) nebo kruhu o zvolené velikosti (viz pole „Size“). Posledním komponentem této záložky je tlačítko s nápisem „Send text to module“, umožňuje poslat ručně napsaný G-kód do vybraného modulu.



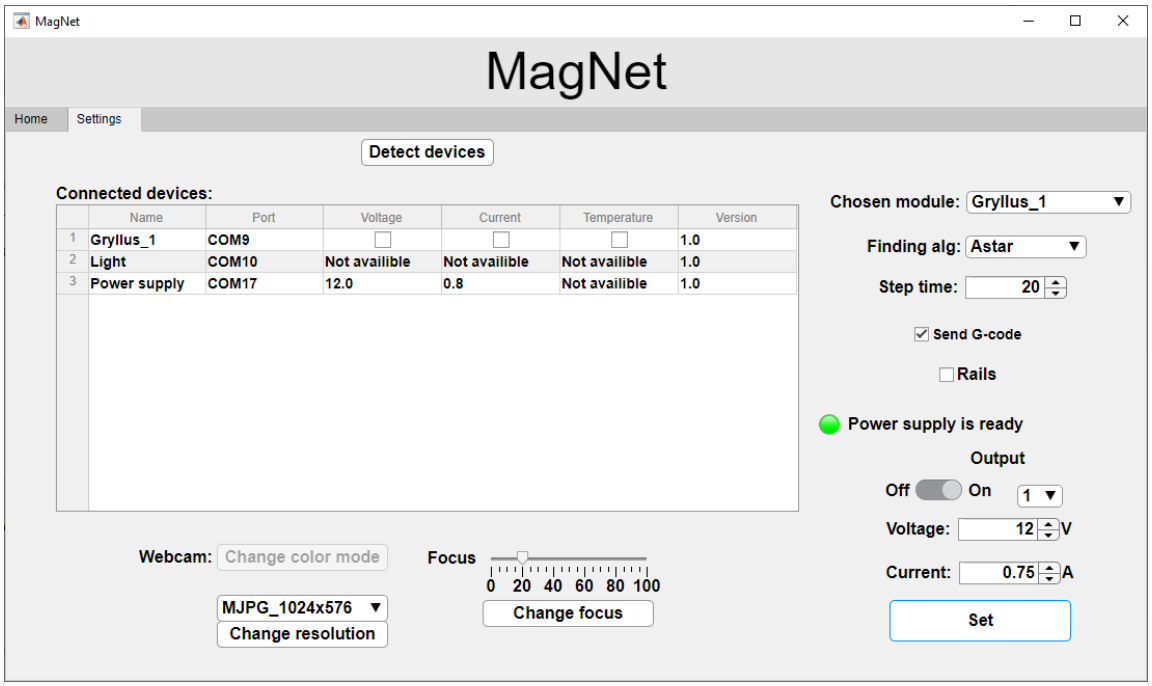
Obr. 10 - Aplikace MagNet, záložka Home

2.2.2 Záložka Settings

Po přepnutí do záložky „Settings“, obsahuje hlavní část aplikace tabulku s detekovanými moduly. Pokud připojíme, resp. odpojíme jakékoli zařízení, pak kliknutím na tlačítko „Detect devices“ aplikace detekuje všechna dostupná zařízení a vypíše je do tabulky i s jejich vlastnostmi. Bližší pohled na problematiku detekce připojených zařízení je popsán v kapitole 2.4.

V pravé části okna lze ručně zvolit z rozbalovacího seznamu modul, kterému bude posílán G-kód. Druhý rozevírací seznam nabízí výběr používaného algoritmu pro hledání cesty v ploše. Uživatel v kolonce „Step time“ může změnit dobu jednoho kroku robota (zadáváno v milisekundách). Opět se zde nachází zaškrtávací políčko „Send G-code“, jehož funkce již byly zmíněny. Poslední zaškrtávací políčko „Rails“ je pro speciální druh pohybu, který umožňuje pevně držet robota v jedné „kolejnici“.

Dále se zde nacházejí nastavení pro externí zdroj napájení. Stav zdroje indikuje barva signalizační lampy. Pokud zdroj není detekován aplikací, světlo svítí červeně a veškerá nastavení jsou pro uživatele nedostupná. Po detekci zdroje je světlo oranžové a při zapnutí jeho výstupu světlo svítí zeleně. Pod tabulkou s připojenými zařízeními se nacházejí nastavení webkamery. Uživatel zde může změnit barevný mód obrazu, změnit rozlišení dle možností kamery nebo změnit zaostření (všechna nastavení jsou blíže popsána v kapitole 2.6.1).



Obr. 11 - Aplikace MagNet, záložka Settings

2.3 Hledání trajektorie

Jak již bylo zmíněno v první části této práce, byly naprogramovány a následně implementovány tři algoritmy pro hledání cesty v ploše. Při tvorbě algoritmů musely být zohledněny již vytvořené hardwarové moduly, na kterých byly algoritmy následně testovány a dále provozovány. Implementované algoritmy umožňují 8 směrů pohybu robotů.

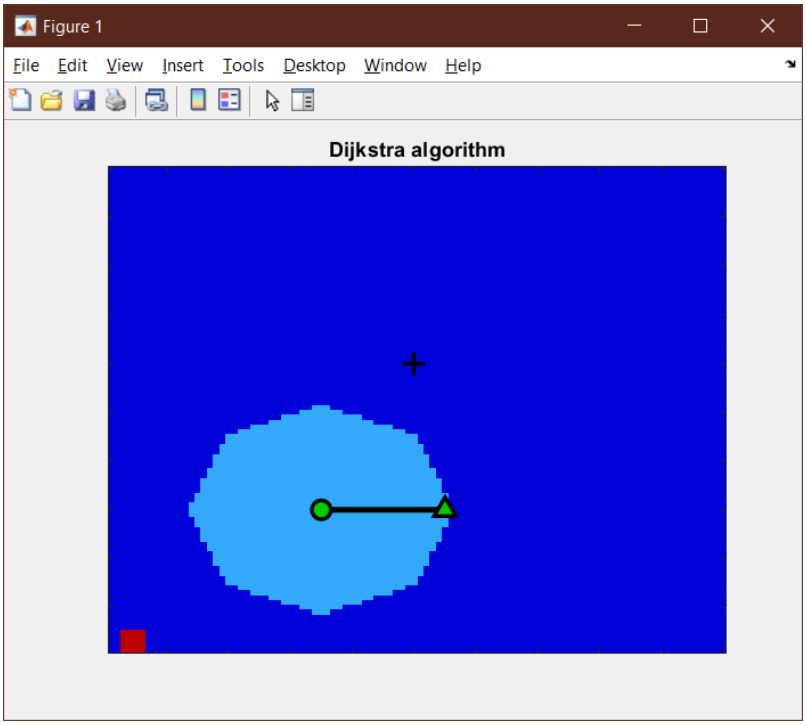
2.3.1 Pracovní plocha

Všechny vytvořené algoritmy pracují ve čtvercové matici čísel pojmenované MainMatrix obsahující 10 000 (100x100) prvků. Tato matice reprezentuje pracovní plochu (volný prostor a překážky) algoritmů, resp. miniaturních robotů. Velikost matice může být kdykoliv změněna. Tato velikost byla určena kvůli dostatečně přesnému rozlišení pro danou aplikaci. Bylo zde využito snadné práce s nekonečnem (značenými jako Inf) v programovacím jazyce MATLAB. Prvku, který je určen jako překážka, je přiřazena hodnota nekonečno. Prvky s touto hodnotou jsou pro všechny algoritmy nedostupné. Naopak dostupné prvky (volný prostor) mají výchozí hodnotu rovnou nule.

Tab. 1 - Segment matice MainMatrix s vytvořenou překážkou

0	Inf	Inf	Inf	0
0	Inf	Inf	Inf	0
0	Inf	Inf	Inf	0
0	Inf	Inf	Inf	0
0	0	0	0	0

Zobrazit celou matici, aby byla uživatelsky přívětivá, nelze praktikovat pomocí tabulky. Matice je převedena do grafického rozhraní a reprezentována třemi základními barvami (viz Obr. 12). Již vytvořené moduly mají pracovní plochu modré barvy, proto byla zvolena stejná barva pro vizualizaci dostupného prostoru. Světlejší odstín modré barvy znázorňuje prohledanou oblast algoritmem hledajícím cestu. Červená barva reprezentuje překážku. Vizualizaci dalších složek určených pro hledání cesty lze vidět na následujícím obrázku. Černý kříž symbolizuje kurzor. Zelený kruh označuje zvolený startovní bod. Cílový bod je reprezentován zeleným trojúhelníkem. Nalezená cesta je vyznačena křivkou černé barvy.



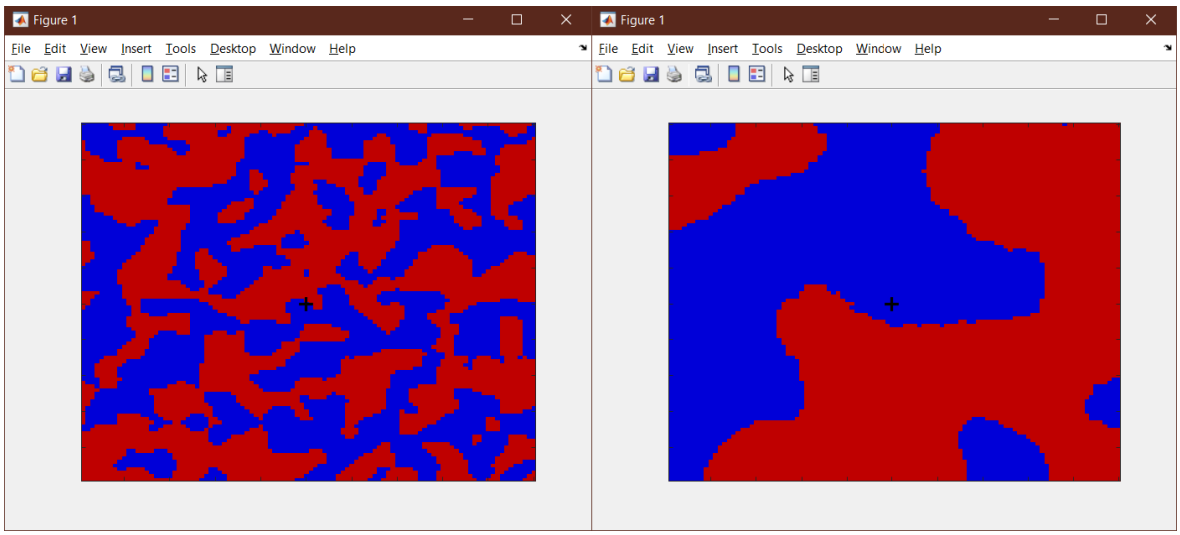
Obr. 12 - Matice MainMatrix převedena do grafického obrazce

V levé dolní části Obr. 12 lze vidět vytvořenou překážku. Jedná se o stejnou překážku, jejíž část je reprezentována pomocí Tab. 1. Na první pohled se může zdát, že je překážka vykreslena chybně, resp. opačně. To je však způsobeno pouze tím, že použitá funkce převádějící matici do obrazce přiřazuje bodu a_{11} souřadnici v levém dolním rohu, kdežto indexování matic má prvek a_{11} v levém horním rohu.

2.3.2 Tvorba překážek

Překážky v aplikaci MagNet lze vytvářet třemi způsoby. První způsob je manuální tvorba překážek. Po stisku příslušného tlačítka gamepadu, se v oblasti kurzoru vykreslí čtvercová překážka o rozměru 3x3 uzly. Další způsob tvorby překážek je jejich načtení ze souboru. Pokud existuje soubor s definovaným jménem „obstacles.mat“, tak je při spuštění aplikace převzat jako pracovní plocha. Třetí možností je použití vytvořeného algoritmu

automatické tvorby překážek. Po zavolání funkce na automatickou tvorbu překážek (stisk příslušného tlačítka na gamepadu nebo v samotné aplikaci), je vytvořena nová matice čísel o stejné velikosti jako MainMatrix (viz kapitola 2.3.1). Tato matice je vytvořena z pseudonáhodné sekvence prvků s hodnotami rovnými 0 nebo 1. Následně je na celou matici aplikován Gaussův filtr[1]. Tím jsou hodnoty elementů „rozmazány“, resp. nabývají různých hodnot mezi 0 a 1. Tyto hodnoty jsou závislé na zvolené velikosti směrodatné odchylky (označované řeckým písmenem σ) Gaussova filtru. Poté je velikost každého prvku porovnána se zvolenou konstantní hodnotou $k = 0{,}5$. Pokud je hodnota prvku vyšší než tato konstanta, je označen jako překážka, v opačném případě je označen jako volný prostor. Následně je celá matice převzata jako MainMatrix a zobrazena uživateli.



Obr. 13 - Automaticky vytvořené překážky Gaussovým filtrem s rozdílnou směrodatnou odchylkou (vlevo σ = 2, vpravo σ = 10)

Na Obr. 13 můžeme vidět vliv velikosti směrodatné odchylky u Gaussova filtru na výsledný tvar překážek. Levý obrazec byl vytvořen s 5x nižší směrodatnou odchylkou, proto je plocha více členitá. Směrodatná odchylka má také vliv na rychlost celého algoritmu. S rostoucí směrodatnou odchylkou klesá rychlost. Bylo provedeno 10 měření časů pro 8 různých směrodatných odchylek a následně vypočítán průměrný čas trvání algoritmu v Tab. 2

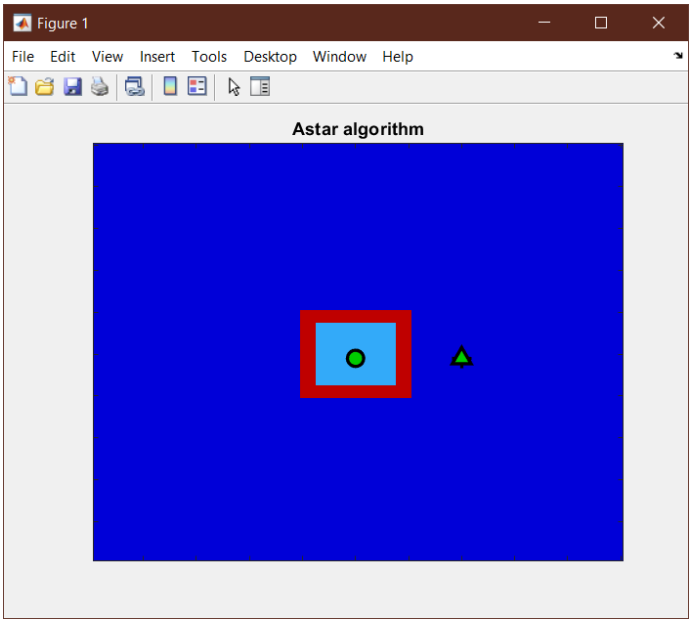
Tab. 2 - Změřené průměrné časy pro rozdílné směrodatné odchylky Gaussova filtru

σ	1	5	10	25	35	50	75	100
t [ms]	0,762	0,941	1,127	1,660	1,858	2,403	4,161	6,092

[1] Gaussův filtr je vysvětlen zde - https://en.wikipedia.org/wiki/Gaussian_filter

2.3.3 Spuštění hledání trajektorie

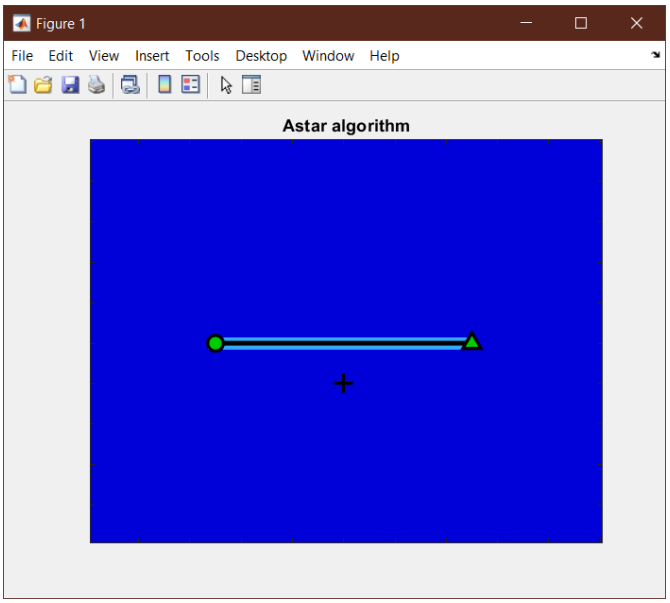
Uživatel pomocí jednoho ze tří hardwarových zařízeních sloužících k ovládání aplikace (viz 2.4) zvolí start, resp. cíl, případně provede dodatečné úpravy simulované plochy (vytvoří nebo odstraní překážky). Jakmile jsou určeny souřadnice startu i cíle, je spuštěn uživatelem zvolený algoritmus pro hledání cesty. Po skončení běhu algoritmu je zvýrazněna prohledaná oblast, a pokud byla nalezena, tak je vyznačena finální trajektorie.



Obr. 14 - Grafická vizualizace prohledané oblasti po skončení A algoritmu, v tomto případě cesta neexistuje*

2.3.4 Finální cesta

Nalezení cílového uzlu v grafu neznamená, že algoritmus zná finální cestu. Jakmile algoritmus nalezne cíl, musí být spuštěn další algoritmus, který najde a vyznačí hledanou cestu. Tento algoritmus funguje na jednoduchém principu. Využije se zde znalosti získané již při hledání cílového bodu, konkrétně ceny gCost (viz 1.2.1) každého prohledaného uzlu. Z cílového uzlu zkontrolujeme tuto cenu všem sousedním uzlům. Ten, který má tuto cenu nejnižší, je nejblíže ke startu. Tento cyklus je prováděn, dokud není dosáhnuto startu. Po jeho dosažení je již známá jedna finální cesta, která je graficky znázorněna černou křivkou, kterou lze vidět na Obr. 15.

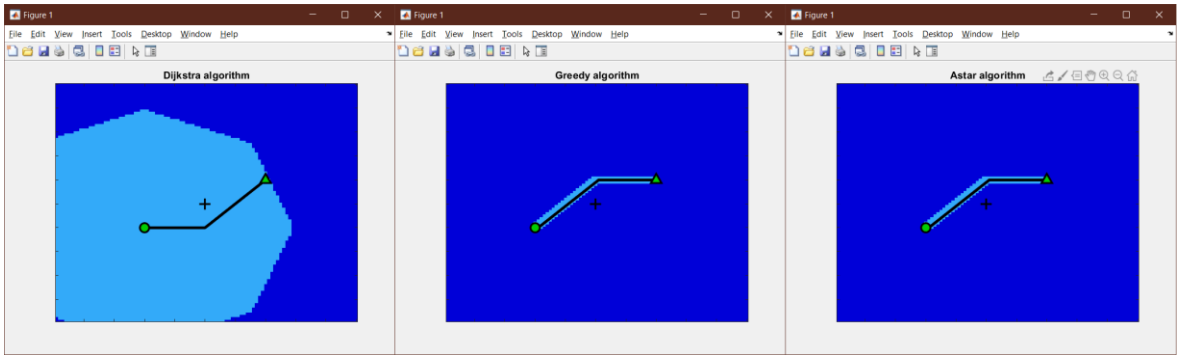


Obr. 15 - Grafická vizualizace výsledné cesty v ploše bez překážek

2.3.5 Porovnání implementovaných algoritmů

Všechny implementované algoritmy byly podrobeny několika testům. Ve všech testech byl změřen průměrný čas trvání chodu algoritmu z 10 měření se stejnými startovními a cílovými souřadnicemi. Do tohoto času nebyl započítán čas, potřebný k vizualizaci výsledné cesty. K měření uplynulých časů byla použita vestavěná funkce v programovacím jazyce MATLAB, která poskytuje přesnost do řádu jednotek mikrosekund [µs]. Další měřenou veličinou byl potřebný počet kroků k vykonání nejkratší nalezené cesty. Poslední pozorovaný aspekt byl počet zkontrolovaných, resp. uzamčených uzlů algoritmem (viz 1.2.1). Výchozí pracovní plocha pro algoritmy je 100x100 uzlů.

Prvním testem pro algoritmy bylo nalezení cíle na volné ploše[2] (bez jakýchkoliv překážek). Výslednou vizualizaci lze vidět na následujícím obrázku. Změřená data jsou v Tab. 3.



Obr. 16 – Výsledná vizualizace prvního testu algoritmů, zleva Dijkstrův algoritmus, Greedy algoritmus, A algoritmus*

Na první pohled je zřejmé, že u Dijkstrovo algoritmu je vyznačená, resp. prohledaná oblast grafu mnohem větší než u zbylých dvou algoritmů. Tato oblast by byla ještě větší, kdyby nebyla omezena velikost pracovní plochy. Celkový počet zkontrolovaných uzlů pro všechny algoritmy je uveden v tabulce. Dále si lze všimnout, že u Dijkstrovo algoritmu je vyznačena rozdílná výsledná cesta než u zbylých dvou. To je způsobeno rozdílným algoritmem vyznačujícím finální cestu (viz 2.3.4) nikoli Dijkstrovým algoritmem.

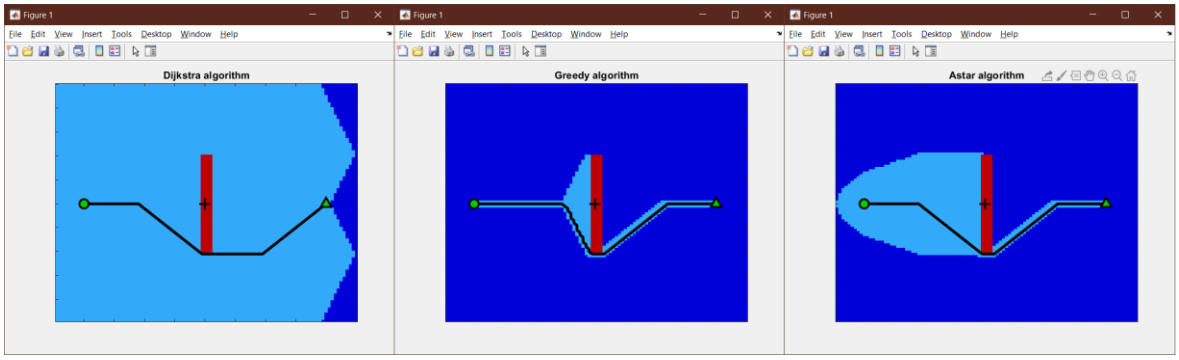
Tab. 3 - Výsledná data prvního testu algoritmů

	Dijkstrův algoritmus	Greedy algoritmus	A* algoritmus
t_{alg} [ms]	232,57	4,5	3,88
Výsledný počet kroků	40	40	40
Počet prohledaných uzlů	5601	40	40

V Tab. 3 lze zjistit, že všechny algoritmy vedly k nalezení cesty o 40 krocích. Markantní rozdíl je však v rychlostech, resp. časech jednotlivých algoritmů. Algoritmy A* i Greedy nalezly cíl více než 50krát rychleji oproti Dijkstrovu algoritmu. To je způsobeno již zmíněným množstvím prohledaných uzlů, které je u Dijkstrova algoritmu 140krát větší.

[2] Souřadnice startu – [30, 40], souřadnice cíle – [70, 60]

Ve druhém testu[3] si algoritmy musely poradit s jednoduchou přímočarou překážkou. Překážka byla umístěna svisle v prostřední části pracovní plochy. Nalezené cesty pomocí algoritmů lze vidět na Obr. 17.



Obr. 17 - Výsledná vizualizace druhého testu implementovaných algoritmů, zleva Dijkstrův algoritmus, Greedy algoritmus, A algoritmus*

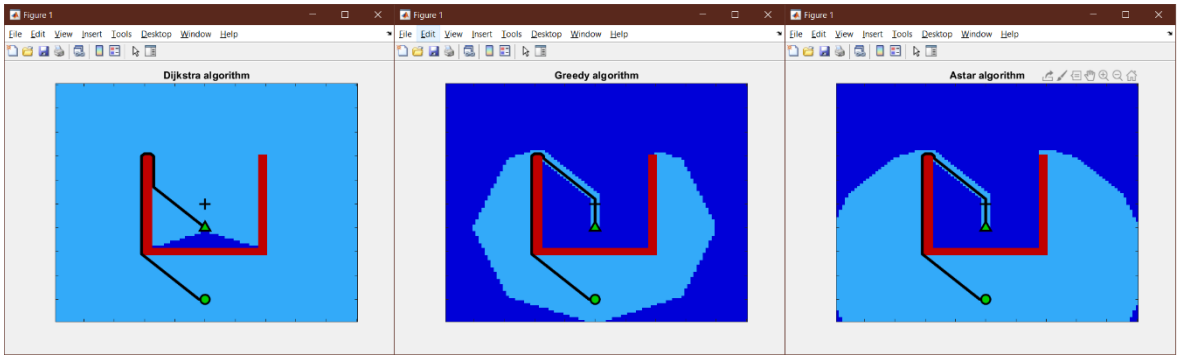
V tomto případě všechny algoritmy vedly k nalezení rozdílné cesty. Je vidět, že u Greedy algoritmu převládá heuristická složka, proto prohledaná oblast směřuje, co nejblíže k cíli. Tato metoda je v tomto případě méně náročná na počet prohledaných uzlů, ale výsledná cesta je delší než nalezené cesty zbylých dvou algoritmů. Greedy algoritmus dosáhl nejkratšího požadovaného času k nalezení cíle, což je opět spojeno s malým množstvím prohledaných uzlů. Na druhou stranu je doba vykonání cesty robotem delší, protože musí urazit více kroků, viz shrnutí závěrem kapitoly.

Tab. 4 - Výsledná data druhého testu algoritmů

	Dijkstrův algoritmus	Greedy algoritmus	A* algoritmus
t_{alg} [ms]	358,44	19,79	67,2
Výsledný počet kroků	80	91	80
Počet prohledaných uzlů	9106	255	1564

[3] Souřadnice startu – [10, 50], souřadnice cíle – [90, 50]

Ve třetím testu[4] byl cílový bod umístěn do prostoru uzavřeného překážkou, která svým tvarem připomíná písmeno ‚U' Tento tvar byl zvolen záměrně, protože je považován za problematický, resp. časově náročný pro všechny tři implementované algoritmy. Na následujícím obrázku jsou zobrazeny výsledky hledání cílového bodu, resp. cesty.



Obr. 18 - Výsledná vizualizace třetího testu implementovaných algoritmů, zleva Dijkstrův algoritmus, Greedy algoritmus, A algoritmus*

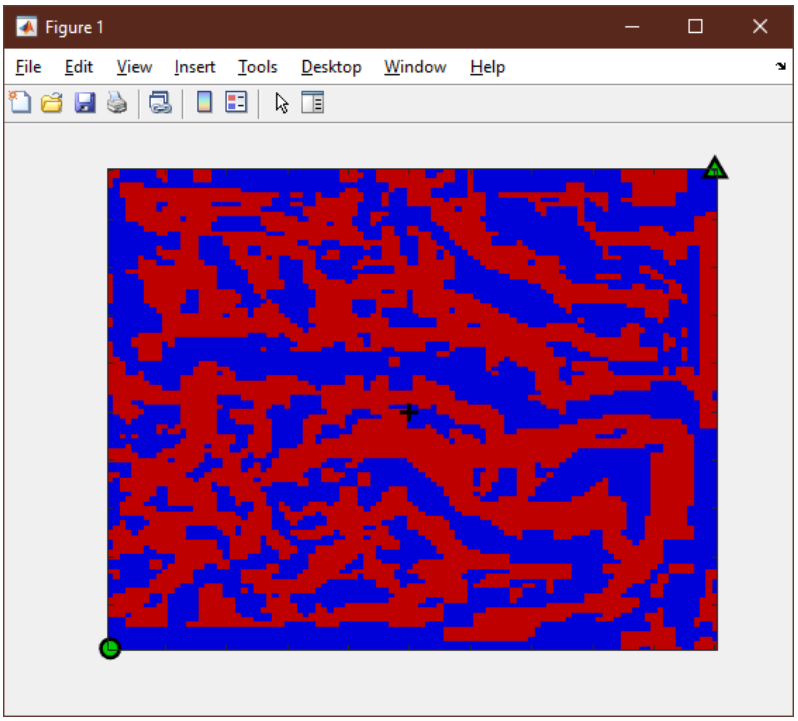
Pro získání bližších informací je třeba nahlédnout do Tab. 5. Nalezení cíle s touto překážkou, trvalo všem algoritmům řádově nižší stovky milisekund. V časově náročnějších aplikacích by bylo vhodné na tento druh překážek použít např. RRT algoritmus (viz 1.2.4). Všechny implementované algoritmy nalezly cestu o stejné délce.

Tab. 5 - Výsledná data třetího testu algoritmů

	Dijkstrův algoritmus	Greedy algoritmus	A* algoritmus
t_{alg} [ms]	386,58	153,92	216,07
Výsledný počet kroků	96	96	96
Počet prohledaných uzlů	9466	2474	4655

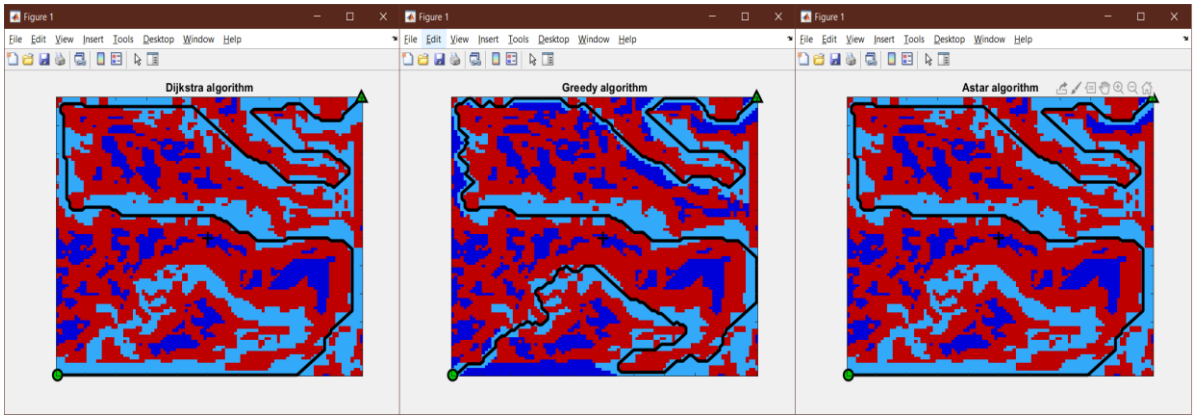
[4] Souřadnice startu – [50, 10], souřadnice cíle – [50, 40]

Posledním, čtvrtým, testem[5] pro vytvořené algoritmy bylo nalezení cíle, resp. cesty v relativně složitém pracovním prostoru. Pro vytvoření této plochy připomínající bludiště byla využita kombinace metod tvorby překážek – automatická společně s manuální. Obě tyto metody jsou vysvětleny v kapitole 2.3.2. Obr. 19 zobrazuje složitost terénu a umístění startovního, resp. cílového bodu.



Obr. 19 - Pracovní prostor testu č. 4

Na Obr. 20 jsou zobrazeny výsledné cesty. Zde si lze všimnout, že Greedy algoritmus zvolil velmi odlišnou cestu, oproti zbylým dvěma algoritmům, která vyšla o 70 kroků delší. To prozrazuje velkou slabinu spoléhání se pouze na heuristickou složku při hledání cílového bodu.



Obr. 20 - Výsledná vizualizace čtvrtého testů algoritmů,
zleva Dijkstrův algoritmus, Greedy algoritmus, A algoritmus*

[5] Souřadnice startu – [1, 1], souřadnice cíle – [100, 100]

Z dat v následující tabulce lze vyčíst, že v tomto testu byl nejrychlejší Dijkstrův algoritmus, a to i přes nejvyšší počet zkontrolovaných uzlů. Z toho vyplývá, že zkontrolování jednoho uzlu Dijkstrovým algoritmem, trvá kratší dobu než A* algoritmem nebo Greedy algoritmem. Tento fakt lze vysvětlit tím, že Dijkstrův algoritmus nepočítá heuristickou složku uzlů. Počítá jen jednu hodnotu každému uzlu.

Tab. 6 - Výsledná data čtvrtého testu algoritmů

	Dijkstrův algoritmus	Greedy algoritmus	A* algoritmus
t_{alg} [ms]	138,31	128,76	173,36
Výsledný počet kroků	416	486	416
Počet prohledaných uzlů	3573	2337	3446

Shrnutí výsledků všech testů lze vidět v Tab. 7. Nejvíce uzlů bylo prohledáno Dijkstrovým algoritmem, což bylo očekávané. Nejméně pak Greedy algoritmem, který má zároveň i nejkratší celkový čas. Tyto dvě výhody Greedy algoritmu jsou kompenzovány vykonaným počtem kroků, kde algoritmus urazil celkově o 81 kroků (cca 13 %) navíc. Dijkstrův algoritmus společně s A* algoritmem vždy našly nejkratší možnou cestu, a proto mají stejný počet vykonaných kroků. Z vypočítaných dat (i z výsledků jednotlivých vykonaných testů) lze konstatovat, že A* algoritmus je dobrým kompromisem mezi dvěma zbylými konkurenty. Nabízí totiž kvalitní řešení jak od velmi jednoduchých map až po ty nejkomplikovanější.

Tab. 7 - Výsledná data všech provedených testů

	Dijkstrův algoritmus	Greedy algoritmus	A* algoritmus
Celkový čas algoritmu [ms]	883	302	456
Celkový počet kroků	632	713	632
Celkový počet uzlů	27746	5106	9705

Následně byl vypočítán čas potřebný k prohledání jednoho uzlu následujícím vzorcem:

$$t_{uzlu} = \frac{Celkový\ čas\ algoritmu}{Celkový\ počet\ uzlů} \qquad (8)$$

Tab. 8 - Vypočítané časy zkontrolování jednoho uzlu

	Dijkstrův algoritmus	Greedy algoritmus	A* algoritmus
t_{uzlu} [µs]	31,8	59,2	47,1

Všechny dosud změřené časy nezahrnují fyzický pohyb robota. V této konkrétní aplikaci je minimální doba jednoho kroku robota, označené jako t_{step}, rovna přibližně dvěma milisekundám. Za předpokladu, že každý krok robota trvá tuto minimální dobu, lze vypočítat celkový čas vykonání cesty následujícím vzorcem.

$$t_{celkový} = t_{alg} + t_{vykonání} \qquad (9)$$

, kde $t_{vykonání} = Výsledný\ počet\ kroků \cdot t_{step}$. Následující tabulka ukazuje skutečné potřebné časy pro vykonání trajektorií robotem. A* algoritmus v tomto testu překonal předešle nejrychlejší Greedy algoritmus. To je způsobené tím, že A* několikrát nalezl kratší cestu, kterou pak robot vykoná rychleji. Tento rozdíl by se zvyšoval s rostoucím t_{step}, např. pro t_{step} = 20 ms, Greedy algoritmus by potřeboval celkově delší čas (12 838 ms) než oba zbylé algoritmy (Dijkstrův algoritmus – 12 027 ms. A* algoritmus – 11 373 ms)

Tab. 9 - Přehled časů potřebných pro vykonání cesty robotem pro čas tstep = 2 ms

		Dijkstrův algoritmus	Greedy algoritmus	A* algoritmus
1. test	t_{alg} [ms]	232,57	4,5	3,88
	$t_{vykonání}$ [ms]	80	80	80
	$t_{celkový}$ [ms]	312,57	84,5	**83,88**
2. test	t_{alg} [ms]	358,44	19,79	67,2
	$t_{vykonání}$ [ms]	160	182	160
	$t_{celkový}$ [ms]	518,44	**201,79**	227,2
3. test	t_{alg} [ms]	386,58	153,92	216,07
	$t_{vykonání}$ [ms]	96	96	96
	$t_{celkový}$ [ms]	482,58	**249,92**	312,07
4. test	t_{alg} [ms]	138,31	128,76	173,36
	$t_{vykonání}$ [ms]	832	972	832
	$t_{celkový}$ [ms]	**970,31**	1100,76	1005,36
Celkem	t_{alg} [ms]	1115	306	460
	$t_{vykonání}$ [ms]	1168	1330	1168
	$t_{celkový}$ [ms]	2283	1636	**1628**

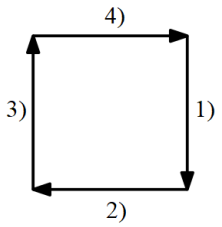
2.4 Komunikace s moduly

Veškerá komunikace s hardwarovými moduly probíhá přes sériový port. Pro navázání tohoto spojení je nutné nejprve detekovat zapojená zařízení. Detekci modulů spustí uživatel stisknutím příslušného tlačítka v aplikaci MagNet. Aplikace se pokusí otevřít sériovou linku mezi počítačem a připojeným zařízením. Po úspěšném otevření této linky je každému modulu zaslán dotaz v podobě definované zprávy. Tato akce může být časově náročná,

protože při každém otevření sériové linky je implementována nucená dvousekundová pauza, která je potřebná k inicializaci spojení. Modul na tuto zprávu odpoví svým „jménem“. Následně jsou modulu přiřazené vlastnosti, které se vypíšou do tabulky v samotné aplikaci. Při ukončení aplikace jsou data z této tabulky uložena, aby při opětovném spuštění, uživatel nemusel znovu detekovat zařízení. Aplikace při spuštění detekuje právě připojené sériové porty a porovná je s porty uloženými v tabulce. Pokud je zjištěno, že port uložený v tabulce není mezi aktuálně dostupnými, tak je zařízení na tomto portu z tabulky odstraněno. Tato funkce částečně odstraňuje riziko chybných dat v tabulce vzniklých po manipulaci s připojenými zařízeními během doby, kdy aplikace není spuštěna.

Po detekci modulů probíhá komunikace s moduly pomocí G-kódu. S tímto kódem se lze nejčastěji potkat u CNC strojů nebo např. 3D tiskáren [14]. Všechny tyto stroje musí pohybovat svými částmi (nástroji) v prostoru. Pro definování tohoto pohybu (sekvence pohybů) se využívá právě G-kód. V této aplikaci se základní příkaz G-kódu skládá ze tří částí. Následující segment kódu nechá miniaturního robota vykonat trasu ve tvaru čtverce.

1) *G0 X0 Y-4*
2) *G0 X-4 Y0*
3) *G0 X0 Y4*
4) *G0 X4 Y0*



, kde G0 značí, že se jedná o funkci pohybu, X0 a Y-4 určuje počet kroků v jednotlivých osách Kartézské soustavy. Novější moduly, které umožňují komplexnější ovládání minirobotů používají G-kód s vícero funkcemi. Tyto další funkce slouží např. na řízení pohybu několika robotů na jedné ploše.

2.5 Možnosti ovládání

Jak již bylo zmíněno v první části práce, pro řízení aplikace (a hledání cesty) byly naprogramovány tři druhy zařízení. V této části je uveden podrobnější popis možností jednotlivých zařízení. Všechna zařízení umožňují zadat startovní, cílový bod, a tím spustit hledání cesty.

2.5.1 Klávesnice

Jako první možnost ovládání aplikace, byla právě klávesnice. Za pomoci tohoto zařízení můžeme ovládat kurzor dvěma předdefinovanými rychlostmi. Pro pohyb kurzoru o jednu

jednotku vzdálenosti jsou použity kurzorové šipky, pro rychlejší pohyb kurzoru pak číslice 8, 4, 5, 6 na numerické klávesnici. Po nastavení kurzoru do požadované pozice a stisku klávesy „S“ je zvolen startovní bod. Klávesami „E“ (end) nebo „T“ (target) zvolíme konečný bod (cíl).

Obr. 21 - Funkce jednotlivých kláves na klávesnici

2.5.2 Gamepad

Veškerý vývoj probíhal na gamepadu s rozložením tlačítek pro herní konzoli Xbox. Požadavky na toto zařízení byly pouze kompatibilita s PC a možnost drátového zapojení, nejlépe přes sběrnici USB pro její dostupnost na všech nynějších počítačích.

Tato periferie má nejvíce funkcí, protože nabízí velmi snadné ovládání. Z tohoto důvodu byl gamepad zvolen jako hlavní možnost ovládání. K naprogramování jednotlivých funkcí byl použit MEX-file JoyMex (volně ke stažení na http://joymex.escabe.org), který poskytuje rozhraní mezi funkcemi napsanými v programovacích jazycích C/C++ a programem MATLAB. Drobnou nevýhodou je, že JoyMex je funkční pouze s operačním systémem Windows. Nepředpokládá se ale spouštění na jiných operačních systémech, a proto je tato nevýhoda zanedbatelná.

Gamepad nabízí čtyři analogové vstupy v podobě dvou joysticků a dvou triggerů. Joysticky slouží jako polohovací zařízení kurzoru, případně k přímému řízení modulu pomocí G-kódu. Triggery („LT/RT“) umožňují nastavit rychlost posuvu kurzoru. Dále zde nalezneme osm naprogramovaných tlačítek.

Tlačítka zde plní různé funkce. Stejně jako u předešlého zařízení i gamepadem můžeme zvolit start („A“) a cíl („B“) pro hledání cesty. Můžeme tvořit („Y“) a mazat („X“) překážky

na pozici kurzoru. Dále je zde možnost vytvoření „náhodné“ sestavy („LB“) překážek (blíže v kapitole 2.3.2). Další tlačítka umožňují smazání všech překážek („RB“) nebo přepnutí používaného algoritmu pro nalezení cesty („START“). Poslední funkcí je zrušení souřadnic posledního zvoleného startovního (cílového) bodu („BACK“). Tato funkce umožňuje uživateli znovu zvolit souřadnice při jejich chybném zadání.

Obr. 22 - Funkce jednotlivých tlačítek gamepadu

2.5.3 Počítačová myš

Aplikace MagNet je okenní, proto pro různé volby nastavení aplikace je potřeba počítačová myš. Tímto zařízením můžeme také ovládat část aplikace pro hledání cesty. Zde je ovládání jednoduché. Kliknutím levého tlačítka myši je zvolen start a následným kliknutím cíl.

2.6 Podpůrné prvky

Aplikace MagNet kromě ovládání jednotlivých modulů také nabízí ovládání dalších zařízení. Mezi tyto zařízení patří webkamera umístěná nad moduly, pro možnost jejich sledování, zdroj elektrického proudu a napětí pro externí napájení modulů a LED lampa.

2.6.1 Webkamera

Prvním podpůrným prvkem je webkamera. Toto zařízení přináší možnost sledování pohybu robotů i ze vzdáleného pracoviště. Pro zakomponování webkamery bylo třeba stáhnout hardwarový balíček do prostředí MATLAB s názvem „Image Image Acquisition

Toolbox™ Support Package for OS Generic Video Interface“. Bez tohoto balíčku nebude webkamera v aplikaci fungovat. Pokud tento balíček aplikace nenajde upozorní uživatele, aby ho stáhnul. Kvůli běhu několika algoritmů paralelně, se pro odlehčení výpočetní zátěže zvolila možnost vykreslení každého 10. snímku kamery. Před samotným spuštěním kamery je uživateli umožněno změnit barevný mód obrazu (černo-bílý nebo barevný). Může také měnit rozlišení podle možností dané webkamery. Dále je možnost při běhu kamery pořídit snímek obrazu, který je následně uložen na disk počítače. Pořízené snímky obsahují ve svém názvu datum a čas vytvoření, čímž je zamezeno vzniku duality jmen souborů. Pokud kamera není zaostřena na požadovanou plochu lze posuvníkem a následně potvrzovacím tlačítkem manuálně měnit její fokus. Tuto vlastnost může uživatel také měnit za běhu kamery.

2.6.2 Zdroj elektrického proudu a napětí

Pro napájení jednotlivých modulů je třeba mít možnost ovládání napájecího zdroje. Použitý zdroj (Keysight technologies – E3648A) má dva výstupní porty. Pro každý lze nastavit rozdílnou hodnotu napětí a maximálního dodávaného proudu. Řízení zdroje je zajištěno sériovou komunikací. Sériovým portem jsou z aplikace posílány definované příkazy do zdroje. Z aplikace je možné ovládat na obou výstupech odděleně napětí, proudový limit a zdali má být výstup zapnut.

2.6.3 Lampa

Posledním podpůrným prvkem je LED osvětlení. Pro toto zařízení je třeba pouze jednoho digitálního pinu z jednodeskového počítače Arduino. Nejdříve pro řízení tohoto prvku byl využit hardwarový balíček „MATLAB support package for Arduino Hardware“. Tento balíček však komplikuje komunikaci s několika Arduiny najednou. V této aplikaci je komunikace s vícero moduly důležitá, proto bylo od tohoto hardwarového balíčku nakonec odstoupeno. Řešením byl stejný typ komunikace (s odlišnými příkazy), jaký je použit pro moduly ovládající miniaturní roboty. Druhým účelem tohoto Arduina je řízení několika relé, které spínají výstupy zdroje elektrického napětí a proudu zapojené do jednotlivých modulů.

2.7 Možnosti dalšího rozšíření

Práci lze dále vyvíjet a rozšiřovat. Problematiku spjatou s plánováním trajektorie robotů v ploše lze rozšířit o další algoritmy. Některé moduly umožňují pohyb robotů vícero než osmi směry, takže v budoucnu lze implementovat algoritmy, které poskytují možnost více směrného pohybu. Dále je možné využit algoritmy na lokalizaci překážek nebo robota. S automatickou lokalizací robota by při hledání cesty bylo nutné zadat pouze cílový bod.

Mezi další možnosti rozšíření mohou patřit jiné způsoby zadávání G-kódů do modulu, např. načtením z textového souboru. Naprogramovat další hardwarová zařízení pro ovládání pohybu robotů. Uzpůsobit pohled kamery na pracovní prostor zvoleného modulu (přesný ořez, rotace obrazu, atp.).

Možností rozšíření je mnoho, a proto se těším na další vývoj této práce.

Závěr

Tato bakalářská práce se věnuje problematice plánování trajektorie objektů v ploše, implementaci algoritmů pro hledání cesty mezi dvěma body a tvorbě okenní aplikace *MagNet*. Hlavní motivací pro vznik této práce bylo chybějící grafické uživatelské rozhraní pro komunikaci s moduly, které umožňují řízení miniaturních robotů. Další motivací bylo rozšíření možností ovládání těchto modulů.

Na základě těchto požadavků byla vytvořena okenní aplikace *MagNet*. Tato aplikace poskytuje možnost řízení modulů, a to hned několika způsoby (přímé řízení, předdefinované cesty, konverze nalezené cesty na G-kód, ruční zadávání G-kódu). Část aplikace je věnována problematice hledání cesty v ploše. Tuto část lze ovládat třemi různými hardwarovými zařízeními (počítačová myš, klávesnice a gamepad). Zároveň uživatel může použít tři různé implementované algoritmy pro hledání cesty (Dijkstrův algoritmus, Greedy algoritmus a A* algoritmus). Aplikace také umožňuje ovládání několika dalších hardwarových podpůrných prvků (zdroj elektrického napětí a proudu, webkamera a osvětlení modulů).

Naprogramované algoritmy byly podrobeny několika testům. Výsledky testů ukázaly, že každý algoritmus je vhodný v jiné situaci. Pro jednoduché pracovní prostory se jeví Greedy algoritmus jako dobré řešení. Naopak pro velmi složitá bludiště s velkým množstvím překážek dosáhl nejlepšího výsledku Dijkstrův algoritmus. Jako skvělý kompromis mezi těmito dvěma extrémy lze považovat A* algoritmus.

Seznam literatury a informačních zdrojů

[1] *MATLAB - tvorba uživatelských aplikací*. Praha: BEN - technická literatura, 2004. ISBN 80-7300-133-0.

[2] Problém nejkratší cesty. *Algoritmus* [online]. Copyright © 2015 [cit. 04.05.2020]. Dostupné z: https://www.algoritmy.net/article/36597/Nejkratsi-cesta

[3] Fitzgerald, A. E., Kingsley, Charles a Umans, Stephen D. Electric *machinery*. 6th ed. Boston: McGraw-Hill, c2003. xv, 688 s. McGraw-Hill series in electrical engineering. Power and energy. ISBN 0-07-366009-4.

[4] Pathfinding Algorithms - The Startup - Medium. *Medium – Get smarter about what matters to you.* [online]. Dostupné z: https://medium.com/swlh/pathfinding-algorithms-6c0d4febe8fd

[5] Heuristics. *A** [online]. Copyright © 2020 [cit. 04.05.2020]. Dostupné z: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

[6] A*. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 22. 12. 2019 [cit. 2020-05-04]. Dostupné z: https://cs.wikipedia.org/wiki/A*

[7] *Carnegie Mellon School of Computer Science* | [online]. Copyright ©3 [cit. 04.05.2020]. Dostupné z: https://www.cs.cmu.edu/afs/cs/academic/class/15494-s18/lectures/pathplan/path-planning.pdf

[8] *MathWorks - Makers of MATLAB and Simulink - MATLAB & Simulink* [online]. Copyright © 1994 [cit. 04.05.2020]. Dostupné z: https://www.mathworks.com/help/matlab/

[9] (PDF) Solving the Path Planning Problem in Mobile Robotics with the Multi-Objective Evolutionary Algorithm. *ResearchGate | Find and share research* [online]. Copyright © 2008 [cit. 04.05.2020]. Dostupné z: https://www.researchgate.net/publication/327150237_Solving_the_Path_Planning_Problem_in_Mobile_Robotics_with_the_Multi-Objective_Evolutionary_Algorithm

[10] GAYLE, Russell, Avneesh SUD, Ming C. LIN a Dinesh MANOCHA. *Reactive Deformation Roadmaps: Motion Planning of Multiple Robots in Dynamic Environments* [online]. Severní Karolína, 2007 [cit. 2020-05-17]. Dostupné z: http://gamma.cs.unc.edu/RDR/media/iros07-final.pdf. Vědecký článek. University of North Carolina at Chapel Hill.

[11] SUN, Chaoyi, Qing LI a Li LI. A Roadmap-Path Reshaping Algorithm for Real-Time Motion Planning. *Fellow* [online]. 2019, 2019(1), 1 - 4 [cit. 2020-05-17]. Dostupné z: https://arxiv.org/pdf/1902.11056.pdf

[12] STENTZ, Anthony. *The Focussed D* Algorithm for Real-Time Replanning* [online]. 1995, 1 - 8 [cit. 2020-05-17]. Dostupné z: http://robotics.caltech.edu/~jwb/courses/ME132/handouts/Dstar_ijcai95.pdf

[13] DIJKSTRA, Edsger Wybe. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*. 1959, (1), 269 – 271.

[14] G-code. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2020-05-19]. Dostupné z: https://en.wikipedia.org/wiki/G-code

[15] *Plánování pohybu technikami RRT*. Praha, 2010. Bakalářská práce. České vysoké učení technické, Fakulta elektrotechnická, Katedra kybernetiky. Vedoucí práce Ing. Jan Faigl.

[16] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, July 1968, doi: 10.1109/TSSC.1968.300136.

[17] Sun L., Liu X., Leng M. (2006) An Effective Algorithm of Shortest Path Planning in a Static Environment. In: Wang K., Kovacs G.L., Wozny M., Fang M. (eds) Knowledge Enterprise: Intelligent Strategies in Product Design, Manufacturing, and Management. PROLAMAT 2006. IFIP International Federation for Information Processing, vol 207. Springer, Boston, MA

[18] KUTHAN, J., AND JUŘÍK, M. Magneticky aktuovaný robotický systém pracující v jedné rovině. In 2017 Elektrotechnika a Informatika (2017), Západočeská univerzita v Plzni.

[19] KUTHAN, J., AND JUŘÍK, M. Klíčové parametry pro polohování magnetických těles v planární rovině. In 2018 Elektrotechnika a Informatika (2018), Západočeská univerzita v Plzni.

Přílohy

Veškerá měření probíhala na PC s následujícími specifikacemi:

Procesor	IntelCore i7-4770k (3,5 Ghz)
Grafická karta	Asus Nvidia gtx770
Operační paměť	Kingston 16GB KIT DDR3 1600MHz
Základní deska	MSI Z87 MPOWER

Použitý software:

Tvorba aplikace, algoritmů: MATLAB 2019a
Grafický editor: IPE
Programování modulů: Arduino IDE

Další použitý hardware:

Webkamera: Logitech C920 a Logitech Brio
Zdroj elektrického napájení: Keysight technologies – E3648A