VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SIMULACE REÁLNÉ TRAJEKTORIE GOLFOVÉHO MÍČKU Z VIDEA

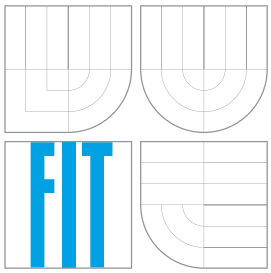
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK HLOBIL

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SIMULACE REÁLNÉ TRAJEKTORIE GOLFOVÉHO MÍČKU Z VIDEA

SIMULATION OF A GOLF BALL TRAJECTORY FROM VIDEO

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK HLOBIL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR CHMELAŘ

BRNO 2011

Abstrakt

Práce se bude zabývat simulací letu golfového míčku na základě videozáznamu. V práci bude popsán způsob, jakým lze na základě prvních několika nalezených bodů trajektorie míčku zjistit její zbytek až po bod dopadu. Pro rozpoznávání míčku bude využito prostředků, které nabízí počítačové vidění. V práci bude také uvedeno, jaké fyzikální jevy na míček během letu působí a způsob, jakým bude dopočítána jeho trajektorie.

Abstract

This work deals with the golf ball flight simulation based on video recordings. The way how to determine the trajectory of a golf ball using several initial points is described here. For the ball recognition there are used computer vision techniques, particulary pattern recognition. The work covers physics of the golf ball, it deals with physical influences that occurs during the ball flight and it tries to describe the trajectory approximation.

Klíčová slova

golf, golfový míček, trajektorie, mechanika, fyzika, video, simulace, rozpoznávání, klasifikace

Keywords

golf, golf ball, trajectory, mechanics, physics, video, simulation, recognition, classification

Citace

Marek Hlobil: Simulace reálné trajektorie golfového míčku z videa, diplomová práce, Brno, FIT VUT v Brně, 2011

Simulace reálné trajektorie golfového míčku z videa

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Chmelaře. Všechny literární prameny a publikace, ze kterých jsem čerpal, jsou uvedeny v seznamu použité literatury.

.....
Marek Hlobil
25. května 2011

Poděkování

Děkuji Ing. Petru Chmelařovi, za jeho odbornou pomoc, cenné rady a připomínky, které mi významnou měrou pomohly při vypracování této práce.

Obsah

Kapitola 1

Úvod

Golf zažívá v poslední době znatelný vzestup. Stále více lidí si k němu nachází cestu, vznikají nová hřiště a díky nárustu počtu hráčů vzniká také mnoho zařízení pro trénování golfu v uzavřených prostorách – tzv. *indoory*.

Nezřídka bývají v indoorech kromě odpaliště k dispozici také golfové simulátory. Většinou se jedná o simulátory, které využívají různých podložek se senzory či radarů k odhadu směru a rychlosti odpalu. Tyto simulátory jsou levné, nicméně ze způsobu, jakým vyhodnocují odpal plyne poměrně vysoká nepřesnost celé simulace. U výše zmíněných technických řešení přicházejí výrobci o možnost sledovat rotaci míčku, jejíž vliv na trajektorii letu je značný.

Lepších výsledků lze dosáhnout za pomoci videozáznamu – simulátory pracující na tomto principu jsou v současné době novinkou a běžně se s nimi v indoorech nesetkáte. Ve své diplomové príci se proto této problematice věnuji a v následujícím textu předkládám návrh systému, který se zabývá simulací odpalu golfového mčku pomocí vysokorychlostních kamer.

1.1 Cíle práce

Cílem této práce je pokusit se o simulaci letu golfového míčku zaznamenaného vysokorychlostními kamerami pomocí algoritmů počítačového vidění.

Zejména se jedná o zjištění trajektorie a nalezení místa dopadu na základě videozáznamu, který zachycuje několik prvních metrů letu po odpalu míčku.

Tato práce by je součástí projektu, který realizuje společnost Inven Solution s.r.o., a který si klade za cíl vytvořit co nejpřesnější golfový simulátor pro použití ve vnitřních prostorách. Demonstrační aplikace, která je výstupem této práce bude integrována do vznikajícího systému a bude také využívat některé již implementované části, například se jedná o modul na kalibraci kamer a získávání reálné pozice míčku ve 3D prostoru.

Tato práce se tudíž nezabývá problematikou stereovize, kalibrace kamer a podobně. Práce je zaměřená hlavně na vyhledání míčku ve videu a na fyzikální vlivy provázející jeho let a to včetně odporu vzduchu, konstantního bočního větru atd.

1.2 Struktura práce

Tato práce zabývá dvěma hlavními tématy. Prvním z nich je počítačové vidění a metody, které se používají pro analýzu obrazu a vyhledávání objektů v něm. Druhým je fyzika, resp. mechanika, jejíchž zákonů využiji při simulaci trajektorie golfového míčku. V práci jsem jako

základ pro teoretickou část využil semestrálního projektu, který jsem dále upravil a rozšířil. Diplomvá práce má následující strukturu.

Kapitola 2 se zabývá problematikou počítačového vidění. Jsou v ní představeny důležité pojmy, které souvisí s počítačovým viděním, zpracováním obrazu a klasifikací objektů v obraze. Jedná se o teoretický úvod do počítačového vidění a rozpoznávání objektů v obraze.

V další kapitole (3) se věnuji fyzikální části práce. Provádím zde analýzu fyzikálních vlastností golfového míče a vlivů, které na něj během letu působí a ovlivňují jeho trajektorii. Je zde navržen způsob výpočtu trajektorie.

Ve 4. kapitole je představen způsob řešení zadaného problému. Uvádím zde konkrétní postupy, které jsem použil. Představuji zde výslednou aplikaci, která byla vytvořena pro demonstraci funkčnosti zvolených metod.

V kapitole 5 shrnuji dosažené výsledky, případně zmiňuji rozšíření, která by bylo možno implementovat.

Kapitola 2

Počítačové vidění

Zrak je nejsilnějším lidským smyslem. Vidění dává lidem možnost vnímat svět kolem nich. Z jednoho obrazu viděného lidským okem dokáže mozek získat obrovské množství informací ve zlomku vteřiny. Počítače takovou možnost nemají. Vzhledem k tomu, že je lidský mozek pořád ještě mnohem vyspělejší než ten nejvýkonnější počítač, musí se počítače spolehnout na to, co umí nejlépe – a sice na zpracování elektronických informací.

Na začátek je nutno říci, že počítačové vidění je obsáhlá vědní disciplína obsahující širokou škálu kategorií. Počítačové vidění zasahuje do mnoha dalších vědních disciplín, např. se jedná o umělou inteligenci, zpracování signálů, matematiku, optiku, pattern recognition & machine learninig a další. V této části se pokusím identifikovat, co to vlastně počítačové vidění je a naznačím některé problémy, které jsou s ním spojené.

2.1 Úvod do počítačového vidění

Počítačovým viděním se rozumí široká škála algoritmů, na základě kterých dokáže počítač činit určitá rozhodnutí o objektech zachycených nejčastěji na obrázku, či videu. Nicméně do počítačového vidění patří i například zpracování radarového signálu. Obraz, který počítač zpracovává je nejčastěji transformován buď na nějaké rozhodnutí (je na obrázku bagr?) nebo do nového objektu (převedení obrázku do grayscale).

Počítačové vidění lze tedy definovat rovněž jako transformaci obrazové informace do reprezentace, které rozumí počítač a dokáže s ní pracovat, modifikovat ji a rozhodovat o/podle ní.

Naučit počítače, aby viděly je nelehký úkol. Lidský mozek dokáže vnímat mnohem více informací o obraze, který zpracovává, kdežto pro počítač je obraz jen matice čísel, ve které se musí zorientovat.

Počítač nemá žádný pojem o hloubce obrazu, vše vníma jako 2D obraz (lze tomu však předejít například pomocí snímání scény více kamerami, čehož budu využívat i v této práci).

V [18] se dočteme, že ztráta informace při převodu obrazu z 3D do 2D je způsobena tím, jak zachycujeme realitu. Kamery a fotoaparáty pracují na principu *camera obscura* (temná místnost). Tento fyzikální model odpovídá matematickému modelu perspektivní projekce, který je charakteristický tím, že nezachovává ani úhly, ani rovnoběžnost. Dalším problémem je nemožnost určit, jak daleko od kamery se objekt nachází, resp. jaká je jeho skutečná velikost.

Předností lidského mozku je, že dokáže lehce analyzovat objekty ve 3D. Dokáže lehce určit polohu objektu v prostoru a odhadnout jeho rozměry a navíc je citlivý na detaily.

Důležitý je také rozhled v prostoru, protože ten umožňuje získat lepší přehled o kontextu scény. Počítače většinou pracují s kamerami, jejichž rozhled ani rozlišení se nedá s tím lidským srovnat. Kamery zaznamenávají pouze 2D průmět 3D obrazu. Převod zpět do třetího rozměru není možný, protože tímto převodem dochází ke ztrátě informací. Tento problém je *formálně neřešitelný* [4] a nezbývá než hledat způsoby, jak ztracené informace co nejlépe rekonstruovat.

Dalším problémem je, že obrázky jsou pouhým výřezem reality a počítače obvykle zpracovávají jen jejich malou část. Je proto těžší získat přehled o globálním kontextu scény.

2.1.1 Reprezentace obrazu

V reálném životě chápeme obraz jako průmět reality na sítnici oka, jako snímek filmu, obrazovku počítače nebo jako fotografii. Ve světě počítačů bývají pro reprezentaci obrazu využívány matematické modely. Obraz je zde vnímán jako signál, který je reprezentován určitou funkcí závisející na nějaké n-rozměrné proměnné:

$$z = f(x,y). \tag{2.1}$$

Jelikož je obraz, který prezentujeme omezený, můžeme dle [27] předpokládat, že definiční obor funkce f je také omezený:

$$f : (\langle x_{min}, x_{max}\rangle \times \langle y_{min}, y_{max}\rangle) \rightarrow H. \tag{2.2}$$

Definičním oborem funkce jsou reálná čísla x, y taková, že: $x : x_{min}$, x_{max} a $y : y_{min}$, y_{max} jsou souřadnice bodu v obraze, v nichž nabývá funkce f nějaké hodnoty z náležící oboru hodnot H.

V počítačové grafice se nejčastěji setkáváme s n-ticí reprezentující některý barevný model (RGB, CMYK, HSV, ...). Nicméně obecně může obor hodnot nabývat libovolného tvaru, důležité je jak jej potom prezentujeme.

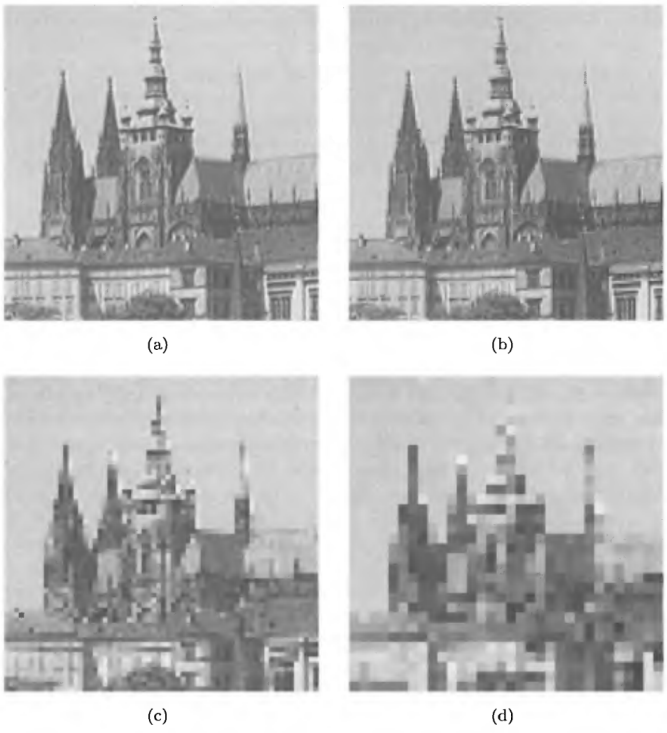
Digitalizace obrazu

V této části jsem vycházel z [18] a [27]. Obraz, který je zachycen kamerou je reprezentován spojitou funkcí *f(x,y)*, kde x a y jsou souřadnice bodu v rovině. Takovýto obraz je potřeba převést do diskrétní podoby, kterou je schopen zpracovat počítač. Pro tento účel je potřeba zvolit datovou strukturu, která bude uchovávat informace o obrazu ve vhodné podobě, v tomto případě je nejvhodnější matice. Nicméně existuje celá škála datových struktur, vhodných pro reprezentaci a analýzu obrazu. Některé z nich jsem popsal v samostatné podkapitole o datových strukturách 2.1.2.

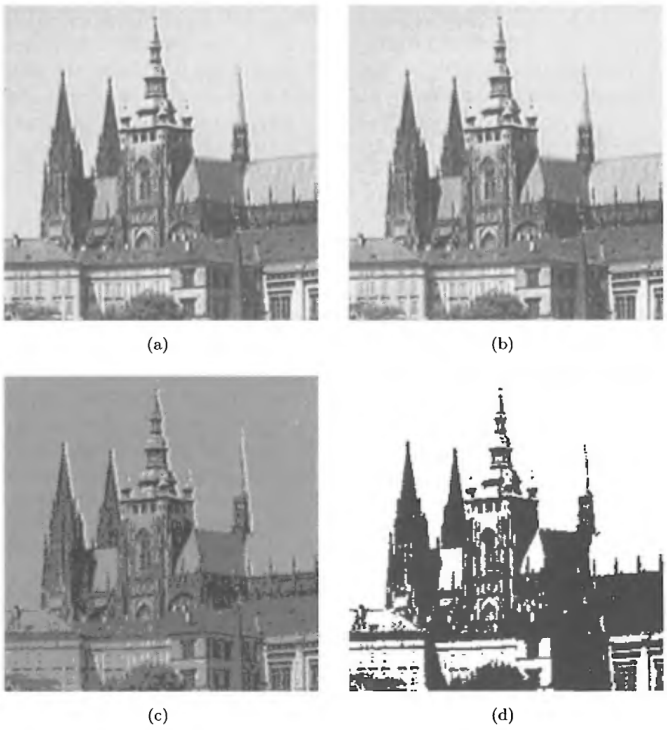
Digitalizací obrazu se rozumí navzorkování spojité funkce *f(x,y)* do matice o M řádcích a N sloupcích. Kvantizací (shlukováním) obrazu přiřadíme každému spojitému vzorku celočíselnou hodnotu – spojitý rozsah obrazové funkce *f(x,y)* je rozdělen do K intervalů. Čím více vzorků a čím více intervalů si můžeme dovolit, tím lepší aproximace funkce *f(x,y)* dosáhneme. Méně používané je takzvané neuniformní vzorkování, kde je délka intervalu K proměnná.

- Vzorkování – jak je naznačeno výše, vzorkování (anglicky sampling) slouží k redukci spojitého signálu na diskrétní signál. Jemností vzorkování řídíme úroveň detailů, které bude výsledný objekt obsahovat.

  Obraz je obvykle vzorkován do čtvercové či hexagonové sítě.



Obrázek 2.1: Příklad vzorkování. (a) 256 × 256 , (b) 128 × 128, (c) 64 × 64, (d) 32 × 32. Zdroj: [18].



Obrázek 2.2: Příklad kvantizace. Za K zvoleno: (a) 64, (b) 16, (c) 4, (d) 2. Zdroj: [18].

- Kvantizace (shlukování) – kvantizací se rozumí převod spojitých hodnot obrazové funkce (jasu) na jejich digitální reprezentaci. Jinak řečeno jedná se o přemapování skupiny hodnot s podobnými rozsahy, na hodnoty se stejným rozsahem. Skupiny (kvanta) hodnot o velikosti K budeme reprezentovat jedinou charakteristickou hodnotou. Kvantizace je logicky navazující funkcí na vzorkování.

2.1.2 Datové struktury pro analýzu obrazu

Vzhledem k tomu, že zpracováváme obraz pomocí počítače, je velmi důležité zvolit pro řešený problém vhodnou reprezentaci obrazových dat. Snažíme se obrázek převést na model, se kterým se nám bude při řešení problému lépe pracovat. Vhodně zvolená reprezentace dat může ušetřit hodně práce. Přechod od obrázku k jeho modelu lze rozdělit do několika úrovní. Dle [18] jsou to tyto:

- **Ikonický obraz** je na nejnižší úrovni a reprezentuje původní matici pixelů. Bývá také výstupem přípravných fází zpracování obrazu.
- Na druhé úrovni je **segmentovaný obraz**. Segmentace je proces sdružování částí obrazu do skupin, které se lépe analyzují. Příkladem segmentace je vyhledávání hran, oblastí se stejnou barvou apod.
- Další úrovní je udržování informací o **geometrických útvarech** nalezených v obraze. Získání geometrických tvarů z obrazu je velmi důležité pro budouci zpracování, zejména pro detekci pohybujicích se objektů apod.
- Poslední úrovní je **relační model**, který nám dává moc nakládat s daty efektivněji a na vyšší úrovni abstrakce. Obvykle je využito toho, že víme, jaký problém je třeba řešit.

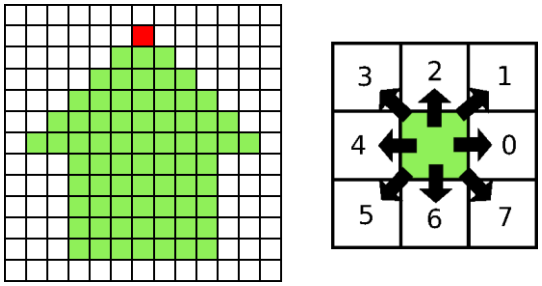
Tradiční datové struktury

- **Matice** jsou nejčastejšími strukturami pro nízkoúrovňovou reprezentaci obrazu. Jak je uvedeno výše, matice je obvykle prostředkem pro uchovávání surových dat získaných z kamery, fotoaparátu či jiného zařízení.

  Výhodou matic je to, že se s nimi setkáme téměř ve všech programovacích jazycích a navíc je jim nakloněna i struktura fyzické paměti počítače – tudíž se s nimi dá relativně lehce pracovat.
- **Řetězy (chains)** jsou datovou strukturou, která neuchovává všechny pixely obrazu, ale jen hraniční body objektů. K reprezentaci hranic se používají tzv. řetězové kódy (chain codes), kterých existuje několik typů, více viz [25].

  Hranice je definována sekvencí relativních souřadnic, které korespondují s umístěním sousedního pixelu. Tyto souřadnice mohou být získány například pomocí matice uvedené na obrázku 2.3.
- **Topologické struktury** se vyznačují tím, že popisují obraz jako množinu elementů a vztahů mezi nimi. Typickým zástupcem je graf přilehlých objektů (region adjacency graph). Využívá se například pro reprezentaci segmentovaných obrázků. Každý segment je uzlem a sousední uzly jsou propojeny.

  Výhodou grafů je, že se dají jednoduše dělit na menší celky, dají se také použít při pattern matchingu pro porovnávání s hledaným vzorkem.

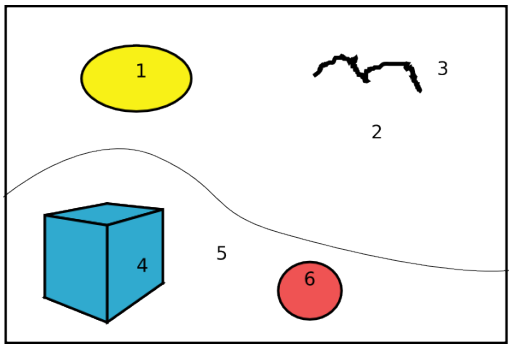


Obrázek 2.3: Příklad objektu a matice pro řetězovou reprezentaci. Řetězový kód pro tento obrázek je: 77777446666644444422222441111 1 – začínáme od červeného čtverečku.

Id	Jméno	Barva	Min. řádek	Min. sloupec	Uvnitř
1	Slunce	žlutá	15	25	2
2	Obloha	modrá	0	0	
3	Vrána	černá	16	16	2
4	Krychle	modrá	60	13	5
5	Louka	zelená	45	0	
6	Míč	červená	75	75	5

Tabulka 2.1: Tabulka s popisem relací na obrázku 2.4.

- **Relační struktury** využívají segmentace obrazu pro uložení důležitých částí obrazu do objektů. Relace mezi objekty a jejich vlastnosti jsou zaznamenány do tabulky (viz tabulka 2.1). Příklad relační datové struktury je uveden na obrázku 2.4.



Obrázek 2.4: Příklad relační datové struktury. Zdroj [18].

Hierarchické datové struktury

Tyto struktury se používají pro urychlení zpracování obrazu. Díky hierarchickému uspořádání dovolují volbu rozlišení, s jakým budou různé části obrazu zpracovány, dovolují vynechat nezajímavé části obrazu a zaměřit se pouze na ty důležité.

- **Pyramidy** jsou nejjednodušší hierarchickou datovou strukturou, ze které jsou odvozovány další, složitější struktury. Máme dva typy pyramid - maticové a stromové.

  Maticová pyramida je množina obrázků $\{M_L, M_{L-1}, \ldots, M_0\}$, kde obrázek M_L je identický s původním obrázkem a M_{L-1} má o polovinu nižší rozlišení než M_L. Tento

typ pyramidy je užitečný, pokud potřebujeme pracovat s obrázkem v různém rozlišení. Často se nám však stane, že stačí pouze určitá část obrázku a zbytek nás nezajímá. Pro tento účel jsou určeny stromové pyramidy.

- **Quadtree** je stromová struktura, kde každý uzel (kromě listů) má čtyři potomky. Na každé úrovni stromu je obrázek rozdělěn na čtyři kvadranty. Není nutné, aby byly ve stromu uloženy všechny úrovně reprezentace pro každou stromovou úroveň.
- **Ostatní** hierarchické struktury jsou odvozeny od základní pyramidy – například úpravou poměru, v jakém se bude snižovat rozlišení mezi snímky.

2.2 Předzpracování obrazu

Obraz, který zpracováváme ve většině případů obsahuje mnoho informací, z nichž je pro nás důležitá jen určitá část. Proto je vhodné nějakým způsobem v obraze zdůraznit ty informace, které potřebujeme a ty, které jsou nepotřebné utlumit. Obraz může být také například příliš tmavý, mít nízký kontrast a podobně. Můžeme požadovat jen lokální či globální změny a tak dále.

Pro tento účel existují techniky předzpracování obrazu, které ikonocký obraz připraví na další zpracování. Musíme počítat s tím, že při předzpracování dochází většinou ke snížení objemu informací, které obraz obsahuje.

2.2.1 Odstranění šumu

Šum v obrázku je zdrojem chyb při rozpoznávání objektů proto je nutné ho nějakým způsobem odstranit. Při odstraňování šumu je třeba zachovat hrany v obrázku. Proto se používají techniky, které počítají nové hodnoty pouze z těch bodů, které mají podobné vlastnosti a běžně je odstraňování založeno na průměrování jasu hodnot v nějakém okolí $\mathcal{O}$.

Gaussův filtr

Tento filtr pracuje na principu snižování váhy pixelů, které jsou ve větší vzdálenosti od právě zpracovávaného pixelu. Pravděpodobně se jedná o jeden z nejpoužívanějších filtrů na odstraňování šumu vůbec [17].

Gaussův filtr pracuje na principu konvoluce obrazu s maskou, která je vytvořena pomocí Gaussovy funkce 2.3.

$$g(x,y) = \frac{1}{\sqrt{2\pi}\ \sigma} e^{-\frac{d^2}{2\sigma^2}}, \tag{2.3}$$

kde $d = \sqrt{(x-x_c)^2 + (y-y_c)^2}$ je vzdálenost sousedního pixelu $[x, y]$ od zpracovávaného pixelu $[x_c, y_c]$.

Tímto způsobem dosáhneme odstranění šumu, nicméně obraz bude rozostřený, což bude komplikovat detekci hran a tím pádem i samotných objektů.

Mediánový filtr

Mediánový filtr pracuje tak, že pro každý pixel obrazu vezme jeho okolí a vybere z něj pixel jehož hodnota odpovídá mediánu[1]. Původní pixel je nahrazen právě touto hodnotou.

[1] Medián je hodnota, která odpovídá prostřednímu prvku v seřazené posloupnosti.

Tímto způsobem dojde k eliminaci pixelů, jejichž hodnoty se výrazně liší od hodnot v jeho okolí.

Nevýhodou filtru je, že degraduje tenké linky a ostré hrany. Proto, pokud potřebujeme zachovat tyto hrany, změníme tvar okolí bodu, které slouží pro výpočet mediánu – například na takové, které nepočítá s diagonálně sousedícími prvky.

Laplacian of Gaussian

Pro zachování hran při odstraňování šumu lze použít pro doostření hran Laplaceův filtr, který aplikujeme na obrázek zpracovaný Gaussovým filtrem.

Laplaceův operátor ∇^2 (tzv. *Laplacián*) vychází z parciálních derivací druhého řádu a jeho aplikace na obrazovou funkci je vyjádřena vztahem:

$$\nabla^2 g(x,y) = \frac{\delta g(x,y)}{\delta x^2} + \frac{\delta g(x,y)}{\delta y^2} \tag{2.4}$$

Laplacián je invariantní vůči natočení obrazu. Aplikujeme-li jej na obrázek zpracovaný Gaussovým filtrem pomocí konvoluce dojde ke zvýraznění hran ([18]). Pro pojmenování kombinace těchto funkcí se používá zkrata LoG (*Laplacian of Gaussian*). Matematické vyádření *LoG* je

$$[\nabla^2 G(x,y,\sigma)] \star f(x,y), \tag{2.5}$$

kde σ je standartní odchylka používaná Gaussovou funkcí a $\star$ značí operaci konvoluce.

2.2.2 Segmentace obrazu

Segmentace obrazu je v počítačovém vidění velmi důležitá. Jejím účelem je rozdělit obrázek na části, které se podobají hledaným objektům.

Pokud se nám podaří rozdělit obraz tak, že vzniklé části reprezentují opravdu hledané objekty (tzn. objekty se např. nedotýkají) hovoříme o kompletní segmentaci, jinak hovoříme o částečné segmentaci.

Pro segmentaci je příznivé, pokud hledané objekty mají jednotné pozadí – což je i náš případ, protože pozadí za míčkem bude tmavé barvy.

Při segmentaci musíme řešit problémy spojené například s nejednotností dat či se zašuměním obrazu.

Dále se věnuji metodám segmentace.

Prahování (thresholding)

Prahování je nejjednodušší, nejpoužívanější a zároveň rychlý způsob segmentace. Vstupem je obraz převedený do stupňů šedi a využívá se toho, že objekty mívají stejnou barvu – resp. jas na celém svém povrchu.

Segmentace obrázku R odpovídá množině regionů (segmentů) $R_1, \ldots, R_S$,

$$R = \bigcup_{i=1}^{S} R_i, \quad R_i \cap R_j = \emptyset, \quad i \neq j. \tag{2.6}$$

Algoritmus projde celý obrázek, a porovná všechny pixely $f(i,j)$ s prahem T.

Odpovídajícímu pixelu $g(i,j)$ ve výstupním obrázku je přiřazena hodnota reprezentující objekty pokud $f(i,j) \geq T$, jinak je mu přiřazena hodnota reprezentující pozadí:

$$f(x) = \begin{cases} 1 & pro\ f(i,j) \geq T \\ 0 & pro\ f(i,j) < T \end{cases}. \tag{2.7}$$

Pro správné fungování prahování je kritické správně určit T. Běžně se setkáváme s nerovnoměrným osvětlěním v obrázku, proto je nutné volit pro různé regiony různé T. Jde o tzv. adaptivní prahování.

Lokální prahy jsou v tomto případě závislé na pozici v obrázku

$$T = T(f, f_c), \tag{2.8}$$

kde f_c je konkrétní část obrázku, pro kterou se práh určuje.

Prahování má mnoho variací, můžeme například pomocí přidání odstínu šedi rozlišovat o jaký objekt jde, určit jeho hranici nebo zanechat objektům jejich barvy a změnit jen barvu pozadí.

2.3 Strojové učení a rozpoznávání vzorů

Rozpoznávání vzorů (*pattern recognition*) je vědecká disciplína, jejímž cílem je rozdělování vstupních objektů do tříd na základě jejich podobných vlastností. Tato metoda se hojně používá v počítačovém vidění pro rozpoznávání objektů či vzorů (patterns) v obraze.

Strojové učení (*machine learning*) patří do odvětví umělé inteligence. Jedná se o vědeckou disciplínu, která se zabývá vývojem algoritmů, díky kterým se stroje dokáží samy zdokonalovat ve svých výsledcích na základě zkušeností, které získají zpracováváním vstupních dat.

Rozlišujeme učení *s učitelem* – výsledek výpočtu je porovnán se trénovacími daty a na základě rozdílu těchto hodnot jsou upraveny vnitřní proměnné algoritmu. Stroj je později schopen rozeznat i ty vlastnosti, které patří do stejné třídy vzorů, ale nebyly trénovány.

Učení *bez učitele* – stroj nemá k dispozici trénovací data a musí se na základě podobností sám naučit jaké vlastnosti sdružovat (clustering) a jak s nimi nakládat.

2.3.1 Statistické rozpoznávání

Tento druh rozpoznávání je založen na třídění objektů do tříd, nikoliv podle toho, o jaký objekt se jedná, ale podle toho, jaké má vlastnosti (*features* 2.4). Objekty se získávají segmentací obrazu.

Statistický klasifikátor je v [18] definován jako klasifikátor který má n vstupů a jeden výstup. Každý ze vstupů slouží k zadávání informací o jedné z n vlastností pozorovaného objektu.

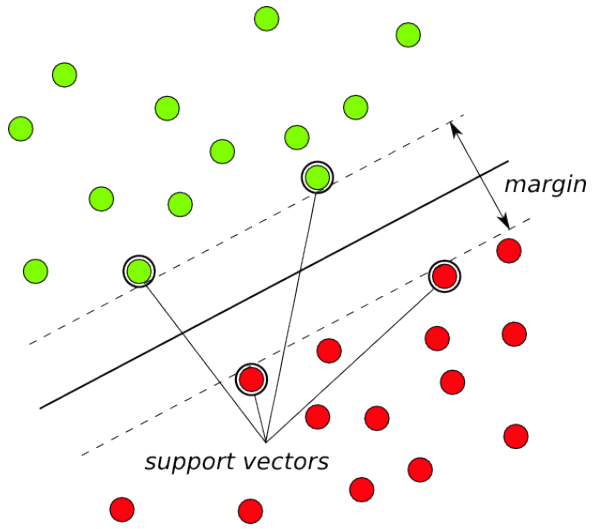
Klasifikátor je trénován pomocí sady trénovacích objektů. Každý z objektů je reprezentován vektorem příznaků a obsahuje zároveň informaci o tom, do které třídy patří. Schopností klasifikátorů je přizpůsobovat se novým trénovacím datům, a díky tomu jsou pak schopny rozpoznat i objekty, které nebyly přímo součástí trénovací sady.

Velikost trénovací sady se určuje obtížně, většinou je třeba klasifikátor spustit, otestovat a vyhodnotit, zda je dostatečně přesný. Záleží také na tom, jak přesná a hodnotná trénovací data jsou k dispozici. Přesnost klasifikátoru závisí na kvalitě trénovacích dat a čase, po který je klasifikátor trénován.

Support Vector Machine

Tento způsob klasifikace je založen na myšlence optimálního rozdělení dvou tříd objektů pomocí maximalizace šířky volné plochy (*margin*) mezi nimi. Pro vymezení této plochy se používají pomocné přímky (*support vectors*).

Tyto přímky jsou vedeny prvky tříd, které jsou nejblíže sousedící třídě a vymezují tzv. diskriminační funkci. Hledáme takové dvě přímky, jejichž vzdálenost je co největší abychom maximalizovali plochu dělící roviny. Pro lepší znázornění viz obrázek 2.5.



Obrázek 2.5: Nalezení maximální dělící plochy.

Kvůli rozložení vstupních dat nemusí být vždy možné jejich rozdělění do dvou tříd pomocí přímky. Místo toho, abychom data rozdělovali nelineární přímkou, namapujeme je raději do vyšší dimenze pomocí jádra (*kernel*) a rozdělíme nadrovinou. Tímto způsobem získáme data, která lze opět lineárně rozdělit do dvou tříd.

2.3.2 Neuronové sítě

Neuronová síť se snaží napodobovat neuronovou síť mozku a je modelem používaným hojně v oblasti umělé inteligence, při rozpoznávání a kompresi videa a zvuku.

Hlavními prvky sítě jsou *neurony*, které jsou propojeny a komunikují mezi sebou. Každý neuron může mít libovolný počet vstupů, ale jen jeden výstup. Jako neuron můžeme chápat i celou síť.

Existují různé způsoby reprezentace neuronu, avšak jedním z nejpoužívanějších je model na obrázku 2.6 jehož autory jsou Warren McCulloch a Walter Pitts.

S každým vstupem neuronu v_i je spojena jeho váha w_i, která určuje míru důležitosti daného vstupu. Výstupem je potom suma vstupů popsaná rovnicí 2.9.

$$x = \sum_{i=1}^{n} v_i \, w_i - \theta, \tag{2.9}$$

kde θ je práh, při kterém se neutron aktivuje.

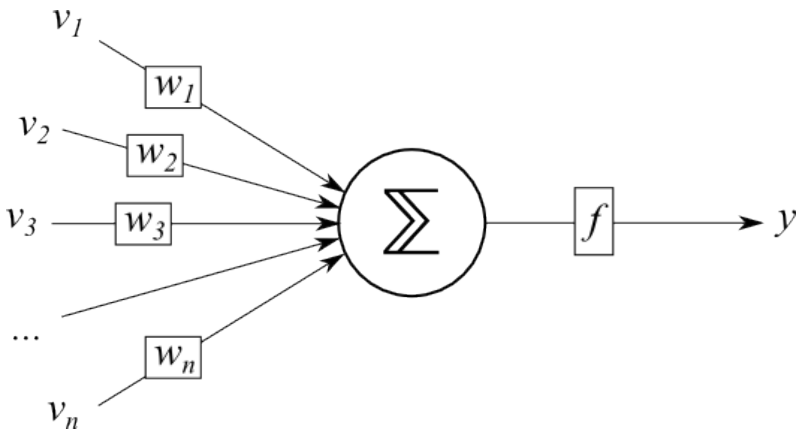
S neuronem je ještě svázána přenosová funkce $f(x)$, která produkuje výstup. Používá se například funkce

$$f(x) \ = \begin{cases} 0 & if\ x \leq 0 \\ 1 & if\ x > 0 \end{cases}. \tag{2.10}$$

Definici neuronu lze popsat rovnicí

$$y \ = \ f\Big(\sum_{i=1}^{n} \ v_i \ w_i - \theta\Big). \tag{2.11}$$

Důležitou vlastností neuronových sítí je jakési zapouzdření implementace sítě. K síti můžeme přistupovat jako k neuronu – dáme jí vstup a očekáváme nějaký výstup. To, co je uvnitř nás příliš nezajímá – může to být složitá síť s mnoha tisíci neuronů nebo mohou být vstupy napojeny přímo na výstupy.



Obrázek 2.6: McCulloch-Pittsův neuron.

Při klasifikaci se používá jako výstup m-rozměrný binární vektor, který obsahuje pouze jednu jedničku. Pozice ve vektoru symbolizuje jednu z m tříd, do které spadá klasifikovaný objekt.

Mezi další využití patří například oblast zpracování řeči. Na vstup zadáváme text, který je sítí transformován na fonémy a tudíž převáděn na řeč.

Feed-forward

Zpočátku se používaly sítě, které měly pouze dvě úrovně – do první (vstupní) vstupovala data a byla předána vrstvě druhé (výstupní), která je vrátila jako výstup. Neexistovala žádná jiná logika, než ta, která propojovala tyto dvě úrovně. To bylo však dost omezující, takže se začaly používat sítě s více úrovněmi.

Informace v těchto sítích proudí pouze jedním směrem, nejsou zde žádné cykly tak, jak je to možné u Hopfieldových sítí.

Hopfieldovy neuronové sítě

Tento typ sítí je specifický tím, že se chováním podobá asociativní paměti. Na počátku se spočítají váhy propojení mezi neurony a při rozpoznávání síť pracuje tak, že se snaží nalézt takové propojení neuronů, které je nejbližší vstupnímu vektoru.

Dalším specifikem je možnost propojení dvou neuronů mezi sebou, čímž mohou vznikat v síti cykly.

2.3.3 Syntaktické rozpoznávání

Syntaktické rozpoznávání se používá, pokud lze hledaný objekt reprezentovat jako množinu primitiv. Používají se relační datové struktury a grafy. Složitější objekty bývají složeny z objektů jednodušších a pro určení, zda jsou objekty podobné se používají gramatiky.

Primitiva by měla být jednoduše segmentovatelná, nemělo by jich být velké množství a měla by se dát vyhledat pomocí nějaké statistické rozpoznávací metody, o kterých jsem se již zmiňoval v 2.3.1. Dále je důležité, aby primitiva reprezentovala charakteristické části objektu a aby se z nich dal výsledný obraz sestavit.

Gramatiky a jazyky

Jestliže budeme množinu primitiv považovat za abecedu, můžeme k popisu slov, která lze z této množiny vytvořit použít gramatiku. Slova v tomto případě reprezentují hledané objekty, které lze z primitiv (písmen abecedy) sestavit.

Gramatika je čtveřice: $G = [N, \Sigma, P, S]$, kde N je konečná množina non-terminálních symbolů, Σ je množina terminálních symbolů (abeceda), v našem případě je to množina primitiv. Platí $N \cap \Sigma = \emptyset$. P je neprázdná konečná podmnožina kartézského součinu množin $N^* \times \Sigma^*$ a je známa jako množina přepisovacích pravidel. Symbol $S \in N$ je počáteční (startovací symbol). Teorie jazyků a gramatik nespadá do rámce této práce, pro více informací doporučuji [12].

Syntaktická analýza - klasifikátor

Můžeme-li sestavit pro každou třídu objektů gramatiku, potřebujeme vytvořit klasifikátor, který rozhodne, do které třídy daný vzor patří. Nejintuivnější cestou je projít postupně všechny gramatiky a testovat, zda gramatika dokáže přijmout vstupní vzor (slovo).

Samotné rozhodnutí, zda gramatika akceptuje slovo je provedeno během syntaktické analýzy. Syntaktická analýza je proces, při kterém se snažíme sestavit derivační strom pro hledané slovo. Pokud strom nalezneme, znamená to, že je slovo gramatikou přijato, v opačném případě je slovo odmítnuto.

Existují dva přístupy k syntaktické analýze.

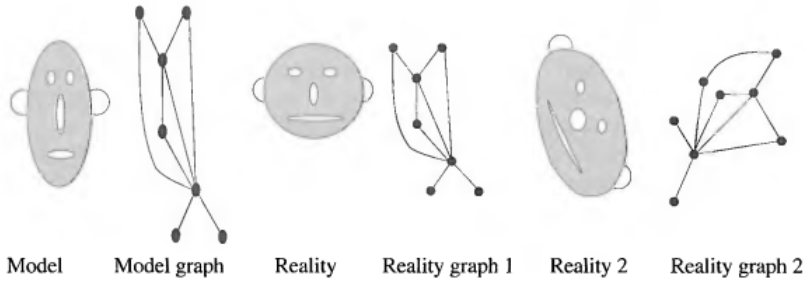
- Shora dolů - začínáme od startovacího symbolu S a postupným aplikováním pravidel $p \in P$ se snažíme doderivovat ke slovu, které je stejné jako slovo hledané.
- Zdola nahoru - aplikováním reverzovaných pravidel se snažíme od hledaného slova doderivovat ke startovacímu symbolu.

Pro další informace doporučuji [12].

2.3.4 Graph matching

Rozpoznávání objektů pomocí grafů je založeno na vyhledáváni izomorfismů mezi grafy či podgrafy. Izomorfismus je z matematického hlediska zobrazení mezi dvěma matematickými strukturami (v našem případě grafy), které každému prvku z první struktury přiřadí právě jeden prvek z druhé struktury a zároveň zachovává vztahy k ostatním prvkům.

Hledání izomorfismů v grafu je klasický problém teorie grafů a existují práce, které ho řeší.



Obrázek 2.7: Graph Matching. Zdroj: [18].

V praxi se však hledá izomorfizmus podgrafů či podgrafů podgrafů (dvojitý izomorfismus), neboť díky nedokonalostem obrazu – šumu, překrývajícím se objektům, osvětlení apod. – nebude graf úplně dokonalý.

Tento problém je však NP-úplný a proto se používají různé heuristiky pro snížení časové náročnosti. Používá se například zjednodušování grafů či rozdělování množiny uzlů podle různých kritérií, například počtu sousedních uzlů, atributů uzlu, počtu zpětných hran apod. Rozdělováním množiny uzlů lze dobře určit, že grafy nejsou izomorfní.

Problém dvojitého izomorfismu podgrafů lze řešit nalezením největší kliky grafu.

V reálných aplikacích nám nestačí hledat izomorfní grafy/podgrafy. Musíme hledat grafy podobné, protože nedokonalost vstupních dat může být vysoká. Používána je metoda vzorů a pružin. Vzory (podgrafy) jsou propojeny pomocí pružin (vazeb mezi podgrafy). Podobnost grafů odpovídá podobnosti podgrafů a napětí, které vzniká na pružinách mezi podgrafy. Napětí mezi podgrafy může být dle potřeby korigováno.

2.4 Klasifikátor

Klasifikátor je program, který slouží k rozdělování objektů do tříd.

Pro rozeznávání objektů v obraze je třeba sestavit vhodný klasifikátor. To, do které třídy objekt patří, určuje klasifikátor na základě specifických rysů objektu (patterns). Prostředky pro trénování klasifikátorů nabízí například knihovna OpenCV [4].

Jak vytvořit klasifikátor je popsáno v části 4.3. Dále vycházím z poznatků uvedených v [18], [17] a [27].

2.4.1 Reprezentace znalostí

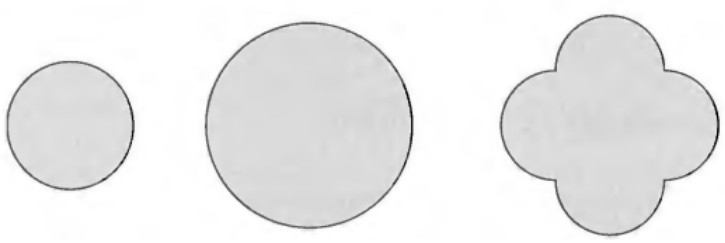
Pro vytvoření dobrého klasifikátoru je důležité, jak jsou reprezentovány jeho poznatky o objektech, které klasifikuje. Proto je třeba zvolit správnou datovou strukturu, která se může v závislosti na klasifikovaných objektech lišit.

Dle [18] stačí malý počet relativně jednoduchých strategií klasifikace k tomu, aby systémy umělé inteligence získaly dobrý přehled o složitém chování, za předpokladu, že mají k dispozici základnu znalostí odpovídající složitosti. Jinými slovy je potřeba mít k dispozici dobře strukturalizovanou reprezentaci velkého objemu dat.

Příznaky

neboli anglicky *features*, jsou nejběžnější formou reprezentace znalostí. Využívají se z praktických důvodů k popisu charakteristických vlastností jednotlivých objektů, nicméně nejsou

považovány za plnohodnotné znalosti a používají se hlavně k doplnění znalostí vyšší úrovně. Jsou však hojně využívány. Například byly využity při tvorbě klasifikátoru na rozpoznávání obličeje v práci [19].



Obrázek 2.8: Příklady objektů. Zdroj: [18].

Příznaky se běžne akumulují do skupin, tzv. vektorů (feature vectors). Například pro objekty z obrázku 2.8 bychom mohli zvolit vektor příznaků jako $\mathbf{x} = (velikost, kulatost)$. Potom bychom jej mohli použít pro klasifikaci objektů do tříd: malé, velké, kulaté, velké nekulaté a podobně (za předpokladu, že dokážeme určit co je malé/velké, kulaté/nekulaté).

Gramatiky a jazyky

jsou používány v případech, kdy není vhodné používat příznaky. Jde o situace, kdy nás zajímá fyzická struktura objektu. Nejobvylkejší reprezentací jsou řetězy, stromy a obecné grafy. Stromem či řetězem můžeme popsat jeden objekt, ale ne celou třídu všech objektů [18].

Predikátová logika

přináší formální nástroj k odvozování znalostí pomocí dedukce. Hlavní síla tkví v tom, že lze aplikovat různé matematické postupy a důkazy. Je to však zároveň i slabinou, protože nejde pracovat s neúplnými či nejistými informacemi – predikátová logika ze své podstaty vyžaduje jasné vymezení toho, co je a není pravda. Vhodnějším přístupem je použití fuzzy logiky (viz dále).

Derivační pravidla

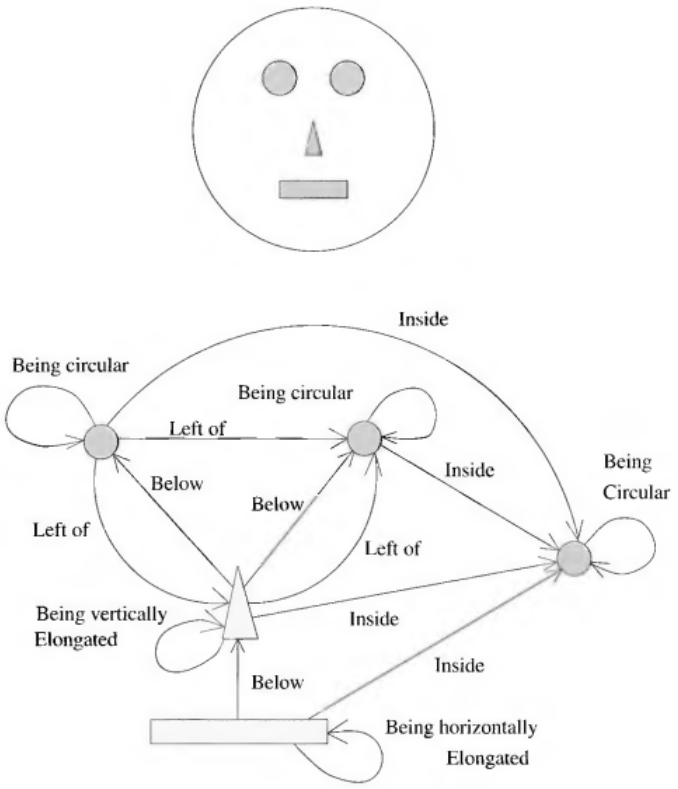
jsou založena na přístupu *podmínka* $\rightarrow$ *akce* a mohou se vyskytovat například v následující podobě:

$$if \text{ podmínka\_splněna } then \text{ proveď\_akci} \tag{2.12}$$

Znalost je v tomto případě reprezentována informací o tom, které pravidlo se má kdy použít.

Fuzzy logika

se od klasické logiky liší tím, že neobsahuje pouze dvě logické hodnoty (*true*, *false*), ale rovnou nekonečně mnoho hodnot, které spadají do intervalu $< 0; 1 >$. To nám umožňuje vytvořit pravidla, která lépe reflektují objekty skutečného světa (některé míče jsou kulaté, jiné šišaté, ale pořád to jsou míče). Tvar fuzzy pravidel je následující:



Obrázek 2.9: Sémantická síť. Zdroj: [18]

$$if \text{ X má\_vlastnost A } then \text{ Y má\_vlastnost B} \tag{2.13}$$

Takovýto způsob reprezentace znalostí je mnohem přirozenější, dovoluje specifikovat pravidla odpovídající popisu skutečných objektů tak, jak to dělají lidé.

Sémantické sítě

jsou složeny z objektů, jejich popisu a popisu vztahů mezi nimi. Od běžných sítí se liší použitím sémantiky, v síti může být použita predikátová logika.

Síť má hierarchické uspořádání – složitější objekty jsou složeny z jednodušších a dají se na ně opět rozložit. Uzly obyčejně reprezentují objekty a hrany vztahy mezi nimi. Příklad sémantické sítě je uveden na obrázku 2.9.

Rámce

(frames, scripts) jsou vysokoúrovňovou reprezentací znalostí. Mohou obsahovat všechny výše zmíněné reprezentace znalostí.

Rámce jsou tvořeny seznamem akcí, které na sebe mají logickou návaznost. Díky těmto seznamům mohou systémy počítačového vidění lépe odhadnout, jaká akce bude pravděpodobně následovat, a reagovat na ni odpovídajícím způsobem.

2.4.2 Boosting

Zřídkakdy se stane, že stačí jeden jediný klasifikátor, který vyřeší daný problém, ať už úplně nebo dostatečně. Z toho důvodu se používá technika zesilující účinek klasifikátorů (tzv. boosting). Při boostingu se kombinují slabší klasifikátory do jednoho silnějšího. U slabých klasifikátorů stačí, aby jejich úspěšnost byla vyšší než 50% tzn., aby problém náležitosti prvku do dvou množin zvládl vyřešit alespoň o trochu lépe než na 50%. V této části budu vycházet z [18] a [8].

Slabé klasifikátory jsou volány v cyklech a jejich výsledek je zkombinován do jednoho klasifikačního pravidla, které je tak účinnější než kterékoliv ze slabých pravidel. Běžně se postupuje tak, že se prvky trénovací množiny předkládají klasifikátorům, které se je snaží vyhodnotit, v dalším cyklu se špatně vyhodnocené prvky předloží jiným klasifikátorům a tak dále, dokud není splněna nějaká mez úspěšnosti.

Nejpoužívanějším boostovacím algoritmem je **AdaBoost** z roku 1997 [6].

Přednost tohoto algoritmu tkví v tom, že dokáže exponenciálně snižovat chybu výsledného klasifikátoru, a to na libovolně nízkou úroveň [8]. Výsledné klasifikátory mívají úspěšnost blížící se ke 100% a čas potřebný k jejich natrénování je relativně nízký.

Algoritmus funguje tak, že máme množinu X, která obsahuje m trénovacích prvků $\mathbf{x}_i$ a k nim odpovídající identifikátory ω_i ($\omega_i \in \{-1, 1\}$).

V každém kroku algoritmu se mění důležitost správné klasifikace jednotlivých testovaných prvků. Pro každý krok k je důležitost dána množinou vah $D_{k+1}(i)$ takovou, že $\sum_{i=1}^{m} D_{k+1}(i) = 1$. Na počátku jsou všechny váhy stejné, s narůstajícím počtem iterací se zvyšuje váha u nerozpoznaných vzorků. Klasifikátory se v následných iteracích zaměřují právě na vzorky, které se nedaří rozpoznat. Tímto způsobem se algoritmus adaptuje na těžko rozpoznatelné vzorky a upravuje svou funkci tak, aby byly i tyto vzorky zařazeny do správné třídy.

V každé iteraci se algoritmus snaží nalézt právě jeden ze slabých klasifikátorů, který si vede nad danými trénovacími vzorky nejlépe.

Uvádím algoritmus AdaBoost dle [18], jehož výsledkem je lineární klasifikátor.

1. Inicializuj K (počet slabých klasifikátorů).
2. Nastav $k = 1$ a inicializuj $D_1(i) = \frac{1}{m}$.
3. Pro každé k trénuj slabý klasifikátor W_k za pomocí trénovací množiny a množiny vah $D_k(i)$ tak, že je každému vzorku $\mathbf{x}_i$; $W_k : X \to \mathcal{R}$ přiřazeno reálné číslo.
4. Zvol $\alpha_k > 0 \in \mathcal{R}$.
5. Spočítej

$$D_{k+1}(i) = \frac{D_k(i) e^{-\alpha_k \ \omega_i \ W_k(\mathbf{x}_i)}}{Z_k}, \tag{2.14}$$

   kde Z_k je normalizační faktor zvolen tak, aby $\sum_{i=1}^{m} D_{k+1}(i) = 1$.
6. Nastav $k = k + 1$.
7. Jestliže $k \leq K$, vrať se ke kroku 3.

8. Výsledný klasifikátor S je definován jako

$$S(\mathbf{x}_i) = sign\left(\sum_{k=1}^{K} \alpha_k \, W_k(\mathbf{x}_i)\right). \tag{2.15}$$

Výpočet α_k může být proveden různými způsoby. Pro klasifikaci do dvou tříd se používá vztah

$$\alpha_k = \frac{1}{2} ln\left(\frac{1-\epsilon_k}{\epsilon_k}\right), \tag{2.16}$$

kde ϵ_k odpovídá chybě, která je definována vztahem

$$\epsilon_k = \sum_{i=1}^{m} P_{i\sim D_k(i)}[W_k(\mathbf{x}_i) \neq \omega_i]. \tag{2.17}$$

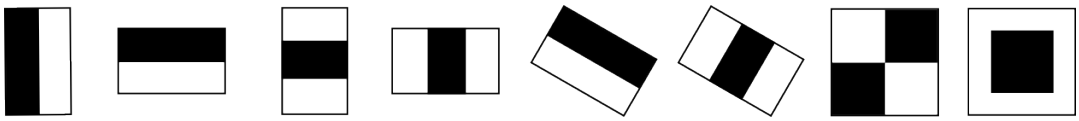
Algoritmus AdaBoost může bez problémů dosahovat úspěšnosti blížící se ke 100%. Stačí k tomu pouze slabé klasifikátory, které jsou alespoň o trochu lepší než úplně náhodné a dostatečný počet trénovacích cyklů. Praxe ukazuje, že algoritmus není většinou náchylný na přetrénování, a to i v případě, kdy je spuštěno několik tisíc trénovacích cyklů.

2.4.3 Haarovy příznaky

Jak je uvedeno výše, při tvorbě silného klasifikátoru můžeme použít jakýchkoliv slabých klasifikátorů, které jsou jen o něco málo lepší než úplně náhodné (jejich chyba je nižší než 0.5). Při zpracování obrazu se velmi často setkáme s použitím obrazových příznaků (*features* 2.4.1).

Nejběžněji prakticky používanými příznaky jsou tzv. Haarovy příznaky. Jedná se o obdélníky, které jsou složeny z černých a bílých oblastí. Klasifikace tímto způsobem využívá toho, že kontrast podobných objektů může být zakódován pomoci takovýchto příznaků.

Typickými zástupci jsou příznaky uvedené na obrázku 2.10. Různé kombinace černých a bílých oblastí naznačují vhodnost pro detekci různých příznaků v obraze. Například první dvojice se hodí pro detekci hran nebo přechodů mezi tmavými a světlými oblastmi v obraze. Zatimco druhá dvojice je vhodná spíše pro detekci tmavých linek nebo tmavých oblastí ležících mezi světlými oblastmi.



Obrázek 2.10: Haarovy příznaky.

Do základní množiny příznaků patří ty obdélníkové, nenatočené. V praxi mohou Haarovy příznaky nabývat různých tvarů, ale čím složitější tvar, tím pomalejší výpočet odezvy.

Odezvu příznaku aplikovaného na místo v obraze spočteme pomocí vztahu 2.18.

$$f(x) = \sum_{w\in W} x(w) - \sum_{b\in B} x(b), \tag{2.18}$$

kde x je vzorek dat, W a B jsou množiny pixelů, které příslušejí k bílé nebo černé oblasti právě aplikovaného příznaku.

Klasifikátor potom získává znalost o objektu na základě jeho typického kontrastu – pomicí specifické kombinace příznaků.

Další důležitou vlastností Haarových příznaků je rychlost výpočtu jejich odezvy. Při trénování klasifikátoru jsou příznaky aplikovány postupně na každé místo obrazu a mění se jejich velikost. Můžeme sice počítat odezvy pixel po pixelu, ale to by nebylo efektivní a v praxi se to nepoužívá. Místo toho se používá speciální reprezentace obrazu – integrální obraz.

2.4.4 Integrální obraz

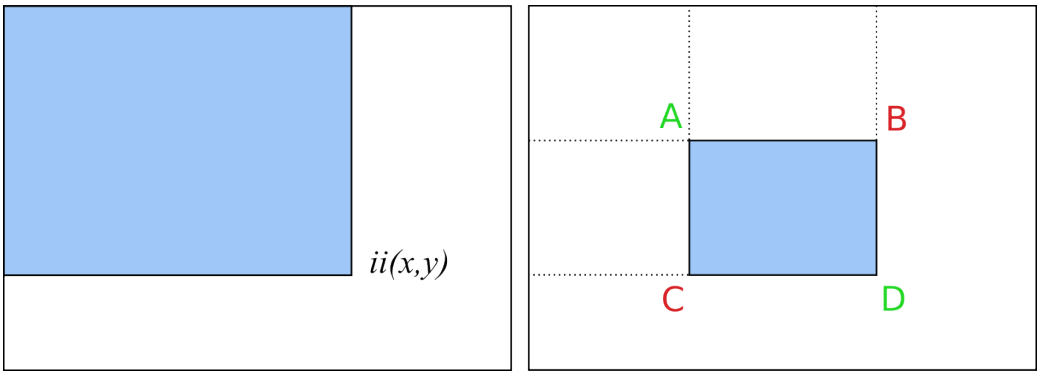
Obdélníkové příznaky mohou být efektivně vypočítány pomocí integrálního obrazu. Jedná se o reprezentaci obrazu, která vznikla v počítačové grafice pro použití v mipmapách ([24]) pod názvem *summed area table*, ale nebyla nijak více využívána až do doby, kdy pro ni našli využití Viola & Jones [19].

Princip integrálního obrazu spočívá v tom, že každý bod x, y odpovídá součtu hodnot pixelů, které se nachází v obraze vlevo nahoře od tohoto bodu. Matematicky můžeme popsat integrální obraz následovně:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'), \tag{2.19}$$

kde $ii(x, y)$ je bod v integrálním obrazu a $i(x', y')$ je pixel původního obrázku.

Díky této reprezentaci je možné spočítat odezvu obdélníkového Haarova příznaku kdekoliv v obraze v konstantním čase, navíc budou vyhledávány hodnoty pouze čtyř rohových bodů, což je samozřejmě neporovnatelné s tím, kdybychom počítali odezvu pixel po pixelu.



Obrázek 2.11: Integrální obraz.

Pro obdélník A, B, C, D na obrázku 2.11 vyjádříme sumu pixelů jako A - B - C + D (D musíme přičíst protože bylo odečteno dvakrát).

Kapitola 3

Pohyb golfového míčku

První golfové míčky byly vyráběny ze dřeva. Na počátku 17. století se objevil tzv. *featherie*. Šlo o kožený váček, který byl naplněn kachním peřím, které bylo do míčku vkládáno mokré. Po vysušení byl míček tvrdý a měl na tehdejší dobu dobré fyzikální vlastnosti i přesto, že se pokaždé nepodařilo, aby měl přesný kulový tvar. Někdy se stávalo, že se díky opotřebení švů během letu rozpadl, nicméně sloužil jako standard po dvě století.

V roce 1848 ([20]) byl vyroben první míček z gutaperči. Gutaperča je druh gumy pocházející z jihovýchodní Asie podobající se kaučuku, ale její pružnost je menší než pružnost kaučuku. V pozdějších letech byly míčky vyráběny z několika vrstev až se vyvinuly do podoby, kterou známe dnes.

Během let, po která se míček vyvíjel, bylo odpozorováno, že míček má lepší letové vlastnosti, pokud jsou na jeho povrchu mírné nerovnosti. V současné době jsou tyto nerovnosti – důlky (angl. *dimples*) pro míček charakteristickým rysem. Na povrchu míčku jsou důležité, protože snižují odpor vzduchu, a proto dosahuje míček lepších letových vlastností.

Rozměry golfového míčku jsou stanoveny Americkou golfovou asociací (USGA[1]). Váha míčku nesmí přesáhnout 45.93g. Průměr nesmí být menší než 42.67mm.

V této kapitole jsem vycházel především z knih [7], [11] a článků [13], [3] a [1].

3.1 Mechanika

3.1.1 Pohyb

Pohyb golfového míčku vychází šikmému vrhu s tím, že na něj působí ještě další síly, které znatelně deformují jeho parabolickou dráhu.

Šikmý vrh lze rozložit na dva na sobě nezávislé pohyby – jeden ve vodorovném a druhý ve svislém směru.

Ve vodorovném směru jde o rovnoměrně zrychlený pohyb, ve svislém směru o svislý vrh.

Rovnice pro rovnoměrně zrychlený pohyb má v našem případě následující tvar:

$$x(t) = (v_0 \; cos \; \theta_0)t, \tag{3.1}$$

kde v_0 je počáteční rychlost míčku, θ_0 je úhel odpalu v radiánech a $x(t)$ je poloha míčku v čase t ve vodorovném směru.

Pro pohyb ve svislém směru použijeme rovnici pro svislý vrh:

[1] http://www.usga.org

$$y(t) = (v_0 \ sin \ \theta_0)t - \frac{1}{2}gt^2, \tag{3.2}$$

kde g je gravitační zrychlení a $y(t)$ je poloha míčku v čase t ve svislém směru.

Vyloučením času z rovnice 3.1 a jejím následným dosazením do 3.2 dostaneme rovnici trajektorie vyjádřenou vztahem:

$$y = (tg \ \theta_0)x - \frac{gx^2}{2(v_0 \ cos \ \theta_0)^2} \tag{3.3}$$

Tvar této rovnice odpovídá rovnici paraboly $y = ax + bx^2$ a míček tedy opisuje parabolickou dráhu.

Z rovnic 3.1 a 3.2 lze také vyjádřit dolet R míčku jako:

$$R = \frac{2v_0^2}{g} \ sin \ \theta_0 \ cos \ \theta_0 = \frac{v_0^2}{g} \ sin \ 2\theta_0 \tag{3.4}$$

Deformace míčku Deformací míčku při odpalu dochází díky jeho pružnosti ke zvýšení jeho počáteční rychlosti vzhledem k rychlosti hole. V simulaci však deformaci nebudu brát v potaz, protože nebudu sledovat hůl, nýbrž samotný míček v letu.

Odraz míčku Odraz míčku můžeme určit se znalostí, že úhel dopadu se rovná úhlu odrazu. Je potřeba však počítat s tím, že odraz je nepružný. To znamená, že při odrazu dochází ke ztrátě kinetické energie. Dle [7] je ztráta kinetické energie u golfového míčku přibližně 60%. Nutno je také vzít v úvahu to, že se míček neodráží vždy od stejně pružného povrchu – záleží, zda dopadá na green nebo třeba do rawu.

3.1.2 Vlivy prostředí

Jestliže se chceme pokusit o skutečnou simulaci pohybu golfového míčku, nevystačíme si pouze s výše uvedenými rovnicemi. Musíme vzít v potaz také vlivy prostředí, ve kterém se míček pohybuje. K těmto vlivům patří Magnusův jev, odpor vzduchu, případně vítr a atmosferické podmínky.

Magnusův jev jsem již zmiňoval výše, a proto uvedu pouze vzorec převzatý z [21].

$$\vec{F_M} = \ S(\vec{\omega} \ \times \ \vec{v}) \tag{3.5}$$

Při pohybu míčku vzduchem na něj působí odporová síla F, která mu brání v pohybu. Síla působí proti směru pohybu míčku a lze ji vyjádřit vztahem:

$$F = \frac{1}{2}C\varrho S \ v^2, \tag{3.6}$$

kde ϱ je hustota vzduchu, S je tzv. účinný průřez tělesa, neboli obsah největšího řezu tělesa rovinou kolmou na směr pohybu. C je součinitel odporu, jenž se zjišťuje experimentálně a jeho hodnota se může v závislosti na rychlosti měnit (zejména k tomu dochází při výrazných změnách rychlosti). Dovolím si tyto změny zanedbat a pracovat s C jako s konstantou. Rychlost míčku reprezentuje proměnná v.

Z toho vztahu vyplývá jedna zajímavá věc; a sice, že velikost síly F roste s rostoucí rychlostí v. To znamená, že pokud těleso získá dostatečně velkou rychlost, resp. letí dostatečně dlouhou dobu, dojde k vyrovnání odporové síly F a tíhové síly $G = mg$. Díky tomu

bude zrychlení tělesa nulové. Tento stav se nazývá dosažení mezní rychlosti v_m, kterou lze zjistit z rovnosti $F = mg$ dosazením do 3.6. Po úpravách dostáváme:

$$v_m = \sqrt{\frac{2mg}{C\varrho S}}. \tag{3.7}$$

Výše uvedené vztahy jsou dobré jako teoretický základ pro další úpravy, ale pro simulaci jsou příliš nepřesné – neberou v potaz skutečné vlivy, které na míček během letu působí ani jeho specifické vlastnosti. V části 3.2 uvádím spolehlivější způsob určení trajektorie pomocí diferenciálních rovnic.

3.2 Reálný pohyb

Golfový míček lze ve skutečném světě považovat za balistický projektil, na který působí během letu různé fyzikální vlivy. Mezi tyto vlivy patří například proměnlivý vítr, hustota vzduchu či teplota prostředí. Nicméně proměnlivost těchto veličin je patrná hlavně v balistice při sledování drah projektilů, které jsou nesrovnatelně delší než trajektorie golfového míčku. Při relativně krátkém letu golfového míčku není nutno se zabývat ani proměnlivou teplotou, ani hustotou vzduchu. Budu tyto veličiny tudíž považovat za konstanty. Co se týče rychlosti větru, ta se v golfových simulátorech běžně považuje také za konstantu.

Na trajektorii míčku mají vliv i další síly, které jsou ovlivňovány jeho povrchovou úpravou. V následující části se zaměřím na funkci důlků na povrchu míčku a představím způsob odhadu jeho trajektorie.

Vycházím zde z [11], [13] a [1].

3.2.1 Funkce důlků

Důlky (*dimples*) mají vliv jak na délku trajektorie míčku, tak na její tvar. Děje se tomu tak proto, že přímo ovlivňují dvě klíčové věci. Zaprvé podporují vznik dynamického vztlaku. Zadruhé snižují odpor prostředí (součinitel odporu).

Větší vliv na trajektorii má podle [13] snížení odporu prostředí, proto dále nebude na dynamický vztlak brán zřetel, a v rovnicích pro odhad trajektorie se tudíž neobjeví. Navíc není možno ani určit ze získaného záznamu díky nízkému rozlišení kamer rotaci míčku, která vztlak způsobuje. Pro úplnost však uvádím i informace o jeho vlivu na trajektorii.

Idealizovaný případ

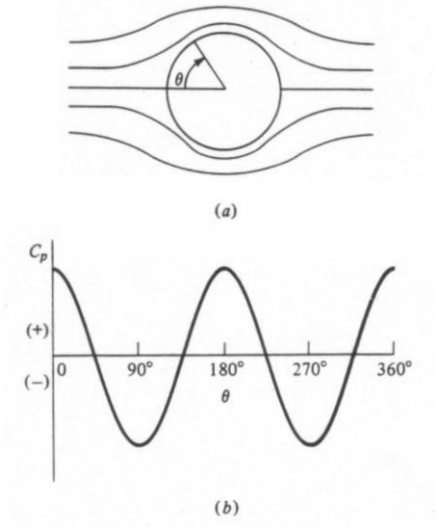
Abychom lépe pochopili vlastnosti golfového míčku, především to, jaký vliv mají důlky na povrchu míčku na trajektorii letu, použijeme nejprve zjednodušený model míčku.

Míček bude mít stejnou hmotnost a poloměr, ovšem jeho povrch bude hladký. V ideálním prostředí bez odporu vzduchu by se míček choval tak, jak je naznačeno na obrázku 3.1.

Úhel θ symbolizuje pozici vzhledem k povrchu míčku. Přední strana, která se nejdříve setkává s tokem vzduchu má úhel $\theta = 0°$, zadní $\theta = 180°$. Horní bod je symbolizován úhlem $\theta = 90°$ a spodní $\theta = 270°$.

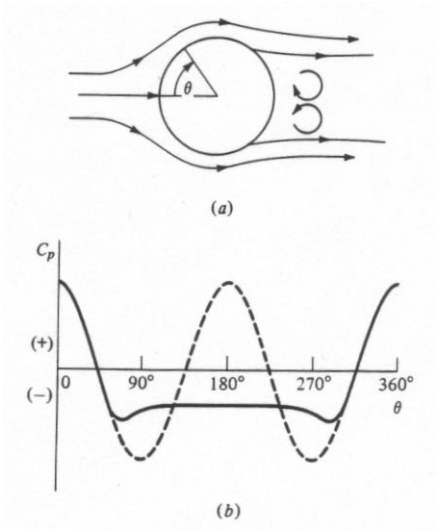
Ve spodní části obrázku 3.1 je znázorněno rozložení tlaku, který na míček během letu působí. Vidíme, že velikost tlaku je stejná jak při $\theta = 0°$, tak při $\theta = 180°$. To znamená, že síla v zadní části míče ruší sílu v přední části, a proto na míček nepůsobí odpor prostředí.

V reálném světě dochází k odlišnému chování, které je znázorněno na obrázku 3.2. Proudění při $\theta = 0°$, se liší od toho při $\theta = 180°$. Vidíme, že dochází ke změně rozložení



Obrázek 3.1: (a) Obtékání idealizovaného míčku vzduchem. (b) Působení tlaku vzduchu na pohybující se idelizovaný míček. Zdroj [1].

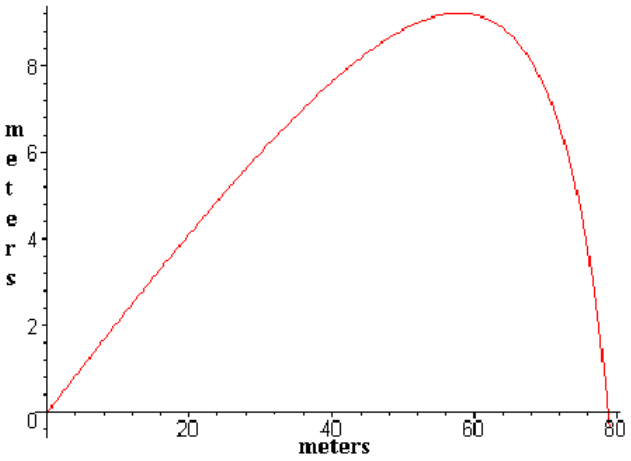
tlaku na povrchu míčku. Dochází k oddělení proudění od povrchu míče a ke vzniku pásma s víry, které zapříčiňují vyšší odpor prostředí.



Obrázek 3.2: (a) Obtékání reálného míčku vzduchem. (b) Působení tlaku vzduchu na pohybující se míček. Přerušovaná čára znázorňuje ideální případ. Zdroj [1].

Pro míček stejné velikosti a hmotnosti jako je ten golfový, jen s tím rozdílem, že nemá na povrchu důlky, bude trajektorie vypadat tak, jak je ukázáno na obrázku 3.3.

Graf na obrázku 3.3 je výsledkem simulace, která byla provedena v [13] s ohledem na skutečné odpaly profesionálních hráčů. Profesionál dokáže odpálit míček s 2000 až 5000 RPM (záleží na typu hole). I když je na první pohled tvar křivky v pořádku, tak oproti odpalu s golfovým míčkem nesouhlasí dolet míčku (příliš krátký) ani vrchol trajketorie (příliš nízko).



Obrázek 3.3: Trajektorie idealizovaného míčku. Zdroj [3].

Zvýšení dynamického vztlaku

Dynamický vztlak, který je díky důlkům zvyšován vzniká při odpalu míčku. Míček dostává velkou zpětnou rotaci, která zapříčiňuje, že vzduch na horní straně míčku proudí jinou rychlostí než na spodní straně.

Jestliže míček letí rychlostí v_0 a jeho úhlová rychlost je ω, můžeme potom zapsat rychlosti proudícího vzduchu na horní a dolní straně míčku následovně:

$$\begin{aligned} v_{top} &= v_0 - \omega r \\ v_{bottom} &= v_0 + \omega r, \end{aligned} \tag{3.8}$$

kde r je poloměr míčku. Na obrázku 3.4 můžeme vidět, že oproti případu s hladkým míčkem dochází tentokrát k vystoupání míčku do vyšší výšky a zároveň je celá fáze stoupání delší a následuje příkřejší pád. Tento druh trajektorie je v golfovém slangu nazýván jako *blow up*.

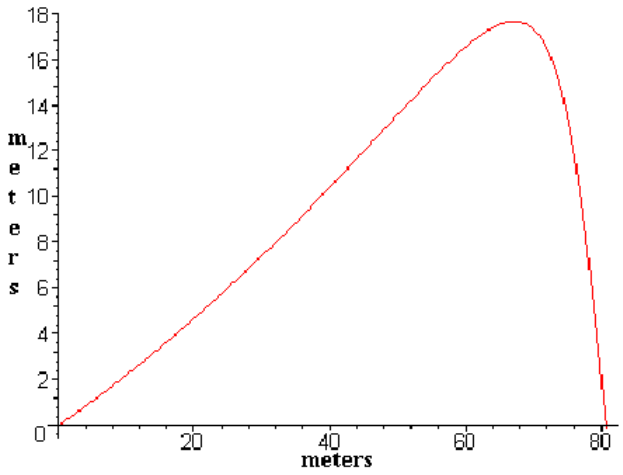
V případě, kdy není osa rotace míčku horizontální, je tento jev dobře znatelný zejména při bezvětří. Jedná se o Magnusův jev. Magnusovým jevem je vlastně vznik boční síly, která vzniká právě při obtékání pohybujícího se tělesa proudícím plynem či kapalinou. Tento jev byl objeven už v roce 1672 Isaacem Newtonem a v roce 1852 podrobně popsán Gustavem Heinrichem Magnusem.

Snížení odporu prostředí

Důlky snižují odpor prostředí právě proto, že vytvářejí u povrchu míčku turbulence. Jako důsledek těchto turbulencí dochází k oddálení místa, kde se oddělí tzv. mezní vrstva (*boundary layer*), což je tenká vrstva vzduchu u povrchu pohybujícího se míče. Díky tomu vzniká v oblasti okolo $\theta = 180°$ menší víření. A to je důvod snížení odporu prostředí.

Pro ilustraci této situace viz obrázek 3.5.

Ve skutečnosti důlky ovlivňují Reynoldsovo číslo, které dává do souvislosti rozměry tělesa s prouděním okolního prostředí. Definice vztahu pro Reynoldsovo číslo Re je vyjádřena vztahem



Obrázek 3.4: Trajektorie se započítaným dynamickým vztlakem. Zdroj [3].

$$Re = \frac{\rho v l}{\mu}, \tag{3.9}$$

kde ρ je hustota tekutiny (vzduchu), v je rychlost tělesa, l je rozměr tělěsa (průměr míčku) a μ je viskozita prostředí. Jinými slovy můžeme říci, že pokud vezmeme dvě rozdílně velká tělesa pohybující se různými rychlostmi ve stejném prostředí, můžeme o těchto tělesech prohlásit, že mají stejné aerodynamické vlastnosti pokud mají stejné Reynoldsovo číslo.

Na obrázku 3.6 je ukázána závislost odporu prostředí C_D a Re. Při nízkém Re obvykle zůstává mezní vrstva laminární, což je u většiny těles žádoucí, nicméně u golfového míčku tomu tak není. Problémem je, že je tato vrstva náchylná na oddělení se od povrchu míčku a dochází tím k efektu, který je naznačen v horní části obrázku 3.5. Důlky zabraňují oddělení laminární vrstvy od povrchu míčku tím, že dojde ke změně vrstvy na turbulentní (začnou se za míčkem vytvářet víry) a víření vzduchu udrží tuto vrstvu déle u povrchu míčku. Dojde ke zrychlení proudění vzduchu za míčkem a tím ke snížení odporu prostředí.

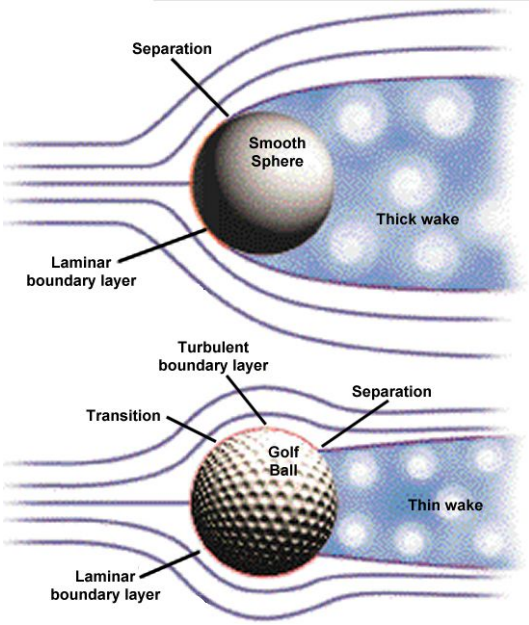
Práce [13] se zabývá aerodynamickými vlastnostmi golfového míčku. Zjišťování vlastností probíhalo ve vzdušném tunelu a bylo zjištěno, že součinitel odporu se pohybuje v rozmezí od 0.27 do 0.32. Při aproximaci trajektorie proto budu pracovat s hodnotou 0.29.

Po zahrnutí dynamického vztlaku a součinitle odporu do modelu z 3.3, dostáváme průběh podobný tomu, který je uveden na obrázku 3.7. Vidíme, že díky dynamickému vztlaku se zvětšila délka trajektorie přibližně o 50 metrů oproti případu, kdy bylo počítáno pouze se součinitelem odporu.

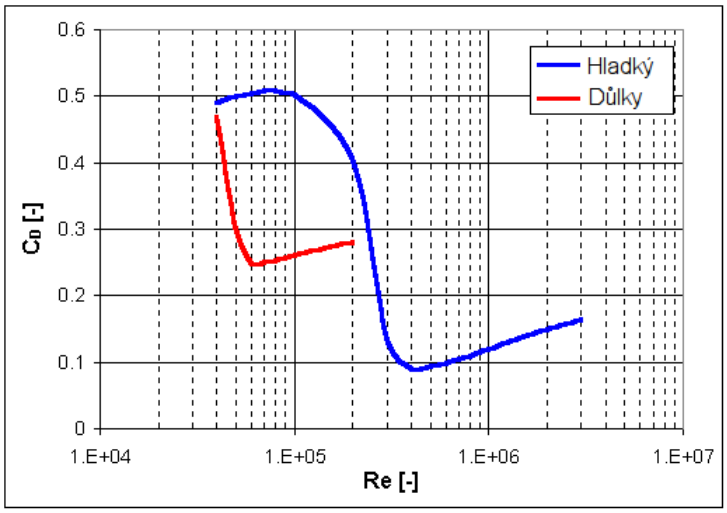
Z výše uvedených informací plyne, že snížení odporu prostředí má na trajektorii míčku větší vliv, než zvýšení dynamického vztlaku. Nicméně vývoj golfového míčku ještě stále není u svého konce. Velká část poznatků byla zjištěna experimentálně. S nástupem počítačů a moderní aerodynamiky by mohlo dojít k optimalizaci vlastností současných míčků.

3.2.2 Výpočet trajektorie

Pro výpočet trajektorie míčku jsem se rozhodl použít balistických rovnic uvedených v [11]. Učinil jsem tak proto, že lze golfový míček přirovnat k vystřelenému projektilu. Vnější balistika, která se zabývá pohybem projektilů ve vzduchu je jednou z oblastí fyziky, která je poměrně dobře probádaná a existují sady rovnic pro aproximaci trajektorie projektilů.



Obrázek 3.5: Porovnání proudění vzduchu kolem idealizovaného a golfového míčku. Zdroj [1].

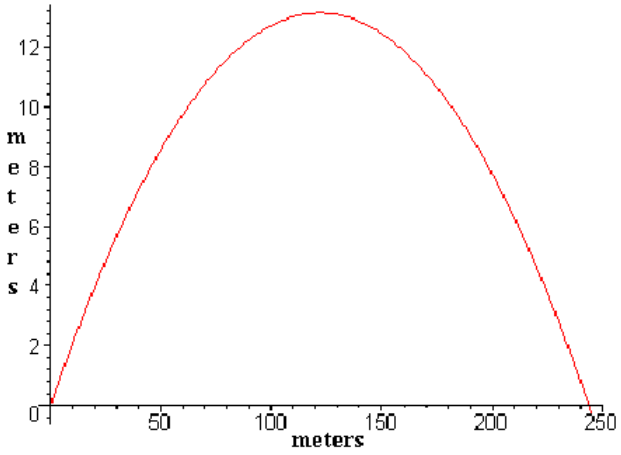


Obrázek 3.6: Závislost odporu prostředí a Reynoldsova čísla. Zdroj [13].

Při výpočtu trajektorie budu předpokládat, že je její počátek v bodě [0, 0, 0] a tento bod leží v nulové nadmořské výšce. Standartní hustotu vzduchu budu značit jako ρ^*. Běžně udávaná hodnota je 1.22521 $kg \cdot m^{-3}$ při 15 °C. Hustota vzduchu se s nadmořskou výškou mění. Tuto závislost vyjadřuje následující vztah:

$$\rho(y) = \rho^* H(y) \tag{3.10}$$

Pro trajektorie, jejichž minimum a maximum se liší jen o několik metrů se používá $H(y) = 1$, tudíž $\rho(y) = \rho^*$. V ostatních případech se používá vztah 3.11:



Obrázek 3.7: Trajektorie se započítaným součinitelem odporu. Zdroj [3].

$$H(y) = e^{-hy}, \tag{3.11}$$

kde konstanta $h = 0.0001036\ m^{-1}$. V této práci se však spokojíme s první variantou.

Pro výpočet bude důležitá také rychlost zvuku. Lze ji vyjádřit v závislosti na nadmořské výšce vztahem 3.12

$$u_s(y) = u_s(0)e^{-a_1 y}, \tag{3.12}$$

kde $a_1 = 2h/21$. Rychlost zvuku v nadmořské výšce 0 je při 15 °C 340.31 $m \cdot s^{-1}$.

Nyní můžeme zapsat rovnice pro pohyb míčku ve 3D:

$$\begin{aligned} \ddot{x} &= -\left(\frac{d^2}{m}\right) \rho^* \, C_D(v/u_S(y)) \, v\dot{x}, \\ \ddot{y} &= -\left(\frac{d^2}{m}\right) \rho^* \, C_D(v/u_S(y)) \, v\dot{y} \; - \; g, \\ \ddot{z} &= -\left(\frac{d^2}{m}\right) \rho^* \, C_D(v/u_S(y)) \, v\dot{z}, \end{aligned} \tag{3.13}$$

kde d je průměr míčku, m je jeho hmotnost, $\rho*$ je hustota vzduchu v 0m n. m. a C_D je součinitel odporu, který jsem v části 3.2.1 stanovil jako konstantu 0.29.

První derivace obecně vyjadřuje změnu dráhy v čase, tudíž $\dot{x}$ reprezentuje rychlost míčku v ose x. Druhá derivace značí změnu rychlosti v čase a proto $\ddot{x}$ znamená zrychlení v ose x. Analogicky toto samozřejmě platí i pro ostatní osy. Vidíme, že zrychlení je záporné, proto je jasné, že míček bude po dobu svého letu stále zpomalovat.

Aktuální rychlost míčku v v bodě $[x, y, z]$ spočteme pomocí vztahu 3.14.

$$v = \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2}. \tag{3.14}$$

Pozorný čtenář si jistě všimnul, že v rovnicích není zahrnuto působení větru. Toto lze napravit tím, že přepíšeme rovnice 3.13 do tvaru:

$$
\begin{aligned}
\ddot{x} &= -\left(\frac{d^2}{m}\right) \rho^* \; C_D(v/u_S(y)) \; v(\dot{x} + w_x), \\
\ddot{y} &= -\left(\frac{d^2}{m}\right) \rho^* \; C_D(v/u_S(y)) \; v(\dot{y} + w_y) \; - \; g, \\
\ddot{z} &= -\left(\frac{d^2}{m}\right) \rho^* \; C_D(v/u_S(y)) \; v(\dot{z} + w_z),
\end{aligned}
\tag{3.15}
$$

kde $\vec{w} = (w_x, w_y, w_z)$ je vektor reprezentující směr a rychlost proudění větru.

Nyní máme dostatek informací o chování míčku v reálném prostředí. Definovali jsme vztahy potřebné pro praktickou část práce. Nyní se budu věnovat demonstrační aplikaci.

Kapitola 4

Demonstrační aplikace

V této kapitole je uvedeno, jak jsem postupoval při implementaci demonstrační aplikace a jakých prostředků jsem využil.

Nastíním zde také schéma golfového simulátoru, především rozestavení kamer a pricip fungování celé aplikace.

4.1 Úvod

Demonstrační aplikace, která je produktem této práce, bude součástí celého systému pro simulaci golfové hry. Tento systém je primárně určen pro použití ve vnitřních zařízeních, tzv. *indoorech*.

V této části práce nejprve představím koncept golfového simulátoru, který bude produkovat firma Inven Solution s.r.o., dále uvedu, jakých jsem použil prostředků při vývoji aplikace, a nakonec představím samotnou aplikaci.

4.2 Simulátor

Jak jsem uváděl výše, simulátor je určen do vnitřních prostor. Při vytváření aplikace byla používána dvojice kamer, nicméně do budoucna se zvažuje možnost přidání třetí kamery pro zvýšení přesnosti systému. Rozmístění kamer je uvedeno na obrázku 4.1. Kameru, která je vpravo dole budu dále v textu označovat jako *přední*, protože je umístěna před hráčem, a kameru umístěnou na stropě jako *horní*.

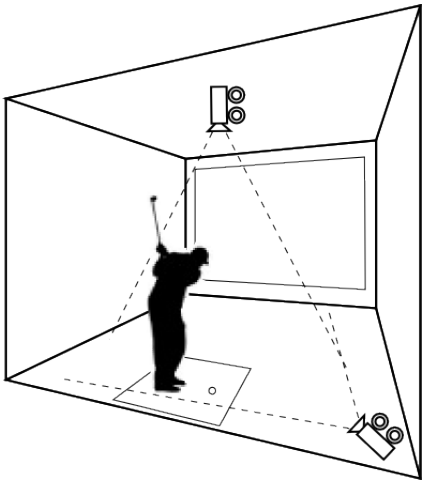
Pro vývojové účely byly použity kamery Sony [22], které zvládají snímat obraz rychlostí až 125 snímků za sekundu při rozlišní 320x240 pixelů.

Pro správnou funkci je důležité, aby pozadí mělo pokud možno jednotnou barvu (nejlépe černou, případně tmavě zelenou), resp. aby bylo pozadí takové, že se na něm bude bez problémů dát rozlišit letící míček. Při tvorbě klasifikátoru bylo použito černé pozadí.

V následující části uvádím, jak jsem postupoval při trénování klasifikátoru. Vycházel jsem z návodů na [16] a [15].

4.3 Vytvoření klasifikátoru pomocí OpenCV

Klasifikátor pro detekování golfového míčku v obraze jsem vytvářel pomocí předem připravené sady nástrojů v OpenCV, kterou obsahuje každá instalace. Jedná se o nástroje, pomocí



Obrázek 4.1: Schéma simulátoru.

kterých vývojáři OpenCV trénovali například klasifikátor pro rozpoznávání obličejů, který můžeme najít mezi demonstračními příklady. Konkrétně jsem využil *opencv_createsamples* a *opencv_haartraining*. Pro vyhodnocení úspěšnosti získaného klasifikátoru jsem použil *opencv_performance*.

Kladnou vlastností je, že nástroje nejsou závislé na konkrétním objektu, který chceme rozpoznávat. Nástroj pro trénování klasifikátoru je založen na haarových vlnkách (2.4.3), proto nezáleží na tom, zda klasifikujeme jablka nebo třeba granáty. Úspěšnost vygenerovaného klasifikátoru závisí na datech, která dodáme pro jeho natrénování.

Potřebujeme shromáždit dvě množiny dat – negativní a pozitivní vzorky. Při rozpoznávání obličejů bylo dle [10] použito 5000 pozitivních a 3000 negativních vzorků, proto je potřeba shromáždit co největší množství vzorků. Pro tento účel jsem použil videozáznamy, ze kterých jsem extrahoval vhodné snímky (vhodné znamená ty, na kterých se vyskytovala stopa od míčku).

Data je dobré si rozdělit na dvě skupiny. Jednu pro trénování a druhou pro testy. Používat pro testování a trénování identická data se nedoporučuje.

Nyní popíšu postup, který jsem použil při tvorbě klasifikátorů.

4.3.1 Vytvoření negativních vzorků

Negativní vzorky mohou být libovolné obrázky, které splňují jedinou podmínku – nesmí se na nich vyskytovat hledaný objekt. V literatuře jsou tyto obrázky někdy nazývány také jako pozadí (*background images*).

Při zpracování pomocí OpenCV se používají textové soubory s popisem umístění vzorků.

Následující adresářovou strukturu:

Kód 4.1: Adresářová struktura s negativními vzorky

```
/negatives
        obr1.png
        obr2.png
        obr3.png
negatives.dat
```

reprezentujeme pomocí souboru *negatives.dat*, jehož obsah je uveden v kódu 4.2.

Kód 4.2: Textový soubor se seznamem negativních vzorků

```
negatives/obr1.png
negatives/obr2.png
negatives/obr3.png
```

Takovýto soubor lze získat použitím příkazu 4.3.

Kód 4.3: Vytvoření seznamu negativních vzorků

```
$ find negatives/ -name '*.png' > negatives.dat
```

4.3.2 Vytvoření pozitivních vzorků

Pro vytvoření pozitivních vzorků se používají obrázky, na kterých je zachycen hledaný objekt.

Na vytvoření pozitivních vzorků má OpenCV nástroj *opencv_createsamples* 4.4.

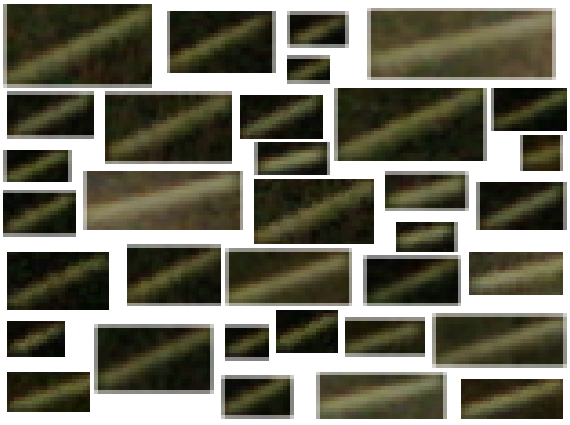
Kód 4.4: Parametry funkce `opencv_createsamples`

```
Usage: opencv_createsamples
  [-info <collection_file_name>]
  [-img <image_file_name>]
  [-vec <vec_file_name>]
  [-bg <background_file_name>]
  [-num <number_of_samples = 1000>]
  [-bgcolor <background_color = 0>]
  [-inv] [-randinv] [-bgthresh <background_color_threshold = 80>]
  [-maxidev <max_intensity_deviation = 40>]
  [-maxxangle <max_x_rotation_angle = 1.100000>]
  [-maxyangle <max_y_rotation_angle = 1.100000>]
  [-maxzangle <max_z_rotation_angle = 0.500000>]
  [-show [<scale = 4.000000>]]
  [-w <sample_width = 24>]
  [-h <sample_height = 24>]
```

S její pomocí lze vytvořit vzorky jak z jednoho, tak z více obrázků. Dále umí vytvořit například testovací vzorky nebo zobrazit obrázky uložené v tzv. *vec* souboru. *Vec* soubor je soubor, který obsahuje již vygenerované vzorky.

V mém případě jsem pro tvorbu použil výřezy z videa zachycujícího let míčku (obr. 4.2). Získal jsem tak sadu asi sta různých obrázků. Nicméně toto číslo nebylo konečné. Na každý z nich jsem použil *opencv_createsamples* a vygeneroval tak dalších několik vzorků. Vzniklé *vec* soubory jsem spojil do jednoho pomocí utility *mergevec* [14].

Nyní ukážu podrobněji, jak se dají generovat vzorky pomocí *opencv_createsamples*.



Obrázek 4.2: Vzorky použité při tvorbě klasifikátoru.

Z jednoho obrázku

Generování vzorků z jednoho obrázku funguje tak, že jsou na původní obrázek postupně aplikovány různé rotace či je měněna jeho intenzita. Tímto způsobem můžeme kompenzovat nízký počet vzorků extrahovaných z videa.

Funkci *opencv_createsamples* musíme spustit s parametry:

-img – obrázek, ze kterého chceme nagenerovat vzorky,

-bg – description file s obrázky pozadí,

-vec – výstupní soubor s vygenerovanými vzorky.

V mém případě jsem použil pro každý obrázek nástroj *opencv_createsamples* s nastavením uvedeným níže (4.5). Proměnnou N jsem nahradil počtem řádků v souboru *negatives.dat* – funkce používá totiž pouze prvních N negativních vzorků, proto by bylo zbytečné použít vyšší číslo.

Kód 4.5: Vytvoření několika vzorků z jednoho obrázku.

```
$ opencv_createsamples -img obr.png -num <N> -bg negatives.dat -vec
    positive<N>.vec -maxxangle 0.8 -maxyangle 0.8 -maxzangle 0.3 -maxidev
     50 -bgcolor 0 -bgthresh 0 -w 20 -h 20
```

Jak jsem uváděl již výše, vzniknuvší *vec* soubory jsem nakonec spojil do jednoho pomocí *mergevec*.

Z více obrázků

Pro generování vzorků z více obrázků použijeme nástroj *opencv_createsamples*, s následujícími parametry:

-info – soubor se seznamem pozitivních obrázků (formát viz kód 4.7),

-vec – výstupní soubor s vygenerovanými vzorky.

Tentokrát už nejsou na obrázek aplikovány žádné rotace či podobně. Z tohoto důvodu je funkci vhodné použít pouze v případě, že máme dostatečný počet obrázků (což znamená několik tisíc). V mém případě jsem si vystačil s prvním způsobem.

Kód 4.6: Vytvoření vzorků z více obrázků.

```
$ opencv_createsamples -info input.dat -vec positives.vec -w 20 -h 20
```

Formát vstupního souboru je následující:

Kód 4.7: Seznam pozitivních obrázků

```
[nazev_souboru] [pocet_objektu] [[x y sirka vyska] [dalsi_objekt] ...]
[nazev_souboru] [pocet_objektu] [[x y sirka vyska] [dalsi_objekt] ...]
...
```

kde x, y jsou souřadnice levého horního rohu objektu.

Výstupní vzorky budou z obrázku vyříznuty a poté bude upravena jejich velikost dle zadaných parametrů **-w** a **-h**.

V momentě, kdy máme nachystaná vstupní data, je možno začít s trénováním.

4.3.3 Trénování klasifikátoru

K trénování klasifikátoru jsem použil další z utilit, která se nachází v OpenCV viz kód 4.8. Jde o tzv. Haarovo trénování, které je založeno na Haarových vlnkách. O tomto tématu jsem se zmiňoval v části 2.4.3.

Kód 4.8: Parametry funkce `opencv_haartraining`

```
Usage: opencv_haartraining
  -data <dir_name>
  -vec <vec_file_name>
  -bg <background_file_name>
  [-bg-vecfile]
  [-npos <number_of_positive_samples = 2000>]
  [-nneg <number_of_negative_samples = 2000>]
  [-nstages <number_of_stages = 14>]
  [-nsplits <number_of_splits = 1>]
  [-mem <memory_in_MB = 200>]
  [-sym (default)] [-nonsym]
  [-minhitrate <min_hit_rate = 0.995000>]
  [-maxfalsealarm <max_false_alarm_rate = 0.500000>]
  [-weighttrimming <weight_trimming = 0.950000>]
  [-eqw]
  [-mode <BASIC (default) | CORE | ALL>]
  [-w <sample_width = 24>]
  [-h <sample_height = 24>]
  [-bt <DAB | RAB | LB | GAB (default)>]
  [-err <misclass (default) | gini | entropy>]
```

```
[-maxtreesplits <max_number_of_splits_in_tree_cascade = 0>]
[-minpos <min_number_of_positive_samples_per_cluster = 500>]
```

Správné nastavení parametrů vyžaduje trpělivost. Trénování klasifikůtoru může při velkém objemu trénovacích dat trvat i několik dní a je téměř nemožné odhadnout správné parametry hned napoprvé. Vzhledem k tomu, že počet vzorků, které jsem shromáždil nebyl nijak vysoký, trénování trvalo v řádu jednotek minut. Dobu trénování lze nejvíce ovlivnit pomocí parametrů **-minhitrate** a **-maxfalsealarm**, ale zároveň na nich závisí přesnost.

-minhitrate je nastavení minimální procentuální úspěšnosti,

-maxfalsealarm značí procento povolených chybných klasifikací (*false alarm* = vzorek označen jako hledaný vzorek, ale ve skutečnosti je chybný)

Použil jsem nastavení **-minhitrate** na 0.999 a **-maxfalsealarm** na 0.4 až 0.7 – jak pro který klasifikátor. Jako příklad uvádím kód 4.9 s nastavením použitým při trénování klasifikátoru pro vyhledání letícího míčku na videu z přední kamery.

Kód 4.9: Trénování klasifikátoru

```
$ opencv_haartraining -data <nazev_klasifikatoru> -vec positives.vec -bg
    negatives.dat -nstages 10 -minhitrate 0.999 -maxfalsealarm 0.6 -npos
    <pocet_pozitivnich_vzorku> -nneg <pocet_negativnich_vzorku> -w 20 -h
    20 -nonsym -mem 2048 -mode ALL
```

Význam dalších parametrů:

-nstages značí počet cyklů trénování, nicméně trénování může skončit i dříve, stačí, aby bylo dosaženo hodnot **-minhitrate** a **-maxfalsealarm 0.5**, nebo jsou odmítnuty všechny vzorky a je třeba je upravit,

-nonsym signalizuje, že vzorky nejsou symetrické podle vertikální osy (stopa od míčku je symetrická pouze při pohledu z horní kamery, přední kamera vidí stopu stoupající směrem do pravého horního rohu), kdyby byly, mohl by být proces učení urychlen, protože by byly vybrány jen některé Haarovy vlnky,

-mode ALL znamená, že bude použita rozšířená sada Haarových vlnek (tzn. navíc se použijí i vlnky otočené o 45°).

Tímto je trénování klasifikátoru u konce. Výstupem je xml soubor s klasifikátorem, který lze lehce použít v jakémkoliv C/C++ programu.

Pro urychlení práce při vytváření klasifikátoru jsem z výše uvedených příkazů vytvořil shell skript. Tento skript je součástí přiloženého CD a je u něj přiložen soubor *README*, který popisuje jeho použití.

4.4 Implementace

Aplikace je implementována v jazyce C++. Využil jsem prostředků knihovny OpenCV, která je určena pro počítačové vidění. Implementace aplikace spočívala především ve vytvoření dvou částí:

- **Detector** pro zpracování videa – vyhledávání míčku a zjišťení jeho co nejpřesnější polohy v obraze,
- **Physics** pro dopočítání trajektorie na základě získané 3D pozice míčku.

Obě části bylo nutné propojit se zbytkem systému. Po startu aplikace jsou načtena vstupní videa se zaznamenaným odpalem. *Analyzer* předává snímky *Detectoru*, který na nich vyhledá míček a vrátí jeho pozici v obou snímcích. Třída *Stereo*, která se stará o stereovizi, převede odpovídající si body na 2D snímcích do jejich skutečné 3D podoby. Po nalezení všech bodů, které jsou součástí trajektorie jsou předány třídě *Physics*, která pro každý bod dopočítá zbytek trajektorie. Získané trajektorie jsou poté proloženy křivkou a tím je získána aproximace reálné trajketorie míčku.

Nyní představím blíže mnou implementované třídy.

4.4.1 Detector

Detector zpracovává snímky z kamer a vyhledává v nich stopy od golfového míčku. Snímky jsou z kamer načítány synchronizovaně. Po nalezení míčku jsou jeho souřadnice v obou snímcích předány modulu, který vrátí skutečnou polohu míčku v prostoru (viz [5]). Modul ze dvou 2D souřadnic vytvoří jednu 3D souřadnici. Každá souřadnice je uložena do vektoru, který je později předán modulu Physics pro dopočítání trajektorie.

Nejprve je vyhledáván ležící míček. Protože z umístění kamer vím, kde se může v obraze vyskytovat, je nastaveno ROI[1] pro urychlení vyhledávání a eliminaci případných falešných detekcí. V této fázi je uložena výchozí pozice míčku, od které se bude počítat trajektorie.

Může se stát, že klasifikátor nalezne kromě míčku také další objekty (*false positives*), které jsou nežádoucí. Můžou to být například golfistovy boty nebo nevhodná část oblečení, případně odraz hole. U přední kamery jsem tento problém řešil tak, že za platný objekt beru ten, který je na snímku nejníže. U horní kamery je to objekt, který je nejvíce vpravo.

V okamžiku, kdy se nepodaří ve snímku vyhledat ležící míček, začnu s vyhledáváním pohybujícího se míčku. Pro tento účel použiji jiný klasifikátor, a to jak pro horní, tak pro přední kameru. Celkem tedy používám tři klasifikátory. První je pro obě kamery společný a hledá ležící míček. Druhý a třetí je pro pohybující se míček – zvlášť pro každou kameru. Vyhledávání letícího míčku je opět optimalizováno pomocí ROI. V rámci nalezeného výřezu se stopou od míčku jsem pro přesné dohledání počátku a konce stopy použil knihovnu cvBlobs[2].

Po tom, co je míček odpálen a jsou nalezeny body jeho trajektorie, přechází detektor zpět do stavu, kdy vyhledává ležící míček. Mezitím dojde k předání nalezených bodů fyzikálnímu modulu a dopočítání trajektorie.

4.4.2 Physics

Physics se stará o fyzikální část aplikace. Především jde o dopočítání trajektorie míčku. Jsou zde nastaveny všechny potřebné koeficienty pro co nejpřesnější predikci.

V průběhu implementace aplikace jsem narazil na článek [26], který se zabývá výpočtem trajektorie golfového míčku. Autor bere v potaz dynamický vztlak i rotaci míčku. Rozhodl jsem se použít rovnice uvedené v tomto článku, protože se blíží více realitě a jsou přesnější (viz obrázek B.1), než rovnice uvedené v kapitole 3.

[1] Region Of Interest - oblast zájmu, výřez z obrazu, který nás zajímá
[2] http://code.google.com/p/cvblob/

Rovnice mají následující tvar:

$$\ddot{x} = -\left(\frac{1}{m}\right)\left(-\frac{D\dot{x}}{v} - \frac{L\dot{y}}{v}\right), \tag{4.1}$$

$$\ddot{y} = -\left(\frac{1}{m}\right)\left(-\frac{D\dot{y}}{v} + \frac{L\dot{x}}{v} - mg\right), \tag{4.2}$$

kde D je odpor vzduchu vyjádřen vztahem 4.3 a dynamický vztlak L vztahem 4.4. Ostatní proměnné odpovídají definicím v kapitole 3.

$$D = (\tfrac{1}{2})\, C_d\, \rho\, v^2\, S, \tag{4.3}$$

$$L = (\tfrac{1}{2})\, C_l\, \rho\, v^2\, S \tag{4.4}$$

Koeficienty C_d a C_l pro odpor vzduchu resp. dynamický vztlak jsou definovány:

$$C_d = 0.1995 + 0.1890s + 1.4650s^2, \tag{4.5}$$

$$C_l = 0.0694 + 0.9879s, \tag{4.6}$$

kde s je rotace míčku vyjádřena vztahem:

$$s = \frac{\sqrt{2\pi\omega r}}{v}, \tag{4.7}$$

kde ω je počet zpětných otáček za sekundu a r poloměr míčku.

Pozorný čtenář si jistě všiml, že chybí rovnice pro osu z. V předchozím případě byla tato rovnice stejná jako rovnice pro osu x, nicméně v tentokrát to neplatí. Výsledky, které rovnice vracela byly chybné; i při nulovém větru docházelo k výraznému zatočení míčku doleva. Proto jsem s rovnicí experimentoval, mírně ji upravil a nakonec se mi povedlo dosáhnout uspokojivých výsledků.

Dále jsem zohlednil ještě působení větru ve všech rovnicích. Vítr vyjádříme stejně jako v kapitle 3 vektorem $\vec{w} = (w_x, w_y, w_z)$. Rovnice pro výpočet rychlosti míčku dostává tvar:

$$v = \sqrt{(\dot{x} + w_x)^2 + (\dot{z} + w_y)^2 + (\dot{z} + w_z)^2} \tag{4.8}$$

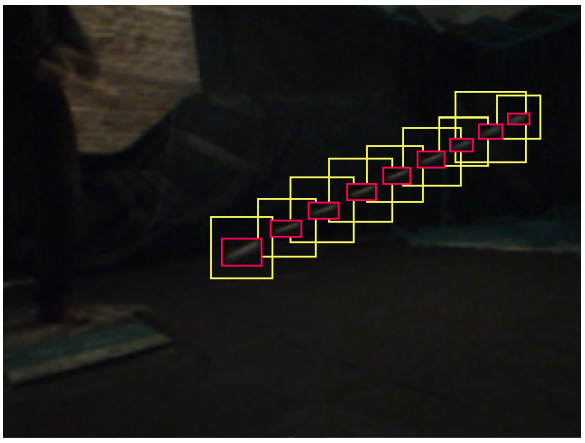
Tímto způsobem jsou v rovnicích zohledněny všechny důležité faktory, které ovlivňují míček v letu. Konečný tvar rovnic, které používám pro aproximaci trajektorie je uveden níže.

$$\begin{aligned}
\ddot{x} &= -\left(\frac{1}{m}\right)\left(-\frac{D(\dot{x} + w_x)}{v} - \frac{L(\dot{y} + w_y)}{v}\right), \\
\ddot{y} &= -\left(\frac{1}{m}\right)\left(-\frac{D(\dot{y} + w_y)}{v} + \frac{L(\dot{x} + w_x)}{v} - mg\right), \\
\ddot{z} &= -\left(\frac{4}{m}\right)\left(-\frac{D(\dot{z} + w_z)}{v} - \frac{L(\dot{z} + w_z)}{v}\right)
\end{aligned} \tag{4.9}$$

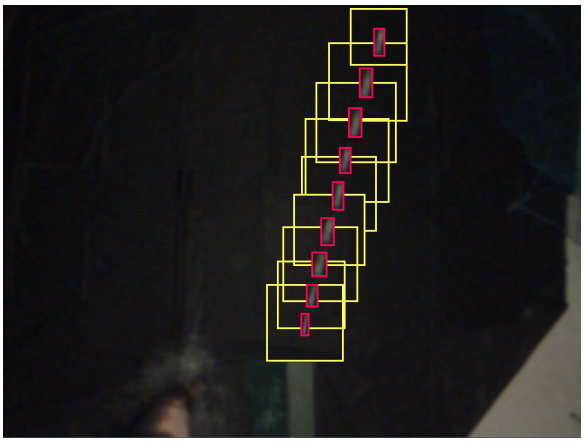
Pro ukázku zlepšení výsledků aproximace použitím nových rovnic viz obrázek B.1.

4.5 Zhodnocení dosažených výsledků

Při tvorbě prototypu golfového simulátoru byl kladen důraz na jeho uzpůsobení pro účel vyhledávání míčku ve videu. Záměrně byly stěny vyrobeny z látky černé barvy kvůli lepšímu kontrastu stopy od míčku. Z tohoto důvodu dochází při klasifikaci k nízkému počtu falešných detekcí tzv. *false alarm*.



(a) Přední kamera.



(b) Horní kamera.

Obrázek 4.3: Detekce míčku.

Klasifikátory jsem vytvořil tři. Nejlepších výsledků dosahují klasifikátory pro pohybující se míček. Díky vhodnému nastavení doby expozice bylo dosažno toho, že je stopa od míčku dobře čitelná a není zbytečně dlouhá. Klasifikátory proto zvládají nalézat správné stopy. V případě, že dojde k nalezení nesprávého objektu (*false alarm*), neprojde objekt většinou filtrem, který dohlíží na to, že má nalezený objekt velikost odpovídající míčku. Na obrázku 4.3 jsou žlutě označeny úspěšné detekce a růžově orámována stopa od míčku, která byla dohledána pomocí CvBlob[3].

Pro vyhodnocení přesnosti aproximace trajketorie míčku bylo třeba zvolit vhodný referenční nástroj. Snažil jsem se najít nějaký program, který by byl zdarma a měl dobré

[3] http://code.google.com/p/cvblob/

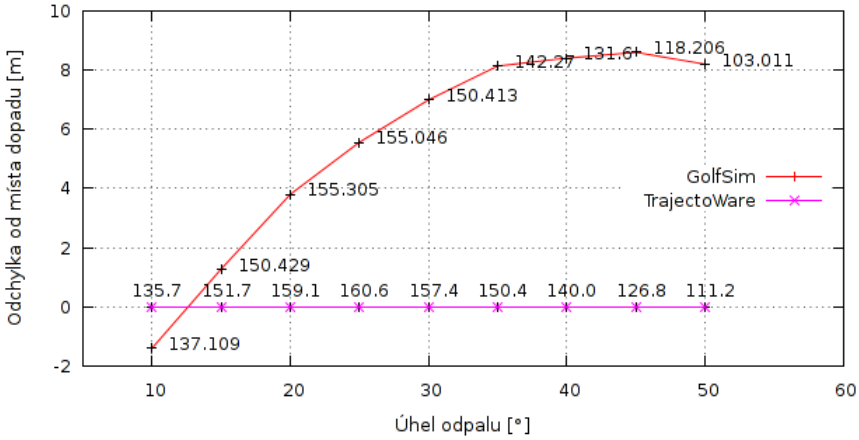
hodnocení. Nejlepších ohlasů[4][5] se dostávalo programu *TrajectoWare*[6], který se umisťoval na vysokých příčkách i mezi placenými programy.

Nyní na několika testovacích případech ukážu, jaké hodnoty jsem naměřil při testování fyzikální části aplikace. Čísla na křivkách ve všech grafech odpovídají vzdálenosti, do které míček doletěl.

Jako první jsem testoval přesnost výpočtu bodu dopadu míčku v závislosti na úhlu odpalu.

Parametry měření byly: rychlost 180 $km \cdot h^{-1}$ a 50 otáček míčku za sekundu ($3000rpm$) – tato hodnota je pro všechna měření stejná. Popisky u jednotlivých bodů na křivkách jsou délky letu míčku.

Na grafu 4.4 vidíme, že odchylka roste se zvyšujícím se úhlem až zhruba ke 45°, kdy začíná klesat. Maximální odchylka je přibližně 9 metrů. Takový výsledek lze považovat za příznivý, i když se to na první pohled nezdá. Musíme totiž vzít v úvahu fakt, že ideální úhel odpalu při reálné hře je 18°při použití driveru.



Obrázek 4.4: Odchylka při výpočtu trajktorie s počáteční rychlostí 180 $km \cdot h^{-1}$.

Driver je hůl, používaná většinou pro první a také nejdelší ránu. Odpaly ostatními holemi mají větší úhel a nižší rychlost. Z tohoto důvodu existuje nízká pravděpodobnost, že se skutečný odpal dostane do tohoto spektra.

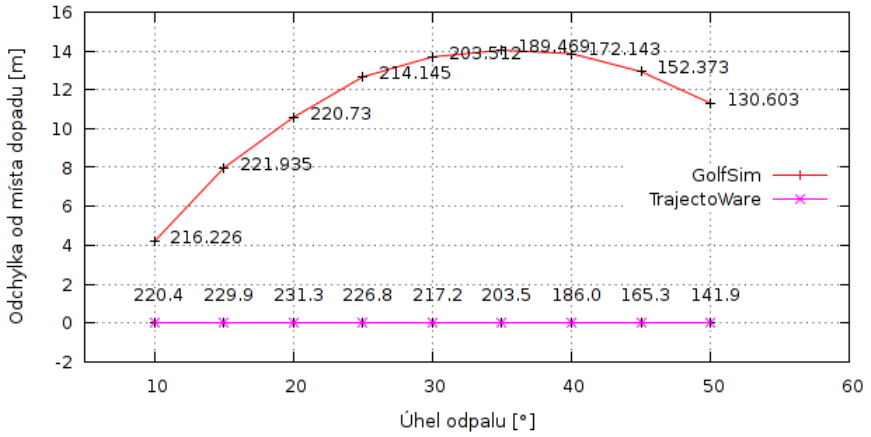
Na následujícím grafu 4.5 uvádím podobný případ jako výše, s tím rozdílem, že jsem zvolil počáteční rychlost 240 $km \cdot h^{-1}$.

Zde už je chyba vyšší, ale pro driver, kterým se odpaluje pod 18°je stále přijatelná – do 10 m.

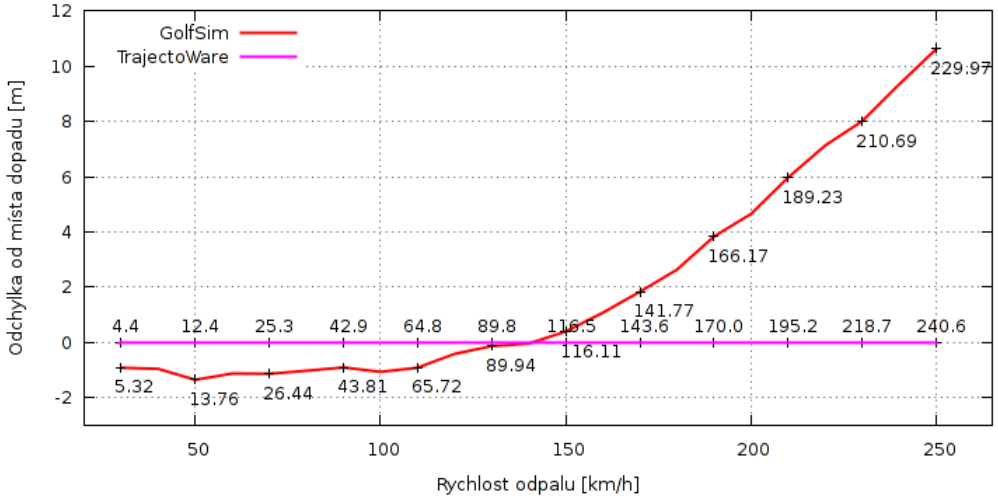
Dalším testem jsem se pokoušel zjistit, jaký vliv má rychlost odpalu na přesnost odhadu trajektorie. Zjistil jsem, že odchylka je přibližně konstantní pro rychlosti menší než 65 $km \cdot h^{-1}$. Potom dochází k růstu.

Při vyhodnocování je třeba si uvědomit, že aplikace je určena do indooru, kde se člověku špatně odhaduje síla švihu. Je rozdíl před sebou vidět projekční plátno a nebo stát na greenu a vidět praporek od jamky na vlastní oči. Aplikaci by bylo vhodné otestovat přímo v terénu

[4]http://www.tutelman.com/golf/design/biblio.php?ref=#trajectory
[5]http://probablegolfinstruction.com/golf-ball-trajectory-software.htm
[6]http://www.trajectoware.com/



Obrázek 4.5: Odchylka při výpočtu trajktorie s počáteční rychlostí 240 $km \cdot h^{-1}$.



Obrázek 4.6: Vliv počáteční rychlosti na přesnost při úhlu odpalu 18°.

na golfovém hřišti a zároveň provést měření, díky kterému by bylo možné upravit koeficienty rovnic.

4.6 Možnosti rozšíření

Možnosti rozšíření se rozrůstají především s výkonnějšími kamerami. Pro vývoj byly k dispozici kamery, které zvládaly 125 snímků za sekundu při nízkém rozlišení 320x240 pixelů. Při takovém rozlišení nejde rozpoznat například rotace míčku. Předpokládám, že pro tento účel by byly vhodné kamery s rozlišením alespoň 1024x768 pixelů a rychlostí snímání okolo 250 snímků za sekundu. Kromě toho, že by bylo možné zachytit rotaci míčku, došlo by i ke snížení chyby hlavně v oblasti vyhledávání bodů v obraze a jejich převodu do 3D. Alternativou pro určování rotace by bylo použití další kamery, která by dokázala přesně zjistit úhel kontaktu hole s míčkem.

Aplikace nedokáže zatím spolehlivě určit moment, kdy došlo k odpalu míčku. Z tohoto důvodu by bylo vhodné detekovat odpal buď pomocí čidla v podložce, nebo použitím

počítačového vidění pro rozpoznání pohybu hole.

Co se týče fyzikální části, bylo by vhodné provést testování ve skutečných podmínkách například na golfovém hřišti a ověřit tak přesnost vytvořené aplikace. Dle naměřených výsledků by se dala provést například úprava rovnic.

Zatím není implementován odraz míčku po jeho dopadu a případné následující dokutálení. V tomto případě bude nutné vzít v potaz také různorodost povrchu (rychlost greenu, voda, písek, raw) a podle toho upravit program.

Dalším vhodným rozšířením je vytvoření uživatelského rozhraní. Informace o stavu letu budou zobrazovány na projekčním plátně před hráčem spolu s 3D mapou hřiště.

Kapitola 5

Závěr

Cílem této práce bylo simulovat let golfového míčku na základě videozáznamu pořízeného během jeho odpalu. Pro tento účel bylo použito počítačového vidění. V kapitole 2 byla shrnuta teorie o počítačovém vidění a představeny prostředky a přístupy, které jsou v počítačovém vidění běžně používány. Byly zde uvedeny nástroje a algoritmy, které slouží pro detekci objektů ve videu.

V kapitole 3 jsem se zabýval fyzikálními vlivy, které provázejí golfový míček během jeho letu. Pro dopočítání trajektorie jsem se rozhodl použít balistickou křivku, pro kterou jsem zvolil vhodné parametry odpovídající fyzikálním vlastnostem míčku. Například se jednalo o volbu součinitele odporu, který je u míčku výrazně ovlivněn důlky na jeho povrchu. Díky nim je odpor vzduchu menší a míček má ve spojení s rotací, která vzniká díky sklonu hole při odpalu, vyšší tendenci stoupat vzhůru. Tato část nicméně nabízí prostor pro zlepšení, protože díky nízkému rozlišení kamer nebylo možné zjistit s jakou rotací míček vzlétá – tudíž nemohla být započítána. Zadání práce ukládalo, abych se zabýval i deformací míčku. Zjistil jsem však, že deformace nemá na trajektorii vliv anžto k ní dochází pouze při kontaktu hole s míčkem a ten se v okamžiku vrací do své původní podoby. Deformace míčku má vliv pouze na jeho rychlost; resp. má na ni vliv jeho pružnost, díky které dosahuje vyšší rychlosti, než je rychlost hole v okamžiku odpalu. Proto, i když připustíme minimální vliv, v konečném zúčtování se deformace neprojeví a ani literatura se jí nezabývá ([13], [3], [26]).

Kapitola 4 popisuje praktickou část práce. Uvedl jsem zde, jak vypadá prototyp simulátoru, a jak jsem řešil vyhledávání míčku ve videu. Vzhledem k tomu, že byla použita dvojice kamer, vytvořil jsem pro každou z nich pomocí knihovny OpenCV sadu klasifikátorů, pomocí kterých se snažím detekovat pohybující se míček. V kapitole jsem vyhodnotil úspěšnost klasifikátorů a navrhnul některá rozšíření. Rozšíření jsou svázána hlavně s pořízením výkonnějších kamer s vyšším rozlišením a vyšší rychlostí snímání.

Dále by bylo vhodné doplnit třetí kameru, která by snímala detailně kontakt hole s míčkem, aby bylo možno určit rotaci míčku. Nicméně v případě dostatečně vysokého rozlišení jedné z kamer by třetí kamera byla zbytečná a rotace by se dala odečíst přímo z pohybujícího se vhodně označeného míčku (vhodný potisk).

V demonstrační aplikaci jsem využil modulu pro získávání reálné polohy míčku v 3D prostoru na základě detekovaných stop od míčku v obraze. Poznatky uvedené v této práci budou prakticky využity v golfovém simulátoru společnosti Inven Solution s.r.o.

Literatura

[1] Aerospaceweb: Golf Ball Dimples & Drag. [Online; cit. 15. května 2011].
URL http://www.aerospaceweb.org/question/aerodynamics/q0215.shtml

[2] Balistika: Balistika. [Online; cit. 19. května 2011].
URL http://moon.felk.cvut.cz/~pjv/Jak/_phys/f598/start.html

[3] Barber, J.: Golf Ball Flight Dynamics. *A & EP 434 Final Project*, neuvedeno.

[4] Bradski, G.; Kaehler, A.: *Learning OpenCV*. O'Reilly Media Inc., 2008, ISBN 978-0-596-51613-0.
URL http://oreilly.com/catalog/9780596516130

[5] Dostál, R.: *Pozice objektu ze soustavy kamer*. Diplomová práce, FIT VUT v Brně, 2011.

[6] Freund, Y.; Schapire, R. E.: A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, ročník 55(1), 1997: s. 119 – 139.

[7] Halliday, D.; Resnick, R.; Walker, J.: *Fyzika - Mechanika*. VUTIUM, 2000, ISBN 80-214-1868-0.

[8] Juránek, R.: *Rozpoznávání vzorů v obraze pomocí klasifikátorů*. Diplomová práce, FIT VUT v Brně, 2007.

[9] Kolektiv autorů: *Pravidla českého pravopisu*. Academia, 2005, ISBN 80-200-1327-X.

[10] Lienhart, R.; Kuranov, A.; Pisarevsky, V.: Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection. *Intel technical report*, 2002.

[11] McShane, E. J.; Kelley, J. L.; Reno, F. V.: *Exterior Ballistics*. The University of Denver Press, 1953, 834 s.

[12] Meduna, A.: *Elements of Compiler Design*. Taylor and Francis, Taylor & Francis Informa plc, 2008, ISBN 978-1-4200-6323-3, 304 s.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=8538

[13] Mehta, R. D.; Pallis, J. M.: Sports Ball Aerodynamics: Effects of Velocity, Spin and Surface Roughness. *Materials and Science in Sports*, 2001: s. 185 – 197.

[14] Seo, N.: Merge vec files created by createsamples. [Online; cit. 15. května 2011].
URL http://note.sonots.com/SciSoftware/haartraining/mergevec.cpp.html

[15] Seo, N.: Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features. [Online; cit. 15. května 2011].
URL https://docs.google.com/View?docID=drw35kw_6gr8r84fs

[16] Seo, N.: Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features). [Online; cit. 15. května 2011].
URL http://note.sonots.com/SciSoftware/haartraining.html

[17] Shapiro, L. G.; Stockman, G. C.: *Computer Vision*. Prentice Hall, 2001, ISBN 0-13-030-796-3.

[18] Sonka, M.; Hlaváč, V.; Boyle, R.: *Image Processing, Analysis, and Machine Vision - 3rd Ed.* Thomson Engineering, 2007, ISBN 0-495-08252-X.

[19] Viola, P.; Jones, M.: Robust Real-time Object Detection. *Accepted conference on computer vision and pattern recognition*, 2001.

[20] Wikipedia: Golf Ball — Wikipedia. [Online; cit. 6. května 2011].
URL http://en.wikipedia.org/w/index.php?oldid=429652743

[21] Wikipedia: Magnusův jev — Wikipedia. [Online; cit. 6. května 2011].
URL http://cs.wikipedia.org/w/index.php?oldid=5906312

[22] Wikipedia: Playstation Eye — Wikipedia. [Online; cit. 2. května 2011].
URL http://en.wikipedia.org/w/index.php?oldid=430212637

[23] Wikipedia: Součinitel odporu — Wikipedia. [Online; cit. 6. května 2011].
URL http://en.wikipedia.org/w/index.php?oldid=430813128

[24] Wikipedia: Summed area table — Wikipedia. [Online; cit. 17. května 2011].
URL http://en.wikipedia.org/w/index.php?oldid=428419976

[25] Wikipedia: Chain code — Wikipedia. 2010, [Online; cit. 20. května 2011].
URL http://en.wikipedia.org/w/index.php?oldid=387734000

[26] Wong, J.-D.: Golf ball simulation. *neuveden*, 2010, [Online; cit. 17. května 2011].
URL http://www.cse.buffalo.edu/~wong4/projects/MTH337/Matlab/proj2.pdf

[27] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2005, ISBN 80-251-0454-0.

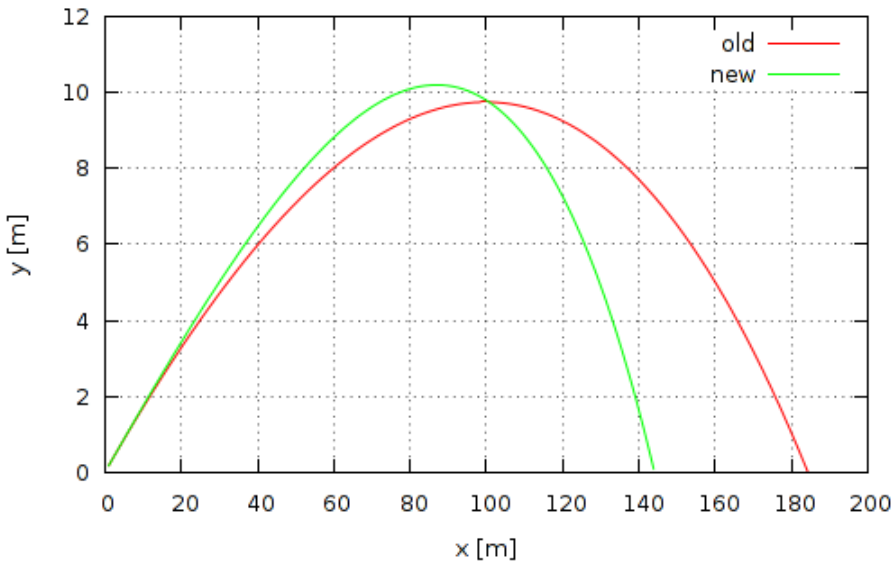
Příloha A

Obsah CD

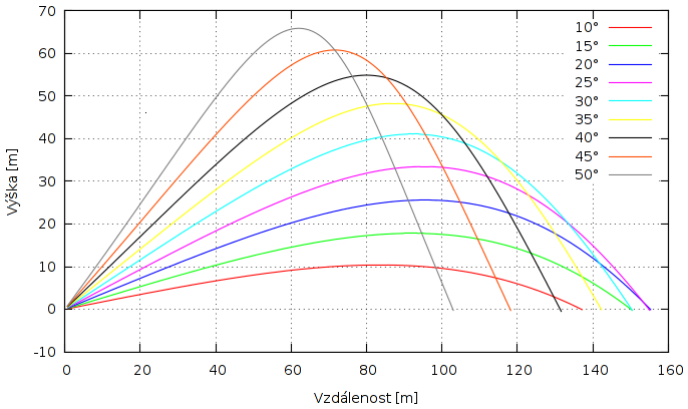
- Zdrojové kódy diplomové práce (LaTeX)
- Dokument pdf s diplomovou prací
- Zdrojové kódy aplikace
- Demonstrační video pro aplikaci

Příloha B

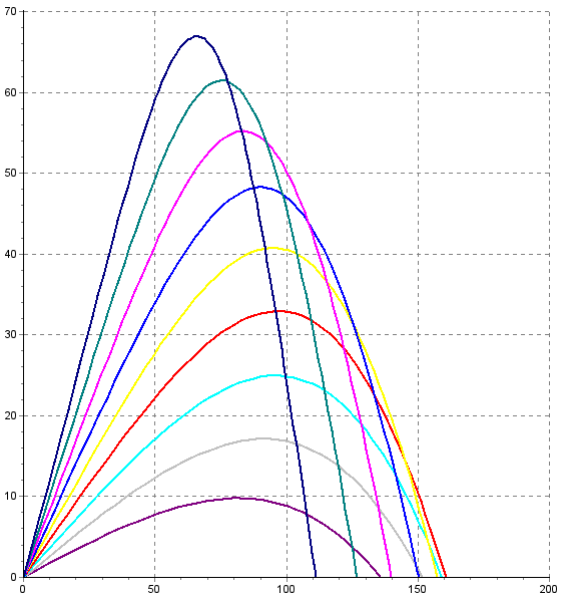
Grafy



Obrázek B.1: Srovnání aproximace trajektorie pomocí původních rovnic 3.15 (*old*) a přesnějších rovnic 4.9 (*new*).

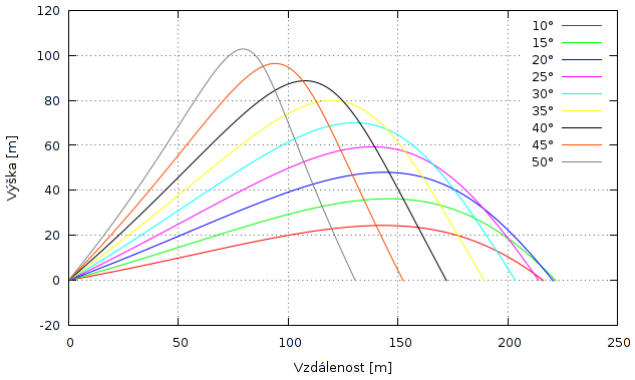


(a) GolfSim

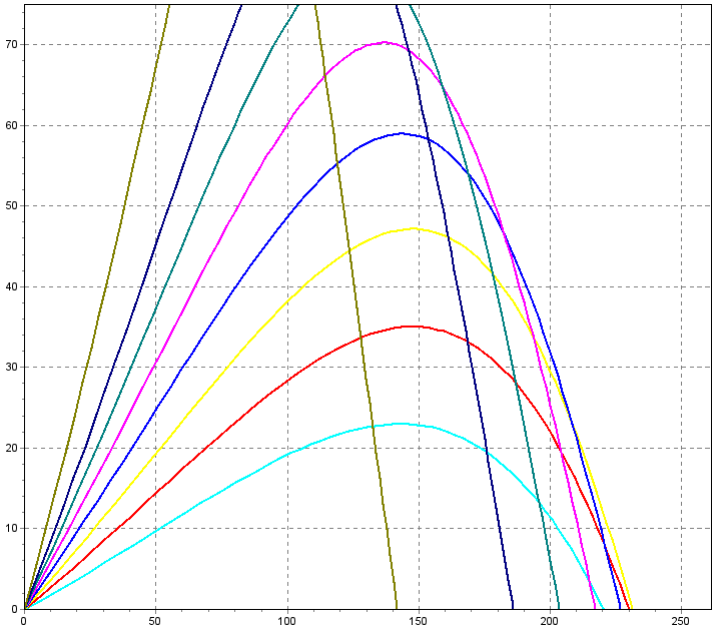


(b) TrajectoWare

Obrázek B.2: Srovnání GolfSim vs. TrajectoWare – trajektorie letu při různých úhlech odpalu a rychlosti 180 $km \cdot h^{-1}$.

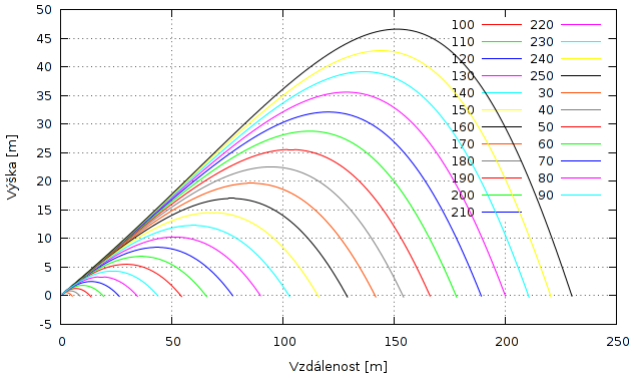


(a) GolfSim

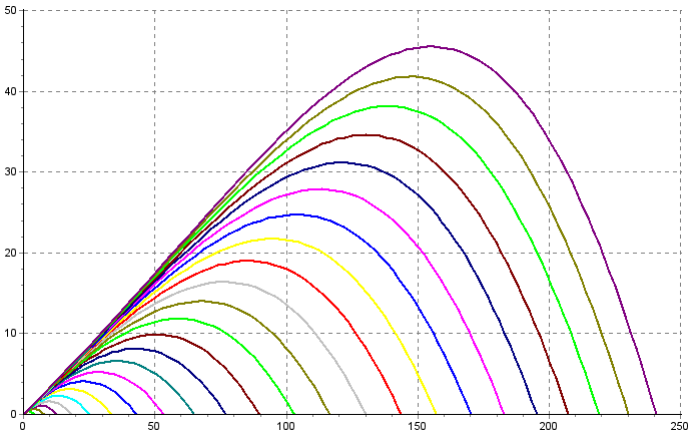


(b) TrajectoWare

Obrázek B.3: Srovnání GolfSim vs. TrajectoWare – trajektorie letu při různých úhlech odpalu a rychlosti 240 $km \cdot h^{-1}$.



(a) GolfSim



(b) TrajectoWare

Obrázek B.4: Srovnání GolfSim vs. TrajectoWare – trajektorie letu při různých rychlostech a úhlu odpalu 18°.