Gymnázium Jana Keplera
Parléřova 2
169 00 Praha 6

Maturitní práce

Lightbeats

Jiří Balhar

Vedoucí práce: Pavel Zbytovský

Předmět Informatika
březen 2015

Anotace

Lightbeats je software pro tracking světelných, žonglovacích míčků. Z obrazu webkamery se získávají pozice, velikosti a barvy míčků a tyto informace jsou pak uchovávány pro další využití, jako je například vizualizace trajektorie.

Zadání

Cílem práce je vytvořit program pro spolehlivé trackování žonglovacích míčků. Reálné míčky by měly mít svůj virtuální „ekvivalent“, který bude uchovávat historii jeho polohy, velikosti a barvy. Tyto informace lze využít k vizualizaci trajektorie jednotlivých míčků.

Prohlášení

Prohlašuji, že jsem jediným autorem této maturitní práce a všechny citace, použitá literatura a další zdroje jsou v práci uvedené.

Tímto dle zákona 121/2000 Sb. ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium Jana Keplera, Praha 6, Parléřova 2 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne 9. března 2015 .....

Dokumentace

Uživatelská část

Instalace

Pro spuštění Lightbeats je potřeba nainstalované prostředí Processing[1]. Projekt také využívá knihovny JMyron[2], ControlP5[3] a CL-Eye[4]. Po instalaci prostředí stačí zdrojový kód otevřít v Processingu, případně ho zkompilovat a spustit ručně pomocí programu `processing-java`. Knihovna CL-Eye je komerční, pro chod programu však není nezbytná (zajišťuje pouze přístup k vysokorychlostní webkameře), pokud tedy licenci pro tuto knihovnu zakoupenou nemáte, nainstalujte[5] náhradní knihovnu ve složce `libraries/cleyemulticam`.

Použití

Lightbeats má dva základní pohledy. Debug mód, který je přizpůsobený pro nastavení kamery a ladění parametrů pro hledání míčků, a prezentační mód, ve kterém se trackované míčky vizualizují. Ve výchozím nastavení je po spuštění programu zapnut debug mód. Do prezentačního módu lze přepnout stisknutím klávesy `D`.

Aplikace se vypíná klávesou `Esc` (*Upozornění: nastavení se při vypínání automaticky neukládají*). Po stisknutí mezerníku aplikace zapíše aktuální snímek obrazovky do složky `screenshots` ve formátu png, klávesa `C` zapíná/vypíná nahrávání obrazu do složky `frames`, video je ukládáno do jednotlivých souborů ve formátu TGA. Nahrávání obrazu může zpomalit běh programu.

Nastavení

Nastavit aplikaci je možné pouze při zapnutém debug módu. V levém horním rohu obrazovky se nachází tlačítka pro přepínání jednotlivých záložek nastavení. Nastavení je roztříděno do kategorií *camera settings*, *ball detection*, *circle finder, other*. Pro základní nastavení aplikace stačí měnit nastavení v kategorii *camera settngs*, případně v *other*. Pod přepínačem záložek se nachází také tlačítko pro uložení aktuálního nastavení.

`debugView` – tento přepínač mění aktivní zobrazení v debug módu. Při hodnotě `0` se zobrazuje nezměněný obraz z webkamery s přehledem identifikovaných `Ball`. Každý `Ball` ohraničuje bílá elipsa, uprostřed elipsy leží unikátní id a nad ní hodnoty podobnosti. Fialová elipsa ukazuje `State`, který byl vypočítán minulý snímek jako predikce. Tudíž lze porovnávat podobnost dané predikce s realitou. Při hodnotě 1 se vykresluje obraz z webkamery po aplikaci *threshold filtru* a vyhledání *Globů* (ty jsou ohraničeny červenými obdélníky). Toto zobrazení je v kombinaci s `debugView=0` vhodné ke kalibraci webkamery. Při hodnotě 2 se vykreslují informace z *circle finderu*. Obdélníky ohraničují oblasti, na kterých se hledaly kruhy. Nad jednotlivými obdélníky je uvedena informace, kolik kruhů se v oblasti hledalo. Červený histogram zobrazuje pravděpodobnost výskytu středu

1 processing.org
2 webcamxtra.sourceforge.net
3 www.sojamo.de/libraries/controlP5
4 codelaboratories.com
5 https://github.com/processing/processing/wiki/How-to-Install-a-Contributed-Library

kružnice (čím výraznější červená barva, tím pravděpodobnější je umístění středu). Zelené kružnice ohraničují nalezené kruhy v obrázku. Červené kružnice ohraničují nalezené kruhy, které byly posléze vyřazeny (na základě hodnoty `minFoundCircleRatio`). Odpovídající barvou jsou pod kružnicemi napsány pravděpodobnosti, podle kterých *finder* kružnice filtruje.

Pokročilé nastavení

Pro lepší pochopení následujících nastavení je nutné míti povědomí o základním algoritmu. Viz programátorská část dokumentace.

Ball detection

`ballStateCount` – Počet uložených *State* v historii *Ball*.

`avgStateCount` – Počet *State*, ze kterých se počítá průměrný *State* pro každý *Ball*. Pokud je `avgStateCount` větší než `ballStateCount`, pak se průměr počítá ze všech uložených *State*.

`existingBallThreshold` – Práh pravděpodobnosti, nad kterou je daný *State* považovaný jako patřící k danému *Ball*.

`newBallThreshold` – Práh pravděpodobnosti, pod kterou je daný *State* považovaný za nepatřící k žádnému uloženému *Ball*. Tudíž se *State* uloží do nové instance *Ball*.

`colorWeight` – Váha rozdílu barvy daného *State* a dané průměrné barvy *Ball* na celkovou pravděpodobnost. Vliv tohoto nastavení můžeme snadno zkontrolovat v `debugView=0` nad jednotlivými *Ball*, jakožto 1. složku celkové pravděpodobnosti.

`positionWeight` – Váha rozdílu pozice daného *State* a poslední známé pozice *Ball* na celkovou pravděpodobnost. Vliv tohoto nastavení můžeme snadno zkontrolovat v `debugView=0` nad jednotlivými *Ball*, jakožto 2. složku (1. ve funkci `max()`) celkové pravděpodobnosti.

`predictedPositionWeight` – Váha rozdílu pozice daného *State* a predikované pozice *Ball* na celkovou pravděpodobnost. Vliv tohoto nastavení můžeme snadno zkontrolovat v `debugView=0` nad jednotlivými *Ball*, jakožto 2. složku (2. ve funkci `max()`) celkové pravděpodobnosti.

`sizeWeight` – Váha rozdílu velikosti daného *State* a průměrné velikosti *Ball* na celkovou pravděpodobnost. Vliv tohoto nastavení můžeme snadno zkontrolovat v `debugView=0` nad jednotlivými *Ball*, jakožto 3. složku celkové pravděpodobnosti.

`dColorMax` – Maximální hranice rozdílu barev (součet rozdílů složek rgb), po které je již příspěvek k celkové pravděpodobnosti nulový.

`dPositionMax` – Maximální hranice rozdílu pozic, po které je již příspěvek k celkové pravděpodobnosti nulový.

`dPredictedPositionMax` – Maximální hranice rozdílu pozice *State* a predikované pozice *Ball*, po které je již příspěvek k celkové pravděpodobnosti nulový.

`dSizeMax` – Maximální hranice rozdílu velikostí, po které je již příspěvek k celkové pravděpodobnosti nulový.

Poznámka: dColorMax, dPositionMax, dPredictedPositionMax a dSizeMax jsou změny za 1

milisekundu, nikoli za celý snímek, aby se zajistila nezávislost na rychlosti snímání kamerou.

Circle finder

`finderThreshold` – Práh pravděpodobnosti, nad kterou je považován daný *Ball* jako vhodný kandidát pro zkoumaný *State*. Počet kandidátů je použit pro algoritmus hledání kruhů.

`firstCircleThreshold` – Práh pravděpodobnosti kruhu, nad kterým je nalezený kruh přidán zpět mezi procházené *States*.

`minNextCircleThresholdRatio` – Poměr pravděpodobností nově nalezeného kruhu a minulého nalezeného kruhu, nad kterým je nalezený kruh přidán zpět mezi procházené *States*.

`minPointCount` – Minimální počet pixelů na hranici *Globu*, pod který již algoritmus pro hledání kruhů nepokračuje v hledání.

Other

`maxPredictedStates` – Maximální počet predikovaných stavů *Ball*, které jsou při nedostupnosti reálných *State* (např. při chvilkovém zakrytí letícího míčku) přidány do historie *Ball*. Po dosažení hranice *Ball* „stojí“ na místě.

`ballProbabilityThreshold` – Druh filtrování stacionárních světelných zdrojů, které by se neměly považovat za míčky (například světlé plochy v pozadí webkamery). Každý *Ball* musí po určitou dobu udržet rychlost nad stanovenou mezí, pokud ji nepřekročí, není zobrazen v prezentačním módu. Tato proměnná určuje tuto dobu. Hodnoty pod 0,5 vypínají filtrování.

`ballProbabilitySpeed` – Rychlost, kterou musí *Ball* udržet, aby nebyl považován za šum.

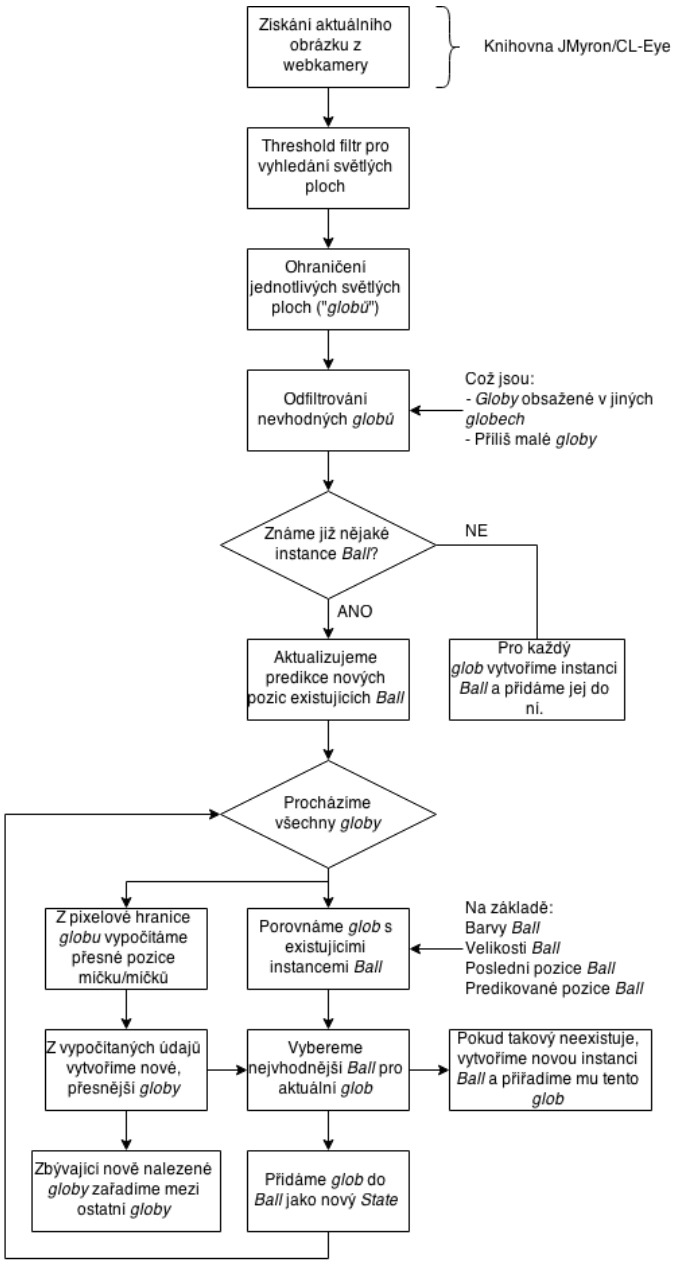
Klávesové zkratky

- `Esc`: Ukončení programu
- `D`: Přepnutí debug módu
- `C`: Zapnutí/Vypnutí nahrávání obrazu
- `Mezerník`: Uložení aktuálního snímku
- `A`: Adaptace obrazu

Programátorská část

Základní popis

Obraz z webkamery je zpracován threshold filtrem pro nalezení světlých oblastí. Program tyto spojité plochy identifikuje a nalezne jejich okraje (= pole pixelů, které plochu ohraničují), nejmenší ohraničující obdélníky (tzv. bounding box) a jejich těžiště. Tyto informace pro jednotlivé plochy nazýváme *globy*.



Globy představují světlá místa na obrázku, triviálním řešením by tedy bylo *globy* prohlásit za míčky, nicméně existují speciální případy, které nám toto zjednodušení nedovolují. Například situace, ve které jsou dva míčky blízko u sebe a společně se jeví jako jeden *glob*. Mimo jiné při tomto triviálním přístupu nerozlišujeme jednotlivé míčky, což nás zásadně omezuje při dalším využívání dat (například při vizualizaci).

Řešením je zavedení třídy *Ball*, jejíž instance reprezentují jednotlivé reálné míčky a uchovávají historii svých pozic. Jednak lze tato historie dobře využít při vizualizaci a jednak lze díky ní lépe třídit *globy* a řešit hraniční případy.

Program tedy prochází nalezené *globy* a třídí je do správných instancí *Ball*. Pokud *globu* neodpovídá žádná instance *Ball*, vytvoří se nová.

Pro větší robustnost je v programu implementován také algoritmus pro hledání kružnic podle okraje *globu*. Výhodou tohoto algoritmu je skutečnost, že nalezne více středů kružnic v hraničním případě překrývajících se míčků. Také správně nalezne střed kružnice, i když je míček částečně zakrytý rukou, případně když je míček na hranici obrazovky a část leží mimo zorný úhel.

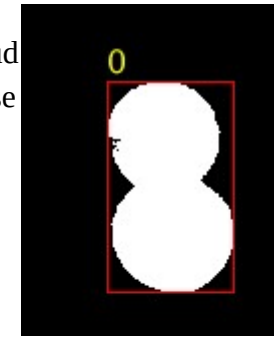
Podrobný popis

Hledání globů

Po získání pole pixelů z webkamery program tento obrázek zpracuje threshold filtrem a vytvoří `globImage`. Pro každý pixel se vypočítá rozdíl mezi jeho barvou a nastavenou barvou pozadí. Pokud je součet absolutních hodnot rozdílů RGB složek barev vyšší než poskytnutá hodnota

`threshold`, pak se do `globImage` zapíše jako bílá barva, v opačném případě jako černá (`myron.pde`, metoda `thresholdFilter()`).

`globImage` se poté projde po pixelech a pro každý bílý pixel se spustí algoritmus pro hledání ploch (`processGlobs.pde`, metoda `fillGlob()`). Pokud má pixel bílého souseda, tento soused se započítá do plochy (=*globu*) a spustí se pro něj stejný algoritmus. Z důvodu nebezpečí přetečení zásobníku je tento algoritmus implementován bez rekurze, přestože se použití rekurze velmi vybízí.(`myron.pde`, metoda `fillGlob()`) Ukládáním pixelů, které nemají všechny bílé sousedy, vzniká pixelová hranice *globu*.



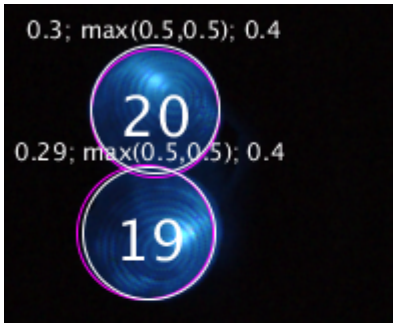
globImage: červený obdélník ohraničuje nalezený glob

Poznámka: Kód hledání globů byl silně inspirován knihovnou JMyron.[6]

Třídění globů

Na začátku třídění aktualizuji predikce nových pozic všech instancí *Ball* (`ball.pde`, metoda `updatePrediction()`). Predikce probíhá až na začátku nového třídění, jelikož až nyní víme, kolik milisekund uběhlo od posledního třídění (predikce počítá s rychlostí a zrychlením, čas je proto důležitým parametrem).

Každý nalezený *glob*, pro jednodušší manipulaci ukládám jako instanci třídy *State*. Při „konverzi“ filtruji *globy*, které nepovažuji za relevantní. Jsou to ty, které jsou umístěné uvnitř jiných *globů*.



Na obrázku vidíme dvě instance Ball spolu s jejich ID (19, 20), složkami pravděpodobnosti (zleva: podobnost barvy (max 0.3), pozice (max 0.5), predikovaná pozice (0.5), velikost (0.4)) a s vizualizací predikce pozice (fialová elipsa je predikovaná)

Po filtru se prochází jednotlivé *globy* (=*States*). Ke každému *State* se projdou evidované instance *Ball* a vypočítá se pravděpodobnost (`ball.pde`, metoda `getProbability()`), že daný *State* k danému *Ball* patří (neboli že daná světelná plocha na obrázku je opravdu dalším „krokem“ daného zachycovaného míčku).

Po vypočítání pravděpodobností se sečte počet *Ball*, ke kterým by mohl *State* patřit a tato informace se využije pro pokročilé hledání kružnic podle okrajů (viz Algoritmus hledání kružnic).

Algoritmus pro hledání okrajů jako výstup nabídne potenciálně přesnější údaje o pozici míčku, tyto údaje je ale nutné brát se stejně zdravým skepticismem, jako údaje *globů*, protože mohou být nalezené kružnice ovlivněné šumem webkamery (což u těžiště tolik nehrozí), případně nedostatečně přesným ohraničením míčku. Naopak lepší výsledky nabízí v již zmiňovaných hraničních případech. Rozhodnutí program dělá znovu na základě vypočítané `probability` (`balls.pde`, řádek).

Pro *Ball*, ke kterým nebyl přiřazen žádný *State*, je zavolána metoda `predict` (`ball.pde`, metoda `predict()`), která jako nový *State* přidá predikovaný *State.* Po určeném počtu predikcí se stavy přestanou přidávat a *Ball* postupně mizí s ubývajícím celkovým počtem uložených stavů.

Pro každý *Ball* se také na konci třídění *globů*, počítá pravděpodobnost, se kterou daný *Ball* opravdu

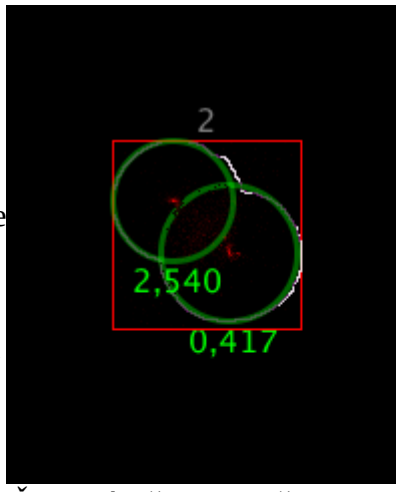
6 https://github.com/jtnimoy/myron/

reprezentuje míček a nikoli náhodný šum či světlé místo v pozadí. Pokud je *Ball* již jednou označen za míček, označení se již nezpochybňuje. Pravděpodobnost se počítá podle pohybu míčku, předpokládáme, že světlé místo v pozadí se nepohybuje.

Algoritmus hledání kružnic[7]

Pro implementaci jsem se inspiroval citovanou prací „Real-time Color Ball Tracking for Augmented Reality“. Pro svoje účely jsem algoritmus ale zjednodušil a upravil, aby podával stabilnější výsledky.

Algoritmus funguje na principu hlasování. Opakovaně se vybírají tři body na pixelovém okraji *globu* a počítá se střed vzniklé kružnice. Středy kružnic se zaznamenávají do 2D histogramu, ze kterého se posléze vybírá globální maximum (tedy vítězí střed s největším počtem hlasů). Práce D. Sýkory, D. Sedláčka a K. Riegera doporučuje obdobný postup i pro hledání poloměru. Pro každý bod na okraji *globu* spočítáme poloměr a vybereme ten s největším zastoupením. Tento postup ale podával výsledek, který byl velmi ovlivněný šumem a poloměr byl v rámci jednotlivých snímků poměrně variabilní. Mé zlepšení spočívá v použití váženého průměru nalezených poloměrů, který situaci značně stabilizuje.



Červené tečky uprostřed nalezených (zelených) kružnic zobrazují počet "hlasů" pro daný střed.

Pokud v obrysu *globu* hledáme více kružnic (pro *glob* jsme našli více kandidátů *Ball*), z obrysu odstraníme body, pro které jsme kružnici již nalezli a algoritmus na zbytku bodů spustíme znovu.

Slepé cesty

Pravděpodobně nejvíce vývojářského času jsem strávil nad řešením problému nalezení kružnic ve světelných plochách. Robustní řešení tohoto problému totiž znamená ošetření případů jako:

- Míček je viditelný pouze částečně.
- Míček je rozdělený do více světelných ploch. (Časté například při držení v ruce)
- Dva míčky (či více) jsou interpretovány jako jedna spojitá světelná plocha.
- Světelná plocha v pozadí je interpretována jako míček, přestože má jiný tvar.

I přes velikou motivaci se mi nepodařilo nalézt obecné řešení, které by řešilo všechny zmíněné problémy, nejúspěšnějším byl výše popisovaný algoritmus využívající hlasování o středu kružnic.

Testovaným algoritmem byla Houghova transformace pro hledání kružnic, která však podávala silně nestabilní výsledky (střed nalezené kružnice fluktuoval kolem reálného středu) i při testování ve velmi dobrých podmínkách, při testování na částečně zakrytých míčcích algoritmus selhával. Použil jsem implementaci z knihovny OpenCV pro zpracování obrazu, tudíž je chyba v implementaci téměř vyloučena, nevím proto, proč byl tento pokus neúspěšný. Pro hledání hran jsem

7 Sýkora, D., Sedláček, D., Riege, K.: Real-time Color Ball Tracking for Augmented Reality, Mohler, B., van Liere, R. (ed.), 2008, strana 4-5

experimentoval s dalšími algoritmy krom threshold filtru, nicméně ani to při Houghově transformaci nepomohlo. Mimo to byl při použití Gaussova rozostření obrazu a Cannyho hranového detektoru algoritmus také v porovnání se stávajícím výrazně pomalejší.

Na hledání kružnice v pixelových hranicích světelných ploch lze pohlížet i jako na optimalizační problém:

$$arg\,min \sum_{k=0}^{n} \left(\sqrt{(x_k - x)^2 + (y_k - y)^2} - r\right)^2$$

Při argumentech x,y (souřadnice) a r (poloměr hledané kružnice). x_k a y_k jsou hraniční pixely.

Pro řešení byl implementován algoritmus SOMA[8], výsledky byly po dlouhém ladění parametrů a optimalizační funkce uspokojivé – program správně našel středy i v nepřehledné situaci[9]. Také správně našel více středů při testování s více míčky. Poměrně dobré výsledky však byly vykoupeny velikou výpočetní složitostí, hledací algoritmus celý program velmi zpomaloval. Pokoušel jsem se SOMA algoritmus zrychlit volbou jiných parametrů, které jsou na výkon šetrnější (například zmenšení populace řešení), nicméně podávané výsledky pak nebyly stabilní.

Ukázka programu v prezentačním módu

8 Zelinka, I: Umělá inteligence v problémech globální optimalizace, Praha, BEN – technická literatura, 2002

9 https://github.com/kukas/Lightbeats/blob/787e5afe5b665cac15c965234fee24e69e4a39f2/finder.pde

Závěr a hodnocení

Program Lightbeats byl již v praxi využit při představení v rámci Žižkovské noci, bohužel kvůli technickým problémům s driverem k webkameře se tracking uplatnil pouze částečně. Další reálné nasazení programu však očekáváme brzy, jakmile se naskytne vhodná příležitost. Kód reálného nasazení Lightbeats lze prohlédnout zde: https://github.com/kukas/Lightbeats/tree/predstaveni-muybridge

Dalším využitím bude zakomponování trackingu do divadelního představení v rámci závěrečné zkoušky Vojtěcha Žáka (který koncept Lightbeats vymysel) na FAMU. Jeho projekt je již schválen a získal dotaci 15 000Kč na realizaci.

Projekt má i oficiální web www.lightbeats.cz, na kterém lze vidět původní koncept programu na videu (video z programu přidáme brzy), případně se dozvědět o nových představeních.

Co se týče výkonu programu, jsem s ním poměrně spokojen. Největším úspěchem pro mě je skutečnost, že na mém 5 let starém notebooku běží program až na plných 120 fps (reálně se využívá mód 75 fps z důvodů vyššího rozlišení obrazu). Díky tomu lze především využívat výhody vysokorychlostní webkamery, krom toho ale také z výkonové nenáročnosti programu těží i samotná vizualizace, která se netrhá a je plynulá.

Samotný tracking funguje dobře, ve složitějších situacích má tendenci míček občas ztratit, zatím jsme ale při testování nenarazili na situaci, ve které by tyto výpadky byly výrazněji vidět. V nejhorších případech míček na několik snímků „zmizí“ a objeví se o několik pixelů dál, to ale znamená pouze několik milisekund dlouhou prodlevu, která při představení nevadí. Výraznějším problémem je situace, kdy žonglování končí a více míčků skončí v jedné ruce. Některé míčky tak bývají kompletně skryty za jinými a není možnost, jak jejich pozici spolehlivě získat. Tento problém se ale zdá neřešitelný, jelikož v získaném obrazu z webkamery opravdu informace o „schovaném“ míčku není.

Lightbeats bylo naprogramováno v jazyce Processing, který je založený na Javě. Tato práce byla mým prvním setkáním s Processingem a prvním větším projektem v jazyce Java, myslím si ale, že kvalita kódu tímto nedostatkem zkušeností příliš netrpěla. Implementace ctí hlavní principy objektového programování, jako DRY, zapouzdření kódu nebo princip jedné odpovědnosti.

Osobně považuji svůj maturitní projekt za úspěšný, nešlo pouze o splnění požadavků maturitní zkoušky, nýbrž vznikl program, který má reálné využití a odevzdáním maturitní komisi jeho smysl nekončí. Po uzavření kódu pro maturitní projekt mě napadla další zlepšení (konkrétně při vytváření `globImage` a při filtrování stacionárních *Ball*), které sice již cestu do maturitní verze nenajdou z důvodu nedostatku času pro otestování, nicméně je jistě zakomponuji do další verze pro představení.