Rozpoznávanie dopravných značiek

Mário Kapusta

21. apríla 2013

Obsah

Zoznam tabuliek

Zoznam obrázkov

Abstrakt

V praci sme sa zaoberali

1 Počítačové videnie

Nejaký obkec o počítačovom videní

1.1 História počítačového videnia

Niečo krátke o histórii počítačového videnia

1.2 Hlavné témy počítačového videnia

Obkec o rozdelení počítačového videnia a rôznych odvetviach venovania

1.2.1 Transformácia

Niečo o trnaformácii.

1.2.2 Filtrovanie a kompresia

Niečo o kompresii.

1.2.3 Vylepšovanie obrazu

Niečo o vylepšovaní obrazu.

1.2.4 Rozpoznávanie objektov

Niečo o rozpoznávaní objektov

1.2.5 Pozíciovanie

Niečo o rozpoznávaní poziciovani

1.3 Technológie

Niečo o o technológiách rozpoznávania vo všeobecnosti

1.3.1 OpenCV

Niečo o opencv - textik k tomu: http://simplecv.tumblr.com/post/19307835766/opencv-vs-matlab-vs-simplecv

1.3.2 Matlab

Niečo o matlabe

1.3.3 SimpleCV

Niečo o simplecv

2 Rozpoznávanie ojektov

3 OpenCV, Android a Java - inak to nazvat

Cieľom práce je vypracovať komplexný návrh riešenia pre vyhľadávanie a rozpoznávanie dopravného značenia a taktiež vytvoriť funkčnú aplikáciu, ktorá bude schopná rozpoznať zvislé dopravné značenia. Táto aplikácia bude naprogramovaná v jazyku Java a bude spustiteľná na operačnom systéme Android 2.3, ktorý je určený pre mobilné zariadenia. Computer vision (počítačové videnie), nám zaručí open-source knižnica OpenCV.

3.1 Matematické metódy

Mnoho matematických metód sa bude priamo vysvetlovať pri predstavovaní danej OpenCV funkcionality. V tejto sekcii si predstavíme také matematické metódy ktoré nám pomůžu lepšie sa orientovať pri opise konkrétnych funkcionalít OpenCV.

3.1.1 Konvolúcia

Konvolúcia je matematická metóda, ktorá systematicky prechádza celý obraz a na výpočet novej hodnoty bodu využíva malé okolie O reprezentatívneho bodu. Táto hodnota

je zapísaná do nového obrazu. Diskrétna konvolúcia má tvar:

$$g(x,y) = \sum_{(m,n)} \sum_{(e^0)} h(x-m, y-n) f(m,n)$$

kde f predstavuje obrazovú funkciu pôvodného obrazu, g predstavuje obrazovú funkciu nového obrazu, h predstavuje konvolučnú masku alebo konvolučné jadro, h nám udáva koeficienty jednotlivých bodov v okolí O. Najčastejšie sa používajú obdĺžnikové masky s nepárnym počtom riadkov a stĺpcov, pretože v tom prípade môže reprezentatívny bod ležať v strede masky.

Transformácie v lokálnom okolí bodu sa delia na dve skupiny:
Vyhladzovanie – tieto metódy sa snažia potlačiť šum v obraze, ale rozostrujú hrany.
Ostrenie – detekcia hrán a čiar, ale zosilňuje šum.

Podľa matematických vlastností môžeme metódy predspracovania rozdeliť na
Lineárne metódy – novú jasovú hodnotu bodu počítajú ako lineárnu kombináciu vstupných bodov. Napr.: priemerovací filter
Nelineárne metódy – berú do úvahy len body s určitými vlastnosťami. Napr.: mediánový filter. [2]

3.1.2 Aproximácia

3.2 Funkcionalita OpenCV

OpenCV je open source knižnica počítačového videnia. Knižnica je napísaná v programovacích jazykoch C a C++. Aktívne sa pracuje na rozhraniach pre Python, Ruby, Matlab, Javu a iných programovacích jazykoch. V našej práci sme sa sústredili na verziu pre programovací jazyk Java, ktorý sa používa pri tvore aplikácii pre Android OS. [3]

OpenCV knižnica bola navrhnutá tak, aby funkcie použité v tejto knižnici, boli čo najefektívnejšie a čo najviac zamerané na real-time aplikácie. Knižnica je napísaná v opti-

Premenná	Dátový typ	Popis
src	Mat	Vstup je 8-bitový, 16-bitový obraz alebo formát čísla s plávajúcou desatinou čiarkou.
dst	Mat	Výstupný obraz s rovnakými parametrami ako na vstupe.
code	int	Farebné spektrum ktoré do ktorého požadujeme obraz previesť.

Tabuľka 1: Tabulka znázorňuje vstupy funkcie cvtColor

malizovanom jazyku C a tak môže jednoducho využiť aj silu viacjadrových procesorov. Taktiež existujú knižnice, špeciálne určené pre procesory s architektúrou Intel. IPP (Integrated Performance Primitives) knižnice sa skladajú z nízko levelových optimalizovaných postupov a rôznych algoritmických olastí, ktoré pracujú na procesoroch s architektúrou Intel oveľa efektívnejšie. [3]

Jeden z hlavných cieľov OpenCV je sprístupniť jednoducho použiteľné prostredie ktoré pomôže developerom ľahko a rýchlo budovať aplikácie s použitím počítačového videnia pre rôzne použitia v oblasti, medicíny, bezpečnosti, robotiky, dopravy, priemyselnej výroby a iných, pre ktoré ma OpenCV dokonca aj špecifické funkcionality. [3]

Pre olasť rozpoznávania ojektov sú taktiež mnohé špecifické funkcionality. Pri problematike rozpoznávania zvislích dopravných značení sme niektoré z nich použili a preto je potrebné si pre lepšie pochopenie problematiky tieto funkcie vysveliť podrobnejšie.

3.2.1 cvtColor

Funkcia *cvtColor* prevedie obraz z jedného farebného spektra do iného. Je to jedna z najpoužívanejších funkcií, keďže na rozpoznávanie objektov je potrené si obraz pripraviť cez mnohé farebné filtre. Vstupné parametre je možné pozorovať pri tabuľke 1. [11] [16]

$$BGR \leftrightarrow HSV$$

$$V \leftarrow max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V - min(R,G,B)}{V}, & \text{pokiaľ } V \neq 0 \\ 0, & \text{pokiaľ } V = 0 \end{cases}$$

$$H \leftarrow \begin{cases} \frac{60(G-B)}{V - min(R,G,B)}, & \text{pokiaľ } V = R \\ \frac{120+60(B-R)}{V - min(R,G,B)}, & \text{pokiaľ } V = G \\ \frac{240+60(R-G)}{V - min(R,G,B)}, & \text{pokiaľ } V = B \end{cases}$$

Pokiaľ $H < 0$, tak $H = H + 360$

Na výstup pôjde $0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 1$

Tabuľka 2: Konverzia RGB modelu na HSV[7][11][16]

Pri používaní funkcie *cvtColor*, je potrebné si určiť o akú konverziu ide. OpenCV, už má k dispozícii predpripravené konštanty, ktoré konverziu lepšie vyjadrujú. Matematický prepočet si OpenCV už spraví v jadre. Konverzií je v OpenCV naprogramovaných už mnoho, my si predstavíme matematický model konverzie, ktorú v našom prípade rálne využijeme. Jedná sa o konverziu z BGR(pri OpenCV je poradie kanálov pre model RGB zoradený opačne) do farebného modelu HSV a späť. [11] [16]

V prípade 8 a 16 bitového obrazu je potrené jednotlivé kanály R,G a B previesť do formátu s plávajúcou desatinou čiarkou a zmenšiť rozsah od 0 do 1.

3.2.2 Canny

Hlavná úloha funkcie *Canny* je vyhľadávať okraje, kontúry a hrany všetkých objektov. Pri kombinácii s rôznymi filtrami, môžeme docieliť, vyhľadanie hrán úmyselného objektu.

Na rozoznávanie sa využíva algoritmus *Canny86*. [10] [16]

Kontúrový alebo hranový detektor by mal spĺňať tri kritéria, ktoré určil John Canny.

1. Detekčné kritérium, detektor nesmie zabudnúť na významnú hranu a na jednu hranu môže byť maximálne jedna odozva.
2. Lokalizačné kritérium, rozdiel medzi skutočnou a nájdenou hranou má byť minimálny.
3. Kritérium jednej odozvy.

Cannyho detektor využíva konvolúciu s dvojrozmerným Gaussianom a deriváciu v smere gradientu. Poskytuje informácie o smere a veľkosti hrany. Nech G je dvojrozmerný Gaussian. Nech Gn je prvá derivácia G v smere gradientu

$$G_n = \frac{\delta G}{\delta n} = n \bigtriangledown G$$

kde n je smer gradientu, ktorý dostaneme nasledovne

$$n = \frac{\bigtriangledown (G * f)}{| \bigtriangledown (G * f)|}$$

Hranu dostaneme v bode, kde funkcia $G_n * f$ dosiahne lokálne maximum, a druhá derivácia sa rovná nule.

$$\frac{\delta^2}{\delta n^2} G * f = 0$$

Pre silu hrany platí:

$$|G_n * f| = | \bigtriangledown (G * f)|$$

Kritérium jednej odozvy sa dosahuje následne prahovaním. [2] [4] [5]

Vstupné parametre je možné pozorovať pri tabuľke 3. Najmenšia hodnota medzi *threshold1* a *threshold2* je použitá na prepájanie kontúr. Tá najväčšia hodnota je použitá ako

Premenná	Dátový typ	Popis
image	Mat	Vstup je 8-bitový obraz s jedným farebným kanálom.
edges	Mat	Výstup je mapa všetkých nájdených kontúr.
threshold1	double	Prvá prahová hodnota (threshold).
threshold2	double	Druhá prahová hodnota (threshold).

Tabuľka 3: Tabulka znázorňuje vstupy funkcie canny

začínajúci segment najsilnejších kontúr. Pri správnom nastavení, sa dá dosiahnuť pomerne kvalitné odstránenie nepotrených kontúr. [10] [16]

3.2.3 GaussianBlur

Vyhladzuje obraz pomocou *GaussianBlur* filtra. [14] [16]

GaussianBlur filter funguje na princípe $N * N$ konvolúcie pri ktorej sa každý pixel prehodnotí na základe *Gaussian* funkcie. Táto funkcia tak prevedie rozostrenie pre každý pixel obrazu. [17]

$$H(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{(x^2)+(y^2)}{2\sigma^2}}$$

Princíp konvolúcie 2D obrazu je postavený na tom, že sa systematicky snažíme spracovávať okolie pixelu a dostať výslednú hodnotu z okolia reprezentatívneho bodu. Konvolúcia sa často používa pri spracovávaní obrazu, ako je vyhladzovanie obrazu, ostrenie, detekcia hrán a obrázkov. [1] [2]

Vstupné parametre pre *GaussianBlur* je možné pozorovať pri tabuľke 4. Pri premennej *ksize* si môžeme napríklad nastaviť veľkosť matice, ktorá sa bude pri konvolúcii používať. Veľkosť matice pri konvolúcii ovplivní rozostrenie. Čím väčšiu maticu používame, tím väčšie rozostrenie dostaneme. [14] [16]

Premenná	Dátový typ	Popis
src	Mat	Vstup je obraz s ľubovolným počtom farebných kanálov.
dst	Mat	Výstup s rovnakými parametrami ako bol vstup.
ksize	Size	Veľkosť Gaussian jadra. Matica konvolúcie.
sigmaX	double	Smerodajná odchýlka Gaussian jadra v smere X.
sigmaY	double	Smerodajná odchýlka Gaussian jadra v smere Y.

Tabuľka 4: Tabulka znázorňuje vstupy funkcie GaussianBlur

3.2.4 inRange

Funkcia *inRange* zisťuje, či sa prvky poľa nachádzajú medzi prvkami ďalších dvoch polí.

Funkcia kontroluje rozsah nasledujúco:

- Pre každý prvok vstupného pola s jedným kanálom

$$dst(I) = lowerb(I)_0 \leq src(I)_0 \leq upperb(I)_0$$

- Pre každý prvok vstupného pola s dvomi kanálmy

$$dst(I) = lowerb(I)_0 \leq src(I)_0 \leq upperb(I)_0 \wedge lowerb(I)_1 \leq src(I)_1 \leq upperb(I)_1$$

- A tak ďaľej...

Vstupné parametre pre *inRange* je možné pozorovať pri tabuľke 5. [15] [16]

3.2.5 bitwise_not

Je jednoduchá funkcia, ktorá invertuje všetky bity v poli ktoré jej pošlete. Taktiež má aj jednoduché vstupné parametre, ktoré vidieť aj v tauľke 6. [8] [16]

Premenná	Dátový typ	Popis
src	Mat	Vstupné zdrjové pole.
lowerb	Scalar	Spodná hranica poľa alebo skalárna veličina.
upperb	Scalar	Vrchná hranica poľa alebo skalárna veličina.
dst	Mat	Výsledné pole, rovnako veľké ako vstup.

Tabuľka 5: Tabulka znázorňuje vstupy funkcie inRange

Premenná	Dátový typ	Popis
src	Array	Vstupné pole plné bitov.
dst	Array	Výstupné pole plné invetovaných bitov

Tabuľka 6: Tabulka znázorňuje vstupy funkcie bitwise_not

3.2.6 threshold

Aplikuje pevnú prahovú úrovní pre každý prvok poľa. Zvyčajne sa používa na získanie binárnej úrovne obrazu v odtieňoch sivej, alebo pre odstránenie šumu. Funkcia *treshold* funguje na princípe filtrovania pixelov ktoré majú príliž veľkú, alebo príliž malú hodnotu. Existuje niekoľko možností ako tento šum odstrániť.

- THRESH_BINARY

$$dst(x,y) = \begin{cases} maxval, & \text{pokiaľ } src(x,y) > tresh \\ 0, & \text{inak} \end{cases}$$

- THRESH_BINARY_INV

$$dst(x,y) = \begin{cases} 0, & \text{pokiaľ } src(x,y) > tresh \\ maxval, & \text{inak} \end{cases}$$

Premenná	Dátový typ	Popis
src	Mat	Vstupný 8-bitový obraz s jedným kanálom.
dst	Mat	Výstupný 8-bitový obraz s jedným kanálom.
thresh	double	Prahová hodnota
maxval	double	Maximálna hodnota ktorú môže použiť na niektoré typy výpočtu.
type	int	Typ výpočtu

Tabuľka 7: Tabulka znázorňuje vstupy funkcie threshold

- THRESH_TRUNC

$$dst(x,y) = \begin{cases} trashold, & \text{pokiaľ } src(x,y) > tresh \\ src(x,y), & \text{inak} \end{cases}$$

- THRESH_TOZERO

$$dst(x,y) = \begin{cases} src(x,y), & \text{pokiaľ } src(x,y) > tresh \\ 0, & \text{inak} \end{cases}$$

- THRESH_TOZERO_INV

$$dst(x,y) = \begin{cases} 0, & \text{pokiaľ } src(x,y) > tresh \\ src(x,y), & \text{inak} \end{cases}$$

Parametre ktoré táto funkcia akceptuje a s ktorými pracuje sú viditeľné v tauľke 7 [8] [16]

Premenná	Dátový typ	Popis
image	Mat	Vstup je 8-bitový obraz ktorý má len jeden kanál, kde všetky hodnoty tohoto kanála ktoré sú väčšie ako 0, sa správajú ako keby mali hodnotu 1.
contours	List:MatOfPoint	Výstup je zoznam kontúr. Každá kontúra je uložená ako vektor bodov.
hierarchy	Mat	Voliteľný výstupný vektor obsahujúci informácie o typológii obrazu. Pre každú kontúru obsahuje množstvo elementov.
mode	int	mód, aleo skôr typ kontúr ktoré budeme chcieť rozpoznávať.
method	int	Metóda aproximácie.

Tabuľka 8: Tabulka znázorňuje vstupy funkcie findContours

3.2.7 findContours

Funkcia *findContours* je prepracovaná metóda hľadania obrysov. Jednoducho nájde obrysy, aleo kontúry v binárnom obraze pomocou algoritmu od Satoshi Suzukiho pre vyhľadávanie čiar v binárnom obraze. Vyhľadané obrysy sú veľmi užitočné pri rozpoznávaní tvarov a objektov. Pri rozpoznávaní dopravných značení je našou snahou taktiež rozpoznať napríklad kruhové tvary zákazových dopravných značení. [13] [16]

obkec o suzuky algoritme - musim si nastudovat jeho pracu [18]

Parametre funkcie vidiet v tabuľke 8

3.2.8 boundingRect

Funkcia *boundingRect* je ďalšia jednoduchá funkcia. Dokáže jednoducho vypočítať a ohraničiť nejaké zoskupenie bodov do odĺžnika. V našom prípade funkciu využijeme na to, aby sme vedeli získať výrez dopravného značenia. Vstup pre funkciu je len samotné

Premenná	Dátový typ	Popis
points	MatOfPoint	Zoskupenie 2D bodov vo vektore.

Tabuľka 9: Tabulka znázorňuje vstupy funkcie boundingRect

zoskupenie bodov, ako vidieť aj na tabuľke 9. [9] [16]

3.2.9 drawContours

Funkcia *drawContours* je vykreslovacia funkcia. Kreslenie kontúr pracuje s maticami. Dokáže vykresliť akýkoľvek tvar, ktorý je definovaný vektorom. [12] [16]

Funkcia je pomerne zložitá na parametre. Podrobnejšie je rozobratá v tabuľke 10

3.2.10 contourArea

3.2.11 fitEllipse

4 Návrh riešenia

Po dôkladnom naštudovaní literatúry a potrebných algoritmov je našim cieľom vyhotoviť riešenie, ktoré by dokázalo detekovať zvislé dopravné značenia. Návrh bude pozostávať z návrhu algoritmov, návrhu objektov a návrhu užívateľského prostredia.

4.1 Návrh algoritmov

Ako metódu rozpoznávania som si zvolil detekciu dopravného značenia podľa tvaru a farby. Algoritmy ktoré som navrhol, sú postavené na princípe rozpoznania farebného rozhrania hľadaného objektu a následné detekovanie potrebného tvaru. Pri opise som sa zameral na detekciu značiek, ktoré sú na cestách najviac početné. Na cestách prevládajú dopravné značenia, ktoré sú červenej a modrej farby. Z tvarov prevládajú kruhy a trojuholníky. Samotné rozpoznanie bolo uskutočnené pomocou neurónových sietí. Táto metóda je najlepšia na počítačové učenie objektov. Na rozdiel od detekcie dopravného značenia, pre riešenie neurónových sietí použijeme už existujúcu knižnicu.

Premenná	Dátový typ	Popis
image	Mat	Obraz do ktorého budú kontúry vkreslené.
contours	List:MatOfPoint	Zoznam všetkých kontúr ktoré chceme vykresliť. Každá kontúra je uložená ako vektor bodov.
contourIdx	int	Index, ktorý určuje ktorú kontúru chceme vykresliť. Negatívne číslo hovorí o tom, že chceme vykresliť všetky kontúry.
color	Scalar	Farba vykreslenej kontúry.
thickness	int	Šírka kontúry.
lineType	int	Typ vykreslenej čiary.
hierarchy	Mat	Voliteľný výstupný vektor obsahujúci informácie o typológii obrazu. Pre každú kontúru obsahuje množstvo elementov.
maxLevel	int	Maximalný level vykreslených kontúr. Tento parameter je funkčný, len v prípade že je použitá hierarchia.
offset	Point	Voliteľný parameter posunov. Posunie všetky kontúry podľa zadaných súradníc.

Tabuľka 10: Tabulka znázorňuje vstupy funkcie drawcontours

4.1.1 Návrh algoritmu pre detekciu farby

Ako prvý algoritmus som si vybral detekciu červenej farby. Pre detekciu farieb sa v literatúre odporúča najprv previesť vstup na farebný model HSV. Vstup prichádza vo farebnom formáte RGB. Farebný model HSV je jeden z dvoch najpoužívanejších valcovo súradnicových reprezentácii bodov pre RGB model. [6]

Na začiatok by sa mal vstup(bitmapa) konvertovať na binárnu maticu.

Najväčšia výhoda dopravného značenia je, že je silne kontrastné od ostatného prostredia. Túto vlastnosť môžeme perfektne využiť v náš prospech a pomocou pomocou rozmazania obrazu, môžeme dosieliť to, že sa zbavíme slabších kontú hneď na začiatku. V OpenCV je pre rozmazávanie obrazu na výber viacero metód, no my použijeme matódu *GaussianBlur*, ktorá už názvom prezrádza použitie známeho filtra *Gaussian blur*

Keďže sa snažíme dostať náš vstupný obraz do formátu HSV, o ktorú sa stará funkcionalita *cvtColor* potrebujeme mu nastaviť vstup tak, aby obraz vedel bez problémov spracovať. Keďže na väčšine mobilných zariadení prichádza do zariadenia obraz vo formáte RGBA, ďalší krok bude nepríklad konvertovanie formátu RGBA na formát RGB.

Ďalej bude nasledovať samotná konverzia obrazu do HSV pomocou už spomínanej metódy *cvtColor*.

Ďalší krok bude spracovať každý kanál farebného modelu HSV samostatne. Ako prvý spracujeme *Hue* kanál, ktorý sa stará o farebný odtien každého pixelu. *Hue* Farba sa v tomto kanáli určuje podľa stupňov. Primárne sa začína na stupni 0°, čo predstavuje zelenú farbu, postupne prechádza do modrej, ktorá sa nachádza na 120° stupňoch z kade prechádza cez červenú na 240° a keďže je to model kruhový, vracia sa do zelenej na 360°. Pomocou funkcie *inRange* by nemal byť problém určiť rozhranie stupňov, ktoré sme schopný akceptovať ako hľadanú farbu pre hľadané naše dopravné značenia. Ďalší kanál je *Saturation*, ktorý predstavuje sýtosť farby. Táto sýtosť sa vyjadruje v percentách, kde 0% predstavuje šedú a 100% je plne sýta farba.[7] V našom prípade je postacuje

metóda *threshold*. Posledný kanál *Value* vyjadruje hodnotu jasu. Keďže v praxi znamená znižovanie jasu pridávanie čiernej do základnej farby, pre hľadanie červenej farby na dopravnom značení nie je potrebné s týmto kanálom pracovať, lebo červená farba použitá na dopravných značeniach je pomerne svetlá. Pri hľadaní modrej je túto farbu potrebné trochu stmaviť a tak použijeme opäť funkciu *threshold*.

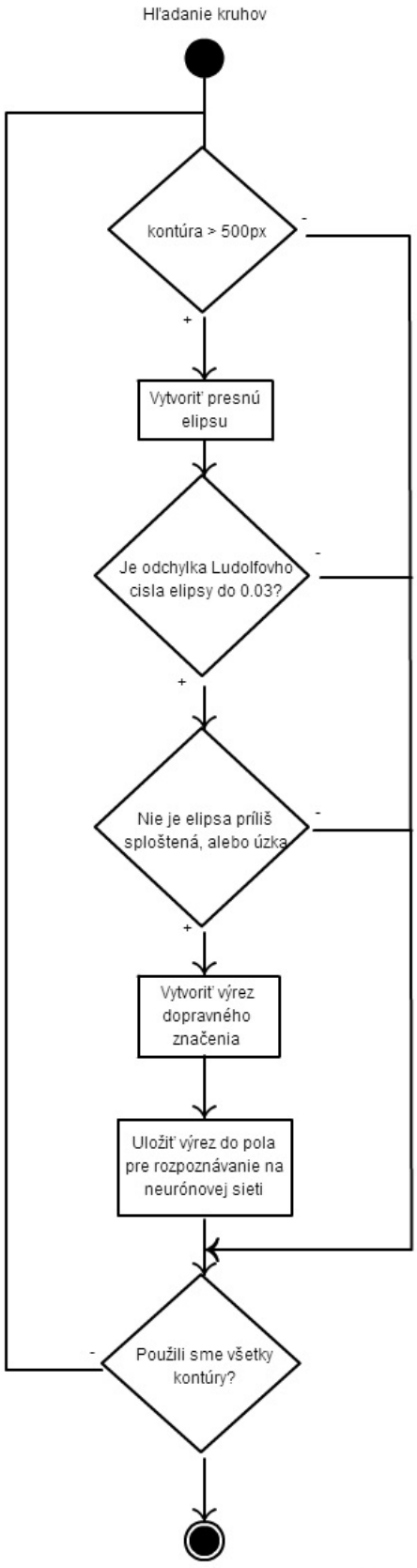
Na koniec potrebujeme dostať len kontúry hľadanej farby. Najpr si budeme musieť spojiť jednotlivé kanály späť do jednej binárnej matice použitím metódy *Canny*. Po tomto kroku by nám mali ostať len čierny obraz a biele škvrny predstavujúce červenú farbu v požadovanom rozsahu. Z týchto bielych objektov, budeme potrebovať len okraje a tak použijeme metódu *findContours*, ktorá sa postará o to, že dostaneme pole kontúr z celého obrazu. S týmito kontúrami potom ďalej pracujeme a rozoznávame z nich hľadané útvary.

4.1.2 Návrh algoritmu pre detekciu kruhov

Pri detekcii dopravného značenia v tvare kruhu, je dôležité počítať s tým, že nehľadáme úplný kruh. Kruhové dopravné značenia sú vyrábané ako dokonalý kruh, no pri ich rozpoznávaní si je potrené uvedomiť, že na objekt sa pozeráme z rôznych uhlov. Táto skutočnosť nám prináša do prolematiky dôležitý fakt, že v skutočnosti to nie sú kruhy čo hľadáme, ale sú to elipsy. Celý algoritmus je možné vidieť na orázku č. 1

Keďže v predchádzajúcej kapitole sme si navrhli riešenie, ktoré nám vracia len kontúry hľadanej farby, môžeme pokračovať od tohto bodu. Ako prvé si spravíme cyklus, ktorým budeme prechádzať všetky naše vyhľadané kontúry farieb. Aby sme eliminovali počet prebytočných kontúr, je potrené spracovávať čo najrelevantnejšie výsledky. Tento úkon vykoná metóda *contourArea*, vďaka ktorej budeme posielať na ďalšie spracovanie len kontúry väčšie ako 500 pixelov.

Vzhľadom na to, že výsledky, ktoré dostávame ešte nemôžeme nazvať elipsami, musíme si naše kontúry na elipsy upraviť. Tento úkon vykonáva metóda *fitEllipse*, ktorá upraví kostrbaté kontúry, ktoré sa aspoň trochu podobajú elipse, na matematicky presnú elipsu.



Obr. 1: Algoritmus vyhľadávania kruhov

Keď už máme detekované elipsy, nastáva posledný krok, a tým krokom je, určiť si toleranciu elipsy dopravného značenia, ktorú vyhľadávam. Táto tolerancia, je vlastne tolerancia nepresnosti, pri výpočte Ludolfovho čísla. Ďalším krokom je tak výpočet už spomínaného ludolfovho čísla a následné overenie jeho nepresnosti. Pokiaľ je výsledná hodnota vyhovujúca, nájdený objekt vyrežeme, a zasielame na rozpoznanie neurónovej sieti, ktorá zistí o akú značku sa presne jedná.

Výpočet Ludolfovho čísla:

$$\pi = \frac{o}{d}$$

Úprava výpočtu Ludolfovho čísla pre elipsu:

$$p = \frac{o}{d} = \frac{o}{(\frac{1}{2}y) * (\frac{1}{2}x)}$$

Získanie tolerancie:

$$\pi - p < 0.03$$

Pre určovanie tolerancie elipsy, je možné použiť ešte jednu metódu, a tou je overovanie podľa osí. Pokiaľ je x-ová os dvoj-násobne väčšia ako y-ová, ide už o elipsu, ktorú by sme ďalej len ťažko identifikovali. Takýto nežiaduci stav môže nastať, pokiaľ sa na značku pozeráme na dopravné značenie z príliž veľkého uhlu.

Dva nežiaduce stavy tvaru dopravného značenia:

$$1.)\ \frac{\frac{1}{2}x}{\frac{1}{2}y} > 2$$

$$2.)\ \frac{\frac{1}{2}y}{\frac{1}{2}x} > 2$$

[1] Song Ho Ahn. Convolution, 2005.

[2] Gábor Blázsovit. Digital image processing - interaktívna učebnica spracovania obrazu, February 2006.

[3] A. Bradski, G. a Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., Gravenstein Highway North, Sebastopol, CA, 2008.

[4] J. Canny. A computational approach to edge detection. 1986.

[5] Wikipedia contributors. Canny edge detector, March 2013.

[6] Wikipedia contributors. Hsl and hsv, February 2013.

[7] Wikipedia contributors. Hsv, March 2013.

[8] OpenCV dev team. bitwise_not, March 2013.

[9] OpenCV dev team. boundingrect, March 2013.

[10] OpenCV dev team. canny, March 2013.

[11] OpenCV dev team. cvtcolor, March 2013.

[12] OpenCV dev team. drawcontours, March 2013.

[13] OpenCV dev team. findcontours, March 2013.

[14] OpenCV dev team. gaussianblur, March 2013.

[15] OpenCV dev team. inrange, March 2013.

[16] OpenCV dev team. *Opencv documentation*, March 2013.

[17] Daniel Rákos. gaussianblur, September 2010.

[18] S. Suzuky. A computational approach to edge detection. 1983.