Hledání sloupku

Marija Pajdaković a Lamija Čaušević

České vysoké učení tehnické v Praze
Kybernetika a robotika
Labaratoře robotiky

17. května 2021

1 Úvod

Cílem projektu bylo naprogramovat Turtlebota (Kobuki) v simulátoru Gazebo, používáním ROSu. Vybraly jsme úlohu střední složitosti. Naším úkolem bylo najít sloupek dané barvy přičemž v prostředí mohli být i sloupky jiných barev, kterým se Kobuki měl vyhnout.

2 Návrh

Úlohu jsme nejdříve rozdělily na dvě části.
První část byla naprogramovat tu základní verze. Kobuki měl najít sloupek předem dané barvy, a dojit k němu. Danou barvu vybiráme na začátku programu. Řešení této verze spočívalo v tom, že jsme udělaly threshold masku RGB obrázku (obrázek máme z funkce *get_rgb_image*() z třídy Turtlebot) a dostaly obrázek ve kterém je viditelný jenom ten hledaný sloupek. Potom jsme otáčely Kobukia tak dlouho, dokud mu ten sloupek nebyl přímo ve středu zorného pole (viz obrázek **1a**). Když se toto stalo, on jel rovně k sloupku a program se ukončil se zprávou "Got it!"

Druhá část spočívala v tom, že se Kobuki měl vyhnout překážkám tj. sloupkům jiných barev. Tady jsme zase udělaly masku RGB obrázku ale tentokrát tak, aby viditelné byli jenom ty sloupky, kterým se Kobuki měl vyhnout (viz obrazek **1b)**. Pokud nějaký takový sloupek viděl a byl příliš blízko od něj, zastavil se a začal se otáčet, dokud daný sloupek nebyl mimo jeho zorného pole.

Obrázek 1: Masky RGB obrázku

(a) Threshold

(b) Mask cone

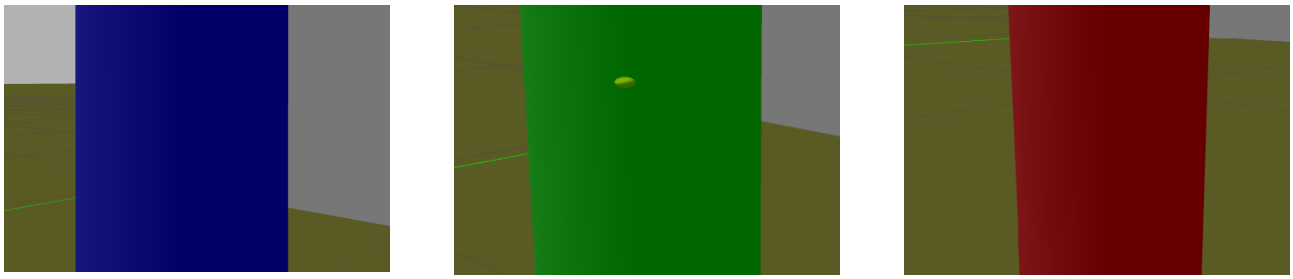
Když obě častí fungovaly bez problémů, daly jsme je dohromady do programu *kobuki.py*.

3 Řešení problému

Abychom mohly řešit problém se hledáním sloupků, nejdřív jsme našly přesné BGR hodnoty jednotlivých barev. Používaly jsme online aplikace color picker[1] a obrázky z obrázku **2**.

Obrázek 2: Sloupky jednotlivých barev

(a) (b) (c)



Hodnoty barev ve simulaci jsou:

$$BLUE = [102, 0, 0]$$
$$RED = [0, 0, 102]$$
$$GREEN = [0, 102, 0]$$

Dane barvy jsme definovaly ve záhlaví programu.

3.1 Matematický popis

Pro výpočet vzdáleností sloupků od robota je možné použít hloubkový senzor a funkce *get_depth_image*() z třídy Turtlebot. Jelikož funkce někdy selhávala a výstup byl NAN, rozhodly jsme se taky používat výpočet vzdálenosti z regrese (funkce *ray_distance*()). Abychom to udělaly, musely jsme použít následující matematické vzorce[2]:

$$\frac{1}{\lambda_i} x_i = K^{-1} u_i$$

Kde u_1 a u_2 jsou okrajové pixely (v homogenních souřadnicích), x_1 a x_2 hledané paprsky. Dále vypočítáme úhel, který tento dva paprsky svírají.

$$cos\alpha = \frac{\frac{1}{\lambda_1 \lambda_2}}{\frac{1}{\lambda_1 \lambda_2}} \frac{r}{sin\frac{\alpha}{2}}$$

Lambdy se nám vykrátí, a poloměr víme, že je $r = 2.5$ cm.

3.2 Funkce

Teď napíšeme nazve všech funkcí, její vstupní a výstupní parametry a krátký popis toho, co dělají. Podrobnější popis funkcí je uveden jako komentář v kódu.

[1] https://imagecolorpicker.com/
[2] https://cw.fel.cvut.cz/wiki/ media/courses/b3b33lar/petrik lar21 en.pdf

def obstacle(image):
Najde překážku na cestě.
Vstupní parametr: binární obrázek kde jsou viditelné jenom překážky (*mask_cones* maska)
Výstupní parametry:
1) *True*, pokud Kobuki narazil na překážku nebo *False* opačně
2) směr, kterým se Kobuki má pohybovat aby se vyhnoul překážce. Např. pokud je překážka vlevo od něho, směr mu řekne, že má jít doprava.
def ray_distance(x, y, w, h, K_RGB)

Vypočítá souřadnici z tj. vzdalenost robotu od bodu komponenty
Vstupní parametry: 1) bod komponenty nejvýše vlevo
2) nejvyšší bod komponenty
3) výška komponenty
4) šířka komponenty
5) matice, kterou dostaneme funkcí get_rgb_K(self) z Turtlebot třídy
Výstupní parametr: vzdálenost daného bodu od robota

def threshold_image$(turtle)$
Udělá masku takovou, že viditelný bude jenom cilový sloupek.
Vstupní paramter: Turtlebot , inicializovaný na začátku main programa
Výstupní parametr: maska rgb obrázku který dostaneme z kamery Turtlebotu. Touto maskou uděláme jenom cílový sloupek "viditelný"(bilá barva), všechno ostatní bude černé.

def mask_cones(turtle, color)
Tato funkce funguje stejně jako funkce *threshold image(turtle)*, jenže tady "viditelné"(bíle) budou všechny překážky. **Vstupní paramter:**
1) Turtlebot , inicializovaný na začátku main programa
2) Barva cíleného slopuku
Výstupní parametr: maska rgb obrázku který dostaneme z kamery Turtlebotu. Touto maskou uděláme překážky 'viditelné' (bíle), všechno ostatní bude černé.

def column_found(image)
Najde cilový sloupek.
Vstupní parametr: binární obrázek, kde "viditelný"je jenom cílový sloupek (threshold maska)
Výstupní parametr: *True*, pokud je cilový sloupek ve středu zorného pole robota nebo *False* opačně

def moment(bin_img) Najde střed cilového sloupku.
Vstupní paramter: binární obrázek (threshold maska)
Výstupní parametry: **1)** x-ová souřadnice středu
2) y-ová souřadnice středu

def get_depth(turtle, x, y)
Vypočítá souřadnici z tj. vzdálenost robotu od bodu komponenty z hloubkové kamery
Vstupní paramtery:
1 Turtlebot , inicializovaný na začátku main programa
2 x-ová souřadnice bodu, ke kterému chceme spočítat z souřadnici (hloubku)
3 y-ová souřadnice bodu, ke kterému chceme spočítat z souřadnici (hloubku)
Výstupní parametr: z-ová souřadnice tj. hloubka daného bodu

4 Způsob spuštění

Pro spouštění potřebujeme 2 terminály. Jmeno našého programu je*kobuki.py* . Použijeme následující příkazy:

1) Spuštění simulace:
singularity shell robolab_melodic_lar.simg
source /lar_ws/devel/setup.bash
roslaunch lar lar_2021_simulation_bringup.launch gui:=true

2) Spuštění Kobukia:
singularity shell robolab_melodic_lar.simg
source /lar_ws/devel/setup.bash
rosrun robolab_turtlebot kobuki.py

5 Diskuze

Jak už bylo zmíněno, problém jsme rozdělily na dvě části. Když jsme téhle dvě části dávaly dohromady, stal se následující problém:
Pokud Kobuki našel cílový sloupek a na cestě se našla překážka, on se u překážky zastavil aby ji objížděl.Začal se otáčet, aby jel dal.Potom se ale taky začal otáčet na druhou stranu, aby zase našel cílový sloupek a tím se zacyklil. Druhými slovy, pokud nastane situace, že vidí cilový sloupek a zároveň vidí i překážku které se snaží vyhnout, skončí v nekonečné smyčce. Tento problém jsme vyřešily tím, že jsme Kobukia po každému vyhýbání se překážky posunuly trošku rovně. Teprvé potom se začne zase otáčet, aby cílovy sloupek našel a jel k němu.

Algoritam funguje pro různé pozice Kobukia, stane se ale problém pokud on nevidí přímo ten cílový sloupek, třeba s pozice (0.5,1.5). Situace je ukázaná na obrázku **3**. V tomto případě modrý sloupek dělá problém, protože mu brání , aby viděl cílový sloupek.

Obrázek 3: Kobuki na pozice (0.5,1.5)