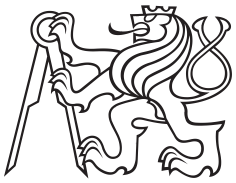
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Automatický vzduchový hokej (Air Hockey)

David Kopecký

Vedoucí: Ing. Jiří Zemánek
Obor: Kybernetika a robotika
Studijní program: Robotika
Květen 2017

Poděkování

Tímto bych rád poděkoval Ing. Jiřímu Zemánkovi za cenné rady a věcné připomínky při zpracovávání této práce. Zároveň bych rád poděkoval svým rodičům za jejich neutuchající podporu při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. V Praze dne 20. května 2017

.....

podpis autora práce

Abstrakt

Tato bakalářská práce se zabývá návrhem autonomního hráče vzduchového hokeje, stolní hry, při níž se dva hráči snaží tomu druhému vstřelit branku pukem nadnášeným proudem vzduchu. Práce se zabývá implementací lokalizačního systému puku s využitím kamery a algoritmů počítačového vidění. Dále se práce zabývá řízením krokových motorů a návrhem herních strategií. Tyto strategie ke své funkci využívají matematického modelu chování puku na hrací ploše a predikci jeho trajektorie. Součástí práce je také implementace všech zmíněných částí do reálného řídicího systému.

Klíčová slova: vzduchový hokej, počítačové vidění, OpenCV, řízení krokových motorů, predikce pohybu

Vedoucí: Ing. Jiří Zemánek

Abstract

This bachelor thesis focuses on designing autonomous player of air-hockey, the board game, where two opponents try to score with the puck hovered by an air flow. The interest of work is to implement localization system of the puck using camera and computer vision algorithms. The work also resolves stepper motor control and designs game strategies. These strategies use the mathematical model of puck movement and puck trajectory prediction. This work also contains an implementation of all mentioned parts to a real control system.

Keywords: air hockey, computer vision, OpenCV, stepper motor control, trajectory prediction

Title translation: Automatic Air Hockey

Obsah

Obrázky

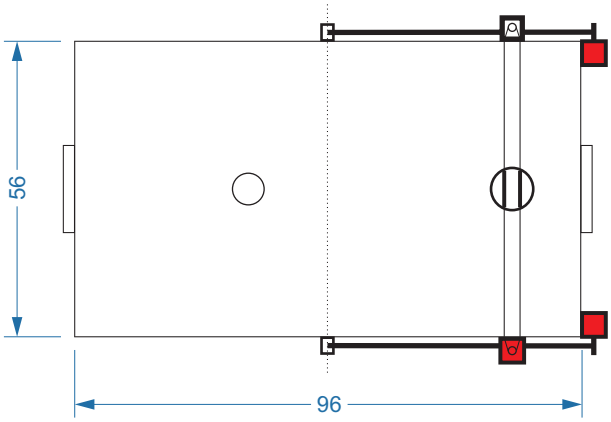
Kapitola 1
Úvod

Vzduchový hokej je hra, kterou tvoří speciálně upravený stůl, jenž má v horní desce otvory. Těmito otvory zespodu stolu proudí vzduch a nadnáší plastový puk pohybující se po stole. „Hokej“ se hra jmenuje z toho důvodu, že proudící vzduch snižuje puku tření, a proto připomíná puk pohybující se po ledě. Hru hrají dva hráči, každý z nich drží plastový odrážeč zvaný *pad*, kterým mohou puk odpalovat, a jejím cílem je zasáhnout puk tak rychle nebo takovým odrazem o stěnu, aby se puk trefil do soupeřovy branky.

Cílem této bakalářské práce je navrhnout a realizovat řídicí systém, který s využitím elektrických komponent nahradí lidského protivníka, a může tak sloužit jako autonomní protihráč. Jeho úkolem je lokalizovat pohybující se puk, vyhodnotit jeho trajektorii pohybu a provést svými akčními členy potřebné zásahy. Motivací k této práci je vytvoření interaktivního modelu, který bude složit k propagaci fakulty na ukázkových akcích, a předvedení oblasti zájmu katedry řízení.

V první části této práce se zaměřím na specifikaci řídicí úlohy, návrh struktury řídicího systému a určení kritérií řízení. V druhé části se budu zabývat vytvořením matematického modelu pohybu puku po hrací ploše, který bude sloužit k porozumění fyzikálního chování puku a k simulacím herních strategií. V dalších částech se poté již zaměřím na řešení řídicí úlohy, která spočívá v lokalizaci puku, řízení akčních členů robotu a aplikaci herních strategií.

Tato bakalářská práce se zakládá na výsledku týmového projektu, jehož úkolem bylo sestavení konstrukce stolu. Zespodu stolu jsou umístěny dva PC větráky, které mají za úkol vytvářet proudění vzduchu skrz stůl a nadnášet tak puk. Akčními členy systému jsou tři krokové motory, které nesou konstrukci s hracím padem autonomního hráče. Rozmístění motorů je zobrazeno ve schématu (1.1).



Obrázek 1.1: Schéma hracího pole (míry jsou uvedeny v jednotkách cm)

Dále se tento projekt částečně snaží rekonstruovat již hotový zdokumentovaný projekt [1] a [2], ale jeho úkolem je veškeré úlohy plnit vlastními postupy. Práce také koncipuje lokalizační systém tak, aby jeho výpočetní systém mohl být připevněný ke konstrukci stolu. Poslední úlohou je navrhnout vlastní herní strategie a zdokumentovat jejich funkce.

Kapitola 2

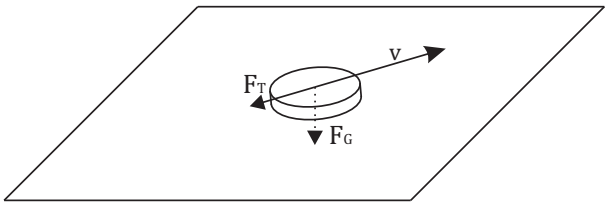
Matematický model puku

V této kapitole se budu věnovat sestavení matematického modelu puku a jeho interakcím s hrací plochou a autonomním hráčem. Tento model povede k lepšímu porozumění fyzikálního chování puku a sestavení simulace, ve které bude možné testovat herní strategie.

2.1 Model pohybu puku po hrací ploše

Model pohybu puku lze rozdělit do dvou částí. První část modeluje úbytek energie puku v průběhu pohybu vlivem jeho tření o hrací plochu. Druhou částí je model jeho pohybu ovlivněný hranicemi hřiště. U něj se budu zajímat o jeho chování při odrazech o tyto hranice. V této sekci se budu věnovat první ze zmíněných úloh.

Pokud neuvážím žádné vedlejší nelinearity hřiště, jako jsou náklon nebo okolní podmínky, tak se puk nebude pohybovat, dokud mu hráč (ať už člověk, nebo robot) nepředá kinetickou energii úderem. Tuto energii puk v průběhu pohybu ztrácí vlivem tření o hrací plochu.



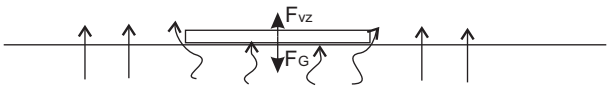
Obrázek 2.1: Působení tření na pohybující se puk

Vzhledem k vypozorovanému chování puku, který na první pohled plynule zpomaluje, jsem se rozhodl modelovat pohyb puku pomocí lineárního modelu tření vyjádřeného jediným parametrem tlumení b. Složka tření zároveň bude i jediná složka pohybové rovnice puku

$$F = -F_t = -bv. \tag{2.1}$$

Ačkoliv rovnice popisující pohyb puku je velice triviální, tření je nelineární jev a jeho identifikace nemusí být vždy patrná hned z prvních experimentů. V úvahu musíme také vzít, že puk je nadnášen prouděním vzduchu zespodu

hrací plochy, které tření snižuje a přidává identifikaci další nelinearity jak samotným prouděním, tak rozmístěním otvorů, skrz které vzduch proudí.

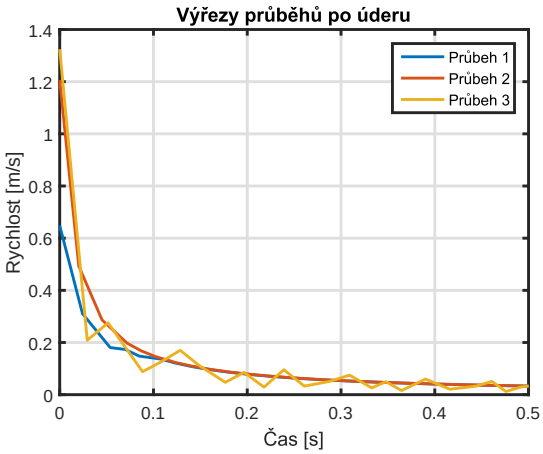


Obrázek 2.2: Působení proudění vzduchu na puk

Snahou bude provést takové experimenty, které povedou k co nejbližší aproximaci této nelinearity a jejímu zanesení do simulace.

2.1.1 Identifikace parametru tlumení

S využitím již hotového lokalizačního systému jsem provedl měření rychlosti pohybu puku. Naměřená data jsem přizpůsobil základním jednotkám rychlosti a poté zpracovával v MATLABu. Z naměřených dat jsem vybral úseky, které zaznamenávaly průběh rychlosti puku těsně po prudkém úderu.



Obrázek 2.3: Výřezy průběhů z naměřených dat

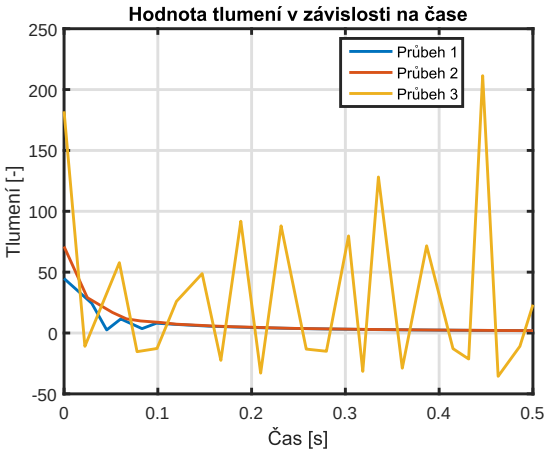
Z rovnice (2.1) vyjádřím parametr b

$$b = -\frac{F}{v} = -\frac{ma}{v}. \tag{2.2}$$

Protože pracuji s diskrétně zaznamenanými daty, vytvořím odpovídající diferenční rovnici. Konstantu hmotnosti puku m zahrnu do identifikovaného parametru společně s b a provedu jejich substituci za parametr b'

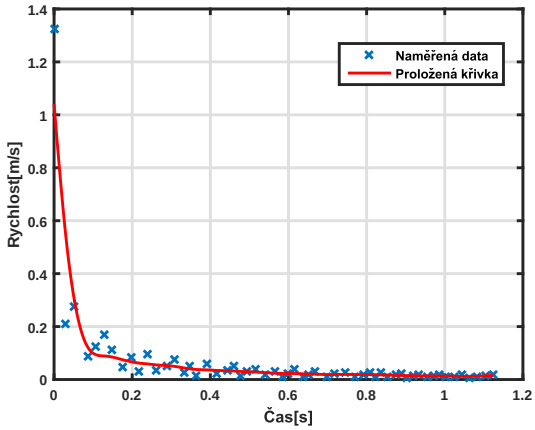
$$\frac{b}{m} = b' = -\frac{1}{v}\frac{\Delta v}{\Delta t}. \tag{2.3}$$

Vzorcem (2.3) jsem přepočítal data z vyříznutých vzorků a získal průběhy zobrazené v grafu (2.4).



Obrázek 2.4: Hodnota parametru tlumení v průběhu pohybu puku

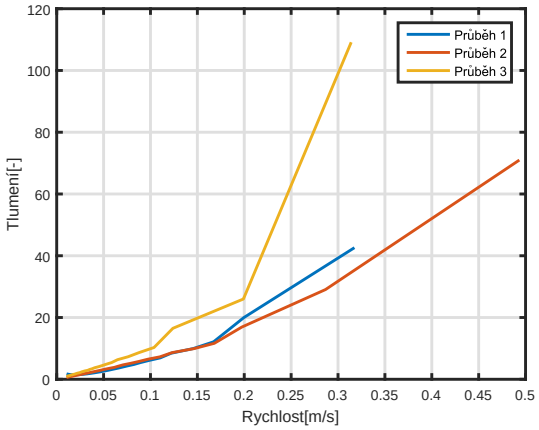
Z důvodu, že se ve vzorci objevuje diference, data s nedostatečným vzorkováním nebo šumem vedou k mylným výsledkům. Tento jev se projevuje v průběhu č. 1 a č. 3 znázorněném na grafu (2.4). Další poznatek měření vyplývá z průběhu č. 2, u kterého výpočet proběhl korektně. Ukazuje se, že parametr tlumení není konstantní, ale mění se v závislosti na rychlosti puku. Abych toto tvrzení ověřil, rozhodl jsem se průběhy č. 1 a č. 3 proložit křivkou. Vyhlazená křivka, která bude odpovídat naměřeným datům ve výpočtech s diferencí, nebude vytvářet chaotické výsledky.



Obrázek 2.5: Proložení naměřených dat křivkou s využitím metody *Smoothing Spline*

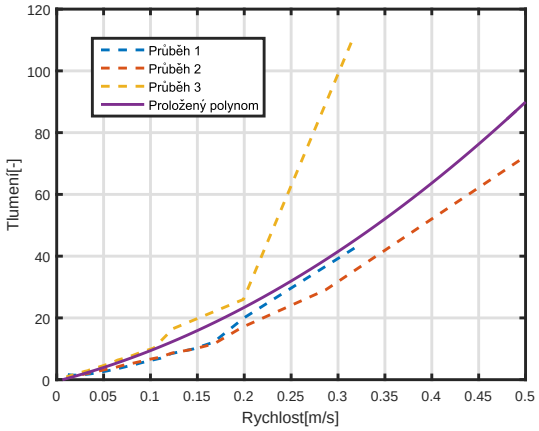
K proložení naměřených dat křivkou jsem využil aplikaci MATLABu *Curve fitting tool*, kde jsem využil k proložení metodu *Smoothing Spline*. U prokládání jsem musel dbát na to, aby výsledná křivka odpovídala skutečnému chování puku. Vzhledem k reálnému chování puku na hřišti jsem usoudil, že kolísavá naměřená data zobrazená v grafu (2.5) jsou zkreslená špatným měřením, pravděpodobně kmitáním závěsu kamery. Z tohoto důvodu jsem proložení dat touto křivkou uvážil jako odpovídající.

Proložená data jsem znovu zpracoval vzorcem (2.3) a v grafu (2.6) vyjádřil výsledky závislostí parametru tlumení na rychlosti puku u všech třech průběhů.



Obrázek 2.6: Závislost parametru tlumení b' na rychlosti

Původním úmyslem bylo chování puku co nejlépe aproximovat pro účely simulace. Z výsledků výpočtu zobrazeného v grafu (2.6) se ukázalo, že chování ve všech analyzovaných průbězích se liší při vyšších rychlostech. Z tohoto důvodu jsem se výsledky rozhodl proložit polynomem druhého stupně, který bude odpovídat chování všech průběhů při menších rychlostech a zároveň vytvoří kompromis na rychlostech vyšších.



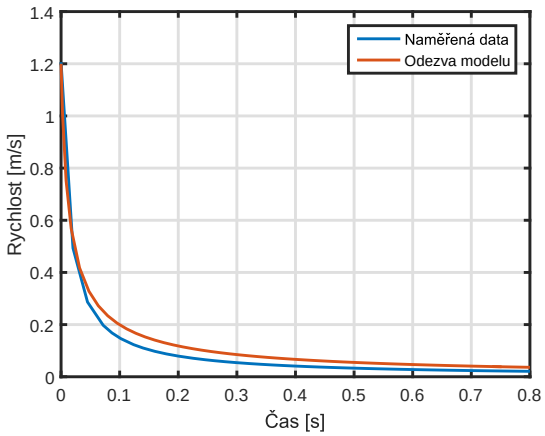
Obrázek 2.7: Proložení výsledků polynomem druhého stupně

Výsledný polynom, který vyjadřuje závislost parametru tlumení na rychlosti má tvar

$$b' = 45,2v^2 + 30,5v. \tag{2.4}$$

Z výsledků měření se ukazuje, že prouděním vzduchu skrz podložku na puk se snižuje jeho tření, ale tento efekt působí hlavně při nízkých rychlostech.

To obhajuje tvrzení o zmiňované nelinearitě, kterou tento jev přináší. Abych ověřil vypočítané výsledky, provedl jsem porovnání modelu s reálnou odezvou. Výsledky jsem zobrazil v grafu (2.8).



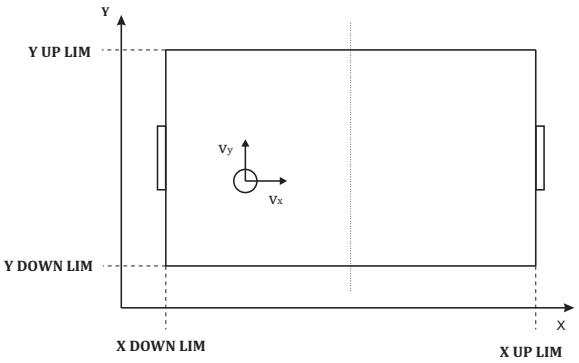
Obrázek 2.8: Porovnání modelu s naměřenými daty při odezvě na počáteční podmínky rychlosti

Z tohoto porovnání je patrné, že odezva matematického modelu naměřeným datum odpovídá.

2.1.2 Model odrazů puku od hrací plochy

V předchozí sekci jsem popsal vliv tření puku a hrací plochy na změnu rychlosti jeho pohybu. V této části se budu zabývat chováním puku v prostředí hřiště, který se vlivem odrazů od okrajů hřiště modeluje hybridním systémem.

Abych mohl popisovat pohyb puku po hrací ploše, zavedu nejprve souřadný systém.



Obrázek 2.9: Souřadný systém hrací plochy

V obrázku (2.9) je zanesen souřadný systém společně s označením pro souřadnice horních a dolních hranic hřiště. Pohyb puku v závislosti na těchto

souřadnicích vyjádřím dvěma rovnicemi

$$a_x = -b'(\frac{v_x}{\sqrt{{v_x}^2 + {v_y}^2}})v_x \qquad (2.5)$$
$$a_y = -b'(\frac{v_y}{\sqrt{{v_x}^2 + {v_y}^2}})v_y.$$

Hybridní systém se bude vyznačovat podmínkami, při kterých skokově mění své parametry. Těmito podmínkami budou odrazy puku od hranic hrací plochy. Z průběhů č. 1 a č. 2 předchozího měření zobrazeného v grafu (2.3) se ukázalo, že i při vysoké rychlosti puku, při které pohyb musel kvůli malé ploše hřiště obsahovat hned několik odrazů, nemají tyto odrazy žádný vliv na změnu rychlosti puku. Z tohoto důvodu nebudu v odrazech uvažovat žádný úbytek energie puku a pouze otočím směr rychlosti pohybu

$$((x + r) \geq X_{\text{UPLIM}} \vee (x - r) \leq X_{\text{DOWNLIM}}) \Rightarrow (v_x = -v_x) \qquad (2.6)$$
$$((y + r) \geq Y_{\text{UPLIM}} \vee (y - r) \leq Y_{\text{DOWNLIM}}) \Rightarrow (v_y = -v_y),$$

přičemž parametr r je poloměr puku.

Tyto podmínky v kombinaci s diferenciálními rovnicemi (2.5) a znalostmi o parametru b' jsou pro simulaci pohybu puku na hrací ploše dostatečné.

2.2 Model interakce puku a autonomního hráče

Další úlohou modelování bude vyjádřit chování puku při interakci s autonomním hráčem. Na tuto úlohu budu nahlížet stejným způsobem jako jsem nahlížel na modelování odrazů puku od krajů hřiště, tedy jako na hybridní systém. Nejprve budu modelovat situaci, kdy se autonomní hráč nepohybuje a puk se od něj odrazí, a dále situaci, kdy se autonomní hráč pohybuje a odrazem předá puku energii.

2.2.1 Odraz puku bez předání energie

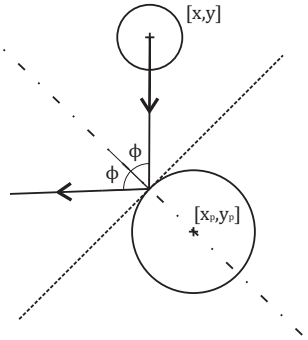
Puk i pad jsou kruhové objekty. Oba mají v zavedeném souřadnicovém systému, zobrazeném na obrázku (2.9), své souřadnice označující jejich střed, vektory rychlosti a parametr svého poloměru. Souřadnice a parametry patřící hracímu padu jsou označeny indexem p. V této části modelování budu uvažovat, že vektory rychlosti padu jsou nulové.

$$v_{px} = 0 \qquad (2.7)$$
$$v_{py} = 0.$$

Podmínkou odrazu puku a padu je přiblížení jejich středů na vzdálenost jejich poloměrů

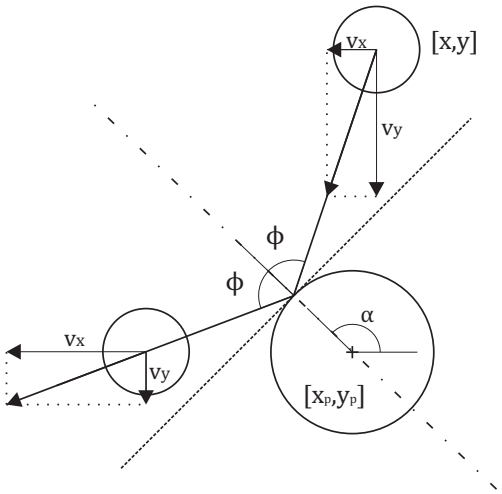
$$\sqrt{(x - x_p)^2 + (y - y_p)^2} \leq (r + r_p). \tag{2.8}$$

Jelikož uvažuji situaci, kdy se pad nepohybuje, při odrazu vytvořím v místě dotyku tečnu simulující hranici, o kterou se puk odrazí.



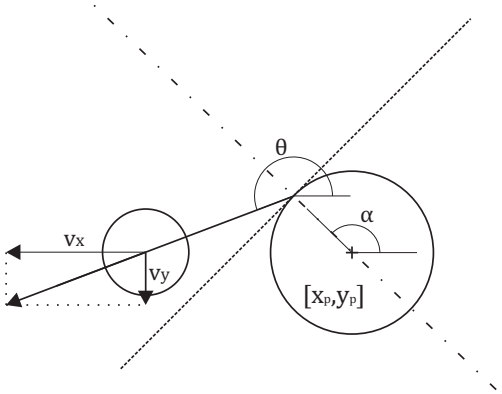
Obrázek 2.10: Úhel odrazu v místě dotyku puku a padu

Pad je upevněný ke konstrukci, s kterou pohybují krokové motory. Pokud pad stojí na místě, krokové motory jsou fixované a brání volnému pohybu konstrukce. Z tohoto důvodu nebudu při odrazech uvažovat přenos energie z puku do padu. Pohyb puku je vyjádřen vektory v_x a v_y. Tyto dva vektory se po odrazu změní v závislosti na dvou parametrech. Prvním je místo, kde se objekty navzájem dotknou. Toto místo vyjádřím úhlem dotyku α. Druhým parametrem je úhel ϕ, pod kterým puk do bodu doteku dorazí.



Obrázek 2.11: Úhel dotyku

Velikosti vektorů v_x a v_y jsou po odrazu závislé na celkové rychlosti puku a úhlu θ, pod kterým se bude puk po odrazu pohybovat. Úhel θ je měřen od osy x zavedených souřadnic.



Obrázek 2.12: Úhel odrazu θ

v_x a v_y se při odrazu přepočítají vzorci

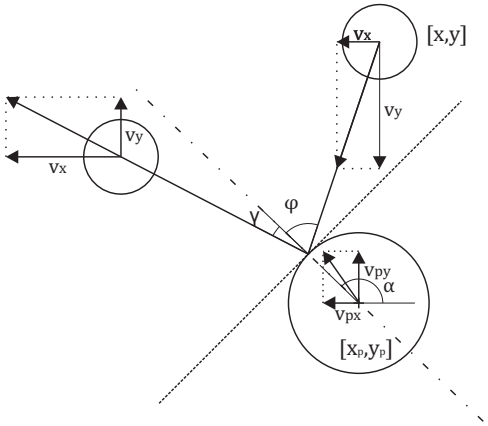
$$v_x = \sqrt{{v_x}^2 + {v_y}^2}\cos\theta \qquad (2.9)$$
$$v_y = \sqrt{{v_x}^2 + {v_y}^2}\sin\theta.$$

2.2.2 Odraz puku s předání energie

V druhé části budu opět modelovat odraz puku od padu, a to s předpokladem, že pad se pohybuje nenulovou rychlostí.

$$v_{px} \neq 0 \qquad (2.10)$$
$$v_{py} \neq 0.$$

Podmínky pro vyhodnocení změny parametrů systému (podmínka vyhodnocení odrazu) bude stejná jako v prvním případě, tedy vyjádřená vzorcem (2.8). I v této situaci budu uvažovat již zmiňované tvrzení, podle kterého výměna energie probíhá pouze z padu do puku. Puk svým nárazem tedy nijak neovlivňuje pohyb padu. Rozdíl oproti situaci, kdy se pad nepohyboval, bude v rozdílném úhlu, pod kterým se puk od padu odrazí.

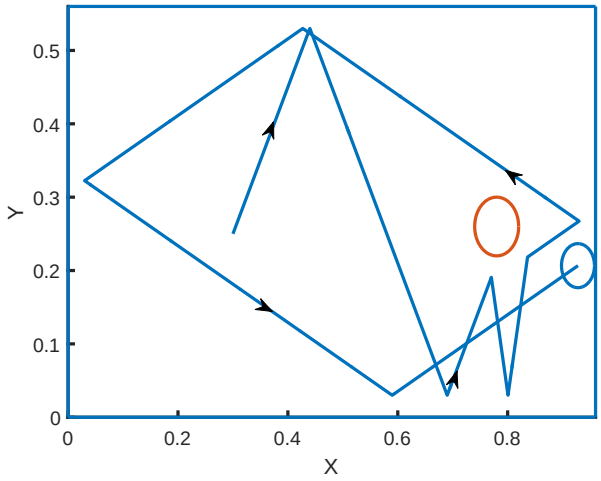


Obrázek 2.13: Odraz puku od pohybujícího se padu

Při odrazech opět přepočítávám vektory rychlosti puku v_x a v_y. První část tohoto výpočtu přímo vychází z první situace, kdy se puk nepohyboval, a vyjádřím ji rovnicemi (2.9). K této části vlivem pohybu padu při kolizi přibývá další složka, která vyjadřuje přenos energie z padu do puku. Tato složka je závislá na rychlosti a směru pohybu padu společně s úhlem α, který vyznačuje místo dotyku padu a puku. Vektory v_x a v_y po odražení lze dopočítat vzorcem

$$\begin{aligned} v_x &= \sqrt{{v_x}^2 + {v_y}^2}\cos\theta + v_{px}\cos\alpha \\ v_y &= \sqrt{{v_x}^2 + {v_y}^2}\sin\theta + v_{py}\sin\alpha. \end{aligned} \tag{2.11}$$

V závěru modelování jsem zaznamenal trajektorii puku, která obsahovala odrazy od hranic hrací plochy i padu, abych ověřil správnost výpočtů. Tyto odrazy jsem zaznamenal v grafu (2.14).



Obrázek 2.14: Trajektorie modelu puku s odrazy

Kapitola 3

Struktura řídicího systému

3.1 Řídicí úloha

Řídicí úlohou autonomního hráče je co nejlépe simulovat lidského protivníka. Tuto úlohu dekomponuji na několik fází.

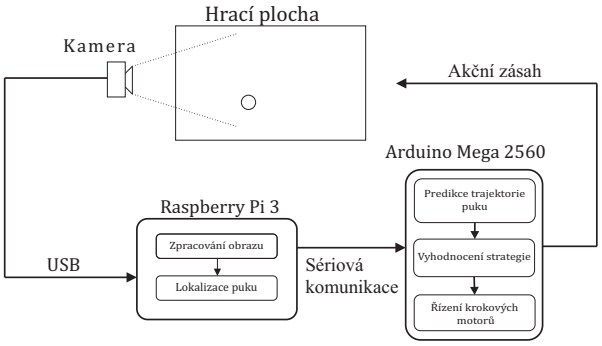
V první fázi musí systém získat informace o pohybujícím se puku. Tyto informace musí získávat tak často, aby mohl celý systém dostatečně rychle zareagovat akčními členy. Vzhledem k tomu, že se puk musí pohybovat volně po hrací ploše, je nutné jeho polohu lokalizovat bezkontaktně a informace o lokaci musí být následovně přeneseny do definovaného souřadného systému.

Další fázi řízení tvoří systém, který z informací o poloze puku a znalosti jeho chování na hrací ploše dokáže predikovat trajektorii, po které se bude pohybovat. Na základě pohybu puku musí vyhodnotit takové zásahy, aby zabránil gólu a co nejlépe útok opětoval.

Poslední fází je řízení akčních členů systému. Tyto akční členy musí být dostatečně výkonné, aby dokázaly provádět požadované zásahy rychle a přesně.

3.2 Návrh řídicí struktury

Lokalizaci puku jsem se rozhodl provádět pomocí kamery pozorující hrací plochu. Z kamery budu získávat informace o poloze puku metodami počítačového vidění. Vzhledem k tomu, že počítačové vidění vyžaduje práci se speciální knihovnou a především ke své práci vyžaduje vyšší výpočetní výkon, rozhodl jsem se tuto úlohu provádět samostatně na zařízení Raspberry Pi 3. Této úloze se budu věnovat v kapitole (4). Informaci o lokaci puku budu přenášet do druhého zařízení, které bude vykonávat zbytek řídicí úlohy. Zbytkem řídicí úlohy je predikce trajektorie a aplikace herní strategie společně s řízením akčních členů. K tomuto řízení budu využívat zařízení Arduino Mega 2560, které je dostatečně výkonné, aby provádělo všechny tři zmíněné úkoly. Výhodou je, že toto zařízení je jednoduše programovatelný mikrokontrolér, který neomezuje uživatele v práci s jeho vnitřními funkcemi. Predikování trajektorie a vyhodnocování herní strategie se budu věnovat v kapitole (6) a řízení krokových motorů se budu věnovat v kapitole (5).



Obrázek 3.1: Blokové schéma řídicího systému

Kapitola 4

Rozpoznání obrazu a lokalizace puku

Jak již bylo zmíněno, lokalizaci puku na hrací ploše budu provádět pomocí kamery, která snímá z vrchu hrací pole. Obraz kamera streamuje rychlostí 50 snímků za vteřinu (FPS) přes USB do Raspberry Pi, kde je nutné implementovat rozpoznávací algoritmus. Tento algoritmus má za úkol v každém snímku lokalizovat pozici puku a tuto pozici následovně transformovat do souřadnic hracího stolu. Zároveň se algoritmus musí postarat o co nejrobustnější filtrování, které mohou ovlivnit různé objekty v záběru podobné barvy, velikosti a tvaru. Tato kapitola se bude zabývat využitelností metod lokalizace objektu v obrazu, filtrováním barev a optimalizací těchto algoritmů z hlediska výpočetní rychlosti. K rozpoznávání budu využívat open-source knihovnu OpenCV, která disponuje optimalizovanými funkcemi pro práci s obrazem. Algoritmus budu implementovat v jazyce Python, ačkoliv knihovnu lze využít i v jazyce C++, C a Java.

4.1 Filtrování barvy

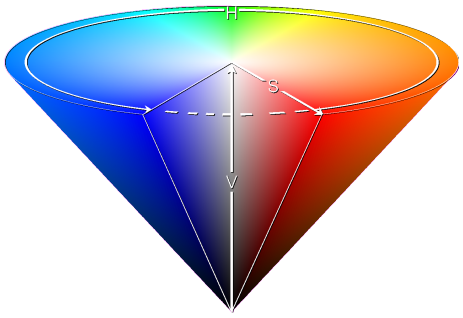
Prvním krokem k lokalizaci objektu v obrazu je uvědomění si jeho základních rysů, které lze využít k jeho rozpoznání. Základním rysem každého objektu je barva. Stejně jako většina částí hracího stolu i puk je vytištěn pomocí 3D tiskárny. Výhodou se tedy stává možnost výběru barvy puku volbou filamentu 3D tiskárny.

Z hlediska filtrace barvy je výhodné zvolit barvu puku tak, aby byla v záběru kamery unikátní. A to z toho důvodu, abych už samotnou filtrací barvy ze záběru odstranil co nejvíce ostatních objektů. Za tímto účelem jsem se rozhodl zvolit barvu puku světle červenou, která se nejen nenachází na hrací ploše, ale i tím, že je světlá, snižuje pravděpodobnost výskytu na podlaze, kde jsou barvy z pohledu kamery naopak většinou tmavé.

4.1.1 Barevný model

Před samotným filtrováním barvy je potřeba se rozhodnout, jaký barevný model budeme k filtrování využívat. Většina kamer svůj výstup interpretuje v barvách RGB, nicméně knihovna OpenCV obsahuje sadu funkcí, s kterou můžeme snímek jednoduše konvertovat i do modelu HSV. V HSV jde velmi

jednoduše interpretovat interval barevných odstínů, ve kterém se filtrovaná barva pohybuje, a proto jsem se ho rozhodl využít.

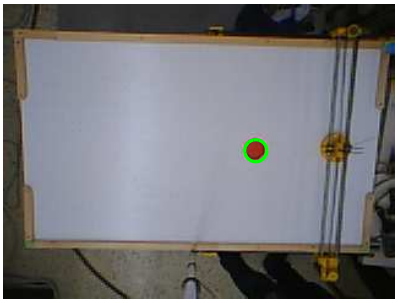


Obrázek 4.1: Barevný model HSV, obrázek převzat z [3]

Barevný model HSV rozkládá barvu do tří složek (Hue, Saturation a Value), což je znázorněno na obrázku (4.1). Složka *Hue* vyjadřuje barevný tón, *Saturation* vyjadřuje sytost barvy a *Value* její jas. Výhoda tohoto modelu spočívá v tom, že zastínění puku výrazně nezmění složku *Hue*, ale bude měnit pouze složky zbývající.

4.1.2 Měření barvy puku

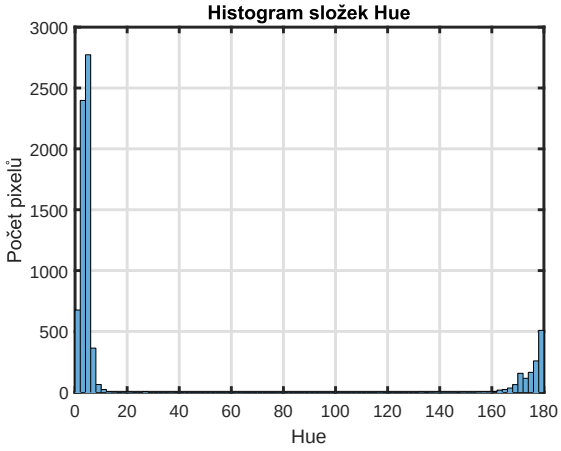
Abych provedl kvalitní filtrování, učinil jsem sérii měření, která měla zjistit, v jakých intervalech se jednotlivé složky červené barvy puku pohybují. Kamerou jsem provedl 50 snímků puku na hrací ploše, který jsem v průběhu snímání posouval do různých oblastí hřiště a cíleně zastiňoval. Z těchto snímků jsem pomocí MATLABu vystřihl oblasti, kde se kotouč nacházel, a zkoumal hodnoty složek barvy všech vystřižených pixelů. Tyto údaje jsem poté zanášel do histogramu.



Obrázek 4.2: Příklad snímku s vyznačenou oblastí střihu

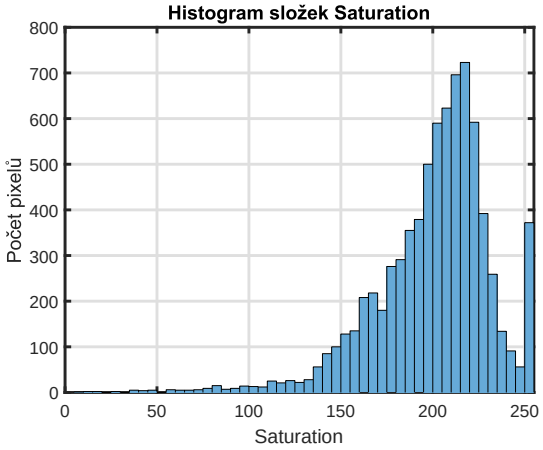
Knihovna OpenCV vyjadřuje složku *Hue* v rozsahu $0-180$ a složky *Saturation* a *Value* v intervalu $0-255$. Měření jsem těmto rozsahům přizpůsobil.

Výsledky měření ukázaly, že složka *Hue* se u červeného kotouče pohybuje v intervalu 0−11 a 169−180. Na první pohled se může zdát zvláštní, že se barva dělí do dvou intervalů, ovšem složka *Hue* je vyjádřená do kruhu a hodnoty 0 a 180 se nacházejí vedle sebe.

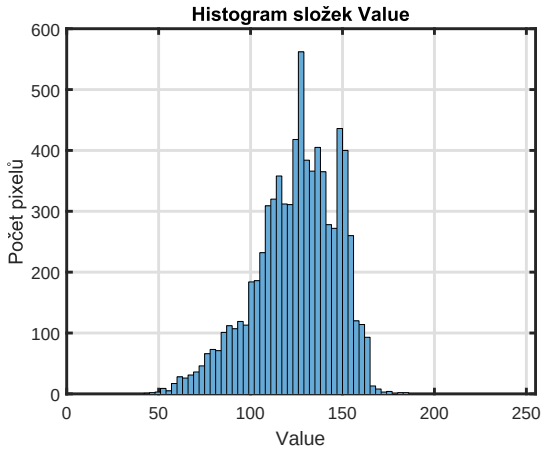


Obrázek 4.3: Histogram složky Hue

Intervaly složek *Saturation* a *Value* již byly širší. *Saturation* se významně pohybovala v intervalu od 110 až po jeho vrchol a *Value* od 45 do 173.



Obrázek 4.4: Histogram složky Saturation



Obrázek 4.5: Histogram složky Value

Změřené intervaly jsem uplatnil v programu a filtrací získal takzvanou bitovou masku. Bitová maska obsahuje pouze pixely filtrované barvy.

Obrázek 4.6: Záznam snímku a jeho filtrované bitové masky

4.2 Lokalizace s využitím HoughCircles transformace

Odfiltrováním barvy puku jsem ze snímku odstranil téměř všechny nežádoucí objekty, ale je pořád možné, že se v záběru vyskytne objekt stejné barvy, jako například tričko hráče nebo batoh položený na zemi vedle stolu. Z tohoto důvodu je nutné lokalizaci přidat další kritérium. Dalším význačným rysem puku je jeho kruhový tvar. Tento rys se dá s knihovnou OpenCV detekovat s využitím funkce *HoughCircles* (viz [6]). Funkce HoughCircles tvoří pro každý pixel množinu kružnic, jejichž parametry transformuje do jedné roviny. V této rovině se hledají shluky bodů vyznačující pixely ležící v jedné kružnici. Touto metodou lze v bitové masce jednoduše puk lokalizovat.

Po několika experimentech se ukázalo, že díky šumu vyskytujícímu se na okrajích hledané kružnice vyžadovala metoda nejprve úpravu bitové masky morfologickými operacemi (dilatací a erozí) a také zpřísnění kritéria počtu testovaných kružnic transformace HoughCircles. I po těchto úpravách systém

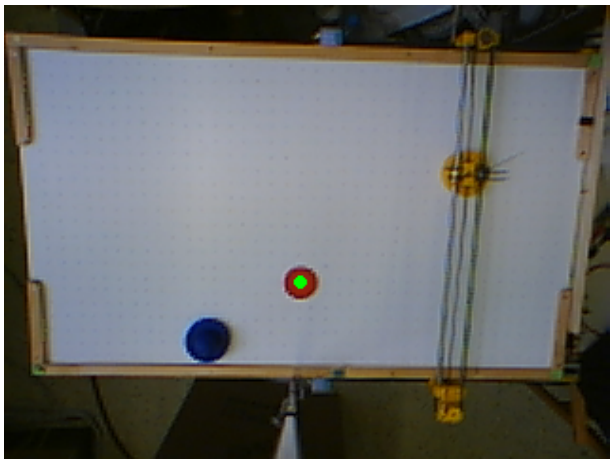
nedokázal puk lokalizovat zhruba na každém pátém snímku.

Další potíží byla příliš dlouhá doba výpočtu, způsobená už samotnou velice náročnou transformací HoughCircles, která společně se všemi nutnými morfologickými operacemi snížila FPS kamery na 20 snímků za vteřinu. Z těchto důvodů jsem se rozhodl tuto metodu nepoužívat.

4.3 Lokalizace těžištěm kontury

Po neúspěšné metodě s hledáním kružnice jsem se rozhodl využít co nejméně výpočetně náročné funkce a k lokalizaci využít velikosti puku. V bitové masce jsem pomocí funkce *findContours* (viz [7]) nalezl obrysy všech objektů a získal velikost jejich plochy. Podle plochy kontury budu vybírat pouze takové objekty, které jsou totožně velké s pukem. Abych zvolil správné limity pro výběr objektu, provedl jsem měření, při kterém jsem ze záběru odstranil všechny objekty, které by mohly být vyhledány funkcí findContours, a zaznamenával plochu obrysu puku v různých pozicích na hrací ploše. Tím jsem vytvořil co nejužší interval, ve kterém se puk pohyboval, abych zabránil lokalizaci jiných objektů. U správně nalezené kontury získávám souřadnice jeho středu dopočítáním těžiště této kontury.

Tato metoda se prokázala jako velmi stabilní a manipulováním s limity pro velikosti kontury lze detekovat i puk z části zakrytý. Tím, že nedetekujeme rys kružnice, není potřeba používat žádné morfologické operace ani výpočetně náročné transformace, a proto nedojde ke snížení FPS kamery. Jediný případ, kdy je algoritmus nestabilní, nastává, pokud kamera uvidí stejně velký objekt stejné barvy, což můžeme vyřešit implementací oblasti zájmu.



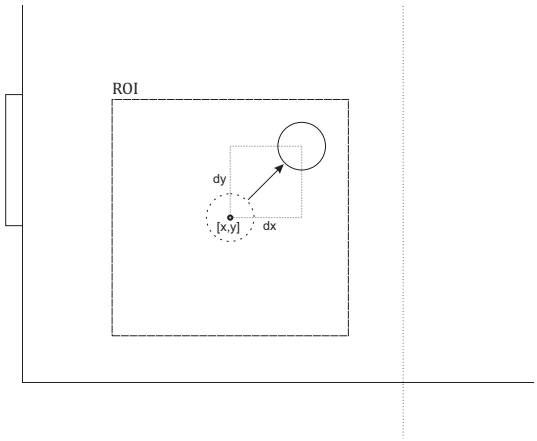
Obrázek 4.7: Snímek s lokalizací puku

4.4 Oblast zájmu

Jak jsem již zmínil v kapitole (4.2), kvalita řízení je silně závislá na kvalitním a rychlém zpracování obrazu. Náročné operace použité na každý snímek

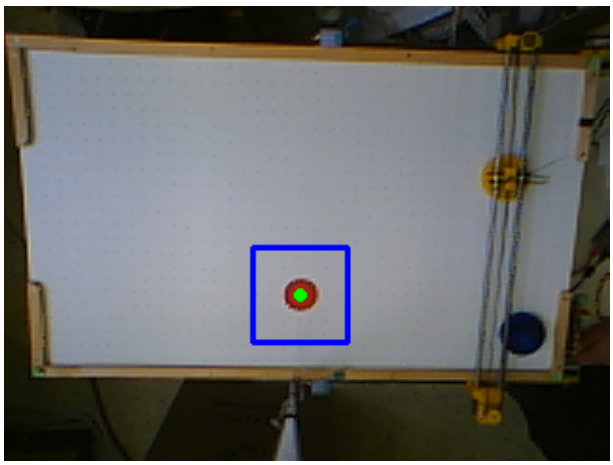
mohou omezit FPS kamery, což vede k nedostatečnému počtu údajů o pozici puku. Pokud je i přesto nutné v záznamu s vyšším FPS využít nějaké výpočetně náročné operace, je nutné provést optimalizaci, která práci se snímky zefektivní. Jednou z možností, jak lokalizaci optimalizovat, je zmenšit objem zpracovávaných dat. K tomuto můžeme využít takzvané oblasti zájmu nazývané ROI (Region of Interest).

Oblast zájmu vychází již z předchozí lokalizace, kdy je známá pozice puku. Zakládá na tom, že puk se mezi snímky jistě nepohnul z jedné strany hřiště na druhou, ale pravděpodobně se bude vyskytovat v blízkém okolí své předchozí pozice. Není tedy nutné prohledávat celý snímek, nýbrž pouze okolí puku, a zaznamenávat diferenci jeho pozice od předchozího snímku. Oblast prohledávání bude mít svůj střed v místě posledního výskytu puku.



Obrázek 4.8: Pohyb puku v oblasti zájmu

V případě, že hráč provede příliš silný úder a puk stihne mezi dvěma následujícími snímky z oblasti zájmu vyletět, program další snímek zpracuje celý a lokalizuje puk znovu.



Obrázek 4.9: Snímek s vyznačenou oblastí zájmu

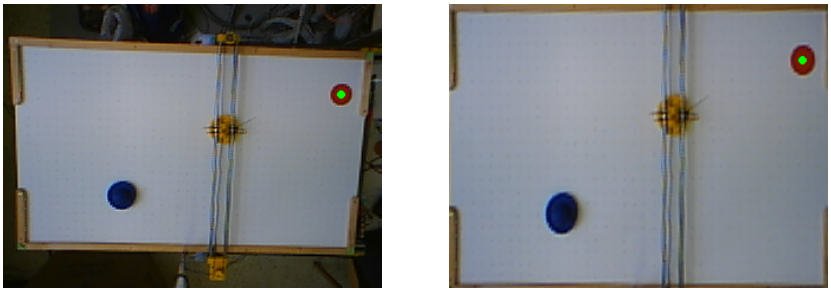
Oblast zájmu má markantní význam z hlediska optimalizace zpracování obrazu. Jelikož jsem již využitím lokalizace pomocí těžiště nijak nesnížil FPS kamery, uplatnění této metody v mém případě není nutné. Pokud by se ovšem ukázalo, že je třeba provádět robustnější lokalizaci, která vyžaduje náročnější operace, je uplatnění ROI naprosto vhodné. Stejně tak by bylo vhodné využít oblast zájmu v případě, že by úloha z jakéhokoli důvodu vyžadovala vyšší frekvenci snímků a bylo by třeba využít kvalitnější kamery. Poté by i lokalizace pomocí těžiště mohla bez uplatnění oblasti zájmu způsobovat snížení FPS.

Nevýhodou oblasti zájmu je při úloze vzduchového hokeje fakt, že puk vyletí z oblasti zájmu pouze při velmi prudkém úderu. V tu chvíli dochází ke ztrátě jednoho snímku, a to v době, kdy je důležitý každý snímek, jelikož při takto rychlém úderu má systém k vyhodnocení a predikci trajektorie puku k dispozici pouze 6−10 snímků. Možností by bylo využít snímek i v případě, že puk nebude v oblasti zájmu lokalizován, a zpracovat ho celý znovu. Nicméně informace po dvojím zpracování jednoho snímku může být zastaralá, a proto je lepší vzít snímek nový. Této nevýhodě oblasti zájmu se dá také předejít správným nastavením její velikosti, která se experimentálně může nastavit na takovou velikost, aby zachytila i opravdu prudké údery.

4.5 Transformace souřadnic puku

V poslední fázi zpracování obrazu je potřeba souřadnice lokalizovaného puku na snímku upravit tak, aby odpovídaly souřadnicím zavedeným z hlediska hrací plochy. Souřadnice získané z lokalizačního systému označují souřadnice pixelu, na kterém se nachází střed puku. Tyto souřadnice jsou silně závislé na pozici samotné kamery, která může být různě natočená nebo může hrací plochu sledovat pod úhlem. Tato nelinearita způsobuje, že pohyb puku po hrací ploše nevyvolává odpovídající změnu souřadnic v lokalizačním systému jako v zavedených souřadnicích stolu. Problém se dá vyřešit velmi přesným nastavením kamery, která bude pozorovat hrací plochu přesně pod pravým úhlem a bude zavěšena přímo nad středem hřiště. Další možností je vytvoření transformace, která bude souřadnice puku lokalizačního systému transformovat do odpovídajících souřadnic.

Ačkoliv upravením konstrukce stolu lze problém vyřešit, využití transformace je mnohem variabilnější řešení. Jedinou podmínkou transformace je pouze záběr na celou hrací plochu pod libovolným úhlem. Tuto transformaci lze vyjádřit transformační maticí, kterou získám vždy po spuštění lokalizačního programu manuální kalibrací. Kalibrace vyžaduje postupnou lokalizaci všech rohů hrací plochy. Transformační matici získáme využitím funkce *perspectiveTransform* z knihovny OpenCv (viz [5]). Krom možnosti upravit pozici puku z lokalizačního systému do ortogonálních souřadnic transformace dále umožňuje libovolně upravovat rozsah těchto souřadnic podle potřeby.



Obrázek 4.10: Transformovaný snímek po manuální kalibraci

Rozsah souřadnic jsem upravil na rozměr 960×560. Toto měřítko odpovídá rozměru stolu v milimetrech. V prvním pokusu jsem nejprve transformoval celý snímek a následně prováděl lokalizaci na transformovaném snímku. Tímto způsobem jsem se dostal k požadovanému výsledku, ale vlivem výpočetně náročné transformace, kdy se transformoval každý pixel snímku, se snížila FPS kamery na 30 snímků za vteřinu. Z tohoto důvodu jsem se rozhodl nejprve provést lokalizaci v zachyceném snímku a až poté transformovat pouze jedny souřadnice.

Transformovat pomocí matice T lze výpočtem

$$\begin{pmatrix} t_i x' \\ t_i y' \\ t_i \end{pmatrix} = T \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{4.1}$$

kde x' a y' jsou souřadnice v ortogonálním prostoru, T je transformační matice 3×3, x a y jsou původní souřadnice lokalizačního systému a t_i je parametr vyplývající z perspektivní transformace. Požadované souřadnice získám podělením prvního a druhého prvku výsledného vektoru parametrem t_i.

Kapitola 5

Řízení krokových motorů

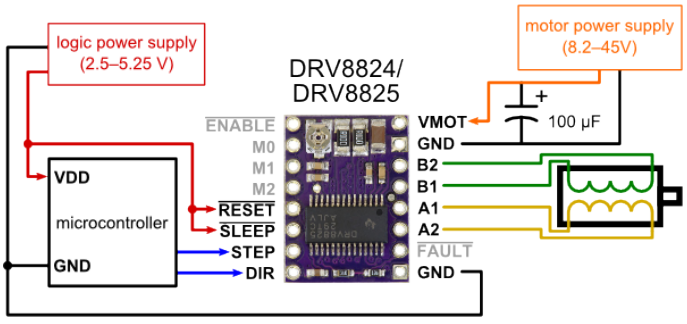
V této kapitole se budu zabývat výběrem hardwaru k řízení krokových motorů a způsobem jejich řízení. Řídicí úloha vyžaduje řízení pohybu hracího padu ve dvou navzájem ortogonálních osách s dostatečnou rychlostí alespoň 1000 kroků za vteřinu, aby dostatečně simuloval lidského protivníka.

5.1 Úvod do problému, popis hardwaru

Řízení všech krokových motorů bude zajišťovat Arduino MEGA 2560 společně s doplňkovým hardwarem. K řízení pohybu padu jsem využil tři dvoufázové krokové motory Microcon Sx17-1003LQCEF s přírubou NEMA 17. Výběr doplňkového hardwaru závisel na několika parametrech. Prvním parametrem bylo jednoduché ovládání krokových motorů umožňující jejich otáčení v obou směrech. Druhým parametrem bylo dostatečně vysoké napětí, kterým lze motory pomocí vybraného hardwaru řídit. S vyšším napětím může motor dosahovat větších momentů a vyšších rychlostí.

Po zvážení jsem využil stejnou strukturu, jakou využil autor v původním projektu [1], kde řídicími prvky jsou h-můstky Pololu DRV8825, připevněné na Shield RAMPS 1.4.

H-můstky DRV8825 dokáží motor řídit oběma směry s možností mikrokrokování a k řízení využít napětí až 45 V.



Obrázek 5.1: Pinout kontroléru DRV8825, obrázek převzat z [8]

Shield RAMPS 1.4 je původně navržen k řízení 3D tiskáren. Výhodou je, že tento shield lze rovnou připevnit k Arduinu, a tím, že je připraven k řízení

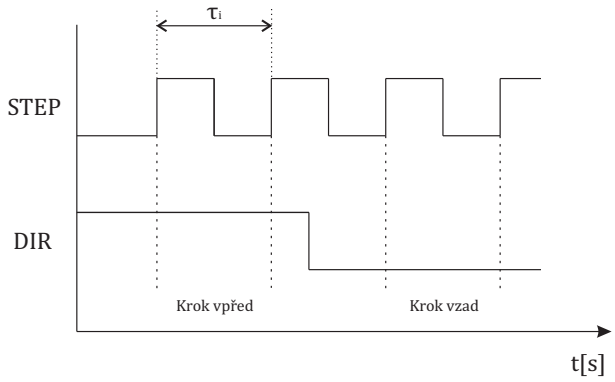
až pěti krokových motorů, je také připraven na přímé upevnění kontrolérů-DRV8825. Nevýhodou shieldu je omezené maximální napájecí napětí 12 V. Toto maximální napětí jsem zvýšil manuální úpravou na 24 V, nicméně omezil jsem tak funkci shieldu napájet Arduino.

I po zvýšení napájecího napětí desky na 24 V omezuji h-můstky DRV8825, které lze napájet až 45 V. Jelikož v originálním projektu k řízení motorů využívají pouze 12 V, rozhodl jsem se i přes omezení shield RAMPS 1.4. využít.

5.2 Řízení rychlosti

Krokové motory jsou akční členy, které jsou velmi přesné při řízení polohy, ovšem nejsou ideálním prvkem pro řízení u rychlých pohybů a vysokých otáček. Při vysokých otáčkách, a zejména prudkému zrychlení nebo zpomalení, může dojít k protočení motoru. Protože nepoužívám žádný další senzor pohybu padu, krom lokalizačního kamerového systému, údaje o poloze hráče řídicí systém udržuje softwarově s každým krokem. Protočení motoru a ztráta kroků by tedy vedla ke ztrátě polohy padu. Jelikož nahrazení krokových motorů stejnosměrnými by také vyžadovalo osazení konstrukce jinými senzory pohybu (například inkrementálními senzory otáček), rozhodl jsem se i přes to využít krokových motorů a problém se ztrácením kroků zahrnout do řídicí úlohy.

Přímé řízení otáčení krokového motoru zajišťuje driver DRV8825. Tento driver dává možnost řídit rychlost otáčení pomocí pulsů přivedených na vstupy *STEP* a *DIR* znázorněných na obrázku (5.1). Zatímco vstup *DIR* na základě přivedeného logického signálu ovládá směr otáčení motoru, vstup *STEP* reaguje na náběžnou hranu přivedeného signálu. Při náběžné hraně dojde k otočení motoru o jeden krok tím směrem, který určuje signál *DIR*.

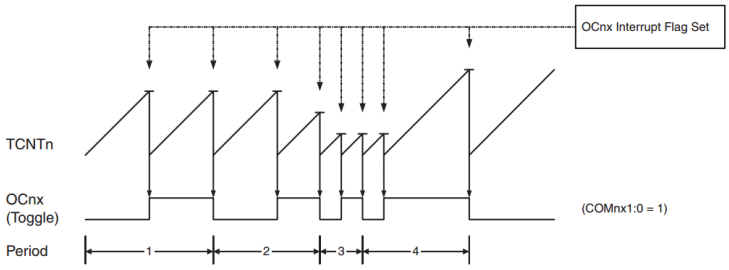


Obrázek 5.2: Ovládání pohybu h-můstkem DRV8825

Tímto způsobem lze pomocí frekvence náběžných hran nastavit rychlost otáčení motoru. Jelikož použité krokové motory mají rozlišení 200 kroků na otáčku, rychlost otáčení motoru ω lze vypočítat rovnicí

$$\omega = \frac{2\pi}{200\tau_i}, \tag{5.1}$$

kde tato rychlost je vyjádřena v radiánech za sekundu. K řízení každé osy motoru je třeba definovat vlastní proces, který bude generovat pulzy se správnou frekvencí. V oblasti mikroprocesorů se rozčlenění programu do těchto různých procesů zajišťuje pomocí časovačů. Arduino Mega 2560 má k dispozici celkem 3 časovače (timery), které lze k časování řídicích pulzů využít. K řízení obou os pohybu využiji šestnáctibitové timery 1 a 3. Oba časovače lze nastavit zapsáním hodnot do jejich registrů TCCR, OCR a TIMSK. Funkce časovače je taková, že při každém hodinovém pulzu procesoru se registr časovače TCNT inkrementuje. Časovač lze nastavit do různých režimů, při kterých dochází k takzvanému přerušení. K tomu dochází při přetečení maximální hodnoty registru či překročení námi zvolené hodnoty. Toto přerušení může například vyvolat pulz na konkrétním pinu procesoru nebo může vést na provedení naprogramované funkce. Za účelem řízení rychlosti otáčení krokových motorů budu využívat režim CTC (Clear Timer on Compare) viz [4, str. 120] znázorněný na obrázku (5.3).



Obrázek 5.3: Znázornění funkce časovače v režimu CTC, obrázek převzat z [4]

V tomto režimu se porovnává registr TCNT, který obsahuje aktuální hodnotu časovače, s hodnotu registru OCR. V případě, že se tyto hodnoty rovnají, dojde k přerušení a vynulování registru TCNT. Přerušením lze zavolat libovolně zvolenou funkci v programu. Nastavením hodnoty v registru OCR lze tedy volit, kdy k přerušení dojde a jak často se zvolená funkce provede. Ve funkci volané přerušením provedu inverzi logické hodnoty signálu STEP přiváděného na vstup h-můstku. Tím funkce každé dvě přerušení otočí krokový motor o jeden krok ve zvoleném směru. Tím pádem lze rychlost motorů v_m v jednotkách kroků za sekundu vypočítat vzorcem

$$v_m = \frac{f_{\mathrm{CPU}}}{2 \cdot OCR}. \tag{5.2}$$

Vzhledem k tomu, že registr OCR je šestnáctibitový, maximální hodnota, kterou ho lze naplnit je 65535. Pokud do registru OCR nastavíme jeho maximální hodnotu, z rovnice (5.2) získáme hodnotu 122 kroků za vteřinu. Protože šlo o maximální hodnotu OCR, nebyl bych schopen nastavit menší rychlost motoru než zmíněných 122 kroků za vteřinu. Z tohoto důvodu jsem nastavil takzvaný prescaler, který lze nastavit změnou registru TCCR. Ten zajišťuje, aby se hodnota v čítači inkrementovala v závislosti na dělicím poměru, na kterou je nastaven. Jelikož jsem prascaler nastavil na hodnotu

8, čítač se bude inkrementovat každý osmý hodinový tik procesoru. Tím je nutné přepsat rovnici (5.2) do tvaru

$$v_m = \frac{16 \cdot 10^6}{16 \cdot \mathrm{OCR}}. \tag{5.3}$$

Hodnota $16 \cdot 10^6$ Hz má význam taktu procesoru ATmega 2560. S využitím prescaleru dokáži minimální rychlost nastavit na 15 kroků za vteřinu. Z hlediska řízení už není důvod nastavovat menší rychlost. Funkce, volaná přerušením, je pouze doplněná o výjimku, kdy se při nastavení OCR na maximální hodnotu ve funkci negeneruje žádný pulz. Tímto dokáži zajistit zastavení motoru.

Abych mohl motory přímo řídit nastavením rychlosti v krocích za vteřinu, přepíšu funkci 5.3 do tvaru, kde rychlost motoru je vstupním parametrem a výstupem je hodnota registru OCR

$$\mathrm{OCR} = \frac{16 \cdot 10^6}{16 \cdot v_m}. \tag{5.4}$$

5.2.1 Omezení zrychlení padu

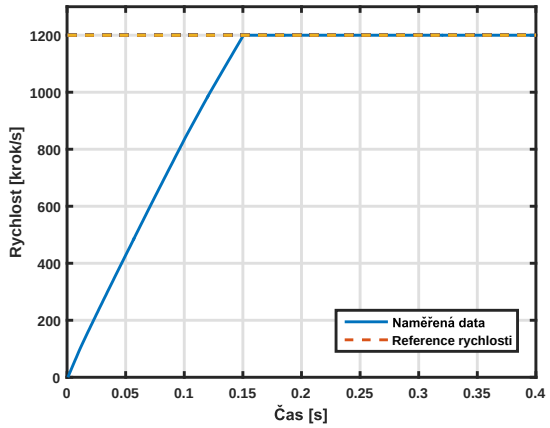
Z důvodů zmíněných na začátku sekce (5.2) jsem se při řízení krokových motorů rozhodl implementovat omezení zrychlení, abych předcházel protáčení hřídele motoru a ztrátě kroků. Toto omezení zamezí prudkým skokovým změnám rychlosti a zajistí plynulé zrychlování.

Experimentálně jsem zjistil, že maximální rychlost padu v obou osách je 1200 kroků za vteřinu. Při vyšších rychlostech hrozí, že dojde k přeskočení kroků, a to i v případě, kdy se pad již pohyboval maximální rychlostí. Nastavení rychlosti jsem koncipoval v intervalu

$$v_m \in \langle -v_{\max}, v_{\max} \rangle, \tag{5.5}$$

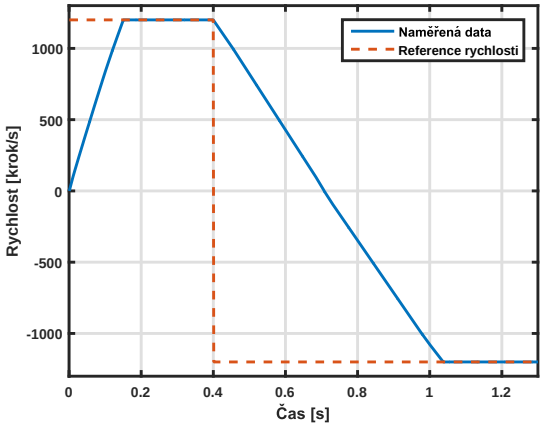
kde $v_{\max}$ je maximální rychlost 1200 kroků za vteřinu. Záporné znaménko u nastavení rychlosti způsobí změnu logické hodnoty na pinu DIR a otočí tak směr pohybu padu.

Omezení maximálního zrychlení musím volit tak, aby co nejlépe zamezilo ztrátě kroků. Zároveň musím dbát na to, abych zrychlením příliš nezpomalil autonomního hráče. Experimentováním se ukázalo, že optimální volba zrychlení je 10000 [kroků/s^2]. S tímto omezením jsem provedl měření, kdy jsem zaznamenával průběh rychlosti padu při rozjezdu.



Obrázek 5.4: Průběh rychlosti padu při rozjezdu

Další měření jsem provedl pro náhlou změnu směru pohybu padu.



Obrázek 5.5: Průběh rychlosti padu při náhlé změně směru pohybu

Z měření zobrazených na grafech (5.4) a (5.5) se ukázalo, že tímto způsobem se povedlo ztrátám kroků ve většině případů zamezit, nicméně v některých situacích (při prudkých změnách směru pohybu) stále docházelo k nepatrnému protočení hřídele a ztrátě kroků. Pokud bych více omezil maximální zrychlení autonomního hráče, rychlost, s kterou by se dokázal po hřišti pohybovat, by již nebyla dostatečná. Rozhodl jsem se proto pro kamerovou kalibraci lokalizačním systémem, který bude lokalizovat pad a údaje o jeho pozici posílat řídicímu systému. Řídicí systém bude provádět kalibraci v čase, kdy bude pad stát na místě v obranné pozici a čekat na protiútok. Tímto způsobem řízení tedy nedokáži ztrátám kroků předcházet, nicméně jejich ztrátu budu schopen řešit a opravit. Omezení zrychlení je minimalizovalo tak, aby nedocházelo k výrazným ztrátám během doby mezi kalibracemi.

5.3 Řízení polohy padu

V návaznosti na řízení rychlosti je nutné vytvořit řídicí systém, který dokáže řídit polohu padu na hracím stole. Toto řízení se bude zakládat na odchylce mezi svou pozicí a referencí polohy. Tato odchylka, označená e_x a e_y, bude vstupním parametrem regulačního systému

$$\begin{aligned} e_x &= x_{ref} - x \\ e_y &= y_{ref} - y. \end{aligned} \tag{5.6}$$

Prvky v rovnici (5.6) s indexem ref značí souřadnice referenční polohy. Jelikož jsem již vyřešil řízení rychlosti, není potřeba řešit zrychlování padu a regulační systém se může rozhodovat jednoduchým způsobem stejně pro obě osy

$$v_m = \begin{cases} v_{max} & (e > 0) \\ 0 & (e = 0) \\ -v_{max} & (e < 0) \end{cases}. \tag{5.7}$$

Prvním problémem, který v této interpretaci regulace nastává, je situace, kdy pad projel referenčním bodem bez zastavení. K této situaci dochází, když je řídicí smyčka pomalejší než smyčka řídící pohyb motorů, tudíž mezi jedním cyklem řídicí smyčky dojde k inkrementaci polohy padu o dva kroky. Tuto situaci jsem řešil definováním intervalu okolo odchylky blížící se nule. Pad, zastaví již při výskytu v tomto intervalu, místo aby zastavil až při nulové odchylce. Experimentováním se ukázalo, že ke správné funkci stačí definovat pouze pět kroků široký interval. Tímto se podmínky regulačního systému změní na

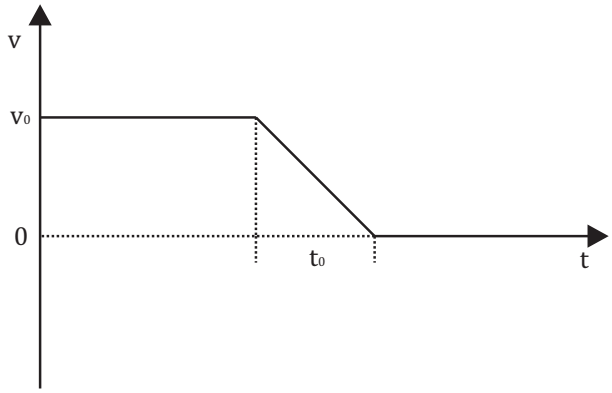
$$v_m = \begin{cases} v_{max} & (e > 0) \\ 0 & (|e| < 2) \\ -v_{max} & (e < 0) \end{cases}. \tag{5.8}$$

Druhým problémem je, že v případě, kdy pad dorazí do oblasti své reference, zastaví bez omezení maximální akcelerace. Tím opět dochází ke skokové změně rychlosti, která způsobit ztrátu kroků. Regulační systém tedy musí rozhodnout, kdy je správný čas začít zpomalovat.

5.3.1 Pohyb po S-křivce

Úloha pozvolného zrychlování a zpomalování při pohybu mezi dvěma polohami se nazývá řízení pohybu po S-křivce. Vzhledem k tomu, že pozvolné zrychlování je již vyřešené, cílem této úlohy je pouze sestavit potřebnou rovnici, která vyhodnotí, kdy je potřeba, aby pad začal brzdit. Tato úloha má jedno hlavní omezení, a tím je opět omezení maximálního zpomalení.

Uvažuji, že pad se pohybuje rychlostí v_0. Zároveň má stále k dispozici informaci o vzdálenosti zbývající do cíle pohybu e.



Obrázek 5.6: Zpomalení padu z rychlosti v_0

Pad může zpomalovat maximálním zrychlením a_{max}. Podmínka, která vyjadřuje zastavení padu zpomalením z rychlosti v_0, je

$$0 = v_0 - a_{max} \cdot t_0. \tag{5.9}$$

Čas t_0 je čas potřebný k zastavení. V rovnici (5.10) vyjádřím dráhu s, kterou pad ujede, než dojde k jeho zastavení

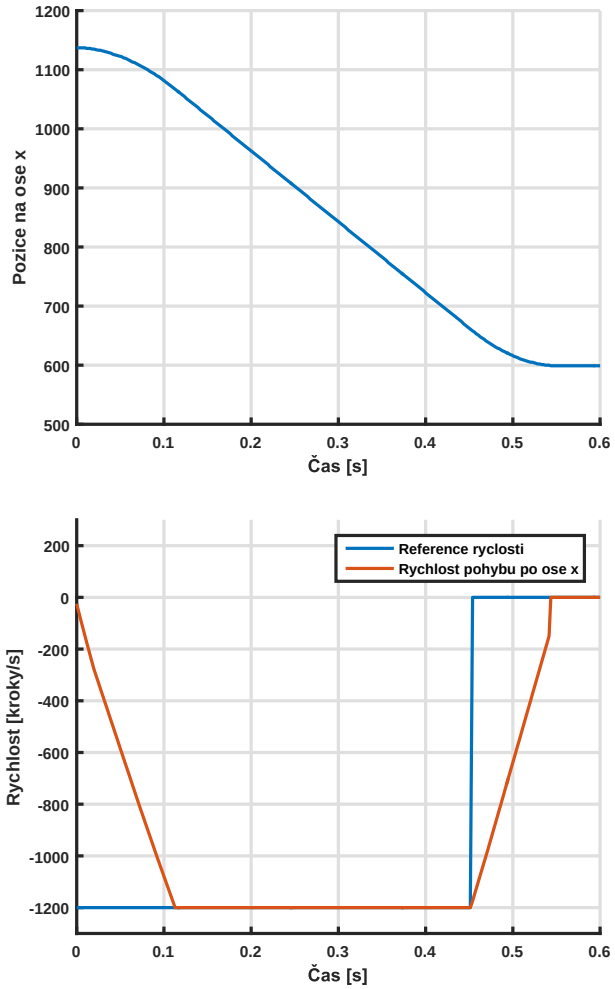
$$s = v_0 \cdot t_0 - \frac{1}{2} a_{max} {t_0}^2. \tag{5.10}$$

Z rovnice (5.9) vyjádřím čas t_0 potřebný k zastavení padu a dosadím ho do rovnice (5.10). Rovnice získá tvar

$$s = \frac{{v_0}^2}{2a_{max}}. \tag{5.11}$$

Podmínkou pro počátek zpomalování padu bude, jestli tato dráha s bude menší než aktuální vzdálenost padu od referenční polohy (odchylka e). Z hlediska řízení rychlosti se v čase zpomalení nastaví nulová referenční rychlost. Řídicí systém rychlosti poté zajistí omezení maximálního zpomalení.

Po implementaci této podmínky do řídicího systému jsem provedl měření pohybu padu z bodu 1138 do bodu 600 na ose x. Výsledky tohoto měření jsou zobrazeny na obrázku (5.7).



Obrázek 5.7: Pohyb padu z bodu 1138 do bodu 600 na ose x

Z měření je vidět, že pad v konečné fázi nezabrzdil včas, ale zastavení proběhlo skokem zhruba z rychlosti 170 kroků za vteřinu. Takovýto skok nezpůsobí žádné protočení hřídele motoru. Problém nastával při pohybech, kdy pad naopak dobrzdil příliš brzy. V takovém případě se stávalo, že vůbec nedorazil do požadované pozice. Nastavení času, od kterého pad začal zpomalovat jsem proto nadsadil tak, aby vždy dorazil do požadované polohy dříve než dobrzdí a zanedbal malý skok na konci rampy.

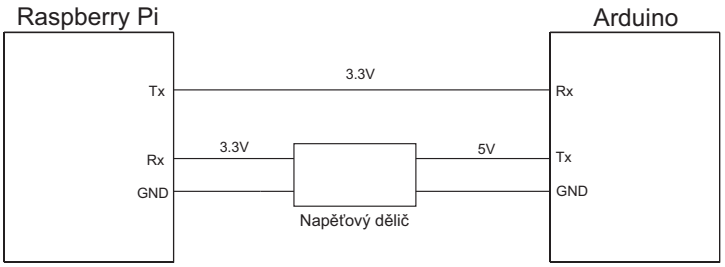
Kapitola 6

Strategie hry a predikce trajektorie puku

6.1 Přenos data mezi systémy

V tuto chvíli je již připravené řízení krokových motorů a je připravena i lokalizace puku a padu pomocí kamery. Zatímco lokalizaci zajišťuje Raspberry Pi, zbytek řídicí úlohy vykonává Arduino. Nutností je zajistit přenesení dat mezi těmito zařízeními sériovou linkou.

Problémem u této komunikace je napětí, které zařízení využívají. Zatímco Raspberry Pi komunikuje v rozsahu 0−3,3 V, Arduino využívá rozsah 0−5 V. Arduino sice dokáže 3,3 V rozlišit jako logickou jedničku, ale problém vzniká na straně Raspberry Pi, které by mohla logická jednička z Arduina pěti volty poškodit. Do komunikační sběrnice musím tím pádem zařadit napěťový dělič, který upraví logickou hodnotu signálu Arduina na odpovídající napěťovou úroveň přípustnou pro Raspberry Pi.



Obrázek 6.1: Blokové schéma sériové komunikace

6.2 Rozhodovací kritéria strategií

Řízení autonomního hráče se musí zakládat na několika kritériích, podle kterých se bude rozhodovat. Chování robotu rozdělím do třech základních kategorií. První kategorií je příprava na obranu, druhou obrana a třetí útok. Podle toho, v jakém stavu se puk nachází, se robot rozhodne, jak se bude chovat. Základním kritériem je samotná poloha puku. Hřiště je rozděleno

na dvě poloviny, přičemž autonomní hráč dokáže puk odehrát pouze na své polovině. Kritériem je překročení poloviny hřiště pukem z hlediska osy x

$$x > X_{\text{HALF}}. \tag{6.1}$$

Druhým kritériem je rychlost pohybu puku. Rychlost puku získám z dvou posledních poloh puku a časové značky dt, kterou společně se souřadnicemi zasílá Arduinu lokalizační systém sériovou linkou. Časová značka zaznamenává uplynulý čas mezi vyhodnocenými daty o poloze puku. Rychlost rozložím do jednotlivých složek obou os

$$\begin{aligned} v_x &= \frac{x_{new} - x_{old}}{dt} \\ v_y &= \frac{y_{new} - y_{old}}{dt}. \end{aligned} \tag{6.2}$$

Posledním kritériem je směr pohybu puku, který je významný zejména pro obrannou strategii. Stejně jako u prvního kritéria, i tento směr pohybu je důležité pozorovat pouze na ose x a primárně rozlišovat pouze směr pohybu směrem k brance autonomního hráče. Kritérium lze rozlišit podmínkou

$$v_x > 0. \tag{6.3}$$

Rychlost pohybu puku vyhodnocuji v jednotkách kroků za milisekundu, abych se vyhnul práci s vysokými čísly.

6.3 Obranná strategie

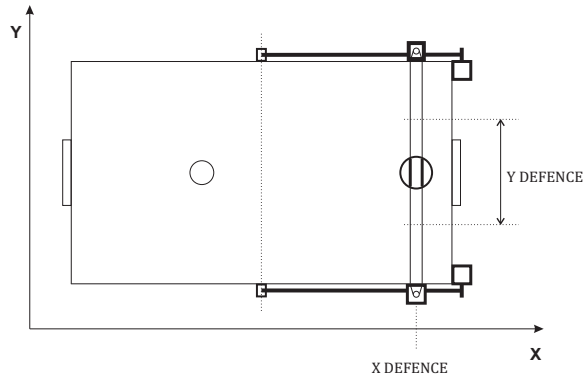
Obranná strategie zahrnuje co nejrychlejší obranu branky při prudkém úderu. Tento stav robot vyhodnocuje podle jediné podmínky

$$\sqrt{{v_x}^2 + {v_y}^2} \geq 15 \quad \wedge \quad v_x > 0. \tag{6.4}$$

Tato podmínka zahrnuje rychlost puku větší než 15 kroků za milisekundu. Pozorováním jsem zjistil, že při této rychlosti a přímém útoku musí pad provést co nejrychlejší obranu, aby dokázal útok odrazit. Druhou část podmínky, kdy je hodnota v_x větší než nula, jsem zvolil za účelem detekce pohybu směrem k brance robotu. Ve chvíli vyhodnocení začne robot predikovat trajektorii puku a snaží se nastavit pad tak, aby zabránil gólu.

6.3.1 Predikce trajektorie puku

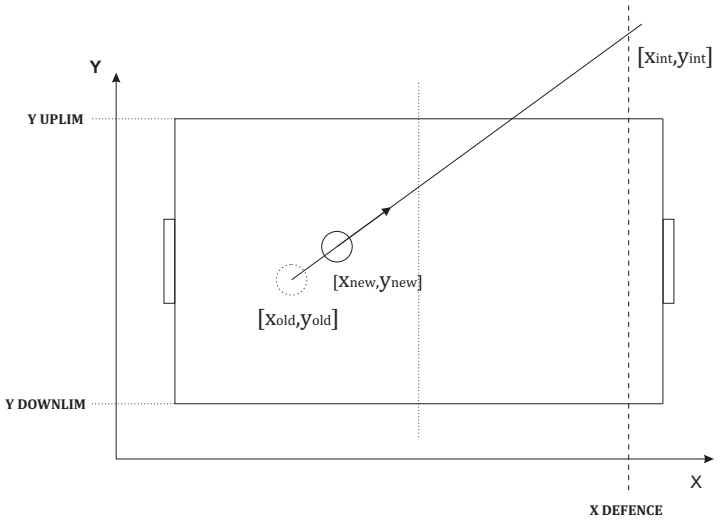
Predikci trajektorie systém vyhodnocuje při každé aktualizaci souřadnic puku, pokud se nachází ve stavu obranné strategie. Při obraně se pad pohybuje pouze na ose y v definovaném intervalu a na ose x se nachází v obranné pozici.



Obrázek 6.2: Znázornění obranné pozice

Cílem predikce bude detekovat souřadnici y, ve které puk protne obrannou osu X_{DEFENCE}. Do tohoto místa poté robot nastaví pad. Z pozorování se ukázalo, že pokud se puk při útoku odrazil dvakrát od hranic stolu, pohyboval se již tak pomalu, že robot již nevyhodnocoval obranu, ale rovnou přešel do útoku. Predikce proto bude uvažovat maximálně jeden odraz o kraj hracího stolu.

Predikce vychází ze znalosti posledních dvou souřadnic puku.



Obrázek 6.3: Predikování trajektorie puku

V první fázi přímku, po které se puk pohybuje, vyjádřím parametricky rovnicemi

$$\begin{aligned} x_i &= (x_{new} - x_{old}) \cdot t + x_{new} = p_x \cdot t + x_{new} \\ y_i &= (y_{new} - y_{old}) \cdot t + y_{new} = p_y \cdot t + y_{new}, \end{aligned} \tag{6.5}$$

kde x_i a y_i jsou libovolné souřadnice bodu ležící na přímce a t je parametr rovnice. Za x_i dosadím souřadnice obranné pozice X_{DEFENCE} (také souřadnice

průsečíku x_{int}), a tím mohu vyjádřit rovnici, ze které získám druhou souřadnici průsečíku y_{int}

$$y_{int} = \frac{(x_{int} - x_{new})p_y}{p_x} + y_{new}. \tag{6.6}$$

Pokud průsečík leží v mezích mezi hranicemi hřiště, vyhodnocuji tento bod jako cíl obrany a pad přemístím co nejblíže k tomuto bodu. Pokud ovšem průsečík splňuje podmínku

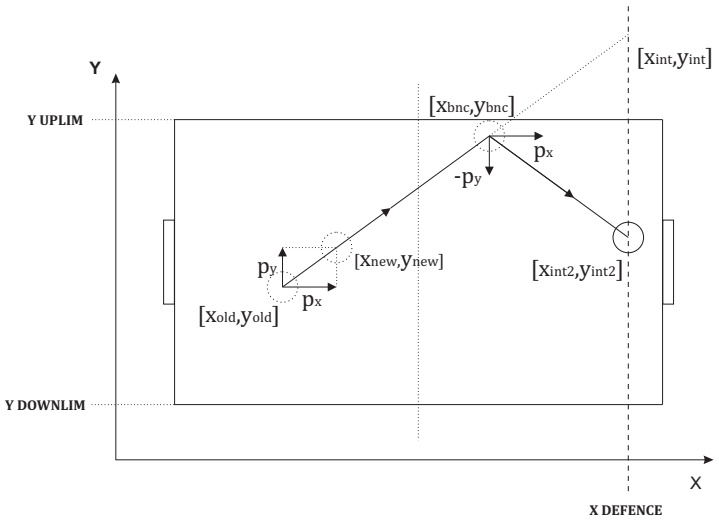
$$y_{int} > Y_{\text{UPLIM}} - r_{puck} \quad || \quad y_{int} < Y_{\text{DOWNLIM}} + r_{puck}, \tag{6.7}$$

znamená to, že puk se bude pohybovat po trajektorii s odrazem a je nutné provést další výpočet.

V tomto dalším výpočtu nejprve zjistím, v jakém místě se puk od hranice hřiště odrazí. Souřadnice odrazu označím x_{bnc} a y_{bnc}. Souřadnici y_{bnc} určím v závislosti na tom, kde se nacházel průsečík vypočítaný v rovnici (6.6). Podle pozice tohoto průsečíku určím, jestli se puk odrazí o horní, či dolní hranici. Souřadnice na ose y této hranice s ohledem na poloměr puku budou i souřadnicemi y_{bnc}. Druhou souřadnici průsečíku vypočítám z rovnice

$$x_{bnc} = \frac{p_x(y_{bnc} - y_{new})}{p_y} + x_{new}. \tag{6.8}$$

Poté, co jsem dopočítal bod, ve kterém se puk odrazí, tak, jak již vyplynulo z rovnice (2.6) u vektoru p_y, se po odrazu změní znaménko. Přepočítáním průsečíku totožným způsobem jako v rovnici (6.6) získám cílový bod obrany x_{int2}, y_{int2}.



Obrázek 6.4: Predikování trajektorie puku s odrazem

I v tomto případě se pad nastaví co nejblíže k tomuto bodu v mezích svého obranného pásma.

Pozorováním jsem zjistil, že při lokalizaci puku občas dochází k šumu způsobeného kmitáním konstrukce kamery. V těchto situacích poté docházelo k drobnému kmitání padu okolo obranné pozice. Z tohoto důvodu jsem implementoval robotu filtr, který se v průběhu predikce plní obrannými pozicemi a v momentě jeho naplnění začne průměrovat jeho obsah. Pokud se výsledek tohoto průměru příliš neliší od nově predikované obranné pozice, považuje robot výsledek z filtru za směrodatný. Tímto způsobem došlo k velmi dobrému potlačení tohoto šumu.

6.4 Útočná strategie

Útočná strategie může být pro autonomního hráče riskantním krokem. Při útoku se robot snaží zasáhnout puk tak, aby ho odrazil co největší rychlostí směrem od své branky. Rizikem pro robot je odkrytí své branky, které může být zásadní při rychlém protiútoku, nebo špatný úder, který může vést k vlastnímu gólu. Těmto situacím se snažím během útoku předcházet.

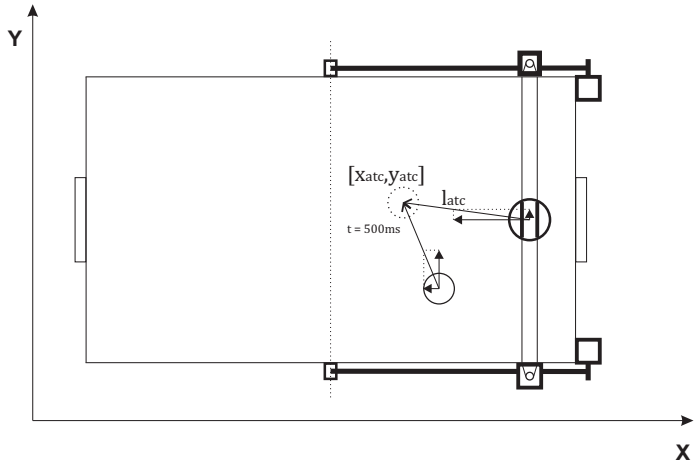
Prvním nutným kritériem pro vyhodnocení útoku je výskyt puku na straně robotu

$$x > X_{\mathrm{HALF}}. \tag{6.9}$$

Druhým kritériem je rychlost puku. Tato rychlost musí být menší než rychlost nutná pro obranu

$$\sqrt{{v_x}^2 + {v_y}^2} \leq 15. \tag{6.10}$$

V první fázi útoku robot predikuje polohu puku v horizontu 500 ms obdobným způsobem, jakým predikoval jeho pohyb u defenzivní strategie. V této poloze x_{atc},y_{atc} robot puk zasáhne. V další fázi robot dopočítá, jak dlouhou trasu l_{atc} do zásahu musí urazit a jakým směrem se má pohybovat.



Obrázek 6.5: Predikce bodu zásahu

Vzhledem k tomu, že chci puk zasáhnout pohybem po přímce, podle směru pohybu omezím maximální rychlost osy, po které se bude vykonávat kratší pohyb. Nejprve zjistím vzdálenost souřadnic padu od souřadnic bodu zásahu

$$\begin{aligned} \Delta x &= |x_{pad} - x_{atc}| \\ \Delta y &= |y_{pad} - y_{atc}|. \end{aligned} \tag{6.11}$$

Podle velikostí těchto rozdílů se rozhodnu, po které ose se bude pad pohybovat kratší dobu, a omezím jeho maximální rychlost. V případě zobrazeném na obrázku (6.5) by robot omezoval maximální rychlost pohybu po ose y, a to vzorcem

$$v_{ymax} = v_{max}\frac{\Delta y}{\Delta x}. \tag{6.12}$$

Ze znalosti rychlostí obou os dokážu vyjádřit celkovou rychlost, kterou se bude pad pohybovat

$$v_{pad} = \sqrt{v_{xmax}^2 + v_{ymax}^2}. \tag{6.13}$$

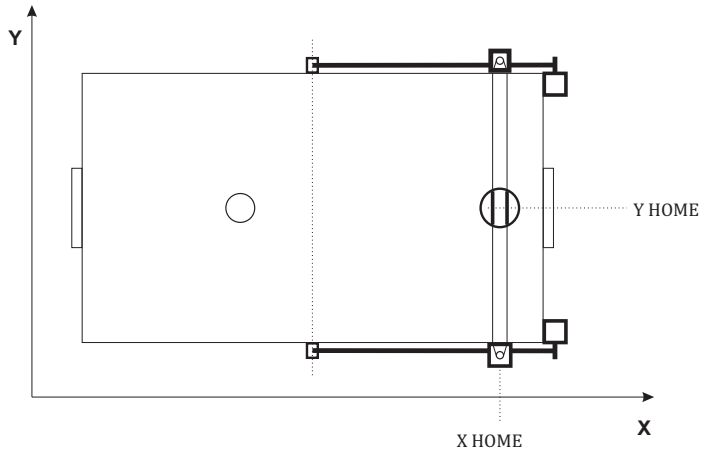
Z této rychlosti poté vypočítám čas, který pad na útok potřebuje

$$t_{atc} = \frac{l_{atc}}{v_{pad}}. \tag{6.14}$$

Se znalostí času potřebného na útok již dokáži útok načasovat tak, aby pad zasáhl puk ve správnou chvíli. Po zásahu puku se pad vrací do obranné pozice, aby byl co nejlépe schopný reagovat na náhlý protiútok.

6.5 Příprava na obranu

Tato strategie způsobí nastavení padu do obranné pozice před bránu, kde vyčkává na další postup hry. Toto chování je vyhodnoceno, pokud není splněna podmínka pro útok ani obranu.

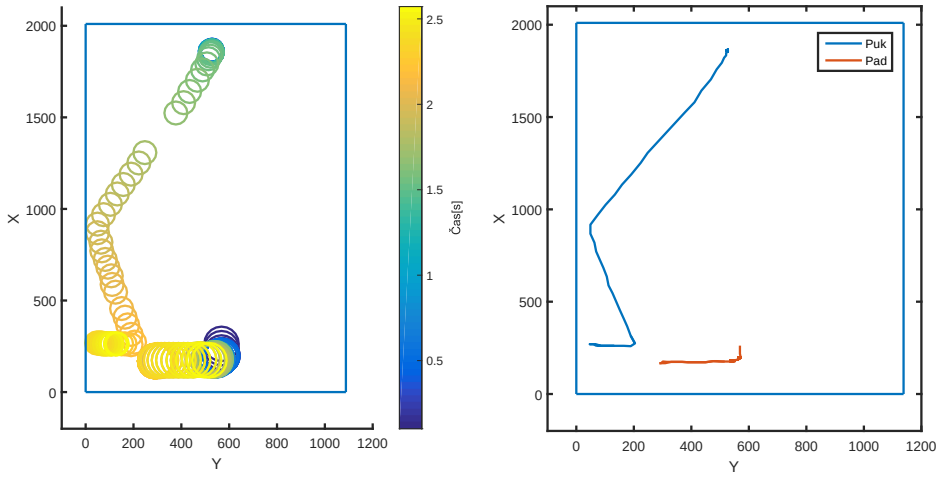


Obrázek 6.6: Pozice přípravy na obranu

V případě, že se již puk nachází v přípravné pozici, začne svou pozici kalibrovat. Kalibrace se provádí pomocí kamerového systému. Aby kalibrace byla co nejpřesnější, opět využívám filtru, který střádá údaje o pozici padu z kamerového systému do zásobníku. Po naplnění zásobníku dojde k jeho zpracování, vyhodnocení kalibrační polohy a k následné kalibraci.

6.6 Aplikace strategií

Po implementaci herních strategií jsem provedl sérii měření, ve kterých jsem pozoroval chování padu v různých situacích. Jako první situaci jsem testoval útok na autonomního hráče s odrazem o hranu hřiště. Při tomto úderu by pad měl vyhodnotit defenzivní strategii a zabránit gólu. Naměřená data tohoto útoku jsem zobrazil v obrázku (6.7).

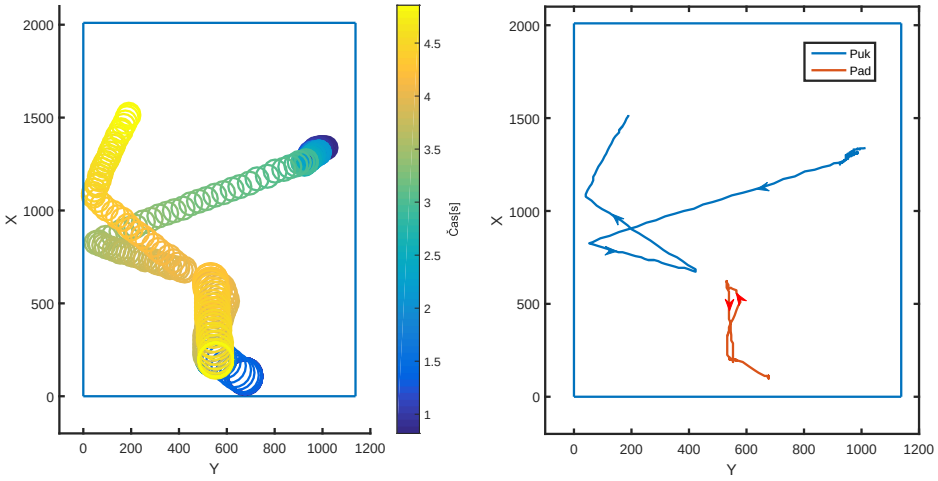


(a) : Časový záznam pohybu **(b) :** Trajektorie pohybu

Obrázek 6.7: Záznam chování padu při obranné strategii

Na obrázku je vidět trajektorie pohybu puku i padu. V tomto případě je patrné, že zásah padu vedl k odražení puku od branky a zabránění gólu. Ze záznamu je vidět, že pad začal reagovat již po získání prvních údajů o lokaci puku. Zároveň je vidět, že v průběhu útoku došlo ke krátké ztrátě informace o poloze padu. Tato chyba byla pravděpodobně způsobena špatnou detekcí polohy, kterou systém nebral v úvahu, aby nedošlo ke špatnému odhadu obranné pozice.

V dalším měření je zaznamenaný pomalý pohyb puku na polovině hřiště autonomního hráče. Tento pohyb splňuje kritéria pro to, aby pad provedl útok.

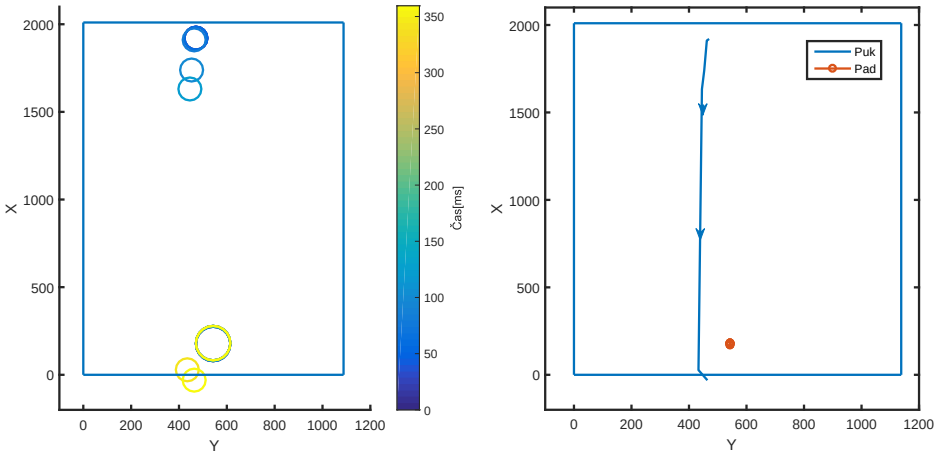


(a) : Časový záznam pohybu **(b) :** Trajektorie pohybu

Obrázek 6.8: Záznam chování padu při útočné strategii

Z tohoto měření se dá pozorovat, že pad začal provádět útok zhruba při prvním odrazu puku o levou hranici hrací plochy. Během útoku pad prováděl korekce útočné pozice, aby jistě zasáhl puk. Z toho důvodu útok není proveden po přesné přímce. Po odražení puku se pad ihned vrací do své obranné pozice, aby dokázal co nejrychleji reagovat na rychlý protiútok.

Ve třetím měření jsem zaznamenal situaci, ve které je veden prudký přímý úder.



(a) : Časový záznam pohybu **(b) :** Trajektorie pohybu

Obrázek 6.9: Záznam prudkého přímého útoku

Z měření zobrazeném na obrázku (6.9) je vidět, že systém nebyl schopen zareagovat a robot dostává gól. I přes to, že robot má k dispozici dvě první pozice útoku, z kterých vypočítal obrannou polohu, vlivem jeho omezeného

zrychlení se nedokázal do této polohy dostat včas. Dalším problémem je omezený počet pozic puku během útoku. Rychlost puku při tomto útoku je 5,3 m/s. Vzhledem k frekvenci snímání kamery 50 Hz by měl být systém schopný puk lokalizovat každých 10,6 cm , což odpovídá alespoň osmi měřením na délku hrací plochy. Z obrázku je patrné, že systém takto puk lokalizovat nedokázal. Tento problém může být způsoben rozmazáním snímku při rychlém pohybu puku, čímž puk na snímku změní barvu a tvar. Řešením by bylo využití kamery, která je schopná zaznamenávat snímky s kratší dobou expozice.

Kapitola 7

Závěr

V první fázi práce jsem se věnoval modelu pohybu puku po hrací ploše, abych lépe porozuměl fyzikálnímu chování puku na hracím stole. Ukázalo se, že vlivem proudění vzduchu je toto chování nelineární a parametr tlumení je silně závislý na rychlosti pohybu puku. Tuto závislost jsem vyjádřil polynomem druhého stupně a model jsem využíval převážně k simulaci herních strategií. V další fázi práce jsem řešil lokalizaci puku pomocí kamery a Raspberry Pi. V průběhu práce jsem narazil na problémy související hlavně s optimalizací rozpoznávacího algoritmu. Nejprve jsem k rozpoznávání využíval kruhového rysu puku, ale kvůli náročným výpočetním operacím jsem poté musel přejít k rozpoznání objektu podle barvy a velikosti objektu. Problémem u výpočetně náročných algoritmů bylo snížení FPS kamery na 20 snímků za vteřinu, což je příliš málo informací o pozici puku. Využitím zmíněné metody (rozpoznáváním objektu podle barvy a velikosti) se problém vyřešil. Lokalizovanou pozici puku jsem dále zasílal Arduinu po sériové lince.

V Arduinu ze získaných dat predikuji trajektorii, po které se bude puk pohybovat, a vyhodnocuji strategii podle které se bude autonomní hráč chovat. V této práci jsem aplikoval pouze defenzivní a základní ofenzivní strategii, nicméně vymýšlet další herní strategie se dá vcelku bez omezení. Obě strategie využívají predikce pohybu puku, aby dokázaly předvídat jeho chování a přizpůsobit mu svůj pohyb. Další strategií, kterou by bylo možné implementovat, může být například odpal puku přesným požadovaným směrem. Strategie tohoto typu je ovšem náročnější jak na požadavky lokalizačního systému, tak na požadavky akčních členů.

Po aplikaci herních strategií se ukázalo, že se pad stává celkem dobrým protihráčem, nicméně rychlost jeho odpalu není dostatečná a jeho hlavní schopností je hlavně obrana. Obranný systém dokáže útok ubránit při různě složitých odpalech o stěnu, ale vzhledem k malé velikosti hřiště a omezenému zrychlení krokových motorů pad nedokáže včas reagovat na přímé prudké pohyby. Dalším rozšířením hry by tudíž mohla být úprava herního stolu a výběr výkonnějších akčních členů.

Příloha A

Literatura

[1] *Air Hockey Robot*, http://www.jjrobots.com/air-hockey-robot-a-3d-printer-hack/ [online].

[2] *Air Hockey Robot Evo*, http://www.jjrobots.com/air-hockey-robot-evo/ [online].

[3] 3ucky(3all, *HSV cone*, 2017, https://de.wikipedia.org/wiki/HSV-Farbraum#/media/File:HSV_cone.png [online].

[4] Atmel Corporation, *Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V*, 2014, http://www.atmel.com/Images/Atmel-2549-8-bit-AVR/Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf [online].

[5] Opencv dev team, *OpenCV 2.4.13.2 documentation - Geometric Image Transformations*, 2017, http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.htmls [online].

[6] ———, *OpenCV 2.4.13.2 documentation-Feature Detection*, 2017, http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles [online].

[7] ———, *OpenCV 2.4.13.2 documentation-Finding contours in your image*, 2017, http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html [online].

[8] Pololu, *Pololu DRV8825*, https://www.pololu.com/product/2132 [online].

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: David K o p e c k ý

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Automatický vzduchový hokej (Air Hockey)

Pokyny pro vypracování:

Cílem individuálního projektu je dokončit a oživit konstrukci automatického vzduchového hokeje (Air Hockey), který částečně replikuje již realizovaný a zdokumentovaný projekt [1] a [2]. Hlavní úkoly jsou:

1. Dokončit HW a SW pro buzení a řízení motorů, aby bylo možné zadávat příkaz z RaspberryPi a motory se pohybovaly dostatečnou rychlostí pro zvládání hry a s dodržením omezení na zrychlení.
2. Připravit systém (kamera + SW) pro počítačové vidění, který bude schopný v obraze najít puk a případně oba hráče. Jeho výstupem bude poloha detekovaných objektů v souřadnicích hracího stolu.
3. Sestavit základní matematický model pro testování řídicích algoritmů a predikci polohy puku.
4. Implementovat herní strategii, která bude schopná automaticky bránit a útočit na základě informací od systému pro počítačové vidění.
5. Celý systém koncipujte tak, aby se dal snadno transportovat a byl vhodný pro prezentace.

Seznam odborné literatury:

[1] Air Hockey Robot EVO. http://www.jjrobots.com/air-hockey-robot-evo/ [online]
[2] Air Hockey Robot (A 3D Printer Hack). http://www.jjrobots.com/air-hockey-robot-a-3d-printer-hack/ [online]
[3] Corke, Peter. Robotics, vision and control: fundamental algorithms in MATLAB. Vol. 73. Springer, 2011.
[4] Franklin, Gene F., et al. Feedback control of dynamic systems. Vol. 3. Reading, MA: Addison-Wesley, 1994.

Vedoucí bakalářské práce: Ing. Jiří Zemánek

Platnost zadání: do konce letního semestru 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 20. 2. 2017