ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra aplikované elektroniky a telekomunikací

DIPLOMOVÁ PRÁCE

Alternativní HW pro závod autonomních vozidel

vedoucí práce: **Ing. Petr Weissar Ph.D.**
autor: **Pavel Votava**

2012

zadání scan

Anotace

Předkládaná diplomová práce je zaměřena na vývoj alternativního HW pro závody autonomních vozidel, doplněna o potřebné SW algoritmy. Konkrétně na auto schopné samonavigace po předem definované černé čáře. První část práce popisuje jednotlivé komponenty použité pro řízení vozidla a ladění řídících algoritmů. V druhé části je podrobně popsán problém implementované neuronové sítě, použité pro samonavigaci vozidla.

Klíčová slova

Autonomní auto, The Freescale Cup, Senzor černé čáry, FPGA, Modelářské servo, H-můstek, Bluetooth modul, RC souprava, Vektorové řízení, Neuronová síť, Perceptron, Back propagation, Forward propagation, Cost function

Abstract

Alternative HW for autonomous car race

The presented diploma thesis focuses on the development of alternative HW used for autonomous car races. The HW is supplemented with necessary SW algorithms. The thesis specifically uses the example of a car capable of self-navigation along a black line defined beforehand. First part of the thesis describes particular components used for driving the vehicle and debugging operating algorithms. The second part of the thesis elaborates on the matter of implemented neural network used for self-navigation of the vehicle.

Key words

Autonomous car, Smart car, The Freescale Cup, Black line sensor, FPGA, servo model, H-Bridge, Bluetooth module, RC set, Vector control, Neural network, Perceptron, Back propagation, Forward propagation, Cost function

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 6.5.2012 podpis diplomanta

……..

Poděkování

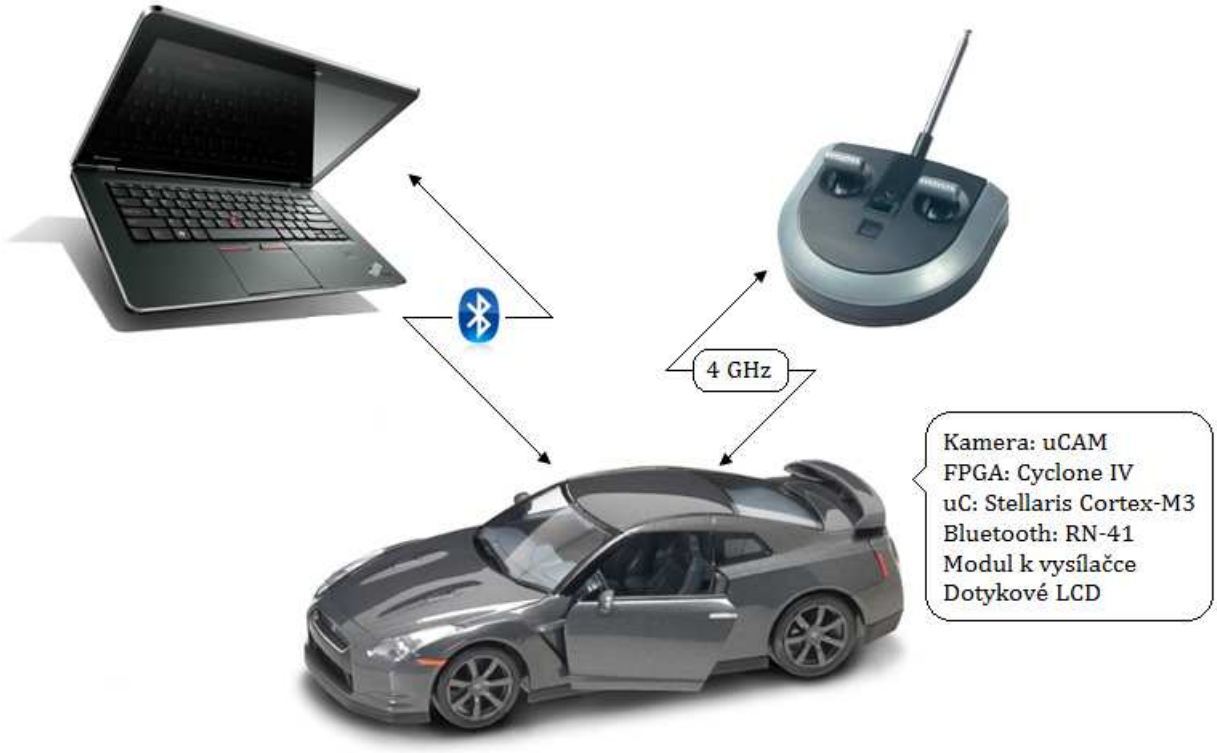
Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Petru Weissarovi Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce. Rád bych také vyjádřil poděkování své rodině, přítelkyni, přátelům a kolegům za podporu při psaní práce.

Obsah

Seznam obrázků

1 Úvod

Vytvořil jsem komplexní systém s několika možnostmi ovládání autonomního autíčka. Autíčko je samostatná jednotka, schopná samonavigace po černé čáře pomocí neuronové sítě typu Perceptron nebo jednoduchého vektorového řízení. Navíc je kdykoliv možný uživatelský zásah do řízení z libovolného počítače přes rozhraní bluetooth pomocí libovolné terminálové aplikace, nebo speciálně napsaného programu na ovládání a kompletní kontrolu nad autem. Druhým způsobem možného uživatelského zásahu je RC souprava z původní konstrukce auta, kterou jsem upravil a připojil k řídícímu procesoru. Pro tyto dvě možnosti uživatelského zásahu bylo nutné vytvořit prioritní systém pro přebírání a poskytování kontroly systému nad autem. Vyšší prioritu řízení jsem přidělil PC, a tedy připojení přes bluetooth, které okamžitě po připojení přebere kontrolu nad autem a bez předání priority RC soupravě jí není možné auto ovládat. Popsaný systém je nastíněn na obrázku 1.1.



Obrázek 1.1: Popis systému
Zdroje obrázků [1]

2 The Freescale Cup

2.1 Představení soutěže

The Freescale Cup: Intelligent Car Racing je celosvětová soutěž pro studenty vysokých škol, kde studenti tvoří auto schopné jezdit po černé čáře bez vnějšího zásahu obsluhy. Soutěž vyžaduje a rozvíjí znalosti studentů v oblastech embedded programování a tvorby jednoduchých obvodů pro kontrolu motoru, serva a pomocných senzorů. S touto vizí, jakožto výukového prostředku jsem navrhoval alternativní HW.

O kvalitě soutěže svědčí její historie a bohatý seznam zastoupených universit z celého světa. The Freescale Cup, dříve známy jako Smart Car Race začal již v roce 2003 na Korejské universitě Hanyang University, kde se zúčastnilo 80 studentských týmů, což je i na první ročník poměrně velké číslo. Od té doby se pohár rozšířil do mnoha zemí po celém světě zahrnující země jako Čína, Indie, Malajsie, státy Severní a Latinské Ameriky a samozřejmě Evropy. Ročně se ho zúčastní na více než 500 universit a 15,000 studentů.

Freescale Cup se neustále rozšiřuje na stále více universit po všech světových regionech ohromnou rychlostí. V posledních dvou letech se například v Indii rozrostl počet zúčastněných týmů o více než polovinu, ze 100 na 250. V Latinské Americe se počet lokací rozšířil z jediné na čtyři v Mexiku a přibyla jedna v Brazílii. Tento rok se přidává hodně týmů z Evropského kontinentu, zahrnující Německo, Francii, Rumunsko, Českou Republiku a další země EU. Ve skutečnosti mnoho škol po celém světě hostuje svoje vlastní soutěže zahrnující projekty jejich magisterských studentů a nejrychlejší posílá do pohárového mistrovství The Freescale Cup.

Firma Freescale se neustále snaží rozvíjet tuto soutěž a zpestřovat ji různými aktivitami. Například tento rok pozvali tři nejlepší týmy z Číny, Mexika a Spojených států aby soutěžily na Freescale Technology Forum v San Antoniu v Texasu.

Na samotný návrh auta jsou dána velmi striktní pravidla a studenti musí použít jeden ze čtyř vybraných typů kamery, zvolit si jeden ze dvou dovolených mikrokontrolerů, a bez výběru jsou povinné části jako H-můstek, stejnosměrný motor, typ baterie, serva a v podstatě i programového vybavení ve formě CodeWarioru. Volitelné jsou pouze přídavné senzory, které jsou podle pořadatelů nezbytné, a jejich maximální počet je omezen na 16 kusů, přičemž řádková CMOS kamera se bere jako senzor jeden.

2.2 Vybraná pravidla soutěže

Jak jsem již zmínil v předchozí kapitole, soutěžní pravidla jsou velmi jasně stanovena a jejich sebemenší nedodržení může vést k diskvalifikaci ze soutěže. Toto je samozřejmostí kvůli konkurenceschopnosti a vyrovnanosti všech zúčastněných týmů z hlediska hardwarových prostředků jako šasi, pneumatiky, DC motor, servo, baterie. Nesmí být použity DC-DC boost obvody pro zvyšování napětí pro motor nebo servo mechanismus. To vše se dá pochopit a zdá se takřka samozřejmostí, na druhou stranu za sporné už můžeme považovat dodržení výběru jedné ze čtyř kamer. Kde každá kamera je jiná a její výběr je důležitý pro zvolený typ řízení a využití kamery, ale nevidím důvod pro použití jiné kamery byť parametry podobné jedné z kamer předepsaných. V mém řešení jsem například volil kameru velmi podobnou jedné z vypsaných byť podle mého vhodnější. To samé se dá říci i o použití H-můstku pro řízení výkonu motoru a také o výběru mikrokontroleru, jejichž typ volí firma Freescale samozřejmě ze součástek, které sama vyrábí.

Ale teď již jen velmi jednoduše k pravidlům evropské části soutěže, podrobnější informace lze nalézt na internetových stránkách [1]. Vybral jsem pouze některé důležité specifikace, spíše jen pro představu jaké by mělo mít závodní auto parametry, jelikož tyto pravidla neovlivnily vývoj alternativního HW vozidla.

Blok 1: Definuje složení studentských týmů na maximum 3 studentů bakalářského studia, maximální počet 3 týmů na universitu a klade důraz na samostatnou práci studentů.

Blok 2: Definuje šasi vozu a možnosti hardwarového vybavení. Zde uvedu pro představu jen elektronické prvky, ke kterým jsem volil alternativy. O specifikacích samotného vozu není nutné se rozepisovat, jelikož na tuto část nebude práce zaměřena.

- není dovoleno používat jiné programovatelné zařízení kromě mikrokontroleru MPC5604B, který může být pouze jeden

- DC-DC boost obvody nejsou povoleny pro regulaci výkonu motoru ani serva

- maximální kapacita všech kapacit nesmí v součtu přesáhnout 2000uF a nejvyšší dovolené nabíjecí napětí jednoho kapacitoru je 25V

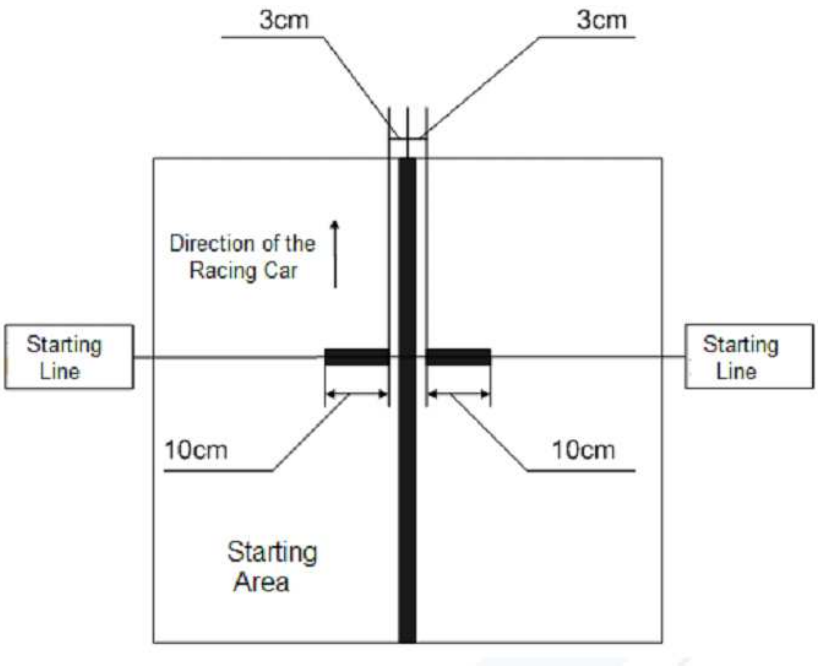
- jsou dovolena nanejvýše 3 serva a 16 senzorů, načež CMOS kamerový modul je brán jako jeden senzor

- musí být využit jeden z uvedených kamerových modulů:

    a. Parallax TSL1401-DB
    b. LinkSprite LS-Y201
    c. Toshiba TCM8240MD RB-Spa-115
    d. CM-26N

- také jeden z vybraných H-můstků: MC33931 nebo MC33932, nebo diskrétní analogové komponenty

Blok 11: Definuje vlastnosti trati, z čehož je nejdůležitější minimální šířka černé čáry na bílém podkladu, která činí 25mm/1in a je důležitá pro rozlišení kamery. Dále minimální rádius zatáčky 500mm/19.7in, který je důležitý pro zatáčivost auta. A na neposledním místě definice startovací oblasti, která je zobrazena na obrázku 2.1.



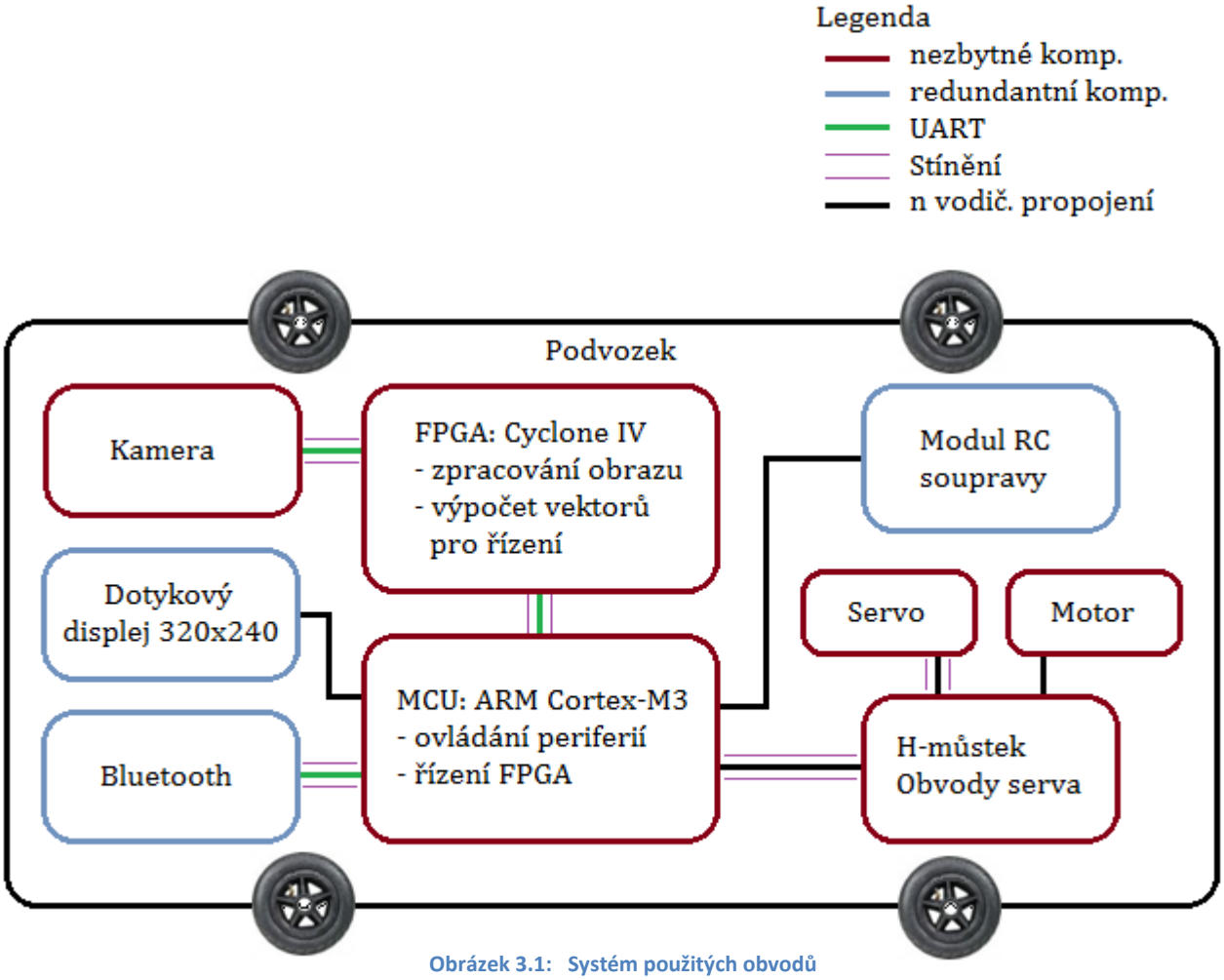
Obrázek 2.1: Definovaná startovní oblast
Zdroje obrázků [2]

Z mého pohledu byl blok 11, tedy definice trati a startovní oblasti jediné potřebné kritérium, kterému jsem musel přizpůsobit vývoj. Jelikož by auto mělo být schopné jezdit na stejné trati jako ostatní závodní auta. Zvolil jsem tedy šasi auta podobné svou velikostí a maximálním poloměrem zatáčení předepsanému závodnímu. Další předepsané parametry z hlediska povoleného HW vybavení jsem nebral v úvahu a nahradil je svojí vlastní alternativou, se kterou se v konečném důsledku samozřejmě nebudu moci soutěže zúčastnit, a tak bohužel nedostanu zpětnou vazbu na výsledky své práce, z tohoto hlediska.

3 Popis systému autonomního autíčka

Celý systém autonomního autíčka se skládá z několika subsystémů, které většinou pro hlavní funkci auta, tedy sledování černé čáry, nejsou potřeba a slouží pouze pro lepší komfort při vývoji řídícího programu nebo uživatelskému zásahu. Nejpodstatnější a jedinou nezbytnou částí pro funkci je pouze kamerový systém a systém kontroly periferií, tedy serva a motoru. Ostatní komponenty jako dotykový LCD panel umístěný na střeše auta, bluetooth modul a RC souprava jsou redundantní. V této kapitole postupně popíši jednotlivé části systému jak z hlediska hardwaru tak softwaru. Na ilustračním obrázku 3.1 pod popisem je uveden kompletní systém komponent a řídících obvodů obsažený v autíčku.

Všechny komponenty jsou určeny k jednoduchému principu řízení. Obraz snímaný kamerou se zpracuje v logickém obvodu FPGA, kde se také vypočítá „řídící vektor“ obsahující informace pro nastavení polohy serva a rychlosti motoru. Vektor je ihned po vypočítání přeposlán do procesoru, který si podle informací přenastaví PWM generátory pro H-můstek a servo. Mezi obvodem FPGA a procesorem probíhá samozřejmě další komunikace například pro přepínání řízení mezi neuronovou sítí a vektorovým řízením a další nadbytečné komunikace pro snazší ladění algoritmů a větší kontrolu nad autem. Jednotlivé subsystémy budou dále popsány v samostatných podkapitolách.



Obrázek 3.1: Systém použitých obvodů

3.1 Volba senzoru černé čáry

Nejdůležitější otázkou celého projektu, kterou bylo nutné vyřešit, byla vhodná volba senzoru na rozpoznání černé čáry. V úvahu připadalo hned několik možností.

První a asi nejjednodušší a ekonomicky nejvýhodnější volbou by bylo použít systém s LED diodami ve funkci osvícení dráhy a fototranzistorů snímajících intenzitu odraženého světla přicházejícího zpět z vozovky. Tento systém má jasné výhody v jeho jednoduchosti, snadnému sběru informace na A/D převodníku v dnešní době obsahující téměř každý mikroprocesor a hlavně rychlosti vyhodnocení dat bez potřeby rozsáhlých výpočtů, jelikož senzorů by bylo jistě podstatně méně než například pixelů z obrazu kamery. Na druhou stranu použití takto jednoduchého obvodu by zajistilo informaci jen z několika míst pod podvozkem auta, kde by byly senzory umístěny, což je jasná nevýhoda oproti kamerovému systému, který při správném nastavení dodá informaci i o vozovce několik decimetrů před autem a zbude tedy dostatek času pro výpočet a jemnější nastavení řízení.

Další možnou volbou je použití řádkové kamery. Tyto kamery dávají jako výstup *n*-bitový datový vektor namísto matice *N*x*M* jako standardní kamery. Využití této řádkové kamery by fungovalo podobně jako systém s LED diodami a fototranzistory, vylepšený o získávání informace o charakteru čáry na vzdálenost nastavenou před autem. Zbyl by tedy dostatek času na výpočet řídícího vektoru a nastavení řízení. Navíc by tyto výpočty nemuseli být nijak složité a hlavně rozsáhlé, stačil by tedy i pomalejší výpočetní systém s větší úsporou energie, než při použití standardní kamery s informačním výstupem *N*x*M*.

Poslední rozmýšlenou možností bylo využití standardní kamery právě s obrazovým výstupem *N*x*M* bitů. Jedinou výhodou takovéto kamery je právě tato datová matice, která poskytne daleko více informace a umožní daleko lepší citlivost nastavení řízení serva i větší možnou maximální rychlost otáčení motoru. Tím se ale objevuje jiný problém a to požadavek na větší výpočetní výkon celého systému zpracovávajícího obraz. Já jsem pro svou práci volil tuto volbu, jelikož má být auto určeno jako alternativa závodního auta a pro to je velká maximální rychlost nepostradatelnou součástí.

3.1.1 Parametry kamery

Na trhu je nabízeno několik typů kamerových modulů s různými parametry, více či méně vhodnými pro tento projekt. Asi nejpodstatnější pro mne bylo vybrat kamerový modul se začleněným předzpracováním obrazu a vhodným komunikačním rozhraním jednoduše začlenitelným do systému bez potřeby dalších redundantních obvodů. Což značně zúžilo výběr jen na pár kamer, jelikož většina modulů obsahovala výstup s televizním NTSC nebo PAL signálem a to by vyžadovalo trigger na další zpracování obrazu. Podařilo se mi tedy najít moduly s implementovaným komunikačním rozhraním UART nebo SPI jak pro nastavení parametrů kamery, tak pro posílání obrazové informace.

Ve stručnosti tedy shrnu parametry vybraného modulu, který jsem zvolil v závislosti na dostupnosti, ceně, a samozřejmě na dále vypsaných elektronických parametrech.

Výrobce: 4D SYSTEMS
Názen: uCAM – TTL Serial Camera Module
Napájení: 3.3VDC @62mA
Komunikační rozhraní: UART s maximální přenosovou rychlostí 1,2Mb/s
Formát obrazu: JPEG snímky nebo nekomprimovaný RAW formát
Rozlišení: 60x80, 240x320, 480x640 a další dohádatelné v Datasheetu [2]
Barevné rozlišení: od 2bitové černobílé do 16b barevné škály

Toto jsou asi nejdůležitější parametry kamerového modulu zobrazeného na obrázku 3.2 s OmniVision OV7640/8 VGA barevným senzorem a JPEG kompresním čipem, který nepotřebuje externí DRAM paměť.

Obrázek 3.2: uCAM – TTL
Zdroje obrázků [3]

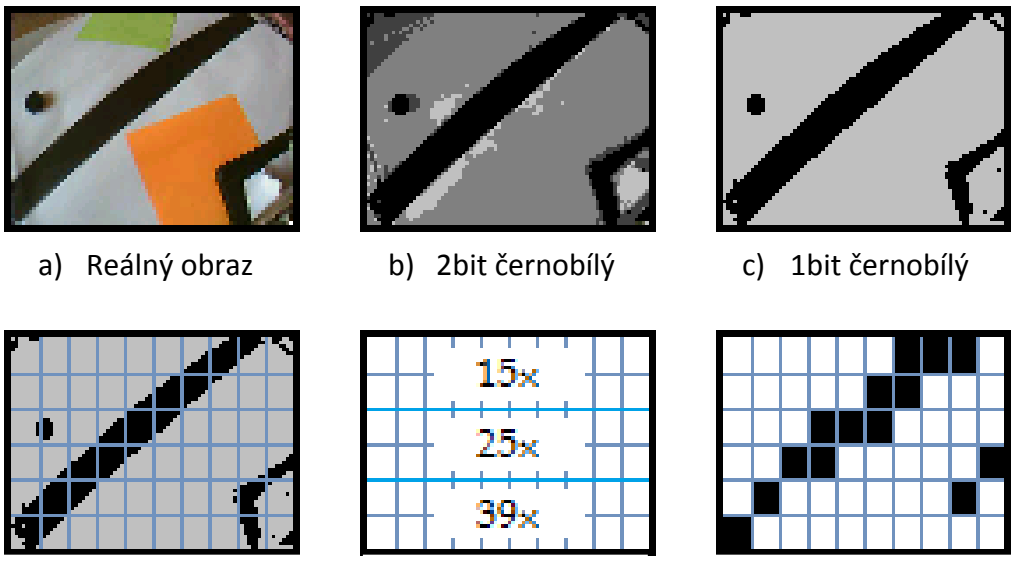
Já používám kameru s nastavením obrazového formátu RAW pro získání nekomprimovaných obrázků s 2bitovou černobílou hloubkou, s rozlišením 60x80 bodů a přenosovou rychlostí UARTu 115,2 kb/s. Zvolil jsem právě tuto komunikační rychlost, jelikož je to maximální autodetekovatelná rychlost kamerou, nemusím tedy mezi nastavením a přenosem obrazu měnit parametry fázového závěsu v obvodu FPGA. To samozřejmě nebyl hlavní důvod, ale při testu jsem zjistil, že rychlost je zcela postačující, fps (frames per second) při tomto nastavení dosahuje přibližně 5,4 snímku, což zcela postačuje a poskytuje to i dostatek času na výpočet řídících vektorů a přenastavení ovládání. Systém tedy nebude přehlcen novými obrazovými daty, aniž by stačil zpracovat předchozí a nebude ani dlouhá prodleva při čekání na nový snímek.

3.1.2 Zpracování obrazu

Ke zpracování obrazu používám obvod FPGA, o kterém se zmíním v další kapitole, nyní pouze představím systém zpracování a vyhodnocení snímků získaných z kamery. Jak bylo již zmíněno snímané rozlišení obrazu je 60x80 bodů s 2bitovou černobílou barevnou hloubkou, prostým vynásobením těchto parametrů lze zjistit, že jeden obrázek se skládá z 9600 informačních bitů plus synchronizační bity komunikačního protokolu. Dohromady se start a stop bitem bez parity to tedy dává rovných 12.000 bitů. To je na vyhodnocení a řízení docela hodně, proto provedu jakousi kompresi tohoto obrázku tím, že sníží rozlišení 80x a ze získané matice vypočítám řídící vektory takzvaným vektorovým řízením nebo pomocí neuronové sítě, volba záleží na uživateli.

Kompresi obrázku provádím tak, že nejprve převedu 2 bitovou černobílou barvu na 1 bitovou s tím parametrem, že za černou považuji pouze nejtemnější části obrazu. To sníží velikost dat na polovinu. Ve druhém kroku komprese rozdělím plochu obrazu na 60 velkých čtverců o velikosti 8x10 pixelů, které vypočítám s rozdílnou vahou v závislosti na vzdálenosti od předku auta, s vahami 15, 25 a 39. To znamená, pokud bude počet černých bodů v jednotlivých submaticích o velikosti 8x10, větší než dané váhy, bude jej výsledná 60ti prvková matice považovat za černý bod. Z toho vychází 60 prvková matice, ze které už se dají slušně a rychle vypočítat řídící vektory. Celá tato situace je v jednotlivých krocích i s váhovací maticí naznačena na obrázku 3.3. Kamera je nad autem umístěna ve výšce cca 17cm a sklonem přibližně 30° k vertikále směrem k přídi auta, díky tomuto nastavení vzniká na obraze malé zkreslení a kamera zabírá se zvětšující se vzdáleností směrem od auta úměrně větší plochu a šíře čáry se zmenšuje, to je důvod použití matice s různým váhováním. Nejen, že je komprese nesporně výhodná při zjednodušení výpočtu řídících vektorů, ale navíc ještě funguje jako jednoduchý filtr a případné šumy nebo malé nerovnosti na vozovce se vyfiltrují a neuronová síť ani vektorové řízení je neberou v potaz.

Z vypočtené 60 bitové matice se dále vypočítají dva řídící vektory, o kterých zatím mluvím pouze v abstrakci a slouží k nastavení polohy serva a rychlosti motoru. O vektorech se podrobně zmíním v dalších kapitolách (kapitola 4).



a) Reálný obraz b) 2bit černobílý c) 1bit černobílý

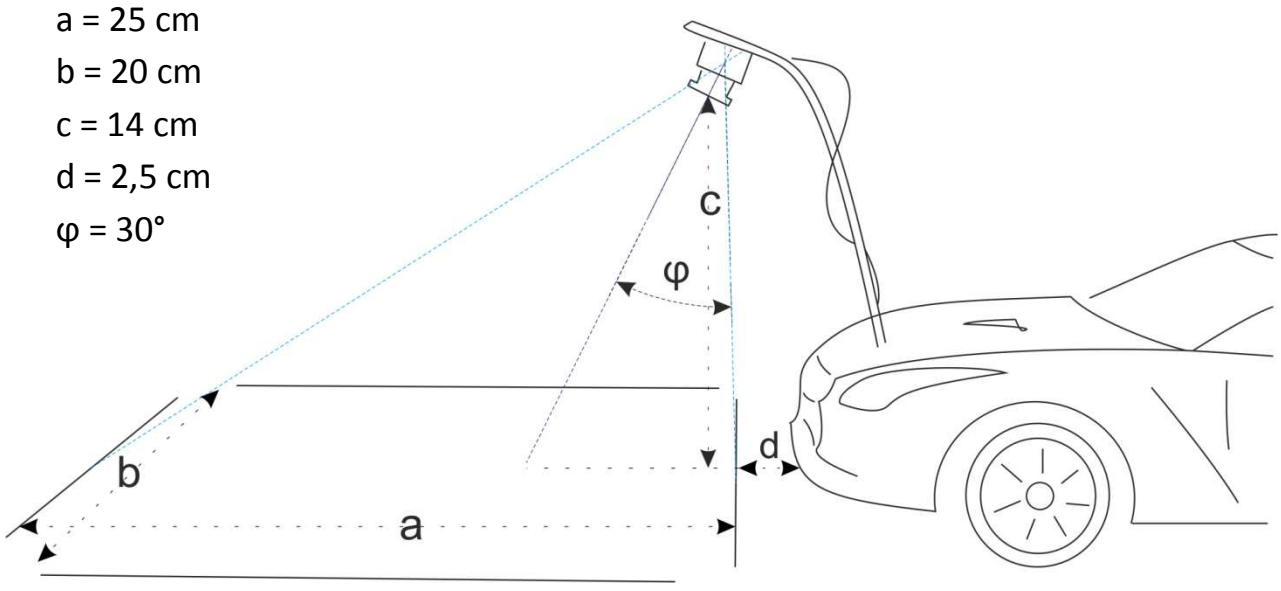
d) Rozdělení submatic e) Váhy submatic f) Výsledná 60b matice

Obrázek 3.3: Zpracování obrazu

Rozdělení komprimované obrazové matice právě na 6 řádků a 10 sloupců a ne jinak, má několik opodstatnění. 60 bitů se pohodlně vejde do 8 posílaných bajtů, ale hlavně jsem potřeboval větší rozlišení na vodorovné ose, kvůli dosažení lepší jemnosti zatáčení, zatím co rozlišení na horizontále mohlo být horší. Proto nepoužívám rozlišení 8x8 = 64 bitů.

Pro úspěšné zpracování a vyhodnocení snímaného obrazu, tedy černé čáry na bílém podkladu je nezbytné správně nastavit úhel pohledu a vzdálenost kamery nad vozovkou. Popřípadě upravit stávající algoritmy novému nastavení kamery, hlavně váhy submatic uvedené na obrázku 3.3e, aby sledovaná černá čára byla stále správně interpretována. Z hlediska mechanické konstrukce má na tuto skutečnost největší vliv nastavení úhlu pozorování φ, který je i s dalšími parametry nastíněn na obrázku 3.4. Pokud bude totiž sledovaný úhel o něco větší, naroste maximální dohled kamery před vozidlem, což by nevadilo vyhodnocovacím algoritmům z hlediska zatáčení. Jelikož by případná zatáčka byla viditelná jako pozvolnější a kola by se natočila o to méně, ale mohl by nastat problém s detekčními algoritmy černé čáry. Ty jsou založeny na již zmíněných vahách submatic, a protože viditelná tloušťka čáry rapidně klesá se snímanou vzdáleností, mohla by se čára vyhodnotit jen jako redundantní šum a algoritmy by ji mohly zanedbat. Nastíněná situace by nebyla nijak kritická, jelikož auto by pouze ztratilo informaci a vývoji černé čáry v dostatečné vzdálenosti pro rychlou jízdu, ale omezilo by predikci na menší detekovanou část, což by mělo za následek větší vyjíždění auta mimo trať.

Nejhorší situace nastane, při kompletní ztrátě detekovatelnosti černé čáry z příčiny přílišného vzdálení kamery od vozovky, čož je až dvojnásobek vzdálenosti aktuálního nastavení a položení úhlu φ do kritických cca. 60°. Při podpoření této situace neupravením algoritmu se stává čára nedetekovatelnou a auto není schopné správné jízdy.



Obrázek 3.4: Nastavení polohy kamery

Při správném nastavení není vyjetí auta mimo černou čáru, tedy mimo oblast, kdy je černá čára detekovatelná kamerou, nijak kritické. Algoritmy při této situaci srovnají kola do neutrální polohy a couvají s autem do doby, než znovu nenajede na černou čáru a nevyhodnotí alespoň 2 pixely 60 prvkové matice jako detekovanou černou čáru. Poté se pokračuje ve standardní jízdě. Tato korekce je dobrá i v případě, narazí-li auto na zatáčku s poloměrem zatáčení menším, než je minimální poloměr zatáčení automobilu, tedy nevytočí-li zatáčku napoprvé, jednoduše kousek couvne a vytočí se napodruhé. Což řeší i použití systému na autě s větším minimálním poloměrem natočení kol, použijeme-li tedy „nevhodné“ šasi vozu, algoritmus za nás tento nedostatek částečně kompenzuje. Problémem je, že se prodlužuje čas na objetí jednoho kola.

Aktuální nastavení vozu dle obrázku 3.4 a nastavení prahovacích vah submatic dle obrázku 3.3e, je přizpůsobeno podmínkám závodní trati soutěže The Freescale Cup. Nehledě na černou čáru, má trať definovanou nějakou šířku, dalo by se říci od mantinelů, a při aktuálním nastavení polohy kamery a detekčních algoritmů, by auto nemělo z dráhy spadnout, i kdyby přejíždělo menší most, či jinou překážku. To o kolik má auto možnost vyjet ven z dráhy, nebo spíše kamera ztratit černou čáru, je ovlivněno nastavenou maximální rychlostí auta a rychlostí snímání a zpracování obrazu z kamery. Čím rychleji totiž auto pojede, tím vzdálenější úseky trati budou kamerou snímány, při stejném nastavení fps, a tím je větší šance na přehlédnutí ostré zatáčky a nezbude než kousek couvnout.

Auto má aktuálně nastaveny 4 velikosti maximální rychlosti, které je možné upravovat z aplikace v PC, pomocí rozhraní bluetooth. Defaultně je nastavena druhá nejnižší rychlost, jelikož při ní auto velmi dobře stíhá kopírovat čáru a je schopné se pohybovat i při částečně vybitých bateriích. Úpravu maximální rychlosti jsem implementoval z důvodů, že auto nemá stejnou rychlost v závislosti na povrchu dráhy a hlavně na stavu vybití baterií.

Jak bylo již napsáno v úvodu ke kapitole, kamera dodává snímky 60x80 bodů celkem na 1200 datových bitech i s bity komunikačního protokolu, tedy start bit a stop bit, při rychlosti fps přibližně 5,4 snímku za sekundu. Prostým vynásobením převrácené hodnoty komunikační rychlosti obvodu FPGA s kamerou a daným počtem datových bitů, tedy (1/115200)*1200 = 12,5 by se mohlo zdát, že kamera musí dávat právě tento počet snímků za sekundu. Bohužel, jak je možné vidět na obrázku 3.8 v jedné z dalších podkapitol, protokol kamery vyžaduje další komunikaci, jako je vyžádání nového snímku, přijetí potvrzení této žádosti z kamery, následované samotným snímkem, odeslání potvrzení přijetí celého snímku do kamery a cca. 3ms čekání před možností vyžádání nového snímku, které není na obrázku znázorněno. Při uvážení všech redundantních událostí, které musí proběhnout navíc při získávání snímku a při blíže nespecifikovaných dobách odezvy kamery na jednotlivé příkazy se počet fps sníží přibližně na polovinu, při daném nastavení všech parametrů.

3.2 Obvod FPGA

Ke zpracování obrazu z kamery jsem zvolil logický programovatelný obvod FPGA Cyclone IV od firmy ALTERA. Obvod je pro funkci zpracování obrazu více než vhodný, pro svou výpočetní rychlost, variabilitu a jelikož komunikace s kamerou probíhá již v sériové digitální podobě TTL logiky není třeba dalších obvodů a kameru je možné jednoduše připojit na vstupní/výstupní piny obvodu FPGA včetně napájení.

Celý nezbytný systém autonomního auta by nepochybně mohl fungovat jen na použitém procesoru, nebo samostatně v obvodu FPGA. Já jsem použil oba obvody z důvodu jejich podle mého lepšího využití a z hlediska výukových účelů, ke kterým je soutěž primárně určena. Takto se může obvod FPGA starat o komunikaci s kamerou, kompresi obrazu a výpočet vektorů pro řízení, které ihned po vypočítání odesílá do procesoru k dalšímu zpracování. Procesor tímto není nijak zatěžován a může se bez problému a nezbytných prodlev starat o komunikaci s počítačem, vykreslování dat na dotykový LCD panel, obsluhu RC modulu vysílačky a sammozřejmě s nejvyšší prioritou o nastavení polohy serva a rychlosti otáčení motoru.

Obrázek 3.5: DE-0 Nano Development and Education Board
Zdroje obrázků [4]

3.2.1 Parametry zvoleného obvodu

Zvolená prototypová deska DE-0 Nano je svými parametry velmi vhodná pro použití v malém nenáročném robotickém systému. Deska vyhovuje jak počtem vstupních/výstupních pinů, možnostmi napájení, počtem logických elementů, paměťovými moduly, tak svojí „jednoduchostí“, jelikož neobsahuje nadbytečné periferní obvody, o které se dá v případě potřeby rozšířit. Má tak velmi kompaktní rozměry a malou váhu, což je nepostradatelná výhoda u tohoto projektu, kde musí být na malé ploše umístěno několik elektronických obvodů.

Parametry DE-0 Nano Development and Education Board:

Cyclone IV EP4CE22F17C6N FPGA

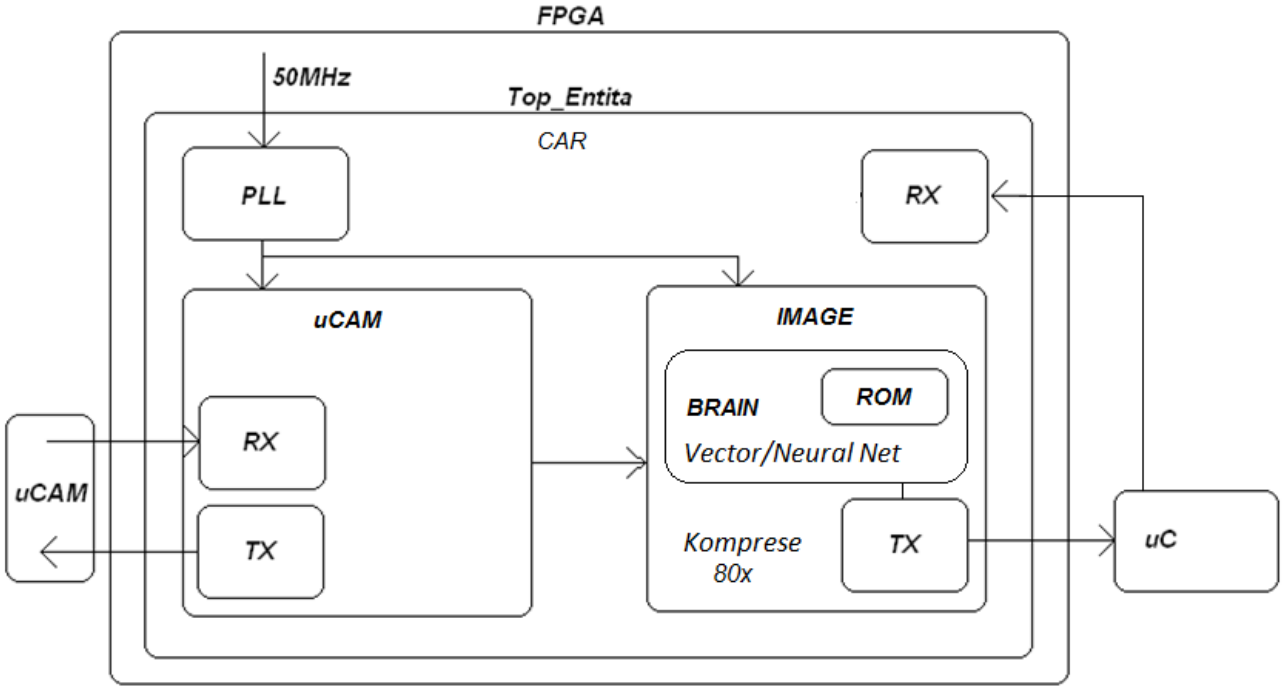
- 22,320 (LEs) logických elementů
- 594 Kb vlastní paměti
- 66 násobiček
- 4 fázové závěsy
- 153 vstupních/výstupních pinů

Vybrané vlastnosti desky

- 32MB SDRAM
- 2Kb I2C EEPROM
- 8 LED diod, 2 tlačítka, 4 přepínače
- AD převodník
- 50 MHz oscilátor
- Akcelerometr

3.2.2 Popis projektu

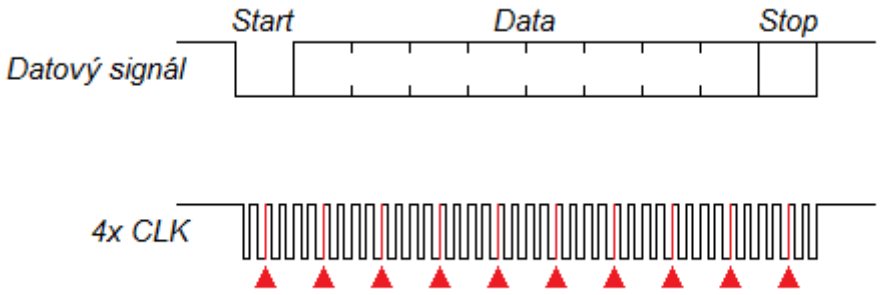
Jak již bylo napsáno v úvodu do této kapitoly, obvod FPGA se stará výhradně pro zpracování videa z kamerového modulu, výpočet řídících vektorů a komunikaci s procesorem. Celý tento systém je přehledně zobrazený na obrázku 3.5, v podobě jednotlivých komponent. Obrázek v tomto případě řekne daleko více než popis, proto se budu dále věnovat jednotlivým komponentám a popis funkčnosti nechám na obrázku 3.5. Vstupní/výstupní porty jednotlivých entit jsou uvedeny v příloze spolu se zdrojovými kódy.



3.2.2.1 UART

Prvním nepostradatelným krokem projektu byla nutnost universální komunikace jak s kamerovým modulem, tak s procesorem. Oba obvody dokáží komunikovat po jednoduchém sériovém rozhraní UART, kamera výhradně a procesor má samozřejmě na výběr z více možností. Proto jsem napsal univerzální komunikační řadiče tohoto rozhraní, jako dvě samostatné komponenty pro příjem a odesílání dat. Univerzálnost je v tomto případě chápána ve formě rychlosti, ostatní parametry přenosu dat a synchronizace jako start bit, jeden stop bit, žádná parita jsem použil dle svého uvážení a podle mého názoru i nejvhodnější konfigurace.

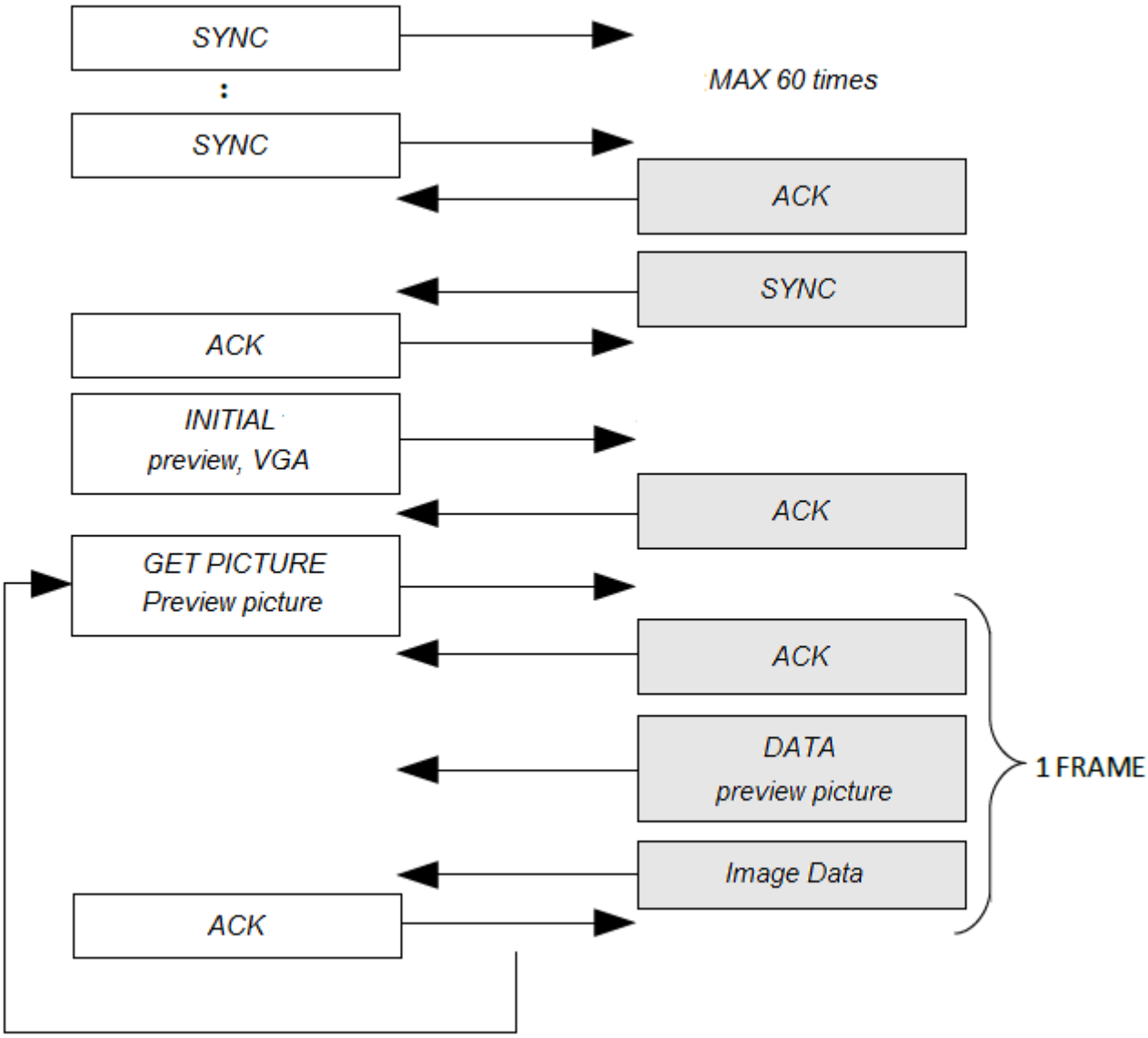
Rychlost odesílání dat komponenty TX je přímo odvozena od vstupních hodin entity. Pro příjem dat v komponentě RX je třeba použít rychlejšího hodinového signálu pro detekci sestupné hrany start bitu a od ní odvozeného vzorkování informačních bitů. Z mé zkušenosti a funkčnosti projektu vyplývá, že plně postačí 4x rychlejší hodinový signál, který spustí čítač při detekci start bitu a vzorkování na každou druhou náběžnou hranu signálu. Lepší představu o funkčnosti podá obrázek 3.6, vzorkování probíhá při červeně vyznačené náběžné hraně a je spolehlivé i při malé chybě fázového závěsu generující signál, jelikož se při každém start bitu resynchronizuje.



Obrázek 3.7: Vzorkování komponenty RX

3.2.2.2 uCAM

Komponenta uCAM poskytuje obsluhu kameře, pro získání video snímků obrazu a sériové přeposílání pouze informačních bitů do dalšího bloku projektu k dalšímu zpracování. Hlavní část entity využívající komunikační komponentu TX protokolu UART je napsaná v bloku Case, jelikož je přesně popsaná posloupnost událostí, které kamera vyžaduje pro synchronizaci, nastavení a vyžádání dat. Druhý proces komponenty se stará o příjem informace závisle na postupu prvního procesu. Rozhoduje o platnosti dat, řídí čekání vysílacího procesu na odpověď kamery a přeposílá informační bity obrazu do další komponenty. Většinu práce této komponenty popisuje obrázek 3.7.



Obrázek 3.8: Obsluha kamery
zdroj: převzato z Datasheetu [2]

3.2.2.3 IMAGE & BRAIN

Další nepostradatelnou entitou je komponenta IMAGE pro samotné zpracování obrazu, která provádí kompresi obrazu popsanou v jedné z předchozích kapitol a výpočet řídících vektorů pomocí komponenty BRAIN. IMAGE provádí výpočet komprese takzvaně za pochodu, tedy při sériovém příjmu obrazu a její velmi dobrou vlastností je, že při přijetí posledního bitu obrazu je dokončen také výpočet komprese, tudíž není ztrácen žádný čas. Při vytváření projektu jsem kladl důraz právě na tuto vlastnost, neboť jsem vyžadoval co nejmenší výpočetní čas, jelikož každá zbytečná prodleva ve výsledku snižuje maximální dostupnou rychlost auta.

Po ukončení výpočtu komprese obrazu je výsledná 60 bitová matice propůjčena komponentě BRAIN, která vypočítá dva 8 bitové řídící vektory. První vektor obsahuje informace pro nastavení polohy serva, tedy zatáčení automobilu a druhý vektor je určen pro nastavení rychlosti otáčení motoru, tedy spíše pro přivedení příslušného napětí na motor pomocí H-můstku. Po ukončení jejich výpočtu jsou oba přeposlány do procesoru, který si upraví PWM generátory.

Komponenta BRAIN využívá automaticky generovanou ROM paměť pomocí MegaVizard Plug-in Manageru, inicializovanou ze souboru. Paměť o velikosti 8192x17 bitových slov obsahuje informace o vahách neuronové sítě, kterou popíši v některé z nadcházejících kapitol stejně jako vektorové řízení (kapitola 4). Mezi těmito metodami výpočtu řídících vektorů je možné přepínat pomocí signálu vyslaného z procesoru, z uživatelského hlediska pak stiskem tlačítka na dotykovém LCD panelu, nebo pomocí rozhraní bluetooth a počítače, či jiného zařízení s terminálovou aplikací.

Důležitým parametrem je doba výpočtu predikce neuronové sítě z právě přijaté matice. Výpočet vektorového řízení se mi podařilo zajistit na jeden hodinový impuls o rychlosti 50MHz, je tedy takřka okamžitý v závislosti na 115,2kHz hodinách ostatních komponent. Výpočet predikce neuronové sítě už nebyl tak jednoduchý, ale ve výsledku jsem ho dokázal stáhnout na 48us, při výpočetní frekvenci 50MHz, což není ani doba kterou musí algoritmus čekat před vyžádáním nového obrazového snímku z kamery. V praxi to znamená, že řídící vektory spolu s obrazovou maticí jsou vypočítány a poslány ke zpracování do procesoru dříve než je obvodem FPGA žádán další obraz z kamery. Co se tedy týče komunikace obvodu FPGA s kamerou a následné problematiky zpracování obrazu, bylo by možné data z kamery žádat až na maximální možné přenosové rychlosti 1,2Mb/s, kterou podporuje kamera. Za těchto okolností by bylo možné rapidně zvýšit maximální možnou rychlost auta, která je omezena právě počtem získávaných snímků z kamery za sekundu.

O toto zvýšení komunikační rychlosti jsem se pokoušel a algoritmy v obvodu FPGA jsou tomu přizpůsobeny tak, že se dá po potřebném nastavení kamery při autodetekční rychlosti 115,2kHz přepnout na 1,2MHz a komunikovat s kamerou na této rychlosti. Realizace tuto možnost bohužel vyvrátila, jelikož je kamera silně rušena motorem a servem, a i při odstínění komunikačního páru vodičů je provozování přenosu na této rychlosti takřka nemožné. Za těchto podmínek jsem z kamery přijal průměrně 2 až 3 obrazy, poté došlo k porušení komunikačního protokolu a kamera vyžadovala reset, který ovšem trvá až několik jednotek sekund. Kamera vykazovala tuto chybu i při prostém připojení k obvodu FTDI a komunikace s PC bez rušení motorem či servem, s trochu menší průměrnou chybovostí, ponechal jsem tedy rychlost na 115,2kHz jelikož na této rychlosti, po odstínění komunikačního páru, vykazovala nejnižší chybovost.

3.2.2.4 Top entita CAR

Všechny výše popsané komponenty jsou obsaženy a řízeny z hlavní entity, která navíc obsahuje fázový závěs a komponentu RX komunikačního rozhraní UART. Fázový závěs generuje hodinový signál pro komunikační rozhraní, jak pro vysílání dat, tak 4 násobný hodinový signál potřebný pro příjem dat. Komponenta RX je určena výhradně pro příjem řídících signálů z procesoru, těmito signály jsou například přepínání mezi vektorovým řízením a neuronovou sítí, dále pro asynchronní reset obsažený ve všech komponentách.

Komunikace mezi jednotlivými funkčními bloky přímo implementovanými v top entitě probíhá typem „hand-shake“, jsou zde tedy obsaženy řídící signály, které přesně definují dané situace, a top entita nad nimi nemá žádnou kontrolu. Můžeme mít maximálně přehled o stavu jednotlivých signálů, čož je prospěšné při ladění a vývoji algoritmů.

Jazyk VHDL je samozřejmě kompatibilní se všemi obvody FPGA a po upravení vstupních/výstupních pinů a zvolení správného syntetizátoru je možné algoritmy nahrát do jakéhokoliv logického obvodu. S jediným omezením a to, že obvod musí, běžet minimálně na 50MHz, což v dnešní době splňuje každé FPGA.

3.3 Procesor

K ovládání periferií, ostatních obvodů i obvodu FPGA jsem využil vývojovou desku Stellaris LM3S1968 Evaluation Board od firmy Texas Instruments. Na desce je umístěn 32bitový mikroprocesor LMS31968 s jádrem ARM Cortex-M3, který operuje s maximální frekvencí jádra až 50MHz, na které ho provozuji. Obvod vyhovuje jak z hlediska výpočetního výkonu, počtem vstupních/výstupních pinů, počtem komunikačních rozhraní, počtem PWM generátorů, velikostí paměti, tak z hlediska energetické náročnosti a power managementu samotného procesoru. Samotný obvod jsem volil na základě jeho parametrů a možností, zato výrobce jsem volil na základě předchozích zkušeností a ze zkušeností z jednoho podporovaného programovacího rozhraní IAR Embedded Workbench.

Vybrané vlastnosti a parametry zvoleného mikroprocesoru:

- Maximální frekvence jádra: 50MHz
- Výkon: 1.25 DMIPS/MHz
- 55 I/O pinů s 3.3V TTL logikou
- 5V napájení s regulátorem
- 256 KB flash a 64 KB SRAM paměti
- 3 komunikační UART rozhraní
- 3 dvouportové PWM generátory
- 10bitový AD převodník

Vybrané vlastnosti vývojové desky:

- Li-Ion baterie pro hibernační modul
- USB port pro komunikaci a napájení
- OLED grafický displej 128x96
- 66 I/O pinů
- magnetický reproduktor

Jak je vidět z úvodního obrázku 3.1, k mikroprocesoru je připojeno celkem 5 periferních obvodů a obvod FPGA, které musí řídit. Nejdůležitější částí je komunikace přes rozhraní UART s obvodem FPGA, který mu posílá řídící vektory pro nastavení PWM generátorů. Používám 2 nezávislé PWM generátory pro nastavení polohy serva a řízení otáček motoru pomocí H-můstku, oba ve stejném Count-Down módu ale na různých frekvencích.

Další, již méně důležitou částí pro samotný závod, avšak nepostradatelnou při vývoji řídících algoritmů, je vykreslování informací na LCD panel, obsluha dotykového senzoru LCD panelu, komunikace s PC přes bluetooth modul a komunikace s modulem RC soupravy. Ke každé části jsem napsal třídu pro její obsluhu a hlavičkové soubory, k použitým třídám následuje stručný popis funkce a kompletně jsou přiloženy na CD.

Napsané algoritmy je možné po drobných úpravách nastavených I/O bran a registrů možno použít na jakékoliv řadě obvodů Stellaris, která má dostatečný počet potřebných periferních obvodů. Jsou to 2 nezávislé PWM generátory, 2 UART komunikační porty a samozřejmě dostatečný počet I/O pinů pro modul RC soupravy a LCD panelu.

Popis jednotlivých tříd:

ARM_CAR Není sice třída, ale je to neméně důležitá „top entita“ obsahující Main() samotného programu. Nekonečná smyčka rozhoduje podle nastavení globálních proměnných prioritního systému o tom, který ze tří možných subsystémů bude auto řídit (umělá inteligence auta, uživatel pomocí PC nebo RC soupravy). Dále pak sleduje, jak dlouho uživatel pracuje s dotykovým senzorem displeje, jež využívám pro přepínání mezi řízením neuronovou sítí a vektorovým řízením.

Ostatní subsystémy jsou obslouženy pomocí interruptových rutin. Handler UARTu_1 je určený pro příjem dat z obvodu FPGA, tedy řídících vektorů a obrazové matice. Handler UARTu_2 přijímá data z bluetooth modulu, kde je možnost přepnutí typu umělé inteligence auta, nastavení prioritního systému ovládání, možnost vyžádání tréninkových dat pro

neuronovou síť, nastavení rychlosti auta a samozřejmě možnost ovládání auta z PC na klávesách W,S,A,D. Poslední interruptový handler zastává funkci jakého si WatchDogu pro kamerový systém FPGA, jelikož kamera je poměrně hodně náchylná na rušení a může se stát, že dojde k narušení komunikačního protokolu a potom je třeba kameru restartovat. Jde tedy o čítač, který je nastaven na určitou hodnotu po každém přijetí dat z obvodu FPGA a pokud by došlo k chybě, žádná data by nepřišla a čítač se dostal na nulovou hodnotu, procesor automaticky vyšle restart kamery a vypíše o tom informaci na displej.

UART_V1 Třída obsluhující inicializaci a obousměrný tok dat všech použitých komunikačních rozhraní UART. Třída je napsaná pouze pomocí registrové sady a je tedy přehledně vidět, co který registr dělá a co je třeba nastavit pro úspěšnou komunikaci.

TFT Je třída pro inicializaci, vykreslování základních textových a grafických prvků na LCD displej. Aktuální nastavení umožňuje použití 16bitových barev, pro vykreslení pozadí, nebo další grafiky. Třída obsahuje funkce pro vykreslení textu ve formě řetězce nebo jednotlivých znaků, celých či reálných čísel s přesností na 3 desetinná místa, jednoduchých grafických prvků jako horizontální či vertikální čára, vyplnění obdélníku zvolenou barvou a funkce pro grafické znázornění přijatých obrazových informací z kamery.

Dále jsem do třídy umístil inicializaci a funkce pro obsluhu odporového dotykového senzoru umístěného přes displej, komunikujícího pomocí rozhraní SPI.

RC V této třídě je umístěna inicializace a funkce pro příjem a vyhodnocení signálů z modulu RC soupravy.

PWM_V1 PWM signál je použit pro ovládání polohy serva a kontrolu rychlosti otáčení motoru pomocí zvoleného H-můstku. Uvedená třída proto obsahuje inicializaci dvou nezávislých PWM generátorů a funkce pro jejich ovládání. K nastavení generátorů jsem opět použil přímý zápis hodnot do příslušných registrů bez připravených funkcí od výrobce, je tedy přehledně vidět, co který registr nastavuje a jaká je jeho funkce.

NN Je poslední třídou primárně určenou pro kontrolu predikce neuronové sítě použité v obvodu FPGA. Třída tedy obsahuje funkci pro výpočet predikční hodnoty stejně jako obvod FPGA a využil jsem jí hlavně při vývoji těchto algoritmů.

3.4 Servo

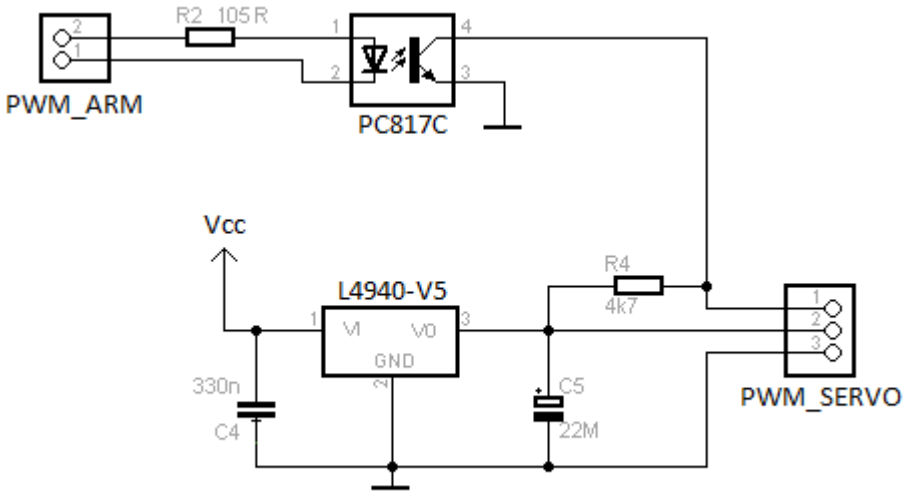
K natáčení předních kol jsem zvolil miniaturní modelářské servo wk-7.6-6, které svými parametry zcela vyhovuje potřebám modelu. Řízení polohy většiny modelářských serv obstarává pulzní šířková modulace (PWM) s frekvencí 50Hz a pracovním rozsahem řídícího impulsu 1-2ms, se středovou polohou 1500us a pásmem necitlivosti 8us.

3.4.1 HW zpracování

Parametry serva: wk-7.6-6

Pracovní napětí:	4.8 V – 6 V
Proudový odběr:	10 – 800 mA
Rychlost otáčení:	0.12s/60°
Síla:	1.4 kg/cm
Hmotnost:	8.4g
Rozměry:	22.5x11.5x24 mm

Z důvodů vyššího proudového odběru serva a rozsahu pracovního napětí, bylo nutností vyrobit obvod pro napájení s galvanicky oddělenou řídící a výkonovou částí. Zapojení tohoto obvodu je na obrázku 3.8.



Obrázek 3.9: Obvod pro kontrolu serva

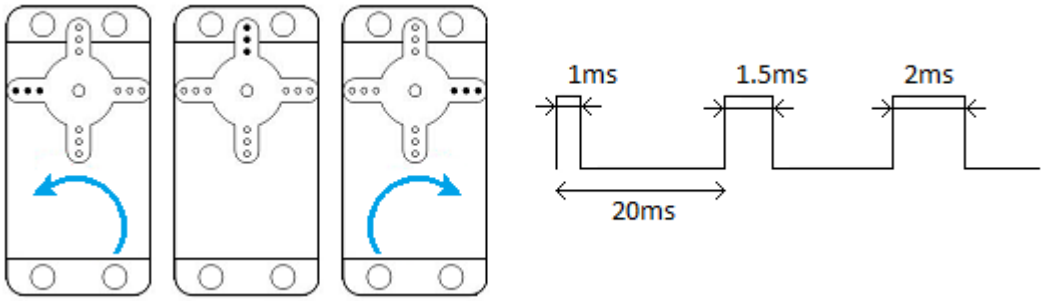
Z výše uvedeného rozsahu pracovního napětí se jeví více než vhodné použití lineárního stabilizátoru napětí na 5V, opatřeného kondenzátory ve funkci filtračního C4 a akumulačního C5, pro napájení samotného serva. Pro tuto stabilizaci používám LDO (Low Drop Output) stabilizátor typu L4940-V5, který má úbytek napětí maximálně 500mV a pro napájení z baterií je tedy vhodnější než běžně používaná 7805.

Dále bylo záhodno galvanicky oddělit řídící a výkonovou část, což je úkol optočlenu PC817C. Odpor R2 upraví 3.3V TTL logiku mikrokontroleru na pracovní hodnoty optočlenu Up = 1.2 V a Ip = 20mA. Odpor R4 obstarává 1mA pulsy řídící polohu serva, je třeba brát na zřetel, že servo je řízeno 1-2ms pulsy logické jedničky, ale na vstup optočlenu je nutné

přivádět signál negovaný, jelikož logická 1 na vstupu optočlenu rozsvítí diodu, ta otevře tranzistor a výstupní signál je uzemněn, což znamená logickou 0 na vstupu serva.

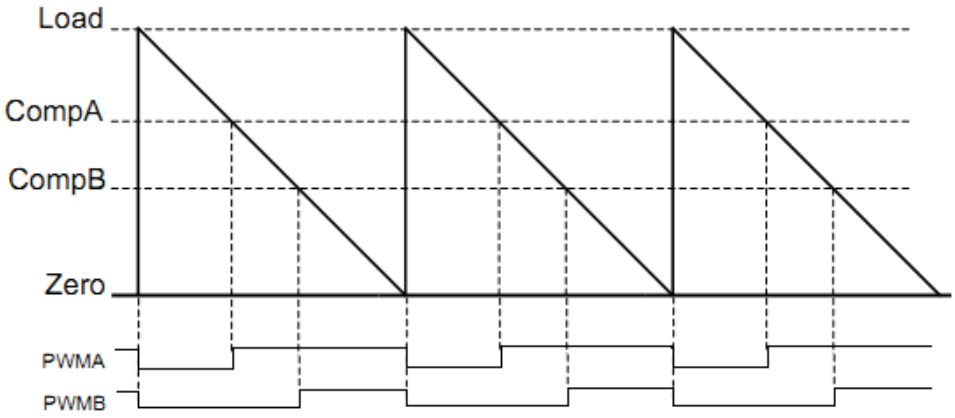
3.4.2 SW zpracování

Jak již bylo zmíněno modelářské servo pracuje s frekvencí 50Hz a šířkou pulsu 1-2ms s centrální polohou 1.5ms, ovládání serva je tedy jednoduše znázorněno na obrázku 3.9. Generování tohoto signálu je pomocí mikrokontroleru řady Stellaris s jádrem Cortex-M3 poměrně komplikovaná záležitost pro svoji velkou variabilitu.



Obrázek 3.10: Ovládání serva PWM generátorem

Hardwarové PWM v mikrokontroleru obsahuje celkem 48 registrů, 2 nezávislé komparátory a jeden 16b čítač, ve výsledku podporuje několik funkčních módů běhu PWM generátoru. Tyto módy nebudu popisovat, vše se dá dohledat v Datasheetu od použitého mikrokontroleru [3]. Popíši jen metodu, kterou využívám pro generování těchto řídících impulsů a impulsů pro H-můstek. Řízení je jednoduše vyobrazeno na obrázku 3.10, kde při dosažení hodnoty komparátoru napětí generované pily o frekvenci f, je nastavena logická 1 a při dosažení nulové hodnoty generované pily je tato jednička shozena na logickou 0, což je ideální řešení pro potřeby negované logiky užité v řízení serva.

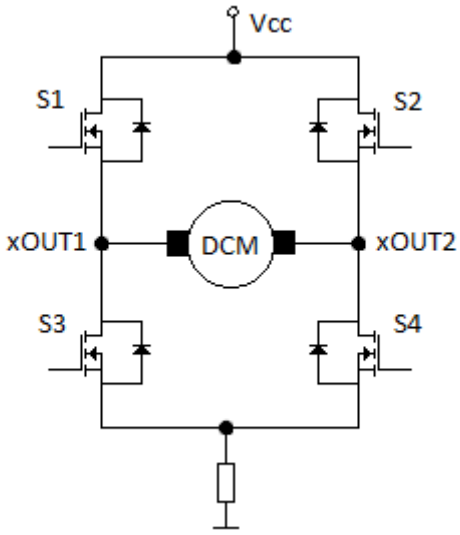


Obrázek 3.11: Generování PWM signálu pro řízení
zdroj: Datasheet [3]

3.5 H-můstek

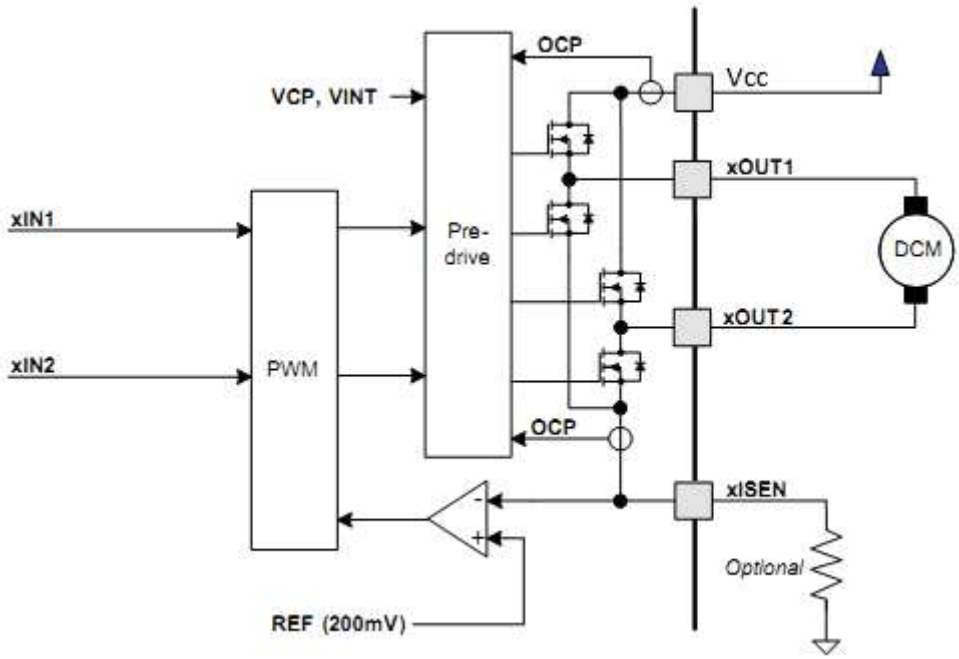
Každý robot a tím spíše auto potřebuje regulaci otáček motoru, kterou v mém případě zajistí zvolený H-můstek, řízený PWM modulovaným signálem z mikrokontroleru. Můstek umožňuje dle přivedených signálů na vstupy omezení příkonu motoru při minimálních ztrátách na samotném obvodu a navíc dopředný a zpětný chod motoru. Nepostradatelnou součástí můstku je také omezení rozběhového proudu pro stejnosměrné motory, pomocí senzoru proudu a oddělení 3.3V TTL logiky od výkonové části řízení.

Funkce H-můstku je založena na velmi jednoduchém principu, který je popsán pomocí obrázku 3.11. Samotný můstek představují celkem 4 polem řízené tranzistory spínané dle potřeby chodu motoru. Při sepnutí spínačů S1 a S4 se bude motor točit jedním směrem a při sepnutí spínačů S2 a S3 směrem druhým, popřípadě brzdit jinou volbou sepnutých spínačů.



Obrázek 3.12: H-můstek
zdroj: Datasheet [4]

Blokové schéma jednoduchého H-můstku je na obrázku 3.12, kde je vidět vstupy xIN1 a xIN2, na které je možné přivést PWM signál a tím řídit výkon motoru. Dále je na blokovém schéma výstup xISEN s volitelným odporem pro omezení výstupního proudu na svorkách xOUT. Více informací je možné najít v datasheetu [4].



Obrázek 3.13: Blokové schéma H-můstku
zdroj: Datasheet [4]

Pro úplnost uvádím parametry vybraného H-můstku. Výběr závisel hlavně na rozsahu výstupního napětí, trvalé proudové zatížitelnosti a kompatibility s 3.3V TTL logikou PWM signálů. Rozsah výstupního napětí je důležité porovnat s možnostmi použitého motoru v autíčku a případně ohlídat nepřekročení maximálního povoleného napětí. V mém případě tento problém nenastal, jelikož většina vhodných modelářských motorů pracuje v mezích od 4,5 do 8,6V a i při plném sepnutí můstku použité baterie nemůžou dodat vyšší napětí než přibližně 8,2V při plném nabití baterií.

- Výrobce: Texas Instruments
- Označení: DRV8833
- Spojité operace: 2.7 Vout – 10.8 Vout
- Špičkový proud: 4 A (trvalý 3 A) při paralelním zapojení
- Frekvence PWM: 50 kHz
- Rds(on): 200 mOhm
- 3.3 V TTL kompatibilní logika
- Proudová a teplotní ochrana

3.6 LCD panel s dotykovým senzorem

Přímo na kapotu auta jsem vyřízl díru pro LCD panel o velikosti 3.2 palce s 320x400 obrazovými body. Tento displej je vynikající ladící nástroj pro vývoj řídícího algoritmu a dotyková část je výborná k ovládání celého systému řízení auta uživatelem. Tedy přepínání mezi jednotlivými algoritmy řízení, v mém případě mezi vektorovým řízením a neuronovou sítí, tak nastavování různých parametrů, zobrazení statusů všech periferií a dokalibrování auta na aktuální podmínky vozovky.

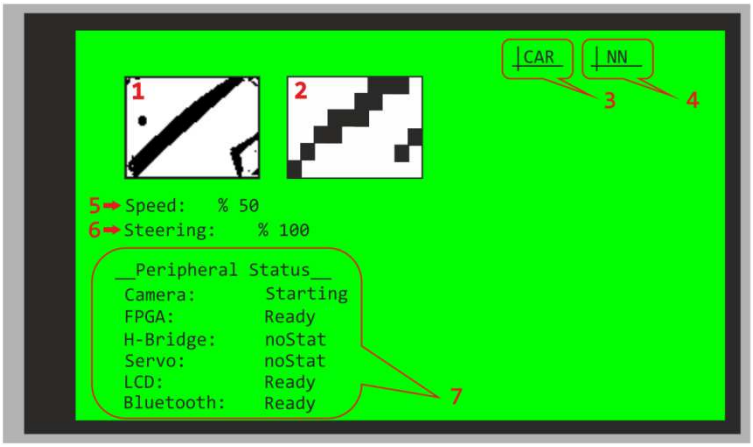
Zvolil jsem modul od ITead Studia s řadičem ITDB02-3.2, jelikož tato varianta nabízí relativně velký přehledný displej za velmi přijatelnou cenu, se zabudovanou SPI komunikací pro dotykový senzor a navíc slotem pro SD kartu, který ale nevyužívám. Řadič displeje komunikuje s okolím volitelně pomocí paralelního 8 nebo 16 bitového datového rozhraní. Navíc využívá napájecí napětí pouze 3.3 VDC a tedy i komunikuje s touto TTL logikou, je tedy ideální pro připojení ke zvolenému typu mikroprocesoru a může být i napájen z jeho obvodové desky.

Obrázek 3.14: Dotykový LCD panel
Zdroje obrázků [5]

Displej podporuje 16bitovou barevnou škálu, tedy 65 tisíc barev a komunikuje po 16bitovém datovém rozhraní s řídícími signály „Read“ a „Write“ představující čtecí a zápisový impuls. Napsat řídící třídu pro ovládání tedy nepředstavovalo problém a s datasheetem nebylo zprovoznění displeje nijak náročné. Napsal jsem několik funkcí pro vykreslení různých grafických prvků a několik pro výpis různých datových typů, hlavičkový soubor s názvy funkcí je uveden v příloze spolu se zdrojovými kódy. Do stejné třídy jsem umístil funkce pro inicializaci a ovládání dotykového senzoru a jeho řídícího čipu ADS7843 komunikujícího přes rozhraní SPI.

Na LCD panel vykresluji hned několik grafických a textových ladících informací, popisujících aktuální stav a činnost automobilu. Při mechanickém nastavování kamery na vhodnou pozici jsem využil grafické okno **1**, které zobrazuje kamerou viditelnou oblast v původním rozlišení, tedy 60x80 obrazových bodů s 1bitovou barevnou hloubkou, viz kapitola *Zpracování obrazu* obrázek 3.3c. Kameru jsem nastavil tak, aby zabírala oblast popsanou na obrázku 3.4.

Druhé grafické okno **2**, na které zobrazuji komprimovanou obrazovou matici o velikosti 6x10 bodů, jsem využil při nastavování vah submatic, viz obrázek 3.3e. Váhy jsem nastavoval intuitivně a pokusně, aby získaná matice představovala ve většině případů rozumný výsledek, ze kterého půjde spočítat hodnota jak vektorového řízení, tak hodnota predikce neuronové sítě.



Obrázek 3.15: Popis grafiky LCD panelu

V načrtnutém textovém boxu **3** zobrazuje prioritní systém přebírání kontroly nad autem, který ze tří možných systémů zrovna plně kontroluje auto (umělá inteligence -> CAR, PC pomocí bluetooth modulu -> PC, nebo RC souprava -> RC). Hned vedle v podobném boxu **4** vypisuji informaci o tom, jaký druh umělé inteligence se aktuálně stará o výpočet řídících vektorů (vektorové řízení -> VCT, neuronová síť -> NN).

Pod grafickými okny s obrazovými informacemi, zobrazuji hodnoty přijatých vypočtených řídících vektorů v procentech. **5** nastavení rychlosti, **6** nastavení natočení kol.

Další textové informace pod hodnotami řídících vektorů „Peripherals status“ **7** zobrazují aktuální statusy připojených periferií. Periferie, jako LCD, H-můstek a Servo jsou zde pouze informativně, jelikož z hlediska procesoru nevykazují žádnou zpětnou vazbu. LCD dokáže zobrazit svůj status, pouze jen jako READY, za podmínky že samo běží. H-můstek by pro zpětnou vazbu s procesorem vyžadoval senzor rychlosti a funkce serva by se dala kontrolovat matematickým výpočtem z informací kamery, za podmínky pohybu auta. Status modulu Bluetooth se dá snadno hlídat, jelikož obsahuje samostatný status pin, takže se dá jednoduše dodělat. Ve výsledku hlídám zatím pouze status kamery timerem použitým v procesoru, zda přichází data (status ->READY) nebo žádné data nepřichází a je vyžadován restart kamery, potom vyšle procesor automatický restart (status -> RESTART).

3.7 Bluetooth

Pohyblivé auto vyžadovalo použití některé bezdrátové technologie pro komfortní ovládání a ladění řídících algoritmů. Vzhledem k tomu, že jsem chtěl auto připojit k počítači bez další zásuvné desky nebo jiného obvodu, zúžil se výběr bezdrátových technologií na 2 a to Bluetooth nebo Wi-fi.

Wi-fi bývá více náročná na obsluhu jak v počítači, tak z hlediska samotného hardwaru, zato Bluetooth jsou v zásadě jednodušší při obsluze na obou stranách ale také pomalejší a mívají větší latenci. Z hlediska rychlosti odezvy, nutnosti rychle řídit auto z počítače a potřeby přenášet velké množství dat ve formě obrazového signálu pro vzdálené ladění by se zdálo, že bude potřeba využít technologii Wi-fi. Podařilo se mi ale najít jednoduchý a relativně levný Bluetooth modul Bluetooth Mate od firmy Sparkfun, který slouží primárně jako bezdrátová náhrada obvodu FTDI, tedy jako náhrada sériového kabelu.

3.7.1 Bluetooth - HW

Parametry a vlastnosti modulu Bluetooth Mate jsou velmi dobré a po navázaní komunikace mezi počítačem a modulem, která trvá obvykle několik sekund je přenos informací velmi rychlý, asi stejně, jakoby byl použit přímo sériový kabel a data z počítače přicházejí bez člověkem zaznamenatelného zpoždění. Se svojí přenosovou rychlostí na straně UART portu, maximálně 115k2 b/s, se kterou mi komunikují i všechny ostatní obvody je více než vhodný pro začlenění do systému a auto reaguje na zmačknutí klávesy na počítači takřka okamžitě.

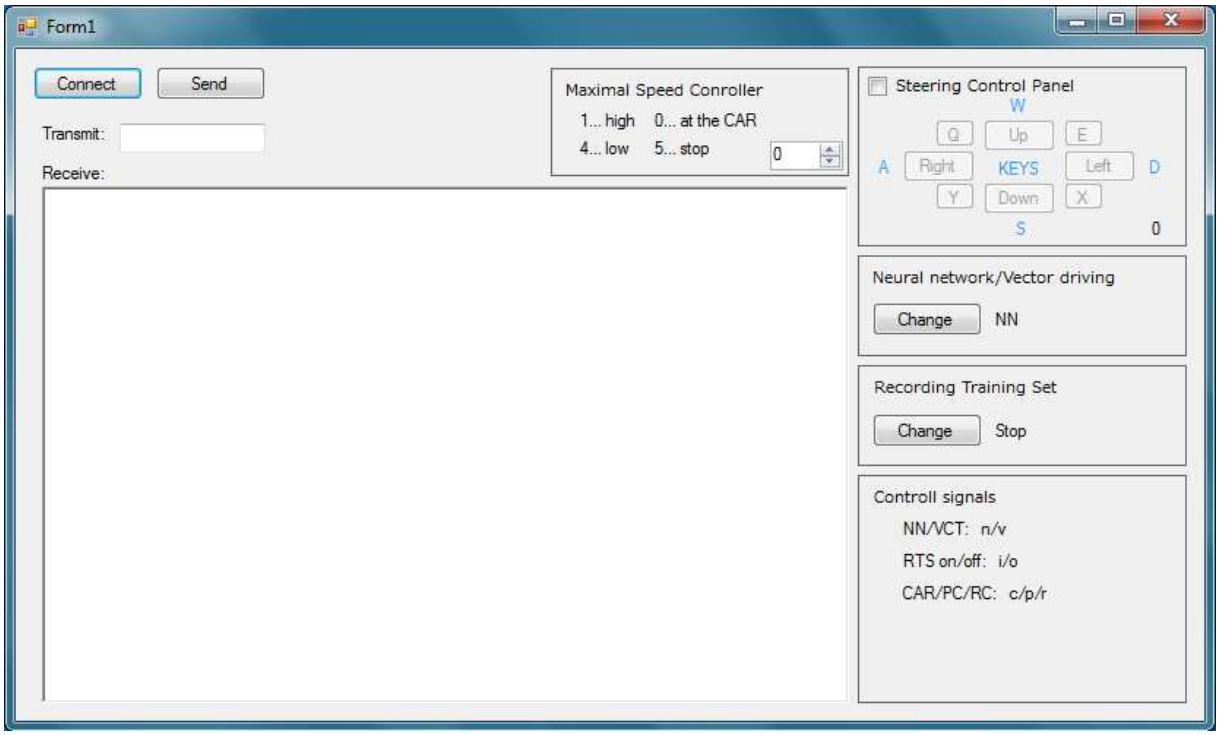
Vybrané parametry Bluetooth modulu:

- Rychlost na straně UART portu: 9600 až 115k2 b/s
- Rychlost bezdrátového přenosu: cca 1 Mb/s
- Frekvence: 2.4 až 2.524 GHz
- Ověřená maximální vzdálenost: 100 metrů
- Operační napětí: 3-6 VDC - vhodné pro 3.3 V TTL logiky
- Odběr: 25 mA

Obrázek 3.16: Modul Bluetooth Mate
Zdroje obrázků [6]

3.7.2 Bluetooth – SW

Řízení auta z počítače je možné provádět z jakékoli terminálové aplikace, jelikož řídící signály nejsou nijak složité a výhradně využívám jednoznakové zprávy, lze si je snadno zapamatovat a používat, nebo použít speciálního programu napsaného přímo pro tuto činnost. Já sem si v C# napsal aplikaci pro komfortní ovládání auta, kterou jsem nejvíce využil při získávání tréninkové množiny dat pro neuronovou síť.



Obrázek 3.17: Aplikace pro ovládání auta z PC

Aplikace obsahuje celkem 4 podpanely, na kterých je možné provádět různé zásahy do řídicího systému auta. Prvním je úprava maximální rychlosti auta, který dovoluje ponechat rychlost na samotném autu, vybrat mezi 4 maximálními rychlostmi, nebo zastavit motor auta.

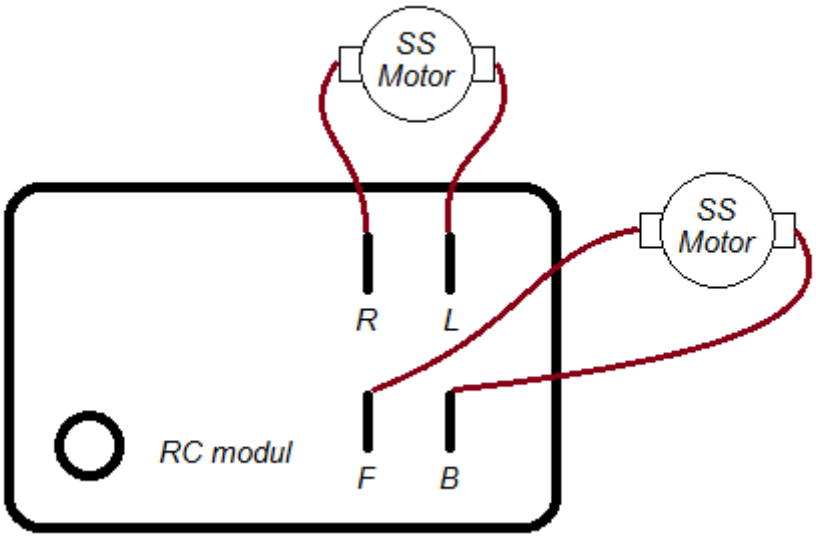
Druhý panel „Steering Control Panel“ dovoluje převzít kontrolu nad řízením auta, tedy vypnout umělou inteligenci auta a nechat auto řídit uživatelem pomocí kláves W,S,A,D nebo tlačítek na panelu. Tento panel jsem využil převážně na záchranu auta před nárazem do překážky při vývoji řídících algoritmů a při získání tréninkové množiny dat pro neuronovou síť, kdy jsem ovládal auto na klávesách a nechával si posílat obrázek po kompresi do TextBoxu na hlavním panelu.

Další dva panely slouží už jen na přepínání mezi řízením neuronovou sítí, nebo vektorovým řízením a zapínáním/vypínáním zasílání dat o činnosti auta, tedy i kompresovaného obrazu jako tréninkové množiny.

3.8 Modul RC soupravy

Původní RC soupravu z autíčka jsem lehce upravil a připojil k mikroprocesoru, aby byl kdykoliv možný uživatelský zásah a tak možnost zachránit autíčko před srážkou s cizím předmětem, nebo ho řídit při získávání tréninkové množiny dat pro neuronovou síť.

Jelikož původní ovládání auta nebylo nijak komplikované, obsahovalo pouze příkazy naplno dopředu, naplno dozadu, plný rejd vlevo, plný rejd vpravo, čemuž odpovídají i 4 výstupní piny, které stáhly svůj signál na nulu při stisku příslušného tlačítka, nebyla úprava nijak složitá. Umístil jsem na desku pouze napěťové děliče ke každému pinu, abych snížil napěťovou úroveň z původních 7 až 8 VDC závislých na stavu vybití baterie, na cca 3.3 VDC připojitelných k mikroprocesoru. Na obrázku 3.16 je ilustrativně naznačena původní konstrukce modulu RC přijímače.



Obrázek 3.18: Ilustrace původního zapojení RC modulu

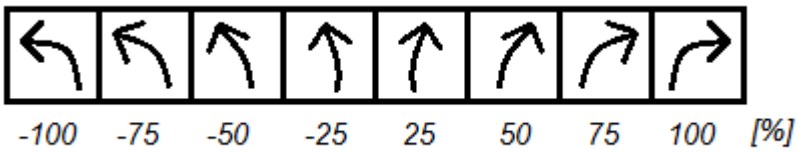
RC vysílačce jsem umožnil převzít kontrolu nad autem stiskem tlačítka dolu, jelikož tento směr je ve většině situací nepotřebný a auto s ním počítá jen velmi omezeně při naprosté ztrátě černé čáry. Po převzetí kontroly se vypne umělá inteligence auta a auto přijímá příkazy pouze od vysílačky, pokud nezakročí nadřazený signál z PC. Takovéto úplné převzetí kontroly bylo nutné udělat, aby vysílačka neřídila auto pouze při stisku tlačítek a při neutrální poloze tlačítek by se kontroly nad autem nezmocnila zase umělá inteligence. Předání kontroly nad autem umělé inteligenci se provede opětovným stiskem tlačítka dolu.

4 Umělá inteligence autíčka

Výpočet řídících vektorů, podle kterých je nastavováno řízení jsem řešil dvěma způsoby. První zdánlivě jednodušší variantou je takzvané „vektorové řízení“, kdy se kompresovaná 60 prvková matice obrazu naváží, to znamená přenásobí dalšími vektory a podle výsledku se vyhodnotí kam zatočit. Druhou variantou, kterou jsem volil je neuronová síť s dopředným šířením typu vícevrstvý perceptron s učením BackPropagation.

Z obvodu FPGA jsou do mikroprocesoru posílány 2 osmibitové řídící vektory, jeden pro nastavení zatáčení a druhý pro nastavení rychlosti otáčení motoru. S nastavením a velikostí citlivosti, kterou aktuálně používám a která je absolutně postačující by šli vektory zakódovat a posílat pouze v jednom znaku. Ponechal jsem je ale ve vektorech dvou pro možnost dalšího upravení, a poslání jednoho znaku po rozhraní UART navíc rychlostí 115k2b/s již nehraje přílišnou roli v možné rychlosti auta.

Pro větší názornost uvádím na obrázku 4.1, řídící vektor pro nastavení natočení kol, s výslednou polohou kol v procentech od podélné osy auta, předpokládáme-li, že 100% je maximální možná výchylka kol. S aktuální citlivostí může být 1 ve vektoru pouze na jednom místě, všechny ostatní prvky musí být nulové, pokud tomu tak nebude, kola se nastaví podle poslední vyhodnocené jedničky. Takovouto podobu řídících vektorů jsem zavedl hlavně z potřeby a nastavení multi-class klasifikace použité neuronové sítě, které bude věnována jedna z následujících podkapitol (4.2.2.3). Důležitým poznatkem, který je také vidět z obrázku je, že auto jede rovně pouze tehdy, jsou-li všechny prvky vektoru nulové.



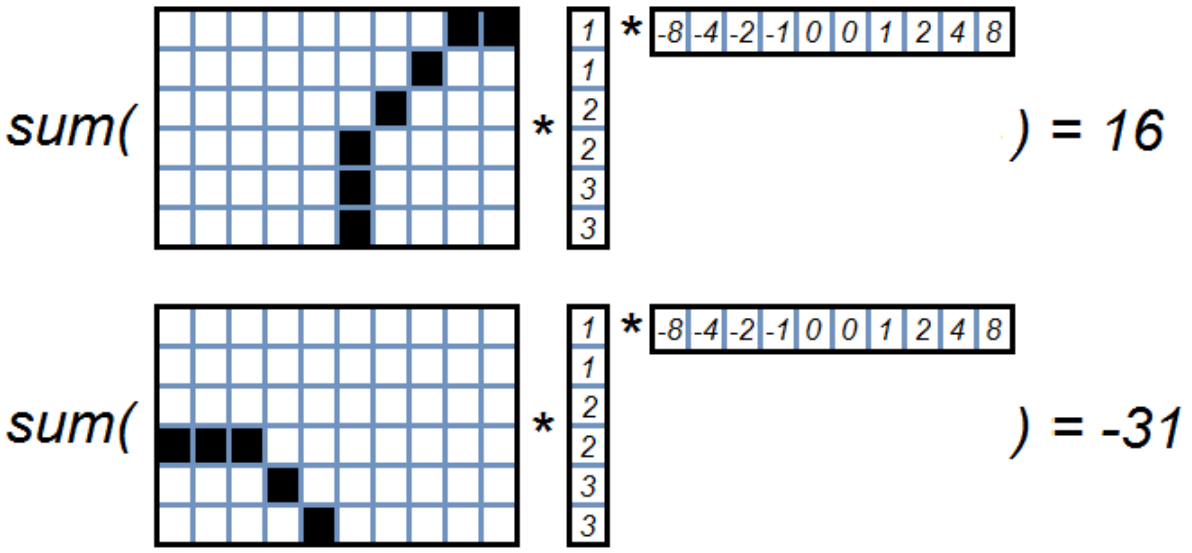
Obrázek 4.1: Řídící vektor pro natáčení kol

S druhým vektorem pro nastavení rychlosti vozidla je to velmi podobné. Jednička na bitu s největší vahou nastavuje nejvyšší možnou rychlost, jednička na bitu s druhou nejnižší vahou udává nejnižší možnou rychlost, které je vozidlo ještě schopné dosáhnout. Poslední jednička na bitu LSB je vyhrazena pro couvání, ztratí-li vozidlo kontakt s černou čárou a je potřeba couvnout pro její znovunalezení.

4.1 Vektorové řízení

Vektorové řízení je velmi jednoduchý součet všech prvků kompresované obrazové matice násobené dvěma vektory, které mají za úkol navážit matici s různou velikostí prvků v závislosti na poloze. Samotné natočení kol je tedy dáno dle velikosti výsledného součtu, větší absolutní hodnota součtu znamená větší natočení kol v závislosti na polaritě hodnoty. Záporné hodnoty znamenají zatáčku vlevo, naopak kladné potom pochopitelně vpravo.

Pro lepší představu je celý tento výpočet znázorněn na obrázku 4.2, kde první případ znamená lehčí natočení kol vpravo a druhý případ o něco větší natočení kol na druhou stranu.



Obrázek 4.2: Vektorové řízení

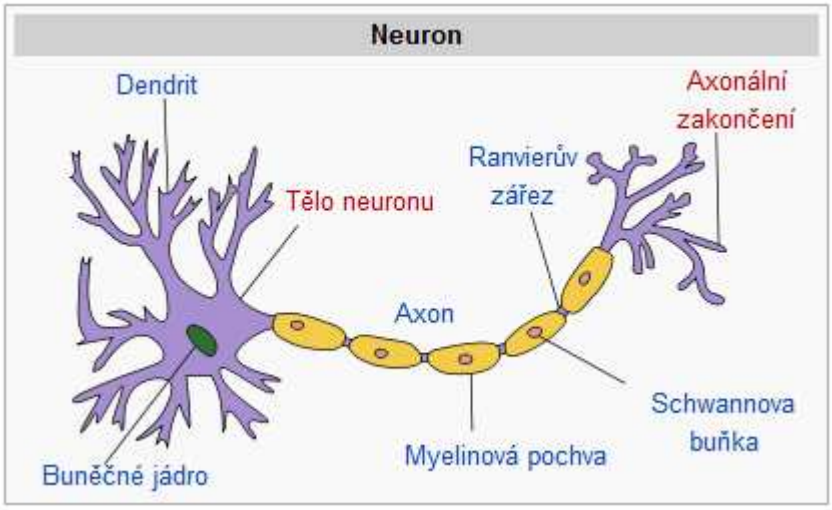
Toto řízení funguje výborně v ideálním případě polohy auta, to znamená, jede-li auto po černé čáře, nebo je-li podélně vedle rovné čáry. V případě, že se dostane do jiné polohy, například zatočí-li auto moc brzo, nastane případ, kdy je černá čára přes více než půl, ne-li celou horizontální osu, je pak matice navážena vektory špatně a vyhodnocení nemusí být korektní v závislosti na dalším vývoji trati. Pro tento nedostatek by chtělo řízení doladit, nebo jinak rozšířit. Já jsem se o tuto část nestaral, jelikož jsem ho používal pouze z počátku vývoje, jen abych odladil ostatní části algoritmu. Jako hlavní řídící program jsem totiž předpokládal neuronovou síť, které budou věnovány následující kapitoly.

4.2 Neuronová síť

4.2.1 Úvod

Původ a rozkvět neuronových sítí, jako algoritmu majícího za úkol napodobit mozkovou činnost člověka se datuje kolem 80 let a začátku 90, odkdy jejich popularita začala postupně uvadat. Nejspíše z důvodu nedostatečného výpočetního výkonu tehdejších výpočetních systémů a o analogovém řešení se v průmyslu nemá cenu ani bavit z důvodu velikosti a náročnosti sítí. V současné době představují neuronové sítě techniku v podobě algoritmů nasazovaných i v průmyslu, například pro rozpoznávání vad různých materiálů. A stále více se s nimi experimentuje v autonomních robotických systémech, viz tato práce.

Mozek člověka se stejně jako simulované neuronové sítě skládá z mnoha neuronů propojených synapsemi. Neurony vytvářejí dílčí sítě s mnoha vjemovými vstupy, optickými, akustickými, hmatovými a dalšími, z nichž každá je určena k jiné činnosti. Každý neuron má několik vstupů, jak z ostatních neuronů, tak z vjemového ústrojí člověka, které se nazývají Dendrity a pouze jeden výstup, vedoucí do dalších neuronů, nebo do svalového zakončení, nazývaný Axon. Propojení, neboli Synapse přenáší akční potenciál vzniklý aktivací neuronu, mezi Axonem daného neuronu a Dendritem neuronu druhého.

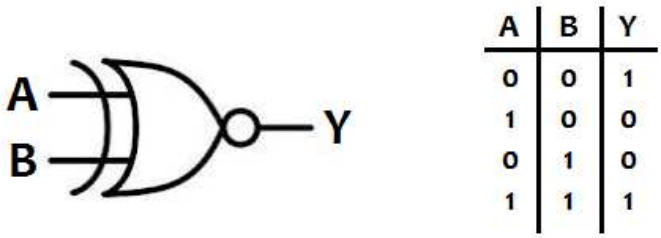


Obrázek 4.3: Model neuronu
Zdroje obrázků [7]

Model takového v elektronické podobě simulovaného neuronu, je velmi podobný jak funkcí, tak svojí strukturou. Obsahuje několik nezávislých vstupů X, jeden výstup Y a funkci pro výpočet akčního potenciálu přenášenou do dalších neuronů, stejně jako tomu je u skutečného neuronu v lidském mozku.

4.2.2 Reprezentace neuronové sítě

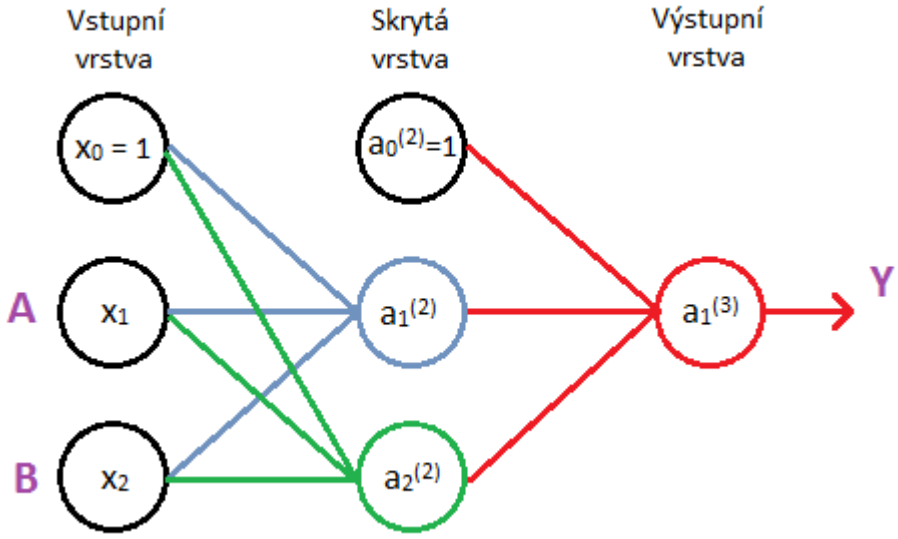
Pro názornost jsem se rozhodl vysvětlit funkci neuronové sítě i samotných neuronů na jednoduchém příkladu. A to simulaci v elektronice dobře známého logického hradla typu XNOR, tedy negovaného exkluzivního součtu. Hradlo i se svojí pravdivostní tabulkou je na obrázku 4.4.



A	B	Y
0	0	1
1	0	0
0	1	0
1	1	1

Obrázek 4.4: Hradlo XNOR se svojí pravdivostní tabulkou
Zdroje obrázků [8]

Jak je vidět z obrázku 4.4 hradlo má dva vstupy a jeden výstup, jinak tomu nabude ani v případě neuronové sítě. Zvolená reprezentace pro tuto úlohu bude mít samozřejmě také 2 vstupy A a B, neboli spíše dvouprvkový vstupní vektor x, což bude představovat vstupní vrstvu sítě. Jednu skrytou vrstvu rovněž s dvěma neurony, a jednu výstupní vrstvu v podobě jediného neuronu. Zvolená modelová reprezentace sítě by tedy mohla vypadat následovně, obrázek 4.5, kde jsou fialově označeny vstupy a výstup jako má logické hradlo.



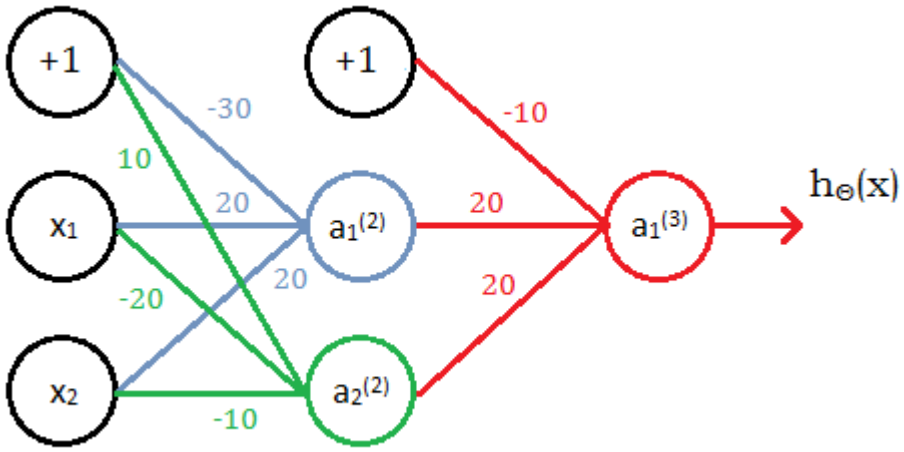
Obrázek 4.5: Neuronová síť pro XNOR

Přidané redundantní členy x_0 a a_0, zvané „bias units“, je velmi vhodné použít, jelikož vždy s určitou vahou napomáhají aktivaci neuronů v následující vrstvě. Musejí být tedy vždy bezvýhradně jednotkové, protože s nulovou hodnotou by to bylo to samé jako by vůbec neexistovali, tedy pouze bereme-li v úvahu možnost pouze dvou vstupních stavů 0 a 1. Jak je taky vidět na obrázku 4.5, k redundantnímu neuronu a_0 nevedou a nesmějí vést synapse ze vstupní vrstvy.

Samotná predikce neuronové sítě, tedy „odhad“ výstupu v závislosti na vstupním vektoru se skládá z několika jednoduchých výpočtů. Ačkoli je výpočet zcela jasný a vždy dostaneme na stejný vstupní vektor jednu jedinečnou odpověď, píši „odhad“, protože tato odpověď závisí na tom, jak dobře má síť nastavené jednotlivé váhy neuronů, neboli jak dobře jsme ji dokázali naučit řešit daný problém.

4.2.2.1 Predikce (Forward Propagation)

Výpočet predikce probíhá na jednoduchém principu postupu paralelních signálů skrze neuronovou síť. V první fázi se přivede vstupní vektor x na vstupní vrstvu neuronové sítě, kde je rozšířen o redundantní signál x_0, jeho velikost je tedy n+1. Dalším krokem je rozšíření signálů po „synaptických“ propojeních do všech neuronů následující skryté vrstvy. Neurony skryté vrstvy zpracují tyto signály s určitou vahou, která má za úkol zesílit, či zeslabit dané signály a tím upravovat míru aktivace neuronu. Neurony skryté vrstvy získají svůj akční potenciál prostým součtem vstupních signálů přenásobených jednotlivými vahami a přivedením tohoto součtu na přenosovou funkci. Takto aktivované neurony skryté vrstvy vytvářejí vstupní vrstvu pro poslední výstupní vrstvu sítě. Slouží tedy jako vstup pro třetí vrstvu a celý proces výpočtu akčního potenciálu se opakuje pro poslední výstupní vrstvu.



Obrázek 4.6: Neuronová síť pro XNOR se synaptickými vahami

Na obrázku 4.6 je znázorněna situace neuronové sítě s jednotlivými vahami u svých synaptických propojení a výstupní funkcí $h_\Theta(x)$. Váhy neuronů budu označovat jako velká theta Θ. Výstupní funkce neuronové sítě $h_\Theta(x)$, je rovna aktivaci výstupní vrstvy, tedy neuronu $a_1^{(3)}$. Spodní index skryté vrstvy s označením *a* je index prvku v dané vrstvě a horní index v závorce je označení vrstvy, tedy druhé.

Jednotlivé výpočty tedy probíhají podle následujících rovnic 4.1, kde první rovnice ukazuje výpočet aktivace neuronu $a_1^{(2)}$ v první skryté vrstvě, druhá rovnice potom pro druhý neuron a rovnice třetí je pouze obecné řešení pro znázorněnou síť.

$$a_1^{(2)} = g(\Theta_{10}^{(1)} * x_0 + \Theta_{11}^{(1)} * x_1 + \Theta_{12}^{(1)} * x_2)$$

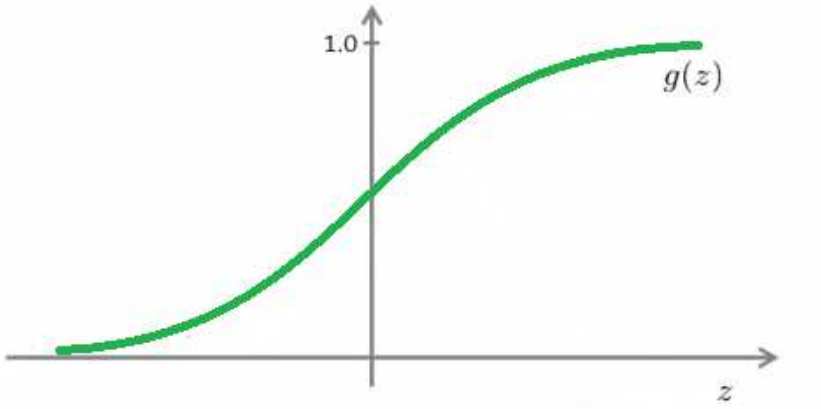
$$a_2^{(2)} = g(\Theta_{20}^{(1)} * x_0 + \Theta_{21}^{(1)} * x_1 + \Theta_{22}^{(1)} * x_2)$$

$$a_i^{(j+1)} = g(\Theta_{i0}^{(j)} * x_0 + \Theta_{i1}^{(j)} * x_1 + \Theta_{i2}^{(j)} * x_2) \qquad \text{(rovnice 4.1)}$$

Funkce g je takzvaná aktivační funkce, která z výsledné hodnoty získané součtem vstupního vektoru vynásobeného jednotlivými vahami vypočítá výsledný akční potenciál daného neuronu. Tato funkce má několik variant a její volba závisí jak na použitém typu neuronové sítě, tak na konkrétním účelu využití sítě. Funkce může být například hyperbolická, znaménková, Heavisideova, sigmoidální nebo jiná. Já využívám funkci sigmoidální s následujícím předpisem dle rovnice 4.2 a obrázku 4.7, kde hodnoty výstupní vrstvy nad 0.5 jsou brány jako logická 1 a naopak:

$$g(z) = \frac{1}{1+ e^{-z}}$$

(rovnice 4.2)



Obrázek 4.7: Sigmoidní funkce

Výpočet hodnoty výstupní funkce $h_\Theta(x)$ je zřejmý, ale pro úplnost jej uvádím také v podobě následující rovnice 4.3. Jak bylo již řečeno, predikce neuronové sítě je rovna aktivačnímu potenciálu výstupní vrstvy.

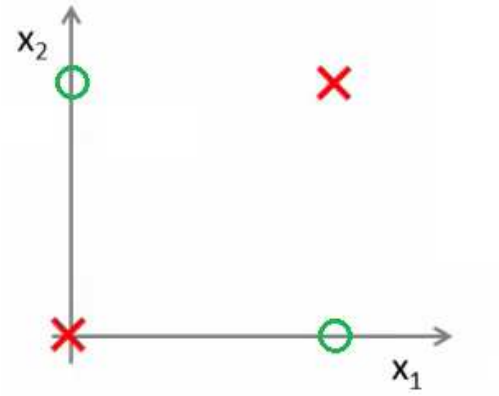
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} * a_0^{(2)} + \Theta_{11}^{(2)} * a_1^{(2)} + \Theta_{12}^{(2)} * a_2^{(2)})$$

(rovnice 4.3)

Dle uvedených rovnic lze snadno vypočítat výstupní odezvu sítě na rozdílné hodnoty vstupního vektoru, jak ukazuje následující rovnice 4.4, která představuje výpočet aktivační hodnoty $a_1^{(2)}$ pro daný neuron. Dále pak bez výpočtů uvádím tabulku pro všechny permutace vstupních hodnot v tabulce na obrázku 4.8, ze které je patrné naprostá shoda s logickým hradlem XNOR i s grafem znázorňujícím situaci, kde křížek značí logickou 1.

Vstupní hodnoty: $x = [0, 1]^T$

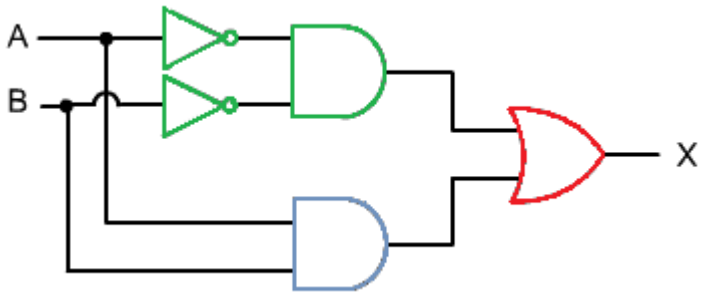
$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} * x_0 + \Theta_{11}^{(1)} * x_1 + \Theta_{12}^{(1)} * x_2\right) == g(-30 * 1 + 20 * 0 + 20 * 1) == g(-10) = \frac{1}{1+e^{10}} = 4.5398e-005 \doteq 0 \qquad \text{(rovnice 4.4)}$$



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Obrázek 4.8: Graf sítě pro XNOR s tabulkou jednotlivých výpočtů

Zvolené uspořádání této demonstrativní neuronové sítě není nijak náhodné, jelikož celkem věrně koresponduje s náhradním schéma logického hradla XNOR, které je na obrázku 4.9. Když se podíváme zpátky na obrázek 4.6 s barevně vyznačenými částmi neuronové sítě, zjistíme, že jednotlivé barevné části se ve funkci plně shodují s logickými hradly na obrázku s náhradním schématem logického hradla XNOR. Propočítáme-li například modře označenou část pro všechny možné vstupní permutace hodnot 0 a 1 vyjde nám logická funkce AND, atd. Dá se tedy říci, že neuronová síť je poskládána stejně jako obvod XNOR z jednotlivých částí.



Obrázek 4.9: Náhradní schéma XNOR
Zdroje obrázků [9]

4.2.2.2 Vektorizace

Z předchozí kapitoly ukázaných rovnic vyplývá možnost vektorizace daných výpočtů. Budeme tedy moci pracovat se vstupním i výstupním vektorem a maticovou reprezentací ostatních parametrů, což značně zjednoduší orientaci.

Jestliže vstupních parametrů sítě je *n*, tak za vstupní vektor můžeme považovat *n+1* prvkový vektor x, n+1 prvkový, protože jsme použily jednotkový vstup $x_0 = +1$.

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

Dále musíme vektorizovat výsledné aktivační hodnoty jednotlivých neuronů ze skryté a výstupní vrstvy $a^{(j)}$. Tím dostaneme dvouprvkový vektor $a^{(j)}$, kde index *j* značí vrstvu, pro kterou jsou tyto hodnoty počítány. $a^{(2)}$ je tedy vektor aktivačních hodnot neuronů pro první a jedinou skrytou vrstvu v příkladu. Dalším krokem vektorizace je definice vektoru *z*, který je definován jako součet součinů jednotlivých parametrů aktivačních funkcí. Jednoduše řečeno, prvek $z_1^{(2)}$ je první prvek vektoru $z^{(2)}$ a jeho velikost je dána parametrem funkce g, při výpočtu aktivační hodnoty $a_1^{(2)}$. Můžeme tedy psát rovnice pro výpočet parametrů skryté vrstvy 4.5.

$$z^{(2)} = \Theta^{(1)} * x$$

$$a^{(2)} = g(z^{(2)}) \qquad \text{(rovnice 4.5)}$$

Potom nesmíme zapomenout přidat jednotkový prvek a_0 vektoru a, udělat tedy z dvouprvkového vektoru vektor tříprvkový. A napsat zbytek rovnic pro poslední výstupní vrstvu neuronové sítě 4.6.

$$z^{(3)} = \Theta^{(3)} * a^{(2)}$$

$$\mathrm{h}\Theta(\mathrm{x}) = a^{(3)} = g\left(z^{(3)}\right) \qquad \text{(rovnice 4.6)}$$

Jelikož jsme provedli kompletní vektorizaci celého výpočtu predikce, je možné výstupní funkci psát ve formě rovnice 4.7:

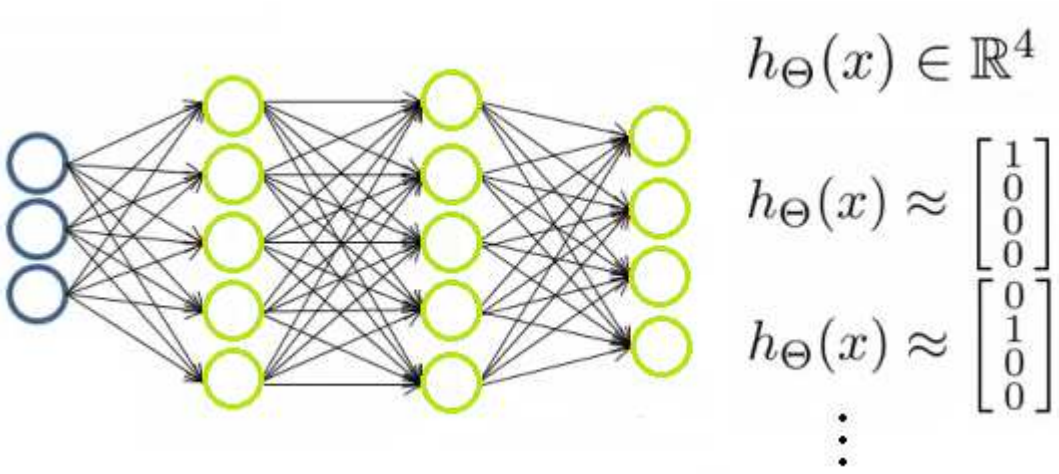
$$\mathrm{h}_\Theta(\mathrm{x}) = \frac{1}{1+ \mathrm{e}^{-\Theta^{\mathrm{T}} * x}} \qquad \text{(rovnice 4.7)}$$

Samozřejmě nesmíme zapomenout vektorizovat váhy theta jednotlivých neuronů, kde $\Theta^{(j)}$ obsahuje váhy synaptických spojů pro neurony ve vrstvě *j*. $\Theta^{(j)} \in R^{jx((j+1)+1)}$, což znamená, že $\Theta^{(1)}$ obsahující váhy pro neurony skryté vrstvy bude matice o velikosti 2x3. První řádek bude obsahovat 3 váhy pro první neuron $a_1^{(2)}$ a řádek druhý 3 váhy pro neuron druhý.

$$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}$$

4.2.2.3 Multi-class klasifikace

Multi-class klasifikace spočívá v definici podoby výstupních vektorů při použití neuronových sítí s víceprvkovou výstupní vrstvou. Má-li výstupní vrstva více prvků, musí být síť naučena tak aby výstupní vektor obsahoval vždy jen maximálně jednu jedničku na kterékoliv pozici. Chceme-li tedy neuronovou síť používat například k identifikaci a rozpoznání postav lidí, to znamená dospělý muž/žena nebo dítě muž/žena, potřebujeme k tomu síť s 4 prvkovou výstupní vrstvou na rozlišení všech osob. A celá síť by mohla vypadat třeba jako ta na obrázku 4.10, i s ukázkou dvou ze čtyř výstupních vektorů.



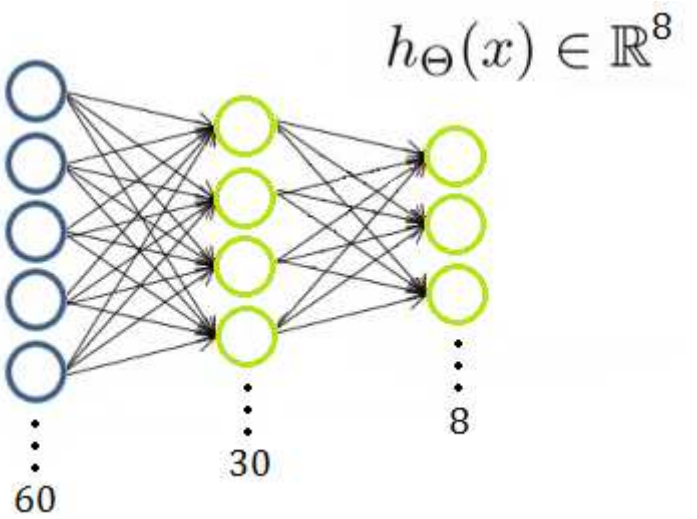
Obrázek 4.10: Multi-Class klasifikace

Takováto podoba výstupních vektorů má mnoho výhod, první z nich je jednoznačnost, pokud bude totiž jednička na více, než jednom místě, můžeme s naprostou jistotou říci, že nastala chyba a hodnota predikce je neplatná. Samozřejmě, že bychom mohli výstupní vektory zakódovat a snížit tak počet prvků ve výstupní vrstvě maximálně na 2 prvky, takováto podoba výstupních vektorů by s sebou však přinesla daleko větší chybovost v predikci a síť bychom sice možná dokázali naučit pracovat s takovouto odezvou, ale jen při takto jednoduché síti a navíc bychom dosahovali daleko více chyb.

4.2.2.4 Implementace v HW

Samotnému autu plně postačí predikční část algoritmu neuronové sítě, nemusí tedy obsahovat část učení. Jak jsem již popsal predikční část, je poměrně jednoduchá a rychlá, a to i v případě rozsáhlejších sítí. Opačně tomu je s částí učení, která je velmi náročná na výpočty i na výpočetní čas, této části budu věnovat zbytek práce.

Jen pro úplnost, algoritmus predikční části, který jsem napsal pro použití v autě, je universální ve smyslu velikosti sítě. Odzkoušel jsem ho tedy na popsané části, která reprezentovala logické hradlo XNOR, a pak jsem jednoduše jen změnil konstanty hodnot, které rozšířily predikční část na velikost potřebnou v autě, tedy 60 prvkový vstupní vektor, 30 prvková skrytá vrstva a 8 prvková výstupní vrstva. Síť implementovaná v autě má tedy pouze jednu skrytou vrstvu, která naprosto postačuje pro funkci sledování černé čáry.



Obrázek 4.11: Použitá konfigurace sítě v autonomním autíčku

Pro výpočet predikce v autíčku, tedy stačilo implantovat dvě matice s vahami neuronů, $\Theta^{(1)} \in R^{30x61}$ a $\Theta^{(2)} \in R^{8x31}$, což s využitím pamětí obvodu FPGA nepředstavovalo velký problém. O trochu větší problém představovala rychlá algoritmizace těchto dvou rovnic 4.8, a to je jedna z hlavních věcí, kterou by bylo potřeba udělat pro možnost predikce zatáčení s autíčkem.

$$a^{(2)}(x) = \frac{1}{1 + e^{-\Theta^T * x}}$$

$$h_\Theta(x) = \frac{1}{1 + e^{-\Theta^T * a^{(2)}}} \qquad \text{(rovnice 4.8)}$$

4.2.3 Učící algoritmus pomocí BackPropagation

4.2.3.1 Gradient descent & Cost function

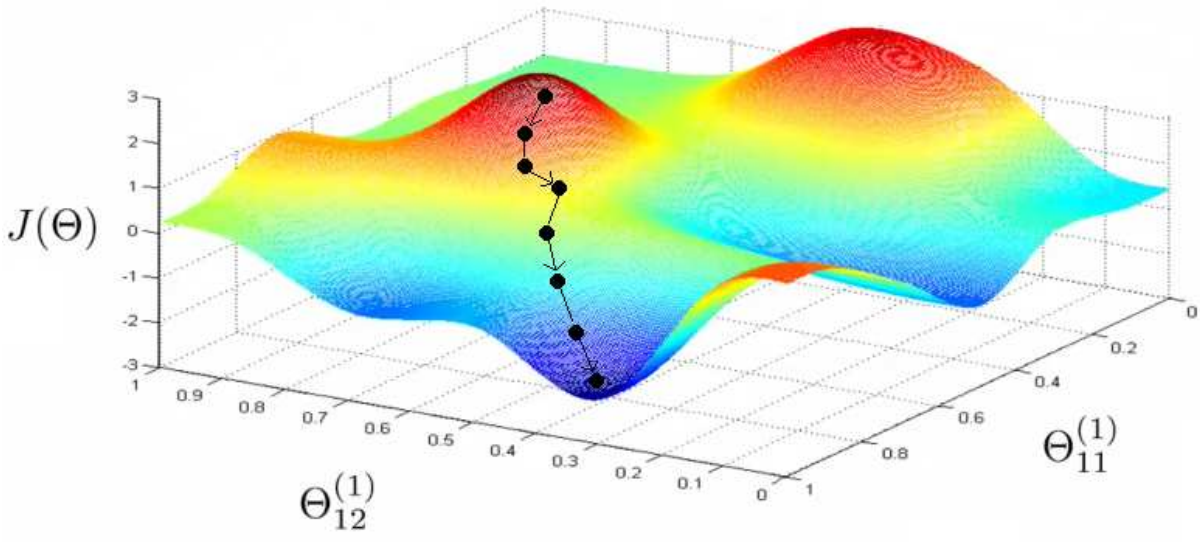
Učení neuronové sítě ve výsledku znamená vhodné nastavení synaptických vah neuronů $\Theta^{(j)}$, které představují jediné konstantní parametry pro správnou funkci celé sítě. K nastavení těchto vah je třeba zavést pojem Cost Function J(Θ), s parametrem synaptických vah, a pomocí funkce váhy vypočítat. Rovnice 4.9 popisuje tuto funkci, kde parametr *m* je počet prvků v tréninkové množině dat, parametr *K* je počet neuronů ve výstupní vrstvě sítě a s_l počet neuronů ve vrstvě l.

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2 \qquad h_\Theta(x) \in \mathbb{R}^K \qquad \text{(rovnice 4.9)}$$

První část rovnice v hranaté závorce je ve skutečnosti výpočet parciální derivace $\alpha\ (\partial/(\partial\Theta_j))$, která má pomocí algoritmu zvaného Gradient Descent, za úkol najít lokální či globální minimum. Druhá část přičtená k parciální derivaci je regularizace, která vylepšuje výpočet hledání minima, bez které se algoritmus obejde a funguje podobně, avšak pomocí regularizace lze mít nad výpočtem větší kontrolu.

Hledání minima přehledně zobrazuje obrázek 4.12, jak je vidět lokálních minim je samozřejmě víc, a na tom do kterého algoritmus nakonec dospěje, závisí na počáteční náhodné inicializaci vah theta. Zásadní otázkou je: Proč vlastně hledat toto minimum? Odpověď je jednoduchá, jelikož právě v globálním minimu funkce J(Θ) jsou hodnoty jednotlivých vah nejlepší a síť funguje s nejmenší možnou chybou.



Obrázek 4.12: Hledání lokálního minima zdroj: MATLAB

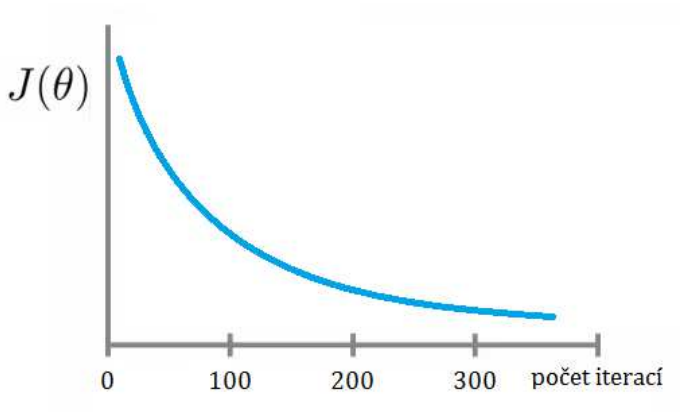
Na obrázku 4.12 jsou zobrazeny pouze 2 váhy Θ, protože zobrazení 3 a více vah už by nebylo přehledné, ve skutečnosti je jich samozřejmě mnohem více. Algoritmus Gradient descent nejlépe popisuje následující rovnice 4.10. Jde o výpočet parciálních derivací a jejich odečtení od aktuálních vah.

while(dokud konverguje) {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

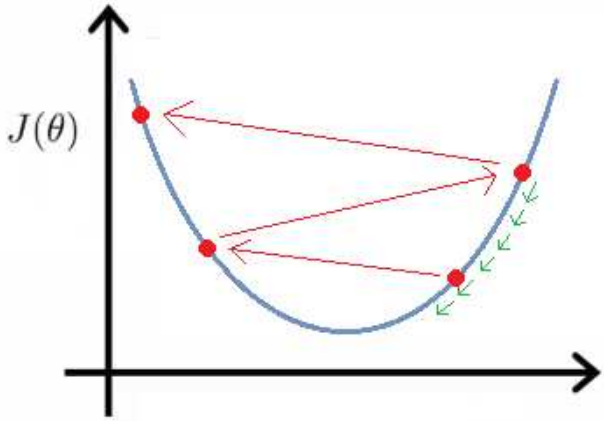
(rovnice 4.10)



Obrázek 4.13: Závislost Cost function na počtu iterací

Při učení a hledání lokálního minima musí Cost function J(Θ) klesat při každé další iteraci. Neklesá-li, je to jasný znak toho, že máme v algoritmu chybu a je třeba ho opravit. Závislost funkce J(Θ) na počtu iterací je znázorněna na obrázku 4.13, v naprosté většině situací bude mít křivka takovýto průběh, s menším či větším stupněm klesání. Podle této závislosti se dá řídit maximální počet iterací potřebný pro výuku sítě, v závislosti na velikosti klesání se dá výuka zastavit. V kompletním algoritmu výuky je tedy dobré zaznamenávat strmost klesání, a pokud není pokles při následující iteraci větší než 10^{-3} je možné algoritmus zastavit a pokládat síť za naučenou.

Parametr α před parciální derivací v rovnici 4.9 a 4.10 je parametr, který udává rychlost učení. Může se pohybovat od relativně malých čísel přibližně 10^{-4} až po 1, která znamená plnou rychlost klesání gradientu, větší číslo by znamenalo násobek vypočteného klesání a téměř jistě by mohlo zapříčinit nefunkčnost algoritmu. Pokud bude lambda pro danou síť moc vysoká, mohlo by dojít k takzvanému přestřelení lokálního minima a závislost Cost function by se chovala podle červených bodů zobrazených na obrázku 4.14. Je ji tedy třeba snížit a vyzkoušet algoritmus znovu, dokud gradient neklesá korektně podle zelených šipek na obrázku.



Obrázek 4.14: Přestřelení lokálního minima

Doba výpočtu a počet potřebných iterací pro úspěšné naučení neuronové sítě je tedy přímo závislá na tomto parametru lambda. Pokud bude moc veliká, může dojít k chybě a přestřelení minima, na druhou stranu pokud bude moc malá, budou výpočty trvat podstatně déle, protože rychlost klesání bude o to zmenšena.

4.2.3.2 Back Propagation

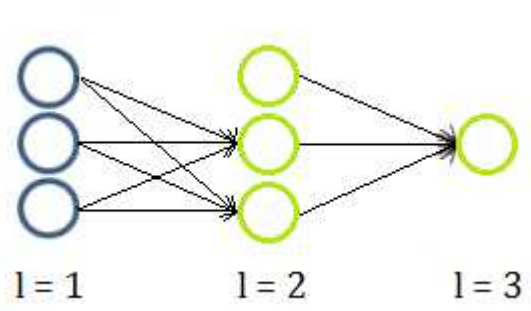
Nejdůležitější část učení je výpočet oné parciální derivace a tím gradient poklesu Cost function, kterou vypočítáme pomocí části algoritmu zvané BackPropagation. Tato kapitola bude tedy popisovat onen výpočet derivace zobrazené jako rovnice 4.11.

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) \qquad \text{(rovnice 4.11)}$$

Před tím než se ale pustíme do výpočtu derivace, musíme provést již popsanou část algoritmu a to ForwardPropagation. Pro příklad s logickým hradlem XNOR, jehož neuronová síť vypadá jako ta na obrázku x, bude ForwardPropagation ve vektorové podobě vypadat následně dle rovnice 4.12.

$$\begin{aligned} a^{(1)} &= x \qquad (add\ x_0) \\ z^{(2)} &= \Theta^{(1)} a^{(1)} \\ a^{(2)} &= g\left(z^{(2)}\right) \qquad \left(add\ a_0^{(2)}\right) \\ z^{(3)} &= \Theta^{(2)} a^{(2)} \\ a^{(3)} &= h_\Theta(x) = g(z^{(3)}) \end{aligned}$$

(rovnice 4.12)



Obrázek 4.15: ForwardPropagation XNOR

Dalším nepostradatelným krokem, než se budeme moci pustit do samotného výpočtu BackPropagation, je třeba definovat jakousi intuici, neboli chybu delta δ, která je nezbytná pro výpočet a počítá se zpětně pro každý neuron zvlášť. Od toho také název BackPropagation, jelikož se chyba neuronů počítá směrem z výstupní ke vstupní vrstvě. Rovnice 4.13 popisují tyto výpočty pro třetí a druhou vrstvu, pro první vrstvu se chyba nemůže počítat, jelikož je to vrstva vstupní a nemá svoje vlastní synaptické váhy. Rovnice jsou opět psány vektorově, ale počítají se pro každý neuron.

$$\delta^{(3)} = a^{(3)} - y \qquad kde: \quad a^{(3)} = h_\Theta(x)$$

$$\delta^{(2)} = (\Theta^{(2)})^T * \delta^{(3)} .* g'\left(z^{(3)}\right) \qquad kde: \quad g'\left(z^{(2)}\right) = a^{(2)} .* \left(1 - a^{(2)}\right)$$

(rovnice 4.13)

První z rovnic 4.13 je pouze pro výpočet chyby výstupní vrstvy, kde akční potenciál je roven výstupu neuronové sítě. Druhá rovnice je použitelná pro každou další skrytou vrstvu, a pokud by jich síť obsahovala více, zvětšily by se pouze indexy o jedničku, obecnou podobu je velmi snadné si odvodit, a proto ji nebudu uvádět.

Nyní, když máme definované vše potřebné, můžeme se konečně pustit do samotných výpočtů algoritmu BackPropagation. Budeme-li uvažovat tréninkovou množinu o *m* prvcích $\{(x^{(1)},y^{(1)}), \dots ,(x^{(m)},y^{(m)})\}$, bude algoritmus obsahovat cyklus o *m* krocích, přičemž v každém kroku se bude brát v úvahu jeden vstupní vektor tréninkové množiny a k němu ekvivalentní vektor množiny výstupní. Jako první před každým započetím cyklu je třeba vynulovat proměnné Δ, které je třeba vytvořit pro akumulaci chyb δ, ve kterých se bude akumulovat chyba synaptických vah po každém kroku cyklu. Samotný cyklus algoritmu potom bude vypadat nějak takto:

for i = 1 to m
- nastavení $a^{(1)} = x^{(i)}$
- spočítat akční potenciál neuronů pomocí FP pro vrstvy $a^{(l)}$ kde $l \in 2,3$
- použít výstupní vektor $y^{(i)}$ pro výpočet chyby $\delta^{(3)}$
- spočítat chyby skryté vrstvy $\delta^{(2)}$
- akumulovat chybu do matice $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}*(a^{(l)})^T$

Po výpočtu akumulované chyby pomocí metody BackPropagation, se dají potřebné parciální derivace získat z následující rovnice 4.14, která je rozdílná pro prvky matic, kdy je index sloupce *j* nulový, jelikož tento sloupec obsahuje údaje o „bias units“, tedy prvků $a_0^{(l)}$.

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} \qquad kde: \quad j \neq 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \frac{1}{m}\Delta_{ij}^{(l)} \qquad kde: \quad j = 0 \qquad \text{(rovnice 4.14)}$$

4.2.3.3 Náhodná inicializace synaptických vah

Pro výpočetní algoritmus Gradient descent nebo BackPropagation, musí být synaptické váhy theta inicializované na nějakou náhodnou hodnotu. V žádném případě nemůžou být nulové, jelikož by akční potenciál všech neuronů ve skrytých a výstupní vrstvě byl stejný, tudíž i výpočet jejich jednotlivých parciálních derivací. To by mělo za následek upravení vah theta na stejnou hodnotu a síť by se nedokázala naučit svoji funkci. Výsledná síť by tedy měla v každé skryté vrstvě všechny neurony stejně reagující na každý z jednotlivých podnětů. Chce to tedy zvolit nějakou konstantu ε a inicializovat váhy na náhodné číslo mezi kladnou a zápornou hodnotou této konstanty, v mém případě dobře posloužila hodnota ε = 0,12. Zvolená konstanta by ve většině případů neměla překročit jedničku.

4.2.3.4 Volba počtu neuronů a skrytých vrstev sítě

Počet neuronů a skrytých vrstev potřebných pro optimálně fungující neuronovou síť je vždy těžká otázka a většinou ji objasní až praxe s několika pokusy. Existují však jakási doporučení a defaultní nastavení, které nám pomohou s počátečním výběrem. Defaultní počet skrytých vrstev neuronových sítí bývá vždy jen jedna, jelikož většina problémů se dá vyřešit pouze s jednou vrstvou a síť funguje dobře. Samozřejmě se můžeme pokusit o více vrstev a prověřit zda nebude tato implementace lepší.

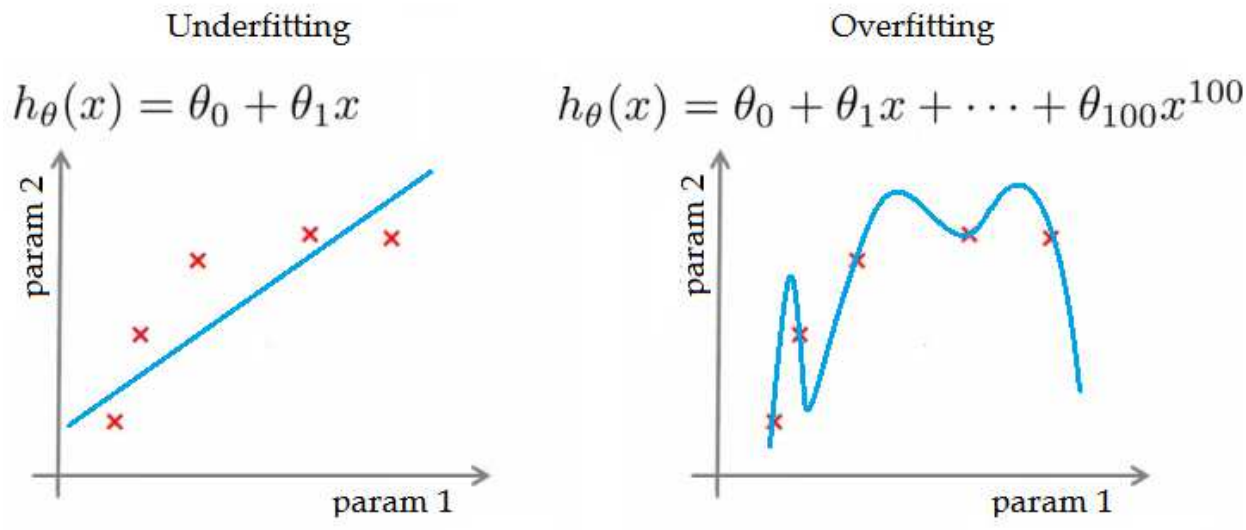
- Pokud jde o počet neuronů v jedné či více skrytých vrstvách, měl by být vždy větší než počet neuronů ve vrstvě výstupní a ve většině případů i ve vrstvě vstupní, nejedná-li se však o zpracování obrazu, kdy je obrazových bodů a tedy vstupních parametrů mnoho. V tom případě může být počet neuronů ve skrytých vrstvách nižší, než ve vrstvě vstupní, protože by byl nadbytečný.

- Jestliže si nejsme jisti při volbě počtu neuronů ve skrytých vrstvách, tak větší číslo je vždy lepší, jelikož nám poskytne větší variability, i když ve výsledku mohou být některé neurony redundantní. Tedy mohou některé pracovat stejně, nebo nepracovat vůbec, ale to je pořád lepší než kdyby jich bylo málo a síť více „chybovala“.

- Posledním doporučením je, že pokud se rozhodneme použít více skrytých vrstev než jednu, měl by být ve všech těchto vrstvách stejný počet skrytých neuronů.

Při volbě počtu neuronů ve skrytých vrstvách se musí brát na zřetel možnost výskytu chyby „přeučením“ neboli overfitting ale také „podučením“ underfitting. Tyto dvě vlastnosti vysvětlím na jednoduchém příkladu týkajícího se problému lineární regrese, která je ve své podstatě velmi podobná problému neuronových sítí, neboli obě mají Cost function J(Θ) na stejném principu a velmi podobnou.

Pokud jde o overfitting, je to jako bychom se snažili aproximovat určité body polynomem o velmi velkém řádu a aproximovaná charakteristika by přesně seděla pouze v daných bodech, ale v okolí těchto bodů by se chovala nepřístojně, jako na obrázku 4.16. Tato chyba může nastat, použijeme-li neuronovou síť s větším počtem neuronů ve skryté vrstvě, nebo více skrytých vrstev než je potřeba. Underfitting je problém opačný, jako bychom se snažily aproximovat body polynomem nedostatečného řádu, jak také ukazuje obrázek 4.16, a tento jev nastává při použití poddimenzované, tedy neuronové sítě s menším počtem neuronů ve skryté vrstvě než je potřeba. Problém underfittingu se nedá vyřešit jinak, než přidáním dalších neuronů do sítě. Zato problém overfitting, může z části kompenzovat vhodná volba velikosti regularizačního parametru lambda λ.



Obrázek 4.16: Under/Over fitting

4.2.3.5 *Shrnutí & Implementace algoritmu*

Implementace algoritmu se dá shrnout do 6 kroků, kterých se je třeba držet k dosažení funkčně naučené neuronové sítě.

1. Náhodná inicializace vah Θ
2. Implementace ForwardPropagation pro výpočet $h_\Theta(x)$ pro všechny $x^{(i)}$
3. Kód pro výpočet Cost function J(Θ)
4. Implementace BackPropagation pro výpočet parciálních derivací
5. Použít kontrolu výpočtu gradientu …viz níže
6. Použít Gradient descent nebo jinou pokročilou metodu s BackPropagation k minimalizaci Cost function J(Θ) s parametry theta, a opakování od kroku 2 až do dosažení požadovaného výsledku

O bodu 5, tedy kontrole správnosti výpočtu gradientu jsem se zatím nezmiňoval, ale jde o velmi jednoduchou proceduru, při které se zkontroluje správnost implementovaného kódu BackPropagation. Parciální derivace se dají vypočítat stejně za pomoci algoritmu BackPropagation, tak pomocí zdlouhavého a výpočetně daleko náročnějšího výpočtu z rovnice 4.9, tedy Cost function bez regularizační části. Takto bychom měli pouze zkontrolovat korektnost obou výpočtů při první iteraci, kde by výsledky měli být velmi podobné. Potom bychom toto ověřování měli vypnout, protože by učení trvalo příliš dlouhou dobu.

Popsaný algoritmus jsem implementoval v MATLABu a váhy naučené sítě jsem přenesl do algoritmů autíčka. Všechny algoritmy jsou přiložené na CD.

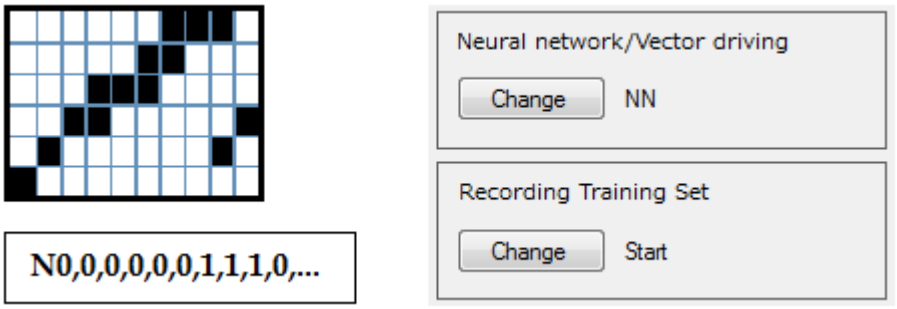
4.2.4 Realizace

V této kapitoly podrobně popíši celý postup učení neuronové sítě, nepůjde už o realizaci algoritmů, kterou jsem popsal v předchozí kapitole, ale spíše o manuální kroky, které bylo potřeba udělat pro úspěšnou implementaci vah sítě do algoritmů auta.

Poté, co se tedy úspěšně povedlo implementovat učení neuronové sítě typu perceptron s jednou skrytou vrstvou v programu MATLAB a odzkoušet jeho funkčnost na logickém hradle XNOR, se naskytl problém vytvoření tréninkové množiny dat pro použitou síť. Bylo potřeba vytvořit *m* řádkovou matici o 60 sloupcích, kde každý řádek představuje jednu komprimovanou obrazovou matici rozloženou do vektoru po jednotlivých řádcích, představující vstupní data neuronové sítě. Druhou maticí, potřebnou k učení byla matice výstupních vektorů, která má samozřejmě rovněž *m* řádků a počet sloupců shodných s rozměrem výstupní vrstvy sítě, tedy 8.

Prvním krokem k vytvoření tréninkových dat, bylo získání potřebné množiny 60 prvkových komprimovaných obrazových matic, které jsem si nechal posílat z auta pomocí komunikačního rozhraní bluetooth. Kvůli ladícím účelům si v autě nechávám posílat z obvodu FPGA do procesoru krom samotných řídících vektorů i onu obrazovou matici, kterou vykresluji na LCD displej. Proto požadavek na přeposílání matice do počítače nebyl velký zásah do algoritmů, ale realizoval jsem jej celkem snadno tak, že po každém detekovaném přijetí dat přerušením a jejich validaci, je ihned přepošlu do PC, pokud je tato volba zapnuta.

Pro příjem dat z auta jsem využil program napsaný v prostředí C# popsaný v kapitole 3.7.2 a zobrazený na obrázku 3.17. Jednotlivé obrazové bity, tedy prvky matice, přicházejí z auta odděleny čárkou a začátek každé nové matice je pro přehlednost označen velkým písmenem „N“. Data v takovéto podobě jsou vypsána v richTextBoxu zabírajícího většinu aplikace. Nutnou podmínkou aby procesor přeposílal data do počítače je správné nastavení parametrů programu, umělá inteligence musí být přepnuta na neuronovou síť a samozřejmostí je povolení zapnutí parametru „Recording Training Set“, který slouží jako hlavní spínač této volby.



Obrázek 4.17: Nastavení parametrů programu a zobrazení posílaných dat

Na obrázku 4.17 je vidět potřebné nastavení parametrů v podpůrném programu a ukázka 60 prvkové matice posílané z auta, která v textové podobě představuje první řádek matice počínaje úvodním písmenem „N“.

Poté, co jsem dokázal přijímat komprimovanou obrazovou matici tímto způsobem, převzal jsem kontrolu nad autem pomocí kláves na PC a několikrát projel postavenou tréninkovou trať a neustále získával nové data. Po získání přesně 369 obrazových matic, jsem proces ukončil a začal získané data připravovat na další zpracování.

Takovým mezikrokem bylo překopírování dat do Excelu, odkud už jsem je dokázal nahrát do MATLABu a vytvořit datový soubor „*.mat“.

Konečně jsem tedy měl matici vstupních dat neuronové sítě, potřebnou pro její učení a mohl jsem se pustit do vytvoření druhé matice výstupních dat, kterou jsem musel napsat ručně ke každému vektoru neboli řádku první matice. Proto jsem si napsal m-file, který mi po každém spuštění přehledně zobrazil jednu obrazovou matici v jejím skutečném formátu, 6x10, do Command Window a jednoduchým skriptem jsem k ní vytvořil výstupní vektor, podle svého nejlepšího uvážení. Takto jsem postupně ohodnotil všech přijatých 369 vektorů, ke kterým mi vznikla matice výstupních dat o velikosti 369x8 prvků. Obě matice jsem souměrně zrcadlil podél vertikální osy, zrcadlená data přidal k původním, a tím vytvořil tréninkovou množinu dat obsahující stejné informace pro učení o zatáčkách na obě strany o velikosti 738 prvků.

Dalším krokem bylo samotné učení neuronové sítě, spuštěním dříve popsaného algoritmu s použitím obou matic. Nepostradatelnou částí bylo také otestování správnosti naučení sítě, jestli vykazuje správnou odezvu na náhodně zvolené vstupní vektory. K tomu je dobré použít část dat z tréninkové množiny, která nebyla použita k učení, ale pouze k tomuto testování, aby se ověřila správnost predikce. V mém případě nebylo prvků tréninkové množiny mnoho, proto jsem je na učení použil všechny a testovací vektory jsem si jednoduše vymyslel podle intuice a vlastností, které jsem chtěl otestovat.

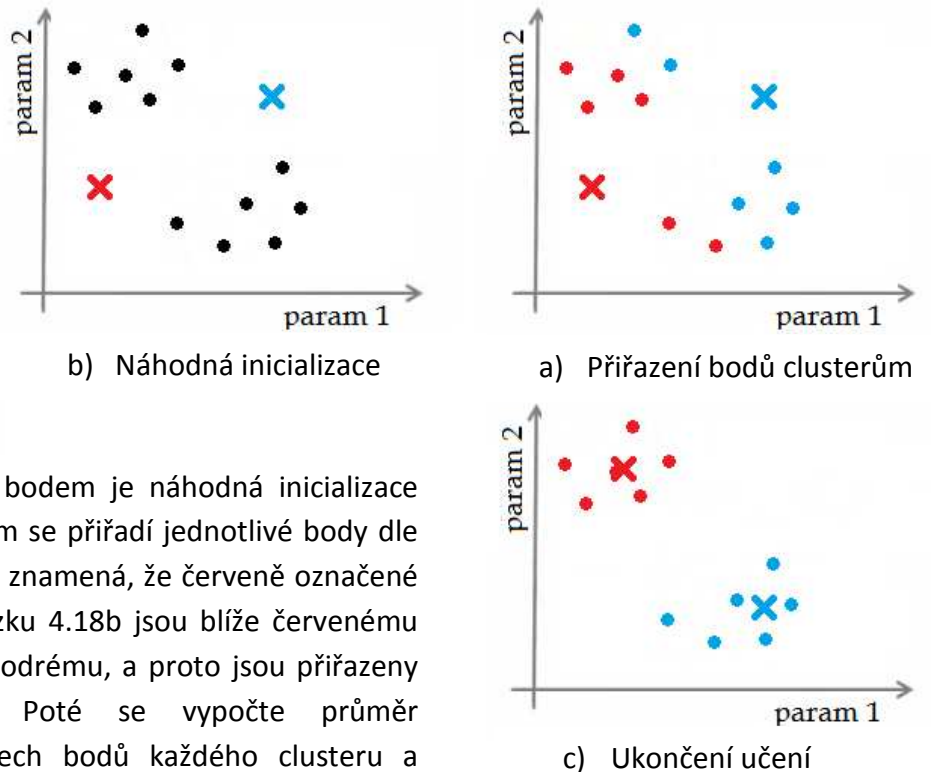
Nakonec jsem ověřené a funkční váhy neuronové sítě vynásobil tisícem a zaokrouhlil na celá čísla, abych v obvodu FPGA nemusel použít reálná čísla a kvůli nim speciální knihovny či funkce, které syntéza těchto obvodů standardně nepodporuje. K dokončení celého procesu už jen stačilo překopírovat získané váhy na příslušná místa v algoritmech predikce neuronové sítě a auto bylo připraveno k první samostatné jízdě.

4.2.5 Výběr sítě s učitelem

Možná trochu netradičně, na závěr, jsem se rozhodl vysvětlit důvod, proč jsem zvolil typ sítě s učitelem, i když je tato varianta náročnější na učení v tom, že vyžaduje matici výstupních odezev sítě a podle mého obsahuje i náročnější algoritmus Back Propagation. Pokud bych totiž zvolil neuronovou síť bez učitele založenou na shlukové analýze dat, nemusel bych vytvářet onu matici výstupních odezev.

Jako jeden z možných typů učení bez učitele jsem zvolil algoritmus „K-means Clustering“, který je založen na shlukové analýze dat. Je to konvergentní iterační algoritmus, který přiřazuje vstupní příznakové vektory jednotlivým třídám na základě minimalizace vzdálenosti mezi příznakovým vektorem a vzorem třídy. Nutnou podmínkou pro zpracování dat tímto algoritmem je znalost počtu tříd, do kterých se mají data rozdělit. V mém případě je jich 8, stejně jako počet výstupních neuronů předchozí sítě, jelikož požaduji stejné rozlišení výstupního vektoru.

Algoritmus se dá asi nejlépe vysvětlit dle obrázku 4.18, na kterém vysvětlím jednotlivé kroky algoritmu pomocí dvou nezávislých parametrů a pouze dvou clusterů. V případě rozšíření algoritmu na stávající problém samonavigace auta by obsahoval 80x60, tedy 4800 nezávislých parametrů a 8 clusterů neboli tříd.



b) Náhodná inicializace

a) Přiřazení bodů clusterům

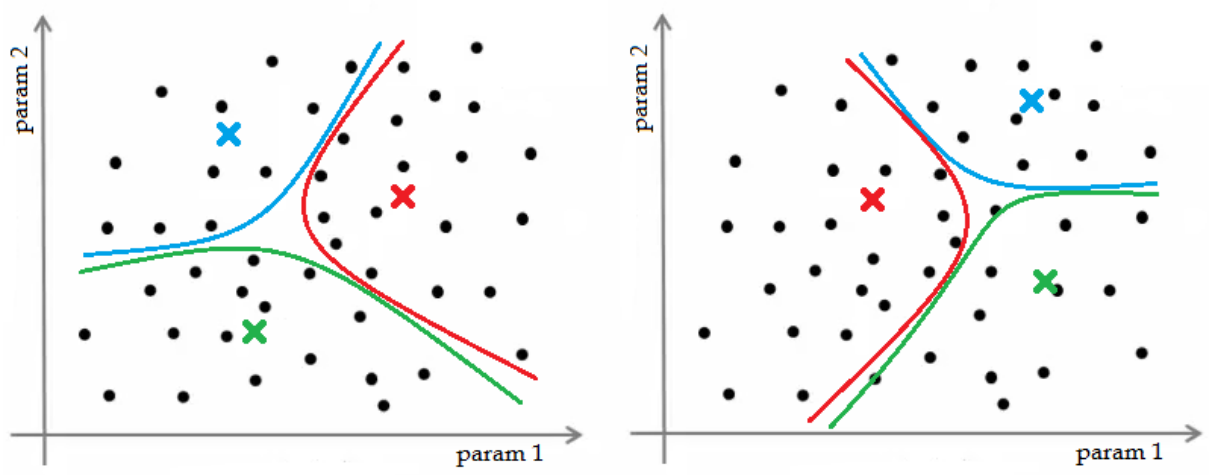
c) Ukončení učení

Obrázek 4.18: K-means Clustering

Prvním bodem je náhodná inicializace clusterů, kterým se přiřadí jednotlivé body dle vzdálenosti. To znamená, že červeně označené body dle obrázku 4.18b jsou blíže červenému clusteru než modrému, a proto jsou přiřazeny právě jemu. Poté se vypočte průměr vzdáleností všech bodů každého clusteru a podle toho se upraví jejich poloha. Takto se clustery přemisťují, dokud algoritmus nedosáhne příslušného počtu iteračních cyklů.

Po popsaném naučení, neboli umístění jednotlivých clusterů na příslušné pozice, algoritmus rozhoduje o náležitosti nových prvků množiny daným clusterům samozřejmě dle minimální vzdálenosti. K přiřazení prvku danému clusteru tedy postačí jednoduchý cyklus, který spočítá vzdálenosti ke všem clusterům a zapamatuje si ten nejbližší.

Největší nevýhodou a vlastností pro kterou jsem se algoritmus K-means Clustering rozhodl nepoužít je právě princip shlukové analýzy ne které je algoritmus postaven. Jelikož pokud budou data „přehledně“ rozdělena tak jako při modelové situaci na obrázku 4.18, nebude algoritmus vykazovat sebemenší problém a bude fungovat výborně. Opačná situace nastane, pokud nebudou data oddělena pomyslnou mezerou a budou se vzájemně prolínat, tak jako na obrázku 4.19. Potom algoritmus rozdělí data do tříd víceméně náhodně a výsledek nemusí být vždy přijatelný. Obrázek ukazuje možné přidělení bodů do jednotlivých tříd, při použití 3 clusterů. Takovýto problém může nastat u každého algoritmu založeného na shlukové analýze dat.



Obrázek 4.19: Možnosti rozdělení bodů do tříd

V mém případě jsou bohužel data rozmístěna v pomyslném grafu rovnoměrně a algoritmus je proto nemusí rozdělit do tříd dle našich požadavků. Pokud tedy použijeme neuronovou síť typu perceptron s učitelem, máme větší kontrolu nad tím, jak má síť reagovat na jednotlivé variace vstupních proměnných, jelikož při učení přesně definujeme, jak by síť měla reagovat (odezvu) na danou vstupní množinu a jí podobné.

Algoritmus K-means Clustering jsem naprogramoval rovněž v prostředí MATLAB, vyzkoušel pro danou situaci, ale nebyl jsem s ním spokojen tak jako s neuronovou sítí a proto jsem se rozhodl ho nepoužít. Tyto Algoritmy jsou rovněž přiloženy na CD.

5 Závěr

Podle mého názoru se povedlo vytvořit spolehlivé autonomní auto, které je schopné sledování černé čáry i za ne zcela dokonalých podmínek, při udržování relativně velké závodní rychlosti. Auto dokáže odfiltrovat neočekávané nerovnosti či jiné černé nečistoty na závodní dráze, poradí si i se zatáčkou s menším poloměrem než je maximální poloměr zatáčení auta a v případě vyjetí z dráhy se dokáže vrátit a bez problémů pokračovat v jízdě.

Vzhledem k přítomnosti LCD displeje, komunikaci přes rozhraní bluetooth a možnosti uživatelského řízení pomocí kláves na PC nebo ovládání RC soupravou jsou poměrně velké i možnosti ladění a vývoje algoritmů jak v obvodu FPGA, tak v procesoru.

Jak bylo již vysvětleno, žádný z programovatelných elektronických obvodů neběží ani zdaleka na jeho absolutním maximu. Proto se nabízí několik možností jak auto vylepšit a ještě tak zvýšit jeho maximální možnou závodní rychlost. Zásadním krokem jak toho dosáhnout by bylo zvýšení komunikační rychlosti s kamerou, což by vyžadovalo vyřešit problém s rušením kamery a zabezpečit tak její dlouhodobý bezchybný chod.

Další možností jak vyřešit celý systém by bylo vynechání procesorové desky Stellaris LM3S1968 Evaluation Board a vyřešit celý systém v obvodu FPGA za využití některého syntetizovatelného softwarového procesoru, například NIOS II podporovaného ALTEROU. Tím by se ušetřilo nejen místo v mechanické konstrukci auta, o trochu by se i zvýšila životnost baterií, v tomto případě minimálně a zanedbatelně, ale zvýšila by se i komunikační rychlost mezi procesorem a syntezovanými obvody, která momentálně běží na frekvenci 115,2kHz a její navýšení by bezpochyby nepatrně vylepšilo rychlost odezvy auta.

Na závěr bych rád bez bližšího srovnání dodal, že auto je podle mého názoru plně konkurence schopné ostatním autům účastnících se soutěže The Freescale Cup. Soudím tak objektivně jen podle videí natočených ze závodů a podle časů dosažených na různě dlouhých tratích.

6 Seznam použité literatury

[1] The Freescale Cup Wiki
http://thefreescalecup.wikidot.com/en:main

[2] uCAM-TTL Datasheet
http://www.robot-electronics.co.uk/datasheets/uCAM-DS-rev4.pdf

[3] Stellaris LM3S1968 Microcontroller Datasheet
http://www.ti.com/lit/ds/symlink/lm3s1968.pdf

[4] DRV8833 Dual H-Bridge Motor Driver Datasheet
http://www.ti.com/lit/ds/symlink/drv8833.pdf

[5] Stanford Engineering online course: Machine Learning
http://www.ml-class.org

[6] LCD Single Chip Driver ITDB02-3.2 Datasheet
http://www.adafruit.com/datasheets/ILI9325.pdf

[7] Touch Screen Controller ADS7843 Datasheet
http://www.ti.com/lit/ds/sbas090b/sbas090b.pdf

7 Zdroje obrázků

[1] Stránky společnosti Diecast Model Store – Model auta Nisan GTR
http://www.diecastmodelstore.com/1-24-diecast-car-models/187-nissan-gt-r-r35-2009-1-24-grey-ym24209mg.html
a stránky společnosti Lenovo – Notebook E420s
http://shop.lenovo.com/ilind/il/en/learn/products/laptops/thinkpad/thinkpad-edge/e420s/

[2] Stránky Robotika SK – Pravidla soutěže TFC
http://www.robotika.sk/events/12TFC/TFC_2011_Rules_EMEA%20v3.pdf

[3] E-shop Robot Electronics – Modul kamery uCam
http://robot-electronics.co.uk/acatalog/Cameras.html

[4] Stránky společnosti terasIC – Deska s FPGA
http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=139&No=593

[5] Stránky společnosti ITead Studio – LCD displej
http://iteadstudio.com/store/index.php?main_page=popup_image&pID=263

[6] Stránky společnosti PV Electronic – Bluetooth modul
http://pvelectronic.inshop.cz/inshop/catalogue/products/pictures/09358-01.jpg

[7] Stránky otevřené encyklopedie WIKIPEDIA – Model neuronu
http://cs.wikipedia.org/wiki/Neuron

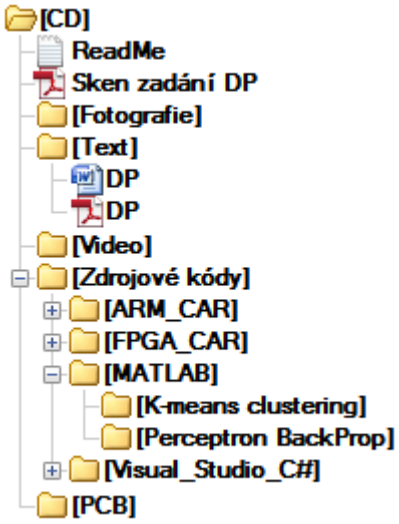
[8] Stránky otevřené encyklopedie WIKIPEDIA – XNOR
http://fr.wikipedia.org/wiki/Fichier:Xnor-gate-en.svg

[9] Stránky CCI Technology Department – XNOR náhradí schéma
http://www.cci-compeng.com/Unit_2_Electronics/2303_How_Logic_Gates_Work.htm

8 Přílohy

Samotné programy jsou vcelku rozsáhlé aplikace a jejich funkce jsem již představil v jednotlivých kapitolách této práce. Proto nebudu přikládat samotný zdrojový kód, ale uvedu pouze obsah CD, na kterém jsou všechny algoritmy přiloženy.

Obsah přiloženého CD:



Fotografie funkčního prototypu auta: