



# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA STROJNÍHO INŽENÝRSTVÍ**  
FACULTY OF MECHANICAL ENGINEERING

**ÚSTAV AUTOMATIZACE A INFORMATIKY**  
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

## VZDUŠNÝ HOKEJ - STROJOVÉ VIDĚNÍ A HERNÍ STRATEGIE

VZDUŠNÝ HOKEJ - MACHINE VISION AND GAME STRATEGY

**DIPLOMOVÁ PRÁCE**  
MASTER'S THESIS

**AUTOR PRÁCE** Bc. Ondřej Sláma  
AUTHOR

**VEDOUCÍ PRÁCE** doc. Ing. Radomil Matoušek, Ph.D.  
SUPERVISOR

BRNO 2020



# Zadání diplomové práce

Ústav: Ústav automatizace a informatiky  
Student: **Bc. Ondřej Sláma**  
Studijní program: Strojní inženýrství  
Studijní obor: Aplikovaná informatika a řízení  
Vedoucí práce: **doc. Ing. Radomil Matoušek, Ph.D.**  
Akademický rok: 2019/20

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

## **Vzdušný hokej – strojové vidění a herní strategie**

### **Stručná charakteristika problematiky úkolu:**

Vzdušný hokej nebo Air hockey je společenská hra, která se vyvinula ze stolního hokeje. Je určena pro dva hráče, cílem je dostat kotouč pomocí speciální „pálky“ do soupeřovy branek. Obsahem diplomové práce bude simulační realizace této hry s využitím umělé inteligence a návrh systému pro rozpoznání pozice puku.

### **Cíle diplomové práce:**

- Rešerše hry Air Hockey v kontextu návrhu automatického hráče.
- Návrh metodiky a praktická realizace strojového vidění pro detekci puku.
- Vytvoření počítačové simulace pro návrh řízení.
- Vytvoření počítačové simulace pro návrh strategie hry s využitím umělé inteligence.
- Zhodnocení výsledků a tvorba prezentace.

### **Seznam doporučené literatury:**

DENNIS, Andrew K. Raspberry Pi home automation with Arduino. Packt Publishing Ltd, 2013.

SPONG, Mark W. Impact controllability of an air hockey puck. Systems & Control Letters, 2001, 42.5: 333-345.

BISHOP, Bradley E.; SPONG, Mark W. Vision-based control of an air hockey playing robot. IEEE Control Systems Magazine, 1999, 19.3: 23-32.



<abstrakt>  
<abstrakt en>  
<klíčová slova>  
<klíčová slova en>

<citace>



<prohlášení>



<poděkování>



# Obsah

<b>1</b>	<b>Úvod</b>	<b>13</b>
<b>2</b>	<b>Vzdušný hokej</b>	<b>15</b>
2.1	Standardizace vzdušného hokeje . . . . .	15
2.1.1	Pravidla hry . . . . .	15
2.1.2	Rozměry . . . . .	16
2.1.3	Ostatní parametry . . . . .	16
2.2	Styl hry . . . . .	17
2.2.1	Obrana . . . . .	17
2.2.2	Útok . . . . .	18
2.2.3	Analýza lidského chování . . . . .	19
2.3	Matematický model hry . . . . .	21
2.3.1	Matematický popis puku . . . . .	21
2.3.2	Interakce puku s herní plochou . . . . .	22
2.3.3	Odraz puku od mantinelů . . . . .	22
2.3.4	Interakce puku s hokejkou . . . . .	23
<b>3</b>	<b>Detekce pozice objektů</b>	<b>25</b>
3.1	Kamerové systémy . . . . .	25
3.1.1	Algoritmy detekce . . . . .	25
3.1.2	Příklady hardwaru . . . . .	28
3.2	Ostatní přístupy . . . . .	31
<b>4</b>	<b>Konstrukce a hardware</b>	<b>33</b>
<b>5</b>	<b>Simulace hry</b>	<b>35</b>
5.1	Struktura programu . . . . .	36
5.2	Simulace fyziky . . . . .	38
5.2.1	Kolize s hokejkou . . . . .	38
5.2.2	Kolize s mantinemem . . . . .	39
5.3	Simalace kamery . . . . .	40
5.3.1	Náhodná odchylka . . . . .	40
5.3.2	Diskretizace polohy . . . . .	40
5.3.3	Zpoždění informace . . . . .	40
<b>6</b>	<b>Návrh strategie</b>	<b>41</b>
6.1	Struktura modulu strategie . . . . .	41
6.2	Základní třída <i>BaseStrategy</i> . . . . .	42
6.2.1	Zpracování polohy puku . . . . .	43
6.3	Navržené strategie . . . . .	45
6.3.1	Strategie A . . . . .	45
6.3.2	Strategie B . . . . .	46

6.3.3	Strategie C . . . . .	46
6.3.4	Strategie D . . . . .	47
<b>7</b>	<b>Detekce puku</b>	<b>49</b>
7.1	Snímání a detekce puku . . . . .	49
7.2	Automatické vyvážení bílé . . . . .	51
7.3	Automatické rozpoznání barvy puku . . . . .	52
<b>8</b>	<b>Softwarová implementace</b>	<b>53</b>
8.1	Konfigurace Raspberry Pi 4B . . . . .	53
8.2	Struktura programu . . . . .	54
8.3	Uživatelské rozhraní . . . . .	56
8.3.1	Struktura programu a základy <i>Kivy</i> . . . . .	56
8.3.2	Rozvržení a design . . . . .	57
8.4	Komunikace s řízením pohonů . . . . .	57
<b>9</b>	<b>Závěr</b>	<b>59</b>
<b>Seznam obrázků</b>		<b>61</b>
<b>Seznam tabulek</b>		<b>61</b>
<b>Seznam algoritmů</b>		<b>62</b>
<b>Použité zkratky</b>		<b>62</b>
<b>Seznam použité literatury</b>		<b>63</b>

# 1 Úvod

Oblíbená arkádová hra vzdušný hokej vznikla jako variace stolního hokeje již v roce 1972. Hra je určená pro dva hráče, kteří se pomocí speciálních hokejek snaží trefit puk, pohybující se po herní desce do soupeřovi brány. Díky ventilátorům tlačící vzduch z pod děrované desky se pod herním pukem vytváří vzduchový polštář a puk se tak pohybuje téměř bez tření. Napodobuje tím chování reálného puku na ledě. Volný pohyb hokejky po polovině herní plochy a schopnost jakoli odrážet puk od bočních mantinelů dává možnost vzniku mnoha rozdílných pohledů na strategii hry a docílit tak vstřelení gólu i pomocí nečekaných trajektorií.



## 2 Vzdušný hokej

V postupu návrhu automatického hráče robotického vzdušného hokeje (dále jen RVH) je důležité zakládat na již zavedených a standardizovaných vlastnostech klasického AirHockey (česky vzdušný hokej) zobrazeného na obrázku 1. Zejména pak z pohledu stylu hry je důležité, aby robotický hráč neporušoval pravidla a byl i tak schopný herně konkurovat lidskému protihráči.



Obrázek 1: Profesionální vzdušný hokej využívaný na turnajích [1]

### 2.1 Standardizace vzdušného hokeje

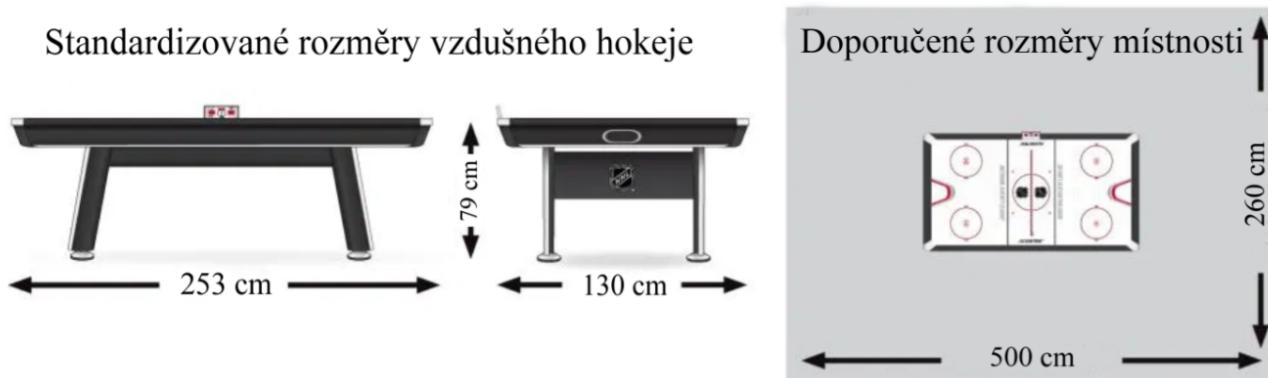
V dnešní době lze nejen koupit množství rozdílných formátů stolu vzdušného hokeje ale může se lišit i umístění branek, popis pravidel nebo styl hry. Proto byly pro turnajové účely tyto parametry standardizovány.

#### 2.1.1 Pravidla hry

Pokaždé hrají dva hráči proti sobě na hrací ploše s jedním pukem a dvěma hokejkami (viz obrázek 3). Tito hráči se mohou vyskytovat kdekoli kolem stolu na své polovině ovšem je zakázáno překročit na stranu k soupeřovi [2]. Volný pohyb hokejky je podobně jako pohyb hráče omezen pouze na polovinu herní plochy, jakmile se ale puk dotýká středové čáry, může jej zasáhnout kterýkoliv hráč [2]. Je zakázáno ovlivnit trajektorii puku jiným způsobem, nežli interakcí hokejky. Hra začíná vhazováním, kdy je puk uvolněn rozhodčím ve středu hřiště, každé další rozehrání pak začíná na straně stolu, na kterém byla naposledy inkasována branka [2]. Puk se nesmí vyskytovat na jedné půlce hrací plochy déle než sedm sekund. Pokud hráč nestihne vystřelit, propadá puk protivníkovi. Hráč, který první vstřelí sedm gólů (bodů) ukončuje a vyhrává hru. V některých případech lze hrát i do devíti bodů [2].

### 2.1.2 Rozměry

Standardní rozměry turnajového stolu vzdušného hokeje odpovídají americkým jednotkám v palcích: délka - 99 1/2" (252.73 cm), šířka - 51 1/4" (130.17 cm) a výška - 31" (78.74 cm) [3]. Pro rozměry místnosti, ve které se hokej nachází, je pak doporučeno počítat z každé strany stolu s alespoň metr širokým volným prostorem (viz obrázek 2) [3].



Obrázek 2: Standardizované rozměry vzdušného hokeje s doporučenými rozměry místnosti [3]

### 2.1.3 Ostatní parametry

Ostatní parametry jako je velikost a váha puku nebo hokejky (viz obrázek 3) pak nejsou definovány přesně a závisí na výrobci nebo na velikosti a výkonu stolu, ke kterému jsou pořizovány. Průměry puků se pohybují od 45 cm do 90 cm a mohou vážit až 42 gramů [4]. Pro turnajové a profesionální účely se většinou používají puky větší a těžší, zejména kvůli snížení rizika ztráty kontaktu puku s hrací plochou při silnějších úderech [3]. Váha hokejky není standardizována. Těžší hokejky ale umožňují prudší údery, proto jsou často na turnajích limitovány do 170 gramů nebo méně [4].



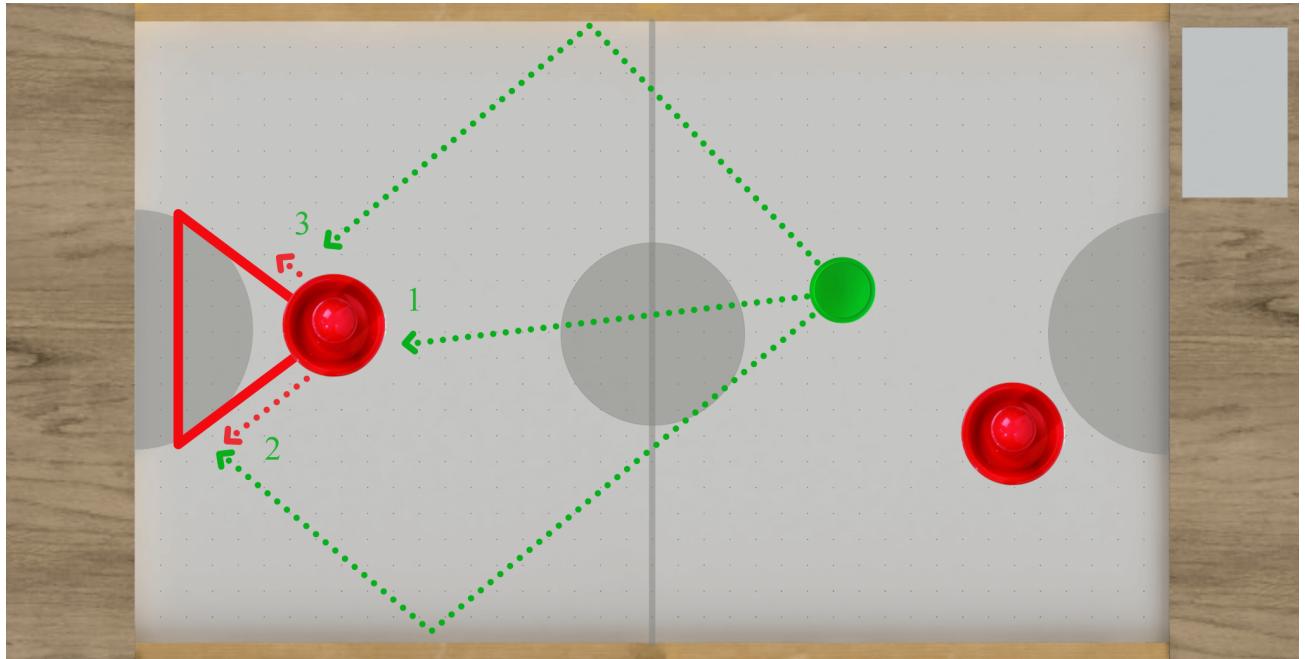
Obrázek 3: Standardní hokejky s pukem pro stolní hru vzdušný hokej [5, 6]

## 2.2 Styl hry

Při návrhu a vývoji počítačové heuristiky, vyhodnocující akční zásah pro daný stav systému, lze často s výhodou využít již zaběhlé a všeobecně známé postupy využívané manuálně lidskou obsluhou. Proto i pro návrh herní strategie RVH lze zakládat na osvědčených stylech hry profesionálních hráčů. Analýza nejčastějších obraných formací a útočných střel v zápasech turnajových úrovní potom může sloužit jako odrazový můstek pro vývoj rozhodovacího algoritmu. Zajímavý může být i pohled na mnohdy instinktivní chování lidského těla při rychlých změnách trajektorie puku a míra schopnosti rychle se pohybující puk sledovat. Pomocí těchto informací je pak možné najít slabá místa v obraně lidského protihráče nebo naopak úspěšně ubránit často nebezpečné údery.

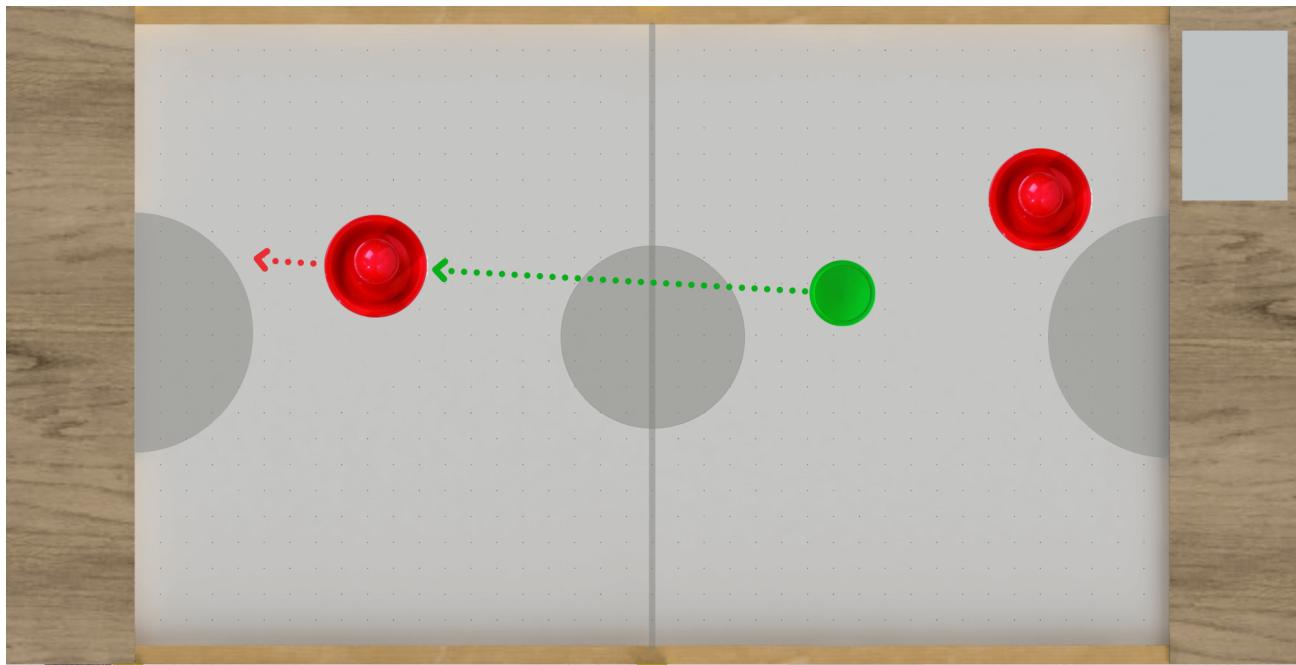
### 2.2.1 Obrana

Základním opěrným bodem v silné strategii je spolehlivá obrana. Ta vychází nejen z dobře kryté brankové oblasti, ale i schopnosti udržet puk po zákroku na obranné polovině hrací plochy. Profesionální hráči nejčastěji uplatňují tzv. „trojúhelníkovou obranu“ [7] znázorněnou na obrázku 4. Výchozí poloha hokejky je v této formaci vysunutá poměrně daleko před brankovou čárou a vykrývá tak střely mířené přímo na brankový otvor (1). Při úderech protivníka s odrazem o boční mantinel bránící hráč zasune hokejku ke kraji branky na straně odrazu (2). Vykrývá tak bližší brankovou tyč. V případě ostřejšího úhlu odrazu, a tedy útoku na vzdálenější tyč branky, se hokejka posouvá směrem k bližší tyči jen částečně (3). Hokejka by měla puk vyrazit směrem vodorovným s brankovou čárou a udržet tak puk v obranné polovině [7].



Obrázek 4: Znázornění trojúhelníkové obrany

Při pomalejších střelách může bránící hráč zastavit puk tím, že malý okamžik před střetem puku s hokejkou posune hokejku ve směru trajektorie střely s plynulým zpomalením pohybu až do úplného zastavení [7] (viz obrázek 5). Puk tak ztratí svoji kinetickou energii a bránící hráč může okamžitě zaútočit bez nutnosti dalšího zpracování.



Obrázek 5: Znázornění pohybu hokejky při zpracovávání pomalé střely

### 2.2.2 Útok

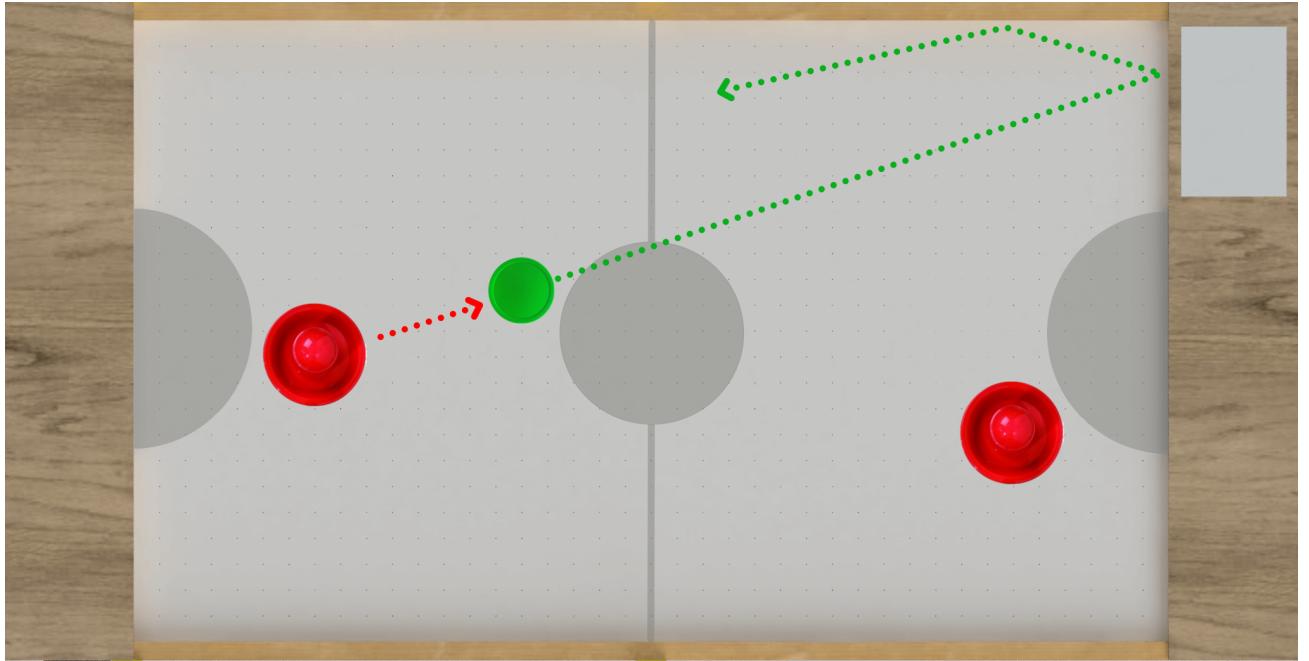
Způsob útoku na bránu soupeře lze rozdělit na základní typy střel:

- Přímá střela na bránu (1)
- Střela s odrazem od bočního mantinelu
  - Střela k bližší tyči (2)
  - Střela k vzdálenější tyči (3)

V případě nepřímé střely pak záleží, zda je puk mířen na bližší či vzdálenější tyč od místa odrazu. Typy střel (1-3) lze vidět na obrázku 4. V některých situacích se v profesionální hře využívá i několikanásobný odrazu puku od obou mantinelů. Puk se tak stává hůře předvídatelný pro bránícího hráče. Takový typ střely má ale zásadní nevýhodu v podobě dlouhé doby doletu k brankové oblasti a dává tak obránci více času. Zároveň odrazy od mantinelů výrazně snižují rychlosť puku a stěžují míření útočníka, proto je jejich využití výjimečné

Vzhledem k pravidlu, které zakazuje držení puku na jedné polovině hrací plochy déle jak sedm sekund (viz kapitola 2.1.1), se kromě úderů s cílem skórovat využívají i takzvané falešné

střely [7] za účelem pouze zmást protivníka a odrazem od zadní stěny získat puk zpátky na svoji polovinu (viz obrázek 6). Odpočítávání maximální doby puku na útočníkově polovině se tímto restartuje. Celá útočná série se tak může skládat s několika falešnými střel zakončená přesným útokem na bránu.



Obrázek 6: Znázornění falešné střely

### 2.2.3 Analýza lidského chování

Detailní analýzou lidského chování při různých sportovních aktivitách lze získat mnoho užitečných informací nejen o vlastnostech, ve kterých lidský mozek vyniká, ale i o situacích, kdy je omezená rychlosť reakce zásadní nevýhodou. V jedné z nejrychlejších stolních her světa - vzdušného hokeje jsou tyto nedostatky násobeny.

V publikaci vydané v roce 2012 [9] byly shrnutý poznatky ze sledování pohybu očí lidských hráčů při předem definovaných herních situacích. Pro rozpoznání sledovaného bodu byl využit přístroj EMR8<sup>1</sup>, připevněný na helmě sledovaného hráče a využívající infračervené záření spolu se snímací kamerou (viz obrázek 7). Získané data byly poté pomocí transformační matice přepočítávané do souřadného systému herního stolu. Transformační matice byla neustále aktualizována a počítána z projektovaných obrazců na herní ploše pro co nejpřesnější výsledky.

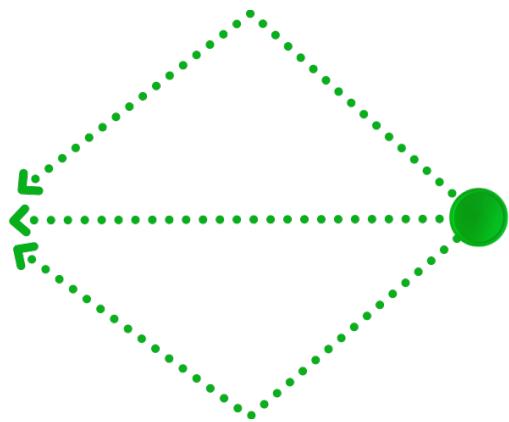
<sup>1</sup>Přístroj pro obrazovou analýzu firmy NAC Image Technology Inc



Obrázek 7: Fotka experimentální sestavy pro sledování pohybu očí lidského hráče [9]

Sledování pohybu očí probíhalo na třech předem definovaných trajektoriích puku se dvěma hodnotami rychlosti. Dohromady bylo tedy opakovaně zkoumáno šest různých situací. Trajektorie puku se náhodně střídala mezi přímou střelou středem, odrazem od levého mantinelu a odrazem od pravého mantinelu [9] (viz obrázek 8). Střely byly prováděny testovací robotickou rukou se dvěma stupni volnosti sestavenou pro tyto účely autory publikace. Tím byla zajištěna reproducibilnost jednotlivých střel.

Cílem bylo zjistit, u jakých typů střel a při jakých rychlostech bylo sledování puku nejproblematičtější a naopak které situace nedělali lidské schopnosti sledovat pohybující se objekt potíže.



Obrázek 8: Schéma testovacích trajektorií puku při sledování pohybu očí lidského hráče

Pro pomalejší verze trajektorií střely ( $2 \text{ m/s}$ ) vyplynuly z výsledku sledování následující poznatky [9]:

- Subjekt při snaze pozorování pohybujícího se puku sledoval oblast, kde se puk nacházel před 0,2 vteřinami.
- V častých případech subjekt změnil pohled na protivníkovu hokejku okamžitě po odražení puku na své polovině.

Při rychlých střelách ( $3\text{-}5 \text{ m/s}$ ) potom [9]:

- Subjekt sledoval oblast, kde se puk nacházel před 0,27 vteřinami.
- V některých případech subjekt ztratil schopnost puk sledovat úplně.

Těsně před úderem robota subjekt v obou případech sledoval hrací plochy před pukem ve směru k subjektu. U začátečnických hráčů se také jevilo problematické sledování trajektorie puku s místem odrazu poblíž hráče. V případě odrazu puku od mantinelu poblíž robotické ruky, bylo sledování puku obecně úspěšnější [9]. Z těchto dat lze poté vyvodit typ střely s největší pravděpodobností ztráty schopnosti sledovat její trajektorii obráncem - a tedy střela s odrazem od mantinelu poblíž obráncovi brány.

## 2.3 Matematický model hry

Vzhledem k vlastnostem systému se matematický model vzdušného hokeje dá po zanedbání teoretických nelinearit jako je náklon stolu nebo nehomogenní proudění vzduchu děrovanou deskou omezit na popis puku a jeho interakci s herní plochou, mantinely a herními hokejkami.

### 2.3.1 Matematický popis puku

Herní puk vzdušného hokeje je možné matematicky popsat jako tenký uniformní disk s hmotností  $m$ , poloměrem  $r$  a momentem hybnosti  $I$  k středové ose vyjádřeným vztahem (1)[10]

$$I = \frac{mr^2}{2}. \quad (1)$$

Popis stavu puku  $q$  v čase  $t$  je potom vyjádřen vztahem (2) [10]:

$$q(t) = (x(t), y(t), \Theta(t)), \quad (2)$$

kde  $x(t)$  a  $y(t)$  je poloha puku a  $\Theta(t)$  je natočení puku v souřadnicovém systému vzdušného hokeje.

### 2.3.2 Interakce puku s herní plochou

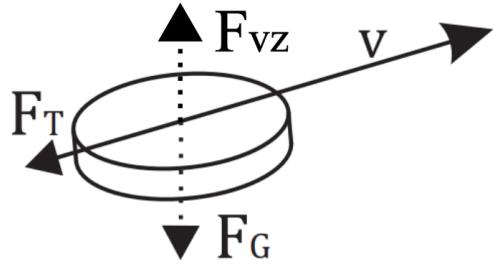
Vzduch, proudící zpod děrované desky a vytvářející vzduchový polštář pod klouzajícím pukem, výrazně snižuje tření a zajišťuje tak pohyb téměř bez ztráty rychlosti. Zvláště při pomalých pohybech ale tento úbytek kinetické energie zanedbatelný není. Při předpokladu, že je stůl vodorovný, lze sílu působící proti pohybu puku vyjádřit vztahem (3) [34]:

$$F_T = \mu \cdot F_N, \quad (3)$$

kde  $\mu$  je koeficient tření a  $F_N$  je síla působící na podložku vyjádřená vztahem (4):

$$F_N = m \cdot g - F_{vz}, \quad (4)$$

kde  $m$  je hmotnost puku,  $g$  je gravitační zrychlení a  $F_{vz}$  je nadnášející síla vzniklá proudem vzduchu. Grafické znázornění působících sil lze vidět na obrázku 9 [34].



Obrázek 9: Znázornění sil působící na puk pohybující se po hrací desce [34]

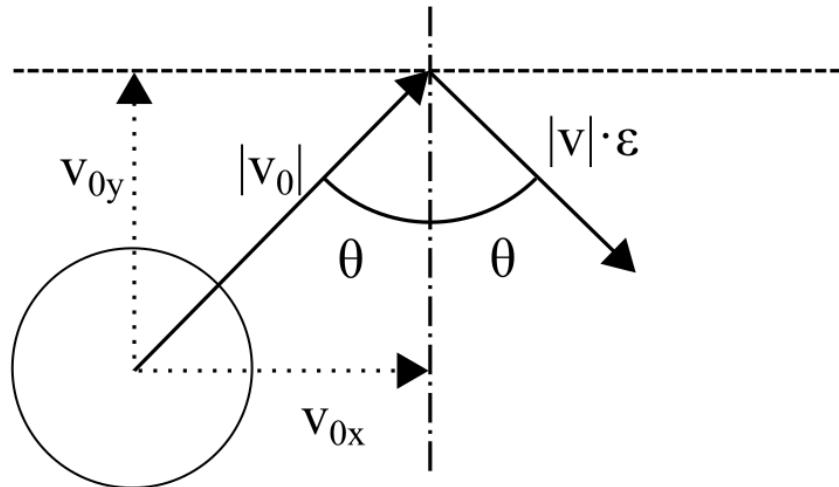
### 2.3.3 Odraz puku od mantinelů

Největší úbytek kinetické energie puku nastává při odrazech od krajních mantinelů [12]. Po zanedbání změny rotace vzhledem k malému tření mezi pukem a mantinem v okamžiku interakce, můžeme výsledný vektor rychlosti po odrazu zjednodušeně vyjádřit zrcadlením vektoru rychlosti přes normálový vektor mantinela a zmenšením jeho velikosti.

Výsledná tlumená velikost je potom vyjádřena vztahem (5):

$$|v| = |v_0| \cdot \epsilon, \quad (5)$$

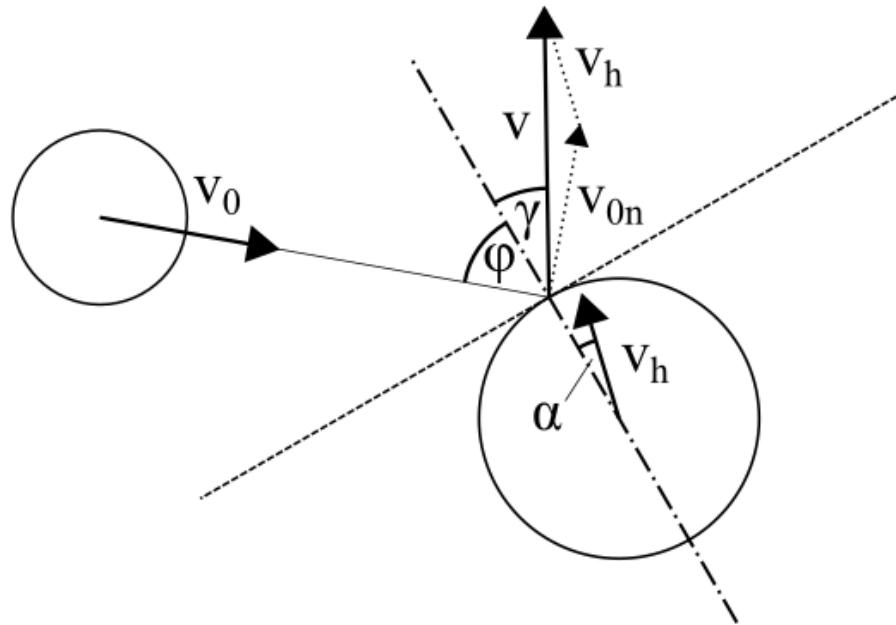
kde  $|v|$  je velikost výsledné rychlosti,  $|v_0|$  je původní velikost rychlosti před odrazem a  $\epsilon$  je koeficient tlumení. Znázornění odrazu lze vidět na obrázku 10.



Obrázek 10: Znázornění odrazu puku od mantinelu

### 2.3.4 Interakce puku s hokejkou

Nejsložitější z interakcí je model kolize puku s hokejkou. Popis je podobný odrazu od mantinelu s rozdílem nenulové rychlosti herní hokejky. Při předpokladu elastické srážky a zanedbání změny pohybu hokejky při srážce lze vyjádřit výslednou rychlosť puku po odrazu  $v$  jako součet vektorů zrcadlené rychlosťi puku  $v_{0n}$  od normály kolizní roviny a vektoru rychlosťi hokejky  $v_h$  [13, 14]. Znázornění odrazu puku od hokejky lze vidět na obrázku 11.



Obrázek 11: Znázornění odrazu puku od pohybující se hokejky



### 3 Detekce pozice objektů

Způsoby řešení sledování polohy pohybujícího se objektu lze rozdělit na dva hlavní přístupy:

- sledování objektu pomocí kamerových systémů
- využití jiných hardwarových prvků pro detekci polohy

V obou případech je hlavní důraz kladen na co nejpřesnější výstup informace reprezentující polohu sledovaného objektu s minimálním efektem šumu a maximální obnovovací frekvencí.

#### 3.1 Kamerové systémy

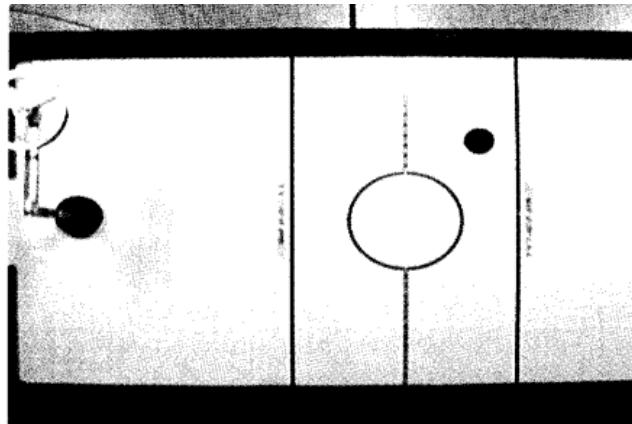
Výhodou sledování polohy objektu pomocí kamerových systémů je zejména obrovská flexibilita ve výběru patřičného hardwaru a relativní jednoduchost v implementaci sledovacích algoritmů. Často je ale zapotřebí poměrně velká vzdálenost kamery od roviny, kde se sledovaný objekt nachází a celý systém tak může zabírat velký prostor. Přesnost detekované polohy a hodnota obnovovací frekvence pak závisí na rozlišovací schopnosti kamery respektive výkonu výpočetní jednotky, která jednotlivé snímky z kamery zpracovává. Díky tomu se může kvalitní a rychlá detekce projevit vysokou pořizovací cenou hardwaru.

##### 3.1.1 Algoritmy detekce

Detekování polohy objektu je prováděno na jednotlivých snímcích získávaných z kamery. Podle typu kamery a vlastnostech sledovaného objektu spolu s jeho okolím lze využít jeden z následujících přístupů (nebo jejich kombinaci) .

##### Hledání objektu podle úrovně světlosti pixelů.

Princip vyhledání objektu je v tomto případě založen na převedení jednotlivých snímků na binární obraz, tzv. masku snímku (viz obrázek 12).



Obrázek 12: Maska snímku hrací plochy vzdušného hokeje

Každému pixelu, který spadá svojí hodnotou odstínu šedi do daného vyhledávajícího intervalu je přiřazena pravdivostní hodnota 1. Naopak pokud je hodnota mimo rozsah, odpovídajícímu pixelu je přiřazena hodnota 0. Matice těchto hodnot je právě maska snímku, kde pixely s hodnotou 1 odpovídají nalezenému objektu. Pozici hledaného objektu poté odpovídá poloha těžiště pixelů masky s pravdivostní hodnotou 1 [15, 16]. Z obrázku 12 je zřejmé, že by poloha těžiště neodpovídala poloze hledaného objektu, proto by v tomto případě bylo zapotřebí změnit vyhledávací parametry nebo zavést další kritéria (minimální velikost shluku nalezených pixelů apod.).

Vzhledem k využití informace pouze o intenzitě jednotlivých pixelů lze tuto metodu s výhodou využít v kombinaci s černobílými kamerami. Zároveň jsou díky jednoduchosti algoritmu kladený nižší nároky na výkonost výpočetní jednotky. Informace pouze o intenzitě pixelu může být v ale i nevýhoda právě v případě, že se v záběru kamery objevují parazitní objekty s podobnou hodnotou odstínu šedi jako má hledaný objekt.

### Hledání objektu podle barvy pixelů.

Metoda je podobná principu hledání objektu pouze pomocí intenzity pixelů, využívá ale dodatečnou informaci o barvě. Snímky se konvertují do formátu HSV<sup>2</sup> a hodnota H (Hue) jednotlivých pixelů se poté porovnává s vyhledávajícím intervalom. Vyhledávání je tak z principu popisu barevného modelu HSV invariantní ke změně intenzity osvětlení scény [17, 18, 19]. Podobně jako u předchozí metody tak vznikne maska snímku, ze kterého se dalším postupem získává poloha nalezeného objektu (viz obrázek 13).



Obrázek 13: Původní obrázek a maska obrázku při hledání objektu za pomocí barvy [20]

Metoda je stále poměrně nenáročná na výpočetní výkon a zároveň nezávislá na rozlišení kamery. Pro zvýšení obnovovací frekvence tak lze jednoduše snížit rozlišení snímků bez potřeby úpravy vyhledávajícího algoritmu. Problém ale nastává se změnou baveného odstínu osvětlení, proto je v tomto případě doporučeno zajistit neměnný zdroj světla [19].

<sup>2</sup>HSV = Hue - odstín, Saturation - sytost barvy, Value - hodnota jasu

### Hledání objektu podle detekovaných tvarů.

V případě, kdy není barva či hodnota stupně šedi hledaného objektu na snímku jedinečná, lze využít algoritmů pro vyhledávání specifických tvarů. Například při detekci hledaného objektu se specifickým kruhovým tvarem lze využít Hoghovu transformaci pro detekci kružnic [21] (viz obrázek 14). Algoritmus tak není závislý na specifické barvě či světlosti objektu což je častý předpoklad při potřebě detekování více předmětů na jednom snímku jako je sledování dopravních prostředků nebo kontrola vadních kusů při výrobě [22].

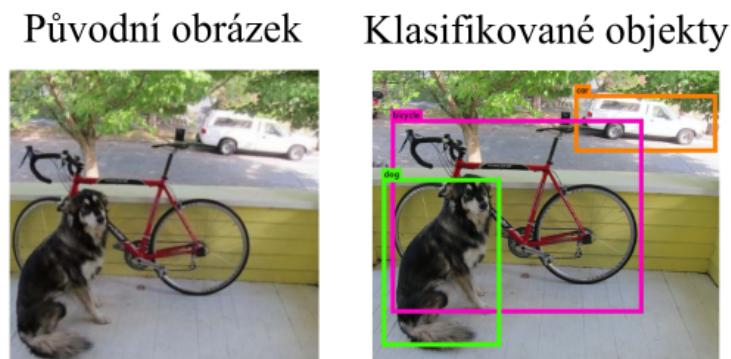


Obrázek 14: Nalezená kružnice na obrázku plechovky pomocí Hoghovi transformace [21]

Zásadní nevýhodou je často velká výpočetní náročnost a v některých případech náchylnost na šum. Proto je metoda nalezení hran a specifických geometrických tvarů využívána spolu s filtračními algoritmy zejména v aplikacích nespoléhajících na vysokou obnovovací frekvenci detekce.

### Hledání objektu pomocí umělé inteligence.

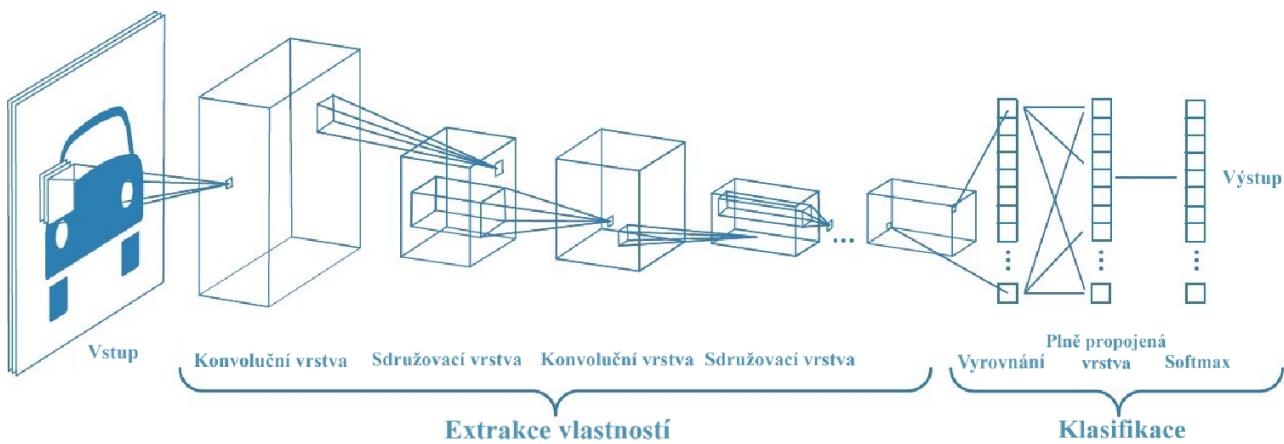
Asi nejobecnější způsob detekce při správné implementaci je právě využití umělé inteligence v podobě naučených neuronových sítí. Výstupem z neuronové sítě pak může být rovnou hledaná poloha bez nutnosti výpočtu těžiště masky apod. Samotná detekce je tak velmi rychlá, u některých algoritmů jako je například YOLO [23] stačí jeden průchod sítě pro kompletní detekci a klasifikaci i několika objektů (viz obrázek 15), což umožňuje využívat tyto typy sítě i u videí a streamů s vysokou obnovovací frekvencí.



Obrázek 15: Detekce a klasifikace objektů neuronovou sítí YOLO [24]

Nevýhoda neuronových sítí spočívá v jejich složité implementaci a nutnosti provádět časově velmi náročné učení před samotným využitím. Častým problémem je i fakt, že se kromě učení nedá ovlivnit, co neuronová síť vnímá jako hledaný objekt. Může se tak stát, že malá změna ve vlastnostech obrazu (šum, natočení kamery) zapříčiní nesprávnou detekci objektu [23].

Z podstaty neuronových sítí je také problematická změna velikosti vstupní matice - tedy rozlišení obrazu. Znamená to, že při této metodě rozpoznávání objektu není možné bez úpravy struktury a opětovného učení měnit rozlišení vstupních snímků [25]. Příklad neuronové sítě využívané pro klasifikaci objektu lze vidět na obrázku 16.



Obrázek 16: Postup strukturou konvoluční neuronové sítě pro klasifikaci objektu [26]

### 3.1.2 Příklady hardware

Rychlosť a kvalitu detekcie ovlivňuje kromě použitého algoritmu i na samotný hardware. Ten i z velké časti určuje celkovou cenu systému. Ne vždy ale platí, že dražší kamera znamená kvalitnější výsledky nebo výkonnější výpočetní jednotka rychlejší detekci. Navíc v případě ne-kvalitní kamery výkonná jednotka ztrácí z velké části smysl a naopak. Níže budou uvedeny příklady možných kombinací kamery a výpočetního hardwaru.

#### Průmyslové kamery.

Průmyslové kamery integrují jak samotnou kameru, tak výpočetní jednotku instalovanou v jenom kompaktním zařízení. Přenos detekovaných dat je pak zajištěn například pomocí ethernetového kabelu. Na trhu existuje několik společností nabízející spoustu různých konfigurací pro specifické případy [27] (viz obrázek 17). Většinou je instalování velice rychlé a systémy obsahují již předprogramované algoritmy pro základní potřeby strojového vidění, takže se zkracuje i doba vývoje. Nevýhodou je ale vysoká cena a často omezené způsoby uživatelské konfigurace.



Obrázek 17: Průmyslové kamery série 7000 společnosti Cognex

### Web kamery s externí výpočetní jednotkou.

Jedna z možností je i využití web kamery připojované standardním USB<sup>3</sup> rozhraním. Celý obraz je tak přenášen do externí výpočetní jednotky na které jsou snímky zpracovávány. Výhodou je velká flexibilita ve výběru hardwaru a možnost tak výrazně ušetřit náklady například využitím již vlastněných web kamer z jiných aplikací. V kombinaci s levnými mikropočítači jako je Raspberry Pi (viz obrázek 18) dokáže být tato možnost cenově velice výhodná [29]. Nutnost přenášet celý obraz skrze standardizované rozhraní může ale znatelně zpomalovat celkovou detekci. Časově náročnější je i samotná konfigurace a vývoj systému bez možnosti využití proprietárních předprogramovaných algoritmů.



Obrázek 18: Příklad web kamery zapojené do mikropočítače Raspberry Pi [29]

---

<sup>3</sup>Universal Serial Bus - univerzální sériová sběrnice umožňující připojení periferií k počítači

## Raspberry Pi s modulem kamery.

Konkrétním příkladem kamer s externí výpočetní jednotkou je kombinace Raspberry Pi s modulem kamery dodávaným speciálně pro tento mikropočítač. Rozdíl od web kamery je ale v připojení komunikace. Kamera v tomto případě nemusí s Raspberry Pi komunikovat skrze USB rozhraní, ale data jsou posílána přímo na grafický čip počítače, kde se dále zpracovávají a nezatěžují proto hlavní výpočetní jednotku [30]. Přenos je tak výrazně urychlen a tím i navýšen počet přijatých snímků za vteřinu. Vzhledem k speciálnímu typu připojení (viz obrázek 19) ale odpadá výhoda flexibility ve výběru hardwaru jako v případě využití web kamer. Naopak je však jednodušší konfigurace a samotný vývoj algoritmů vzhledem k velké rozšířenosti mezi uživateli a dobré podpory ze strany výrobce. K dispozici jsou i knihovny, zajišťující jednoduché ovládání a získávání dat z kamery v několika programovacích jazycích.



Obrázek 19: Raspberry Pi s připojeným modulem kamery [31]

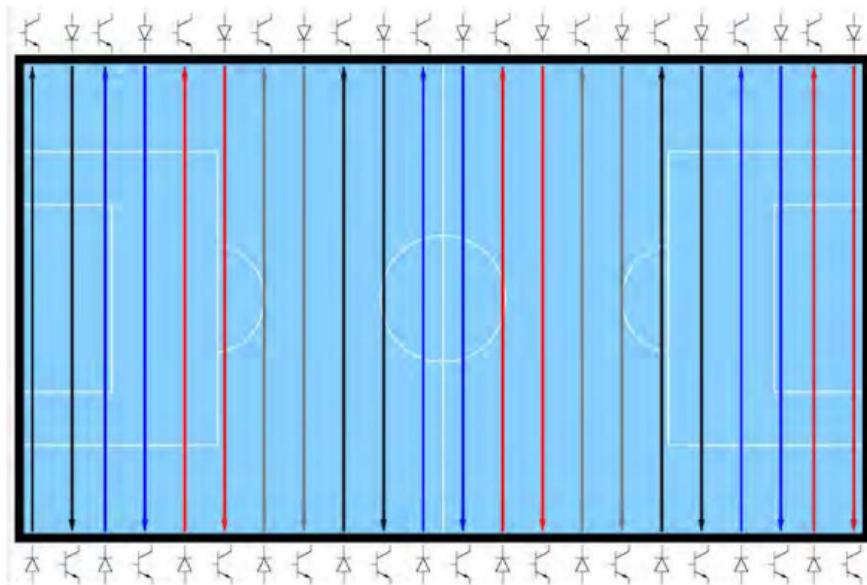
Aktuálně existují dvě verze těchto kamer dodávané přímo výrobcem samotného Raspberry Pi počítače. V nejnovější verzi kamera umožňuje snímání obrazu s rozlišením až 3280x2464 pixelů a nahrávání videa v rozlišení 1920x1080 s rychlosťí 30 snímků za vteřinu nebo v rozlišení 640x480 s rychlosťí 90 snímků za vteřinu. Pokrývá tak situace využití jak při potřebě kvalitních snímků, tak při nutnosti rychlého snímání s menším rozlišením (například při detekci polohy objektu).

### 3.2 Ostatní přístupy

Metody využívající kamerové systémy jsou v řadě případů jediná rozumná řešení při požadavku bezpečné detekce polohy objektu. Ve specifických situacích je ale naopak vhodnější využít jiný přístup. Řadí se mezi ně například detekce polohy pomocí dotyku s měřící sondou, využití magnetických vlastností objektu nebo detekce pomocí senzorů vzdálenosti. Často je pak takové řešení rychlejší, levnější a někdy i spolehlivější než již zmíněné kamerové systémy.

Příkladem tomu může být přístup, se kterým přišli studenti z polytechnické univerzity z Ohia, kteří řešili problém detekce míčku na herní ploše stolního fotbalu. V tomto případě by byl pohled kamery umístěn nad fotbalem blokován samotnými hráči a detekce by tak mohla být problematická [35]. Studenti využili sérii infračervených LED diod a fototranzistorů umístěných naproti sobě ve vnitřním obvodu stolu. Detekce tak nebyla blokována herními osami a vynikala velmi vysokou rychlostí odezvy.

Problém ale vznikal rušením a vzájemným ovlivňováním sousedních dvojic (dioda – tranzistor). Toto bylo řešeno dvěma způsoby. První z nich byl zahľoubení diod do boční stěny stolu. Tímto se omezila šířka emitovaného signálu a velikost ovlivněné plochy na protější straně. Druhým řešením bylo fázové posunutí intervalů sepnutí diod. Integrací obou zmíněných opatření byly pak nejbližší, ve stejný okamžik sepnuté diody od sebe vzdáleny asi 32 cm. Celkový počet takto fázově posunutých dvojic byl 16 [33]. Princip zapojení je naznačen na obrázku 20 (různé časové intervaly zapnutí jsou znázorněny barevným odlišením).



Obrázek 20: Princip zapojení a uspořádání infračervených LED diod a tranzistorů [35]

Takto zapojené dvojice byly orientovány v horizontálním (117 páru) a vertikálním (68 páru) směru. Z obrázku je také patrná hlavní nevýhoda tohoto řešení. A tedy, že přesnost detekované polohy silně závisí na počtu páru. Oproti tomu počet páru zase zesiluje rušení a zvyšuje cenu celé soustavy.



## 4 Konstrukce a hardware

Brutálně vychváli Doma, kterej je hlavní šéf projektu a totální bombič.

(Nevim jestli takhle, někam to ale musím dat)

Popsat raspberry pi, kameru, displej atak

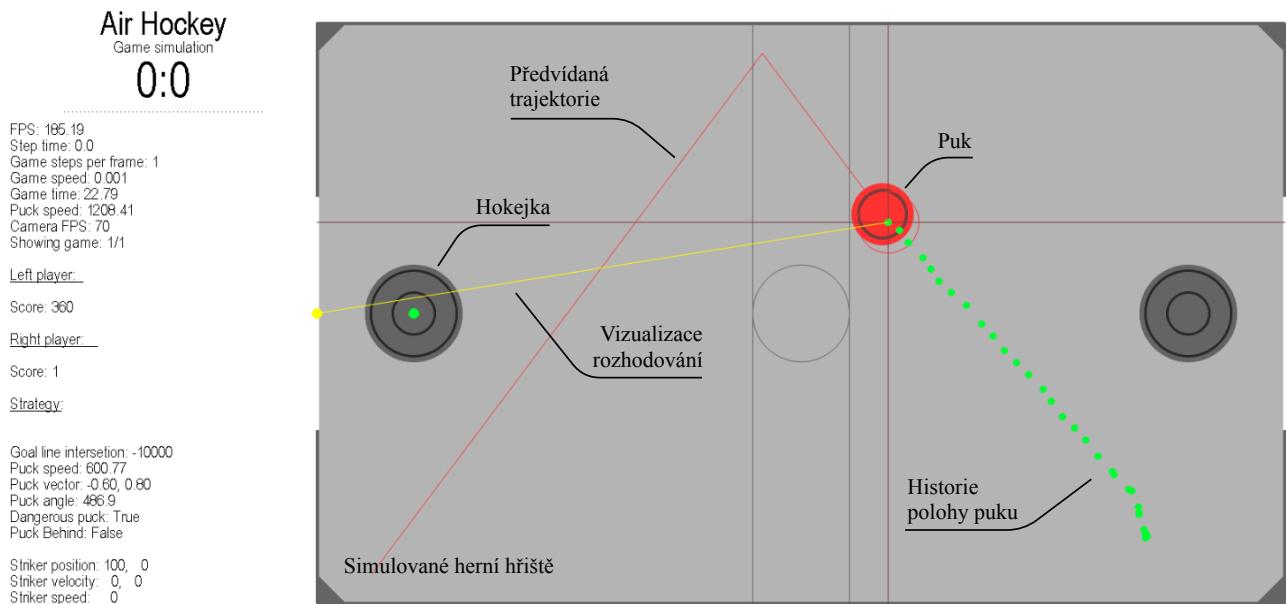


## 5 Simulace hry

Z důvodu potřeby vývoje, intenzivního testování strategie hry a vyladění rozhodovacích algoritmů při specifických situacích byla vytvořena počítačová simulace hry vzdušného hokeje. Simulace pak umožňovala jak zpomalení a zrychlení simulovaného času pro detailní analýzu stylu hry aktuálně aktivní strategie, tak i paralelní zpracovávání několika spuštěných her současně, umožňující zkoumat statistické údaje úspěšnosti naprogramovaných strategií hrajících proti sobě. Vývoj rozhodovacích algoritmů navíc nemusel záviset na kompletně sestaveném a dokončeném RVH.

Pro tyto účely bylo nutné zajistit co nejpřesnější reprezentaci reálného chování systému kompletně v simulovaném prostředí spolu i s náhodnými nepřesnostmi snímání puku na ploše hokeje a dynamickými vlastnostmi robotické hokejky. Součástí simulace jsou také algoritmy ohodnocující střelu vyslanou jedním z hráčů. Pomocí těchto metrik jsou poté přidělovány body každému z hráčů. Ty můžou sloužit jako ukazatel výkonnosti dané strategie i při nevstřelení žádného gólu.

V průběhu simulace je možné interagovat s pukem a jedním z hráčů pomocí myši a zajistit tak opakovatelnost specifických situací. V kombinaci s možností zpomalení simulovaného času se tak stává simulační software velice užitečným vývojovým nástrojem. V neposlední řadě bylo umožněno i nahrání uložených záznamů her přímo z RVH pro možnost dodatečného zkoumání situací i z reálného stolu. Výslednou podobu vytvořeného simulačního softwaru i s informačním panelem zobrazující aktuální stav hry lze vidět na obrázku 21.



Obrázek 21: Simulační program pro vývoj a analýzu strategie RVH

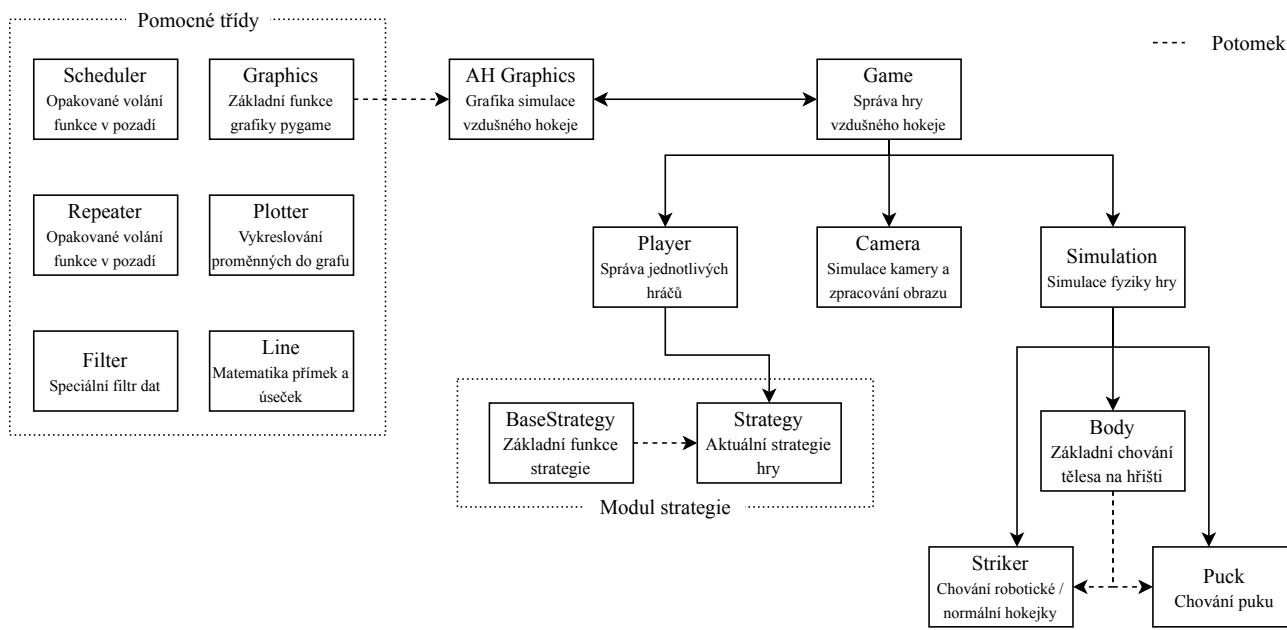
V následující kapitole budou popsány jednotlivé stěžejní části vývoje simulačního softwaru v jazyce Python spolu s vysvětlením celkového rozvržení struktury programu.

## 5.1 Struktura programu

Vzhledem k nutnosti simulace celého systému RVH se struktura simulačního softwaru v mnoha ohledech podobala výsledné struktuře softwaru aplikovaného přímo v řídící jednotce stolu. Na rozdíl od výsledného softwaru pro RVH nebyl kladen příliš velký důraz na optimalizaci kódu, ale spíše na zkrácení doby vývoje pro co nejrychlejší možnost návrhu a testování strategií. Některé dodatečné funkce simulačního softwaru pak vznikaly přímo při návrhu strategií podle aktuální potřeby testování.

Logika programu je rozdělena na jednotlivé třídy spravující určitou oblast problému. Třídy mohou z podstaty problému spadat pod třídy jim nadřazené. Například třída *Player*, která řeší logiku spojenou s jednotlivými hráči (aktualizace skóre, interpretování dat ze strategie atd.) spadá pod třídu *Game*, zahrnující nejen správu obou hráčů, ale například i zpracování dat ze simulované kamery snímající polohu simulovaného puku apod. Tímto přístupem při vývoji softwaru lze i při rozsáhlejším programu zajistit přehlednost a jednoduchou navigaci ve zdrojovém kódu.

Kromě takto hierarchicky navržených tříd byly vytvořeny i třídy pomocné, které nespadaly pod žádnou nadřenou třídu a byly využívány napříč celou strukturou a později i na softwaru reálného RVH. Zjednodušené znázornění celé struktury softwaru spolu s pomocnými třídami lze vidět na obrázku 22.



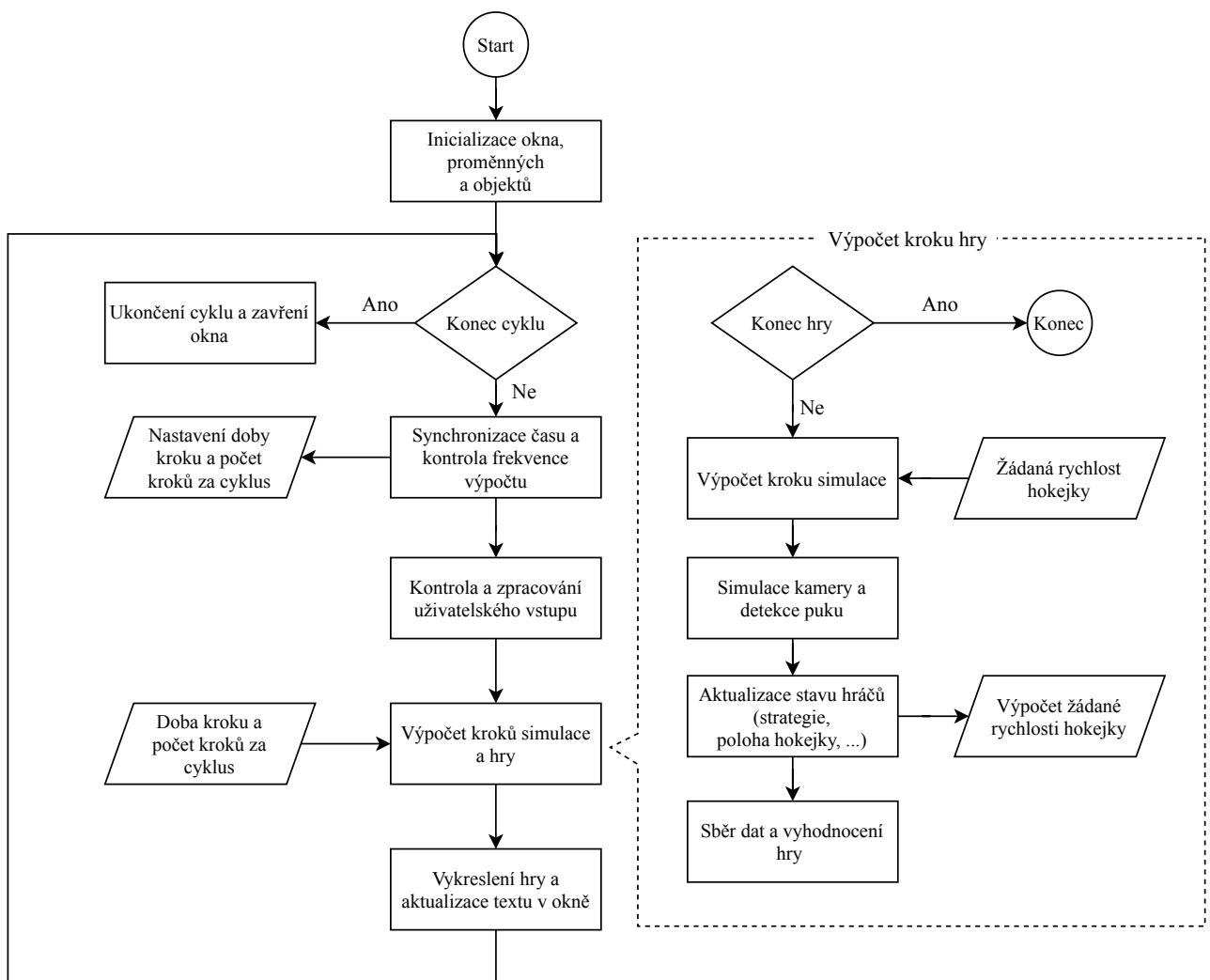
Obrázek 22: Znázornění struktury simulačního softwaru vzdušného hokeje

O některých třídách bude více pojednáno níže. Například strukturu modulu strategie, který je hlavním výstupem z následného návrhu strategie v tomto simulačním prostředí, popisuje kapitola 6.1. Většinu vzniklých pomocných tříd ale není možné z důvodu velké obsáhlosti a komplexnosti více přiblížovat. Pro více informací lze nahlédnout do zdrojových souborů v příloze.

Základem softwaru je knihovna pygame, která zpracovává uživatelský vstup z klávesnice a myši, zajišťuje vykreslování grafiky ve vytvořeném okně a poskytuje základní matematické funkce pro vektorový výpočet.

Po inicializování objektů a definování nastavovacích proměnných začne probíhat až do uzavření okna cyklus, který v každém svém průběhu kontroluje uživatelský vstup, přepočítává stav hry a simulace, aktualizuje grafický výstup a synchronizuje simulovaný čas.

Frekvence výpočtů závisí na aktuálně nastavené hodnotě rychlosti simulace. Při hodnotě vyšší jak 1 (zrychlená simulace) se provede výpočet kroku simulace i několikrát za jeden průběh hlavního cyklu (vykreslení snímku). Naopak při hodnotě menší jak 1 (zpomalená simulace) se při každém průběhu cyklu počítá krok simulace s menší dobou kroku *steptime*, tzn. že při jednom kroku se simulace posune o menší časový úsek. Vykreslování je tak i při zpomalení simulace stále plynulé. Vývojový diagram simulačního softwaru vzdušného hokeje lze vidět na obrázku 23.



Obrázek 23: Vývojový diagram simulačního softwaru vzdušného hokeje

## 5.2 Simulace fyziky

Implementace simulace fyziky hry RVH byla založena na teoretickém rozboru z kapitoly 2.3. Hodnoty tření a míra tlumení rychlosti puku po odrazech byly z počátku odhadovány. V pozdější fázi vývoje, kdy byly k dispozici data z kamery reálného systému, byly tyto hodnoty upravovány pro co nejpřesnější matematickou reprezentaci vzdušného hokeje.

Stěžejní částí simulace bylo řešení kolizí puku s ostatními objekty na hrací ploše. Konkrétně se kolize daly rozlišit na dvě specifické situace:

- Kolize s hokejkou
- Kolize s mantinelem

Pro každou situaci bylo nutné řešit mírně odlišný problém.

### 5.2.1 Kolize s hokejkou

Neboli typ kolize kruh-kruh (circle-circle). V každém kroku simulace se kontroluje vzdálenost puku ke každé z hokejek  $i \in \{1,2\}$ . Pokud je vzdálenost polohy středu puku  $P$  od středu jedné z hokejek  $H_i$  menší než součet poloměrů puku  $r_p$  a hokejky  $r_{hi}$  (6).

$$|PH| < r_p + r_{hi}, \quad (6)$$

pak dochází k překrytí hokejky  $i$  pukem a tedy ke kolizi. Postup řešení kolize typu kruh-kruh je pak vidět na zjednodušeném pseudokódu:

```

relativní rychlosť = rychlosť puku - rychlosť hokejky
normálka = normalizuj(pozice puku - pozice hokejky)
rychlosť ve směru normálky = relativní rychlosť x normálka
pokud rychlosť ve směru normálky > 0

```

ukonči výpočet

jinak

```

j = -
(1 + restituce) * rychlosť ve směru normálky/(1/celková hmotnosť)
impuls = normálka * j
rychlosť puku = impuls * 1/hmotnosť puku
rychlosť hokejky = impuls * 1/hmotnosť hokejky

```

**Algoritmus 1:** Řešení kolize typu kruh-kruh

### 5.2.2 Kolize s mantinelem

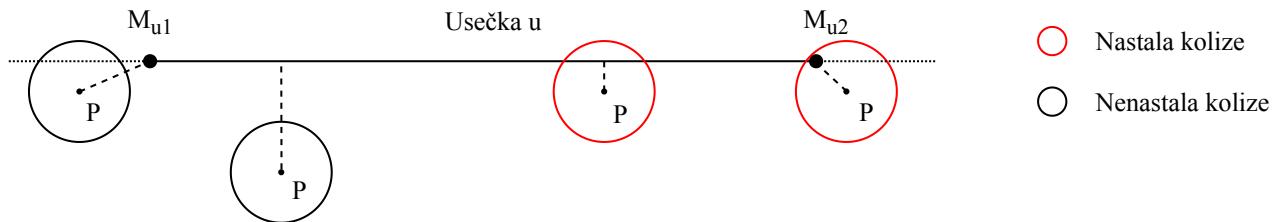
Neboli typ kolize kruh-úsečka (circle-line). V každém kroku simulace se kontroluje vzdálenost puku ke každé úsečce  $u \in \{1, 2, \dots, N\}$  definující ohraničení herní plochy. V tomto případě se však musí kontrolovat nejen zda kolmá vzdálenost polohy středu puku  $P[p_1, p_2]$  od přímky  $m : ax + by + c = 0$  procházející danou úsečkou  $u$  je menší jak poloměr puku  $r_p$  (7)

$$\frac{|a \cdot p_1 + b \cdot p_2|}{\sqrt{a^2 + b^2}} < r_p, \quad (7)$$

ale zároveň i jestli poloha průsečíku přímky  $m$  a kolmice od přímky  $m$  procházející bodem  $P$  leží mezi koncovými body úsečky  $u$ :  $M_{u1}$  a  $M_{u2}$ . Pokud průsečík úsečce nenáleží, je nutné kontrolovat zda vzdálenost puku  $P$  od koncových bodů úsečky  $M_{u1}$  a  $M_{u2}$  není menší jak poloměr puku  $r_p$  (8)

$$|PM_{uj}| < r_p \quad (8)$$

Pokud jedna z těchto dvou podmínek platí, dochází ke kolizi mezi kruhem a úsečkou a je nutné ji řešit. Znázorněné situace, které mohou nastat lze vidět na obrázku 24.



Obrázek 24: Možné situace při kontrole kolize typu kruh-úsečka

Postup řešení kolize typu kruh-úsečka lze vidět na zjednodušeném pseudokódu:

```

přesah = poloměr puku - vzdálenost od úsečky
dokud přesah > 0
    vektor posunu = škáluj_vektor(rychlota puku, poloměr puku/10)
    poloha puku = poloha puku - vektor posunu
    přesah = poloměr puku - vzdálenost od úsečky
pokud průsečík náleží úsečce
    rychlosť puku = zrcadli_vektor(rychlota puku, normála úsečky)
    rychlosť puku = rychlosť puku * tlumení
jinak
    řeš jako kolizi kruh-kruh pro puk a nejbližší koncový bod

```

**Algoritmus 2:** Řešení kolize typu kruh-úsečka

## 5.3 Simalace kamery

Při sledování herní plochy RVH kamerou a následném detekování polohy puku z jednotlivých snímků nelze nikdy obdržet dokonale přesnou a nezpožděnou informaci. Při simulaci by ale bez dodatečného simulovaného šumu strategie přesná a okamžitá data k dispozici měla. Proto by takto navržená strategie nemusela na reálném systému vykazovat uspokojivé výsledky. Z toho důvodu bylo nutné simulovat tři základní vlastnosti zapříčňující nepřesné snímání polohy.

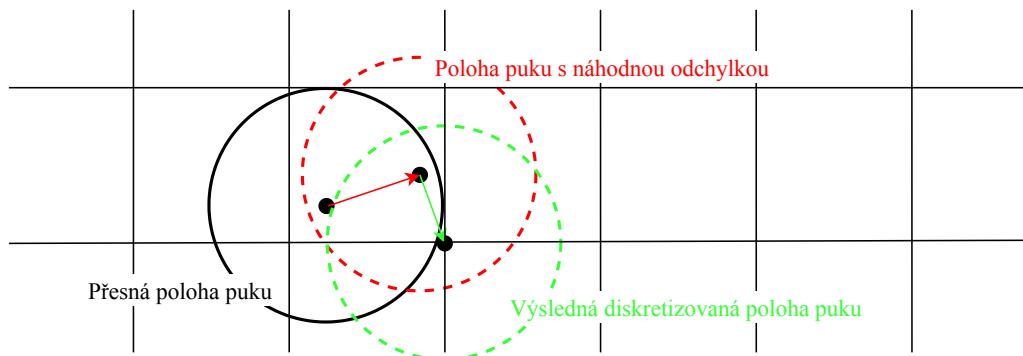
### 5.3.1 Náhodná odchylka

Nepřesnosti způsobené šumem snímaného obrazu byly zjednodušeně simulovány náhodnou odchylkou od přesné polohy puku. Odchylka byla modelována normálním rozdělením s nulovou střední hodnotou a směrodatnou odchylkou s hodnotou 2 mm.

### 5.3.2 Diskretizace polohy

Vzhledem ke omezenému rozlišení reálné kamery není možné pro algoritmus detekující polohu puku rozlišit rozdíl mezi dvěma polohami s menší vzdáleností než je odpovídající vzdálenost jednotlivých pixelů na snímku herní plochy. Při nastaveném rozlišení kamery 320x192 a předpokladu snímání celé herní plochy RVH o rozměrech 1000x600 mm vychází minimální detekovatelný rozdíl vzdálenosti dvou různých poloh na přibližně 3 mm.

Hrací plocha proto byla v simulaci rozdělena na mřížku poloh s rozestupem 3.5 mm, ve kterých lze puk detektovat. Detekovaná poloha pak odpovídala bodu na mřížce, který se nacházel nejblíže aktuální poloze puku s již započítanou náhodnou odchylkou. Znázornění simulace snímání a detekce puku lze vidět na obrázku 25.



Obrázek 25: Znázornění simulace snímání a detekce puku.

### 5.3.3 Zpoždění informace

Zpoždění informace o aktuální poloze vznikající na reálném systému zpracováváním snímku a komunikací s řídící jednotkou bylo možné simulovat velice jednoduše ukládáním několika posledních poloh a využíváním pouze té nejstarší ve strategii. Součin doby kroku a počet ukládaných poloh pak definoval dobu zpoždění.

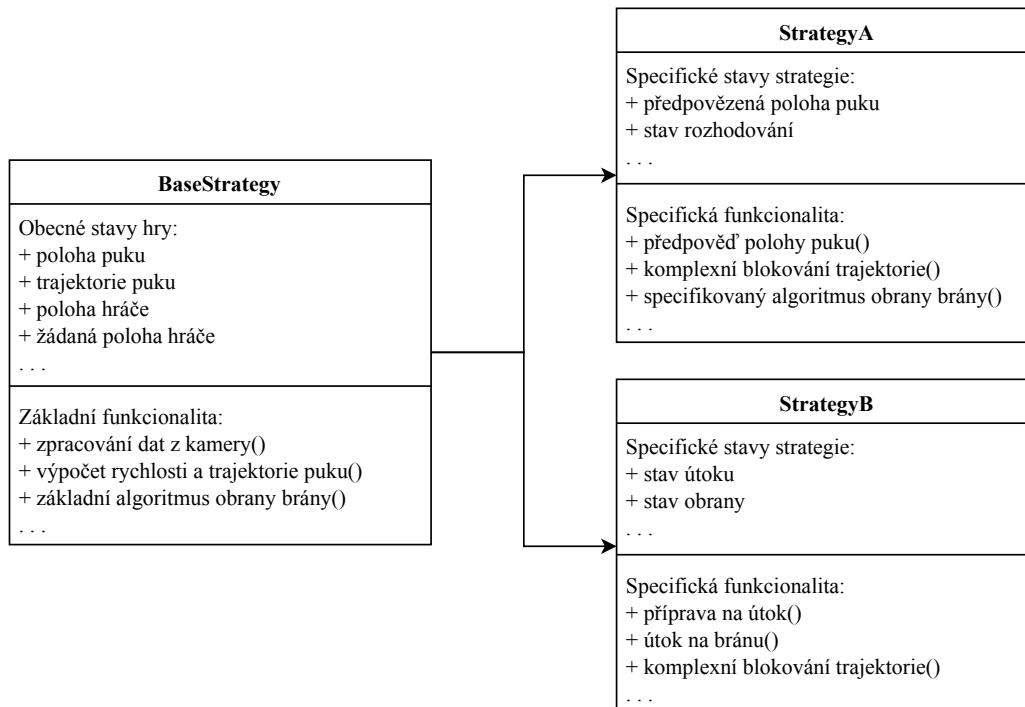
## 6 Návrh strategie

Strategií je v tomto kontextu rozuměna série rozhodovacích algoritmů, které určují okamžitou žádanou polohu hokejky robotického hráče na základě aktuálního stavu hry, jako je poloha a rychlosť snímaného puku nebo robotické hokejky. Z pravidla platí, že čím jsou k dispozici přesnější a rychleji aktualizované data o aktuálním stavu hry, tím přesněji a rychleji dokáže strategie reagovat. Samotná účinnost reakce ale závisí na rozhodovacích algoritmech ve strategii, o kterých bude tato kapitola pojednávat.

Strategie byla vytvořena jako softwarový modul v programovacím jazyce Python, možný bez nutnosti úprav používat jak v simulaci, tak při samotné implementaci na cílovém hardwaru RVH. Modularita tohoto řešení zkracuje celkový čas vývoje a umožňuje pravidelně analyzovat herní situace v simulačním prostředí a upravovat strategii i po dokončení projektu RVH.

### 6.1 Struktura modulu strategie

Struktura softwarového modulu strategie je založena na objektových principech dědičnosti. Základním stavebním kamenem programu je abstraktní třída *BaseStrategy*, která zaštiťuje obecnou funkcionality jako je zpracování vstupních dat z kamery, výpočet trajektorie pohybu puku, základní strategické algoritmy apod. Z této třídy poté dědí jednotlivé podtřídy specifikující výsledné chování robotického hráče. Tento přístup zajišťuje snadný vývoj nových typů strategií a bezproblémovou možnost změny strategie přímo za běhu programu. Zjednodušený diagram s příkladem dědění dvou tříd strategie lze vidět na obrázku 26.



Obrázek 26: Zobecněný UML diagram dědičnosti tříd modulu strategie

Každá z jednotlivých strategií (*StrategyA* a *StrategyB*) tak může implementovat specifické rozhodovací algoritmy s diametrálně odlišnými výsledky a zároveň využívat základní funkcionality a přistupovat k aktuálním stavům hry bez redundance napsaného kódu. Pro příklad k obrázku 26: strategie *StrategyB* vyniká v útoku s komplexními algoritmy pro míření, ale pro obranu využívá základní algoritmus obrany z rodičovské třídy *BaseStrategy* bez nutnosti definice celé funkce znova. Zdrojový kód této strategie je pak uložen do vlastního souboru. Oddělí se tak specifická logika dané strategie od obecných funkcí což vede k přehledné a lehce rozšířitelné struktuře celého projektu.

## 6.2 Základní třída *BaseStrategy*

Třída *BaseStrategy* obsahuje tři speciální metody, pomocí kterých se řídí celý výpočet strategie nadřazeným programem využívající tento modul:

- metoda ***cameraInput(pořada)***

Volána nadřazeným programem pokaždé, když jsou k dispozici nová data o pozici puku. Tyto data jsou zpracována a na jejich základě vyhodnocena aktuální trajektorie puku. Více o této metodě bude pojednáno níže.

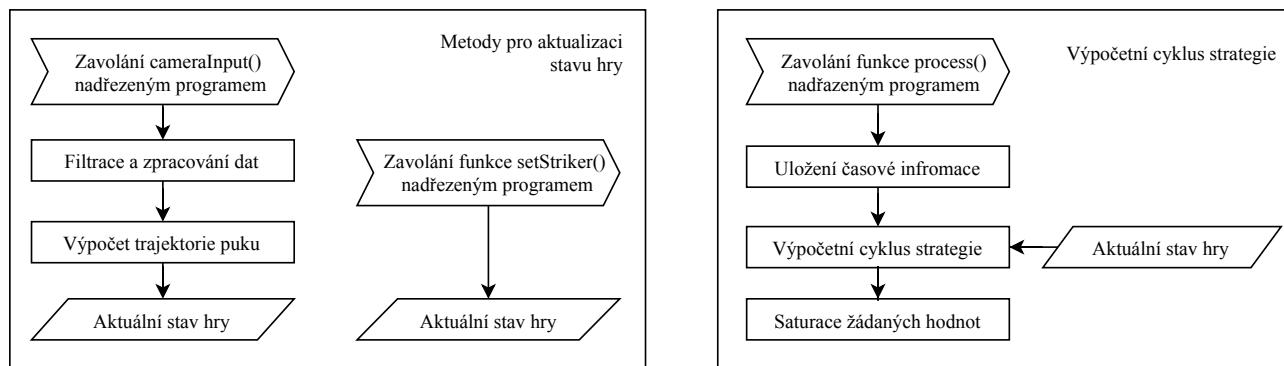
- metoda ***setStriker(pořada, rychlosť)***

Volána nadřazeným programem pokaždé, když jsou k dispozici nová data o pozici nebo rychlosti hokejký robotického hráče.

- metoda ***process(doba kroku)***

Metoda, kterou každá dceřinná třída musí přepsat a ve které je definován průběh výpočetního cyklu strategie. Je volána nadřazeným programem v pravidelných intervalech a frekvence těchto volání reprezentuje výpočetní frekvenci strategie.

Je zřejmé, že výpočetní frekvence spolu s aktualizací stavů hry je v celé míře řízena nadřazeným programem, který modul strategie využívá. Modul strategie je tedy v podstatě „pouze“ zapouzdřená logika výpočtu rozhodnutí pro zadaný stav hry. Vývojový diagram znázorňující průběh řídících metod strategie lze vidět na obrázku 27.



Obrázek 27: Vývojový diagram průběhu řídících metod třídy *BaseStrategy*

### 6.2.1 Zpracování polohy puku

Při obdržení nových dat o poloze puku  $\vec{p}_0$  je z pohledu strategie nutné rozhodnout, zda nové data odpovídají vypočítané trajektorii v minulém cyklu nebo zda nenastal odraz a tím i změna pohybu puku. Její, který značně komplikuje toho rozhodování je právě šum detekovaných poloh z kamery. Proto byl navrhnut algoritmus minimalizující chybné detekovaní odrazu puku založený na analýze historie doposud obdržených dat.

Každá příchozí informace o poloze puku  $\vec{p}_0$  je umístěna do zásobníku historie a je jí přiřazena časová známka  $t_0$  odpovídající času, kdy byly data obdrženy a speciální stavová proměnná *state*, která může nabývat následující stavy:

- ACCURATE

Reprezentuje záznam polohy puku, který lze použít pro výpočet aktuální trajektorie a rychlosti pohybu.

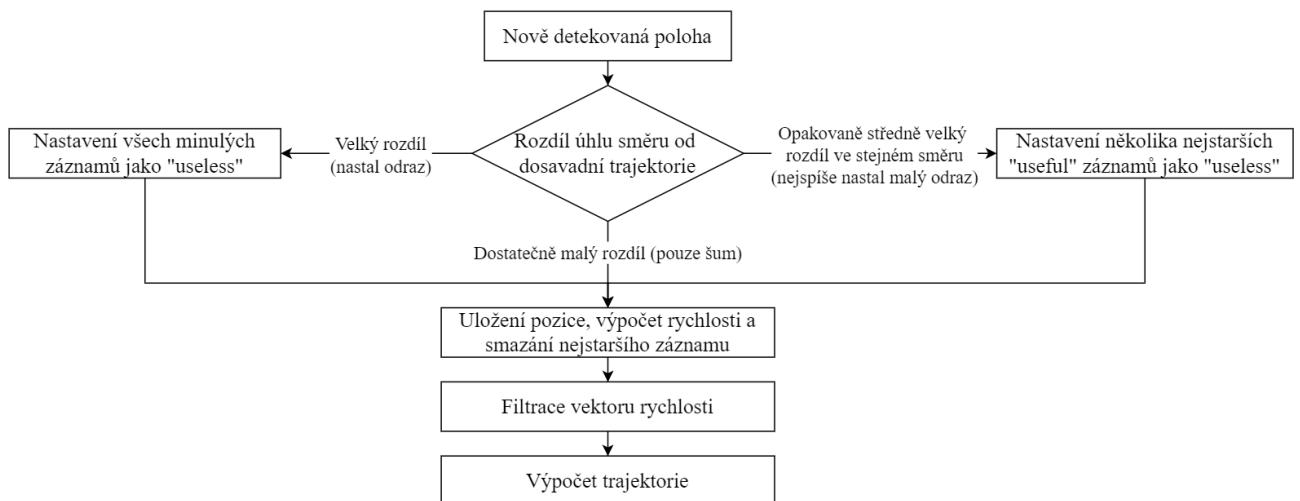
- INACCURATE

Reprezentuje záznam polohy puku, který lze použít pro výpočet aktuální trajektorie a rychlosti pohybu, ale výsledky nemusí být kvůli nesplněným podmínkám přesné (například při pomalu se pohybujícím puku má šum detekovaných poloh mnohonásobně vyšší vliv na vypočítanou trajektorii než při rychlostech vyšších).

- USELESS

Reprezentuje záznam polohy puku, který nelze použít pro výpočet trajektorie ani rychlosti puku. S velkou pravděpodobností tedy nastal odraz puku někdy v čase po tomto záznamu.

Na rozdíl od časové známky se tyto stavy mohou u záznamů v zásobníku změnit. Vývojový diagram algoritmu detekující odraz puku a spravující záznamník lze vidět na obrázku 28.



Obrázek 28: Vývojový diagram algoritmu detekce odrazu puku

Při vložení nové polohy  $\vec{p}_0$  do zásobníku se indexy ostatních záznamů inkrementují  $\vec{p}_i \rightarrow \vec{p}_{i+1}$  a záznam s indexem větším jak maximální velikost záznamníku  $n$  je zahozen. Ideální velikost  $n$  závisí na frekvenci nových příchozích poloh z kamery - tedy frekvenci detekce kamery. Experimentálně byla zjištěna dostatečná velikost zásobníku  $n$  taková, aby byla schopná udržet historii za poslední vteřinu pohybu. Pro kameru detekující polohu puku průměrně 80 krát za vteřinu tomu odpovídá velikost  $n = 80$ .

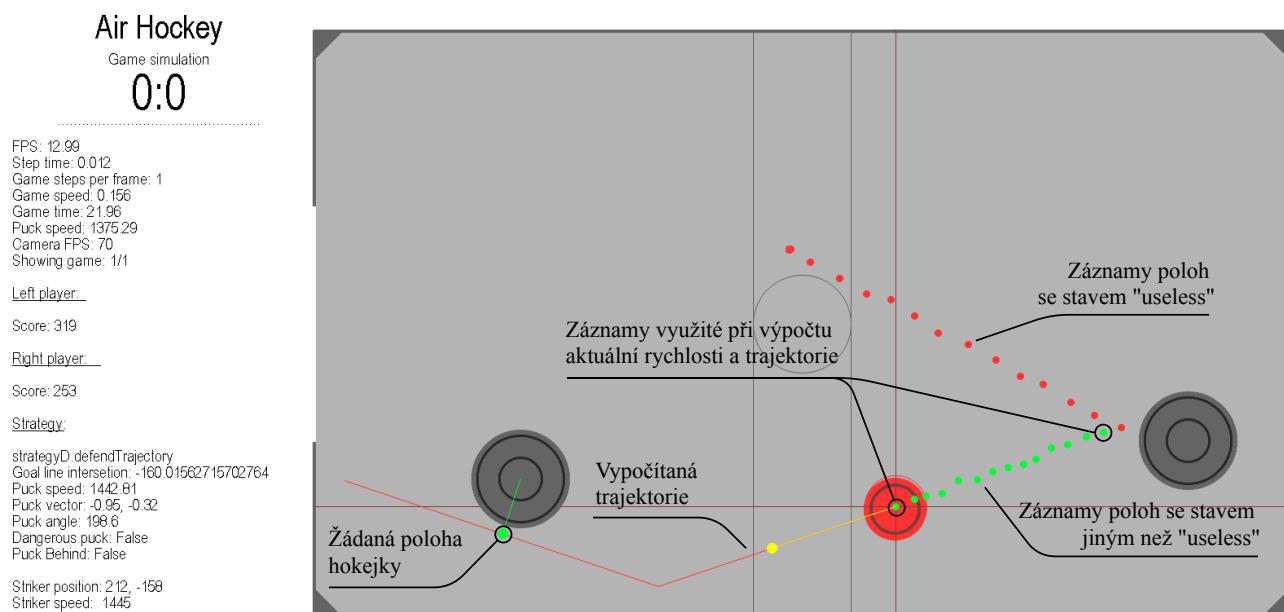
Při výpočtu trajektorie a rychlosti puku se pak prochází zásobník od nejstaršího záznamu  $\vec{p}_n$  po nejnovější  $\vec{p}_0$ . První nalezený záznam se stavem jiným než *USELESS*  $\vec{p}_u$  je následně využit pro výpočet aktuální rychlosti  $\vec{v}$  a směru pohybu puku  $\vec{s}$ . Odečtením vektoru polohy  $\vec{p}_u$  od aktuálně nejnovější polohy  $\vec{p}_0$  (9) je zjištěn takzvaný krokový vektor  $\vec{k}$

$$\vec{k} = \vec{p}_0 - \vec{p}_u, \quad (9)$$

který reprezentuje aktuální směr pohybu puku. Podělením krokového vektoru  $\vec{k}$  rozdílem časových známk nejnovějšího záznamu  $i = 0$  s nalezeným záznamem v záznamníku  $i = u$  pak dostáváme aktuální rychlosť puku definovanou vztahem (10)

$$\vec{v} = \frac{\vec{k}}{t_0 - t_u}, \quad (10)$$

kde  $t_0$  je časová známka nejnovějšího záznamu a  $t_u$  je časová známka záznamu zvoleného pro výpočet. Znázornění detekce odrazu v simulaci lze vidět na obrázku 29.



Obrázek 29: Znázornění detekce odrazu pomocí analýzy historie poloh puku

Následný algoritmus výpočtu trajektorie z takto zpracovaných dat je pak založen na principech popsaných v podkapitole 2.3.3.

### 6.3 Navržené strategie

Celkově byly v rámci práce navrženy čtyři různé strategie s rozdílným stylem hry. Tímto bylo kromě možnosti testování jejich efektivity simulováním her proti sobě umožněno i nastavení požadované strategie při hře proti lidskému protihráči. Obtížnost hry proto nemusela být definována pouze omezením rychlosti pohybu robotické hokejky ale i samotným stylem hrané hry, navozující tak pocit, že se robotický hráč s nastavenou vyšší obtížností stává nejen rychlejší, ale i chytřejší. Stručný popis vlastností jednotlivých strategií lze vidět na obrázku 30. Vývoj strategií byl mimo jiné založen i na znalostech osvědčených herních stylů profesionálních hráčů přiblížených v kapitole 2.2.

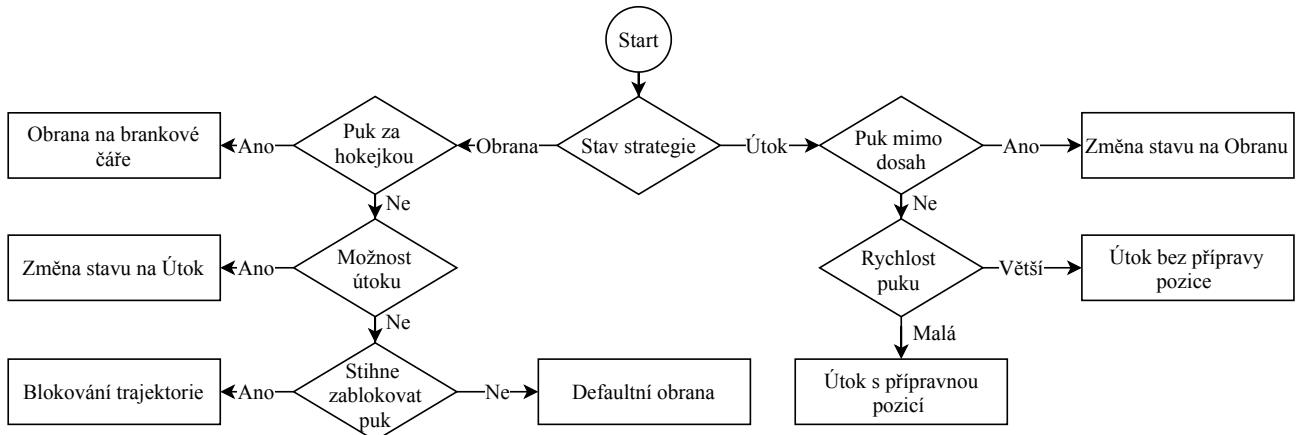
Strategie A	Strategie B	Strategie C	Strategie D
Základní obrana	Základní obrana	Mírně pokročilá obrana	Vysoko pokročilá obrana i útok
Mírně pokročilý útok	Základní útok	Pokročilý útok	Míření přes mantinely
Vnitřní stav ("paměť strategie")	Odhad budoucího stavu hry	Vnitřní stav ("paměť strategie")	Schopnost zpracování puku
		Odhad budoucího stavu hry	Prvky náhody při rozhodování
			Vnitřní stav ("paměť strategie")
			Odhad budoucího stavu hry

Obrázek 30: Navržené typy strategií pro RVH

Vzhledem ke komplexnosti každé ze strategií a výpočtu k nim potřebným bude jejich popis velice zjednodušen. Úplný algoritmus lze najít v přiložených zdrojových souborech.

#### 6.3.1 Strategie A

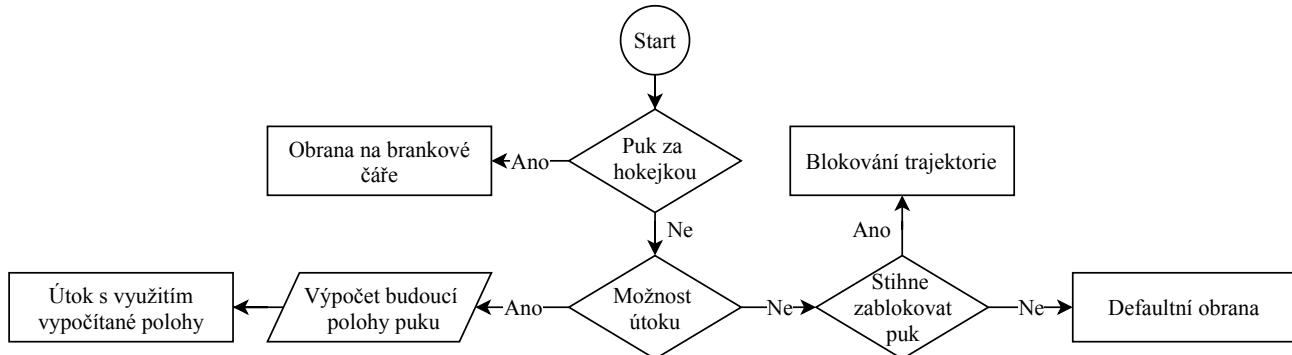
Cílem návrhu Strategie A bylo využití vnitřních stavů pro umožnění rozhodování nejen na základě aktuálního stavu hry, ale i na jeho historii. Využívá základní algoritmy obrany a útoku. Zjednodušený vývojový diagram rozhodování lze vidět na obrázku 31.



Obrázek 31: Vývojový diagram algoritmu rozhodování strategie A

### 6.3.2 Strategie B

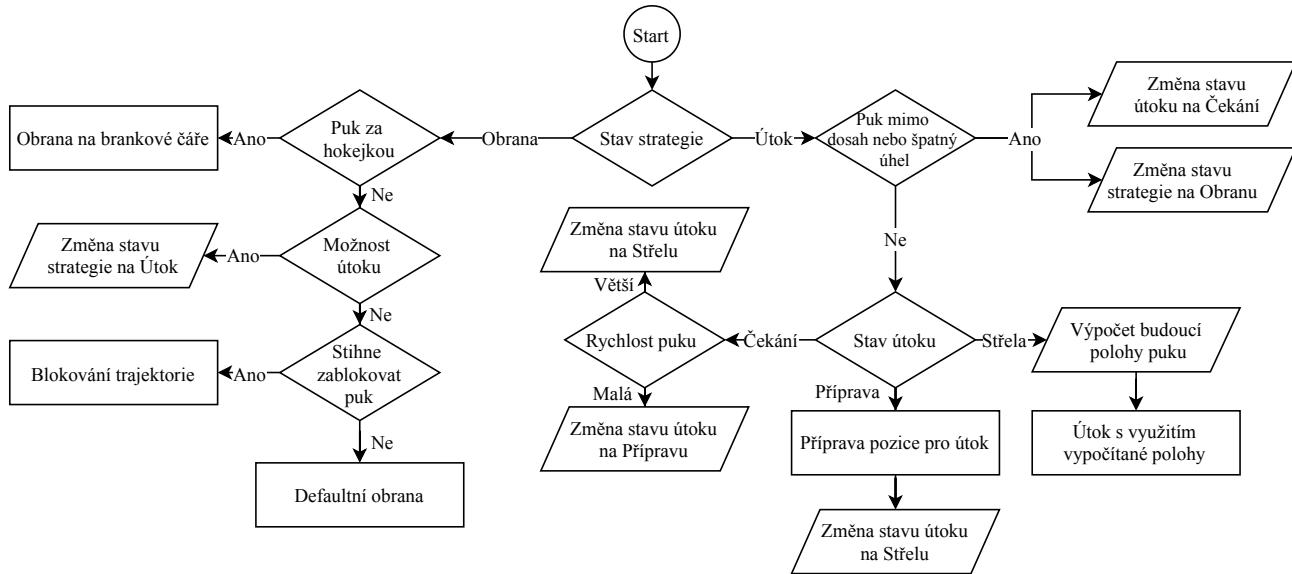
Strategie B oproti strategii A nevyužívá vnitřní stavy a rozhodování tak závisí pouze na aktuálním stavu hry. V každém cyklu ale přepočítává odhadovanou pozici puku v čase, kdy robotická hokejka dosáhne žádané polohy. Tímto je odhadován stav hry v budoucnu a zajištěna tak přesnější interakce s pohybujícím se pukem. Zjednodušený vývojový diagram algoritmu rozhodování lze vidět na obrázku 32.



Obrázek 32: Vývojový diagram algoritmu rozhodování strategie B

### 6.3.3 Strategie C

Strategie C vznikala jako kombinace strategie A a strategie B s dodatečným laděním obranných a útočných algoritmů v simulačním prostředí. Velký důraz byl kladen na využívání odhadované budoucí polohy puku při útočných střelách robotického hráče. Zjednodušený vývojový diagram algoritmu rozhodování lze vidět na obrázku 33.

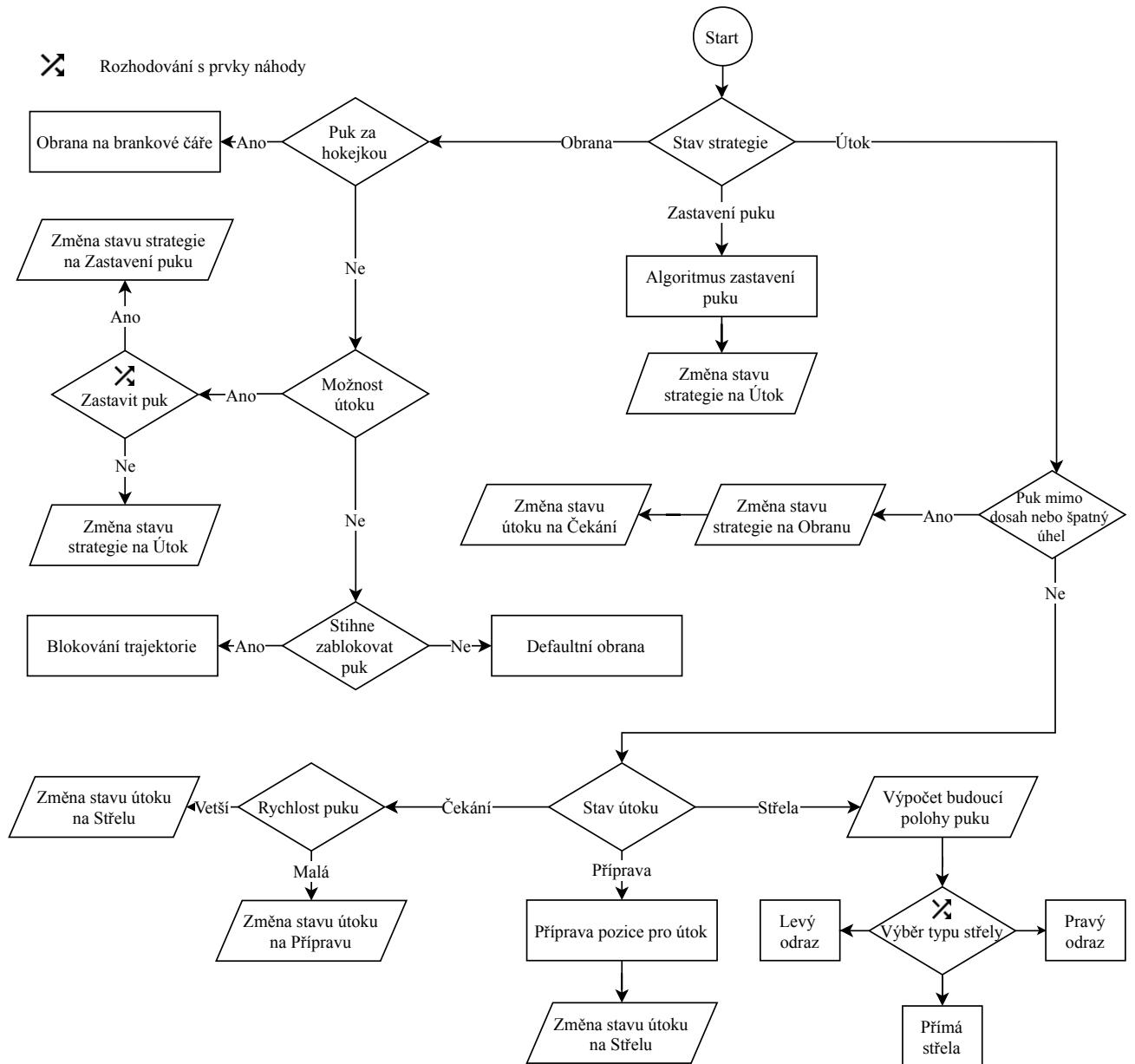


Obrázek 33: Vývojový diagram algoritmu rozhodování strategie C

### 6.3.4 Strategie D

Strategie D vykazuje ze všech strategií nechytrější chování. Oproti strategii C byl výrazně vylepšen algoritmus útoku s přesnějším mířením a možností plánovaného útoku odrazem přes mantinely. Náhodné prvky v rozhodování zajišťují nepředvídatelnost hry a schopnost zastavit pohybující puk pak snadnější přípravu útočné pozice (viz obrázek 5).

Vývoj rozhodovacího algoritmu byl výrazně ovlivněn analýzou stylů hry profesionálních hráčů zmíněných v podkapitole 2.2. Velice zjednodušený vývojový diagram algoritmu rozhodování lze vidět na obrázku 34.



Obrázek 34: Vývojový diagram algoritmu rozhodování strategie D



## 7 Detekce puku

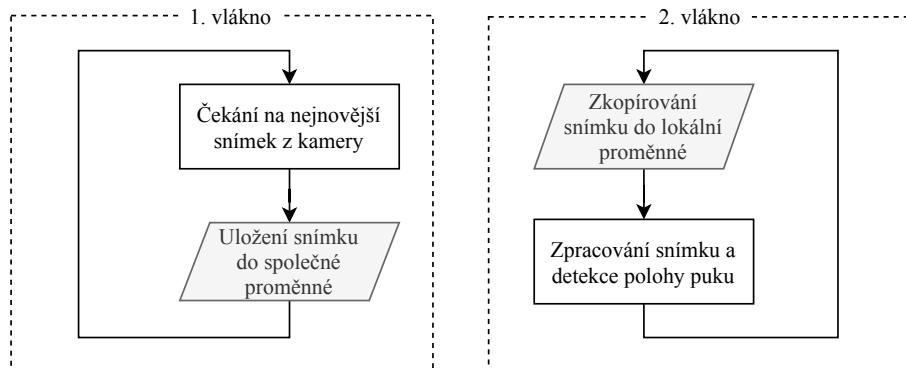
Základem pro dobře hrající strategii je co nejpřesnější a často aktualizovaná informace a aktuální poloze puku. Ta je detekována na herní ploše RVH pomocí kamery připojené k řídící jednotce Raspberry Pi 4B, která jednotlivé snímky zpracovává. Kromě spolehlivé a co nejpřesnější detekce je zároveň nutné detektovat puk s co největší frekvencí. S vyšším rozlišením pořízených snímků roste díky více informacím přesnost detekce, ovšem nutnost zpracování více pixelů výrazně prodlužuje dobu pořízení a zpracování snímku. Iterativním postupem byla zjištěna ideální kombinace rozlišení a rychlosti snímání, která odpovídala hodnotám 320x192 pixelů na snímek respektive 70-80 zpracovaných snímků za vteřinu.

Kromě algoritmu pro detekování puku na snímku hrací plochy byly v rámci práce navrženy pomocné algoritmy pro automatické vyvážení bílé barvy kamery v aktuálních světelných podmínkách a automatické rozpoznaní hledané barvy puku. Tyto pomocné algoritmy výrazně urychlují kalibraci při změně světelných podmínek, ve kterých se RVH nachází respektive při výměně stávajícího puku za puk s jinou barvou. Všechny tyto automaticky zjišťované parametry spolu s dalšími pak bylo navíc umožněno doladit manuálně skrze uživatelské rozhraní, které je více přiblíženo v kapitole 8.

### 7.1 Snímání a detekce puku

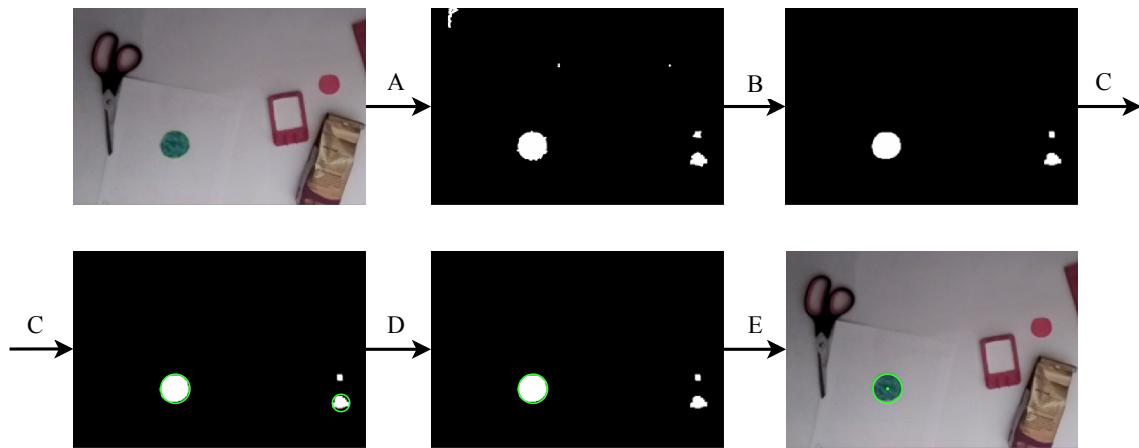
Pro jednoduchou integraci všech softwarových modulů bylo zpracování snímků z kamery (stejně jako ostatní funkcionality RVH) implementováno v jazyce Python. Pro komunikaci s kamerovým modulem byla využita knihovna PiCamera a pro usnadnění rozpoznaní puku na jednotlivých snímcích potom knihovna OpenCV.

Pro maximální využití dostupného výkonu byl proces získávání snímku z kamery a následné zpracování paralelizován. Byly vytvořeny dvě vlákna. První pro obsluhu kamery a ukládání neaktuálnějšího snímku do paměti a druhé pro zpracování neaktuálnějšího snímku uloženého v paměti a detekce polohy puku. Algoritmus detekce tak nemusel při každém cyklu čekat na nový snímek, ale průběžně zpracovával nejnovější snímky v paměti zatímco první vlákno nepřetržitě nejnovější snímek aktualizovalo. Znázorněný proces paralelního zpracování snímku lze vidět na obrázku 35.



Obrázek 35: Proces paralelního snímání a zpracování snímku

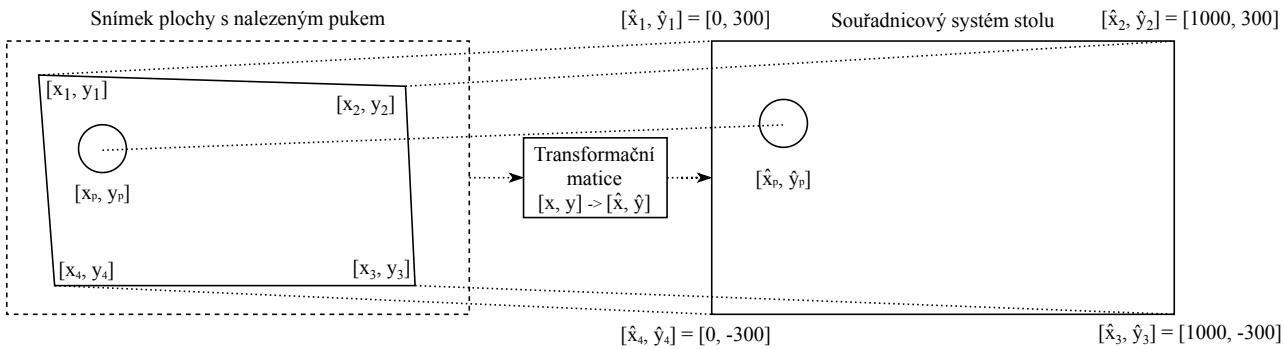
Samotná detekce je založena na algoritmu rozpoznávání objektů pomocí barvy pixelů popsáném v podkapitole 3.1.1. Pro stabilnější výsledky je po vytvoření masky snímku (A) provedena série erozí a dilatací shluků pixelů spadající svou barvou do hledaného intervalu. Tím se odstraní malé shluky, které mohly v masce vzniknout. Pro takto vyfiltrované shluky pak algoritmus nalezne jejich opisující kružnice (C). Kružnice s menšími poloměry než nastavitelná hodnota počtu pixelů jsou ignorovány. Střed největší nalezené kružnice je potom požadován za souřadnice polohy nalezeného puku v pixelech (D). Celý postup detekce polohy puku z pořízeného snímku je vidět na obrázku 36.



Obrázek 36: Postup detekce polohy puku ze snímku.

Výsledné souřadnice polohy v pixelech jsou poté přepracovány do souřadnicového systému stolu pomocí transformační matice (viz obrázek 37). Ta je vytvářena při mapování polohy rohů hřiště na snímcích z kamery. Více o implementované kalibraci polohy hřiště bude zmíněno v kapitole 8.

Poloha je nakonec filtrována speciálním filtrem navrženým a využívaným již v bakalářské práci autora z roku 2018 [35]. Filtr využívá dodatečnou informaci o maximální hodnotě šumu, díky které oproti klasické dolní propusti rychle reaguje na změny filtrované hodnoty, které se šumem nesouvisí.



Obrázek 37: Transformace souřadnic polohy puku v pixelech do souřadnicového systému RVH.

## 7.2 Automatické vyvážení bílé

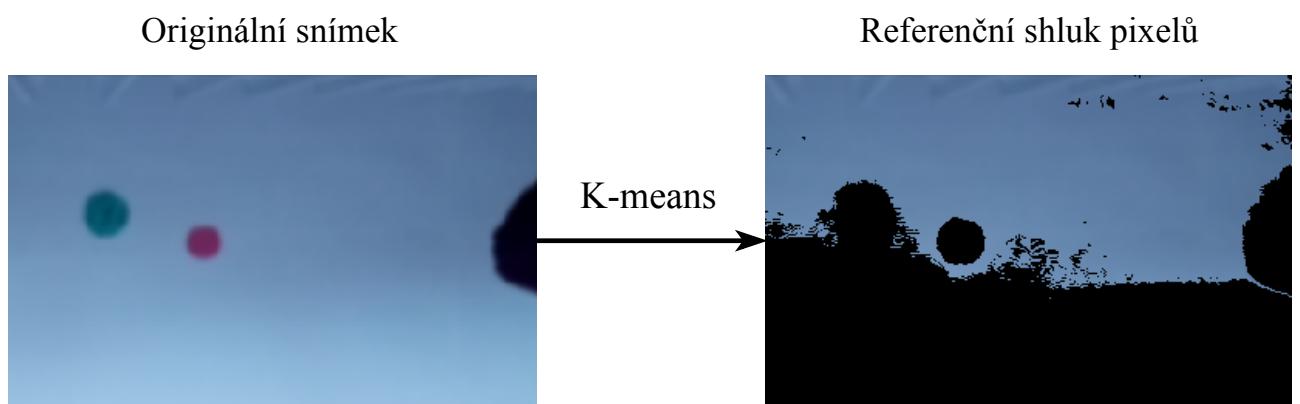
Použitý kamerový modul sám o sobě automatické vyvážení bíle využíval. Výsledky byly ale zejména v umělém osvětlení nedostatečné. Nejen, že bílá herní deska se na snímcích často zbarvovala do červena, ale zároveň se v průběhu snímání vyvážení neustále měnilo, což pro účely stabilního vyhledávání neměnící se barvy nebylo vhodné.

Byl proto navržen alternativní způsob automatického vyvážení bíle barvy, které po dokončení až do dalšího manuálního spuštění funkce uzamklo nalezené hodnoty. Hodnoty násobicích složek (hodnoty, kterými se násobí úroveň modré a červené barvy oproti barvě zelené zachycené senzorem kamery) se tak nastavily vždy při počáteční kalibraci stolu a po dobu snímání zůstávaly konstantní.

Základem algoritmu byl předpoklad, že kamera vždy sleduje plochu, kde většina pixelů odpovídá ideální bílé barvě. To v případě kamery umístěné nad bílou herní plochou RVH platí vždy. Kamera tak má vždy k dispozici přesnou referenci pro co nejpřesnější vyvážení.

Matice RGB pixelů reprezentující aktuální snímek byla upravena tak, že od hodnot RGB každého pixelu byla odečtena jeho G složka. Například pixel s RGB vektorem [100, 80, 70] by po této úpravě byl roven vektoru [20, 0, -10]. Takto upravená matice nereprezentovala barvy jednotlivých pixelů, ale rozdíl R a B složek od G složky. V takové matici vektor [0, 0, 0] reprezentuje pixel s libovolným odstínem bílé barvy (bílá, šedá nebo černá), ale bez jakéhokoliv zbarvení. Pro nalezení referenčního pixelu byl pak využit K-means algoritmus, který všechny takto upravené pixely rozdělil na 8 shluků (hodnota s nejspolehlivějšími výsledky zjištěná experimentálně). Shluk s největším počtem pixelů byl potom označen jako *referenční shluk* a z něho náhodně vybraný pixel jako *referenční pixel*.

V několika iteracích byly pomocí referenčního pixelu upravovány hodnoty vyvážení bíle tak, že se k modré a červené složce přičítala malá část rozdílu R respektive B složky od G složky vektoru referenčního pixelu. V každé iteraci se potom celý postup opakoval až do dostatečného přiblížení hodnoty referenčního vektoru k hodnotě [0, 0, 0]. Tímto bylo zajištěno správné nalezení referenční bílé barvy i při špatných světelných podmírkách. Nalezení referenčního shluku po aplikaci algoritmu lze vidět na obrázku 38.

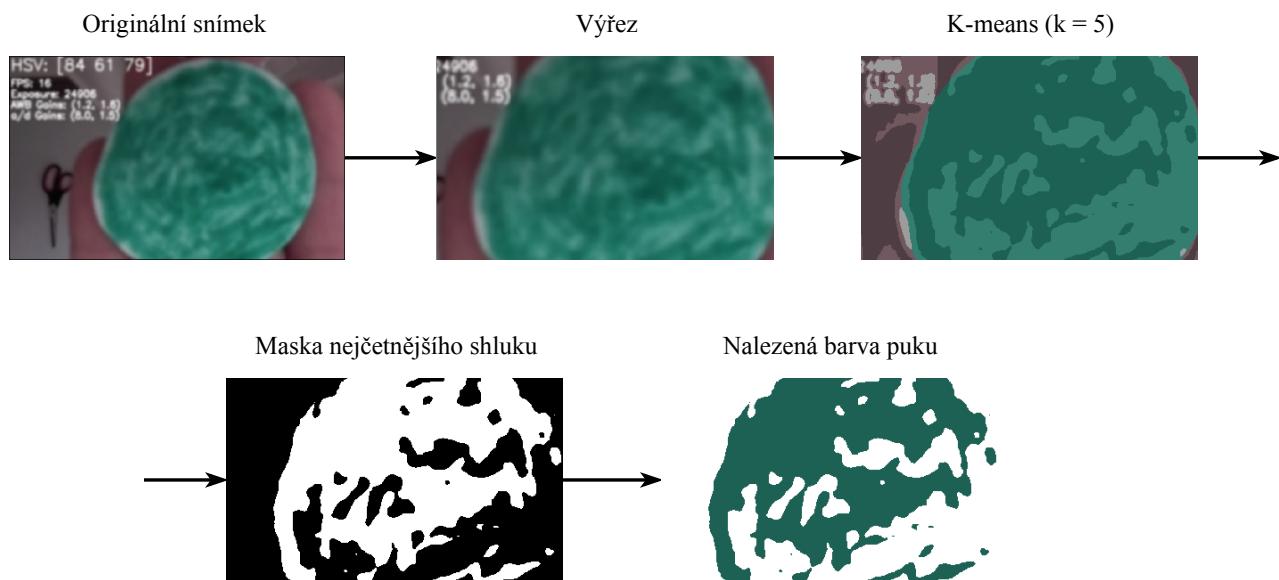


Obrázek 38: Nalezení referenčního shluku pomocí K-means pro automatické vyvážení bílé barvy.

### 7.3 Automatické rozpoznání barvy puku

Podobně jako algoritmus automatického vyvážení bílé barvy i algoritmus automatického rozpoznání barvy puku vznikl zejména pro výrazné urychlení počáteční kalibrační fáze RVH při změně okolních světelných podmínek. Algoritmus obdobně využívá K-means pro zjištění nejčetnějšího shluku pixelů s podobnou barvou (viz obrázek 39) ale s tím rozdílem, že neupravuje hodnoty RGB vektoru jednotlivých pixelů jako tomu bylo zapotřebí v 7.2. Pro správné rozpoznání hledané barvy je proto nutné puk dostatečně přiblížit kameře. Důležité je také dodržet posloupnost akcí, tedy před samotným rozpoznáním barvy puku automaticky nastavit vyvážení bílé barvy pomocí algoritmu 7.2.

Po spuštění algoritmu skrze uživatelské rozhraní program vyčká několik vteřin pro umožnění přiblížení puku ke kameře. Poté je na částečný výřez vyfoceného snímku aplikován postup popsaný výše. Výsledná barva nejčetnějšího shluku je považována za hledanou barvu puku  $[R_p, G_p, B_p]$ . Ta je převedena na HSV reprezentaci barev  $[H_p, S_p, V_p]$  a intervaly hledané barvy nastaveny na spodní hodnotu:  $[H_p - 15, S_p - 75, V_p - 75]$  a horní hodnotu:  $[H_p + 15, S_p + 75, V_p + 75]$ . Tento interval hodnot je následně využíván při detekci hledané barvy.



Obrázek 39: Postup analýzy barvy puku.

## 8 Softwarová implementace

Implementace ovládacího softwaru spolu s vytvořením uživatelského rozhraní pro intuitivní ovládání prvků RVH skrze dotykový displej sice nebylo vymezeno v cílech práce, ale bylo neméně důležitou součástí projektu. Celý řídící software běžel na mikropočítači Raspberry pi modelu 4B v jazyce Python, s připojeným obrazovým výstupem k dotykovému 7" displeji od firmy Waveshare. Po startu systému se řídící program spolu s uživatelským rozhraním automaticky spustil na celou obrazovku displeje bez nutnosti manuálního zásahu. Pro maximální využití dostupného výkonu bylo co nejvíce úkonů napříč celou softwarovou implementací RVH paralelizováno do vlastních výpočetních vláken pomocí knihovny *Threading*. Po ošetření korektní výměny dat mezi vlákny tak všechny hlavní řídící procesy běžely nezávisle na sobě a využívaly veškerý dostupný výkon procesoru.

V této kapitole bude stručně popsána konfigurace řídící jednotky Raspberry Pi, nastíněna celková struktura řídícího softwaru spolu s vysvětlením pracovního cyklu programu, přiblíženo uživatelské rozhraní navržené od základu pro RVH a vysvětlena komunikace mezi řídící jednotkou a Arduinem, zajišťující správu hardwarových prvků stolu, čímž se zabývala práce Dominika Jaška [35].

### 8.1 Konfigurace Raspberry Pi 4B

instalace knihoven

nastavení displeje

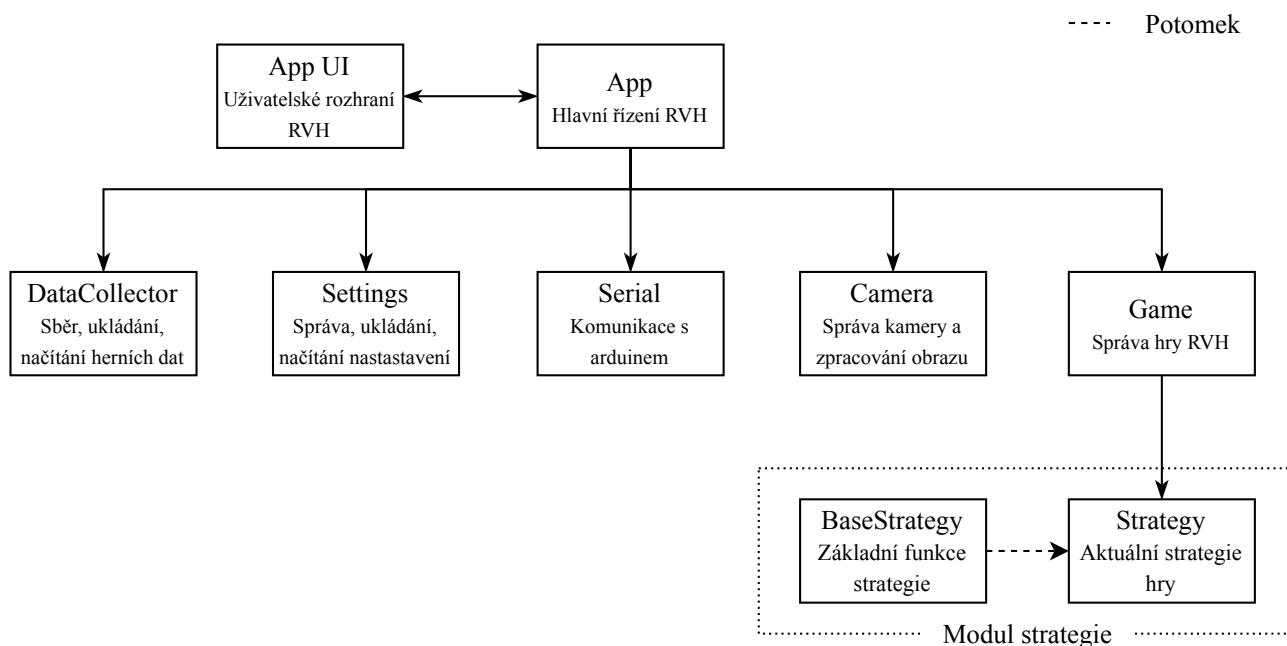
přetaktování

start se spuštěním systému

## 8.2 Struktura programu

Struktura řídícího softwaru RVH se v některých oblastech podobala struktuře simulačního softwaru, na rozdíl od simulace ale bylo nutné komunikovat s Arduinem, integrovat komplexní uživatelské rozhraní pro plnou kontrolu nad všemi důležitými aspekty stolu a spravovat modul kamery spolu se zpracováním přijímaných snímků.

Logika programu byla podobně jako u simulace rozdělena mezi jednotlivé třídy spravující určitou ucelenou část problematiky, jak je vidět na obrázku 40.

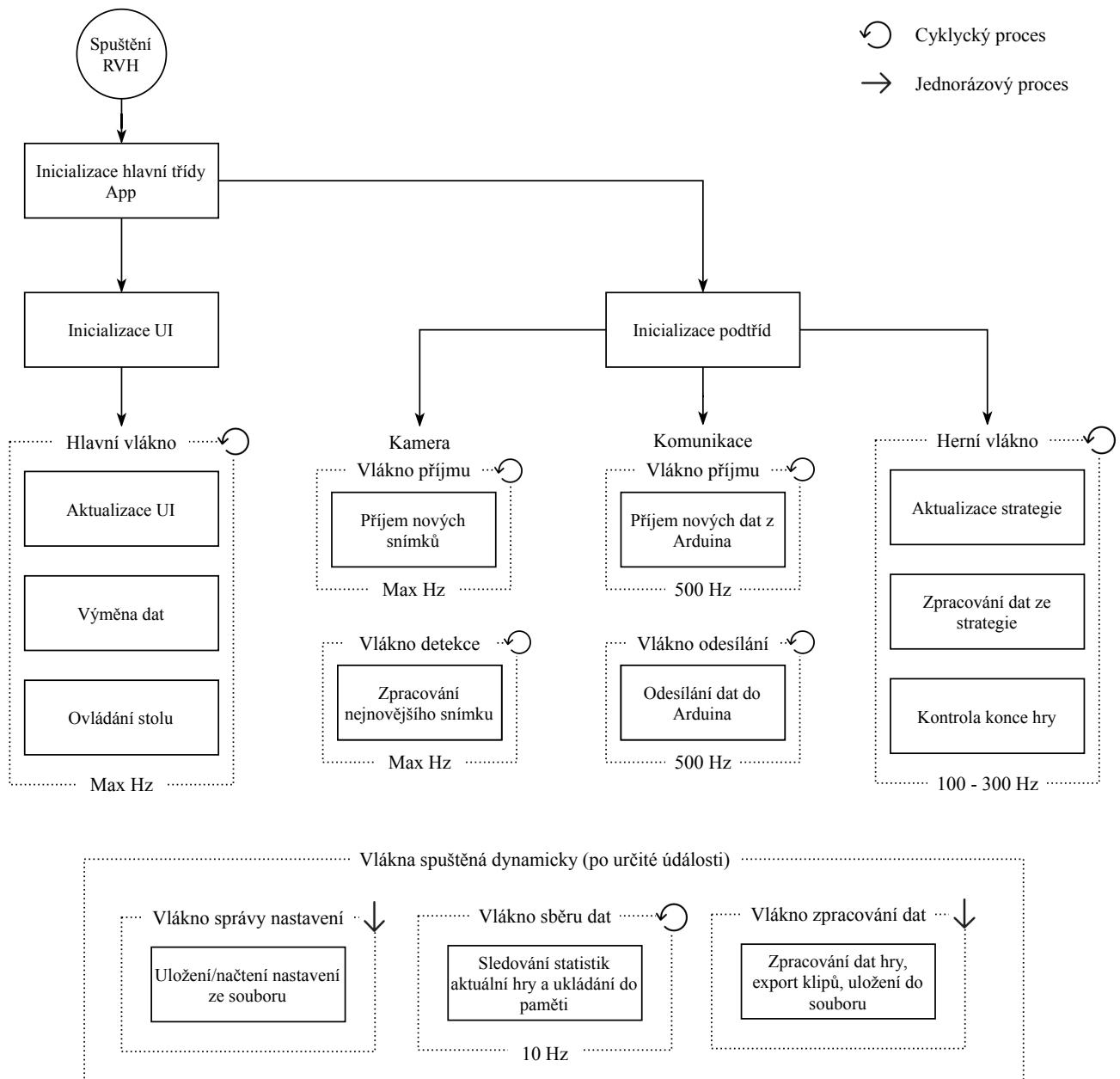


Obrázek 40: Znázornění struktury řídícího softwaru RVH.

Hlavní třída *App* je specifická tím, že integruje logiku hlavní smyčky programu spolu s ovládáním uživatelského rozhraní. Více bude o implementaci uživatelského rozhraní pojednáno v podkapitole 8.3. Zároveň má hlavní třída přístup ke všem podtřídám RVH a zajišťuje tak komunikaci mezi nimi buď přeposíláním dat, nebo pomocí odkázání referencí. Například ke třídě *Settings*, obsahující veškeré nastavitelné parametry RVH, mají přístup všechny ostatní podtřídy skrz referenci na její instanci uloženou pod hlavní třídou *App*.

Logika každé podtřídy využívá jedno nebo více vlastních vláken pro paralelní zpracování jednotlivých úkonů. Třída *App* pak zajišťuje bezkonfliktní výměnu důležitých dat. Kromě efektivnějšího využití dostupného výkonu má takto paralelní výpočetní struktura obrovskou výhodu ve vzájemné nezávislosti časové náročnosti jednotlivých procedur. Například třída *Serial*, která zasílá a přijímá data z pomocné řídící jednotky (Arduina) spravující hardwarové prvky stolu, nutně potřebuje kontrolovat sériovou komunikaci s vysokou frekvencí. Pokud ale Arduino začne posílat vyjímečně dlouhou zprávu, celý přenos zabere mnohonásobně delší dobu než byla původní doba cyklu kontroly komunikace. V případě, že by třída *Serial* neběžela pod vlastním

vláknem, celé řízení stolu by po dobu přenosu zprávy „zamrzlo“. Naopak při využití paralelního zpracování může jakákoli třída „zamrznout“ na jakkoliv dlouhou dobu bez omezení výpočetní rychlosti ostatních tříd. Další výhodou je umožnění probíhání výpočetních cyklů každé třídy s různou frekvencí, takže například třída *Camera* může detekovat polohu puku na nových snímcích 70 krát za vteřinu, zatímco třída *Game* aktualizovat žádanou polohu robotické hokejky 170 krát za vteřinu. Velice zjednodušené znázornění výpočetního procesu vláken celého programu lze vidět na obrázku 41.



Obrázek 41: Znázornění výpočetního procesu vláken řídícího programu RVH

## 8.3 Uživatelské rozhraní

Uživatelské rozhraní zajišťující grafický výstup stavů RVH a možnost interakce s ovládacími prvky vznikalo za pomocí python knihovny *Kivy*, která se zaměřuje na usnadnění tvorby UI s moderním designem a podporou dotykového ovládání. Nebyla však využita žádná šablona a veškeré prvky byly designovány od základu pro využití pouze na RVH. Díky tomu mohlo vzniknout intuitivní a na pohled příjemné prostředí pro rychlé ovládání aktuální hry, nastavení různých parametrů stolu, kontrolu stavů nebo i přehled statistik her a možnost procházení historie odehraných zápasů spolu s nahranými krátkými klipy důležitých okamžiků hry.

Kvůli rozsáhlé komplexnosti a obsáhlosti tématu a faktu, že vytvoření uživatelského rozhraní nebylo tématem cílů práce bude popis výsledného produktu velice zjednodušen. Pro pochopení všech okolností, které souviseli s návrhem a implementací UI pro RVH a přehled designu jednotlivých sekcí rozhraní lze nahlédnout do přiloženého zdrojového kódu respektive na fotky stránek v příloze .

### 8.3.1 Struktura programu a základy *Kivy*

Knihovna *Kivy* je v základu velice jednoduchá. Kód potřebný pro vytvoření základní aplikace zobrazující text uprostřed okna vypadá následovně:

```
import kivy
from kivy.app import App
from kivy.uix.label import Label
class MyApp(App):

    def build(self):
        return Label(text='Hello world')

    if __name__ == '__main__':
        MyApp().run()
```

**Algoritmus 3:** Základní aplikace vytvořená pomocí knihovny Kivy.

Za předpokladu správně nainstalované knihovny se po spuštění skriptu otevře okno s textem „Hello Word“ uprostřed. Metoda build svojí návratovou hodnotou definuje, co se v okně zobrazí. Místo widgetu *Label* tak lze nahradit libovolný jiný *Kivy* widget, který sám může obsahovat další widgety. Ve třídě *MyApp* pak můžou být umístěny i metody obsahující logiku aplikace. Kompletní popis postupu vytváření aplikací pomocí knihovny *Kivy* lze nalázt v online dokumentaci knihovny [34]. Z příkladu je patrná možnost kombinace kódu definující vzhled spolu s logikou aplikace. Zatímco pro jednoduché projekty může být taková struktura výhodou, pro komplexnější aplikace lze s výhodou část definující vzhled a rozvržení od logiky oddělit. *Kivy* pro tyto účely využívá speciální .kv soubor definující vzhled pomocí zjednodušené stromové struktury. Ten se pak interně zpracovává a překládá na syntaxi Pythonu. Pro více informací

lze opět nahlédnou do oficiální dokumentace *Kivy* knihovny [34]. Jednoduchý příklad definice widgetu s dvěma tlačítky bez přidělené funkce pomocí .kv souboru:

MyRootWidget:

BoxLayout:

Button:

Button:

**Algoritmus 4:** Příklad definice widgetu MyRootWidget se dvěma tlačítky pomocí .kv souboru

V tomto případě by pro spuštění aplikace bylo potřeba v hlavním skript zobrazený v algoritmu 3 mírně upravit:

```
import kivy
from kivy.app import App
from kivy.uix.label import Label
class MyApp(App):

    def build(self):
        return Label(text='Hello world')

    if __name__ == '__main__':
        MyApp().run()
```

**Algoritmus 5:** Hlavní skript aplikace při navázání na .kv soubor definující vzhled widgetu MyRootWidget.

### 8.3.2 Rozvržení a design

## 8.4 Komunikace s řízením pohonů

Threading, rozvržení jednotlivých modulu, celkový diagram programu nevím



## 9 Závěr

Závěr ke každému k cílů

Porovnání skore strategii

Spolupráce s Domem



## Seznam obrázků

1	Profesionální vzdušný hokej využívaný na turnajích [1] . . . . .	15
2	Standardizované rozměry vzdušného hokeje s doporučenými rozměry místonosti [3]	16
3	Standardní hokejky s pukem pro stolní hru vzdušný hokej [5, 6] . . . . .	16
4	Znázornění trojúhelníkové obrany . . . . .	17
5	Znázornění pohybu hokejky při zpracovávání pomalé střely . . . . .	18
6	Znázornění falešné střely . . . . .	19
7	Fotka experimentální sestavy pro sledování pohybu očí lidského hráče [9] . . . . .	20
8	Schéma testovacích trajektorií puku při sledování pohybu očí lidského hráče . . . . .	20
9	Znázornění sil působící na puk pohybující se po hrací desce [34] . . . . .	22
10	Znázornění odrazu puku od mantinelu . . . . .	23
11	Znázornění odrazu puku od pohybující se hokejky . . . . .	23
12	Maska snímku hrací plochy vzdušného hokeje . . . . .	25
13	Původní obrázek a maska obrázku při hledání objektu za pomocí barvy [20] . . . . .	26
14	Nalezená kružnice na obrázku plechovky pomocí Hoghovi transformace [21] . . . . .	27
15	Detekce a klasifikace objektů neuronovou sítí YOLO [24] . . . . .	27
16	Postup strukturou konvoluční neuronové sítě pro klasifikaci objektu [26] . . . . .	28
17	Průmyslové kamery série 7000 společnosti Cognex . . . . .	29
18	Příklad web kamery zapojené do mikropočítače Raspberry Pi [29] . . . . .	29
19	Raspberry Pi s připojeným modulem kamery [31] . . . . .	30
20	Princip zapojení a uspořádání infračervených LED diod a tranzistorů [35] . . . . .	31
21	Simulační program pro vývoj a analýzu strategie RVH . . . . .	35
22	Znázornění struktury simulačního softwaru vzdušného hokeje . . . . .	36
23	Vývojový diagram simulačního softwaru vzdušného hokeje . . . . .	37
24	Možné situace při kontrole kolize typu kruh-úsečka . . . . .	39
25	Znázornění simulace snímání a detekce puku. . . . .	40
26	Zobecněný UML diagram dědičnosti tříd modulu strategie . . . . .	41
27	Vývojový diagram průběhu řídících metod třídy <i>BaseStrategy</i> . . . . .	42
28	Vývojový diagram algoritmu detekce odrazu puku . . . . .	43
29	Znázornění detekce odrazu pomocí analýzy historie poloh puku . . . . .	44
30	Navržené typy strategií pro RVH . . . . .	45
31	Vývojový diagram algoritmu rozhodování strategie A . . . . .	45
32	Vývojový diagram algoritmu rozhodování strategie B . . . . .	46
33	Vývojový diagram algoritmu rozhodování strategie C . . . . .	46
34	Vývojový diagram algoritmu rozhodování strategie D . . . . .	47
35	Proces paralelního snímání a zpracování snímku . . . . .	49
36	Postup detekce polohy puku ze snímku. . . . .	50
37	Transformace souřadnic polohy puku v pixelech do souřadnicového systému RVH. . . . .	50
38	Nalezení referenčního shluku pomocí K-means pro automatické vyvážení bílé barvy. . . . .	51
39	Postup analýzy barvy puku. . . . .	52
40	Znázornění struktury řídícího softwaru RVH. . . . .	54
41	Znázornění výpočetního procesu vláken řídícího programu RVH . . . . .	55

## Seznam tabulek

### Seznam algoritmů

1	Řešení kolize typu kruh-kruh . . . . .	38
2	Řešení kolize typu kruh-úsečka . . . . .	39
3	Základní aplikace vytvořená pomocí knihovny Kivy. . . . .	56
4	Příklad definice widgetu MyRootWidget se dvěma tlačítky pomocí .kv souboru .	57
5	Hlavní skript aplikace při navázání na .kv soubor definující vzhled widgetu MyRootWidget. . . . .	57

## Seznam použité literatury

- [1] Tournament Pro Air Hockey Table. In: Gold-standard-games [online]. 3020 N. Water Street Bay City [cit. 2020-03-22]. Dostupné z: <http://gold-standard-games.com/tournament-pro-air-hockey-table.shtml>.
- [2] FIALA, Dominik. Airhockey\_pravidla. Airhockey [online]. 2011 [cit. 2020-03-22]. Dostupné z: [http://www.airhockey.cz/wp-content/uploads/2011/03/Airhockey\\_pravidla.pdf](http://www.airhockey.cz/wp-content/uploads/2011/03/Airhockey_pravidla.pdf)
- [3] Regulation Air Hockey Table Dimensions. Game Table Planet [online]. Harrisburg [cit. 2020-03-22]. Dostupné z: <https://gametableplanet.com/regulation-air-hockey-table-dimensions/>
- [4] What is the Regulation Air Hockey Puck Size? Game Table Planet [online]. Harrisburg [cit. 2020-03-23]. Dostupné z: <https://gametableplanet.com/what-is-the-regulation-air-hockey-puck-size/>
- [5] Plastic Mallet Pusher Puck. In: Online Shopping UAE [online]. [cit. 2020-03-23]. Dostupné z: <https://bit.ly/2vJGUZ1>
- [6] Mini Air Hockey Pusher Mallet [online]. In: . [cit. 2020-03-23]. Dostupné z: <https://cz.pinterest.com/pin/734860864169276331/>
- [7] ROBBINS, Mark. How to Play Air Hockey. In: Youtube [online]. 2015 [cit. 2020-03-24]. Dostupné z: <https://www.youtube.com/watch?v=jAYcpvd8jGE>
- [8] Air Hockey Strategies. Bubble & Air Hockey [online]. [cit. 2020-03-24]. Dostupné z: <https://www.bubbleairhockey.com/air-hockey-strategy.html>
- [9] OGAWA, M, K IKEUCHI, Y SATO, S KUDOH, T TOMIZAWA, T SUEHIRO a S SHIMIZU. Towards air hockey robot with tactics - Statistical analysis from measurement of eye movement. In: 2012 IEEE International Conference on Mechatronics and Automation [online]. IEEE, 2012, s. 34-39 [cit. 2020-04-02]. DOI: 10.1109/ICMA.2012.6282343. ISBN 9781467312752.
- [10] W. SPONG, Mark. Impact controllability of an air hockey puck. Systems & Control Letters [online]. Elsevier B.V, 2001, 42(5), 333-345 [cit. 2020-04-05]. DOI: 10.1016/S0167-6911(00)00105-5. ISSN 0167-6911.
- [11] KOPECKÝ, David. AUTOMATICKÝ VZDUCHOVÝ HOKEJ. Praha, 2017. Bakalářská práce. České vysoké učení technické. Vedoucí práce Zemánek Jiří.
- [12] LEONG, Eric. An Approach to Line-Circle Collision Detection and Response. 2009. William A. Shine Great Neck South High School.
- [13] JAMIESON, Darran. Simulating Circle-Circle Collisions. Evanto Tuts [online]. 27 Sep 2012 [cit. 2020-04-06]. Dostupné z: <https://gamedevelopment.tutsplus.com/tutorials/when-worlds-collide-simulating-circle-circle-collisions--gamedev-769>

- [14] BARA, Matthew, Sneha GOLLAMUDI a Samuel WILLIAMS. Developing an Air Hockey Game in LabVIEW. In: 2017 25th International Conference on Systems Engineering (ICSEng) [online]. IEEE, 2017, 2017-, s. 333-336 [cit. 2020-04-06]. DOI: 10.1109/ICSEng.2017.65.
- [15] BISHOP, B.E a M.W SPONG. Vision-based control of an air hockey playing robot. IEEE Control Systems [online]. IEEE, 1999, 19(3), 23-32 [cit. 2020-04-08]. DOI: 10.1109/37.768537. ISSN 1066-033X.
- [16] MARKUŠ, Nenad, Miroslav FRLJAK, Igor PANDŽIĆ, Jörgen AHLBERG a Robert FORCHHEIMER. Object Detection with Pixel Intensity Comparisons Organized in Decision Trees. ArXiv.org [online]. Ithaca: Cornell University Library, arXiv.org, 2014 [cit. 2020-04-08]. Dostupné z: <http://search.proquest.com/docview/2084910553/>
- [17] ROSEBROCK, Adrian. OpenCV and Python Color Detection. PyImageSearch [online]. August 4, 2014 [cit. 2020-04-08]. Dostupné z: <https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/>
- [18] ROSEBROCK, Adrian. Ball Tracking with OpenCV. In: Learn OpenCV [online]. FEBRUARY 11, 2019 [cit. 2020-04-08]. Dostupné z: <https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/>
- [19] SADEKAR, Kaustubh. Color Detection and Segmentation with OpenCV. In: Learn OpenCV [online]. FEBRUARY 11, 2019 [cit. 2020-04-08]. Dostupné z: <https://www.learnopencv.com/invisibility-cloak-using-color-detection-and-segmentation-with-opencv/>
- [20] Color Detection Using HSV Color Space. In: MathWorks [online]. 24 Jan 2008 [cit. 2020-04-08]. Dostupné z: <https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/18440/versions/1/screenshot.jpg>
- [21] ROSEBROCK, Adrian. Detecting Circles in Images using OpenCV and Hough Circles. In: PyImageSearch [online]. July 21, 2014 [cit. 2020-04-08]. Dostupné z: <https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/>
- [22] KAZÍK, Martin. Houghova transformace pro detekci kružnic. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií, 2009.
- [23] RIEKE, Johannes. Object detection with neural networks. In: Towards data science [online]. Jun 12, 2017 [cit. 2020-04-08]. Dostupné z: <https://towardsdatascience.com/object-detection-with-neural-networks-a4e2c46b4491>
- [24] You only look once (YOLO) : unified real time object detection. In: SlideShare [online]. [cit. 2020-04-09]. Dostupné z: <https://www.slideshare.net/AshishKumar207/you-only-look-once-yolo-unified-real-time-object-detection>

- [25] MENEZES, Richardson Santiago Teles, Helton MAIA a Rafael Marrocos MAGALHAES. Object Recognition Using Convolutional Neural Networks. IntechOpen [online]. May 6th 2019 [cit. 2020-04-09]. DOI: 10.5772/intechopen.89726. Dostupné z: <https://www.intechopen.com/books/recent-trends-in-artificial-neural-networks-from-training-to-prediction/object-recognition-using-convolutional-neural-networks>
- [26] SAHA, Sumit. A Comprehensive Guide to Convolutional Neural Networks. In: Towards data science [online]. Dec 15, 2018 [cit. 2020-04-09]. Dostupné z: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- [27] SIJKOVÁ, Simona. Robotický stolní fotbal - optimalizace snímání hrací plochy. Vysoké učení technické v Brně. Fakulta strojního inženýrství, 2018.
- [28] In-Sight 7000 Series Vision System, ©2018. Cognex [online]. Cognex Corporation [cit. 2018-04-20]. Dostupné z: <https://www.cognex.com/downloads/in-sight-7000-seriesvision-system-18659>
- [29] MAHESH. Real time Surveillance System using Raspberry Pi 2 and Webcam. Rhydolabz [online]. June 3, 2016 [cit. 2020-04-12]. Dostupné z: <https://www.rhydolabz.com/wiki/?p=16181>
- [30] Raspberry Pi camera module. Penguin Tutor [online]. 29 May 2013 [cit. 2020-04-12]. Dostupné z: <http://www.penguintutor.com/news/raspberrypi/raspi-camera>
- [31] Adafruit Raspberry Pi Camera Board v2. In: Mouser [online]. [cit. 2020-04-12]. Dostupné z: <https://cz.mouser.com/new/adafruit/adafruit-raspberry-pi-camera-board-v2/>
- [32] SLÁMA, O. Robotický stolní fotbal – řízení herních os. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2018. 71 s. Vedoucí bakalářské práce Ing. Roman Parák.
- [33] Alsalihi, A.; Najjar, K.; Van Scoy, B.; Zifer, J. Automated Foosball Table [PDF dokument]. The University of Akron, November 28, 2011 [cit. 2020-04-12]. Supervisor Dr. Hartley, Dr. Tran. Dostupný z: <<https://www.uakron.edu/dotAsset/1e2fb3d4-8c59-475e-9473-ed98b2504f17.pdf>>.
- [34] Programming Guide: Kivy Basics. Kivy [online]. [cit. 2020-06-12]. Dostupné z: <https://kivy.org/doc/stable/guide/basic.html>
- [35] JAŠEK, Dominik. Vzdušný hokej - realizace a návrh automatického hráče. Brno, 2020. Dostupné také z: <https://www.vutbr.cz/studenti/zav-prace/detail/125432>. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Radomil Matoušek.