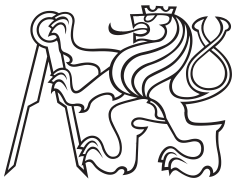
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Následování člověka mobilním robotem

Mykhaylo Zelenskyy

Školitel: Ing. Jan Chudoba
Květen 2017

Poděkování

Prohlášení

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 10, 2017

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 10. května 2017

Abstrakt

Klíčová slova: robotika, počítačové vidění, řízení robotu

Školitel: Ing. Jan Chudoba

Abstract

Keywords: robotics, computer vision, robot control

Title translation: Following of Human by Mobile Robot

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Pomocný robot se může zdát mnhým z nás jako futuristický sen. Takový robot by místo nás nosil věci, pomáhal při nákupech, asistoval v nemocnicích či ošetřoval raněné ve válkách. Měl by tolik výhod, že by v budoucnu bylo trendem tohoto pomocníka vlastnit.

Bylo provedeno mnoho různých výzkumů ohledně návrhu robotů, které by dokázaly sledovat člověka. S tímto problémem jsou spojené dvě důležité otázky. Jaké senzory použit pro lokalizaci člověka? Jak řešit řízení a navigaci robotu tak, aby udržoval určitou vzdálenost a vyhýbal se případným překážkám?

Nejdříve je třeba definovat, co vlastně je robotem následující člověka. Jedná se o mobilního robota, který sleduje určitou osobu a zároveň objíždí překážky a chová se k ostatním lidem jako k překážkám, tj. nezmění cíl sledování během jízdy.

Takový robot typicky může být vybaven mnoha různými senzory, např. laserovým, zvukovým nebo IR dálkoměrem, aby mohl měřit vzdálenost od překážek, kamerou pro detekci sledovaného objektu, bezdrátovým přenášečem signálu, GPS apod. Tyto senzory by měly fungovat současně, aby robot dokázal všechno, co se od něj očekává.

1.1 Cíl práce

Cílem této práce je navrhnout systém pro řízení robotu, tak aby dokázal splnit úkoly definované v úvodu.

Při návrhu tohoto systému je třeba uvědomit si několik věcí:

1. Cíl se nesmí pohybovat větší rychlostí, než je maximální rychlost robotu. Může se stát, že robot už nikdy nedokáže najít sledovaného člověka, dokud se dotyčný nevrátí na dostatečně blízko k robotu.

2. Aby značka byla dobře detekovatelná, musí být umístěna v ochranném prostředí, tj. být kontrastní v porovnání s pozadím.

3. V případě použití dálkoměru pro měření vzdálenosti mezi objekty je důležité, aby systém nedetekoval sledovaný cíl jako překážku.

Obrázek 1.1: Robot následující člověka

1.2 Motivace

Jak již bylo řečeno, robot s podobnou funkcionalitou by byl velice užitečný jako pomocník v různých oborech lidské činnosti. V medicínských zařízeních a domovech pro seniory by se takový robot mohl starat o pacienty (viz obr. 1.1), také by našel své uplatnění v armádě, kde by pomáhal vojákům s převozem nákladů. Vylepšená varianta robota by mohla posloužit jako tělesná stráž.

Kapitola 2
Návrh systému

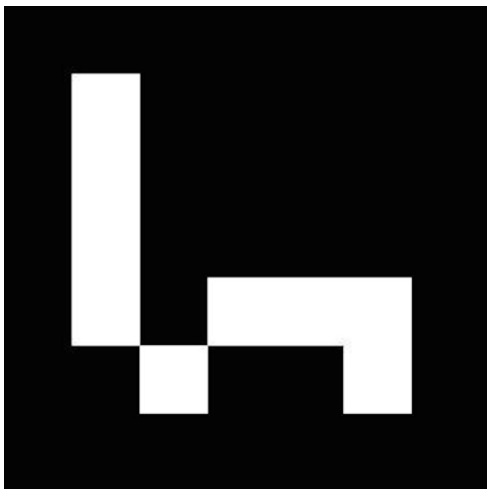
Existuje několik výzkumných projektů, které se zabývají problematikou pomocných robotů, které by následovaly člověka. Většina těchto projektů zakládá na detekci pohybujícího se objektu, které je následně sledován. V některých pracích jsou použity neuronové sítě pro detekci obličeje a background subtraction pro nalezení pohybu v obraze [4], např. pohybujících se nohou člověka. Avšak použití tohoto algoritmu vede k tomu, že sledovaný člověk musí být otočen k robotu obličejem, a prezence několika lidí v místnosti přivede k tomu, že robot nedokáže určit, koho musí následovat. V případě, že neuronová síť bude naučena pouze na konkrétního člověka, což by tento problém mohlo řešit, znovupoužitelnost algoritmu klesá, protože by bylo třeba síť přeučovat pro každý konkrétní případ.

Dalším možným řešením této problematiky je použití barevného markeru [5], což znamená, že robot hledá určitou shodu barev, kterou pak sleduje. V tomto případě člověk může být otočen k robotu zády, a algoritmus může být použit i v plných lidí prostorech. Avšak detekce barevného vzoru nemusí být robustní kvůli změně osvětlení nebo špatné volbě barev, které nebudou dostatečně kontrastní v porovnání s oblečením sledovaného člověka.

Proto pro návrh této práce bylo rozhodnuto použít specifický druh vizuálních značek, což bude popsáno v následujícím textu. Cíl pro sledování musí být unikátní, a algoritmus detekce musí být robustní, aby nedocházelo k tomu, že robot nebude vědět, koho má sledovat. Robot dopředu bude vědět, jaký typ značky musí hledat, proto se snižuje šance, že si splete cíl s jiným člověkem nebo překážkou.

2.1 Volba vizuální značky

Mobilní robot bude následovat člověka, který má na sobě umístěnou předem známou vizuální značku. Pro navigaci robotu se v praxi běžně používají markery pro rozšířenou realitu [6][7][8]. Jejich detekce je většinou jednoduchá a můžou v sobě uchovávat užitečnou informaci, např. identifikační číslo, podle kterého robot pozná, koho sleduje. Jednu z možných vizuálních značek, které jsou v této práci použity, lze vidět na obrázku 2.1.

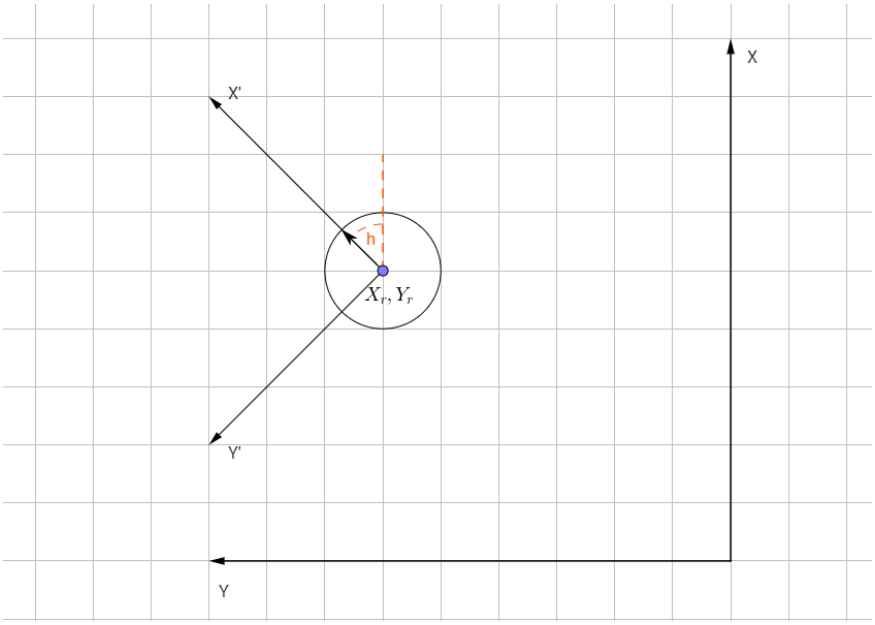


Obrázek 2.1: Příklad vizuální značky

2.2 Volba souřadnicového systému

Pro řízení robotu je třeba zvolit souřadnicový systém, ve kterém se bude pohybovat.

Počátek zvolené souřadnicové soustavy umístěn do počáteční pozice robotu a osa X je ve stejném směru, jako počáteční směr jízdy robotu. Kladný směr jízdy robotu je v kladném směru osy Y, jak je vyobrazeno na 2.2, a je v rozmezí $(-\pi; \pi)$. Poloha cíle (viz sekci 3.3) je poté vztažená k poloze robotu a je počítána v čárkované soustavě, jejíž počátek je umístěn do aktuální pozice robotu, a je vůči původní otočená o h, kde h je směr jízdy robotu.



Obrázek 2.2: Použitý souřadnicový systém

2.3 Implementační záležitosti

Protože se očekává, že běh software bude v reálném čase, musí být rychlý a stabilní. Z těchto důvodů bylo rozhodnuto, že bude použit jazyk C++.

Také v práci je použita knihovna OpenCV[9], která nabízí obrovské možnosti pro zpracování obrazu. Tato knihovna obsahuje přes 2500 optimalizovaných algoritmů pro počítačové vidění a strojové učení. Některé z nich budou použity pro detekci vizuální značky ve výstupu z kamery. Navíc v této knihovně je implementován algoritmus pro detekci ArUco markeru [6], který je podrobněji popsán v sekci 3.2.2.

Kapitola 3

Detekce

3.1 Zpracování obrazu z kamery

Než bude možné použít rozpoznávací algoritmy, výstup z kamery musí být náležitě zpracován, aby odpovídal formátu, se kterým algoritmy pracují.

Načtený snímek je převeden do černobílé podoby, je tedy použito prahování. Jelikož se předpokládá, že se robot může pohybovat v prostředí s nerovnoměrným osvětlením, pro binarizaci obrazu je vhodné použit adaptivní prahování [1] (viz obr. 3.1). Tato metoda počítá práh pro malé části obrazu místo globálního nastavení prahu pro celý snímek.

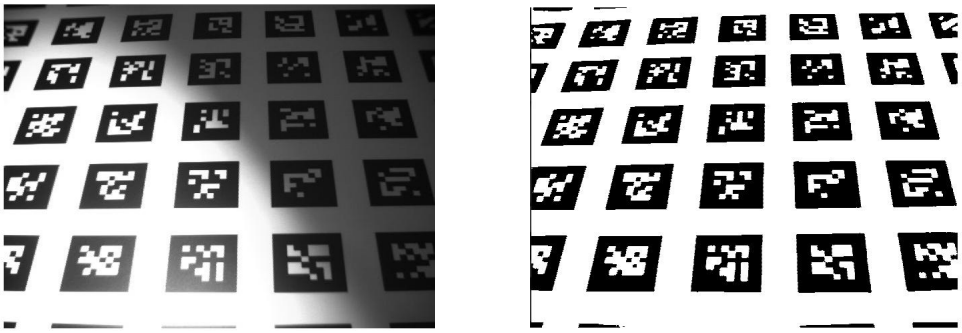
Dále je třeba najít kontury. Cannyho algoritmus pro nalezení kontur je implementován v knihovně OpenCV. Tato funkce navíc zajišťuje zachování hierarchie kontur[2], tedy topologii obrázku, jak je vidět na 3.2. Tato informace je užitečné pro následné rozpoznávání, protože pak kontury na nejnižší nebo nejvyšší úrovni hierarchie můžou být ignorovány, pokud se předpokládá, že vizuální značka bude umístěna v ochranné zóně a bude obsahovat další pod-vzory.

3.2 Rozpoznávání vzoru

Pro rozpoznávání vizuální značky je vhodné použít algoritmus, který zakládá na hledání známého vzoru v obraze. Nejběžnějším algoritmem, který se pro této účely používá, je Template Matching [10]. Podstatou tohoto algoritmu je to, že se pro různé velikosti vstupného obrazu počítá jeho korelace se známým vzorem. Vytváří se tzv. pyramida obrazů [11], která se skládá ze vstupních obrázků různé velikosti a orientace. Každý obraz v této pyramidě je rozdělen na oblasti o stejné velikosti, jako je velikost vzoru. Pro každou z těchto oblasti se výpočte korelace se vzorem. Následně je zvolena oblast, kde je korelační koeficient největší.

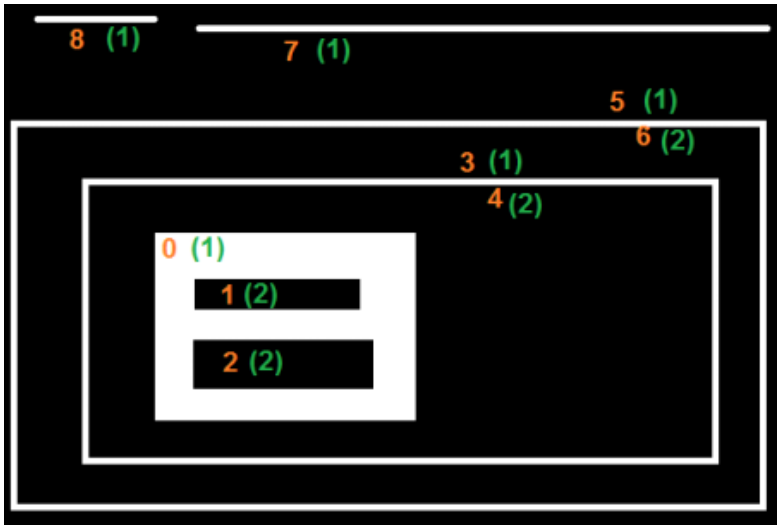
Kvůli tomu, že korelace musí být spočtena pro velký počet vstupních obrazu o různé velikosti, Template Matching může být pomalý. Jelikož se předpokládá použití čtvercové vizuální značky, je praktičtější nejdříve najít oblast zájmu (Region of Interest, ROI), pro níž se spočte korelace se známým vzorem.

Proto pro detekci vizuální značky byl navržen korelační algoritmus, který



(a) : Před prahováním **(b) :** Po prahování

Obrázek 3.1: Příklad použití adaptivního prahování [1]



Obrázek 3.2: Hierarchie (topologie) obrazu [2]

je popsán v následujícím textu. Pro porovnání funkčnosti tohoto algoritmu byl použit ArUco marker detektor [12], který je k dispozici přímo v knihovně OpenCV.

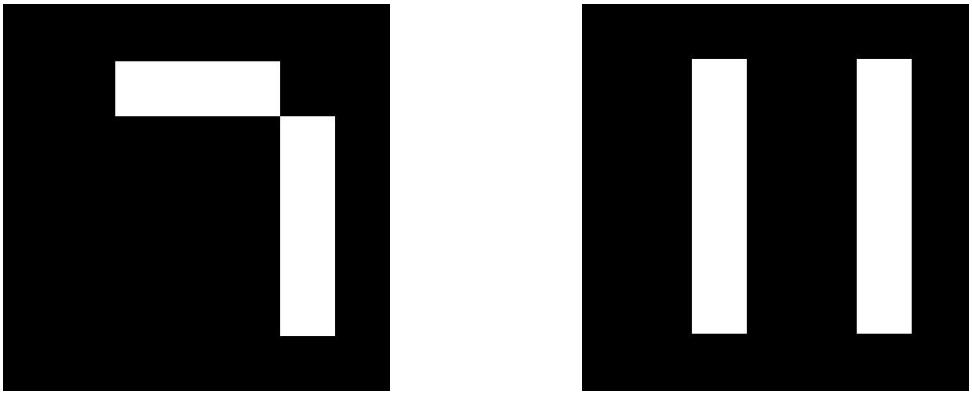
3.2.1 Korelační algoritmus

Tento algoritmus využívá principu korelaci mezi dvěma veličinami, tedy mezi známým vzorem a nalezenou ROI. Korelace uvádí, jak jsou na sobě tyto veličiny závislé, a může nabývat hodnot od -1 do +1. Hodnota korelačního koeficientu blížící se -1 značí nepřímou závislost veličin, koeficient blízký +1 naopak značí přímou závislost, což je zobrazeno na A.1.

Hledaná ROI je v zjednodušeném případě konvexní čtyřúhelník, proto stačí s použitím informace o topologii obrazu spočítat polygony nalezených kontur a vybrat pouze ty, co obsahují čtyři strany a nejsou konkávní. Pro aproximaci křivky se používá Ramerův Douglasův Peuckerův algoritmus [13] (viz obr. 3.4), který je již implementován v OpenCV.

(a) : Referenční marker

(b) : $\mathrm{cor}(I_1, I_2) = 0.47$



(c) : $\mathrm{cor}(I_1, I_2) = \text{-}0.55$

(d) : $\mathrm{cor}(I_1, I_2) = \text{-}0.04$

Obrázek 3.3: Příklad korelace mezi markery vůči referenčnímu

Následně je spočítána transformační matice mezi nalezeným polygonem a známým vzorem. Pomocí této matice se polygon převede do takové podoby, aby byl porovnatelný se vzorem.

Dále dojde k vyhodnocení korelace nalezené ROI a vzoru. Pokud je vypočtený korelační koeficient větší, než zadaný práh, ROI se vyhodnotí jako správně označena a algoritmus vrátí souřadnice jejích rohů pro následné zpracování.

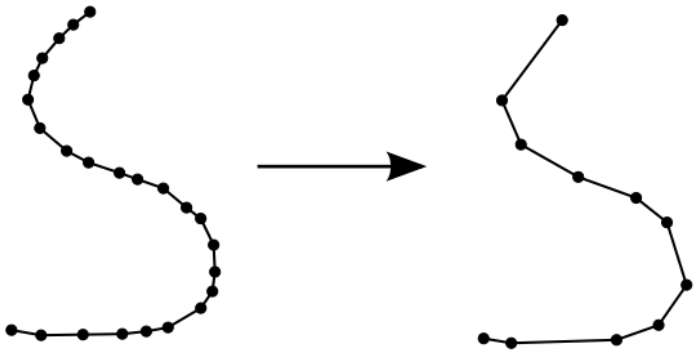
Korelaci mezi dvěma obrázky I_1 a I_2 s N pixely je vypočtena následujícím způsobem:

$$\mu_{1,2} = \frac{\sum_{i,j} I_{i,j}}{N}, \tag{3.1}$$

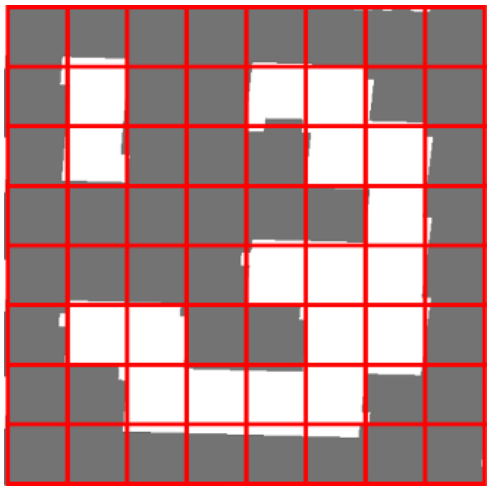
$$\sigma_{1,2} = \sqrt{\left(\frac{\sum_{i,j}(I_{i,j} - \mu_{1,2})^2}{N}\right)^2}, \tag{3.2}$$

$$covar(I_1, I_2) = \frac{(I_1 - \mu_1) \cdot (I_2 - \mu_2)}{N}, \tag{3.3}$$

$$cor(I_1, I_2) = \frac{covar(I_1, I_2)}{\sigma_1 \sigma_2}. \tag{3.4}$$



Obrázek 3.4: Aproximace křivky pomocí Ramerova Douglasova Peuckerova algoritmu

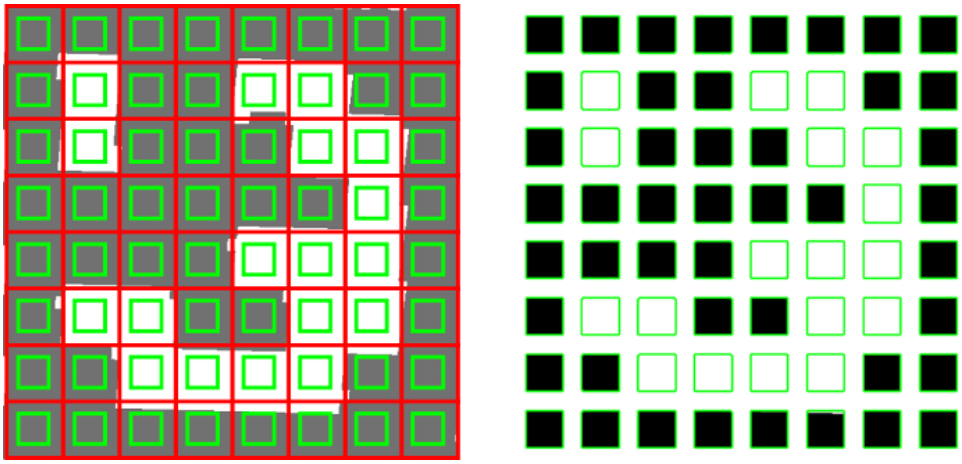


Obrázek 3.5: Buňky ArUco markeru

3.2.2 ArUco marker detektor

Tento algoritmus je implementován v OpenCV knihovně a umožňuje rozpoznávání ArUco markerů nebo podobných vizuálních značek. Podobně jako výše uvedený korelační algoritmus, hledá konvexní čtyřúhelník. Rozdíl pak spočívá v odlišném zpracování nalezené ROI, kde se místo korelace využívá binární mapa kandidáta.

ROI je rozdělena na mřížku s počtem buněk rovným počtu bitů hledaného vzoru (viz obr. 3.5). Následně se spočítá, kolik bílých a černých pixelů obsahuje každá buňka, na základě čehož se vyhodnotí, jestli buňka je černá nebo bílá (viz obr. 3.6). Pokud detektor ve svém slovníku obsahuje vzor se stejnou binární maskou, vrátí souřadnice rohů nalezené ROI.



Obrázek 3.6: Vyhodnocení obrázku pomocí detektoru ArUco

3.3 Měření polohy cíle

Algoritmy popsané výše naleznou polohu vizuální značky v obraze, z čehož lze spočítat její umístění vůči kameře, je-li známa velikost této značky. Pro výpočty lze použít model ideální (dírkové) kamery [14], kde vzdálenost mezi kamerou a značkou je vyjádřena jako

$$d = \frac{fX}{x}, \tag{3.5}$$

kde x značí nejkratší vzdálenost mezi dvěma rohy nalezené značky, f je ohnisková vzdálenost a X je známá délka hrany značky .

Pro nalezení posunutí značky vůči kameře při odklonění od její osy musí být znám zorný úhel kamery. Ten lze spočítat dle vztahu

$$\alpha = 2 \arctan\left(\frac{w}{2f}\right), \tag{3.6}$$

kde w je šířka výstupního obrazu z kamery.

Potom úhel, který pokrývá jeden pixel snímku, je vyjádřen vztahem

$$APP = \frac{\alpha}{w}. \tag{3.7}$$

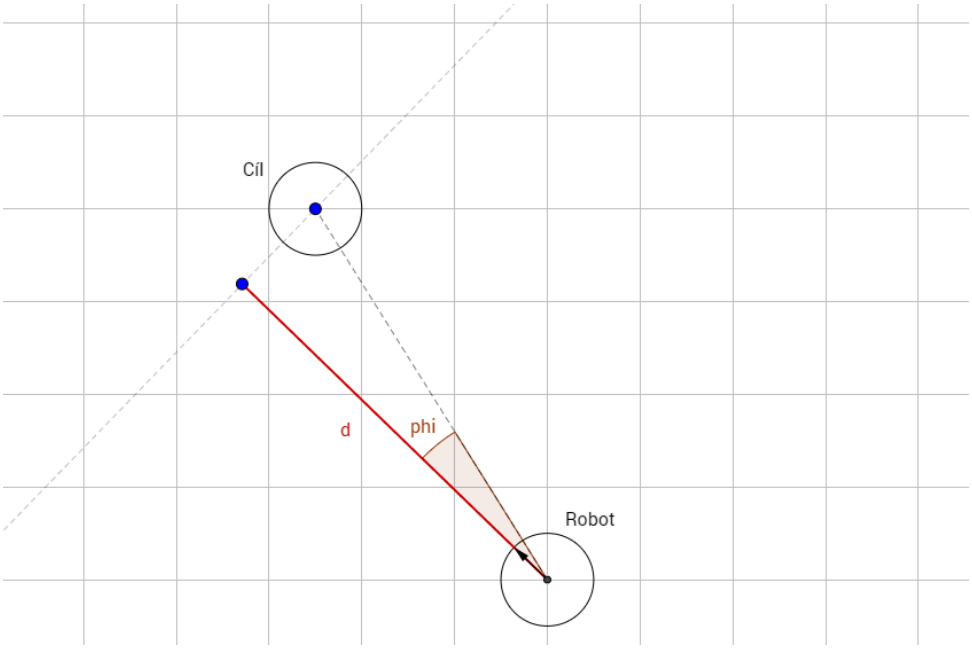
Následně posunutí objektu vůči středu kamery je

$$\phi = (x_c - x_t)APP, \tag{3.8}$$

kde x_c je pixel vyjadřující střed kamery v horizontálním směru, x_t je pixel vyjadřující střed nalezené značky v horizontálním směru.

3.4 Kalmanův filtr

Při sledování člověka může nastat situace, že vizuální značka nebude detekovatelná. Potom není možnost spočítat polohu značky vůči robotu, a tak robot



Obrázek 3.7: Měřené veličiny d a ϕ

buď zastaví, nebo bude pokračovat pohyb dle posledního měření. Toto může přivést k tomu, že se člověk bude muset vrátit k robotu, aby ten ho znovu začal sledovat. Aby tato situace nenastala, je je třeba predikovat polohu cíle z předchozích měření. Pro tyto účely je v práci použit Kalmanův filtr [15].

Model lze popsat pomocí následujícího stavového vektoru:

$$\boldsymbol{x} = \begin{bmatrix} x & y & v_x & v_y \end{bmatrix}^T, \tag{3.9}$$

kde v_x, resp. v_y, značí rychlost ve směru osy X, resp. Y.

Stavový model je pak vyjádřen jako

$$\boldsymbol{x}_{t+1} = \boldsymbol{F}\boldsymbol{x}_t, \tag{3.10}$$

kde $\boldsymbol{F}$ je matice přechodu stavů $\boldsymbol{x}$ z času t do času $t+1$ a platí, že

$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & dt & 0 \\ 0 & 0 & 0 & dt \end{bmatrix}. \tag{3.11}$$

Pro sledování cíle lze Kalmanův filtr rozdělit do tří kroků:

1. Inicializace ($t = 0$). Během tohoto kroku je nastavena počáteční pozice cíle $\boldsymbol{x}_0$ a výchozí hodnota pro kovarianční matici $\boldsymbol{P}_0$. Jelikož počáteční pozice cíle nemusí být známa, je vhodné zvolit $\boldsymbol{x}_0$ tak, aby v případě, že kamera nedetekuje značku po první iteraci, zůstával robot na místě.

2. Predikce ($t > 0$). V tomto kroku se provádí predikce polohy cíle v čase $t+1$, tj. $\boldsymbol{x}_{t+1}$, dle 3.10. Také se počítá nová kovarianční matice dle následujícího vztahu

$$\boldsymbol{P}_{t+1} = \boldsymbol{F}\boldsymbol{P}_t\boldsymbol{F}^T + \boldsymbol{Q}_{t+1}, \tag{3.12}$$

kde $\boldsymbol{Q}$ značí matici kovariancí šumů (výpočet viz [16]).

3. Filtrace ($t > 0$). Během tohoto kroku je poloha cíle upřesněna na základě provedeného měření. Nejdříve se spočte rozdíl mezi reálnou polohou a predikovanou v kroku 2.:

$$\boldsymbol{y}_t = \begin{bmatrix} x_{m_t} \\ y_{m_t} \end{bmatrix} - \boldsymbol{H}\boldsymbol{x}_t, \tag{3.13}$$

kde

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{3.14}$$

Dále se vypočítá Kalmanovo zesílení, pro nějž platí:

$$\boldsymbol{K}_t = \boldsymbol{P}_t\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}_t\boldsymbol{H}^T + \boldsymbol{R})^{-1}, \tag{3.15}$$

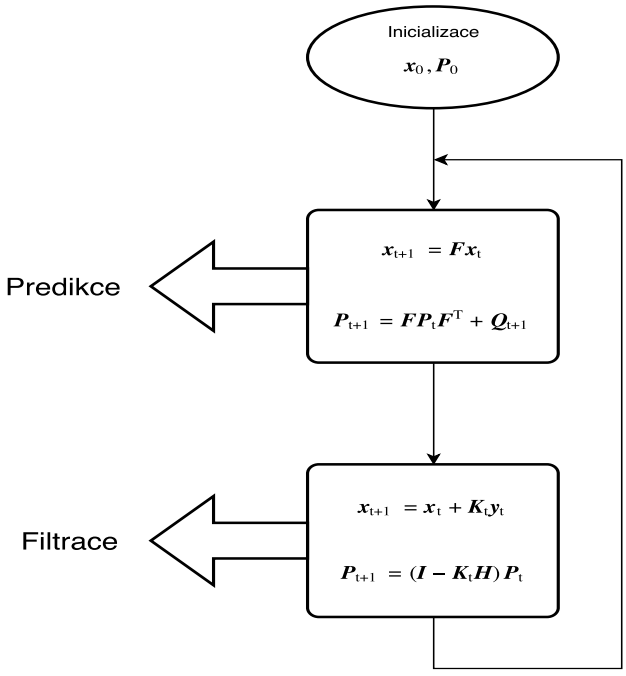
kde $\boldsymbol{R}$ ja matice kovariancí šumů (výpočet viz [16]).

Následně se provede zpřesnění polohy cíle a aktualizuje se kovarianční matice:

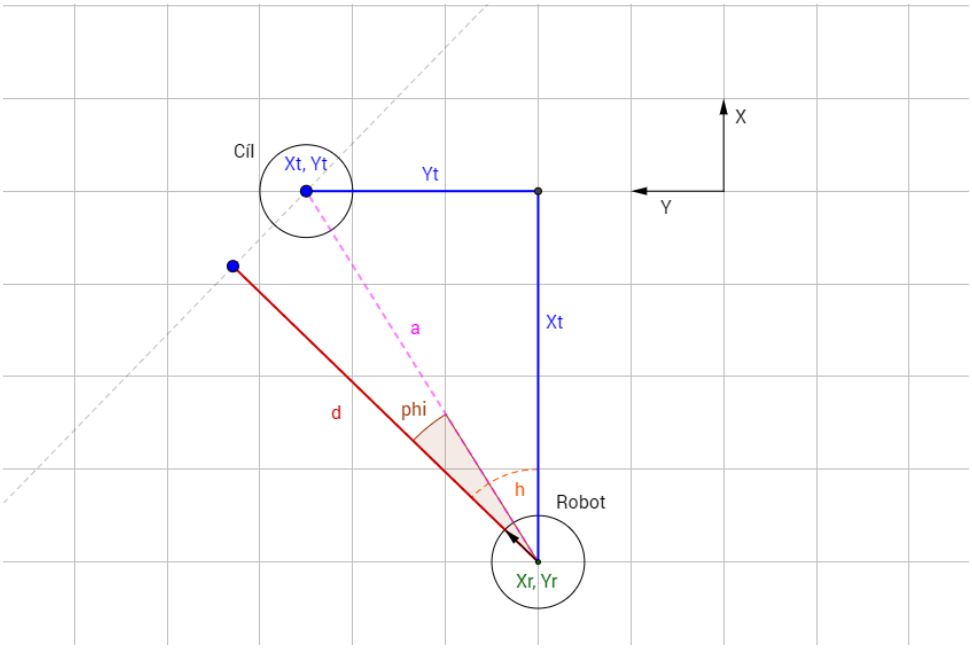
$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{K}_t\boldsymbol{y}_t, \tag{3.16}$$

$$\boldsymbol{P}_{t+1} = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})\boldsymbol{P}_t. \tag{3.17}$$

Celý proces lze shrnout pomocí diagrame 3.8.



Obrázek 3.8: Shrnutí algoritmu Kalmanova filtru



Obrázek 3.9: Odvození hodnot d a ϕ ze známé polohy cíle

Z nalezených hodnot x a y lze jednoduše odvodit hodnoty d a ϕ, což je také vidět na 3.9. Jelikož se předpokládá, že poloha robotu je známa, lze spočítat x_t a y_t, což je poloha cíle vůči robotu, jako:

$$x_t = x - x_r, \tag{3.18}$$

$$y_t = y - y_r. \tag{3.19}$$

Dále jsou pak ϕ a d nalezeny dle vztahů:

$$\phi = h - \text{atan2}(y_t, x_t), \tag{3.20}$$

$$d = \sqrt{x_t + y_t}\cos(\phi). \tag{3.21}$$

Kapitola 4
Řízení robotu

4.1 Vyhýbání se překážkám

Pro řízení autonomního robotu je důležité, aby se během sledování cíle dokázal vyhýbat překážkám. Algoritmů pro řízení robotu v prostředí s překážkami je několik [3], jejichž přehled je k nalezení v příloze A.

Omezením na výběr algoritmu pro vyhýbání se překážkám je to, že se robot bude pohybovat v prostředí, které není dopředu známo. Proto je výhodnější použít algoritmus, který pracuje s lokální mapou prostředí vytvořenou kolem robotu místo globální. Navíc při použitý vztažné soustavy a diferenciálního výpočtu polohy a rychlosti robotu nelze přesně stanovit, kde se nachází on nebo překážky kolem něj. Proto je třeba použít algoritmus, který by pracoval pouze se zakládní kinematikou robotu a prostředí.

S uvážením těchto omezení, nejvhodnějšími algoritmy pro řešení zádaného problému jsou algoritmus Bug [3, s. 272–276] nebo Vector Field Histogram [3, s. 276–278][17].

4.1.1 Algoritmus Bug

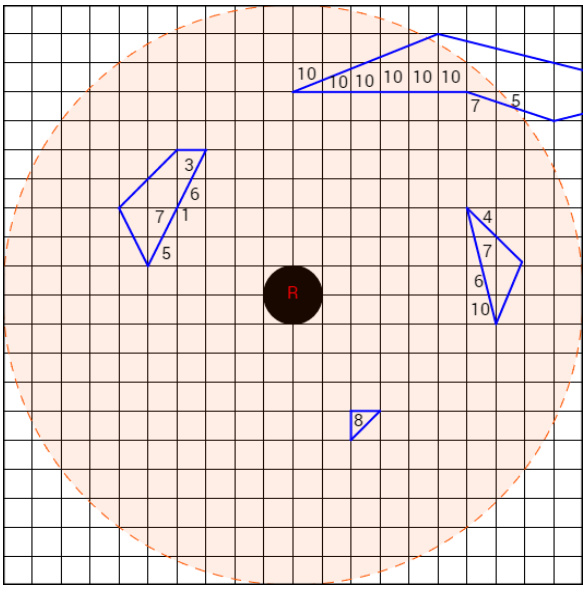
Tento algoritmus je nejjednošším algoritmem pro vyhýbání se překážkám. Podstatou algoritmu je to, že robot narazí na překážku a následuje její konturu. Existuje několik obdob toho algortimu, které se liší v tom, jak je obvod překážky sledován robotem. Například algoritmus Bug1 prvně s použitím dotykového senzoru objede celou překážku, následně zvolí bod, který je nejblíže cíli, vrátí se tam a pokračuje směrem k cíli. Tento postup je dost neefiktivní, ale zajistí, že robot vždy dosáhne cíle.

Možným vylepšením tohoto algoritmu je Tangent Bug, který používá dálkoměr pro měření vzdálenosti k překážkám a vytváří model lokálního prostředí, který je reprezentován jako graf.

Nevýhodou algoritmu Bug je to, že když nalezne jakoukoliv překážku, snaží se sledovat její konturu, nikoliv se jí přímo výhnout. Proto při následování cíle se může stát, že kvůli sledování obrysu překážky dojde ke ztatě cíle ze zorného pole a již nebude možnost odhádnout jeho pozici. Proto v práci je použit Vector Field Histogram, který je navržen tak, aby se robot snažil překážky objíždět dříve, než bude v nebezpečné zóně blízko k překážkám.

4.1.2 Vector Field Histogram (VFH)

Princip tohoto algoritmu spočívá v tom, že se na základě dat naměřených dálkoměrem vytvoří lokální mřížková mapa prostředí o poloměru r_{map} a se zvoleným rozlišením. Každá buňka této mapy tak obsahuje hodnotu obsazenosti odpovídající oblasti v reálném světě (viz obr. 4.1). Z této mapy se následně spočítá polární histogram, podle kterého se určí směr jízdy robotu.



Obrázek 4.1: Lokální mapa prostředí o poloměru $r_{map} = 10$

Celý proces tímto způsobem lze rozdělit na tři kroky: vytvoření primárního polárního histogramu, prahování primárního histogramu (vytvoření binární reprezentace), a nakonec výběr kandidátů pro směr pohybu.

Primární polární histogram

Polární histogram je rozdělen na sektory tak, aby každý z nich odpovídal úhlu α, který je volen takovým způsobem, aby $\frac{360}{\alpha}$ bylo celé číslo. Tedy například pro $\alpha = 15°$ histogram obsahuje 24 sektorů (viz obr. 4.2).

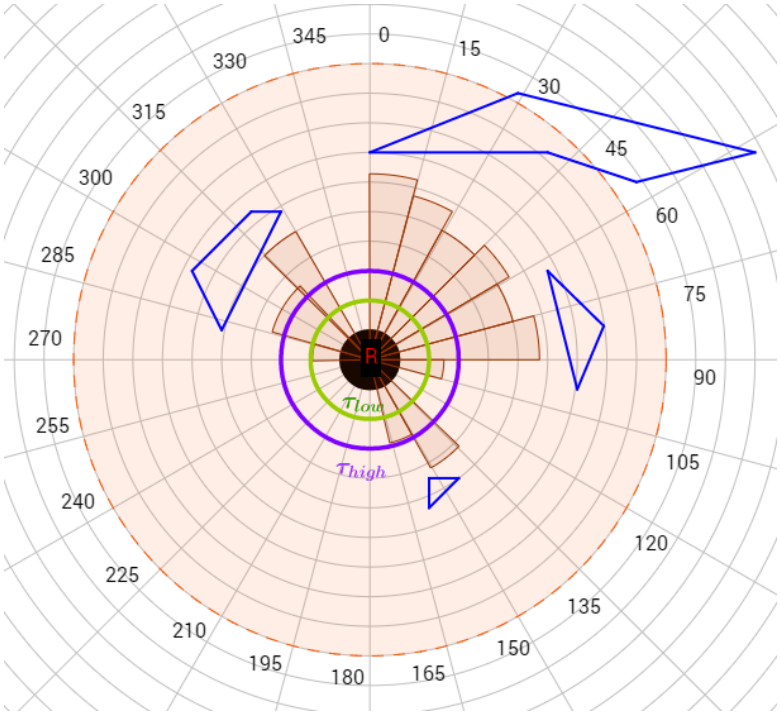
Pro každou buňku aktivního regionu, tedy lokální mapy vytvořené kolem robotu, se spočte směr, ve kterém se vůči středu nachází, a její význam. Směr je spočítán dle vztahu:

$$\beta_{i,j} = \text{atan2}(y_o - y_i, x_o - x_i), \tag{4.1}$$

kde x_o, y_o značí souřadnice střed mapy, x_i, y_i jsou souřadnice buňky.
Dále pak významnost buňky je:

$$m_{i,j} = c_{i,j}^2(a - bd_{i,j}^2), \tag{4.2}$$

kde $c_{i,j}$ je obsazenost buňky a $d_{i,j}$ vzdálenost buňky od pozice robotu.
Parametry a a b jsou voleny dle vztahu:



Obrázek 4.2: Polární histogram s vyobrazenými prahy τ_{low} (zeleně) a τ_{high} (fialově)

$$a - b\left(\frac{r_{map} - 1}{2}\right) = 1. \tag{4.3}$$

Aby robot nejel blízko k okrajům překážek, zavádí se kompenzace jeho velikosti pomocí poloměru robotu r_{rob} a minimální povolené vzdálenosti mezi robotem a překážkou d_{safety}:

$$\gamma_{i,j} = \arcsin\left(\frac{r_{rob} + d_{safety}}{d_{i,j}}\right)$$

Histogram je potom spočten vztahem:

$$H_k^p = \sum_{i,j \in C_\alpha} m_{i,j} h_{i,j}, \tag{4.4}$$

kde

$$h_{i,j} = \begin{cases} 1 & \text{jestli } k\alpha \in [\beta_{i,j} - \gamma_{i,j}; \beta_{i,j} + \gamma_{i,j}], \\ 0 & \text{jinak.} \end{cases}$$

Binární polární histogram

Aby bylo možné určit kandidáty pro další směr pohybu, primární histogram musí být převeden do binární podoby.

$$H^b_{k,i} = \begin{cases} 1 & \text{jestli } H^p_{k,i} > \tau_{high}, \\ 0 & \text{jestli } H^p_{k,i} < \tau_{low}, \\ H^b_{k,i-1} & \text{jinak.} \end{cases}$$

Výběr kandidátů

Kandidáty pro nový směr pohybu se volí dle toho, do jaké kategorie je zařazeno volné místo v binárním histogramu. Ty průjezdy, které mají velikost (tedy vzdálenost mezi pravým okrajem k_r a levým k_l) menší, než s_{max}, se nazývají úzké, jiné jsou naopak nazývány široké.

Pro úzké průjezdy lze zvolit pouze jednoho kandidáta, a to:

$$c_n = \frac{k_r + k_l}{2} \quad \text{centrální sektor}$$

Široké průjezdy mají tři možné kandidáty:

$$\begin{array}{ll} c_r = k_r + \frac{s_{max}}{2} & \text{pravý sektor,} \\ c_l = k_l - \frac{s_{max}}{2} & \text{levý sektor,} \\ c_t = k_t & \text{pokud } k_t \in [c_r; c_l] \end{array}$$

Nový směr pohybu se zvolí dle minimální ceny spočítané pro kandidáta c_i pomocí vztahu [18]

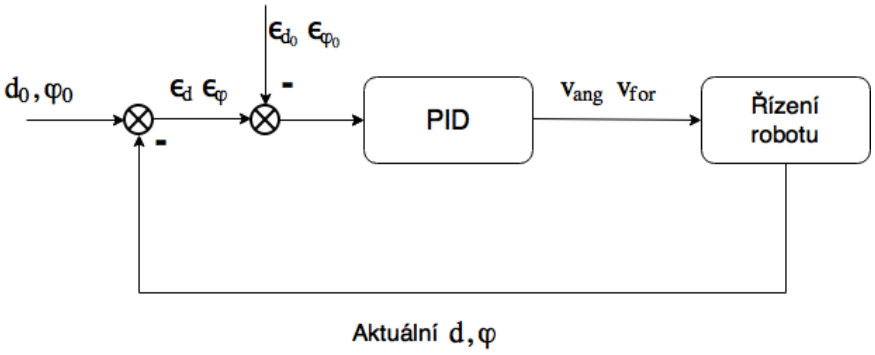
$$g(c_i) = \mu_1 \Delta\left(c_i, k_t\right) + \mu_2 \Delta\left(c_i, \frac{h}{\alpha}\right) + \mu_3 \Delta\left(c_i, k_{d,n-1}\right) \tag{4.5}$$

přičemž $k_{d,n-1}$ je minule zvolený kandidát, μ_1, μ_2, μ_3 jsou konstanty, zvolené dle vztahu $\mu_1 > \mu_2 + \mu_3$ a také platí, že

$$\Delta(c_1, c_2) = \min\left\{|c_1 - c_2|, |c_1 - c_2 - \frac{360°}{\alpha}|, |c_1 - c_2 + \frac{360°}{\alpha}|\right\} \tag{4.6}$$

4.2 PID regulátor

Za jízdy je třeba, aby byla mezi robotem a cílem udržována určitá vzdálenost d a hodnota úhlu ϕ byla co nejmenší, nejlépe nulová. Pro tyto účely je navržen PID regulátor [19], který pro aktuální odchylku od referenčních bodů spočte dopřednou a úhlovou rychlost, které se následně převedou na rychlosti motorů. Zjednodušený nákres algoritmu je zobrazen na 4.3. Pokud je ϵ_d, resp. ϵ_ϕ, v pásmu $[-\epsilon_{d_0}; \epsilon_{d_0}]$, resp. $[-\epsilon_{\phi_0}; \epsilon_{\phi_0}]$, pak bude výstup PID regulátoru nulový. Jinak je od odchylky odečtena hodnota ϵ_{d_0}, resp. ϵ_{ϕ_0}, což zaručí to, že robot nebude kmitat na místě, pokud bude blízko referenčního bodu, a řízení bude plynulé.



Obrázek 4.3: PID regulace

Výsledná dopřední, reps. úhlová, rychlost je vypočtena pomocí vztahu

$$V_{for,ang} = K_p error + K_i error \tag{4.7}$$

Kapitola 5

Simulace

Pro simulaci byl zvolen simulátor V-Rep [20]. V tomto prostředí byly vytvořeny dva roboty: jeden s vizuální značkou, druhý vybaven kamerou a laserovým dálkoměrem sledoval prvního (viz obr. 5.1). Robot s vizuální značkou je řízen uživatelem pomocí klávesnice. Referenční vzdálenost mezi dvěma roboty byla zvolena 1 m, požadovaná odchylka vizuální značky od středu kamery 0 rad. Maximální dopředná rychlost obou robotů byla nastavena na $2.5\,\mathrm{m\,s^{-1}}$

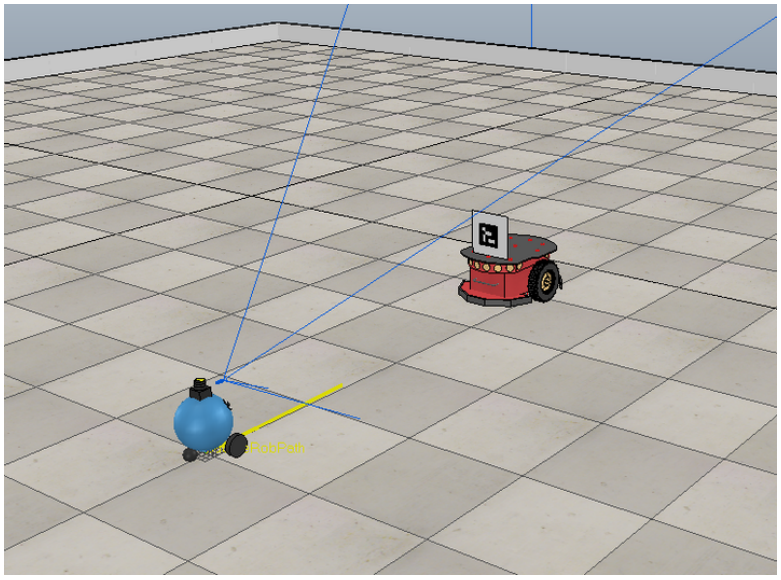
5.1 Porovnání algoritmů detekce

Oba algoritmy byly porovnávány při použití značky o velikosti 10x10 cm a rozlišením kamery 512x512 pixelů. Bylo porovnáno několik parametrů, a to maximální vzdálenost, na které je vizuální značka ještě detekovatelná, stabilní vzdálenost, při které je značka detekována bez přerušení. Dále byl porovnán maximální a stabilní úhel otočení vizuální značky vůči robotu a také průměrný výpočetní čas na jednu iteraci algoritmu. Výsledky jsou znázorněny v tabulce 5.1. Je patrné, že detekční schopnosti korelačního algoritmu jsou poněkud horší, než u ArUco detektoru. Avšak je třeba zdůraznit, že ArUco detektor sice dokázal rozpoznat vizuální značku na vzdálenosti přes 5 metrů od kamery, úspěšnost detekcí na tak velké vzdálenosti byla podprůměrná, přibližně 40 %. Co se týká úhlu otočení značky vůči robotu, tam se korelační algoritmus ukázal být lepší a stabilně detekuje značku při 60° otočení. ArUco má na hranicích detekovatelnosti, tj. při 70° otočení, pouze 15% úspěšnost.

Název algoritmu	Korelační algoritmus	ArUco detektor
Max. vzdálenost [m]	3.5	5.4
Stabilní vzdálenost [m]	3.5	3.8
Max. úhel [°]	60	70
Stabilní úhel [°]	60	55
Výpočetní doba [ms]	15	15

Tabulka 5.1: Porovnání korelačního algoritmu a ArUco detektoru

Hlavní rozdíl mezi těmito algoritmy spočívá v možnostech jejich využití. Zatímco ArUco detektor je zaměřený pouze na ArUco markery, korelační



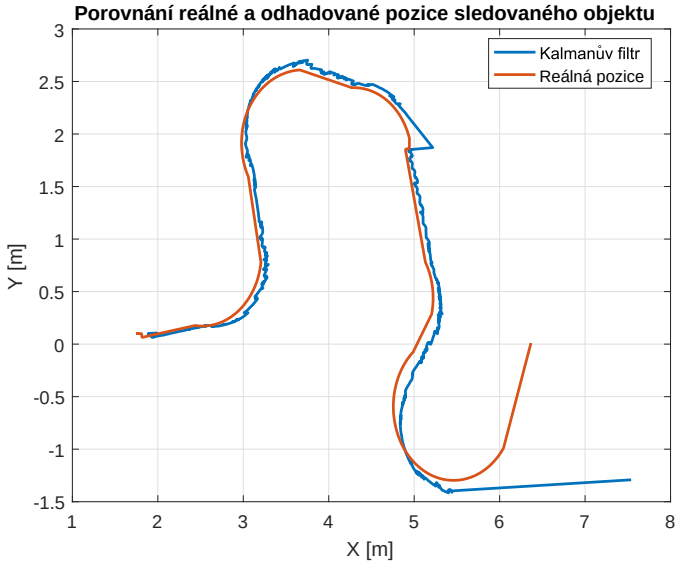
Obrázek 5.1: Roboty vytvořené v simulátoru V-Rep

algoritmus je obecnější a dá se použít na jakýkoliv typ vizuální značky. ArUco detektor umožňuje přidání dalších značek do své knihovny, ale jedná se pouze o binární obrázky podobné znázorněnému na obr. 3.2.2. Korelační algoritmus by měl detekovat i značku, kterou nelze reprezentovat jako bit-mapu, viz např. obr. 5.2.

Obrázek 5.2: Příklad značky pro korelační algoritmus

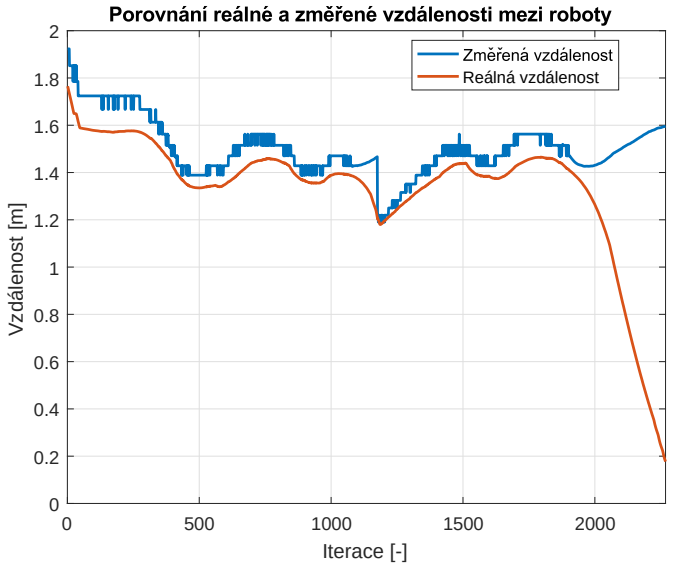
5.2 Porovnání naměřených a reálných hodnot

Jelikož se v implementovaném programu provádí různé výpočty, je třeba porovnat, jakou mají tyto hodnoty odchylku vůči reálným. Kalmanův filtr, popsaný v 3.4, predikuje polohu cíle a v případě neúspěšné detekce vizuální značky je umístění sledovaného objektu vůči robotu odvozeno právě z predi-



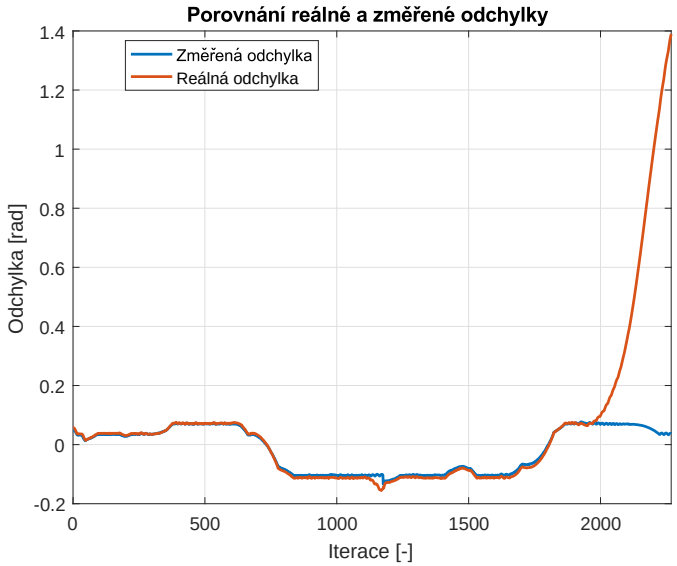
Obrázek 5.3: Poloha sledovaného objektu během simulace

kované polohy. Na obrázku 5.3 lze vidět, že výpočet pomocí Kalmanova filtru skoro po celou dobu simulace odpovídá reálné poloze sledovaného objektu. Velká odchylka pak je až na konci simulace, kdy vizuální značka brzo zmizela ze zorného pole robotu a nebylo možné ji v krátké době najít. Toto také odpovídá výsledkům, které jsou na 5.4 a 5.5, kde je vidět, že po cca 2000. iteraci je rozdíl mezi změřenými a reálnými hodnotami výrazný.



Obrázek 5.4: Vzdálenost mezi robotem a sledovaným objektem (d)

Na obrázku 5.4 je také vidět, že změřená vzdálenost není stejná, jako reálná. Je to způsobeno nepřesnou kalibrací referenční vzdálenosti mezi



Obrázek 5.5: Odchylka sledovaného objektu od středu kamery ϕ

následovaným a sledujícím robotem. Kdyby kalibrace byla přesnější, bylo by možné dosáhnout podobného výsledku, jako pro odchylku od středu kamery (viz obr. 5.5), kde měření odpovídá reálné hodnotě.

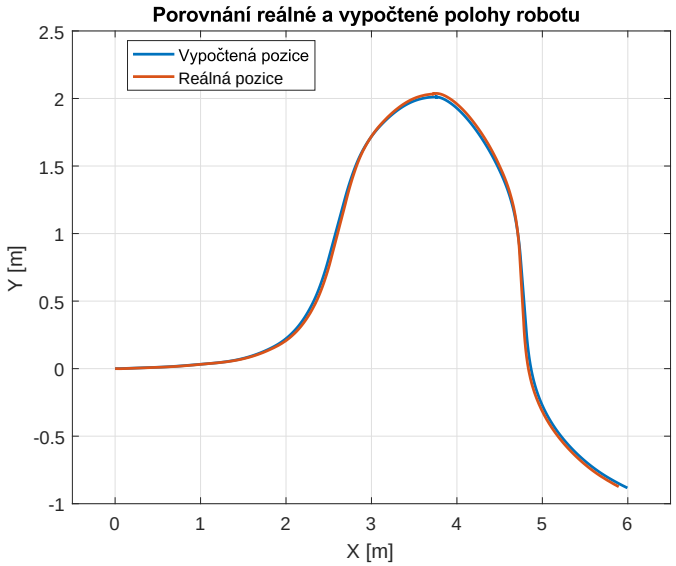
5.3 Měření polohy robotu

Jak bylo řečeno v 3.4, aby bylo možné spočítat polohu cíle vůči robotu, je třeba znát polohu robotu v globálních souřadnicích. Ačkoliv simulátor V-Rep nabízí možnosti pro přesné měření polohy objektů ve světových souřadnicích, v reálném světě tato možnost nemusí být použitelná, protože např. při navigaci robotu uvnitř budov nelze použít GPS. Proto byl v sekci 2.2 zaveden relativní souřadnicový systém, ve kterém se poloha robotu počítá. Pro toto měření lze v simulátoru odečítat vnitřní polohu rotačních kloubu robotu, na které jsou přidělány kola. V případě robotu s diferenciálním řízením, jeho pozici lze spočítat jako:

$$\begin{bmatrix} x \\ y \\ h \end{bmatrix} = \begin{bmatrix} x + \frac{(\Delta R + \Delta L) r_k}{2} \cos(h) \\ y + \frac{(\Delta R + \Delta L) r_k}{2} \sin(h) \\ h + \frac{(\Delta R - \Delta L) r_k}{2 r_r} \end{bmatrix}, \tag{5.1}$$

přičemž ΔL a ΔR značí změnu vnitřní polohy levého a pravého kloubů za čas Δt, r_k je poloměr kola a r_r je poloměr robotu, neboli polovina vzdálenosti mezi koly.

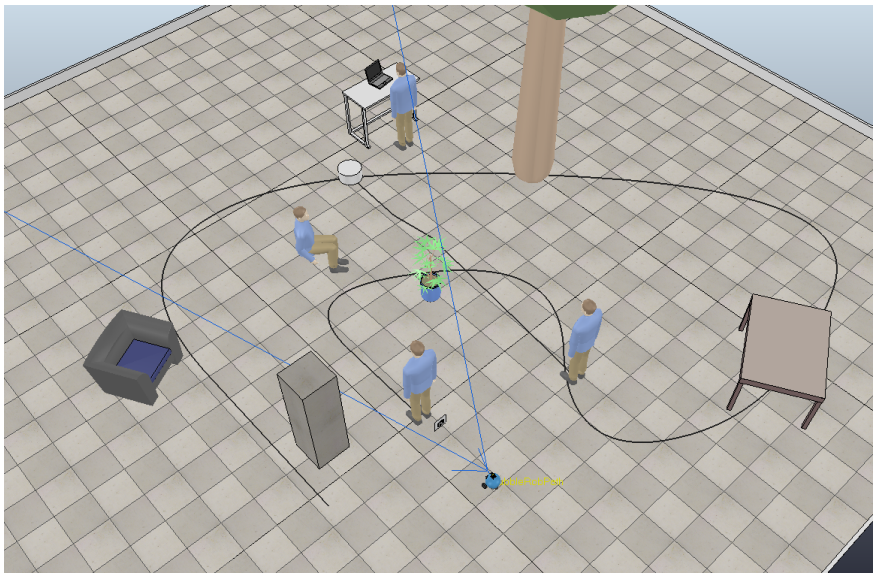
Porovnání reálné, tj. v simulátoru změřené, a vypočtené polohy robotu ve zvoleném souřadnicovém systému lze vidět na obrázku 5.6. Je patrné, že pro použitý typ robotu a relativní souřadnicovou soustavu je výpočet přesný a při použití přesných enkodérů lze tento výpočet aplikovat i na reálném robotu.



Obrázek 5.6: Poloha robotu během simulace

5.4 Trajektorie robotu při jízdě s překážkami

Po ověření funkčnosti detektorů a řízení robotu byl v simulátoru vytvořen model prostředí, ve kterém by robot měl sledovat člověka se značkou. Do scény byli umístěni i jiní lidé, a také různé překážky: kytka, stůl, strom, křeslo, sloup, jak je vidět na 5.7.

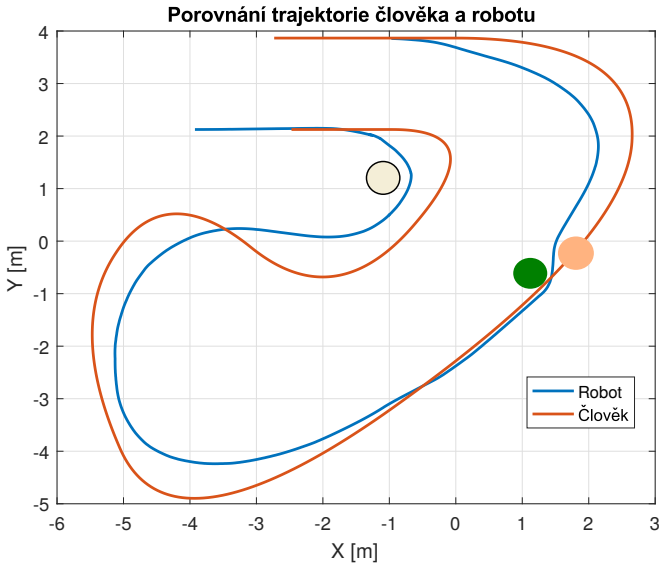


Obrázek 5.7: Simulace s člověkem a překážkami

Člověk měl předem definovanou cestu, kterou přesně dodržoval nehledě na překážky, proto procházel skrz ostatní objekty bez jakýchkoliv následků.

Robot se ostatním objektům měl vyhýbat.

Trajektorie člověka a robotu jsou na obrázku 5.8. Šedě je na obrázku znázorněna kytka, zeleně strom a růžově kolemjdoucí, který během simulace přišel na uvedenou pozici. Jak je vidět, robot se snažil dodržovat trajektorii, podle které šel sledovaný člověk a přitom dokázal objíždět překážky, které mu v tom bránily. Nejproblematičtější část simulace byla na začátku, kdy člověk strmě zahnul doprava. V ten okamžik byla vizuální značka špatně detekovatelná, protože byla otočena vůči robotu téměř o 90°. Navíc v určitém časovém intervalu byla značka skryta za listím, takže ji použitý algoritmus nedokázal najít ve snímku. Avšak tady je vidět, jak přínosné je použití Kalmanova filtru, který dokázal předpovědět na základě mála měření, kde se člověk nachází, proto ho robot rychle našel a sledoval i nadále.



Obrázek 5.8: Trajektorie robotu a člověka

Dalším problematickým úsekem byla cesta mezi stromem a kolemjdoucím. Jelikož sledovaný člověk mohl procházet skrz objekty, došlo k tomu, že, když mu v cestě stal kolemjdoucí, prošel skrz něj a pokračoval dále. Značka tedy opět nebyla detekovatelná. Navíc mezi stromem a kolemjdoucím byl velice úzký průjezd a robot musel snížit rychlost, aby tam dokázal projet. Nakonec však robot nalezl sledovaného člověka a jel za nim až do konce simulace.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout software pro autonomní řízení robotu následujícího člověka.

Pro detekci člověka v obrazu bylo rozhodnuto použit vizuální značku, která byla robotem známa, funkčnost takového řešení byla ověřena simulací. Robot dokáže spočítat vzdálenost a směr, ve kterém se detekovaná značka vůči němu nachází. Díky dálkoměru robot také ví, jestli na cestě k cíli jsou překážky. Pokud ano, pomocí Vector Field Histogram algoritmu je vypočten nový směr jízdy tak, aby robot nenarazil na žádnou překážku a zároveň byl co nejblíže člověku. Na základě spočtených parametrů se pomocí PID regulátoru vyhodnotí rychlosti, které je třeba aplikovat na motory, aby robot dosáhl cílové pozice vůči sledovanému člověku. Navíc pro případ, kdy robot nedetekuje značku, a tedy ani nedokáže spočítat vzdálenost k cíli, je použit Kalmanův filtr, který předpovídá polohu cíle na základě předchozích měření.

V simulátoru bylo provedeno několik testů s roboty, nastavení simulací a výsledky některých testů jsou popsány v kapitole 5. Průměrná vzdálenost d mezi roboty byla 1.99 m a odchylka sledovaného robotu od středu kamery ϕ stanovila 0.0023 rad. Jelikož rychlost robotů byla stejná a počáteční vzdálenost mezi nimi byla 2 m, sledující robot se nedokázal přiblížit referenční hodnotě 1 m. Dále pak byla změřena průměrná absolutní odchylka mezi reálnou a vypočtenou polohou sledujícího robotu, která stanovila 0.17 m. Rozdíl mezi reálnou a spočtenou hodnotou směru jízdy robotu byla -0.02 rad. Nakonec byl spočítán rozdíl mezi reálnou a vypočtenou pomocí Kalmanova filtru polohou sledovaného robotu, ten stanovil 0.32 m. Tak velká odchylka byla způsobena tím, že byly simulovány situace, kdy značka zmizí ze zorného pole robotu. Avšak robot dokázal pokaždé najít značku znovu a pokračovat v jízdě.

Tato práce byla velkým přínosem hlavně proto, že v ní byla možnost vyzkoušet různé techniky používané pro řízení mobilních robotů. Navíc díky této práci jsme se seznámili s simulačními prostředí pro robotiku a OpenCV knihovnou, která je běžně používána pro implementaci softwaru pro počítačové vidění a strojové učení.

Příloha A

Přehled algoritmů pro vyhýbání se překážkám

method		model fidelity			view	other requisites			sensors	tested robots	performance		remarks
		shape	kinematics	dynamics		local map	global map	path planner			cycle time	architecture	
Bug	Bug1 [101, 102]	point			local				tactile				very inefficient, robust
	Bug2 [101, 102]	point			local				tactile				inefficient, robust
	Tangent Bug [82]	point			local	local tangent graph			range				efficient in many cases, robust

Obrázek A.1: Souhrn algoritmů pro vyhýbání se překážkám [3, s. 287–290]

method		model fidelity			view	other requisites			sensors	tested robots	performance		remarks
		shape	kinematics	dynamics		local map	global map	path planner			cycle time	architecture	
Vector Field Histogram (VFH)	VFH [43]	simplistic			local	histogram grid			range	synchro-drive (hexagonal)	27 ms	20 MHz, 386 AT	local minima, oscillating trajectories
	VFH+ [92, 150]	circle	basic	simplistic	local	histogram grid			sonars	nonholonomic (GuideCane)	6 ms	66 MHz, 486 PC	local minima
	VFH* [149]	circle	basic	simplistic	essentially local	histogram grid			sonars	nonholonomic (GuideCane)	6 ... 242 ms	66 MHz, 486 PC	fewer local minima
Bubble band	Elastic band [86]	C-space			global		polygonal	required		various			
	Bubble band [85]	C-space	exact		local		polygonal	required		various			

Obrázek A.1: Souhrn algoritmů pro vyhýbání se překážkám [3, s. 287–290]

method		model fidelity			view	other requisites			sensors	tested robots	performance		remarks
		shape	kinematics	dynamics		local map	global map	path planner			cycle time	architecture	
Curvature velocity	Curvature velocity method [135]	circle	exact	basic	local	histogram grid			24 sonars ring, 30° FOV laser	synchro-drive (circular)	125 ms	66 MHz, 486 PC	local minima, turning into corridors
	Lane curvature method [87]	circle	exact	basic	local	histogram grid			24 sonars ring, 30° FOV laser	synchro-drive (circular)	125 ms	200 MHz, Pentium	local minima
Dynamic window	Dynamic window approach [69]	circle	exact	basic	local	obstacle line field			24 sonars ring, 56 infrared ring, stereo camera	synchro-drive (circular)	250 ms	486 PC	local minima
	Global dynamic window [44]	circle	(holonomic)	basic	global		C-space grid	NF1	180° FOV SCK laser scanner	holonomic (circular)	6.7 ms	450 MHz, PC	turning into corridors

Obrázek A.1: Souhrn algoritmů pro vyhýbání se překážkám [3, s. 287–290]

method				model fidelity			view	other requisites			sensors	tested robots	performance		
Other	Gradient method [89]	Global nearness diagram [110]	Nearness diagram [107, 108]	ASL approach [122]	Schlegel [128]	shape	kinematics	dynamics		local map	global map	path planner			cycle time	architecture	remarks
	circle	circle (but general formulation)	circle (but general formulation)	polygon	polygon												
	exact	(holonomic)	(holonomic)	exact	exact												
	basic			basic	basic												
	global	global	local	local	global												
		grid		grid													
	local perceptual space	NF1			grid												
	fused			graph (topological), NF1	wavefront												
	180° FOV distance sensor	180° FOV SCK laser scanner	180° FOV SCK laser scanner	2x 180° FOV SCK laser scanner	360° FOV laser scanner												
	nonholonomic (approx. circle)	holonomic (circular)	holonomic (circular)	differential drive (octagonal, rectangular)	synchrodrive (circular), tricycle (forklift)												
	100 ms (core algorithm: 10 ms)			100 ms (core algorithm: 22 ms)													
	266 MHz, Pentium			380 MHz, G3													
			local minima	turning into corridors	allows shape change												

Obrázek A.1: Souhrn algoritmů pro vyhýbání se překážkám [3, s. 287–290]

Příloha B

Literatura

[1] Derek Bradley and Gerhard Roth. Adaptive thresholding using the integral image. *Journal of Graphics Tools*, 12(2):13–21, 2007.

[2] Opencv: Contours hierarchy. `http://docs.opencv.org/trunk/d9/d8b/tutorial_py_contours_hierarchy.html`.

[3] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT Press. 272–290, 2004.

[4] J Blanco, W Burgard, R Sanz, and JL Fernandez. Fast face detection for mobile robots by integrating laser range data with vision. In *Proc. of the International Conference on Advanced Robotics (ICAR)*, volume 2, pages 953–958. Citeseer, 2003.

[5] Muhammad Sarmad Hassan, Mafaz Wali Khan, and Ali Fahim Khan. Design and development of human following robot. *Student Research Paper Conference*, 2, Jul 2015.

[6] Andrej Babinec, Ladislav Jurišica, Peter Hubinský, and František Duchoň. Visual localization of mobile robot using artificial markers. *Procedia Engineering*, 96:1–9, 2014.

[7] Xu Liu, David Doermann, Huiping Li, K. C. Lee, Hasan Ozdemir, and Lipin Liu. A novel 2d marker design and application for object tracking and event detection. *Lecture Notes in Computer Science*, 5358.

[8] S. Garrido-Jurado, R. Muñoz-Salinas, F.j. Madrid-Cuevas, and M.j. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

[9] Opencv: About. `http://opencv.org/about.html`.

[10] F. Jurie and M. Dhome. A simple and efficient template matching algorithm. *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001.*

[11] Template matching. `http://docs.adaptive-vision.com/4.7/studio/machine_vision_guide/TemplateMatching.html`.

[12] Opencv: Detection of aruco markers. `http://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html`.

[13] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244 – 256, 1972.

[14] Paul Prober and Bill Wellman. Optimum pinhole camera design. *Hue Candela*, 2002.

[15] Erik Cuevas, Daniel Zaldivar, and Raul Rojas. Kalman filter for vision tracking. Aug 2005.

[16] Rogger Labbe. *Kalman and Bayesian Filters in Python.*

[17] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[18] R.J. van Breda and W.J. Smit. Applicability of vector field histogram star (vfh*) on multicopters. *International Micro Air Vehicles, Conferences and Competitions*, 2016.

[19] Rich LeGrand. Closed-loop motion control for mobile robotics. *Circuit Cellar*, (169):34–46, Aug 2004.

[20] Coppelia robotics v-rep. `http://www.coppeliarobotics.com/index.html`.