CHALMERS

Självkörande autonom bil i miniformat

Examensarbete inom Data- och Informationsteknik

Tomas Bäckman

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2021

CHALMERS

Självkörande autonom bil i miniformat
Tomas Bäckman

Handledare: Sakib Sistek, Chalmers Tekniska Högskola
Examinator: Jonas Duregård, Chalmers Tekniska Högskola

Kandidatarbete 2021
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
412 96 Göteborg
Telefon: 031-772 1000

Omslagsbild: JetRacer, byggd kring NVIDIAs enkortsdator Jetson Nano [1]

Institutionen för Data- och Informationsteknik Göteborg 2021

Sammanfattning

Ett högst aktuellt ämne inom bilindustrin idag är självkörande fordon. Många företag utvecklar sina egna varianter. Detta eftersom det finns potential till stora kostnadsbesparingar med självkörande fordon. Syftet med det här projektet var att konstruera en mindre självkörande bil som kan navigera på egen hand runt en markerad bana. Bilen som användes kallas JetRacer och är byggd kring enkortsdatorn Jetson Nano från NVIDIA. Styrningen regleras av ett artificiellt neuronnät. Neuronnätet tränades med bilder insamlade från en kamera som sitter längst fram på JetRacer:n. Förutom bildinformationen i sig innehåller bilderna information om vilken riktning bilen ska köra. Huvudmålet med arbetet var att få bilen att köra runt en markerad bana, vilket uppfylldes. För att uppnå god navigering visade sig kvaliteten i bildindata vara en avgörande faktor. All träningsdata från bilen samlades in under medursvarv. Testningen efteråt visade att bilen klarade att köra även motursvarv, vilket visar att det rör sig om verklig maskininlärning. Detta arbete genomfördes inom ramen teknikföretaget Knightecs utveckling av självkörande fordon. I framtida undersökningar skulle det vara intressant att utrusta bilen med förmågan att stoppa eller väja för hinder, bromsa om den kommer utanför banan och öka toleransen för olika ljusförhållanden.

Nyckelord: självkörande bilar, autonoma fordon, artificiell intelligens, neuronnät, neurala nätverk, maskininlärning, djupinlärning, NVIDIA, Jetson Nano

Abstract

Self-driving cars are a hot topic among car manufacturers and in the tech industry today. Many companies develop their own models. One of the driving forces behind a wider use of self-driving cars is the potential for great cost savings. The purpose of this project was to design a miniature self-driving car that can navigate on its own around a track. The car that was used in the project is called JetRacer and is built around the single card computer Jetson Nano from NVIDIA. The navigation of the car is regulated by an artificial neural network. This neural network was trained with images collected from a camera located at the front of the JetRacer. In addition to the image information itself, the images contained information about which direction the car should drive. The main goal of the work was to get the car to drive around a track. That goal was fulfilled. To achieve good navigation, the quality of image data proved to be a crucial factor. All training data from the car was collected during clockwise turns. The testing afterwards showed that the car was also able to drive counterclockwise, which shows that this is truly machine learning. This project was carried out within the framework of the technology company Knightec's development of self-driving vehicles. In future investigations it would be interesting to equip the car with the ability to stop or give way to obstacles, brake if it comes off the track and increase the tolerance for different lighting conditions.

Keywords: self-driving cars, autonomous vehicles, artificial intelligence, neural networks, machine learning, deep learning, NVIDIA, Jetson Nano

Förord

Ett stort tack till André Dankert på Knightec som kom med idén till arbetet och som har handlett mig och kommit med viktiga synpunkter under vägens gång. Även min handledare på Chalmers Sakib Sistek förtjänar ett stort tack för sin uthållighet och för sitt sätt att inspirera mig under våra samtal. Mina tankar går också till de fina kurskamrater jag haft under utbildningens gång – Aleksander, Johan, Jonatan, Julius, Mellard och Sam – hoppas ni har det bra. Sist men inte minst vill jag tacka min sambo som orkat sitta i samma lägenhet som mig medan jag skrivit detta arbete och som stöttat mig med korrekturläsning och uppmuntran.

Terminologi och förkortningar

Aktiveringsfunktion:	På engelska *activation function*. En funktion som bestämmer en neurons slutliga utdata. Antingen slutlig utdata för neuronnätet eller utdata som blir indata till nästa lager. Tidigare användes ofta en sigmoid funktion, numer används oftare styckvis linjär funktion (på engelska *rectified linear unit, ReLU*)
AlexNet:	Det faltningsnätverk som sparkade igång dagens våg av forskning på neuronnät.
Artificiell intelligens (AI):	Ett paraplybegrepp för maskininlärning, artificiella neuronnät och djupinlärning som syftar till maskiners förmåga att efterlikna mänsklig intelligens.
Artificiella neuronnät (ANN):	På engelska *Artificial Neural Network (ANN)*. Benämns också ofta som artificiella neurala nätverk på svenska men Svenska datatermgruppen artificiella neuronnät. Ett nät av neuroner som tar beslut på ett sätt som inspirerats av den mänskliga hjärnan.
Bakåtpropagering	På engelska *back propagation*. På detta sätt tränas ett neuronnät för att kunna ta bättre beslut.
Bias:	Sista korrigeringen av utvärdet innan det skickas till aktiveringsfunktionen.
Djupinlärning:	På engelska *deep learning*. En underkategori till artificiella neuronnät. Djup p.g.a. att neuronnäten har många dolda lager.
Faltningsnätverk (CNN):	På engelska *convolutional neural network*.
Helanslutet neuronnät:	På engelska *fully connected neural network*.
JetPack SDK:	NVIDIAs utvecklingsplattform för djupinlärning på deras Jetson-produkter.
JetRacer:	En minibil som är byggd kring en Jetson Nano.
Jetson Nano:	En enkortsdator från NVIDIA som är minstingen i deras Jetson-serie.
Jupyter Notebook:	Ett integrerat sätt att presentera text och körbar kod på en webbsida.
Maskininlärning:	Ett paraplybegrepp för artificiella neuronnät och djupinlärning. Metoder som försöker lära datorer att göra intelligenta val.
Neuron:	En artificiell neuron inspirerad av den biologiska neuronen. Tar in många invärden och spottar ur sig ett utvärde. Också kallad nod.
PyTorch:	Facebooks programbibliotek för maskininlärning.
ResNet-18:	Det faltningsnätverk som används i detta arbete.
TensorFlow:	Googles programbibliotek för maskininlärning.
Vikt:	Ett mått på hur mycket tidigare utdata kommer påverka nuvarande neuron.

Innehållsförteckning

1 Introduktion 1
1.1 Bakgrund 1
1.2 Syfte 1
1.3 Mål 1
1.4 Avgränsningar 2

2 Teori 2
2.1 Artificiellt neuronnät 3
2.2 Bildigenkänning med neuronnät 4
2.3 Speciella modeller 5
2.3.1 AlexNet 5
2.3.2 SqueezeNet 5
2.3.3 ResNet-18 5
2.3.4 DenseNet 6
2.4 Djupinlärning i Python 6
2.5 Hårdvara 6

3 Metod och genomförande 6
3.1 Material 6
3.1.1 Hårdvara 6
3.1.2 Mjukvara 9
3.2 Anslutningen till bilen 10
3.3 Inställningar av mjukvaran 11
3.4 Styrning 12
3.4.1 Manuell styrning 12
3.4.2 Styrning med en PS4-kontroll 12
3.5 Indata 12
3.6 Träning av neuronnät 14
3.7 Testkörning 14

4 Resultat 15
4.1 Montering 15
4.2 Navigering 16
4.3 Olika platser 17

5 Diskussion 18
5.1 Etik 19
5.1.1 Ansvar 19
5.1.2 Kommunikation med neuronnät 19
5.2 Hållbarhet 20

6 Slutsats 20

Litteraturförteckning 21

Bilagor 23

1 Introduktion

Autonoma fordon eller självkörande fordon är ett aktuellt ämne inom dagens bilindustri. I stort sett alla större bilmärken har börjat utveckla sina egna självkörande bilar. Även klassiska teknikföretag har visat intresse. Google har t.ex. sitt Waymo-projekt [2] och Apple håller på med något liknande [3]. I Göteborg fick vi i januari 2021 vår första självkörande buss i Västtrafiks linjetrafik som går på Lindholmen mellan Hugo Hammars Kaj och Regnbågsgatan [4]. Det är inte konstigt att nyheter om självkörande bilar får stort utrymme i media. Vi kan själva tänka oss hur stor samhällspåverkan fordon som kan transportera människor och varor utan förare skulle innebära.

1.1 Bakgrund

Det finns många problem i utvecklingen av självkörande fordon så som kostnader att bygga en fullskalig bil och kostnader för att laga testfordon som krockat på grund av att programvaran inte betett sig som den ska. Även lagar och regler gör det svårt att testa fordon på vanliga vägar. Och vad händer om fordonet skadar människor eller egendom?

För att komma runt dessa problem kan man skapa en mindre självkörande bil som kan testa mjukvaran utan att riskera skada vare sig på människor eller egendom samt markant minska tillverkningskostnaderna. Det är precis vad som görs i detta examensarbete. Konsultföretaget Knightec har tagit initiativ till att utveckla en liten bil, ca 25 cm lång och 20 cm bred, byggd på NVIDIA:s enkortsdator Jetson Nano [5]. Bilen ska styras runt en bana med hjälp av mjukvara som använder sig av maskininlärning för att tolka bilder som finns på en kamera längst fram på bilen och sedan välja hur bilen ska svänga. Bilens hårdvara utgörs av en byggsats som kallas JetRacer. Byggsatsen består av enkortsdatorn Jetson Nano, ett batteri och karossdelar (se bilaga 1 för byggsatsens fullständiga innehållsförteckning).

Det här examensarbetet genomfördes hos Knightec som är ett svenskt konsultföretag med ca 700 anställda med huvudkontor i Örnsköldsvik och ett lokalt kontor i Göteborg. I Göteborg har Knightec flera uppdrag inom fordonsindustrin, bl.a. relaterat till utvecklingen av autonoma fordon. Därav intresset att utveckla en autonom bil i miniformat för att enkelt kunna testa olika sätt att styra en självkörande bil.

1.2 Syfte

Syftet med det här projektet är att få en mindre bil med längden 25 cm att styra sig själv runt en markerad bana. Styrningen sker med hjälp av ett artificiellt neuronnät som finns på en enkortsdator som är monterad på bilen.

1.3 Mål

Huvudmålet som framgår av syftet är att få bilen att själv navigera runt en markerad bana. För att nå detta huvudmål ställs följande delmål upp:

- Sätta samman bilens hårdvara och mjukvara till en fungerande enhet.
- Styra bilen med hjälp av manuellt inskrivna kommandon, i en Jupyter Notebook [6]

- Bilen ska kunna styras med en spelkontroll eller ett tangentbord för att förenkla insamling av data. Den ska också kunna ta bilder och spara dem på disk, så att de sedan kan användas som träningsdata vid maskininlärning.
- Träna ett artificiellt neuronnät.
- Testköra bilen på banan.
- När bilen tränats ska den kunna navigera autonomt längs den markerade banan.
- Bilen skall klara av att navigera korrekt med banan placerad på olika platser och utan påverkan från omgivningen utöver banan.

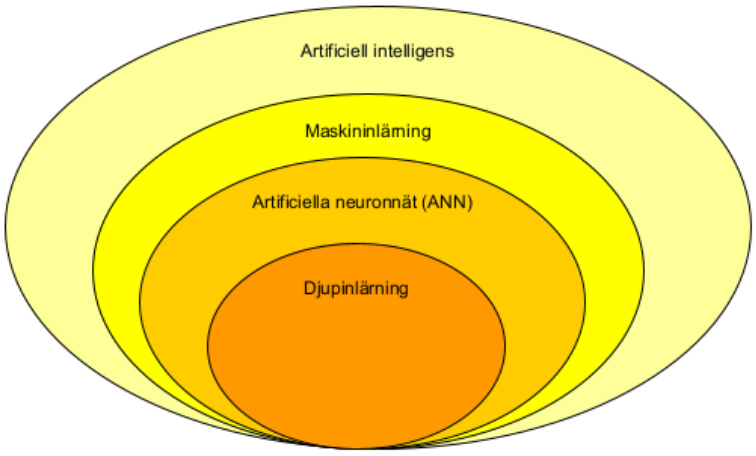
1.4 Avgränsningar

Bilen ska endast kunna följa den väg som finns markerad på en grå plastmatta. Inga andra underlag kommer att testas. Bilen kommer inte heller att utrustas med förmågan bromsa eller väja för olika föremål och hinder som eventuellt kommer i vägen. Medan bilen körs kommer gaspådraget att vara konstant och endast styrningen kommer regleras. Olika modeller (se teoriavsnittet) av neuronnät kommer inte att testas mot varandra.

2 Teori

I detta avsnitt kommer ofta den engelska termen nämnas när ett nytt begrepp tas upp, eftersom engelskan är så dominerande inom detta fält och många kan vara bekanta med den engelska termen.

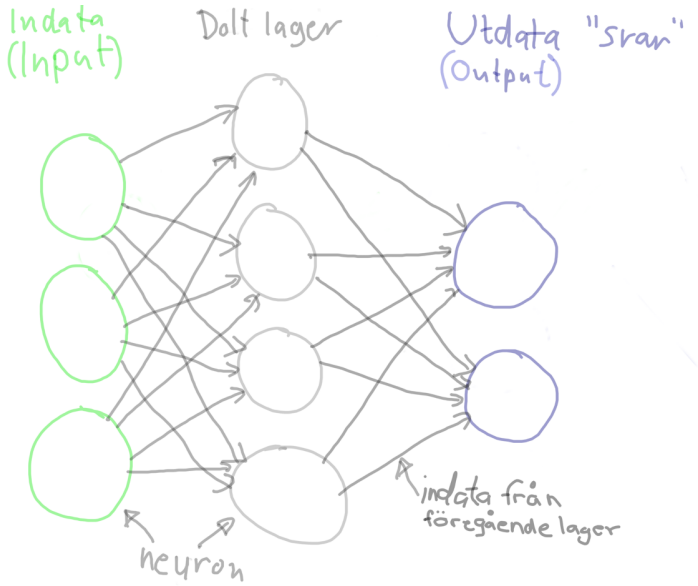
Begreppen *artificiella neuronnät* (ANN) eller bara neuronnät, *maskininlärning* och *artificiell intelligens (AI*) används i arbetet som om de vore synonymer. Men som Figur 1 visar är det AI som är det allra bredaste och minst specifika begreppet. AI-fält utvecklas fortfarande mycket och ibland kan det vara svårt att veta om man pratar om samma sak när den ena säger maskininlärning och den andra säger neuronnät. Figur 1 visar på ett bra sätt att det som kallas artificiella neuronnät också samlas under maskininlärning och att allt som tillhör maskininlärning ingår i begreppet artificiell intelligens. Ett annat begrepp som ofta förekommer i dessa sammanhang är *djupinlärning* (på engelska *deep learning*). Detta ingår som en del av neuronnät. Den term som bäst beskriver vad som används i detta arbete är djupinlärning med artificiella neuronnät. Nedan följer en kort genomgång av hur ett artificiellt neuronnät fungerar.



Figur 1. Förhållandet mellan artificiell intelligens, maskininlärning, neuronnät och djupinlärning.

2.1 Artificiellt neuronnät

Ett artificiellt neuronnät, på engelska Artificial Neural Network (ANN), är som en gigantisk matematisk funktion inspirerad av den mänskliga hjärnans neuroner. Om artificiella neuronnät verkligen fungerar likt en mänsklig hjärna spelar inte så stor roll här utan det viktiga är att de klarar att lösa de problem de är satta att lösa. Ett artificiellt neuronnät är uppbyggt av flera lager med neuroner (Figur 2). Ibland kallas neuronerna noder, celler eller enheter. I det första lagret, längst till vänster, finns indata och i det sista finns utdata eller ”svaret”. Alla lager däremellan är dolda lager och används för att göra nätverket bättre på att bestämma rätt svar.



Figur 2. Ett litet neuronnät med ett dolt lager.

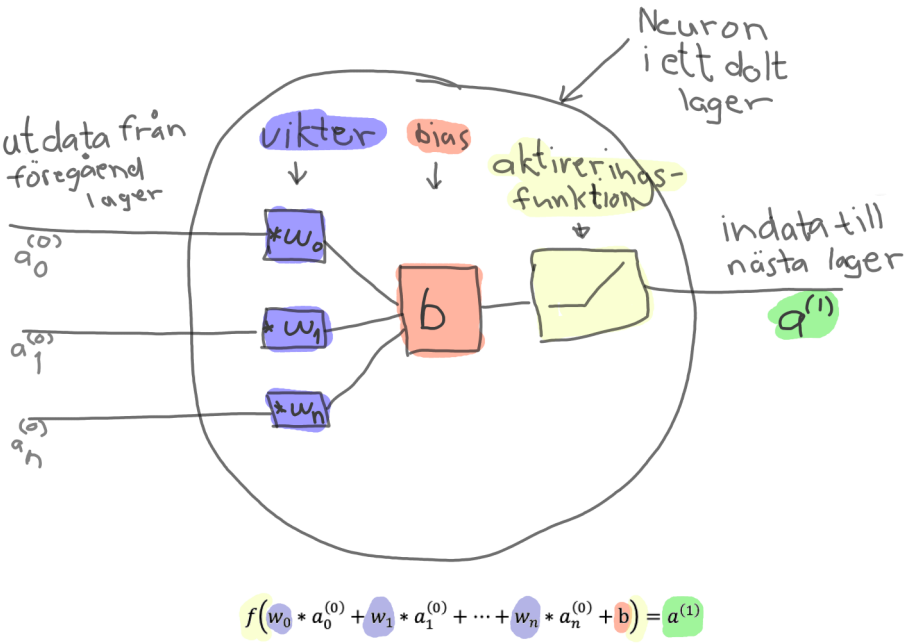
Ett artificiellt neuronnät kan till exempel hjälpa oss att avkoda handskrivna siffror. Om den handskrivna siffran finns sparad i en bild (se Figur 3) på 28x28 pixlar behövs det 784 (28*28) neuroner i indatalagret för att representera alla pixlar. Eftersom det finns 10 siffror, 0 till 9, behövs det 10 neuroner i utdatalagret för att få plats med alla 10 siffror. Värdet i varje neuron är oftast mellan 0 och 1. Innan informationen om bilderna sätts in i indatalagret omvandlas de till svartvita bilder och värdet i en neuron går från 0, helt vit pixel, till 1, helt svart pixel.

Figur 3. Handskriven siffra på en 28x28 pixlar stor yta.

För att beräkna värdet för varje neuroncell använder man sig av samtliga *utdata* från lagret före (se $a^{(0)}$ i Figur 4) samt en *bias* för den aktuella noden (se b i Figur 4). Utdata från föregående lager multipliceras med en vikt (w i Figur 4), summeras och körs genom en

funktion för att få ett värde mellan 0 och 1. Funktionen som används kallas aktiveringsfunktion. Förr användes ofta en sigmoid funktion medan man numer använder en styckvis linjär funktion (på engelska *rectified linear unit*, ReLU). Beräkningarna i en neuron ser ut som i ekvation (1) nedan. Där är $a^{(1)}$ utdata från neuronen. Detta kan vara utdata från det hela neuronnätet eller indata till nästa lager.

$$a^{(1)} = f\left(w_0 * a_0^{(0)} + w_1 * a_1^{(0)} + \cdots + w_n * a_n^{(0)} + \mathrm{b}\right) \quad (1)$$



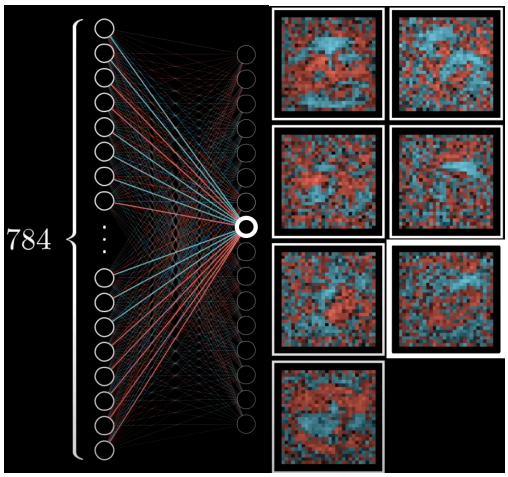
Figur 4. En översikt av vilka beräkningar som görs i en enskild neuron och vad de representerar i ekvarionen (1)

Det som har beskrivits här ovan där samtliga celler är kopplade till samtliga celler i lagren intill är ett så kallat helanslutet neuronnät (på engelska *fully connected neural network*).

För att ett neuronnät ska kunna ge korrekta svar måste det tränas med samma typ av indata som det sen ska känna igen. När nätet ger fel svar använder man olika algoritmer för att ändra vikter och bias i nätet. Detta kallas *bakåtpropagering* (på engelska *back propagation*).

2.2 Bildigenkänning med neuronnät

De artificiella neuronnät som brukar användas [7] för bildigenkänning kallas på svenska för *faltningsnätverk* [8]. Termen är kanske mer bekant på engelska: *convolutional neural network* (CNN). Den här typen av neuronnät är konstruerade för att känna igen delar av mönster eller motiv i en bild. I de första lagren av nätverket är mönsterbitarna mindre för att i senare lager bli allt större. Det är oftast svårt för människor att förstå vilka mönster det artificiella neuronnätet har hittat (se Figur 5).



Figur 5. Visar vilka pixlar i första dolda lagret som aktiveras i ett neuronnät som identifierar handskrivna siffror. [9]

I Figur 5 syns bilder som representerar aktiverade pixlar i det första dolda lagret i ett artificiellt neuronnät som känner igen handskrivna siffror. Nätet har 784 (28*28) indata neuroner, en för varje pixel. Bilderna till höger i Figur 5 verkar inte föreställa någonting och det är svårt att förstå vad neuronnätet kan använda dessa mönster till.

2.3 Speciella modeller

Det finns flera olika modeller, eller strukturer, för hur man ska bygga ett neuronnät. I alla nätverk finns samma byggstenar: lager, neuroner, vikter, bias och funktioner. Modellerna beskriver hur nätverken är uppbyggda och anger hur många lager det finns, ungefär hur många neuroner som varje lager har, hur kopplingarna mellan neuronerna ser ut, hur nätverket ska tränas och hur vikter och bias ska sättas för att uppnå önskad kvalitet.

2.3.1 AlexNet

AlexNet är klassat som ett faltningsnätverk (CNN) och är den modell som startade dagens våg av forskning med CNN-nät. Detta var också en av de första modeller som använde sig av grafikkort (GPU) i större skala, för att göra beräkningar i neuronnätverks vikter och bias. När skaparna Alex Krizhevsky, Ilya Sutskever och Geoffrey E. Hinton presenterade AlexNet 2012 rönte det stor uppmärksamhet eftersom det klarade att klassificera 1000 bilder från ImageNet [10] på ett mycket bättre sätt än vad som tidigare gjorts. Nätverket har åtta lager där de fem första är faltningslager och de tre sista är helt sammankopplade lager [11].

2.3.2 SqueezeNet

SqueezeNet skapades 2016 av forskare på University of California, Berkeley och Stanford University med intensionen att vara lika bra på att klassificera bilder som AlexNet utan att ha lika många parametrar. SqueezeNet har 50 gånger färre parametrar än AlexNet [12] med samma träffsäkerhet och dessutom en annan struktur. SqueezeNet är ett faltningsnätverk som i stora drag består av lager som först minskar parametrarna (squeeze) för att sedan expandera dem igen.

2.3.3 ResNet-18

ResNet-18 är ännu ett faltningsnätverk som är byggt för att kategorisera bilder. Som namnet indikerar har detta nät 18 lager [13, 11]. Det finns olika varianter av ResNet med ända upp till 152 lager. De olika varianterna passar olika bra för olika träningsdata. Finns det lite träningsdata brukar ett nät med färre lager väljas. ResNet står för ”residual neural network”, där residual kan översättas med *finnas kvar* eller *återstå*. I nätverket betyder det att vissa vikter (kopplingar) hoppar över ett eller flera lager, de lever kvar och påverkar nästkommande lager.

Detta gör att nätverket klarar sig bättre från överträning[1] (overfitting) än både AlexNet och SqueezeNet och kan därför tränas till en bättre klassificeringsförmåga.

2.3.4 DenseNet

DenseNet är en vidareutveckling av ResNet, där neuronerna i ett lager är kopplade med vikter till alla föregående lager [14, 4]. Denna typ av nät har visat upp ännu högre precision vid klassificering av ImageNet-bilder men tar också mer minne eftersom det hanterar så många kopplingar.

2.4 Djupinlärning i Python

De två av de vanligaste ramverken för djupinlärning (på engelska deep learning framework) i Python är PyTorch och TensorFlow. PyTorch är ett ramverk eller programvarubibliotek för maskininlärning utvecklat av Facebook (Facebook's AI Research lab). Även TensorFlow är ett ramverk för maskininlärning men detta utvecklas av Google (Google Brain). Båda kan användas i Python. TensorFlow kom först 2015 [15] och blev mycket populärt. PyTorch kom året därpå 2016. Det har varit mycket debatt om vilket som är det bästa ramverket att välja [16]. TensorFlow har varit mer populärt på grund av sin bättre prestanda medan PyTorch har varit lättare att använda och felsöka i. Nu 2021 har PyTorch gått om TensorFlow i popularitet, framförallt bland de som bedriver forskning inom maskininlärning [17].

2.5 Hårdvara

Grafikprocessorer (GPU) har visat sig vara snabbare är vanliga processorer (CPU) på att utföra uppgifter som är av maskininlärningstyp [18]. Det är på grund av att GPU:erna är bättre på matrismultiplikation eftersom de har fler kärnor. En GPU kan använda större bandbredd än en CPU och den klarar även av att exekvera fler uppgifter samtidigt än en CPU. En GPU kan jämföras med en lastbil och en CPU med en Ferrari. Om det ska fraktas många paket från A till B är lastbilen snabbare eftersom den inte behöver åka lika många gånger.

3 Metod och genomförande

I detta kapitel redogörs det för material och tillvägagångsätt som använts under arbetet.

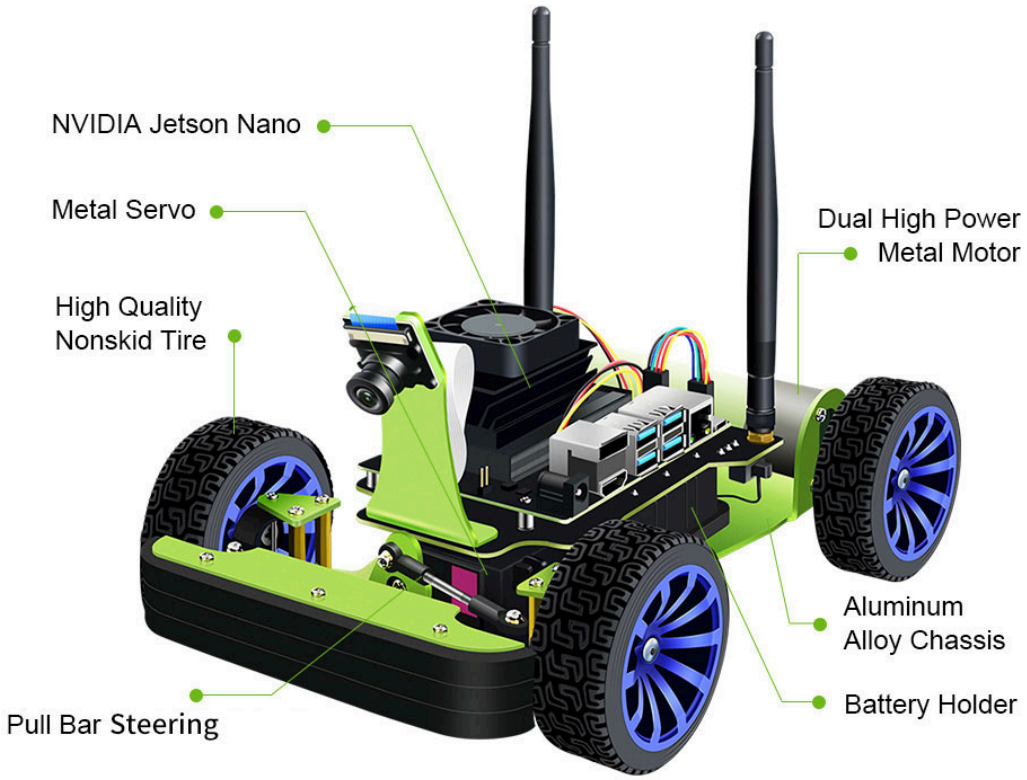
3.1 Material

Till material hör både hårdvara och mjukvara.

3.1.1 Hårdvara

Hårdvaran som användes i detta projekt kallas JetRacer (Figur 6) och är inköpt på HiTechChain.se [19]. Det är en liten bil där en NVIDIA Jetson Nano-enkortsdator finns monterad ombord och gör de beräkningar som behöver utföras. Bilen har en kamera längst fram för att kunna skicka bildinformation till datorn. Datorn processar informationen och sänder en styrsignal till styrservon. Alla chassits komponenter finns listade i Bilaga 1.

[1] Detta kan också beskrivas som att nätet lär sig saker utantill, vilket man inte vill att det ska göra.



Figur 6. Figuren visar JetRacer:ns olika delar. [1]

Teknisk specifikation för NVIDIA Jetson Nano	
CPU	Quad-core ARM A57 @ 1.43 GHz
GPU	128-core NVIDIA Maxwell
Memory	4 GB 64-bit LPDDR4 @ 25.6 GB/s
Storage	64GB microSD card support
Display	HDMI and Display Port or HDMI

Teknisk specifikation för övriga JetRacer-delar	
Camera	8MP, 160° FOV, 3280x2464 Re solution
Motor	37-520 motor, 1 :10 reduction rate, 12V voltage, 74ORPM idle speed
Servo	MG996R, 9kg/cm (4.8V) torque
Power	rechargeable 18650 batteries
Display	0.91inch OLED,128x32 pixels
WiFi	Intel 8265AC, 802.11ac
Bluetooth	Intel 8265AC, version 4.2

Tabell 1 – En tabell över hårdvaran i projektet.

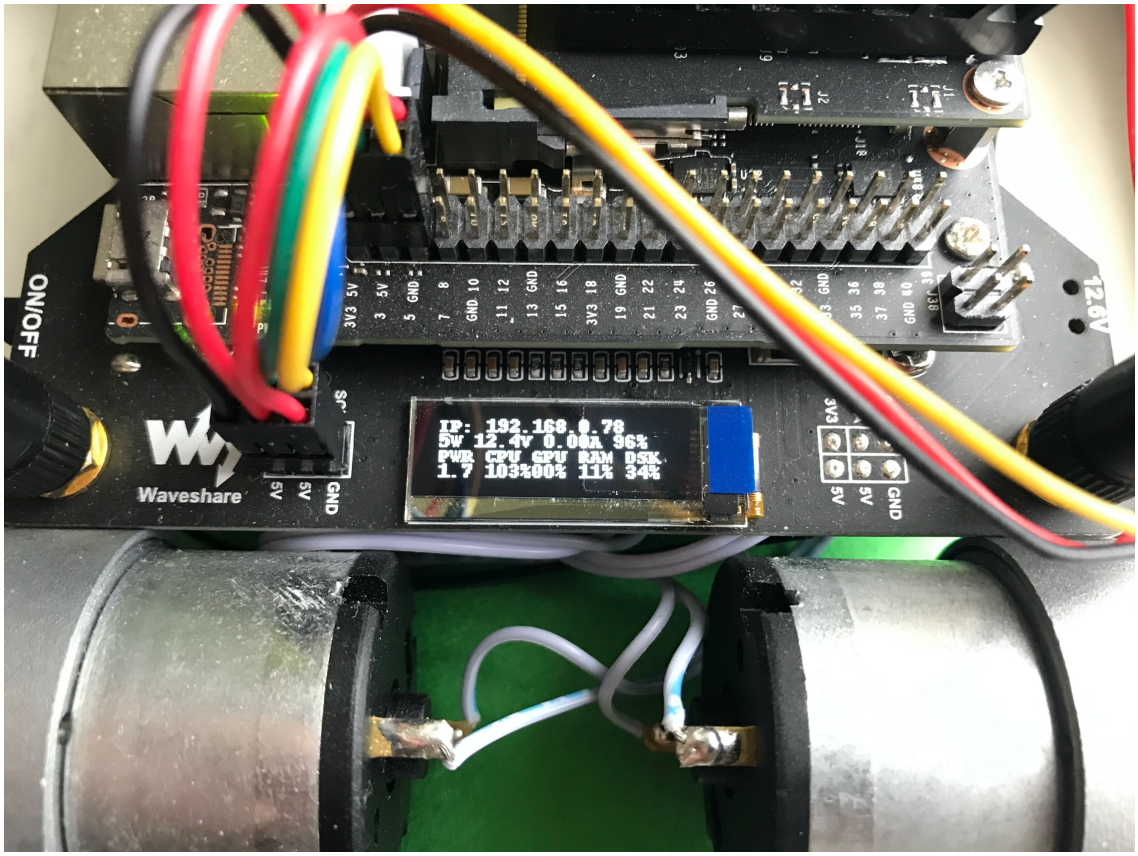
Inkluderat i paketet var ett microSD-kort på 32GB men det är i minsta laget. För att få tillräckligt med utrymme under utvecklingen användes ett 64GB-kort istället.

För att kunna göra alla beräkningar för neuronnätet på fordonet självt behöver datorn ombord vara relativt kraftfull. NVIDIAs Jetson Nano är den första enkortsdator som klarar av mängden beräkningar som behövs. Jämfört med en av de mest populära enkortsdatorerna Raspberry Pi 4 har Nano:n ett mycket kraftfullare grafikkort på 128 kärnor, medan Raspberry Pi 4:s grafikkort bara har fyra kärnor. Vid beräkningar relaterade till maskininlärning är grafikkortet viktigast, vilket visades i teoriavsnittet. Utan Jetson Nano hade detta projekt inte

varit möjligt. Därför föll valet av enkortsdator på en sådan. Den fläkt som monterades på Jetson Nano:s kylfläns hade en 3-pin-anslutning. Mjukvaran i Jetson Nano kan inte reglera en 3-pins-fläkt vilket gör att den körs på maxfart hela tiden. Därför vore en 4-pins-fläkt att rekommendera för andra liknande projekt.

Chassit (se Bilaga 1) byggdes samman med enkortsdatorn och testades så att de fungerade väl tillsammans. Saker som kontrollerades var t.ex. om bilen får tillräcklig spänning. Det finns två effektlägen som bilen kan köra på: 5W och MAX. Vilket av de två lägena som bilen befinner sig i syns på den lilla displayen på bilens ovansida (se Figur 7). Bilen kan inte köra på effektläge MAX när den endast är kopplad till batteriet eftersom detta inte själv kan leverera tillräckligt stor effekt. Om adaptern på 12,6 V är inkopplad kan effektläge MAX användas. Inkopplingen för adaptern syns till höger i Figur 7.

Det kontrollerades även att hjulen rullade lätt. De främre hjulen har en glidkoppling och skruvarna som håller fast dem behövde justeras mycket noggrant. Satt de för hårt rullade hjulen trögt och satt de för löst började hjulen röra sig i sidled. De främre hjulen är också i allra högsta grad inblandade i styrningen. De måste stå rätt i förhållande till varandra. Båda ska peka framåt samtidigt och i förhållande till styrservon. De måste peka framåt när styrservon är i neutralläge. Om man är van vid vanliga bilar pratar man om hjulinställningen. Mer om lösningen på detta problem redogörs för i resultatdelen.



Figur 7. Den lilla displayen på bilens ovansida visar att bilen är i 5-watts-lägen (5W).

Till den bana som bilen navigerade runt användes en grå plastduk med en vitstreckad mittlinje och heldragna gula linjer på sidorna (se Figur 8).

Figur 8. Matta med den markerade banan som bilen navigerar efter.

För att enklare kunna köra omkring med bilen på mattan och samla in bilder konfigurerades en PlayStation 4-kontroll för att kunna styra bilen. Kontrollen användes för att köra runt banan och samtidigt aktivera bilens kamera från datorn, istället för att behöva gå fram och tillbaka mellan bil och dator för att samla in bilder från banan.

3.1.2 Mjukvara

Mjukvaran som användes med enkortsdatorn Jetson Nano laddas först in på ett microSD-kort från datorn. Detta kort sätts sedan in i enkortsdatorn, sedan startats den upp och konfigureringen fortsätter. Det mjukvara som laddas in på SD-kortet var en variant av NVIDIAs officiella *software development kit (SDK)* som heter JetPack SDK [20]. Detta SDK är i grunden Ubuntu 20.04 LTS där NVIDIA har lagt till firmware för att Nano:n ska starta, egenskrivna drivrutiner för grafikkretsarna och en utvecklingsmiljö med flera exempel.

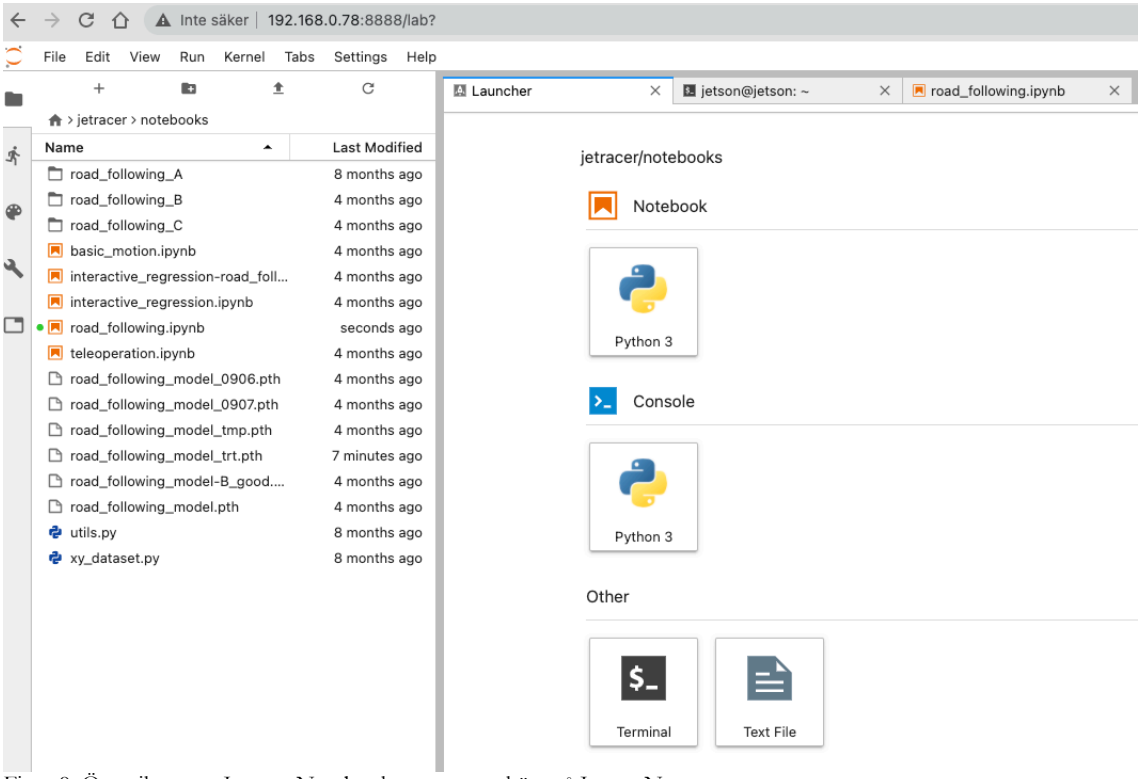
Installationen av programvaran finns till största del beskriven på waveshare.com [21]. Den installerar bl.a. flera GitHub projekt:

- JetCard: https://github.com/waveshare/jetcard
  Innehåller:
  - Jupyter Notebook-server som startar när Jetson Nano:n slås på
  - Script för att visa Jetson Nano:ns IP
  - PyTorch och TensorFlow som nätverk för djupinlärning
- JetRacer: https://github.com/NVIDIA-AI-IOT/jetracer
  Innehåller:
  - Jupyter Notebooks exempel för att komma igång

En detaljerad information av vad som installerades finns i Bilaga 2.

Jupyter Notebook är en del av Project Jupyter som tidigare hette IPython. Ett Jupyter Notebook-dokument innehåller olika celler med körbar kod och beskrivande text om vartannat. Dokumentet ”hostas” av en Jupyter Notebook server och visas direkt i en webbläsare (se Figur 9). Python-koden kan köras genom att exekvera en eller flera celler i taget. Detta format gör det enkelt att dela både kod och text på ett interaktivt sätt. I de GitHub-repo som laddats ner i installationsfasen finns flera Notebook-filer (.ipynd-filer) som

underlättar att komma igång med programmeringen av JetRacer-bilen. Det går även att få en vanlig terminal (shell prompt) till bilen från webbgränssnittet, så man enkelt kan exekvera shell-kommandon.



Figur 9. Översikt av en Jupyter Notebook server som körs på Jetson Nano

För att styra bilden användes alltså Python-kod som kördes genom webbläsaren. Det var Facebooks öppna ramverk för maskininlärning PyTorch som användes för konstruera och träna de artificiella neuronnäten.

3.2 Anslutningen till bilen

Anslutningen till bilen gjordes på flera olika sätt. Det går att koppla Jetson Nano:n direkt till en skärm via en HDMI-kabel och ansluta tangentbord och mus till USB-portarna. Då kan man använda den som om det vore en vanlig dator med Ubuntu som operativsystem. Detta var bra att kunna göra under det initiala skedet för att få igång nätverket och installera ovannämnda GitHub-repo.

När installationen var gjord och nätverket fungerade blev det bara krångligt att ansluta en HDMI-kabel för att styra bilen. Då användes antingen en SSH-anslutning eller inloggning via en Jupyter Notebook server. SSH-anslutningen gjordes till port 22 på bilens IP-adress. IP-adressen kan läsas av på den lilla LCD-display som sitter på bilens ovansida (se Figur 7). Anslutningen till Jupyter Notebook servern gjordes genom att logga in via en webbläsare på bilens IP-adress och port 8888: http://jetson-ip-address:8888/. Det var primärt denna styrning via Jupyter Notebook som användes genom arbetet.

Ännu ett sätt att ansluta till Nano:n är genom att ansluta den till en dator via micro-USB-porten. Då finns Jupyter-servern tillgänglig via ett USB-nätverk som skapas automatiskt och det går att ansluta på adressen http://192.168.55.1:8888

3.3 Inställningar av mjukvaran

För att få Jetson Nano:n att öka prestandan ytterligare gjordes några fler mjukvaruinställningar.

Det var onödigt att köra en grafisk skrivbordsmiljö på Nano:n när det inte fanns någon som tittade på den. Att köra det grafiska gränssnittet belastar bilens CPU, GPU och internminne i onödan. Då blir det mindre kapacitet kvar till att köra maskininlärningsalgoritmen som bestämmer hur bilen ska svänga. Det grafiska gränssnittet stängdes av med följande kod:

```
# To disable GUI on boot, run:
$ sudo systemctl set-default multi-user.target

# To enable GUI again, issue the command:
$ sudo systemctl set-default graphical.target

# To start Gnome session on a system without a current GUI just execute:
$ sudo systemctl start gdm3.service
```

Nano:n kan köras på två olika lägen som drar olika mycket effekt. De kallas MAXN och 5W och kan också avläsas på LCD-displayen som sitter på bilen (se Figur 7). I läget MAXN drar datorn 10 watt, processorns alla fyra kärnor körs och både CPU och GPU går på maximal klockfrekvens [22]. I 5W-läget drar datorn bara 5W, CPU och GPU klockas ner och bara två av processorns fyra kärnor används. När bilen går på batteri finns det inte tillräckligt med strömförsörjning för att köra på MAXN. Om man ändå försöker att köra på MAXN när endast batteriet är inkopplat kan bilen helt plötsligt stoppa och belastningen blir för hög. Under projektets gång växlades det mellan MAXN och 5W så fort bilen var inkopplad till den fasta strömkällan. Det går nämligen mycket snabbare att träna neuronnäten när MAXN-läget används. Så här växlar man mellan de olika lägena:

```
# set power mode to MAXN
$ sudo nvpmodel -m0

# set power mode to MAXN
$ sudo nvpmodel -m1

# get current power mode
$ sudo nvpmodel -q
```

Ännu ett sätt att öka prestandan är att stänga av operativsystemets loggning. Om man inte läser loggarna kan tjänsten rsyslog stoppas med:

```
$ sudo systemctl stop rsyslog.service

#This will only stop rsyslog for the current session, to make sure rsyslog
doesnt start on reboot, use:
$ sudo systemctl disable rsyslog.service
```

3.4 Styrning

Detta stycke handlar som styrning som inte sköts av ett artificiellt neuronnät utan av direkta kommandon från användaren.

3.4.1 Manuell styrning

För att testa bilens styrning och rörelse kunde Python-kod användas för att se om allt fungerade som det skulle. Detta gjordes med den kod som visas nedan, där den slutliga styrningen beräknades som (car.steering_gain * car.steering + car.steering_offset). För att få bilen att köra rakt när car.steering = 0 ändrar man på car.steering_offset och för att få den att svänga lagom mycket ändrar man på car.steering_gain.

Gaspådraget ändras genom att sätta ett värde på car.throttle. Det gick också att ändra hur snabbt gasen skulle ändras genom att ändra car.throttle_gain. Det slutliga gaspådraget räknas ut som (car.throttle * car.throttle_gain)

```
from jetracer.nvidia_racecar import NvidiaRacecar

car = NvidiaRacecar()

# Styrning
car.steering = 0.9
car.steering_gain = 1.50
car.steering_offset = 0.15

# Gaspådrag
car.throttle = 0
car.throttle_gain = 0.8
```

3.4.2 Styrning med en PS4-kontroll

För att underlätta insamlandet av bilder i senare steg konfigurerades bilen att fungera med en Playstation 4-kontroll. Det finns ett HTML5 Gamepad API som är gjort för att webbläsare ska kunna ta emot input från spelkontroller. Först testades att kontrollen fungerade tillsammans med webbläsaren. Detta gjordes på gamepad-tester.com.

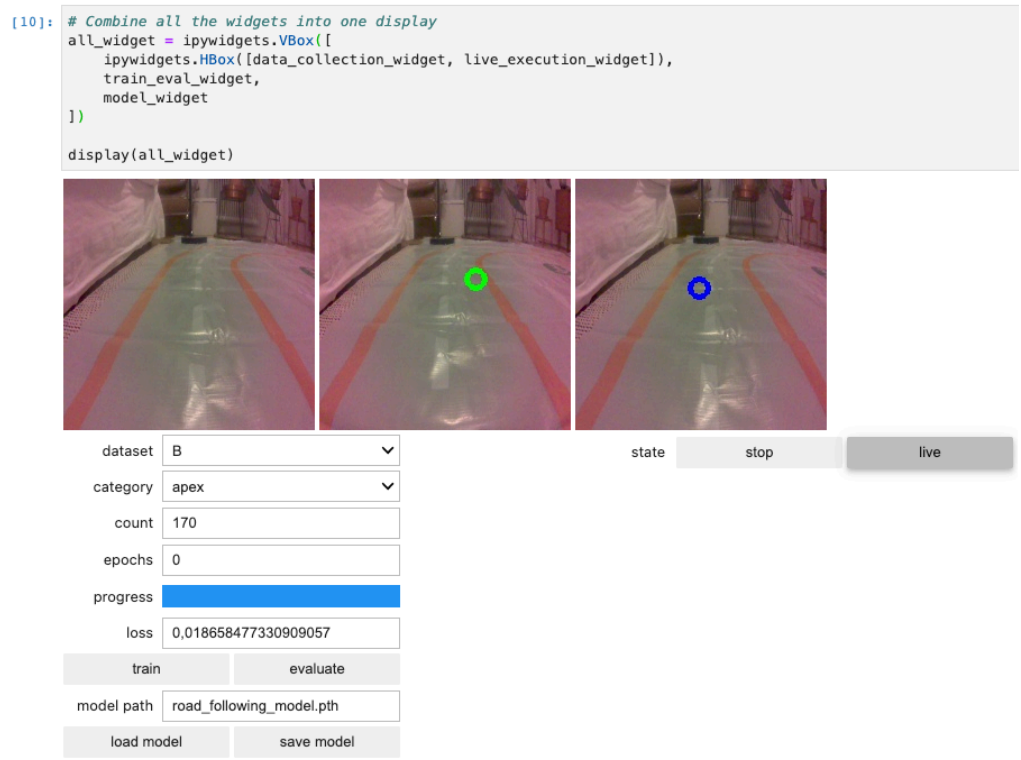
Med hjälp av en Notebook mappades spelkontrollens styrknappar till bilens car.steering och car.throttle. Då kunde JetRacer-bilen kontrolleras med PS4-kontrollen inkopplad till en bärbar dator via en Jupyter Notebook.

3.5 Indata

Insamling av indata till det artificiella neuronnätet är mycket viktigt om man ska få välfungerande utdata. Det var tydligt i detta arbete då bilen hade mycket svårt att ta sig runt banan om bilderna och informationen om hur bilen skulle styra inte var bra. Innan det gick att träna ett neuronnät var man tvungen att ha ca 20 bilder för att träningen skulle bli acceptabel. Efter det var insamlingen av data och ny träning av neuronnätet tätt integrerad.

Under insamlingen användes båda PS4-kontrollen och manuell utplacering för att få tillräckligt många bra bilder i datasetet. Den vy i Jupyter Notebook som användes för att spara bilderna hade tre olika foton (benämns widgets i koden) från bilens kamera (se Figur 10). Den längst till vänster var en direktsändning av vad kameran hade framför sig.

Mittenbilden visar den senaste lagrade bilden med en grön ring. Den gröna ringen är information från den senast tagna bilden om vart bilen ska röra sig i just den specifika bilden. När ca 20 bilder hade samlats in kunde neuronnätet tränas. Efter att nätet tränats visades den tredje bilden i webbgränssnittet. Detta var också en direktsändning av det som kameran visade men med en blå cirkel som visade vart bilen skulle åka, baserat på träningen av det artificiella neuronnätet. Med andra ord visar den blå pricken utdata eller beslut från neuronnätet.



Figur 10. Vy över datainsamling och träning av neuronnätet

Processen med att samla in data var sådan att bilen flyttades till en önskvärd position, därefter kunde man klicka på bilden längs till vänster i Figur 11 dit man ville att bilen skulle köra. Då sparade bilden på disk och den mittersta bilden med en grön ring uppdaterades med den nyligen sparade bilden. Sen flyttades bilen till en ny position, där körriktningen återigen bestämdes genom att klicka på den vänstra bilden.

Den insamlade data, indata för träningen av systemet, sparades som JPG-bilder i storleken 224x224 pixlar och med pixelkoordinater i bildens namn. Koordinaterna i filens namn visar i vilken riktning bilen ska styra i just den bilden. Det filnamn som börjar med 149_107_... betyder att bilen ska styra mot pixel 149 i x-led och 107 i y-led, den blå cirkeln i Figur 11.

Figur 11. Filnamnet 149_107_...jpg innehåller information om i vilken riktning bilen ska köra.

En tanke med insamlingen av bilder var att placera mattan med banan på olika ställen för att neuronnätet skulle ”lära sig” att bakgrunden inte hade någon betydelse. Oavsett var mattan låg skulle bilen följa vägen. Det lyckades båda bra och mindre bra. Bilen klarade av att ta sig runt banan både i hemmet och på Knightecs kontor med samma träningsdata, så länge inte solen lyste. Det syns på de bilder som är tagna när solen lyser att ljus och färger blir väldigt annorlunda. Så fort solen började lysa eller då mattan placerades nära ett fönster fick bilen problem att navigera. Mer om solskensproblemen och kvaliteten på indata i resultatdelen.

3.6 Träning av neuronnät

Träningen var relativt integrerad med insamlingen av data. Därför finns information om träningen av det artificiella neuronnätet även i stycket innan om indata.

Den algoritm som användes för att skapa neuronnätet var ResNet-18 och som tidigare nämnts gjordes detta med hjälp av ramverket PyTorch i programspråket Python. ResNet valdes efter en snabb utvärdering av bilens förmåga att navigera jämfört med algoritmerna AlexNet, SqueezeNet och DenseNet 121 men här finns det utrymme för att göra mer jämförelser i framtida arbeten. ResNet är ett faltningsnätverk (CNN) som har en god förmåga att t.ex. identifiera föremål

3.7 Testkörning

När träningen av bilens neuronnät var klar kom det mest spännande momentet: Testkörning av banan på egen hand.

I Python-koden (se nedan) sattes styrinställningarna för att bilen skulle köra rakt på raksträckan och svänga lagom mycket i svängarna. Det var svårt att få dessa värden att bli bra. Om bilen svängde för lite i kurvan var det svårt att exakt veta varför. Antingen var det för lågt värde på ”steering_gain” eller så hade neuronnätet inte tränats att svänga tillräckligt mycket. När detta problem uppstod fick man hoppa fram och tillbaka mellan steering_gain och olika träningsdata för neuronnätet för att hitta bra inställningar. När man en gång har hittat ett bra värde för styrningen visste man nästa gång att det var neuronnätet som var problemet nästa gång.

Bilen hade gaspådraget 1 (car.throttle = 1) hela tiden så länge som den körde och den körde så länge som for-loopen i koden exekverade. Genom att ändra range(500) till ett mindre eller större värde kunde längden på körtiden minskas eller ökas.

```
from utils import preprocess
from tqdm.notebook import tqdm
```

```
import numpy as np

# Same effect as car.steering_gain
STEERING_GAIN = 1.50

# Same effect as car.steering_offset
STEERING_BIAS = 0.14

# Starta bilen, gaspådrag
car.throttle = 1

# x i range(x) för att ändra hur länge bilen körs
for i in tqdm(range(500)):
    image = camera.read()
    image = preprocess(image).half()
    output = model_trt(image).detach().cpu().numpy().flatten()
    x = float(output[0])
    car.steering = -x * STEERING_GAIN + STEERING_BIAS

print("loop ended")

# Stoppa bilen
car.throttle = 0
```

4 Resultat

Under detta projekt har en självkörande bil byggts. Med hjälp av Jetson Nano-enkortsdator, maskininlärningsalgoritmer och en kamera riktad i bilens åkriktning har den navigerat runt en markerad bana.

4.1 Montering

Monteringen gjordes enligt beskrivningen i bilaga 1. Två större problem uppstod under monteringen av bilen som båda hade med styrningen att göra. Det första problemet hade med hjulinställningen att göra (punkt 6 och 7 i bilaga 1). Det finns något som kallas ”long pull bar” och som är ett stag mellan hjulen (se Figur 12). Staget har gängor i båda ändar där det sitter två kulleder som kan justeras oberoende av varandra. Detta stag måste ha precis rätt längd för att båda hjulen ska vara riktade framåt samtidigt. Om staget är för långt kommer hjulen att peka utåt i neutralläget och om staget är för kort kommer staget att peka inåt.

Figur 12. Hjulinställningar. [1]

Det andra problemet var med det korta styrstaget (se Figur 12. När servohjulet är i neutralt läge ska bilen köra rakt fram. Det förutsätter att det korta styrstaget varken är för långt eller för kort. Om staget är för långt svänger bilen åt vänster och om det är för kort svänger bilen åt höger.

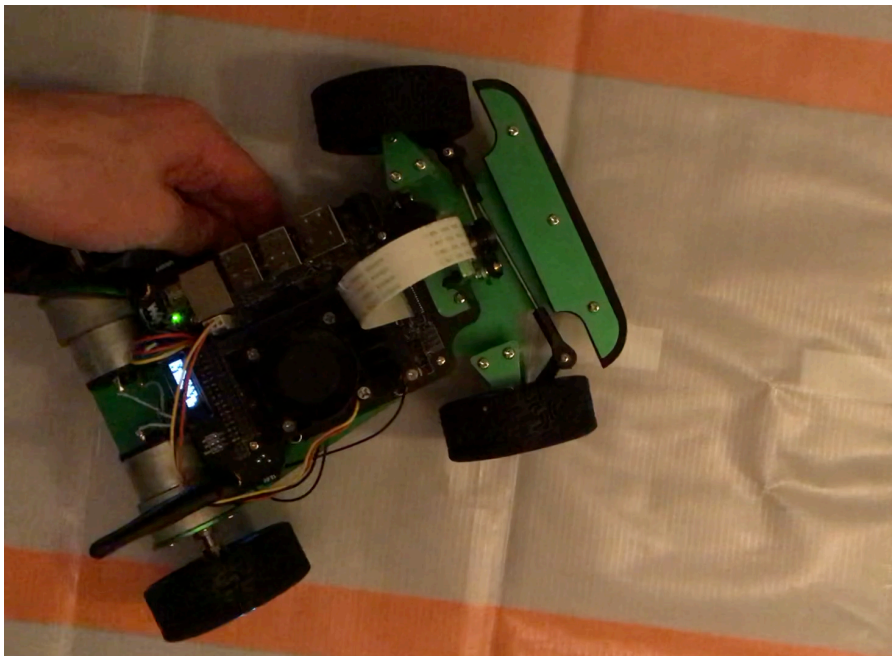
Dessa två problem såg ut att enkelt kunna lösas genom att skruva på de plastbitar som bildar kulleder men det gick inte att helt se hur det färdiga resultatet skulle bli innan bilen var färdigmonterad. Det gick inte heller att justera längderna på stagen när bilen var färdigmonterad. Detta resulterade i att framdelen av bilen behövde monteras samman och isär flera gånger för att kunna justera de båda stagen till rätt längd.

4.2 Navigering

Huvudmålet med detta arbete var att få bilen att köra runt den markerade banan. Detta mål uppfylldes. Bilen kunde köra flera varv på banan utan någon påverkan utifrån. Den följde oftast en liknande bana som den hade haft varven innan. Om bilen var nära den orangea kanten på ett ställe längs varvet (se den röda ringen i Figur 13) körde den nära kanten på samma ställe även nästa varv. Detta kunde oftast lösas genom att utföra mer träning på just det partiet, men det var inte alla gånger mer träning gav bättre resultat. Mer träning kunde också försämra styrningen.

Figur 13. Bilen var ofta när kanten på samma ställe, varv efter varv. [1]

När bilen hölls precis över mattan och roterades kunde man se framhjulen svänga åt det håll som vägen gick (se Figur 14).



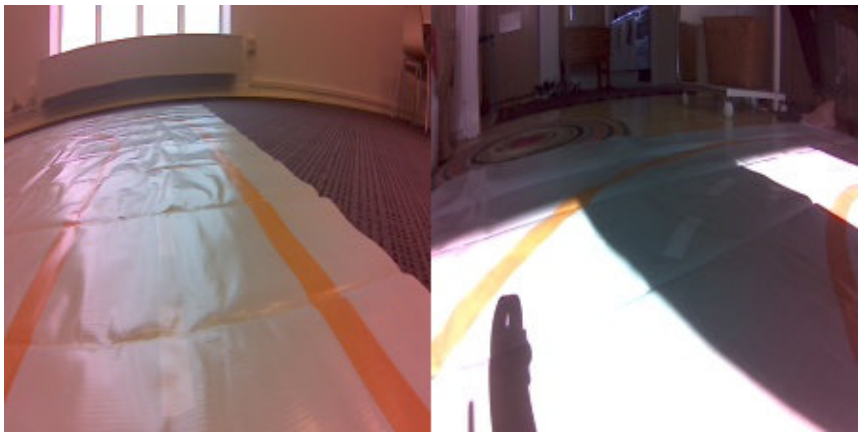
Figur 14. Framhjulen svänger i vägens riktning när bilen hålls 1 cm över mattan och roteras.

All träning av bilen gjordes när den körde medursvarv. För att säkerställa att bilen inte bara lärt sig det hållet av banan utantill vändes den om och testkördes åt andra hållet. På första försöket klarade den av att styra längs banan i motursvarv. Detta visar att det verkligen är maskininlärning och inte utantillinlärning.

Träningen fick bäst resultat på bilens styrförmåga då det fanns upp emot 100 bra bilder och nätverket tränades i 10 rundor (epochs) med bilderna. Styrningen kanske hade blivit ännu bättre med fler bilder i träningsdan och med mer träning men det började ta lång tid att genomföra träningen på Jetson Nano:n. Det finns en möjlighet att testa att träna neuronnätet på en annan mer kraftfull dator och sedan föra tillbaka resultatet till Nano:n.

4.3 Olika platser

En god kvalitet på indata till neuronnätet visade sig vara av största vikt. Om solen lyste och ljuset reflekterades i mattan eller skapade för starka skuggor hade bilen mycket svårt att följa banan (se Figur 13). Det kanske inte är så konstigt, om man jämför bilderna i Figur 15. I bilden till höger är det nästan ingen skillnad alls på mattans färger medan det i bilden till vänster är stora skillnader mellan den upplysta delen av banan och den del som ligger i skuggan.


Figur 15. Bilen hade svårt att följa banan när det var soligt. På kontoret (till vänster) och i lägenheten (till höger).

Om mattan flyttades mellan olika platser klarade bilen av att köra utan att behöva tränas förutsatt att ungefär samma ljusförhållanden rådde. När bilen tränades på mattan i hemmet och sedan flyttades till Knightecs kontor klarade den av att navigera längs banan utan att bakgrunden ändrade styrningen.

Mer indata förbättrade dock inte alltid navigeringen. När bilen hamnat på fel kurs och endast delar av banan syntes i bild hade den svårt att hitta tillbaka till banan. För att hjälpa bilen tillbaka i rätt riktning tränades neuronnätet med bilder där endast en begränsad del av vägbanan fanns i kamerans synfält (se Figur 16). Om bilden såg ut som nedan borde bilen svänga kraftigt åt höger för att komma tillbaka på banan igen. Men efter att några bilder av denna typ hade lagts till i träningsdatan fick bilen problem att köra när den redan hade en bra position. Den svängde då ibland in på innerplanen av mattan trots att den var på rätt väg. Det var som att den långa orangea linjen i högerkanten av bilden berättade att den skulle svänga höger, precis som i Figur 16, trots att den andra orangea linjen också var i bild.


Figur 16. Indata som försämrade navigeringen.

5 Diskussion

Bilen klarade av att styra sig själv runt den utstakade banan, vilket var målet med projektet. Om man ska vidareutveckla den här självkörande bilen finns det flera olika möjligheter. För att undersöka om det finns något annat neuronnät än ResNet18 kan man göra olika

prestandatester mellan ResNet-18 och flera andra neuronnät. Det går att testa både med avseende på snabbhet och med avseende på korrekthet i navigationen, att till exempel byta det neuronnät som används och ha kvar samma indata för att se hur utdata förändras och hur bra bilen då klarar av att styra.

I denna undersökning kördes bilen hela tiden med samma hastighet runt banan och det var endast styrningen som ändrades. Här finns det flera förbättringsmöjligheter. Bilens förmågor skulle kunna utökas med möjligheten att identifiera olika objekt som dyker upp i bilens kamera. Om det är något som ligger i vägen skulle bilen kunna väja undan eller bromsa in beroende på hur objektet befinner sig i förhållande till bilen. För att göra bättre beslut om vad som ska göras kan objekten identifieras med hjälp av ett annat neuronnät. Om bilen kan klara av att bromsa måste gaspådraget kunna ändras och inte vara konstant, som det var under dessa försök.

Det finns också möjlighet att utvärdera hur indata ska väljas för att bilen ska prestera så bra som möjligt. Undersökningen av indata i denna rapport visar att det är väldigt viktigt att den är bra. Men vad är bra indata? Det går att undersöka om det finns något bra sätt att välja indata för att få bilen att klara av olika ljusförhållanden på ett bättre sätt. Bättre indata i kombination med en bättre förmåga att identifiera objekt skulle kunna användas för att få bilen att registrera om den kör utanför banan. Som det ser ut nu fortsätter bilen med samma hastighet även utanför banan. Om bilen märkte att den var utanför banan skulle den kunna stoppa eller till och med börja backa för att komma tillbaka in på banan.

5.1 Etik

Det finns många etiska frågor att beakta gällande artificiell intelligens. Frågor som rör utvecklingen av AI är t.ex. vad som händer med människan om våra AI-maskiner blir smartare än oss själva, den så kallade teknologiska singulariteten och om artificiella intelligenser bör ha egna rättigheter. Dessa frågor är än så länge till synes långt borta men eftersom långt utvecklad artificiell intelligens skulle ha stor påverkan på samhället kan det vara bra att fundera över sådana frågor redan nu.

5.1.1 Ansvar

Ett etiskt problem som redan nu är aktuellt är vem det är som har ansvar om ett fordon skadar människor eller egendom. Är det föraren som sitter i bilen, företaget som gjort programvaran eller kanske till och med den enskilde programmeraren? I mars 2018 blev en kvinna i Arizona, USA påkörd och dödad av en självkörande bil [23]. Bilen var en Volvo XC90 men det var Ubers programvara som kontrollerade bilen. Innan kollisionen hade bilen inte varnat föraren i tid eller börjat bromsa trots att den registrerat ett föremål på vägen. Detta berodde på att kvinnan hade korsat vägen där det inte fanns något övergångsställe. Senare har Uber blivit friade från ansvar för kollisionen. Vi kommer troligtvis få vara med om fler sådana här fall i framtiden när självkörande bilar blir fler på gatorna. Frågan om vem som har ansvar för en självkörande bil är inte avgjord utan kommer att fortsätta debatteras.

5.1.2 Kommunikation med neuronnät

En fråga gällande artificiell intelligens som det inte pratats så mycket om men som jag anser bör lyftas upp är: Hur kommunicerar vi människor med en artificiell intelligens? Vi människor är nyfikna och har ett behov att förstå oss på hur saker och ting fungerar. I nuläget är det knappt så att programmeraren förstår sig på hur ett neuronnät tar sina beslut. Man diskuterar indata, utdata, träning och uppbyggnad av neuronnäten, men exakt vilka pixlar

som ger upphov till ett beslut är svårt att veta. Många kan säkert uppfatta det som obehagligt att en dator utan en intuitiv moraluppfattning tar viktiga beslut för oss i trafiken. Just nu är artificiella neuronnät som svarta boxar för slutanvändaren. Kan vi göra något för att förändra det?

5.2 Hållbarhet

Hållbarhet vilar på tre ben: social, ekonomisk och miljömässig hållbarhet [24]. Även här kan neuronnät (eller AI) ha stor påverkan. Lastbilar utan förare skulle kunna köra mer miljövänligt än mänskliga förare. De skulle kunna koppla sig samman med andra AI-förare och köra konvojkörning. Då kan lastbilarna ligga tätt efter varandra och på så sätt få mindre luftmotstånd och därmed minskad bränsleförbrukning. Transporterna skulle också kunna göras när det är som minst trafik på vägarna, för att minska köbildning.

Men hur kommer detta påverka vårt samhälle på det sociala planet? Att dagligen utföra ett arbete som någon annan behöver är viktigt för att skapa mening i tillvaron [25]. Vad händer om AI-maskiner allt mer tar över våra jobb? Detta är svåra frågor som inte kan lösas med teknik av hård- och mjukvaruutvecklare utan kräver att också samhället och politikerna är involverade.

6 Slutsats

Syftet med den här studien var att konstruera en bil i leksaksformat styrd av neuronnät med grundläggande självkörande funktioner. Resultaten från studien har visat att det går utmärkt att konstruera en autonomt självkörande bil som navigerar runt en markerad bana. En viktig lärdom som går att dra från resultaten är att man måste vara mycket noggrann med den indata som väljs. En annan sak att ta med sig från det här arbetet är att man bör hitta sätt att testa neuronnätets navigeringsförmåga på data som det inte tränat på, vilket här gjordes genom att vända på bilen och köra den åt motsatt håll. Studien visar alltså på möjligheten för Knightec att minska kostnader och reparationer som skulle uppstå vid testning med fullskaliga fordon.

Metoden att skapa självkörande fordon med hjälp av maskininlärning skulle även kunna användas för att utöka bilens förmågor att navigera, t.ex. att kunna väja eller bromsa för hinder. Det skulle därför vara intressant att följa en fortsatt utveckling av sådana förmågor hos bilen.

Ett annat intressant spår skulle vara att bevaka utbudet av ny hårdvara för maskininlärning eftersom utvecklingen går snabbt inom fältet. Detta för att uppgradera NVIDIAs Jetson Nano för att få mer beräkningskapacitet och därmed kunna använda andra typer av artificiella neuronnät.

Sammanfattningsvis utgör det här arbetet ett startskott på Knightecs utveckling av mindre autonoma fordon för testning. Det pågår redan fortsatt arbete av andra studenter med att utöka bilens navigeringsfunktioner. Framöver kommer vi sannolikt att se mycket av AI:ns potential förverkligas.

Litteraturförteckning

[1] Waveshare, ”JetRacer AI Kit, AI Racing Robot Powered by Jetson Nano,” 26 Mars 2021. [Online]. Available: https://www.waveshare.com/jetracer-ai-kit.htm.

[2] Google, "WAYMO," 25 Februari 2021. [Online]. Available: https://waymo.com.

[3] The Guardian, ”Apple plans self-driving car 'in 2024 with next-level battery technology',” 22 December 2020. [Online]. Available: https://www.theguardian.com/technology/2020/dec/22/apple-plans-self-driving-car-in-2024-with-next-level-battery-technology.

[4] Chalmers, ”Självkörande buss nu i linjetrafik,” 22 Januari 2021. [Online]. Available: https://www.chalmers.se/sv/styrkeomraden/transport/nyheter/Sidor/Sjalvkorande-buss-i-linjetrafik.aspx.

[5] NVIDIA, ”AUTONOMOUS MACHINES,” April 2021. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/.

[6] Project Jupyter, April 2021. [Online]. Available: https://jupyter.org.

[7] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* pp. 2278-2324, November 1998.

[8] Computer Sweden; IDG, ”Ord och uttryck i it-branschen,” 12 September 2017. [Online]. Available: https://it-ord.idg.se/ord/faltningsnatverk/.

[9] G. Sanderson, ”Gradient descent, how neural networks learn | Deep learning, chapter 2,” 16 Oktober 2017. [Online]. Available: https://youtu.be/IHZwWFHWa-w?t=841.

[10] ImageNet, 2012. [Online]. Available: http://www.image-net.org.

[11] A. Krizhevsky, I. Sutskever och G. E. Hinton, ”ImageNet Classification with Deep Convolutional Neural Networks,” *Communications of the ACM,* Januari 2012.

[12] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally och K. Keutzer, ”SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size,” 24 Februari 2016. [Online]. Available: https://arxiv.org/abs/1602.07360.

[13] K. He, X. Zhang, S. Ren och J. Sun, ”Deep Residual Learning for Image Recognition,” 10 December 2015. [Online]. Available: https://arxiv.org/abs/1512.03385.

[14] G. Huang, Z. Liu, L. v. d. Maaten och K. Q. Weinberger, ”Densely Connected Convolutional Networks,” 25 Augusti 2016. [Online]. Available: https://arxiv.org/abs/1608.06993.

[15] Wired, 11 September 2015. [Online]. Available: https://www.wired.com/2015/11/google-open-sources-its-artificial-intelligence-engine/.

[16] Reddit, ”tensorflow vs pytorch, 22 200 hits on Reddit,” Mars 2021. [Online]. Available: https://www.google.se/search?q=tensorflow+vs+pytorch+site%3Areddit.com .

[17] The Gradient, ”PyTorch vs TensorFlow,” 10 Oktober 2019. [Online]. Available: http://horace.io/pytorch-vs-tensorflow/.

[18] E. Buber och B. Diri, ”Performance Analysis and CPU vs GPU Comparison for Deep Learning,” Oktober 2018. [Online]. Available: https://www.researchgate.net/publication/334168063_Performance_Analysis_and_CPU_vs_GPU_Comparison_for_Deep_Learning.

[19] HiTech Chain, ”AI Racing Robot Powered by Jetson Nano,” 2020. [Online]. Available: https://hitechchain.se/iot/ai-racing-robot-powered-by-jetson-nano.

[20] NVIDIA, ”JETPACK SDK,” 2020. [Online]. Available: https://developer.nvidia.com/embedded/jetpack.

[21] Waveshare Electronics Wiki, ”JetRacer AI Kit,” 26 Oktober 2020. [Online]. Available: https://www.waveshare.com/wiki/JetRacer_AI_Kit. [Använd Juni 2020].

[22] NVIDIA, ”Supported Modes and Power Efficiency,” Oktober 2020. [Online]. Available: https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/power_management_nano.html#wwpID0E0FL0HA.

[23] NyTeknik, ”Uber-olyckan: Bilen såg kvinnan, men reagerade inte,” 8 Maj 2018. [Online]. Available: https://www.nyteknik.se/fordon/uber-olyckan-bilen-sag-kvinnan-men-reagerade-inte-6913333.

[24] FN, ”Agenda 2030 och de globala målen för hållbar utveckling,” Mars 2020. [Online]. Available: https://fn.se/vi-gor/vi-utbildar-och-informerar/fn-info/vad-gor-fn/fns-arbete-for-utveckling-och-fattigdomsbekampning/agenda2030-och-de-globala-malen/.

[25] Modern Psykologi, ”Meningen med att gå till jobbet,” 21 Augusti 2015. [Online]. Available: https://modernpsykologi.se/2015/08/21/meningen-med-att-ga-till-jobbet/.

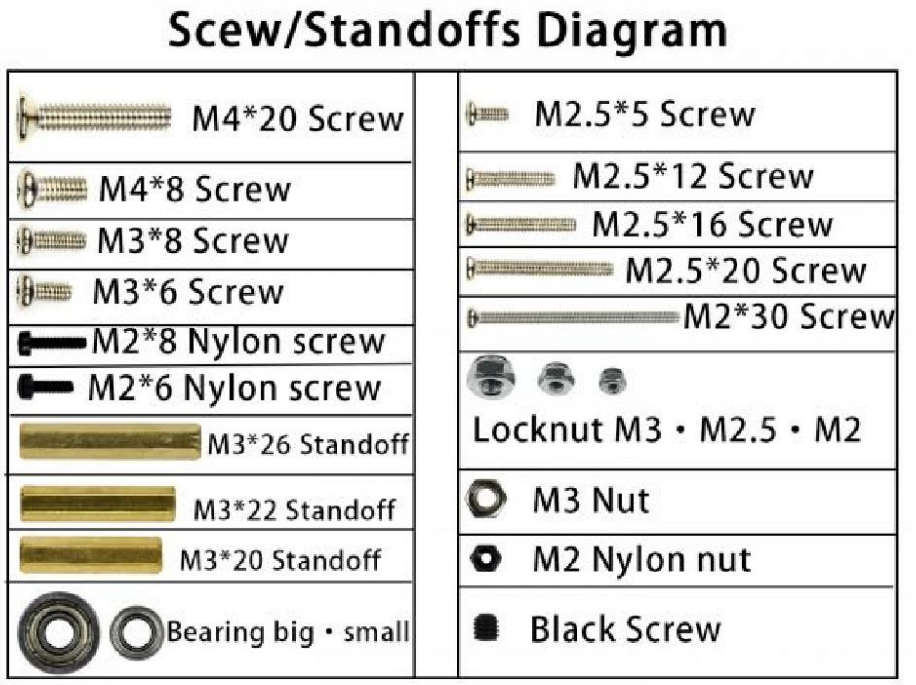
Bilagor

Bilaga 1 – JetRacer-manual:

JetRacer Assembly Manual

Screws/standoffs diagram

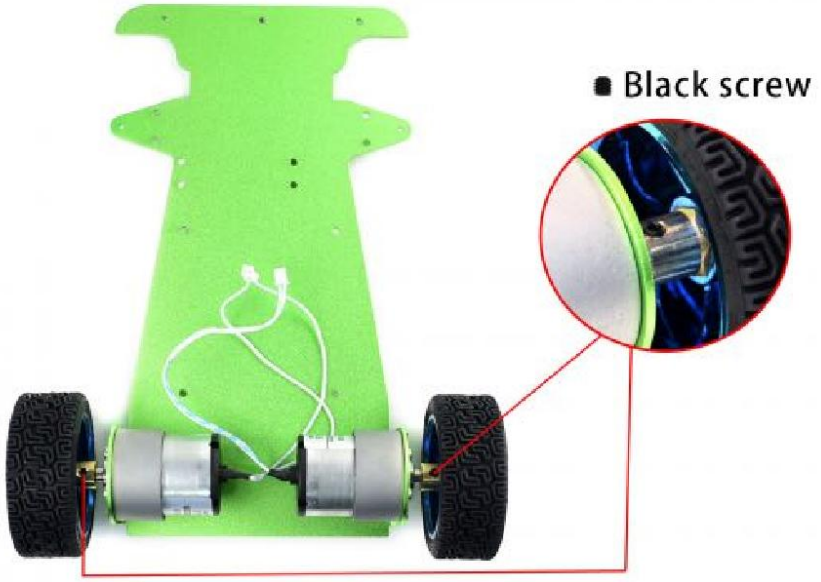
To let you find the screws easily, we make this diagram for reference. Note that the screws come with servo wheel and cooling fan are not listed here.



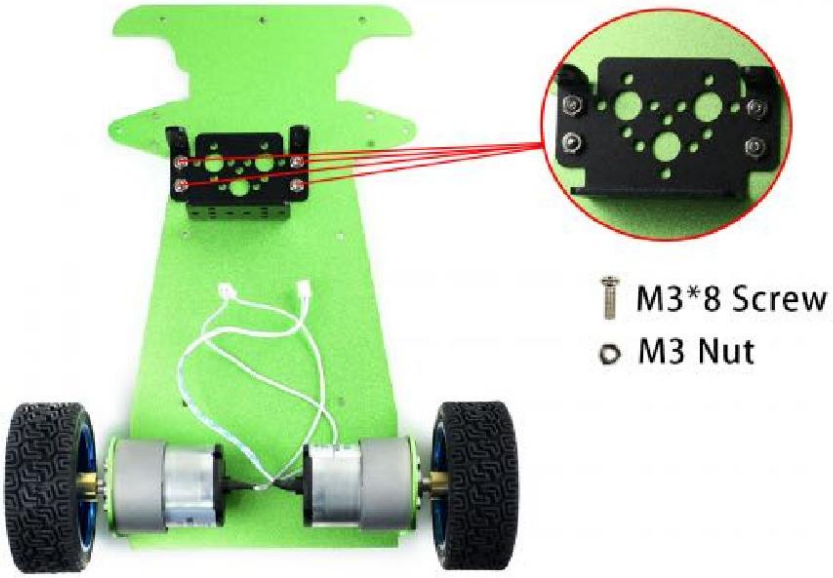
1. Fixing the motors to metal chassis with screw M3*6. **Note that you cannot replace the M3*6 screws by longer screw, otherwise the motor cannot work.**

2. Put the coupler to wheels. You may need to hit it into the wheel by tools and fix it by M4*8 screw.

3. Assemble the wheels. Turn it tightly with the Black screw. You should turn the screw by the longer side of the little spanner.



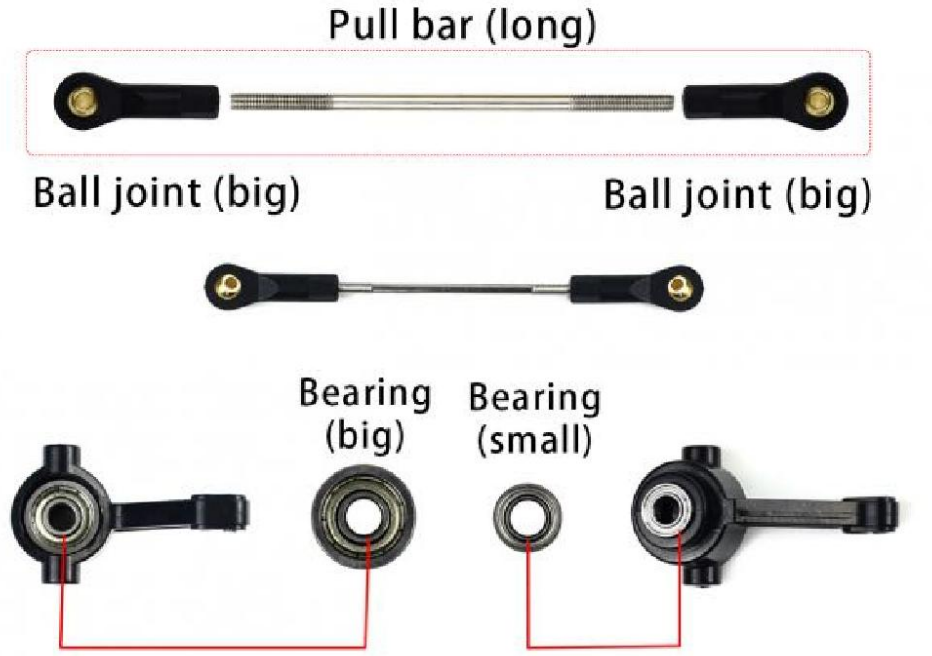
4. Mount the servo holder on metal chassis.



5. Set servo on the holder and fix it by screws and nuts. Please make sure that you put the sever in the correct way.

6. When you assemble the servo pull bar, please refer to the image below carefully. The pull bar is combined by two small ball joint and the short bar, Note that the two small ball should be perpendicular to each other. Fix servo wheel to the servo wheel holder by its own screws. Then fix the screw pull bar (the flat side) on the servo wheel holder by M2.5*12 screw and M2.5 locknut.**Note that the groove of the servo wheel is toward outside.**

7. Assemble the front-wheel pull bar. The front-wheel pull bard is combined by two bard joint (big) and the long bar. Then put the bearings in steering knuckle.



8. Together the servo pull bard, front-wheel pull bard, and steering knuckles. The servo pull bar in on the top, and then the front-wheel pull bard, finally the knuckles. The bigger bearing should toward inside. **Please refer to the image below carefully.**

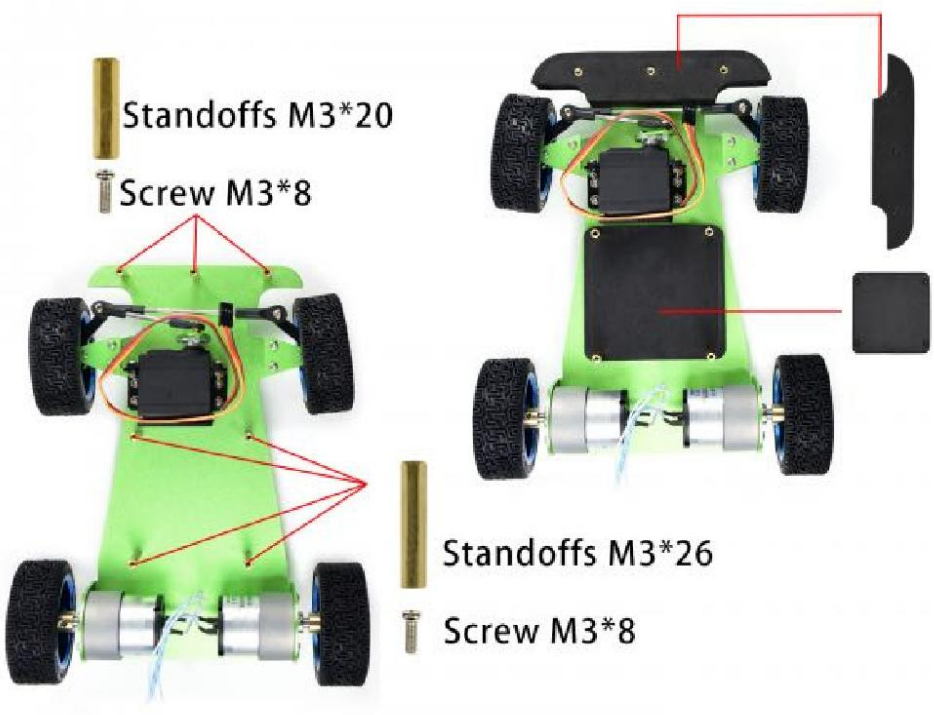
9. Fix wheel on the steering knuckle by M4 screws and locknut. **Note that**

you cannot fix the wheel too tight or too loose. Please test if the wheel can run smoothly after fixing.

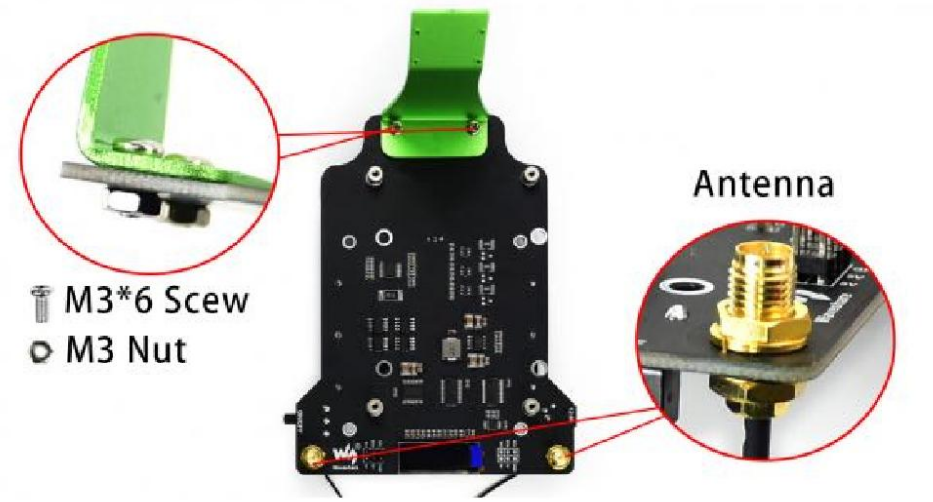
10. Set M3 standoffs for the front wheels.

11. Assemble the front-wheels combination. Put the servo wheel to servo, fix it by M3 screw. Fix the wheels by M2 screws and locknut and the triangle board.

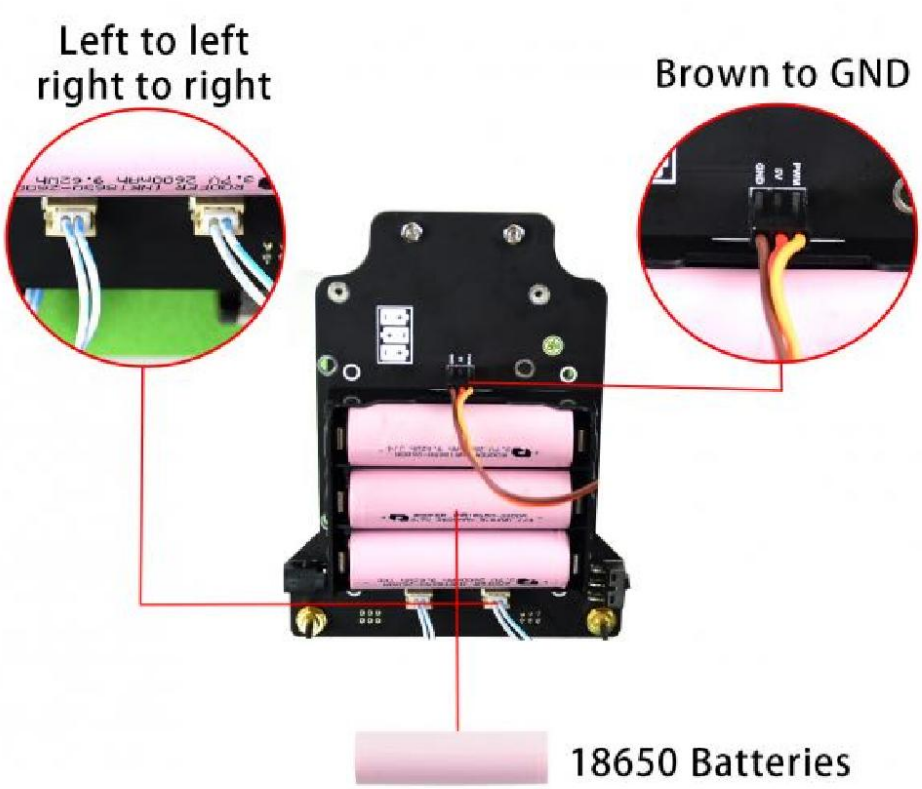
12. Put standoffs for JetRacer expansion board and bumper. Put the EVA felt pad.



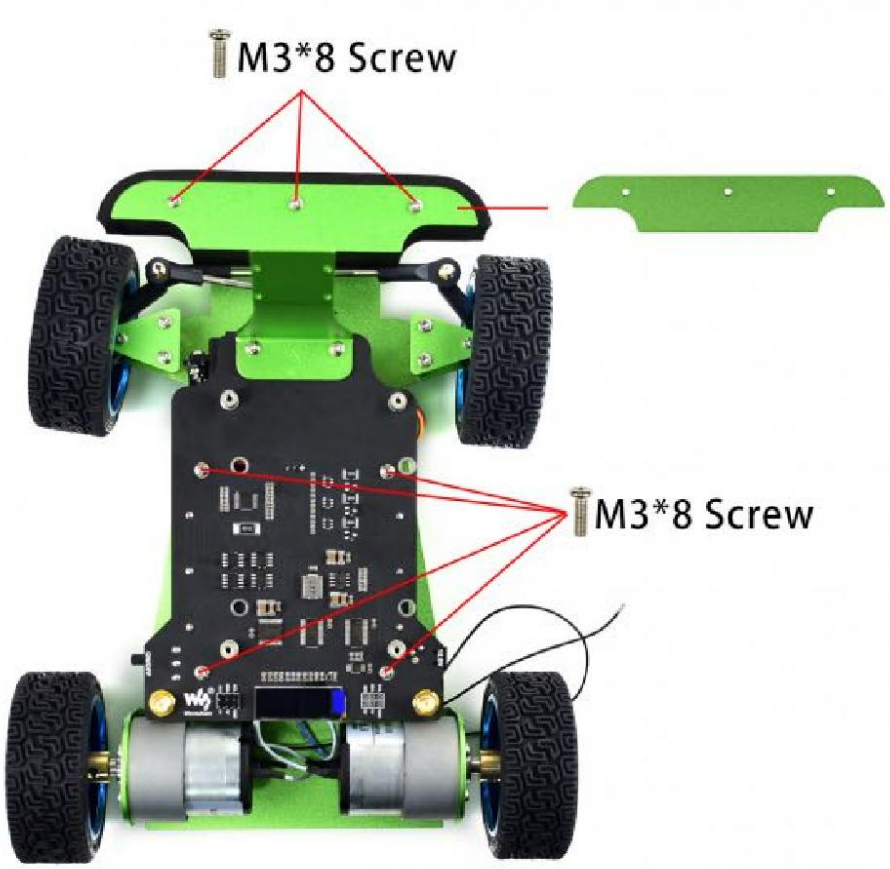
13. Set camera holder and antenna on JetRacer Expansion board, note that refer to the image below about the direction.



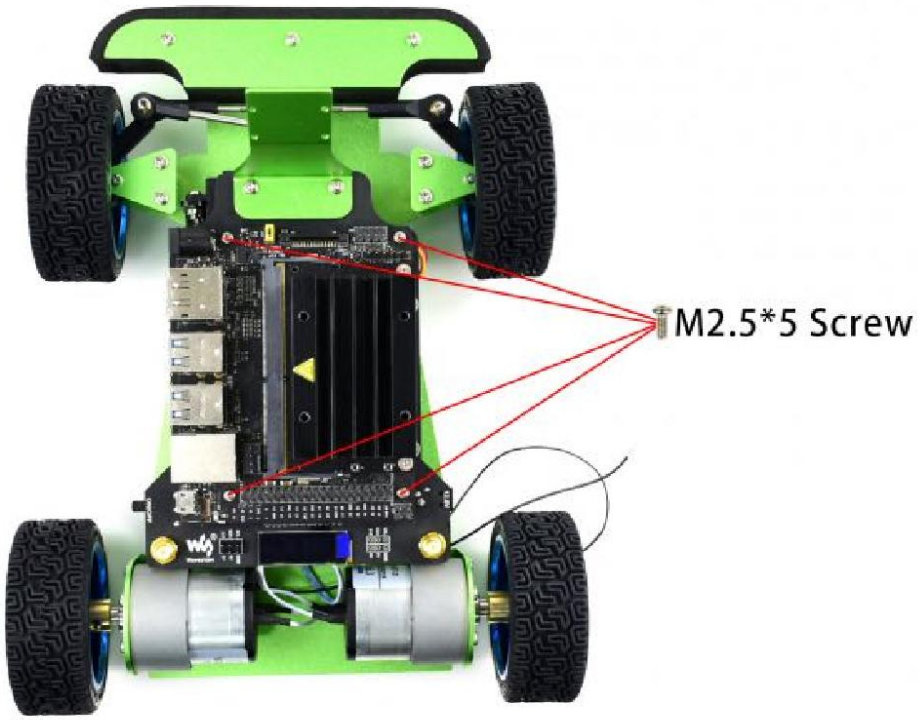
14. Assemble batteries in the correct direction. Connect wires of motor and servo to JetRacer Expansion board.



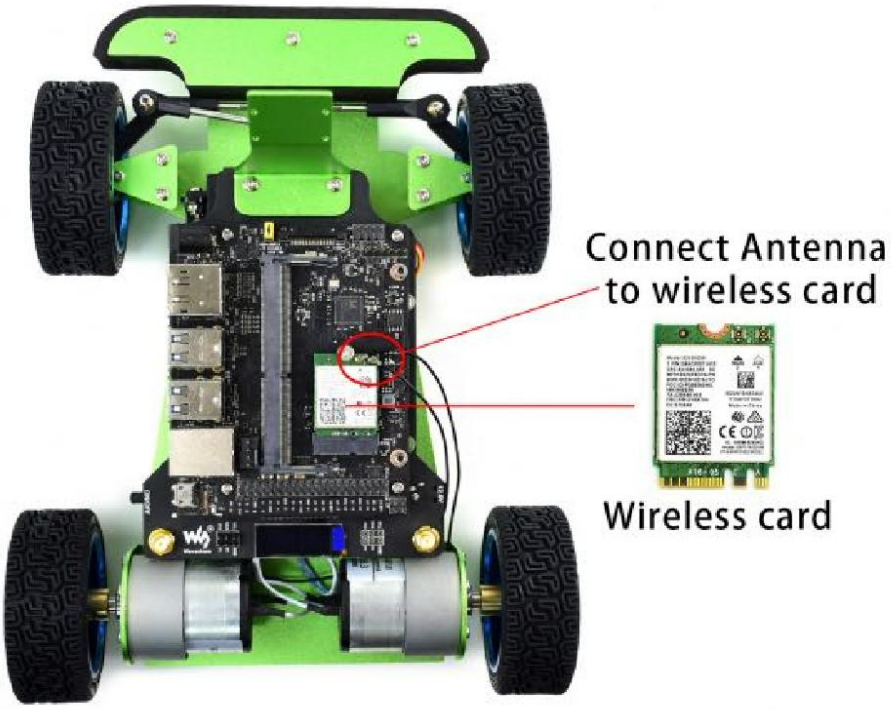
15. Adjust the place of wires then fix JetRacer Expansion board on metal chassis. Fix the metal bumper by M3 screws.



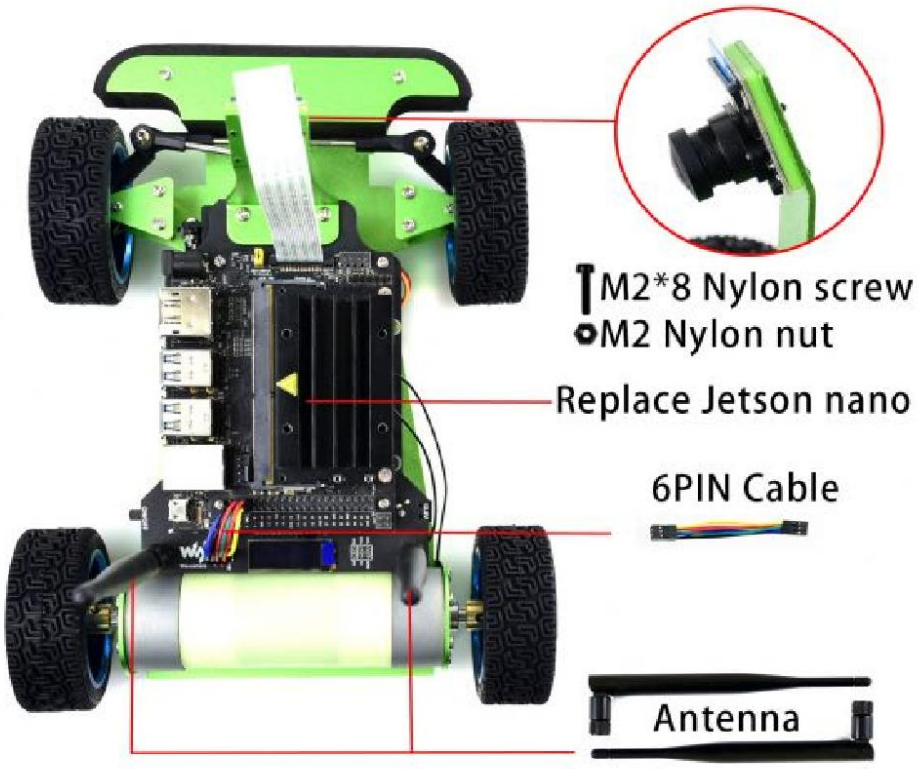
16. Put the Jetson Nano Developer Kit and fix.



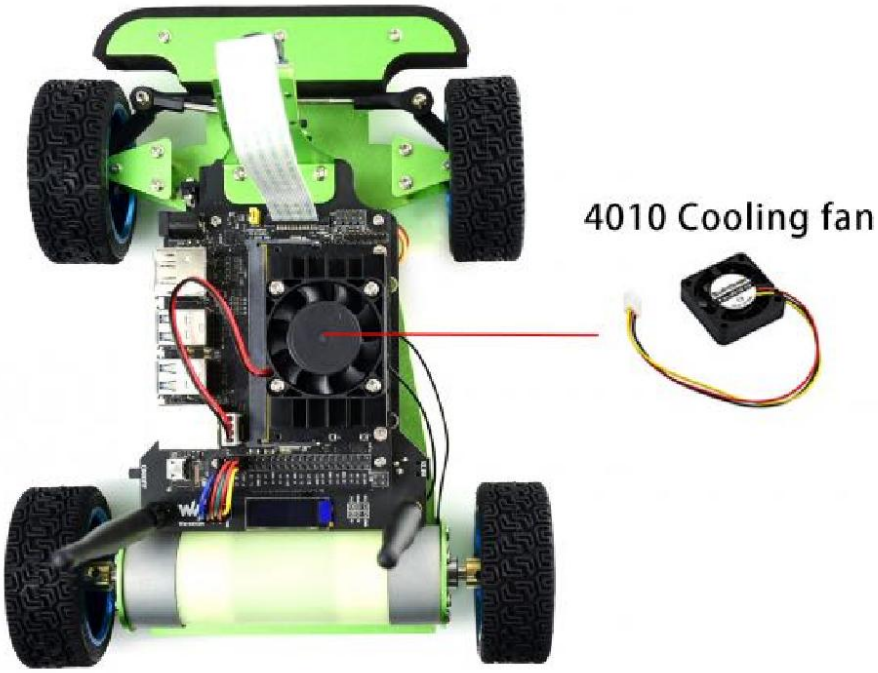
17. Remove the Jetson Nano board, connect the wireless card and connect the antenna.



18. Replace Jetson Nano. Mount camera to its holder by nylon screws. **Note that the Acrylic board should be put between camera and the metal holder to avoid shorting.** Set the 3D-printed motor enclosure on motors. Connect the Jetson Nano Developer Kit to JetRacer Expansion board by 6PIN wires. **You should connect 5V to 5V, 3.3V to 3.3V, please take care about it.**



19. Assemble cooling fan by its own screws. Connect the wires to the fan interface. The black wire should be connected to the far-left pin (GND).


4010 Cooling fan

Bilaga 2

Fil 1: setupJetRacer_1-will_reboot_nano.sh

```
#!/bin/bash

# Script made from https://www.waveshare.com/wiki/JetRacer_AI_Kit
# This script starts at "Step 5. Install the python package"

LOGFILE="$0.log"
TIMESTAMP=`date "+%Y-%m-%d %H:%M:%S"`
echo $TIMESTAMP >> $LOGFILE 2>&1

echo "View output in $LOGFILE"

echo "# Step 5. Install the python package" >> $LOGFILE 2>&1
echo "## Update Jetcard Python" >> $LOGFILE 2>&1
cd $HOME >> $LOGFILE 2>&1
mkdir ws >> $LOGFILE 2>&1
cd cd $HOME/ws >> $LOGFILE 2>&1
git clone https://github.com/waveshare/jetcard >> $LOGFILE 2>&1
cp jetcard/jetcard/ads1115.py ~/jetcard/jetcard/ >> $LOGFILE 2>&1
cp jetcard/jetcard/ina219.py ~/jetcard/jetcard/ >> $LOGFILE 2>&1
cp jetcard/jetcard/display_server.py ~/jetcard/jetcard/ >> $LOGFILE 2>&1
cp jetcard/jetcard/stats.py ~/jetcard/jetcard/ >> $LOGFILE 2>&1
cd $HOME/jetcard >> $LOGFILE 2>&1
pip3 uninstall jetcard -y >> $LOGFILE 2>&1


echo "Nano will reboot. Continue with setupJetRacer_2" | tee $LOGFILE 2>&1
TIMESTAMP=`date "+%Y-%m-%d %H:%M:%S"`
echo $TIMESTAMP >> $LOGFILE 2>&1
sudo reboot
```

Fil 2: setupJetRacer_2.sh

```
#!/bin/bash

# Script made from https://www.waveshare.com/wiki/JetRacer_AI_Kit

LOGFILE="$0.log"
TIMESTAMP=`date "+%Y-%m-%d %H:%M:%S"`
echo $TIMESTAMP >> $LOGFILE 2>&1

echo "View output in $LOGFILE"


echo "# continue from setupJatRacer_1" | tee $LOGFILE 2>&1
echo "# continue Step 5. Install the python package" | tee $LOGFILE 2>&1


echo "## Finish Jetcard setup" | tee $LOGFILE 2>&1
cd $HOME/jetcard >> $LOGFILE 2>&1
sudo python3 setup.py install >> $LOGFILE 2>&1

echo "## Install the JetCam Python package" | tee $LOGFILE 2>&1
cd $HOME >> $LOGFILE 2>&1
git clone https://github.com/NVIDIA-AI-IOT/jetcam >> $LOGFILE 2>&1
cd jetcam >> $LOGFILE 2>&1
sudo python3 setup.py install >> $LOGFILE 2>&1

echo "## Install the torch2trt Python package" | tee $LOGFILE 2>&1
cd $HOME >> $LOGFILE 2>&1
git clone https://github.com/NVIDIA-AI-IOT/torch2trt >> $LOGFILE 2>&1
cd torch2trt >> $LOGFILE 2>&1
sudo python3 setup.py install >> $LOGFILE 2>&1

echo "## Install the JetRacer package" | tee $LOGFILE 2>&1
cd $HOME >> $LOGFILE 2>&1
git clone https://github.com/NVIDIA-AI-IOT/jetracer >> $LOGFILE 2>&1
cd jetracer >> $LOGFILE 2>&1
sudo python3 setup.py install >> $LOGFILE 2>&1

echo "# Step 6. Configure power mode" | tee $LOGFILE 2>&1
echo "## To prevent the Jetson Nano from drawing more power than the battery
can supply, we set it to 5W mode" | tee $LOGFILE 2>&1
sudo nvpmodel -m1 >> $LOGFILE 2>&1
# Check if mode is correct
sudo nvpmodel -q | tee $LOGFILE 2>&1


echo "# Personal stuff" | tee $LOGFILE 2>&1
sudo apt install nano -Y        # Install Nano
```