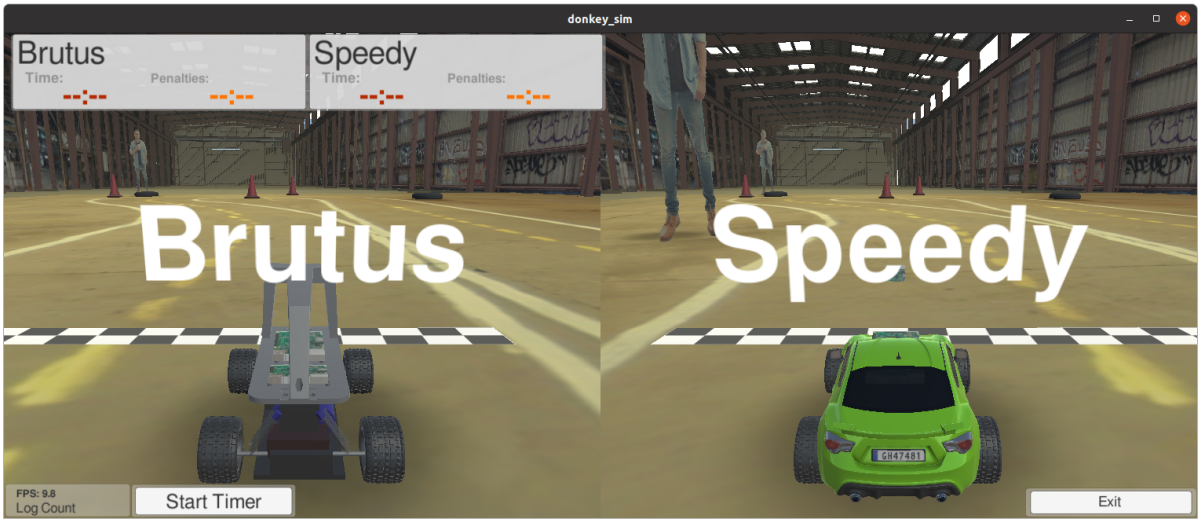
Experience autonomous driving:

From the simulator to the race track

Your entry into the world of artificial intelligence



No gray theory but a practical step by step introduction to:

- Artificial Intelligence and Deep Learning
- Generating synthetic data in the simulator
- Training of an autopilot / neural network
- Testing the autopilot in the simulator
- Step by step instructions for building a real model robot car

Author: Ingmar Stapel (Version 0.7 | 2021)

E-Book Download Seite: *https://custom-build-robots.com/donkey-car-e-book-de*

Content

1 Introduction

There are several projects that approach the topic of artificial intelligence and autonomous driving model cars. Some of these projects exist as simulators and others for the real world. The Donkey Car project exists both as a simulator and as a physical model car. So I am very happy to introduce the Donkey Car project to you in this book.

In November 2016, Adam Conway and Will Roscoe launched the Donkey Car project. The idea that drove both was to give interested hobbyists a possibility to enter the world of self-driving cars in model format. The most important reasons for me to use the Donkey Car framework are the large international community, the free availability of the software, the Donkey Car Simulator and the fact that standard modeling technology is used to build the physical robot car. The use of classic model cars reduces the costs significantly.

Originally, the Donkey Car framework was developed for the Raspberry Pi, which is the best-selling single board computer and with its huge community helped lay the foundation for the success of the Donkey Cars project. With the availability of the NVIDIA Jetson Nano 2019 which has a powerful GPU architecture and its similar ARM processing unit as used by the Raspberry Pi, the Donkey Car community immediately recognized the benefits and additionally extended the framework with support for the Jetson Nano. One of the biggest advantages of the Jetson Nano compared to the Raspberry Pi is that neural network training is possible directly on the Jetson Nano thanks to its GPU architecture. With the built-in GPU units, the Jetson Nano can perform parallel calculations that allow it to train neural networks in a high-performance manner. The Raspberry Pi 4 with 4 GB RAM but without GPU support has no chance against the Jetson Nano when it comes to executing and training neural networks. The Donkey Car Framework written in Python and its good inline documentation additionally facilitate the introduction to the project of autonomously driving robot cars and to the topic of artificial intelligence. At this point, the simulator for the Donkey Car should also be mentioned. This allows you to get started with the Donkey Car project in a very easy and really inexpensive way. This is possible because you can easily install the Donkey Car Simulator on existing hardware such as a laptop or PC.

My conclusion in sum is that the Donkey Car Framework in its combination of simulator solution and physical model car is very practice-oriented, easy to understand and therefore ideal for the entry into the topic of artificial intelligence.

The official Donkey Car project page can be reached at the following URL.

URL: *https://www.donkeycar.com/*

You can always find the latest version of the Donkey Car e-book from me on my blog at the following URL for download.

URL: *https://custom-build-robots.com/donkey-car-e-book-de*

I am very happy to receive your feedback on this e-book. You can reach me as follows.

E-mail: *ebook@custom-build-robots.com*

2 The basics of the project

Have you always wanted to understand how autonomous driving cars work and get an idea of the state of the art yourself? Then you are exactly right here! I will show you how to teach a neural network to control a robot car in a simulator and in real life.

With this e-book you have a step-by-step guide in your hand that focuses on doing it yourself. When writing this e-book, it was important to me to convey the basics and theories you need to know when dealing with artificial intelligence in an easy to understand way. So the e-book will not dive into the theoretical depths of neural network design. Instead, you will simply apply neural networks as easily as withdrawing money from an ATM. Here, you don't need to understand the concepts and structures that contribute to using and operating the ATM. Rather, we as a society must recognize that it is time to deal with artificial intelligence and simply apply it. Only in this way can the following generations grow up with it and get the opportunity to question the technology.

2.1 How do neural networks learn?

There are different approaches how neural networks can be learned. A short overview should help you to get to know the possibilities and essential differences without having to spend a lot of time with theory. I would like to introduce three concepts how neural networks learn.

Behavioral Cloning: In behavioral cloning, all actions that a person performs to achieve a certain goal are recorded. In a subsequent step, a neural network is trained with the recorded data. By observing the human behavior, the machine learns to solve the same problem or to predict the possible behavior of the human as good as possible. A neural network is a network of nodes and relationships. The nodes have weights and by looking at what the human has done, these weights are set and the neural network starts to learn.

Un-Super Wise Learning: Un-Super Wised Learning has become possible because a great deal of data on, for example, a technical topic has already been collected and is available in machine-readable form. Added to this development is the low-cost computational pipeline in the form of GPUs that has made it possible to search massive amounts of unstructured data for patterns. This combination of lots of data and cheap computational power enabled neural networks to search for patterns in huge datasets that deviated from the general noise. By adjusting the weights of the nodes (neurons) of the neural network, it then learns to make predictions based on the data.

Reinforcement learning**:** In reinforcement learning (RL), the neural network is trained in a simulator. At the beginning, the neural network does not know the task and learns to make the right decisions, i.e. to set the weights of the nodes in the network, through a system of rewards and punishments. After many repetitions the neural network in the simulator learns to solve the problem. If the simulated world corresponds closely to reality, such as a digital image of a factory floor, the trained neural network can be used outside the simulator in the real world. An example is the DeepRacer from Amazon, where the neural network is trained in the simulator by RL and then steers the DeepRacer around the racetrack.

The human BIAs: When you have worked through this book you will notice how you as a human being will consciously or unconsciously influence the predictions of the neural network with your behavior. Exactly this unconscious passing on of personal experiences and behavior patterns is called the human BIAs. In projects dealing with the training of neural networks the BIAs can lead to problems. For example, if you drive the donkey car very slowly while recording the training data, the neural network will later steer the robot car very slowly around the track. This was a very simple example but very good to try out in this project. Or drive the race track only clockwise and then let the donkey car drive counterclockwise. Most likely it will not succeed because it has not learned enough to steer to the left.

That's enough of the theory. Get started right away in the following chapter with the installation of the Donkey Car Simulator. Try using the simulator to teach a neural network to drive on a race track.

3 The Donkey Car Simulator

Before you spend money on hardware such as an RC model car and associated electronics, start with the simulator. Here you can already experience many of the concepts behind the term Deep Learning that are necessary to train a neural network. But be sure, in the real world a neural network behaves quite differently.

For the Donkey Car Framework, there is an extra simulator in addition to the hardware robot car. You have to install this on an x86 environment, the so-called host PC, if you want to try it out. The background for this is that this simulator uses the Unitiy engine and this is only available for the x86 CPU environment. Therefore, the simulator can currently be installed on Windows, Linux and MAC OS. The simulator provides you with an exact replica of the Donkey Car framework.

Figure 3.1: Donkey Car Simulator (Unity environment)

A lot of what you learn in the simulator about the Donkey Car framework would apply later when you build the real Donkey Car as a physical model.

In the simulator you collect training data and with this training data you train the autopilot that steers the car around the race track. You use exactly the same commands and concepts that you would later use in the real model car.

Very interesting about this simulator is the possibility to test different things at your desk. This includes, for example, teaching the neural network special driving behaviors such as overtaking, or if you want to go deeper, to test different designs of neural networks before you use them in the real world on your donkey car. You can also try out image manipulations like changing the brightness in the images to test if a neural network tends to become more stable and steers the car around the track more reliably.

With all the possibilities in the simulator, you must be aware that the real world is many times more complex than the one in a tidy and sterile simulator. Here, the community with the Donkey Car Simulator still has a long way to go when it comes to transferring a neural network from the simulator directly into the real world.

I will now guide you step by step through the installation of the Donkey Car Simulator under an Ubuntu and Windows environment. I am using two old Lenovo laptops for this. One is a T520 and the other is a T450s both of which do not have GPU support yet. As mentioned above, the simulator cannot be

installed on a Jetson Nano because the Unitiy engine is only available for the X86 architecture. I tested this guide for ***Ubuntu 18.04***, ***Ubuntu 20.04*** as well as ***Windows 10,*** although there were minor challenges under ***Windows 10 for*** which I will suggest a solution in each case.

If your computer has a GPU I will list the commands you need to use to install the GPU version of TensorFlow.

You can find the original Donkey Car Simulator manual at the following address.

URL: http://docs.donkeycar.com/guide/simulator/

In the following chapter I will guide you step by step through the installation of the simulator under a ***Windows 10*** environment. If you are using Ubuntu then please skip the following chapter.

3.1 The Windows Host PC Installation

I would like to inform you in advance that there is a minor problem that I have not yet been able to resolve when installing the Donkey Car framework and simulator under Windows. I could not activate the direct support of a joystick in the Anaconda environment under Windows. Nevertheless, a joystick can be used if the Donkey Car is controlled remotely via the web interface.

Thus, the Donkey Car Simulator is fully functional under Windows and with the training data recorded in the simulator you can train a working neural network. In my experience and estimation, the Donkey Car framework works a little bit better under Ubuntu so far. The community for the Linux variant of the Donkey Car is larger and correspondingly faster at solving problems. But also with the problems I had to solve under Windows to write this tutorial as good as possible I could count on the help of the community.

But now we start with the setup of the Donkey Car Framework under Windows. First we will start with the necessary tools that you need to install in order to be able to install the Donkey Car Framework and Simulator extension under Windows.

3.1.1 Necessary tools:

In order to install the Donkey Car Framework on ***Windows 10,*** you must first download and install the following programs.

Miniconda

Download the latest version of Miniconda3 from the following URL.

URL: *https://docs.conda.io/en/latest/miniconda.html*

In my case it was the version ***Miniconda3 Windows 64-bit with Python 3.8***. Now install this or a newer one and leave all suggested settings for the installation on default, i.e. as suggested by the installer.

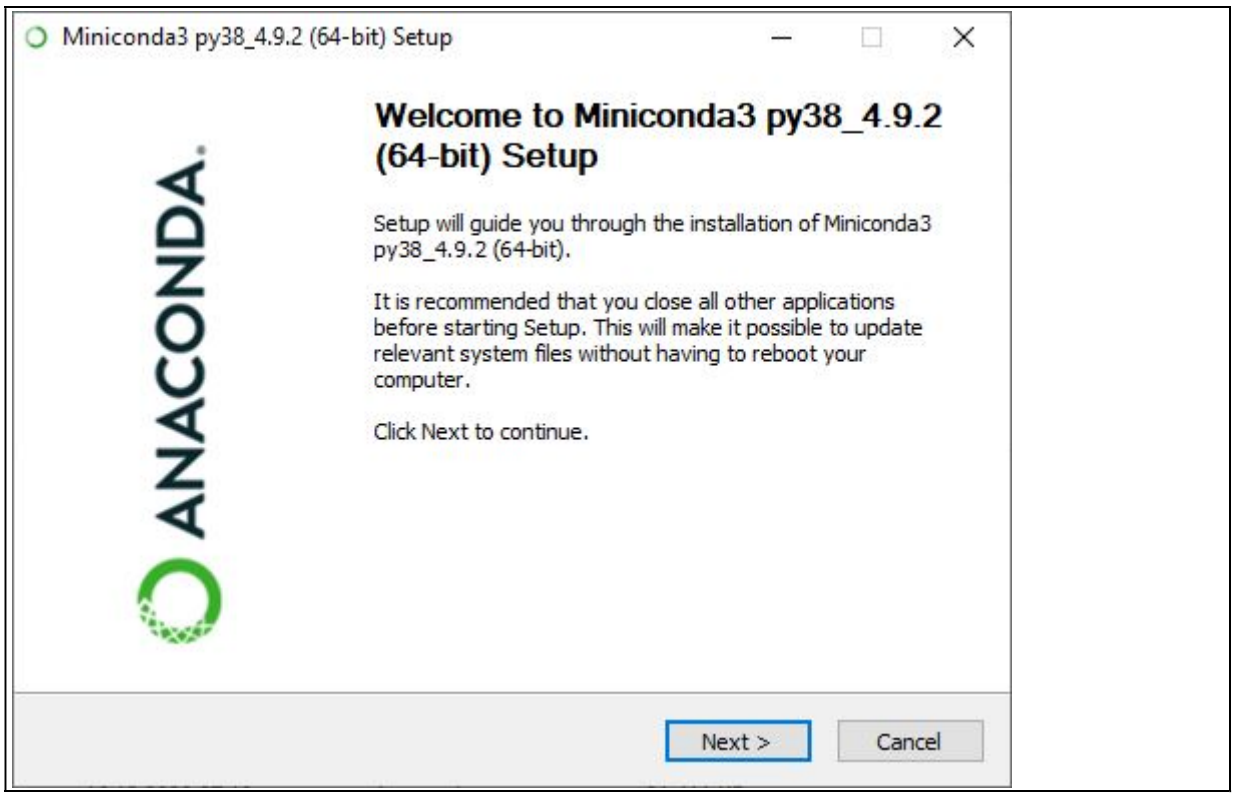


Figure 3.2: Miniconda3 installation

After tapping Next a few times, the installation should complete without errors. You should now find an entry ***Anaconda Promt (miniconda3)*** under Programs. When you start the program, a terminal window or prompt opens. In this you will work later and install the Donkey Car Framework.

The window or prompt looks like this.

Figure 3.3: Anaconda Promt (miniconda3)

Git client

You will download the Donkey Car framework from GitHub over the next few sections. To do this, you will need the Git Client on your Windows PC. Therefore, download the latest version of the Git Client from the following URL.

URL: *https://git-scm.com/download/win*

The Git Client I downloaded and installed was the ***Git Client 2.30.0*** for my 64-bit Windows system. Again, I used the suggested default settings for the installation of the Git client 1:1 as suggested and did not change anything.

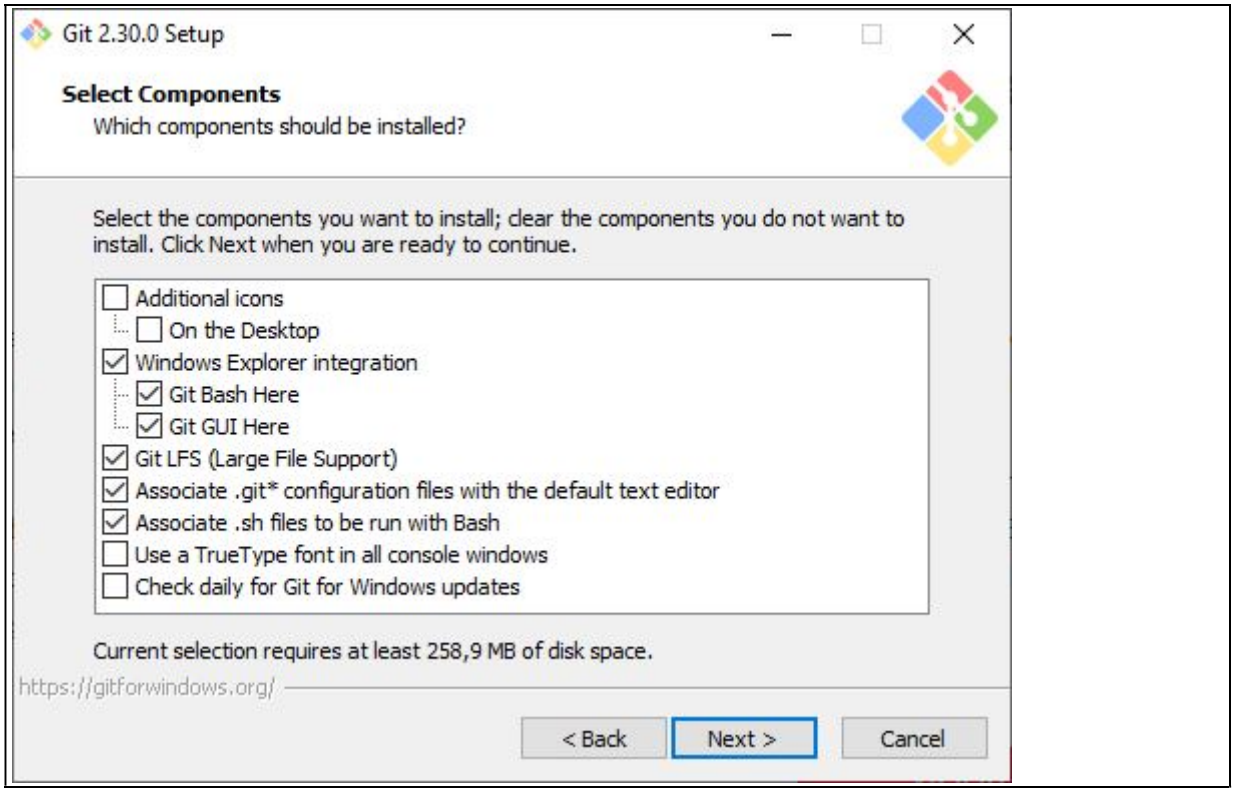


Figure 3.4: Git client installation

After the previously shown image of the Git installation process, there are many more dialog boxes. Here I always tapped Next and did not change anything.

Microsoft Visual Studio Code

For small customizations to Python programs I recommend to install Visual Studio Code from Microsoft. Of course you can also use any text editor but I still recommend Visual Studio Code. With Visual Studio Code the customizations will be very easy and you have a modern development environment installed on your computer.

URL: *https://code.visualstudio.com/download*

3.1.2 Installing the Donkey Car Framework

Now create a folder on your drive with the name **donkeycar_sim.** You can do this easily with the Windows file explorer.

Now start the Anaconda console or promot and change to the **donkeycar_sim** folder.

Command: cd donkeycar_sim

Now the next thing you need to do is download the Donkey Car Git repository. To do this, now run the following command in the **donkeycar_sim** folder.

Command: git clone https://github.com/autorope/donkeycar

After downloading the Donkey Car repository, please change to the **donkeycar** directory with the following command. The Donkey Car framework will be cloned into this directory.

Command: cd donkeycar

Then check out the master branch with the following command.

Command: git checkout master

Now you need to set up the ***Miniconda3*** virtual environment named ***donkey.***

Please run the following command to start the Donkey Car Framework installation.

Command: conda env create -f install\envs\pc.yml

The setup of the Conda environment ***donkey*** takes several minutes depending on the internet connection until the installation is completed.

Note: If you get an error message like "PackagesNotFoundError: The following packages are not available from current channels: - tensorflow==2.2.0", this is because there is no TensorFlow version 2.2.0 for Conda. Please change the pc.yml file to TensorFlow version 2.3.0 and start the installation again.

After the installation has been completed successfully, you should see the following message in the Conda terminal window.

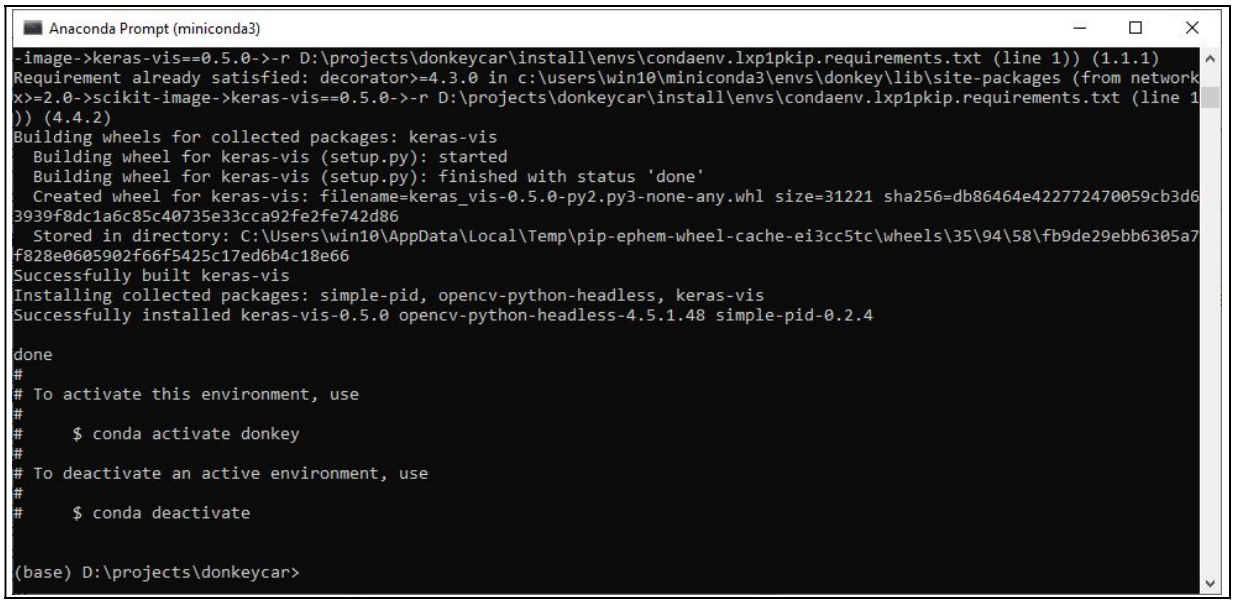


Figure 3.5: Successful installation of the donkey virtual environment

Now activate the virtual environment ***donkey*** with the following command. This should also be displayed at the end of the previously started installation as you can see in the picture before.

Command: conda activate donkey

In case you also want to remove a virtual environment once then you can use the following command to delete it along with all installed packages.

Command: conda env remove --name myenv

If everything worked without an error message then you can now install the donkey car framework. Start the installation with the following command within the active Conda environment ***donkey***.

Command: pip install -e .[pc]

The following image shows the final message after the Donkey Car Framework has been successfully installed.

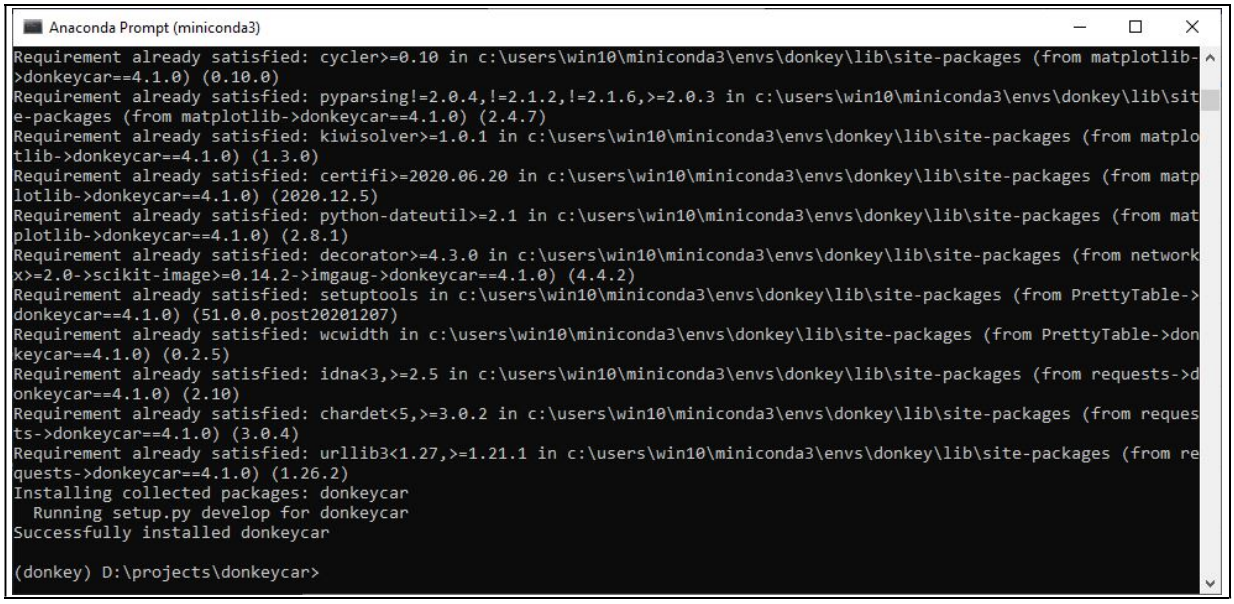


Figure 3.6: Successful Donkey Car Framework installation

Now the Donkey Car Framework Host PC installation on Windows is successfully completed. In the following section we will set up the simulator on your host PC step by step.

3.1.3 Setting up the simulator under Windows

Now you need the Unity environment for Windows, which provides the race tracks for the simulation that you will drive with the Donkey Car Simulator afterwards. To do this, you still need to download the package with the simulator for Windows. All versions released so far can be found at the following URL.

URL: *https://github.com/tawnkramer/gym-donkeycar/releases*

Then unzip the simulator to a location of your choice. If you want to start the simulator already, you can execute the exe-file ***donkey_sim.exe.*** Then the simulator starts without the Donkey Car Framework.

Note: If you get a message that an EXE file is being executed that Windows does not trust, please confirm in this dialog that you trust the EXE file so that Windows also executes the simulator.

Now switch back to the active Conda environment ***donkey*** and download the simulator environment gym-donkeycar ***of*** the donkey car framework. To do this, change to the **donkeycar_sim** folder within your active ***donkey*** Conda environment. In the folder **donkeycar_sim** you had already downloaded the Donkey Car Framework before.

Command: git clone https://github.com/tawnkramer/gym-donkeycar

Change to the **gym-donkeycar** folder with the following command.

Command: cd gym-donkeycar

Now you need to install the simulator so that this can be executed.

Command: pip install -e .[gym-donkeycar]

The installation can take several minutes depending on how fast your host PC is or your internet connection.

After the Donkey Car Simulator framework is set up, you will need to create an instance of it to work with later. You will configure this instance to your needs. To do this, run the following command which will create a folder named **mysim** on your drive.

Command: donkey createcar --path /mysim

You should now get output like this and the Donkey Car Simulator environment is initialized. Various folders and files have been created, such as the ***myconfig.py*** file. In this ***myconfig.py file*** all configurations concerning the Donkey Car framework, i.e. your personal individualization of the Donkey Car instance, take place.

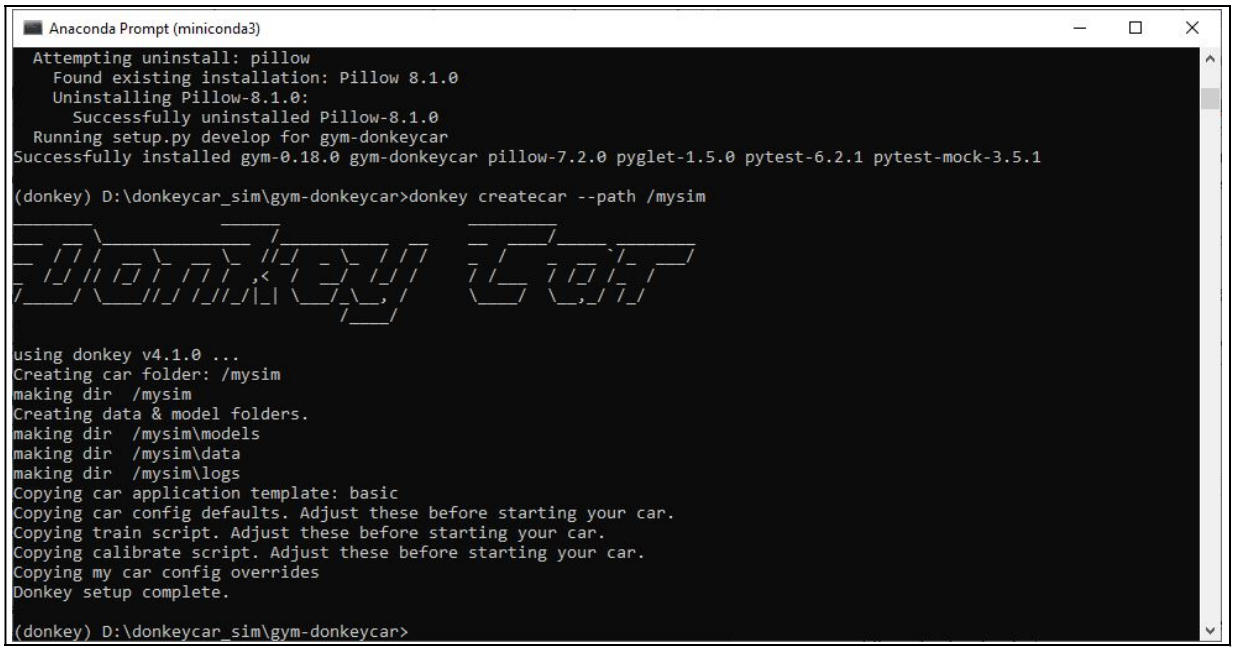


Figure 3.7: Successful installation of the Donkey Car Simulator framework

Next you have to configure the simulator. How to do this exactly is described in chapter 3.2.4for Windows.

Only after you have carried out the configuration, everything is set up and you can drive the first laps with the Donkey Car Framework in the simulator. To start the simulator, execute the following command in the mysim folder.

Command: python manage.py drive

The Donkey Car Framework will start and run the Unity Simulator and place your Donkey Car in front of the finish line. Now you can do a few laps.

For this you have to call the URL in the browser under which the Donkey Car Simulator Framework is reachable. This is displayed in the console output of the Conda environment and the address should be structured as follows.

URL: *<IP address of host PC>:8887*

Once you have opened the correct address in the browser, you should see the following screen and be able to control the Donkey Car via the web controls.

3.2 The Ubuntu Host PC Installation

This chapter is about installing the Donkey Car Framework including the simulator under Ubuntu. In my experience, the installation under Ubuntu is somewhat easier, and if you have difficulties, you will get help sooner and faster than under Windows. Since this tutorial is also aimed at beginners in the Linux world, this section begins with tools and tips to make your work under Linux easier in case you are not yet so familiar with Linux.

3.2.1 Little helpers and tips:

If you are not so familiar with Linux, then I recommend you to install a few helpful tools like the text editor Nano, the file explorer Midnight-Commander as well as Screen. These will help you to make the configuration of the robot car a little easier. This will reduce the special Linux command line commands to a minimum. All this I will explain to you now step by step

Update Ubuntu installation

First, however, update the local repository information of your host PC's Ubuntu installation. To do this, run the following command.

Command: sudo apt-get update

Use the upgrade command to update the programs already installed on Ubuntu.

Command: sudo apt-get upgrade

Choice of Display Manager

During the installation of the latest software packages there is also a question about the display manager and the option to change the display manager. The gdm3 display manager is preselected as default option as the following picture shows. Since the gdm3 display manager is very easy to use in my personal opinion, please press OK to continue using it.

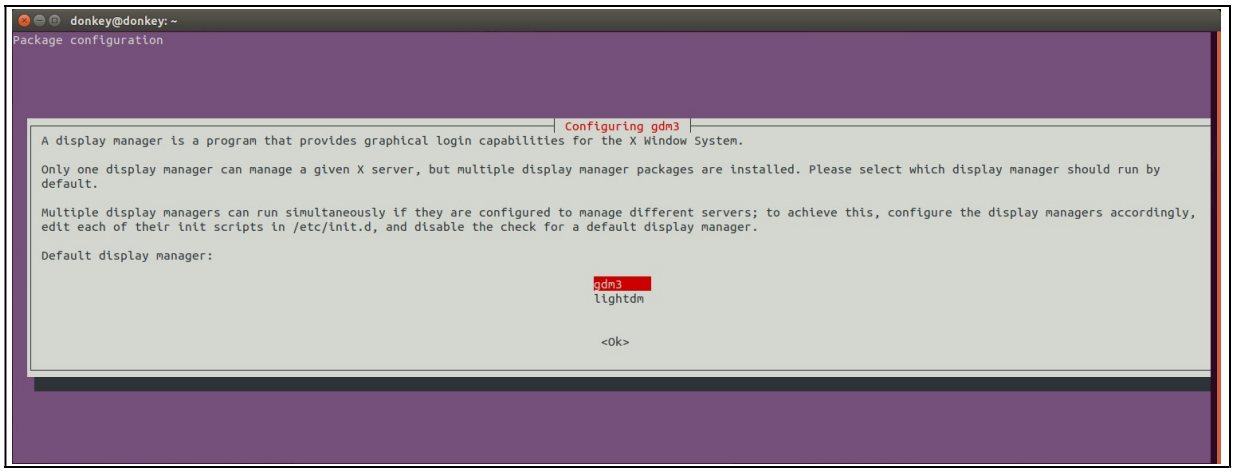


Figure 3.8: DGM3 Display Manager

Text editor Nano

Now install the text editor ***Nano in*** addition to the already existing text editors. Nano is an easy to use text editor for the terminal window. You will use Nano as text editor in the further course again and again to make e.g. small adjustments in configuration files or Python programs.

Command: sudo apt-get install nano -y

With the key combination **CTRL + X** you close the text editor Nano. If you have changed a file you have to confirm the saving of the change with a Y for Yes by pressing the Enter key. If you want to discard the change, press N for No and the Enter key.

In the terminal window, type nano <file name> to start the text editor and open a specific file at the same time.

Command: nano <filename>

Midnight Commander

With the Midnight ***Commander*** you can easily edit or copy files in the terminal window without having to know the necessary Linux commands for the terminal window. Install the Midnight Commander (mc) with the following command.

Command: sudo apt-get install mc -y

If you want to start the ***Midnight Commander*** then enter the following command in the terminal window.

Command: mc

With the F-keys on your keyboard you can call up the most important commands of the Midnight Commander. This small overview should help you to get started with the operation of the MC more easily.

- F4: Opens the selected file, e.g. in Text Editor Nano.

- F5: Copies a file or folder selected e.g. in the left window to the path opened in the right window.
- F6: Moves a file or folder selected e.g. in the left window to the path opened in the right window.
- F7: Creates a new folder.
- F8: Deletes the file or folder that is currently selected.
- F10: exits the Midnight Commander

FFmpeg video library

With ***FFmpeg*** you can record, convert and also stream digital video and audio material under Linux. So that you can create a video from the recorded training data, i.e. the thousands of images you need to train the neural network, for easier quality assurance, I therefore recommend that you install the versatile tool ***FFmpeg.*** Later in this book I will introduce you to a script that allows you to automatically create videos from your training images. Use the command below to install ***FFmpeg***.

Command: sudo apt-get install ffmpeg -y

Network file sharing

In order to be able to easily access the files of your computer and later the training data for backups, quality assurance etc. from e.g. Windows, it is advisable to install ***SAMBA*** for file sharing in the network. You install the ***SAMBA*** server with the following command.

Command: sudo apt-get install samba -y

After SAMBA is installed you have to modify the file smb.conf in the folder /etc/samba. To do this, open it in the text editor Nano with the following command.

Command: sudo nano /etc/samba/smb.conf

At the very end of the file, add the following lines. With this small adjustment, you share the /home/ folder of your Ubuntu machine on the network.

```
[Robot Car]
    path = /home/
    writeable = yes
    public = yes
    guest ok = yes
    guest only = yes
    guest account = nobody
     browsable = yes
```

Since a typical Ubuntu installation does not have the ***OpenSSH server installed,*** I installed it with the following command. This is only needed if you want to log in remotely via SSH on the Ubuntu host PC.

Command: sudo apt install openssh-server

Now you have updated your Linux Ubuntu installation so far and perhaps also installed the small helpers in the form of programs as recommended by me. Now it goes on step by step with the installation of the programs which are necessary for the startup of the Donkey Car Simulator.

3.2.2 Installation - Miniconda

Now first download the Miniconda3 installation file to install ***Miniconda3*** on the host PC. With the help of Miniconda3 a virtual environment is provided in which the Donkey Car Framework is installed.

Use the following two commands to download Miniconda3 to your user's home directory.

Command: cd ~/

Command: wget *https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh*

When the download is complete, start the installation of ***Miniconda3*** with the command below.

Command: ./Miniconda3-latest-Linux-x86_64.sh

Now it can happen that a message appears after executing the command as shown below and the installation is not possible, so it aborts.

Error message: -bash: ./Miniconda3-latest-Linux-x86_64.sh: Permission denied

If you see this message, change the file permissions with the following command and start the installation again.

Command: sudo chmod 777 Miniconda3-latest-Linux-x86_64.sh

If you were able to start the installation, you should see the terms of use after a few seconds. Press the ENTER key until you are prompted to accept or reject the terms of use. Please enter "yes" at the prompt to accept them and press the Enter key again to confirm.

Immediately afterwards you will be asked where Miniconda3 should be installed. Press the Enter key again if Miniconda3 should be installed in the home directory of your user. I recommend the default installation in the home directory of your user. Now the installation takes several seconds to minutes.

After the installation you will be asked if you want Miniconda3 to start the Conda base environment when you log in. For the sake of simplicity, confirm this step with "yes".

Note: If you always want to start conda manually, you can enter "no" instead of "yes" as suggested. In this case, enter the following line at the very end of the .bashrc file in your home directory.

```
export PATH=~/miniconda3/bin:$PATH
```

Now please reload the .bashrc file with the following command to start the base virtual environment.

Command: source ~/.bashrc

Now enter the command ***Conda in*** the terminal window to test whether Miniconda3 has been installed and set up correctly.

Command: conda

It now starts ***Conda*** and shows you the parameters that you can enter as parameters after calling the command conda. If this worked then the next step is to download the Donkey Car framework.

Installing the Donkey Car Framework

The Donkey Car Framework should be installed in a folder **donkeycar_sim** which is located below the home directory of your user. Therefore, create this folder in your home directory with the following command.

Command: mkdir ~/donkeycar_sim

Then change to the **donkeycar_sim** folder with the following command.

Command: cd ~/donkeycar_sim

To be on the safe side, install Git on your Ubuntu machine. If Git is already installed then the following call is very fast and Git will not be installed again.

Command: sudo apt-get install git

Now that you have made sure that Git is available on your computer, clone the donkey car framework into the ***donkeycar_sim*** folder. To do this, run the following command in the terminal.

Command: git clone https://github.com/autorope/donkeycar

After the donkey car framework is downloaded, it is located in the **/donkeycar_sim/donkeycar** folder. Change to the **donkeycar** folder and check out the repository.

To do this, execute the following two commands now.

Command: cd donkeycar

Command: git checkout master

In the next section follows the creation of the virtual environment named ***donkey***.

Setting up the ***Conda*** environment

Now the time has come to set up the virtual ***Miniconda3*** environment ***donkey in*** which you will then install the donkey car framework. By downloading the donkey car framework from the Git repository, the necessary installation files have already been downloaded. Now execute the following command e.g. in the **/donkeycar_sim/donkeycar** folder depending on where you downloaded the framework.

Note: There are different installation files for Linux and MAC. Therefore, the distinction between Linux and MAC follows now.

For Linux:

Command: conda env create -f install/envs/ubuntu.yml

For MAC OS:

Command: conda env create -f install/envs/mac.yml

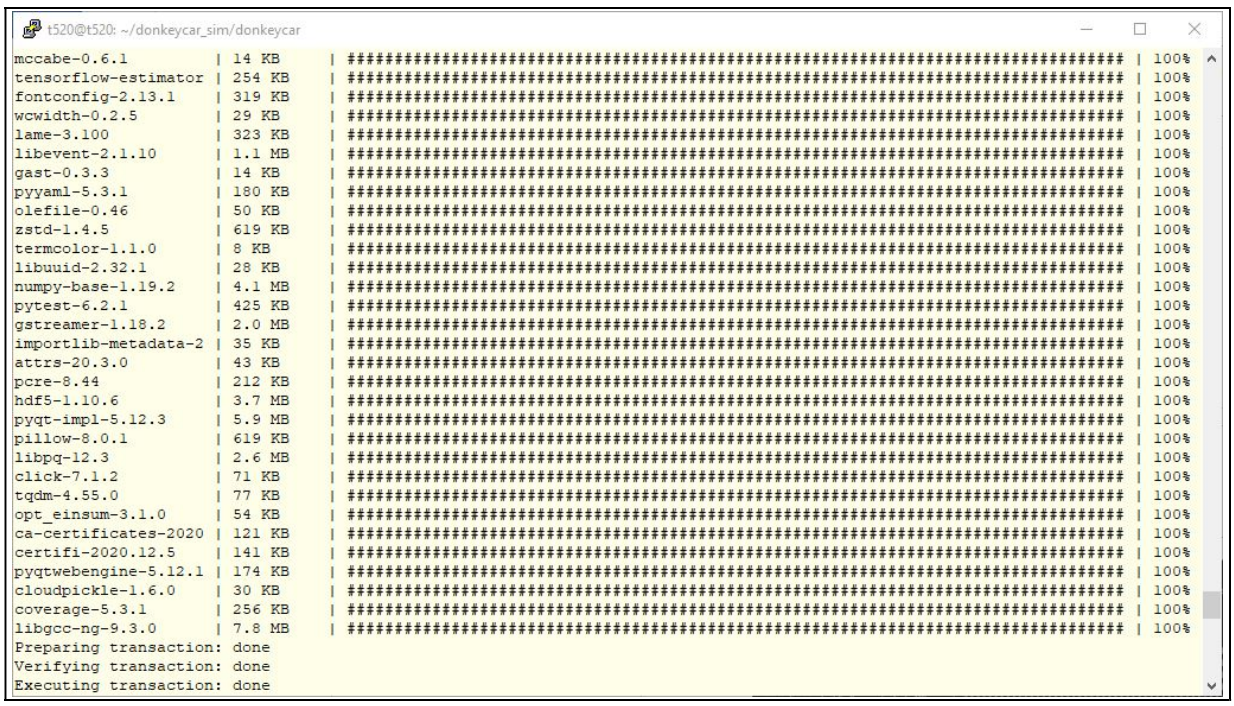


Figure 3.9: Setting up the Conda environment

Since I could test the installation only for Linux I would like to give them here nevertheless a reference for the case you many errors and warnings are indicated. With me it then in this case always so that the virtual environment with the name donkey could not be activated, because this was not created. The solution should be the same for all operating systems.

Note: If errors appear now stating that selected packages do not match the Conda installation and dependencies cause conflicts then most likely the installation file ubuntu***.yml*** is outdated and you need to check the Git Hub repository for a current version.

With the following command you can list all environments that conda knows and manages.

Command: conda info --envs

You should see the base and donkey environment as shown in the following picture.

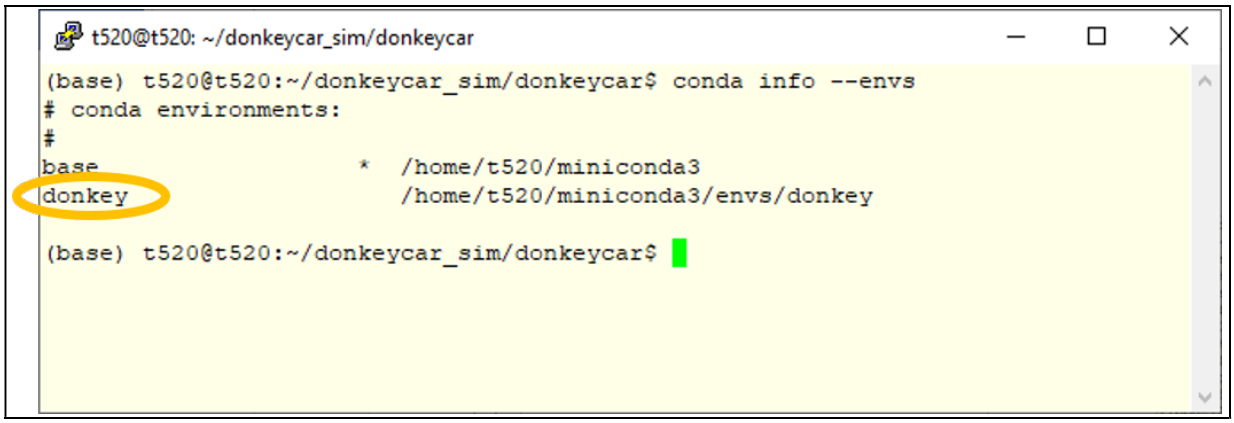


Figure 3.10: List of Conda environments

If the setup of the ***donkey*** environment under ***Miniconda3*** worked then activate it now. Always remember to activate the virtual environment with the name ***donkey*** when you have restarted the computer, for example.

Command: conda activate donkey

In case you also want to remove a virtual environment once then you can use the following command to delete it along with all installed packages.

Command: conda env remove --name myenv

If you were able to activate the virtual ***donkey*** environment then now install the donkey car framework on your PC system using the following command.

Command: pip install -e .[pc]

For a MAC, please enter the following command.

Command: pip install -e .[pc]

The following section is about installing the Donkey Car Simulator framework on the host PC.

3.2.3 Setting up the simulator under Ubuntu

The Donkey Car Simulator framework must first be downloaded again from the Git Hub repository. To do this, go back to the **donkeycar_sim** folder in your home directory.

Then run the following command to clone the gym-donkecar repository to your host PC.

Command: git clone https://github.com/tawnkramer/gym-donkeycar

After the simulator framework has been downloaded, please change to the **gym-donkecar** directory that is now present.

Command: cd gym-donkeycar

If you have not already done so, and to avoid errors, now activate the ***donkey*** virtual environment with the following command.

Command: conda activate donkey

If you are in the virtual environment "donkey" then start the installation of the donkey car simulator framework with the following command.

Command: pip install -e .[gym-donkeycar]

After the installation is finished, initialize a Donkey Car instance of the simulator. To do this, go back to the **donkeycar_sim** folder and execute the following command.

Command: donkey createcar --path ~/mysim

This command would create the **mysim** folder in your user's home directory. In this folder you will find various files like the ***myconfig.py*** file as well as folders.

Set up Unity environment

Now you need the ***Unity*** environment, which provides the virtual world with various race tracks in which you'll do your laps with the Donkey Car Simulator. To do this, you need to download the Unity environment as a ZIP file. If you would like to know which version of the Unity Simulator environment has been released, you can do so via the following URL.

URL: https://github.com/tawnkramer/gym-donkeycar/releases

At the time I wrote this guide, ***Race Edition v20.11.17 was*** the current version I used for this description.

Now download the appropriate ZIP file for your operating system using the following command.

Linux

Command: wget https://github.com/tawnkramer/gym-donkeycar/releases/download/v20.11.17/DonkeySimLinux.zip

MAC:

Command: wget https://github.com/tawnkramer/gym-donkeycar/releases/download/v20.11.17/DonkeySimMac.zip

After you have downloaded the *.zip file, unzip it in the **donkeycar_sim** directory. Now you have a new folder named **DonkeySimLinux** for the Linux installation. It will be similar at this point for the MAC.

Now run the simulator under the graphical interface of your host PC operating system. This is simply a test to see if everything is working with the simulator. Under Linux and probably also MAC you start this with the following command.

Command: ./donkey_sim.x86_64

Note: If you do not have the rights to execute the file "donkey_sim.x86_64" then change it with the following command.

Command: sudo chmod 777 donkey_sim.x86_64

After you have successfully executed the command to run the simulator, Donkey Car Simulator will start and show you the following interface. You can clearly see the different race tracks that are already available.

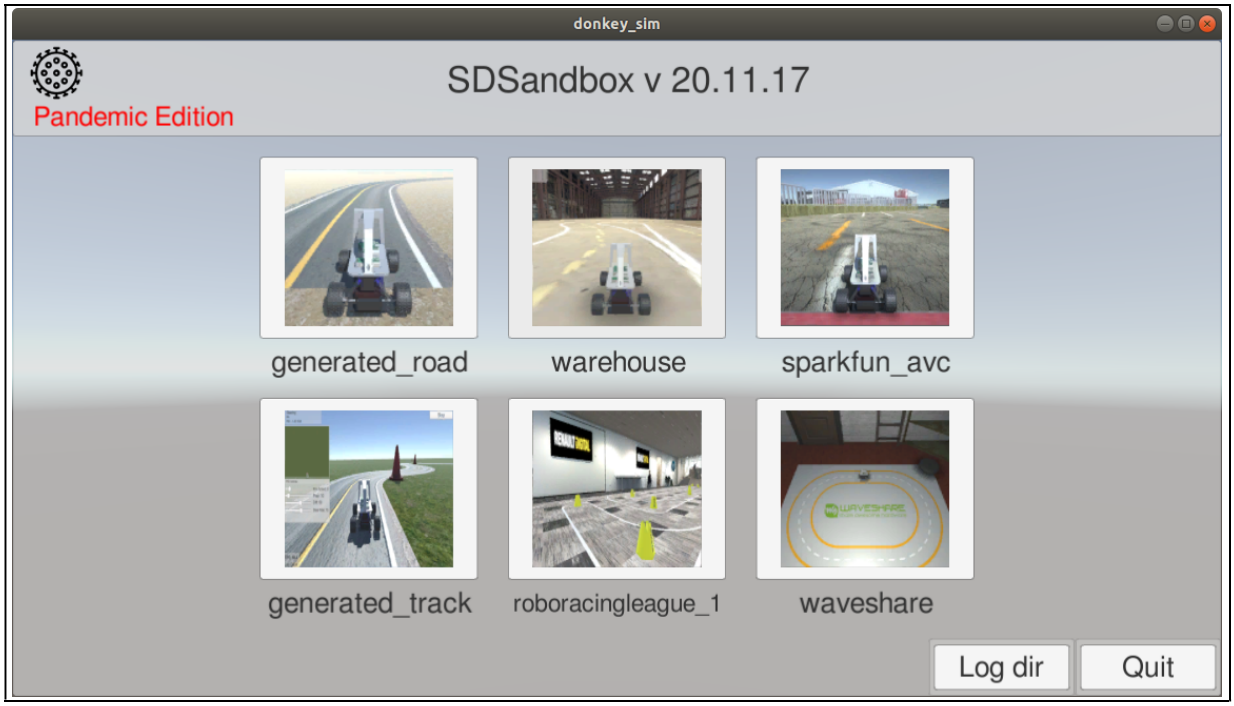


Figure 3.11: Donkey Car Simulator

You can now take a few laps in the simulator to just practice driving.

3.2.4 Configuration of the Donkey Car framework (myconfig.py)

In order for the Donkey Car framework to load the simulator environment when it is started, the ***myconfig.py*** file must be modified. The myconfig.***py file*** can be found in the **mysim** folder where you previously initialized the Donkey Car environment.

Windows:

To edit the ***myconfig.py*** file it is best to use Visual Studio Code.

Ubuntu:

Command: cd ~/mysim

To edit the ***myconfig.py*** file it is best to use the text editor Nano.

Command: nano myconfig.py

If you make several runs with your Donkey Car in the simulator, then it is very good for a later quality control if you can also distinguish the individual runs. This is possible by setting the parameter ***AUTO_CREATE_NEW_TUB*** in ***myconfig.py to*** True. Adjusted, the line should now look like this.

- AUTO_CREATE_NEW_TUB = True

  Note: It is also only possible to exchange training data from different computers or in a group with friends if these are in individual folders and not all collected in one folder.

Next you have to go pretty much to the end of ***myconfig.py*** to find the following lines. You need to enable them accordingly by removing the # presented. These lines ensure that the Donkey Car Simulator environment consisting of the Unity simulator is started when you run the Donkey Car framework from the terminal window.

DONKEY_GYM = True

Windows specifics:

It is important that you store the correct path to the Unitiy Simulator and the EXE file ***donkey_sim.exe. The simulator*** or the Unity environment is started via the ***myconfig.py*** file. You have already downloaded the simulator and unpacked the ZIP file of the simulator.

- DONKEY_SIM_PATH = **"D:\\DonkeySimWin\\donkey_sim.exe"**
- **Note: Be sure to** use double \ when specifying paths on Windows.

Ubuntu specifics:

Be sure to enter the correct path to your Unity Simulator at the <user name> placeholder.

DONKEY_SIM_PATH = "/home/<user name>/donkeycar_sim/DonkeySimLinux/donkey_sim.x86_64"

Select race tracks

There are different race tracks available. You can simply insert all of them here and activate or deactivate them by setting the #.

- #DONKEY_GYM_ENV_NAME = "donkey-generated-track-v0"
- DONKEY_GYM_ENV_NAME = "donkey-warehouse-v0"
- #DONKEY_GYM_ENV_NAME = "donkey-generated-roads-v0"
- #DONKEY_GYM_ENV_NAME = "donkey-avc-sparkfun-v0"
- #DONKEY_GYM_ENV_NAME = "donkey-roboracingleague-track-v0"
- #DONKEY_GYM_ENV_NAME = "donkey-waveshare-v0"

The customized and inserted lines should now be identical for you as shown in the following image for ubuntu except for the <User Name>.

Figure 3.12: myconfig.py simulator customization

If you want the Donkey Car in the simulator to have your name and color, you can add the following lines directly under the list of race tracks or adjust them if they already exist.

- GYM_CONF = { "body_style" : "donkey", "body_rgb" : (185, 240, 5), "car_name" : "Brutus", "font_size" : 100} # body style(donkey|bare|car01) body rgb 0-255
- GYM_CONF["racer_name"] = "Brutus"
- GYM_CONF["country"] = "Germany"
- GYM_CONF["bio"] = "I race robots."
- AI_THROTTLE_MULT=1.0

If you take over the lines 1:1 your Donkey Car will be named "Brutus" and the color will be a kind of green like the following picture from the Unity Simulator shows.

Figure 3.13: Individual adjustment in the simulator

Joystick configuration Windows

You have various options for the control. Just try out which one works best for you. Again, I use my ***Sony PS4*** controller to control the Donkey Car via the open browser window.

In the image below I have highlighted the button with an orange colored squiggle that allows you to control the Doneky Car with a controller like a Sony PS4 controller in the Chrome browser.

> **Note:** Unfortunately, I have not yet found a way to use a PS4 controller directly to control the Donkey Car via the ***Anaconda*** environment. The only way out is to start the web interface in e.g. Chrome Browser and then use it to control the Donkey Car with a hardware controller. Here it is also possible to use the PS4 controller.

While you are now making your rounds, the first training data is already being recorded. You can find them in the folder **mysim/data/images**. More about what makes good training data and how to train the neural network with this data can be found in chapter 3.3".

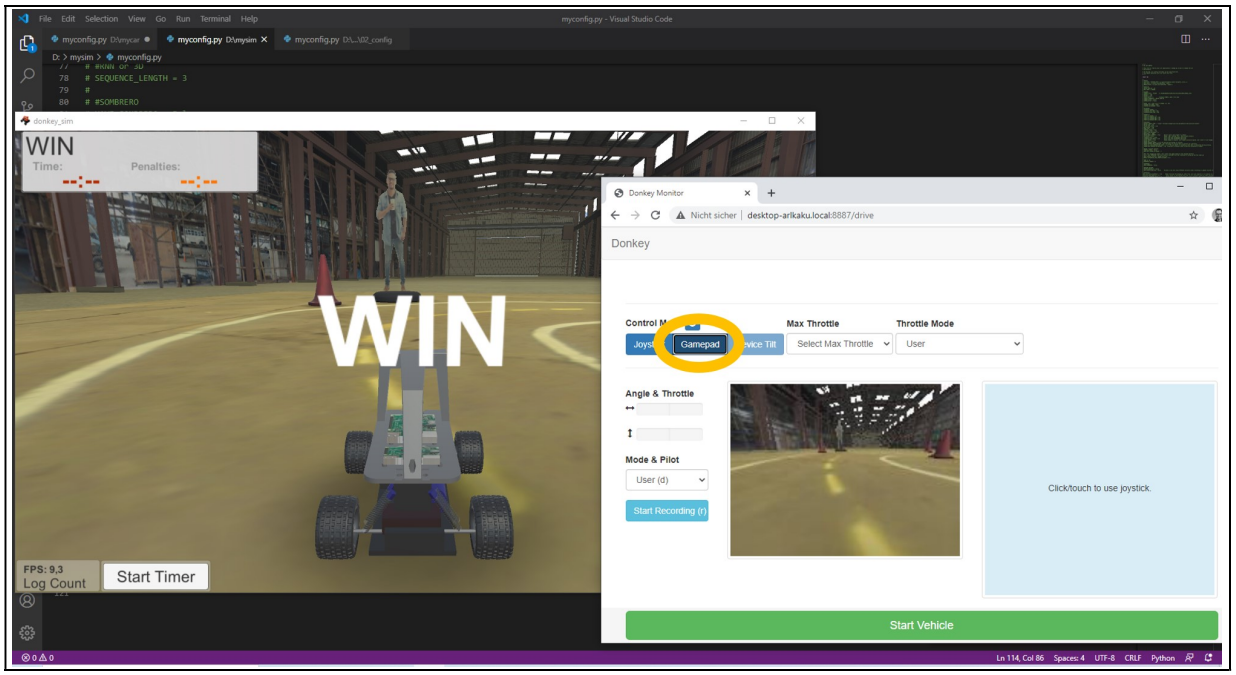


Figure 3.14: Controlling the Donkey Car via the web interface

Joystick configuration Ubuntu

If you prefer to use a joystick or a gamepad on the host PC to control the Donkey Car in the simulator, you have to modify the myconfig.py file accordingly. How to do this has already been explained in chapter 3.4.1

The following image shows as a small reminder the adjustments that need to be made.

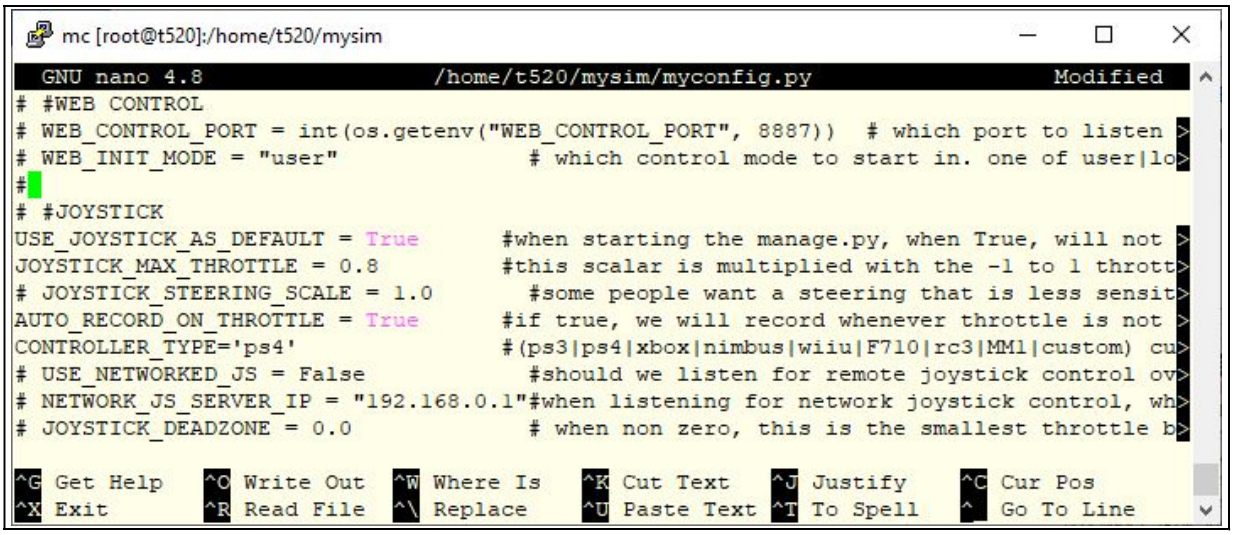


Figure 3.15: Customization myconfig.py

Now everything is prepared so far in the myconfig.py. Please save the changes. Next you can do a few laps and test all settings. The first training data will already be recorded.

If you want to drive a few laps with your Donkey Car in the simulator, execute the following command.

Command: python manage.py drive

Also the direct support of your controller should work now. If not or if you think that keys are assigned strangely, you can learn more about the general configuration of a controller in the following section.

Note: On my laptop the image is always upside down when I run the manage.py drive command. Remedy is the following command executed in an extra terminal Fesnter.

Command: xrandr -o normal

3.2.5 Gamepad configuration under Ubuntu

Many of the settings that affect the gamepad as a controller can be adjusted directly via the Donkey Car framework. These kinds of adjustments are made once in the ***myconfig.py*** file as well as the ***controller.py*** file. The "myconfig.py" file can be found in the directory where you have instantiated the Donkey Car Framework. For the simulator this is the folder ~/mysim. The ***controller.py*** file can be found in the folder ~/donkeycar_sim/donkeycar/donkeycar/parts, depending on the paths you chose for your installation.

Adjustment of the steering input

If you want to swap the steering direction on your controller (left / right) then open the ***myconfig.py*** file of your project and search for the following line.

JOYSTICK_STEERING_SCALE = 1.0

Instead of the 1.0 you now write a -1.0 and remove the comment at the beginning of the line. As a result, the adjusted line now looks like this and the steering should now work mirror-inverted.

JOYSTICK_STEERING_SCALE = -1.0

Save the ***myconfig.py*** file and test if you can steer your Donkey Car better now.

Note: If your Donkey Car steers too fast or too strongly, you can also enter 0.5 here instead of 1.0. The steering will then be much softer and no longer as jagged.

Example: JOYSTICK_STEERING_SCALE = 0.5

Adjustment of the acceleration input

I would like to explain the adjustment or change of the input for the acceleration in more detail below. As far as my PS4 gamepad required changes in the control, I was able to make them via software. If you want to change the assignment on your gamepad then you have to change the assignment in the controller.*py* file. The ***controller.py*** file can be found in the path **~/donkeycar_sim/donkeycar/donkeycar/parts/,** depending on the installation path of the **Donkey** Car Framework. In the ***controller.py*** file for e.g. a PS4 gamepad, search for the following class "PS4Joystick(Joystick)" to change the button and axis messages on the PS4 gamepad. In the ***controller.py*** file you will find at the beginning the mapping to the buttons and axes of e.g. the PS4 gamepad.

If you are not sure which ID belongs to which key or axis, you can find out with the following small Python program called Joystick Tester. I have made the Python program available for download as a *.ZIP file on my blog.

Download: *https://custom-build-robots.com/jetson-nano-download-de*

Share button customization (change driving mode)

I still had the problem that when the donkey car should drive autonomously in the simulator but also in the real model robot car the switch to change the driving mode (user, local angel and local) did not work. I had to change the technical ID of the share button in the "controller.py" file so that I could change the user modes with the share button.

You can do this customization in the class class PS4Joystick(Joystick):. Look for the share button in the definition of the class and you should see the following line.

0x138 : 'share',

Modify them as follows:

0x13a : 'share',

Note that a few lines above you now have a double assignment for 0x13a. Therefore change the assignment for L3 also to the now free assignment 0x138.

The finished result of the adjustments should now look like this.

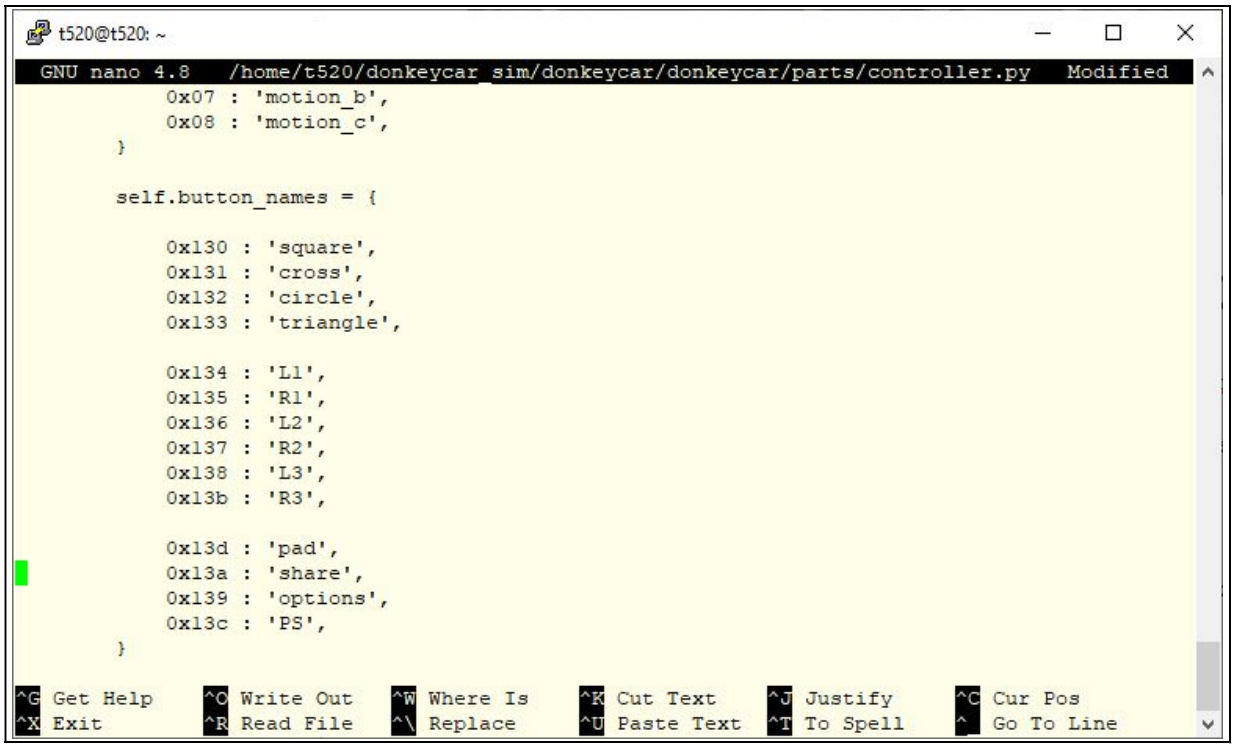


Figure 3.16: Adjusting the assignment of the PS4 buttons

After this change of the joystick control it is possible to accelerate the donkey car with the right lever and to change the mode with the share button. Changing the mode will be important a little later when the donkey car is supposed to drive autonomously in the simulator.

Then it's finally time to do a few laps with the Donkey Car in the simulator and record training data. I drove about 8 laps or recorded 16,000 records. With this training data I trained my neural network afterwards. The result of the final trained neural network was somewhat mixed but ½ lap mostly did it without much error.

The so recorded training data can be found in the folder **~/mysim/data/** and there in different tub folders per run.

> **Note:** When recording the training data, it is important that these are only written when the donkey car is also accelerated a little. This means when you drive into a curve you must not bring the right joystick into the neutral position. If you do this, no data record will be written during cornering and you will miss important training data for cornering.

Also pay attention to the quality of the training data. Training data that you have lost or that are simply bad should also be deleted immediately. The best way to do this is to remove the corresponding tub* folder. For the training of the neural network, the folder **~/mysim/data** should only contain images and json files that you really want to use for the training of the neural network.

The following chapter is only about the recording of good training data and the training of the autopilot, which then steers the donkey car in the simulator.

3.3 From creating training data to autonomous driving

The process is almost always the same as how our robot car is prepared for its use in the simulator or in the real world. First of all, we need to determine how the training data will be generated. In our case we will create them manually by behavioral cloning. But combinations are also possible, e.g. training data that are also partially synthetically generated or training data that are alienated by e.g. the use of filters that make the images brighter or darker. By combining different techniques, a much more stable neural network can be learned in the end. Together we will now teach the robot car or the neural network autonomous driving by Behaviroal Cloning. In principle, the process works as visualized in the following diagram. Within a large loop, three further loops are shown in yellow, blue and green.

- The yellow loop represents the training data generation.
- The blue loop performs the task of preparations such as the definition of the neural network and its training.
- The green loop describes how the completed trained model is transferred to the robot car and evaluated on the racetrack how well it works.

Then the process repeats itself again until the robot car has learned to drive without error.

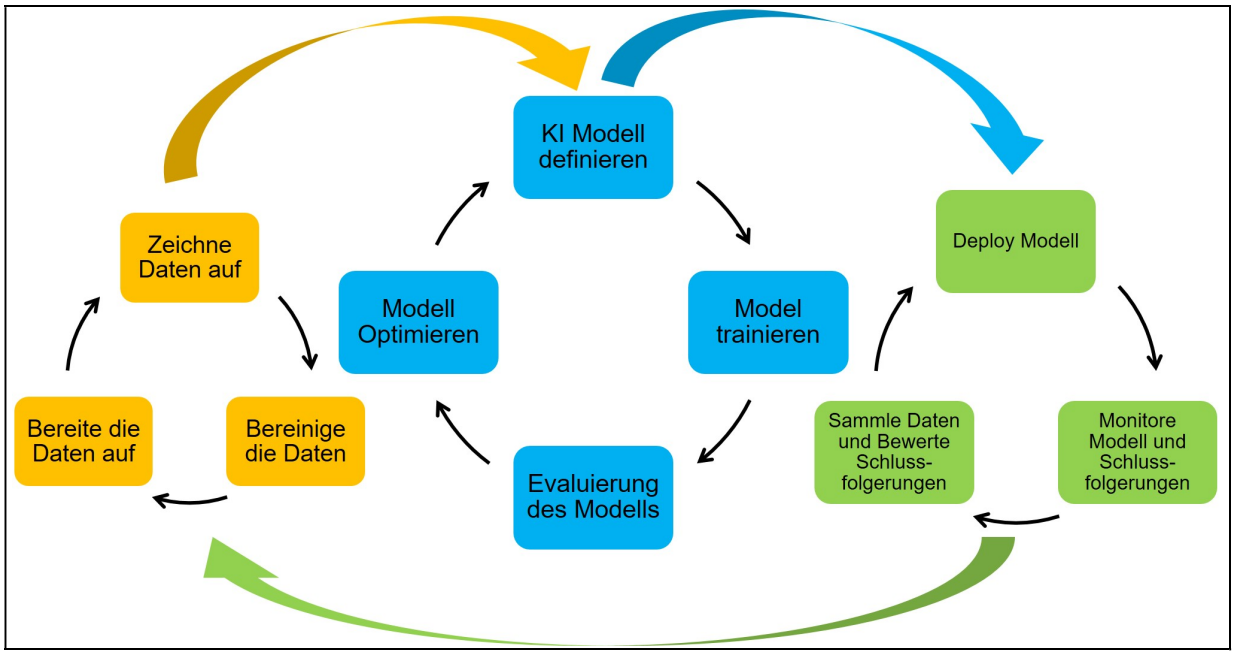


Figure 3.17: AI training and application loop

To give you an idea of the complexity, I have taken out the box "Record data" and presented it in more detail below. In ***Behaviroal Cloning,*** the human specifies how the robot car should behave. The input of the steering angle and the speed is recorded via the joystick and stored together with an image. This process is repeated about 30 times a second depending on the configuration in ***myconfig.py***. The so-called drive loop looks in detail as follows.

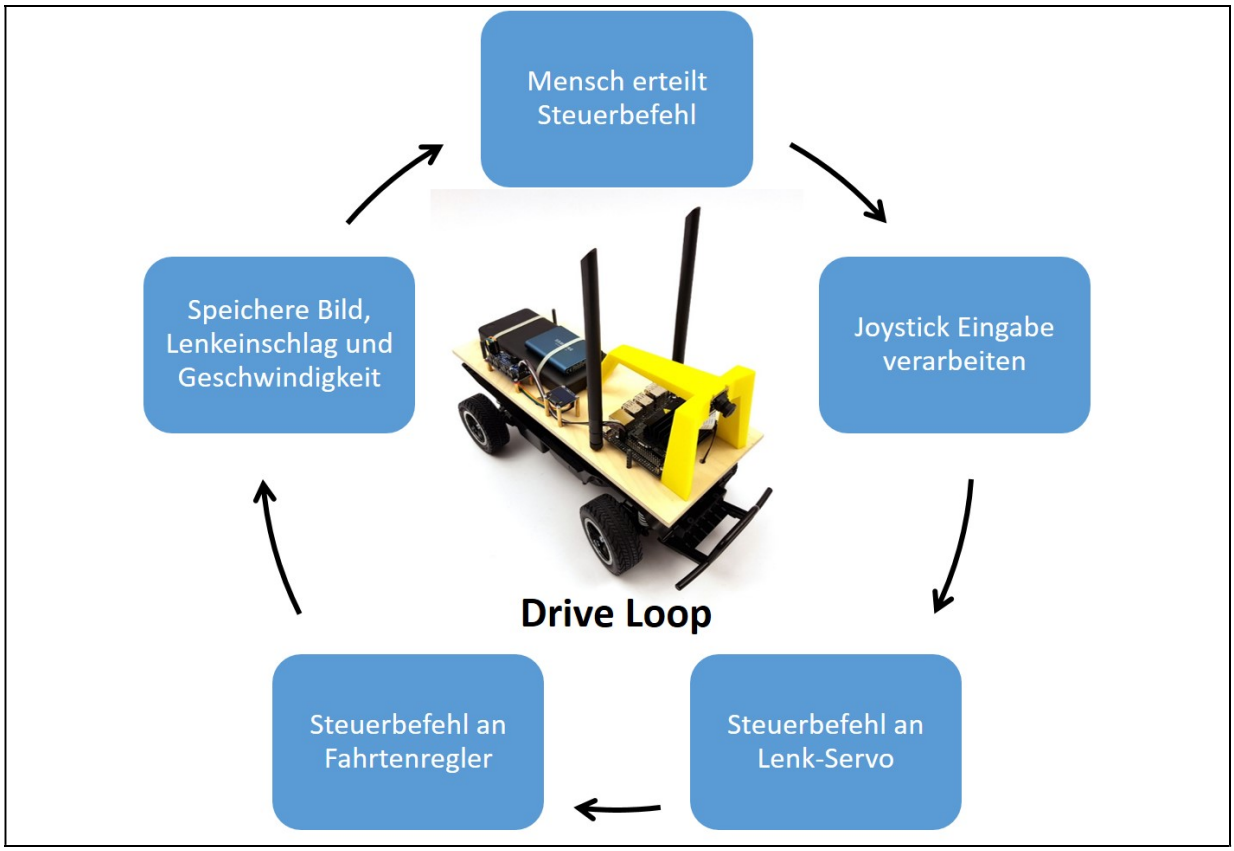


Figure 3.18: Drive loop

But now it's enough with the gray theory and you better start the training data recording in the simulator right away.

3.3.1 Record training data in the simulator

From now on there are no big differences between the Linux world and the Windows world. Therefore, in case there is a difference, I will make it visible in the text. To start the recording of the training data change to the folder **mysim** where the customized ***myconfig.py*** file is located. Now execute the following command in your active ***donkey*** Conda environment.

Note: Remember, if not already done now to start the virtual Conda environment ***donkey.***

Command: python manage.py drive

The Unity Simulator should open after a few seconds, load the race track configured in ***myconfig.py*** and place the Donkey Car in front of the finish line. After a few seconds you should again be able to control the Donkey Car with the joystick. Now turn the first laps on the race track and thus create the training data.

Figure 3.19: Donkey Car Simulator live

Now it can happen that you can't accelerate the Donkey Car with the right joystick and maybe the left and right steering is swapped. If you want to customize your gamepad, you can read in the chapter "Gamepad configuration under Ubuntu " how this works in detail.

3.3.2 Train the neural network Windows

With the current Donkey Car version, a *.json file is no longer created for each *.jpg file that stores the steering and acceleration values for the individual image. Instead, a ***catalog file*** is created in a folder above the **images** folder that stores exactly this information per image.

Once you have created enough training data around the 8,000 images then you can use the following command to exit the Donkey Car framework in the Conda prompt.

Command: **CTRL + C**

Now familiarize yourself with the recorded training data and take a look at the recorded images at your leisure. The steering and acceleration values corresponding to the images are stored in the ***catalog*** file one level above the **images** folder.

If you want to train the neural network now, please execute the following command if all images are in one folder.

Command: python train.py --tubs=data --model models/mypilot.h5

If you have several folders in which the images for training the neural network are located, the command looks as follows. The individual folders that are to be used for training are clearly visible.

Command: python train.py --tubs=data/tub_test/,data/tub_2_21-01-16/,data/tub_1_21-01-16/,data/tub_3_21-01-16/ --model models/big.h5

The training should now start and you will see in the terminal window how ***TensorFlow*** splits the training data into a set for training and a set for validation.

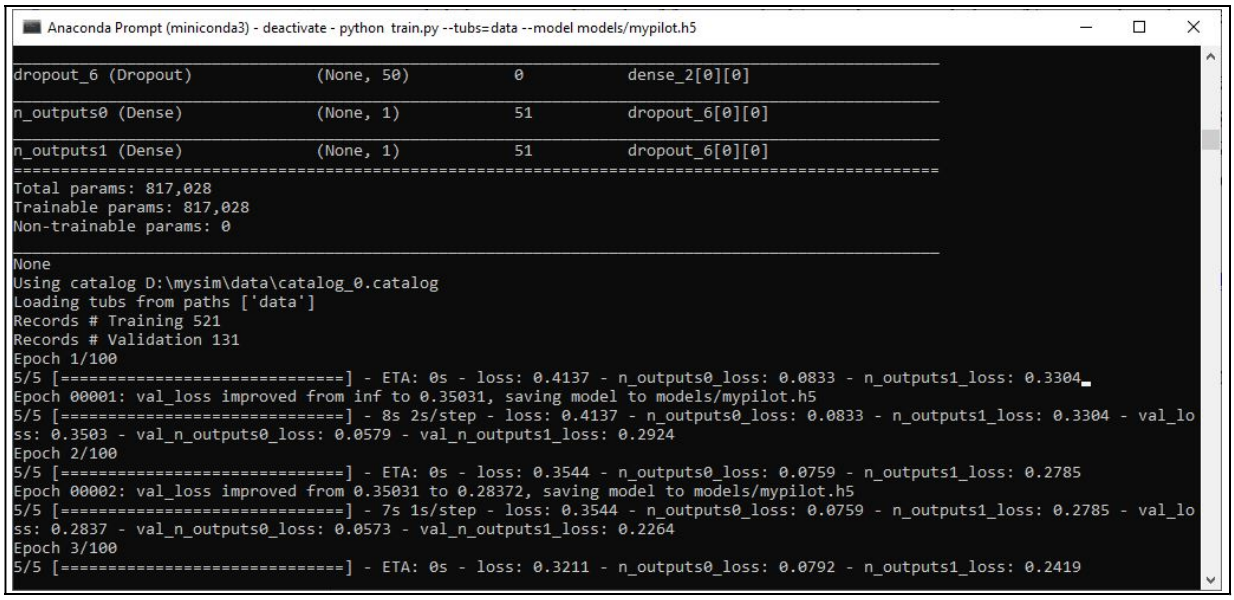


Figure 3.20: Output TensorFlow training

The trained neural network will be saved with the name ***mypilot.h5*** in the **models** folder when the training is finished.

3.3.3 Train the neural network Ubuntu

In the following I will explain how to train the neural network with the training data you recorded in the simulator. To do this, exit the simulator and the running Donkey Car framework in the terminal window with the following command.

Command: CTRL + C

The recorded training data, i.e. all images and associated catalog files, can be found in the corresponding **images** folders per run in the **~/mysim/data/ folder**. You might want to do a quality control and delete folders with training data that are not so good.

Now use the following command to train a neural network. Execute the command in the terminal window on your host PC.

Command: python train.py --tubs=data --model models/mypilot.h5

If you have several folders in which the images for training the neural network are located, the command looks as follows. The individual folders that are to be used for training are clearly visible.

Command: python train.py --tubs=data/tub_test/,data/tub_2_21-01-16/,data/tub_1_21-01-16/,data/tub_3_21-01-16/ --model models/big.h5

Depending on the performance of the host PC and the amount of training data, the training of the neural network can be completed quickly or take some time.

The following image shows the output in the terminal while training a neural network for the Donkey Car.

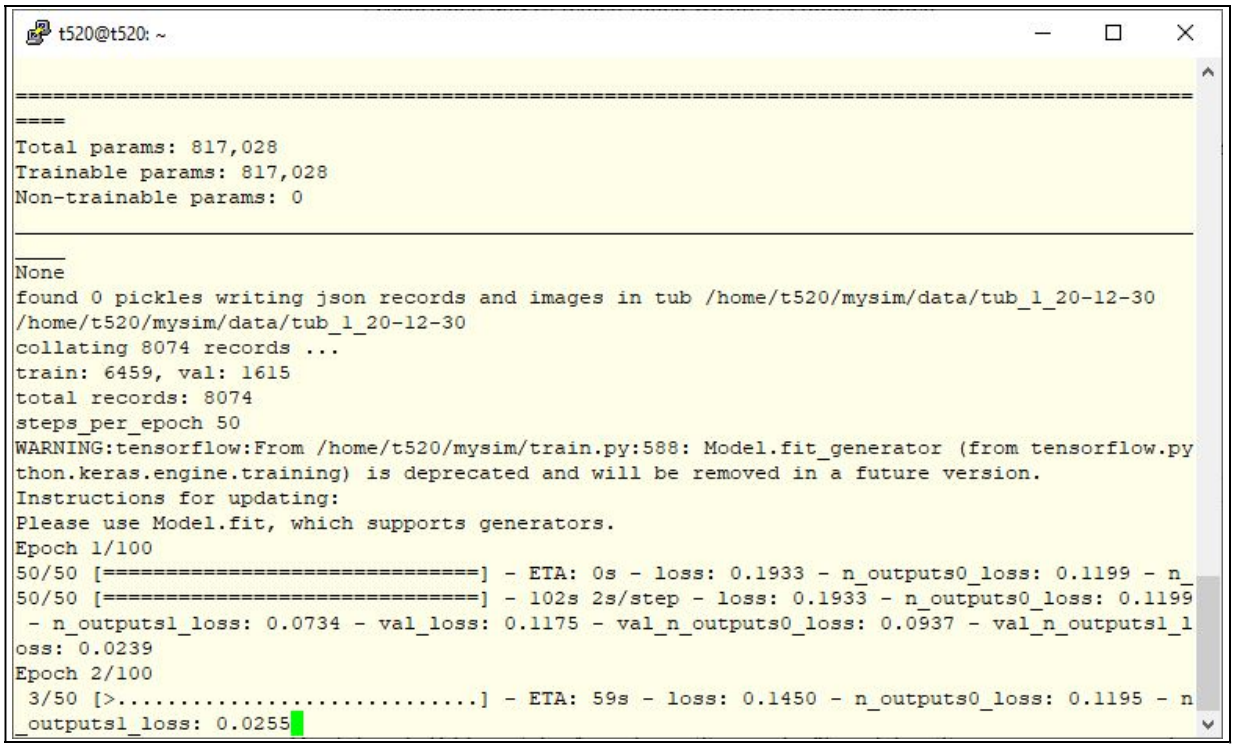


Figure 3.21: TensorFlwo output during training

After the training has been completed, it is still displayed how the training as such has proceeded. The graph can be used to draw conclusions or comparisons between different training runs and training data. The picture shows such a graph and how well or fast the network has learned.

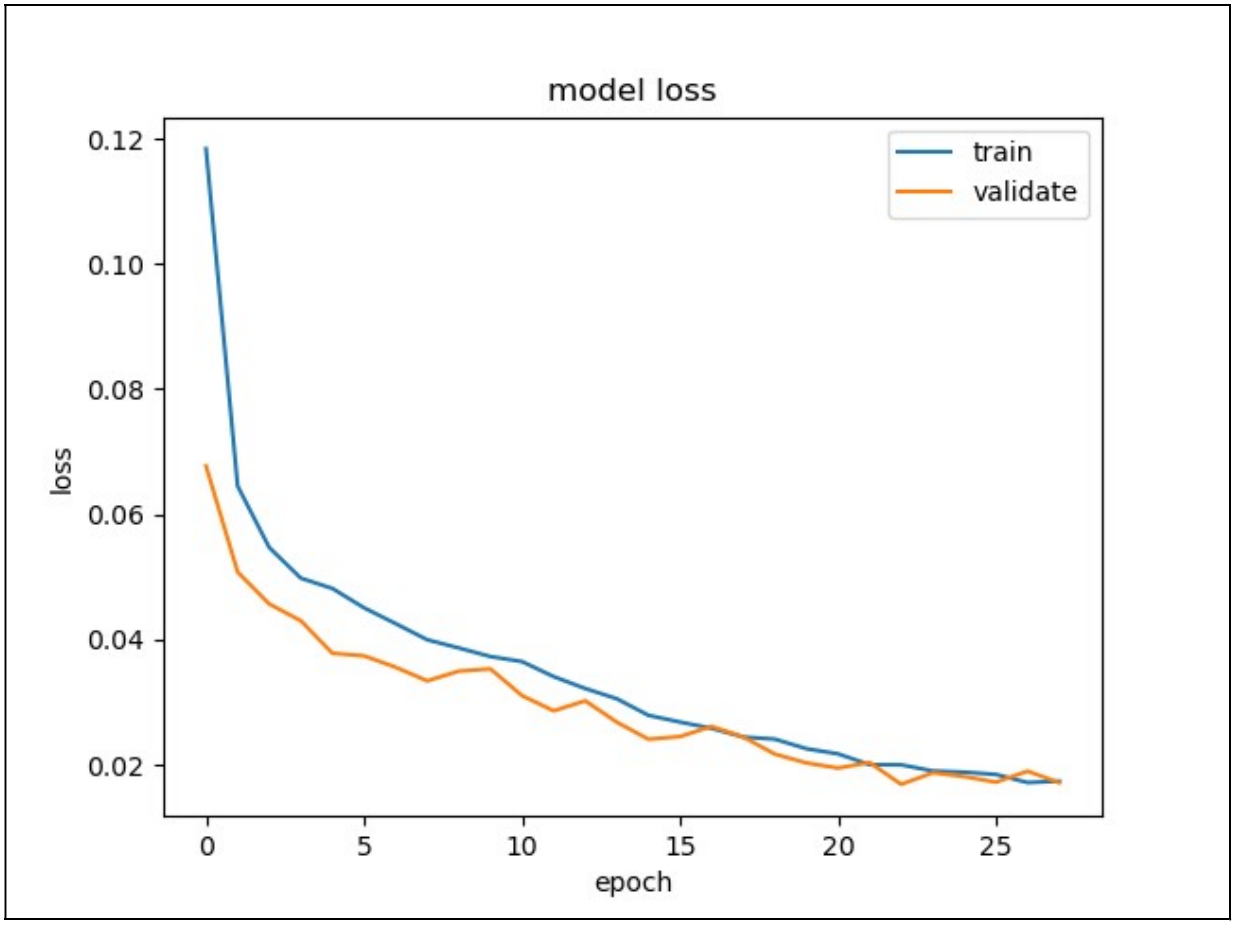


Figure 3.22: Representation of the training course

3.3.4 Test the finished neural network

Now it is time to check how well the trained neural network is able to steer the donkey car around the course in the simulator. To test the finished neural network, execute the following command in the terminal window. The neural net called in the following command is named mypilot.h5. When you run the command, it loads the simulator environment and places the donkey car in front of the finish line, but the donkey car does not start yet.

Windows:

Command: python manage.py drive --model models/mypilot.h5

Ubuntu / MAC:

Command: python ~/mysim/manage.py drive --model models/mypilot.h5

Control under Windows:

Open a browser again and log in to the Donkey Car Framework web interface. You had already used this interface to control the training data recording the Donkey Car in the simulator. The URL is structured as follows:

URL: <IP address of your host PC>:8887

Switch to the "Local Angle" mode and let the neural network steer your donkey car. You can set the speed yourself in the right field with the mouse. In the Conda Promt in which you have executed the Donkey Car framework, you can see which mode is currently active. If everything goes well now, your Donkey Car should be able to drive a few laps without any errors.

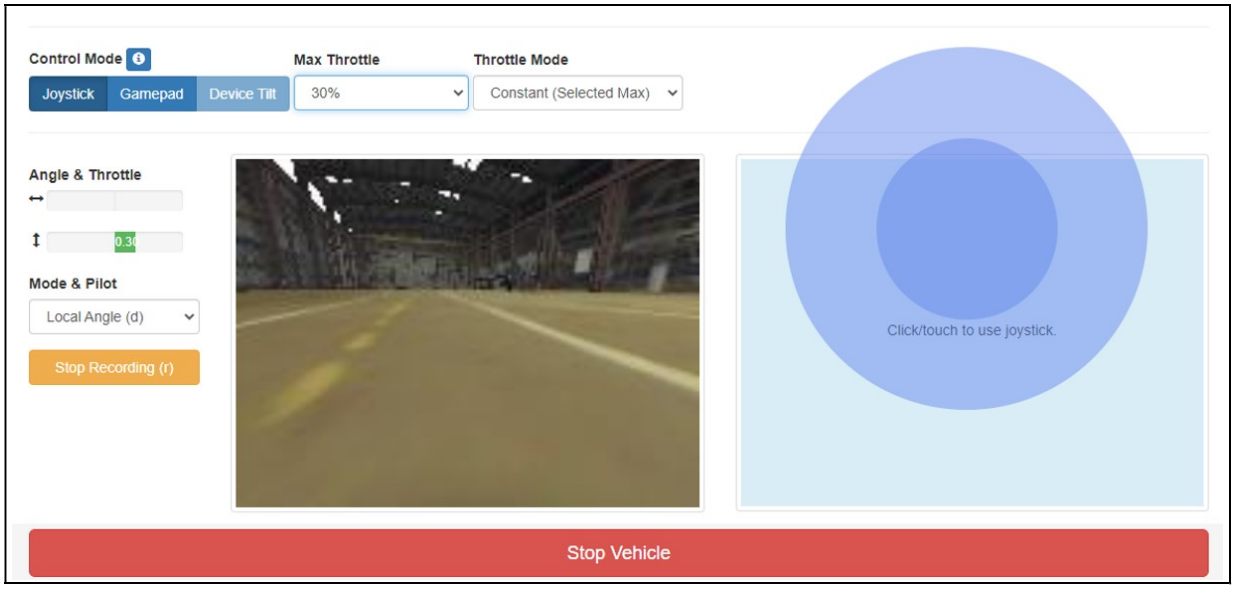


Figure 3.23: Donkey Car Framework web control

Next, you could change the modes to ***Local Pilot*** and the Donkey Car should drive around the track completely autonomously.

Control under Ubuntu:

Now press e.g. the Share button 1x on your PS4 controller to switch the driving modes. In the terminal output you can see which mode you currently have active. You should select the mode local_angle so that your Doneky Car steers by itself in the simulator and you set the speed with the controller. When the neural network is able to steer the donkey car around the course you can switch to local mode by pressing the share button again. Now the donkey car should turn its rounds completely autonomously.

If everything has worked up to this point, the Donkey Car should hopefully complete its laps without error. If not, record additional training data and train a new autopilot that will hopefully work better on the track.

3.4 Race in the simulator

It is also possible to run several Donkey Cars or neural networks simultaneously in the simulator. Thus, for example, different teams can let their neural networks compete against each other. But not only for competitions the parallel execution of neural networks is interesting but also for the generation of special training data. Such special training data could include, for example, data on swerving and overtaking. But it would also be possible to collect data to e.g. push other cars off the short in a race. But you can guess that creating such special training data is not that easy.

The following picture shows two autopilots trained with different training data competing against each other in a race.

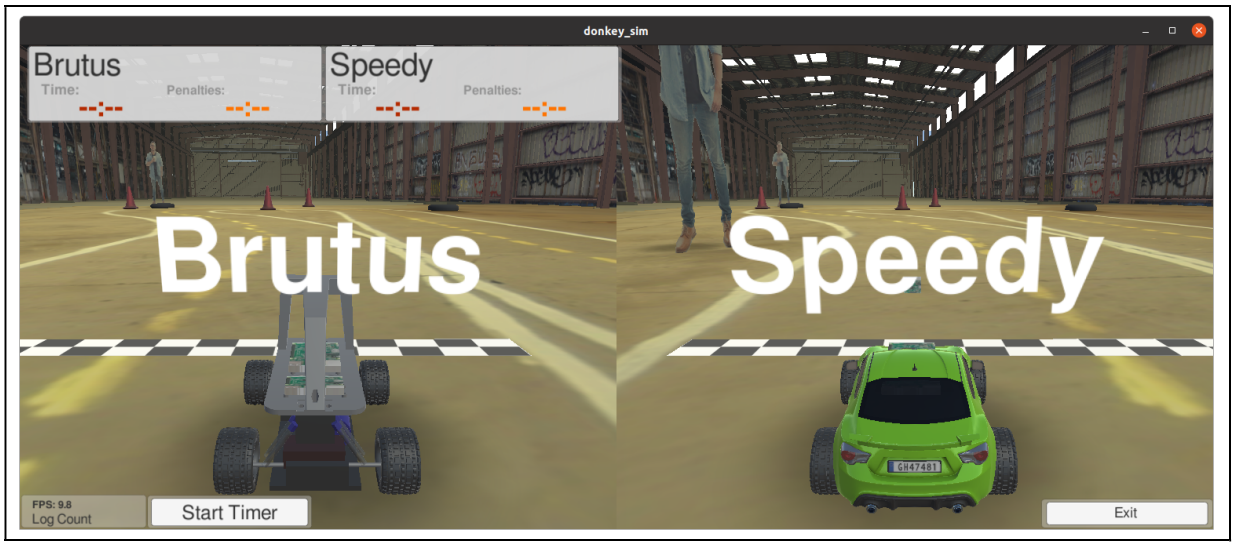


Figure 3.24: Racing in the simulator

It is possible to start several autopilots on one computer which then log on to the Unity Engine or remotely via the Internet. Thus, for example, participants or teams can test their networks worldwide. A challenge that must be taken into account is certainly the latency in communication when the computers are far away from each other.

In the following section I will go into detail about the ***myconfig.py*** configuration file.

3.4.1 Customization of myconfig.py

In order to load multiple autopilots or neural networks in a simulator at the same time, a few settings have to be made in ***myconfig.py.***

WEB CONTROL

This adjustment is only important for you if you start several neural networks at the same time on one and the same computer so that they log on to the simulator. Now you have always called a URL with port for starting the web interface for controlling the Donkey Car. This URL had the following structure:

URL: <IP address of host PC>:8887

If you now start several instances of the Donkey Car Framework on one computer, the port must be different in order to be able to distinguish between the individual instances when calling the URL.

To be able to assign different ports per instance you have to create a copy of the ***myconfig.py*** file with its own name e.g. ***myconfig_port.py.***

If you have created the copy ***myconfig_port.py*** then please open it and search for the following line.

#WEB CONTROL

One line down now adjust the port to e.g. 8889. The line should then look like this.

WEB_CONTROL_PORT = int(os.getenv("WEB_CONTROL_PORT", 8889))

One line below you can still influence in which mode the web interface should start or the neural network should start.

If you want the Donkey Car to start driving immediately after loading and logging into the simulator, change the mode from ***user*** to ***local.***

The line should then look like this.

WEB_INIT_MODE = "local"

DonkeyGym

Now follow two adjustments to call the simulator or the Unity Engine. Look for the following line that starts the Unity Engine. For example, this looks like the following for Windows.

DONKEY_SIM_PATH = "D:\\DonkeySimWin\\donkey_sim.exe"

Modify this line as shown below and replace the path to Unity Engine with ***remote***. As a result, the line should now look like the following.

DONKEY_SIM_PATH = "remote".

With these adjustments, you can now launch multiple autopilots on one PC and have them run in Unity Simulator.

Start simulator from n computer

For example, if you want to participate in a race with your autopilot over the local network or the Internet, you must enter the remote address of the computer hosting Unity Simulator in the DonkeyGym section.

To do this, search for the following line and enter the IP address or server name of the target computer with the simulator.

SIM_HOST = "<address of the destination computer> "

At the beginning of this section, I briefly discussed the problem with latency when the participating machines are farther away from the host that provides the simulator. To compensate the latency problems in the delay of the control you can set the following parameter. Here it is worth to test and play a bit how the setting works best in milliseconds.

SIM_ARTIFICIAL_LATENCY = 0

Now you know both possibilities how to configure your Donkey Car instance via ***myconfig.py,*** e.g. to start several instances locally on one computer or to race remotely with several Donkey Cars in one simulator.

3.4.2 Train overtaking - record training data

An exciting topic is the collection of training data to teach an autopilot to overtake other donkey cars. First of all, you need an autopilot that will turn many laps at a comfortable speed on the race track in a stable way without any errors. Now run this autopilot several times so that e.g. 2 or 3 cars are driving on the track at the same time. Start the autopilots in such a way that there is always a sufficient distance between the donkey cars and they do not immediately bump into each other.

Now let's just hope that these 2 to 3 Donkey Cars don't crash and you can record training data with another one that they control themselves now. Now just try to overtake the slow moving Donkey Cars or drive behind them without bumping into them. Record about 20 overtaking maneuvers and then train the neural network. When you test this neural network on the race track you should see that it can overtake other donkey cars or that it tries to overtake the other donkey cars.

For sure, the creation of such special training data is time-consuming and data-intensive. Maybe you have a better approach than the one I have outlined here.

3.5 What you have achieved so far

You have successfully installed the Donkey Car Simulator environment on your PC and are now able to generate training data on your PC. You have also trained your neural network and tested it on the race track. You could try all this from the comfort of your desk. You didn't have to go out into the garden or to a big parking lot to do it. This is the great advantage of having a simulator for training neural networks. You can now try different things on the basis of the recorded training data to make your neural network more stable so that the donkey car reaches the finish line safely. The knowledge gained in this way will be partially transferable to the real world, i.e. to your real donkey car, the construction of which will be dealt with in the following chapters.

Personally, I still find it fascinating how differently a neural network behaves under real environmental conditions and what surprises the artificial intelligence in the form of the neural network has in store. But you can experience all this for yourself when your real model robot car takes off.

I very much appreciate your feedback on this chapter. Write me your thoughts, questions and suggestions to the following e-mail address: *ebook@custom-build-robots.com*

4 Build your own Donkey Car - introduction to the required components

You already know your way around the software side very well. Now you will dive into the world of the hardware of a robot car. The choice of the right chassis and suitable components is crucial for the success of this project. Therefore, the chapter that follows now deals in detail with the electronics and hardware of the real Donkey Car.

You will teach the robot car we are building together how to drive autonomously through Behavioral Cloning. You have already learned the concept and procedure behind Behavioral Cloning in the simulator. Don't worry RC model building will not be the main focus of this project. Rather, you will focus on the software and training of your neural network or autopilot and build your knowledge here. Together with the converted model car, you will gain valuable experience in the field of Artificial Intelligence (AI), which will help you to assess the complexity and challenges of other problems later on.

You don't need to be able to program for this project but will use the mature Donkey Car Frameworks and just install and configure it. During the configuration I will give you the exact program lines to change.

The Donkey Car Framework was designed by Will Roscoe and Adam Conway at the end of 2016 and enables an easy entry into the world of autonomous driving model cars. Thanks to the great popularity and the large community, this framework is rapidly evolving and is now one of the most stable as well as mature open source solutions. The Donkey Car project relies on classic RC model car technology that forms the mechanical basis of the robot car. Thus, you will only deal with the topic of model building at the beginning and concentrate on the topic of AI and collecting training data after the assembly. So you will learn practically what it means the data are the new oil that will drive us and bring us further.

The Jetson Nano, which is the central computing unit of the robot car, and the neural network that runs on it must first be trained. For this purpose, you will generate the required training data yourself by manually driving the model car. In the world of artificial intelligence this kind of training of neural networks is called behavioral cloning as you have already learned at the beginning. It becomes exciting when you recognize your personal driving style in your self-trained model and thus get a first feeling for the human bias that you have casually given your model through the training data you have recorded. A human bias is an unconscious behavioral pattern that you have adopted in the course of your life and which is always mentioned when neuronal networks are built up and trained, for example, only by Western-influenced and male scientists.

4.1 Hardware selection

For the construction of the robot car you need different hardware and electronic components which are explained in detail in this chapter. I made sure that the costs for the construction of the robot car are as low as possible. Not everything must necessarily be bought new. For example, I suggest using a used Tamiya model car chassis for the robot car. There is a wide range on the Internet with some hardly used model cars that are already offered with battery and charger. I will dive deeper into the details in the individual sections on the components why which component was chosen by me.

To give you a general overview of the necessary components for the construction of the robot car, I have compiled them for you in the following table.

Description:	**Price in €:**
Jetson Nano 4GB Ram as central processing unit	109,-
Camera module with 160° wide angle lens	28,-
Micro SD card 32 GB	7,-

SSD hard disk 250 GB (optional)	75
Fan 5V for Jetson Nano with PWM control	15,-
WIFI - Intel AC8265 Wireless NIC Module	25,-
Power Bank with 5V and about 5A	64,-
RC battery with 7.2V and 4,000 to 5,000 mAh	30,-
RC Battery Charger	60,-
External power supply for stationary operation	30,-
Tamiya chassis for the robot car (used)	140,-
Wooden base plate	4,-
3D Printing Roll Bar	38,-
Brass spacer	11,-
Adafruit PCA9685 Servo Controller	19,-
Gamepad for controlling the robot car	26,-
OLED display	4,-
Miscellaneous cables	6,-
RC battery voltage indicator	8,-

Table 4.1Component list

According to this overview with all its components you need for the construction of the robot car the costs amount to approx. 700,- €. But for that this robot car is then usable outdoors, very stable in case the neural network behaves surprisingly and the spare parts supply in case of a Tamiya chassis is very good in Germany.

A list of components with links to corresponding web stores can be found on my blog at the following URL.

URL: *https://custom-build-robots.com/jetson-nano-komponenten-de*

The following picture shows the main components needed to build and operate the robot car. The model car shown was purchased used on the Internet.

Figure 4.1: Components for the Donkey Car

4.1.1 Central processing unit and accessories

For all robot cars, whether in real life or as a model as in this case, a powerful computing and sensor architecture as well as a mature AI model is a crucial success factor. The acquired sensor data must be evaluated by a neural network or a series of interconnected neural networks in the shortest possible time. This architecture of hardware, software and neural networks in its entirety then makes a prediction as to how the steering angle must be selected at the current speed in order to successfully steer the robot car through a curve, for example. Another neural network in this network then evaluates, for example, right-of-way rules, traffic signs and traffic lights, and in turn intervenes in the speed, for example, to stop the

robot car at a red light. All these decisions have to be made in the shortest possible time by the central computing unit and passed on to the robot car's controller.

NVIDIA Jetson Nano

In this project, the NVIDIA Jetson Nano with 4GB Ram is used because it offers the best GPU computing power for its size and price. Thanks to its Maxwell™ architecture and 128 NVIDIA CUDA® processing units, it offers significantly more performance than the current Raspberry Pi 4 with 4GB RAM when running and especially training neural networks. Since the training of the neural network is very computationally intensive, you would have to wait many hours to days with a Raspberry Pi 4 with 4 GB RAM until the neural network would be completely trained. With the Jetson Nano this training is much faster and a working autopilot that is trained with about 6,000 training data sets is calculated in about 20 minutes.

> **Note:** If you have a modern computer that may also have an NVIDIA GPU installed, the training of the neural network will be completed much faster on this computer than on the Jetson Nano. However, you don't always want to take your laptop or desktop PC with you, so training the neural network directly in the robot car is a good idea.

Camera module

When choosing a camera, it is important that it is compatible with the Jetson Nano. NVIDIA officially supports various camera models, but they are all based on the Sony IMX219 chip. NVIDIA provides the necessary drivers for this chip in the image for the Jetson Nano. A list of compatible cameras can be found on my blog at the URL mentioned at the beginning of this chapter.

I always recommend installing a camera with a wide-angle lens of 150° to 160° in the robot car. This way, the field of view is not too small and captures the route markings on the left and right very well, even when driving through a curve. The field of view of the camera should not be too large (> 160°) in order to capture few external factors off the track, such as moving people or furniture.

Memory - micro SD card

For the Jetson Nano you need a micro SD card on which you install the operating system and later also the software. Therefore, this card should be very fast when reading and writing data. You should choose a 32 GB model so that the SD card does not fill up right away. But please still read the following recommendation on the subject of SSD hard drive as storage instead of the micro SD card of the robot car.

Memory - SSD hard disk

The memory of the robot car is accessed very frequently in read and write mode. Since the neural network is also to be calculated directly on the Jetson Nano. My experience has shown that the classic micro SD cards very quickly show defects in an intensive use as it represents the training of a neural network. Therefore, my clear recommendation is to use an SSD hard drive for running the operating system and also for training the neural network. An SSD that is connected to the USB 3.0 port of the Jetson Nano and is 250 GB to 500 GB in size is completely sufficient.

> **Note:** If you want to start with the micro SD card first, then you can switch to an SSD hard drive and move the data to the SSD later, even if you have already recorded training data and your robot car is already driving autonomously.

WLAN for the Jetson Nano

The Jetson Nano itself does not have a built-in WIFI module. Therefore you have to add a WIFI module to the Jetson Nano first. To get the best driver support, a large signal range and a very good WIFI reception you should install the Intel AC8265 Wireless NIC module as recommended by NVIDIA. Experience at a number of events has shown that there are always problems with disconnections with USB modules that are not significantly cheaper.

Fan for the Jetson Nano

The Jetson Nano quickly gets very hot when training a neural network. Therefore I recommend to mount a small 5V fan on the heat sink. So that the fan is not too loud, e.g. during stationary operation on the desk, it is also sufficient to operate it with 3.3V. If you want you can also buy a PWM controlled fan whose speed can be controlled depending on the temperature of the Jetson Nano.

Energy supply - introduction

The appropriate energy supply of the robot car together with the Jetson Nano is still a crucial point for the success and fun of the project. I have already built about 8 of these robot cars and gained a lot of experience with them at international trainings and lectures around the world. There are two possibilities to implement the energy supply in the robot car. The first one is a RC battery which supplies the Jetson Nano and the drive motor with power. The second possibility is a separate power supply of the Jetson Nano with a power bank and the power supply of the drive motor with a RC battery.

> **Note:** If you use only one RC battery to power the Jetson Nano and the drive motor, the Jetson Nano may crash if the drive motor consumes too much power during acceleration. Therefore I recommend a separate power supply for the Jetson Nano and the drive motor.

Power supply - Jetson Nano

I recommend to connect the Jetson Nano to a power bank when the robot car is running and not stationary e.g. on the desk. If you already have a power bank that supplies 5V and at least 2.0A to 2.5A at one of its outputs, then try it out first to see if the Jetson Nano runs stably with it before you buy a new one. If you find that the Jetson Nano does not run stable under load then I recommend a power bank with about 28,000 mAh, the typical voltage of 5V but an extra strong current of about 5A. I have always had very good experiences with such a power bank in a variety of robot cars based on the Jetson Nano. On my blog you can find a corresponding model in the component list.

Energy supply - drive of the robot car

For the drive motor of the robot car, it is best to use a 4,000 mAh to 5,000 mAh RC battery. Here you can use a LiPo RC battery if you are familiar with this technology or a classic NiMh model that tolerates undervoltage of the battery a little better. For my trainings with up to six of these Donkey Cars I always used NiMh batteries to be allowed to transport the 14 batteries also e.g. in the airplane what is difficult with LiPo batteries.

Power supply - Stationary operation with external 220V power supply unit

An external power supply is highly recommended so that the Jetson Nano can be operated in stationary mode without time restrictions. This should be able to supply the known 5V at approx. 5A. If you later train the neural network on the Jetson Nano directly, the power consumption of the Jetson Nano increases to about 20W. Therefore the power supply should deliver 5A instead of 4A to have some reserves left. Also, the higher power of the Jetson Nano is only called up when it is supplied with energy via the external power supply.

4.1.2 Chassis selection

Since this isn't supposed to be a model building project, I recommend buying a Ready-to-Run (RtR) 1/10th scale model car. These RtR model cars are ready to go and you don't have to deal with model building issues. This is exactly how Waymo does it, they don't develop and build a car themselves but equip existing platforms like the Chrysler Pacifica with sensors and computing units for their purposes and thus convert them into autonomous driving robot cars.

I always recommend Tamiya models with a CC-01 chassis for building a robot car. This is available with different bodies but the chassis is always the same if it is a CC-01 chassis. Well suited but not quite as stable because much sportier is also the Tamiya TT-02 chassis. The Tamiya chassis can be purchased cheaply used on various platforms on the Internet and the CC-01 chassis can withstand many accidents without immediate damage thanks to the stable construction. It will happen to you for sure that your

neural network suddenly drives forward or backward depending on what it has learned at top speed and it comes to an accident with your donkey car. Also important to me is the easy availability of spare parts which is generally given with Tamiya models.

A used chassis - What is important

When buying a used chassis, make sure that it is ready to drive. It should be a steering servo and speed controller installed, wheels included and no major damage. I had the negative experience of buying a used CC-01 chassis with a defective rear right suspension, contrary to the seller's product description. But with a few spare parts I could repair it quite easy and cheap. For the speed controller, please ask the seller for the type designation so that you can also find the instructions for this on the Internet. The manual is important so that you can find out if this speed controller has a BEC (Battery Eliminator Circuit) function and how the forward gear, the motor brake and the reverse gear can be configured. The BEC function is not all that important, but it will save you some effort later when wiring the servo controller and powering the steering servo. Because exactly the power supply of the steering servo takes over the BEC function of the speed controller.

> **Note: The** official Donkey Car site lists compatible models for the various 3D print files, but these are sold in North America and in my experience can only be imported individually in Europe.

Base plate

In order to mount all components like the roll bar, Jetson Nano, servo controller etc. of the robot car on the CC-01 chassis you need the base plate. This base plate will be attached to your chassis instead of the body. I recommend you to buy a multiplex board with a maximum thickness of 6mm. This plate should have approximately the following dimensions:

> **Dimensions:** 360 x 150 x 6 mm

In the hardware store you can have the base plate cut to size for a few euros.

Brass spacers

For mounting the Jetson Nano, the OLED display and the servo controller, I recommend using brass spacers. These are ideal to be screwed into the base plate. You then attach the components to these spacers.

3D Printing - Roll Bar

To ensure that the Jetson Nano is well protected in your robot car, I recommend printing out a roll bar including camera mount with a 3D printer. The required STL files are provided online by the Donkey Car project. There are other STL files for this roll bar like a base plate and an adapter plate for the Jetson Nano. In my experience these plates are not needed because they are not compatible to the CC-01 chassis from Tamiya. Therefore I recommend you to do without these two plates and save the money. The required STL file for the roll bar can be found at the following URL.

> **URL:** *https://www.thingiverse.com/thing:2566276*

4.1.3 Servo controller

A servo controller is required so that the Jetson Nano or the software can pass on the commands to the steering and the drive motor. You connect the steering servo and the electronic speed controller (ESC) directly to this. This way you can later take control of the robot car with a gamepad or even the neural network.

4.1.4 Gamepad

You will record the training data for the training of the neural network itself. To ensure that you get good training data and that it is not jagged and jerky in the steering movement and acceleration, I recommend using a gamepad. A cheap model like the "EasySMX Controller" has proven to be completely sufficient. However, if you have a Playstation or X-Box controller, you can test it first to see if you can use it. The radio signal of different gamepads I could test could not be disturbed by the open

electronics of the robot car or by the countless smartphones of training participants. I had some negative experiences with wireless keyboards that no longer functioned properly when the camera's CSI cable was connected, for example.

4.1.5 OLED display

Very recommendable is also a small OLED display on which you can see the occupancy of the RAM and the CPU load. But in my opinion the most important function of the OLED display is the display of the current IP address of the Jetson Nano. So you can quickly see the IP address of your Jetso Nano in foreign networks and if the Jetson Nano is connected to WIFI or not. Thus, an OLED display with corresponding output saves you a lot of frustration and the search for a monitor, mouse and keyboard.

4.1.6 Cables and adapters

In order to be able to connect the various components in the robot car with their different interfaces, you need different cables. The following section describes exactly which cables you need for what.

USB-A extension

A USB-A extension cable of about 15cm or an already existing small USB hub I recommend to use for space reasons. This way you can easily connect the radio receiver of the gamepad to the Jetson Nano. Because depending on the setup of the robot car, it can happen that the USB ports of the Jetson Nano are no longer easily accessible for long and large adapters.

Cut USB-A cable

You will need an extra USB A cable if your speed controller does not have the Battery Elimination Circuit (BEC) function. In this case you have to supply the servo motor for the steering of the robot car separately via the servo controller with power. An old cable, e.g. from a computer mouse that is no longer needed, which you can cut off, is sufficient here.

Female-to-Female Jumper Cable

You will connect the servo controller to four pins of the GPIO pin header of the Jetson Nano. You will need four of these female-to-female jumper cables. You will need four more if you want to add an OLED display and connect the also.

Female connector adapter

To avoid problems with loose contacts on the female-to-female jumper cables between the Jetson Nano and the servo controller I recommend to solder the 8 contacts for this connection. To avoid soldering the four female-to-female jumper cables directly to the Jetson Nano and servo controller, please use two 2x6 female headers that you can plug onto the GPIO pins of the Jetson Nano and onto the six pins of the servo controller. Solder the four female-to-female jumper cables directly to the two female connectors.

4.1.7 RC Battery - Voltage Display

As the last component of this list, I would like to recommend a voltage indicator for the RC battery. With this small display you have the current voltage at the motor in view. With this display it is easier for you to avoid a deep discharge of the RC battery.

4.1.8 Components list - online

A constantly updated list of recommended components can be found on my blog at the following URL.

URL: *https://custom-build-robots.com/jetson-nano-komponenten-de*

I very much appreciate your feedback on this chapter. Write me your thoughts, questions and suggestions to the following e-mail address: *ebook@custom-build-robots.com*

5 Building the robot car and setting up the Jetson Nano

Now it's time for tinkering and smaller soldering tasks, because all the components from the Jetson Nano to the roll bar have to be assembled into a functioning robot car.

If your RC car has a body, remove it and familiarize yourself with the electronics already installed in the chassis. If the radio receiver does not interfere, simply leave it installed. Just disconnect the power supply and unplug the steering servo cable and the speed controller cable from the receiver. The goal is to set up the robot car so that you can switch back and forth between the robot car and the radio-controlled car in a few easy steps if you want to.

The following description assumes that you are using the CC-01 chassis from Tamiya. However, the instructions are so general that they will also work for many other RC car chassis since the typical construction of RC cars is very similar. You may need to modify the base plate on which the electronics are mounted to fit your model and mounting points.

5.1 Required tools

This short overview should help you to have the right tools at hand when you start building the robot car.

- 2mm, 3mm, 5.5mm, 6.5mm and 12mm wood drill bit
- 16 mm hole drill
- Drilling machine or ideally a pillar drilling machine
- Soldering station and a 3rd hand
- Set square and pencil
- Screwdriver set

In addition to the tool, you will need a few wood screws, e.g. to fix the roll bar to the base plate.

- Various small screws such as M3 with a length of approx. 25mm

5.2 Preparing the base plate

The CC-01 and TT-02 chassis have two body mounts at the front and rear, which are typical for model cars. Measure the position of the four pins exactly and transfer the position to the base plate which you have already cut to size e.g. in the hardware store. The holes should be positioned so that the base plate is flush with the chassis at the rear and is centered on the chassis when viewed from above.

You can drill the four holes with a diameter of 5mm for the body mount into the base plate as indicated in the following drilling template. Before drilling, however, please check whether the dimensions actually fit your chassis.

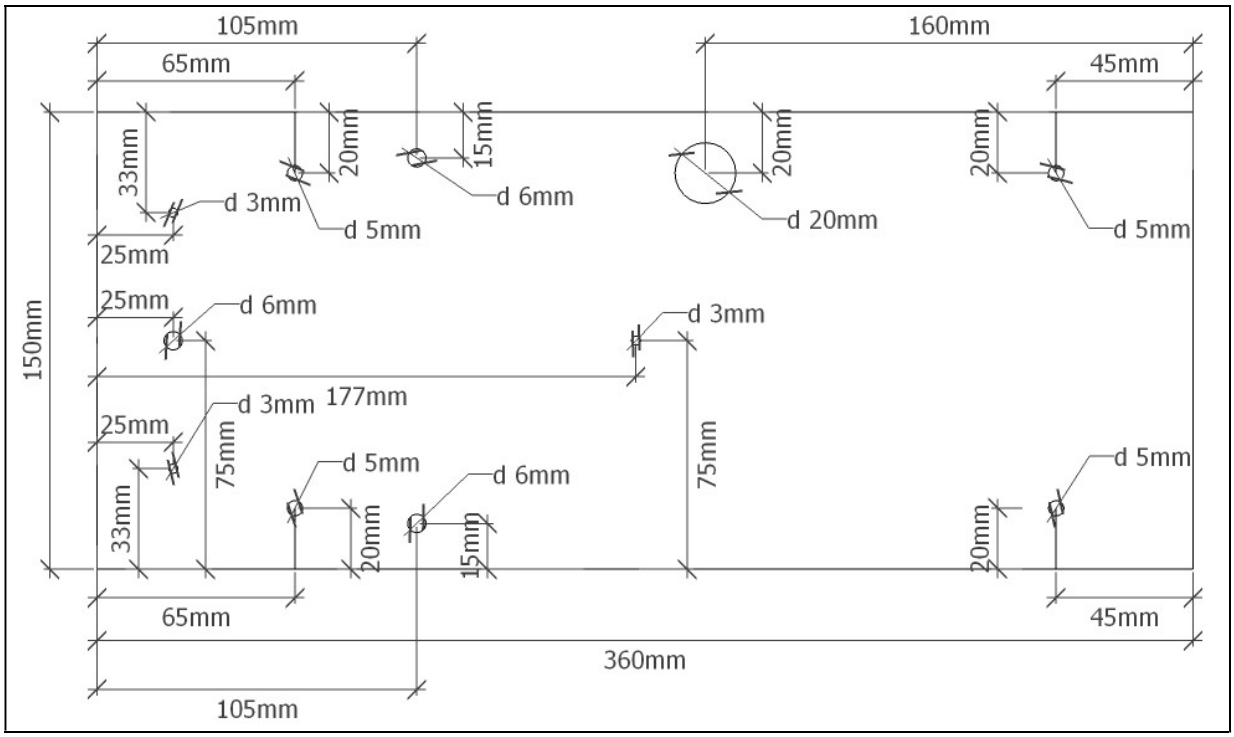


Figure 5.1: Base plate

You now have the base plate mounted on your chassis and it fits tightly without bending the four body mounts.

Figure 5.2: Base plate mounted on the chassis

The other holes already marked in the drilling template will be drilled in the base plate later. Therefore, please read the section Fixing components carefully before drilling the additional holes.

5.3 Attach components

After the base plate is centered and firmly seated on the chassis, the other components are attached to the base plate. The roll cage determines the position of all other components. Therefore, the roll cage is now attached first.

5.3.1 Fasten roll cage

Now find the right position for the roll cage on the base plate. Make sure that the wide-angle camera is positioned approx. 3 cm to 4 cm behind the bumper so that it or parts of the chassis are not visible in the picture. You can now slide the camera into the holder provided for the camera at the front of the roll

cage to determine the position of the roll cage. Viewed from the front, the camera or the roll cage is again centered on the base plate. Now measure the position of the three holes on the bottom of the roll cage and transfer them to the top of the base plate. Now drill the three holes in the base plate with a drill bit of approx. 3 mm in order to screw the roll cage from below.

Now screw the roll cage to the base plate from below with three screws and check if it is tight. The base plate with roll cage should look something like the following picture.

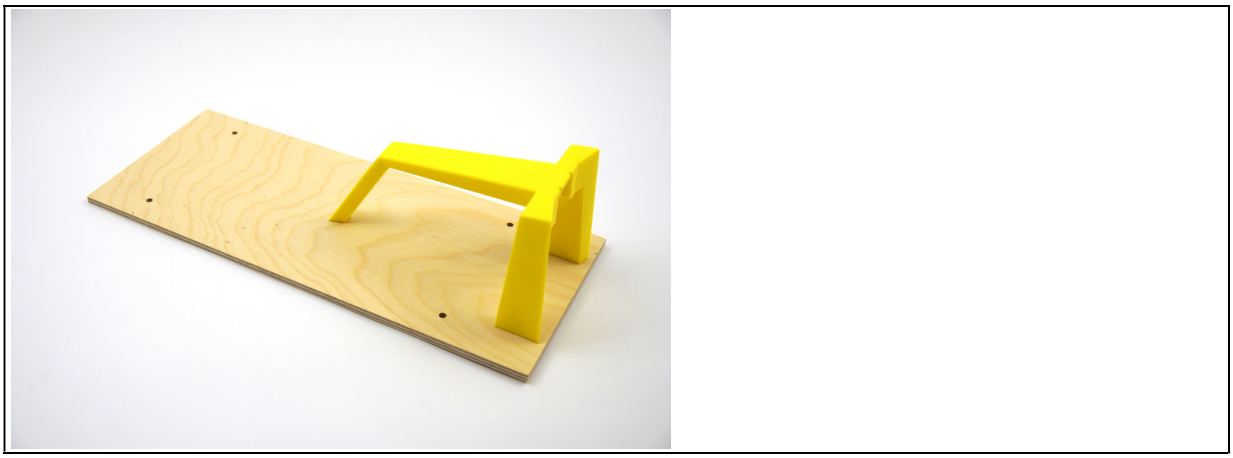


Figure 5.3: Base plate with roll cage

5.3.2 Set the position for Electronic Components

Unpack the Jetson Nano, the radio receiver of the gamepad, the servo controller and the OLED display if you have not already done so.

I recommend mounting the Jetson Nano under the roll cage. The connections such as USB, HDMI, LAN, etc. point to the rear. This way the Jetson Nano is protected when transporting the robot car to e.g. meetups and in case of accidents.

> **Background:** During a workshop, my robot car drove under a radiator at full speed and, since there was not even minimal protection, shaved off the GPIO pin header of the Jetson Nano.

Place the Jetson Nano on the base plate at the corresponding position under the roll cage. Make sure that the connections such as HDMI, network and USB are freely accessible. Now plug in the USB A extension cable or the mini USB hub. Connect the wireless receiver of the gamepad to the extension cable or mini USB hub. If you use a USB-WIFI module instead of the Intel NIC module to make your Jetson Nano W-LAN capable, then plug it into a free USB port.

Once you have found the right position for the Jetson Nano with the connected electronics, now mark the four holes on the base plate with which the board of the Jetson Nano is screwed.

Now you have to drill a slightly larger hole in the base plate through which you can insert the cable of the servo motor and the cable of the speed controller. It is best to have the hole on the right side of the base plate, so that you still have the option to move the roll cage backwards or forwards. For the hole, use a 16mm hole drill if available or drill two holes with some overlap next to each other with e.g. a 12mm wood drill.

Put the base plate back on the chassis. Now connect the servo controller to the steering servo and speed controller. Bring both cables of the steering servo and the speed controller up through the 16mm hole provided for this purpose. Connect the speed controller to channel 0 and the steering servo to channel 1. When looking for a suitable position for the servo controller, make sure that the cables are not live. Now mark the position of the four holes with which the servo controller will be fixed also on the base plate.

Now find a place for the power bank in the rear of the robot car, for example. You can easily attach it to the base plate using two cable ties and four holes in the base plate. Mark the four holes again if you want to copy the solution with the cable ties.

The following picture shows an intermediate state how the setup should look like now. The Jetson Nano, the power bank and the servo controller are already mounted in this picture. You can also see the cables from the speed controller and the steering servo which were led up through the 16mm hole.

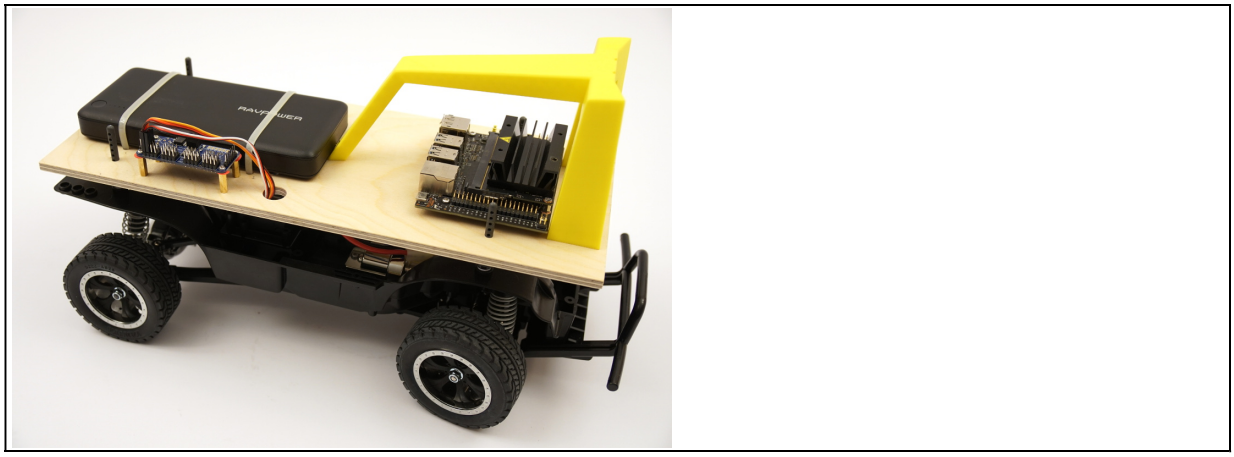


Figure 5.4: Robot car current assembly

You should choose the position of the OLED display so that you can read it well and that it is not too far away from the servo controller because it is connected to its I²C bus output. In order not to lose any space, you may mount the OLED display directly to the right of the servo controller. Then mark again the four holding holes of the display on the base plate.

If you use an SSD hard drive, as recommended, you can attach it directly together with the power bank under one of the two cable ties. The attachment with the cable ties allows you to pull out the SSD hard drive as well as the power bank and also to put it back under the cable ties.

> **Note:** My practical experience with the mounting solution of the power bank with cable ties has shown that you should still secure the power bank in the rear with a screw behind the power bank from accidentally slipping out.

5.3.3 Drill holes

You may have to unscrew the roll bar again if you now drill all the holes you marked. Drill the holes for the three boards Jetson Nano, servo controller and OLED display with a 2mm drill into the base plate. You should now be able to screw the brass spacers into these 2mm holes well and stably. Drill the four holes for mounting the power bank with the cable ties with a suitable drill bit that matches the width of the cable ties.

5.3.4 Intel AC8265 Wireless Antennas Mount

If you have decided to use the Intel AC8265 Wireless NIC module, then mount it now. To do this, loosen the two screws that secure the Jetson Nano module to the carrier board. Click the module out like you used to do with the SO-Dimm memory bars. Connect the two cables for the antennas to the NIC module, remove the fixing screw and put the NIC module into the designated slot below the Jetson Nano. Fasten the module again with the screw, now put the Jetson Nano back into the holder and screw it tight with the two screws.

Drill two holes with a 6.5mm drill bit into the base plate to mount the two WIFI antennas. When choosing the position of the holes for the two antennas and their SMA sockets, make sure that the two cables from the NIC module to the holes of the SMA sockets are long enough. I drilled a 6mm hole in the center of the base plate in front of the Jetson Nano through which I led the thin antenna cables to the SMA sockets.

> **Note:** So that you can screw the SMA sockets to the 6mm thick base plate, you may have to drill out the two holes from the bottom to the middle of the base plate with a 12mm drill bit. So the thread

looks a little further through the base plate and you can still screw the antennas on the SMA socket itself.

In the following picture you can see the hole in front of the Jetson Nano through which the antenna cables coming from the NIC module are led to the SMA sockets. At the back right of the picture you can see one of the two antennas that is already attached.

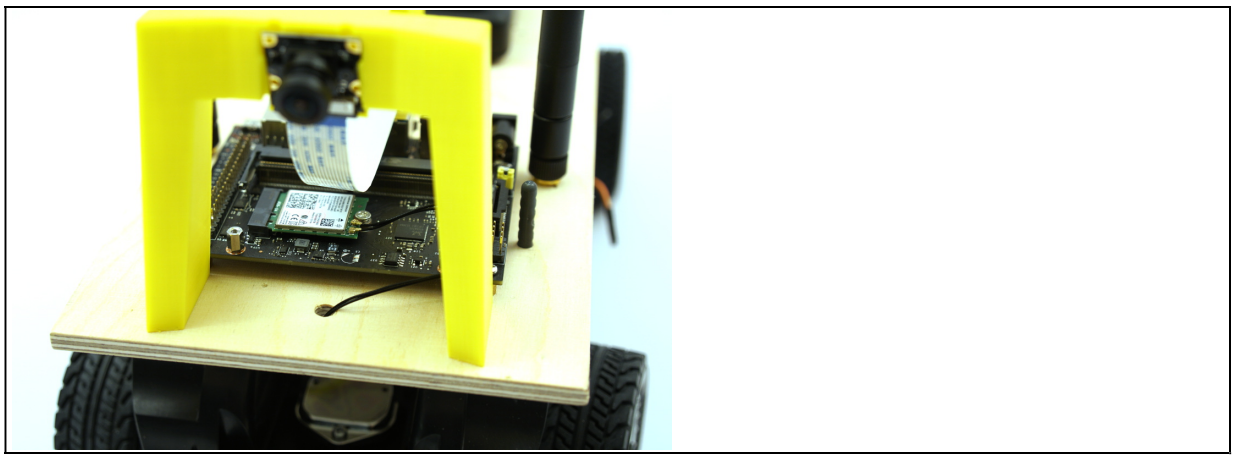


Figure 5.5: Base plate with antenna cable bushing

Now reattach the roll bar and all electronic components such as the Jetson Nano, servo controller, etc. to the base plate.

5.4 Wiring of the components in the robot car

In this section you will do the wiring of the single components in the robot car. But first I will explain what exactly the I²C bus is to which the servo controller and the OLED display are connected.

5.4.1 I²C Bus a short introduction

The Inter-Integrated Circuit Bus (I²C Bus) is a serial bus developed by Philips Semiconductors in 1982 for communication between a controller and connected components. The Jetson Nano has an I²C bus which is connected to pin 3 (SDA) and pin 5 (SCL) of the GPIO pin header. On the I²C bus you will connect the servo controller and the OLED display in series.

5.4.2 Wiring I²C bus

To ensure that the I²C connection between the Jetson Nano and the servo controller is free of loose contacts even when the robot car is shaken, I recommend that you do not just plug in the four female-to-female jumper cables, but solder them firmly to a 2x6 female connector strip. By using a 2x6 female connector you don't have to solder the four cables to the Jetson Nano itself but solder them to the female connector. You then connect them to the first six pins of the GPIO pin header of the Jetson Nano and thus get a connection to the servo controller that is free of loose contacts.

In the following picture the GPIO pin header with the I²C bus pins is highlighted again. You can also see the first pin with the 3.3V and the GND pin opposite the SCL pin.

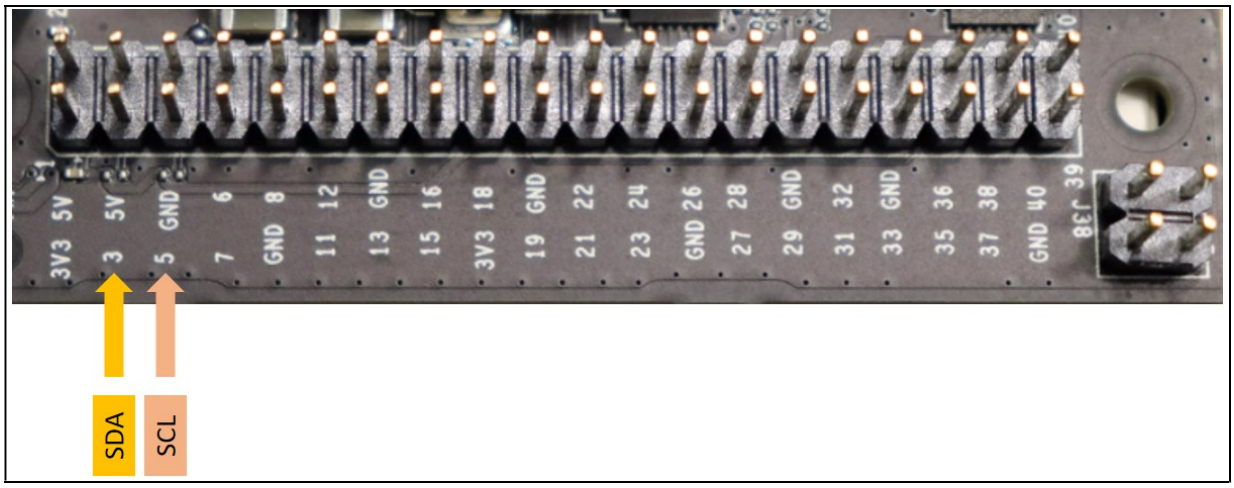


Figure 5.6: GPIO Pinout of the Jetson Nano (I2C)

Familiarize yourself with the pin assignment of the female connector strip which pin of the Jetson Nano which pin on the female connector strip now extends.

Remove the plastic sleeves from four of the Jumpel cables so that you can see the bare cable or the eight exposed sockets of the cable on both sides. Before soldering the cables to the socket strip, pull them back off the Jetson Nano to protect it from the heat of the soldering iron. Slide one cable after the other together with a small piece of heat shrink tubing onto the pin of the female header for e.g. the GND connection. Now solder the cable to the female connector and insulate it with the heat shrink tubing. Now repeat this for the three other connections SDA. SCL and 3.3V.

The following table helps you to identify the correct pins on the Jetson Nano for the I²C bus. Enter in the table the colors of the cables of the I²C bus you have used.

Jetson Nano Pin Designation	Cable color	I²C bus pin designation
3,3V		VCC (3.3V)
3		SDA
5		SCL
GND		GND

Table 5.1I²C bus mapping

> **Note:** Please make sure that the servo controller and the OLED display are operated with a voltage of 3.3V. Otherwise, you may damage the Jetson Nano when using a 5V voltage.

The female connector that is plugged into the Jetson Nano looks soldered as shown in the following picture. In the picture the pins with their respective function of the I²C bus as well as the 3.3V and GND pin are marked.

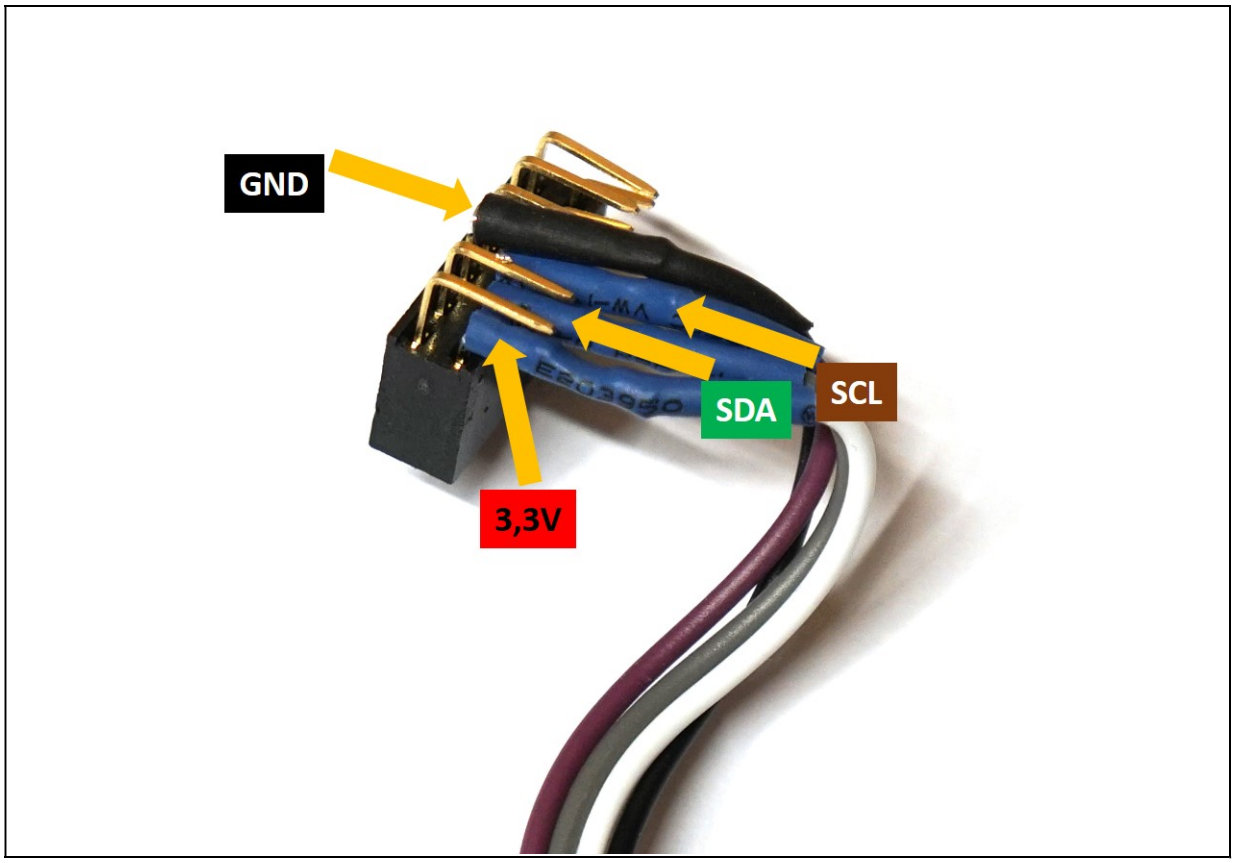


Figure 5.7: Female connector strip for the Jetson Nano

At the four other ends of the female-to-female cable, remember again to use the heat shrink tubing to insulate the cables. Now solder the four cables to the second female connector according to the pin assignment of the servo controller.

On the following picture you can see on the left side the socket connector which is plugged on the servo controller and on the right side the socket connector which is plugged on the Jetson Nano. Both are connected by a four-wire jumper cable which is soldered to both connectors and insulated with heat shrink tubing. Connecting the I²C in this way is much more free of loose contacts than just plugging the jumper cables.

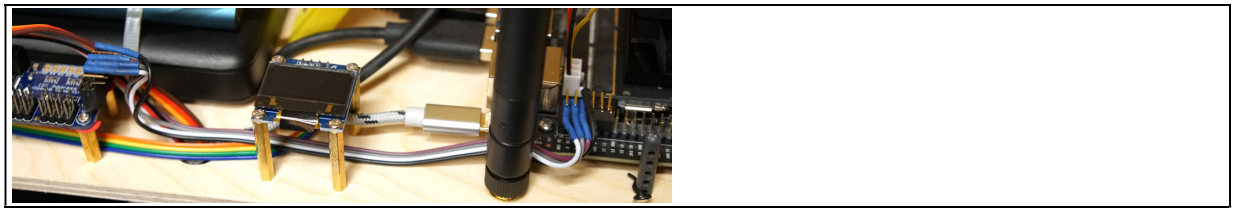


Figure 5.8: Socket connector right and left firmly soldered Jumpe cable

The OLED display is also connected to the I²C bus (output) of the servo controller. If this fails, e.g. because of a loose contact, this is not as annoying as with the servo controller. Therefore you can simply plug in the OLED display and do not have to work with additional socket connectors.

The following picture shows how the wiring of the robot car looks like so far.

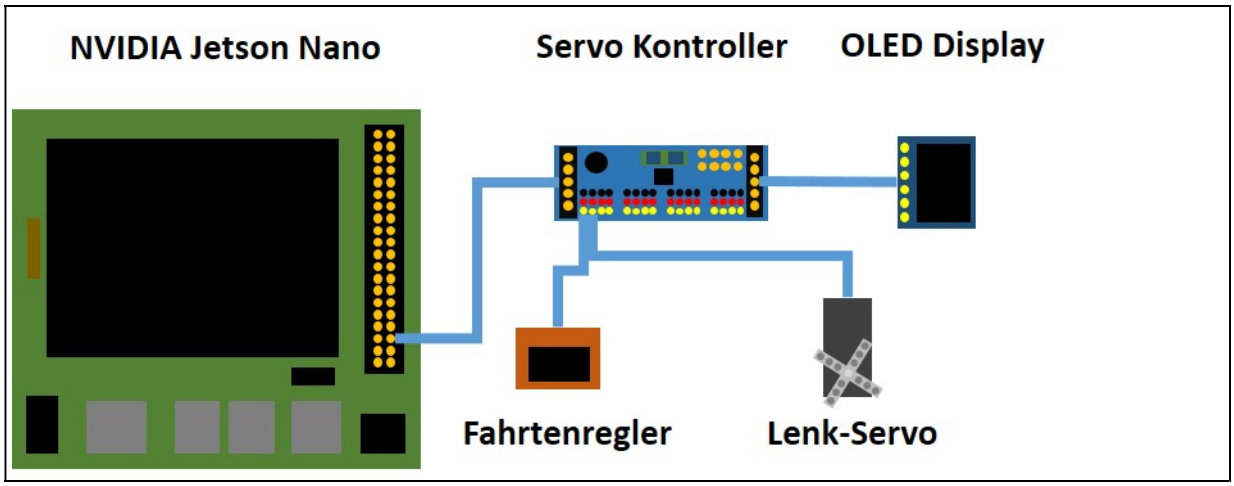


Figure 5.9: Logical wiring of the Donkey Car

5.4.3 Speed controller with active BEC:

If your speed controller has an active BEC (Battery Elimination Circuit) function, then the speed controller supplies the servo controller and thus also the steering servo with a voltage of 5V. In this case you do not have to supply the servo controller with an extra 5V voltage. An existing BEC function is actually the normal case with the speed controllers.

5.4.4 Speed controller without BEC:

If your speed controller does not have a BEC function, then you must connect the servo controller to the power bank so that the steering servo is supplied with power.

To establish the power supply you now need a cut USB A cable that is approx. 30cm long from an old USB mouse or USB keyboard. Connect the black and red wires of the USB cable to the connector terminal of the servo controller. The white and green wires in the USB cable are the wires for data transfer and are not needed. You can cut them very short so that they do not interfere. When you have connected the two cables (red/black) to the servo controller, plug the USB cable into the power bank and supply the servo controller and the connected steering servo with a voltage of 5V.

5.4.5 Connect camera

You must plug the camera's CSI cable into the Jetson Nano. Please make sure that you carefully open the latch of the CSI connector on the Jetson Nano. The two ends of the CSI cable are each marked in blue. The Blue side of the CSI cable points away from this path when connected to the Jetson Nano.

Now connect the camera to the other end of the cable and plug it into the holder on the roll cage. When connecting the CSI cable to the camera, there may be technical differences depending on the model whether the blue marked side of the CSI cable points towards or away from the lens. Please check with the manufacturer of your camera or just try it out.

Finished, your robot car should now look similar to the following picture.

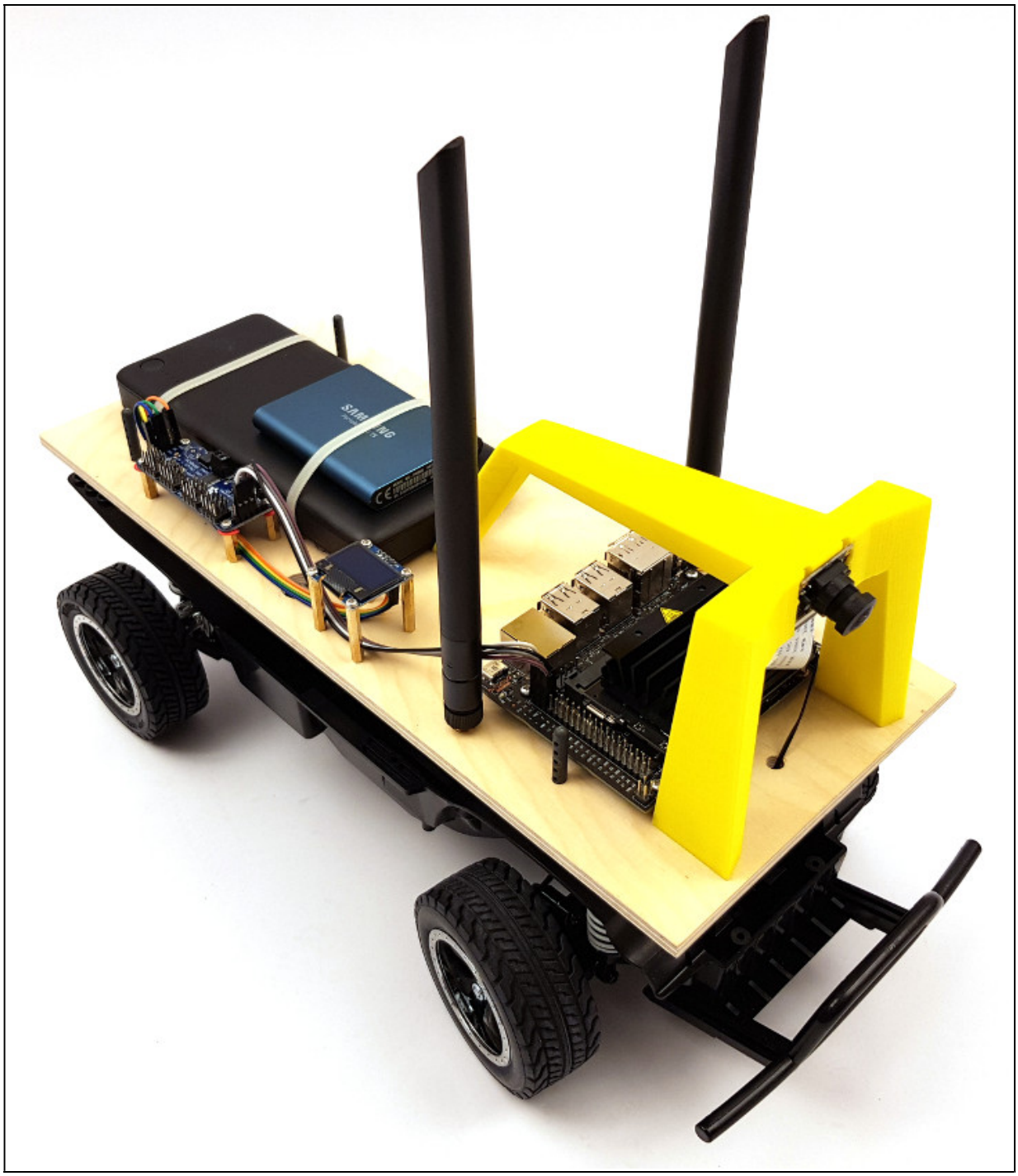


Figure 5.10: The completed robot car

Now that you have completed the assembly of the robot car and the wiring of the electronic components, the following chapter will tell you how to put the Jetson Nano into operation.

I very much appreciate your feedback on this chapter. Write me your thoughts, questions and suggestions to the following e-mail address: *ebook@custom-build-robots.com*

6 Setting up the operating system of the Jetson Nano

Finally the time has come, the soldering iron has cooled down and the drill has fallen silent. The Jetson Nano is now brought to life and you are about to dive into the depths of artificial intelligence and training data collection.

After you have built your robot car so far, the next step is to put the Jetson Nano into operation. To do this, you need to install the image file provided by NVIDIA onto the micro SD card.

Note: Regardless of whether you will be running your robot car with a micro SD card or SSD hard drive, you will need to complete the following steps.

Download the latest image file from the NVIDIA web page for the Jetson Nano now. For this e-book the image named "jetson-nano-4gb-jp441-sd-card-image.zip" and the Jet Pack version 4.4 was used.

The latest image file can always be found on the NVIDA website at the following URL.

URL: *https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write*

There are several tools with which you can write the image file to the micro SD card. The tool from Balena Etcher is quite widely used and available for various platforms like MacOS, Windows and Linux. You can download the program from the following address.

URL: *https://www.balena.io/etcher/*

Now write the image file to your micro SD card using the Etcher tool.

After you have successfully written the image to your micro SD card, insert it into the Jetson Nano. Connect the Jeston Nano to a monitor, mouse and keyboard. Power up the Jetson Nano with the external power supply and turn it on.

Note: Remember to set the jumper J48 directly behind the DC connector, otherwise the Jetson Nano will not start when you connect the external power supply. If you don't have a jumper at hand, you can also use a female-to-female jumper cable instead of a jumper.

You should now see the Ubuntu operating system that you previously wrote to the micro SD card booting up. Follow the installation wizard and configure your Jetson Nano.

Note: In the installation wizard questions, pay attention to the question about automatic login to the set up WIFI. Please enable this function.

Tip: If you are not yet so familiar with Linux and Ubuntu then the following chapter "Little helpers and tips: " will certainly help you to get started. In this I describe the small tools under Linux that I always install myself directly. These can make your work under Linux much easier, depending on your own experience and knowledge.

The next section is about updating the Ubuntu installation and customizing it for the needs of the Donkey Car project.

Therefore, first update the repository information of the Ubuntu Jetson Nano installation to the latest version.

Command: sudp apt-get update

After the respository information is up to date, use the following command to update all the already installed programs.

Command: sudo apt-get upgrade.

While the update is running you will be asked if you want to change the display manager. Please simply accept the default setting, i.e. the gdm3 display manager. Since you will later operate the Donkey Car completely without a monitor and graphical interface, the somewhat larger and more resource hungry gdm3 Display Manager can be installed here. In a later step, when the software on the Donkey Car is fully functional, the operating system is configured to boot without starting a display manager.

After the update has been completed, you will now create the prerequisite that no errors are issued due to insufficient memory during the following heavyweight installation processes. Therefore you create now a swap file for the working memory the so-called SWAP file.

6.1.1 SWAP file setup

Let's start with the creation of the SWAP file. If the operating system can access a SWAP file, then the operating system can swap parts of the working memory into it. Now create a SWAP file with a size of about 12 GB if you have enough free space on the SD card. If there is not enough space then make the SWAP file a little smaller but at least 6 GB in size.

With the following command you can display how much memory is still available on the micro SD card.

Command: df -h

If the free space still allows a 12 GB SWAP file, then accept the following command. Otherwise, adjust the number 12 according to the desired size of the SWAP file in the following command.

Command: sudo fallocate -l 12G /var/swapfile

Now you have to set the correct permissions to access the SWAP file.

Command: sudo chmod 600 /var/swapfile

Then the SWAP file is created with the size specified before.

Command: sudo mkswap /var/swapfile

The following command activates the SWAP file.

Command: sudo swapon /var/swapfile

In order to activate the SWAP file after each reboot, the following command must be executed which enters the SWAP file with its path into fstab.

Command: sudo bash -c 'echo "/var/swapfile swap swap defaults 0 0" >> /etc/fstab'

If this command throws an error message, then start Midnight Commander and add the following line to the /etc/fstab file.

/var/swapfile swap swap defaults 0 0

The /etc/fstab file should now look like this. The entry with the SWAP file is clearly visible.

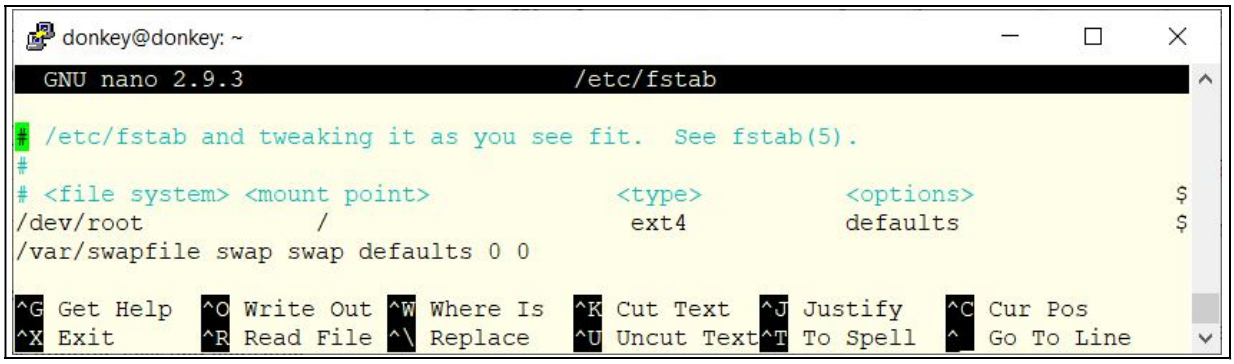


Figure 6.1: fstab SWAP file

Now everything is set up, from the small tools and optimizations that help you to get around easier in the Linux terminal window to the SWAP file. Now restart the Jetson Nano

Command: sudo reboot

After restarting the Jetson Nano, you can use the following command to verify that the SWAP file has been actively mounted.

Command: free mem

The output you get in your terminal window should look like this. The output memory size may differ according to your hardware and configuration.

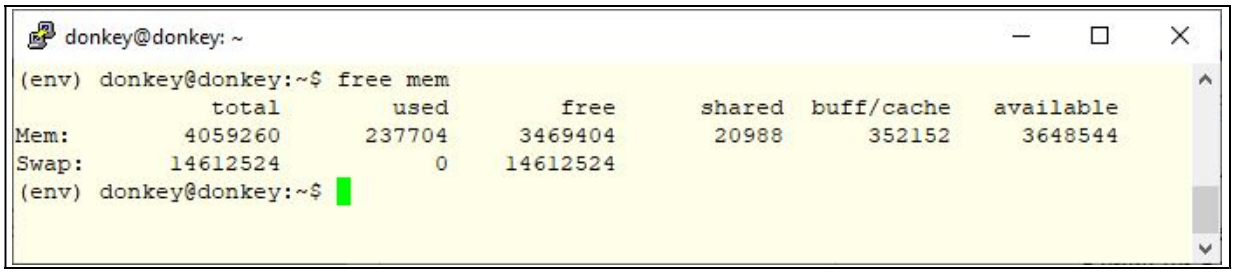


Figure 6.2: free mem output

6.2 SSD hard disk setup - recommended (but optional)

The demands of the Donkey Car framework on the hardware of the Jetson Nano are very high and an SSD hard disk brings stability and speed to the robot car project. You will notice a big difference when you train the neural network, where the SSD hard disk brings its advantages in read and write speed fully into the project.

In the Internet there is the web page *www.jetsonhacks.com* which always puts very good instructions including scripts to the Jetson family of NVIDIA. online. Exactly here there is also a tutorial that describes exactly how the Jetson Nano can be configured to boot from an SSD hard drive. We will follow this tutorial without going into the background. These are very well explained on the page itself.

First, you need to download the rootOnUSB project from the JetsonHacks GitHub account. To do this, please change to the home directory of your user on the Jetson Nano and execute the following command.

Command: git clone https://github.com/JetsonHacksNano/rootOnUSB

Change to the rootOnUSB directory

Command: cd rootOnUSB

The next step is to build the initramfs with USB support. Only then it is possible to access the USB interface of the Jetson Nano and thus the connected SSD very early in the boot process. The script addUSBToInitramfs.sh does exactly this task. Execute the script with the following command.

Command: ./addUSBToInitramfs.sh

After you run the command, you will see several warnings that various files could not be found. The warnings should not worry you and the USB support for booting from the SSD will work despite these messages.

The output and the displayed truths should look like this.

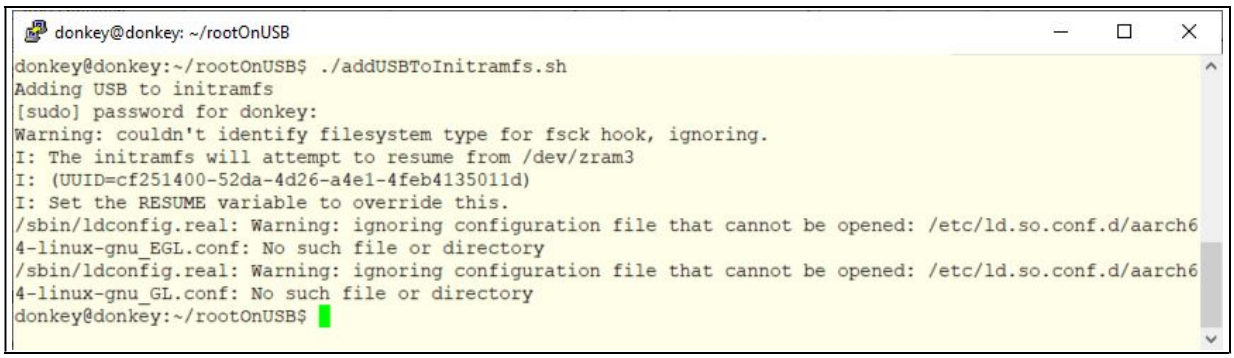


Figure 6.3: rootOnUSB perceptions

Now you have to prepare the connected SSD hard disk. Create a large partition that is ext4 formatted on it. With Ubuntu the program Disks is already installed which you need now. If you now create a partition with Disks on your SSD then name this partition e.g. SSD. The volume name of the hard disk we need in the following step when all data are copied from the micro SD card to the hard disk.

The hard disk should now look like the following when set up in Disks. You can see in the image that it has been given the volume name SSD. You can also see that the disk has been detected as a /dev/sda device.

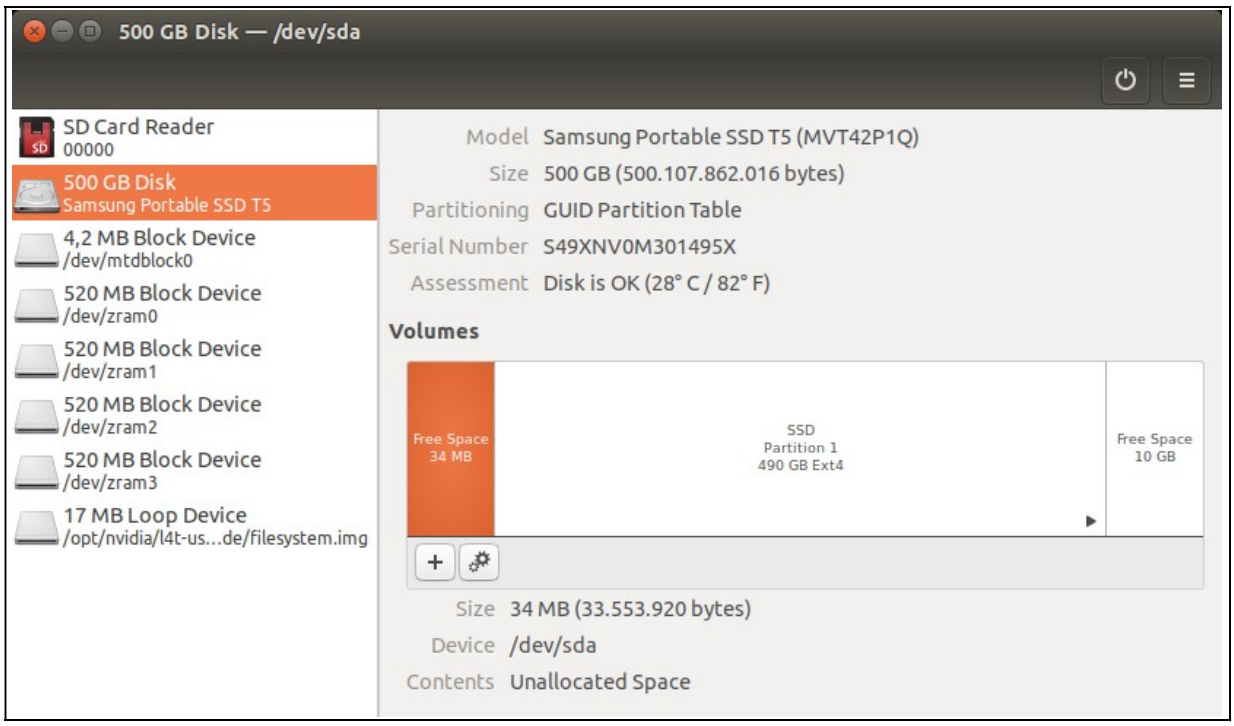


Figure 6.4: Program Disks under Ubuntu

Then copy all the data from the micro SD card to the hard drive with the volume name SSD.

> **Note:** Before executing the following command, please make sure that the hard disk or the partition with the name SSD is also mounted. To mount the partition you can click on the SSD icon in the left menu of Ubuntu to mount the partition.

Now execute the following command to start the copy process.

> **Command:** ./copyRootToUSB.sh -v **SSD**

The copying process takes several coffees.

6.2.1 First entry - LABEL primary

After the copy process is complete, you will make one last configuration. With this you define that the Jetson Nano switches from the micro SD card to the SSD hard disk during the boot process and boots the operating system from there.To do this, open the extlinux.conf file with the text editor nano.

> **Command:** sudo nano /boot/extlinux/extlinux.conf

Copy the complete label primary with its five lines once and now adjust the configuration as described below.

The label entry "primary" is only adjusted in that the path of the micro SD card is replaced by the path of the SSD hard disk. With the already known command df -h you can make sure exactly what the partition is called that you have created. This should be called sda1, for example, and be accessible via the path /dev/sda1.

6.2.2 Second entry - LABEL sd-card

Since you have copied the first entry, you only have to change the name of the label to e.g. "sd-card". Also adjust the remaining lines as described.

Once configured, the extlinux.conf file looks like this.

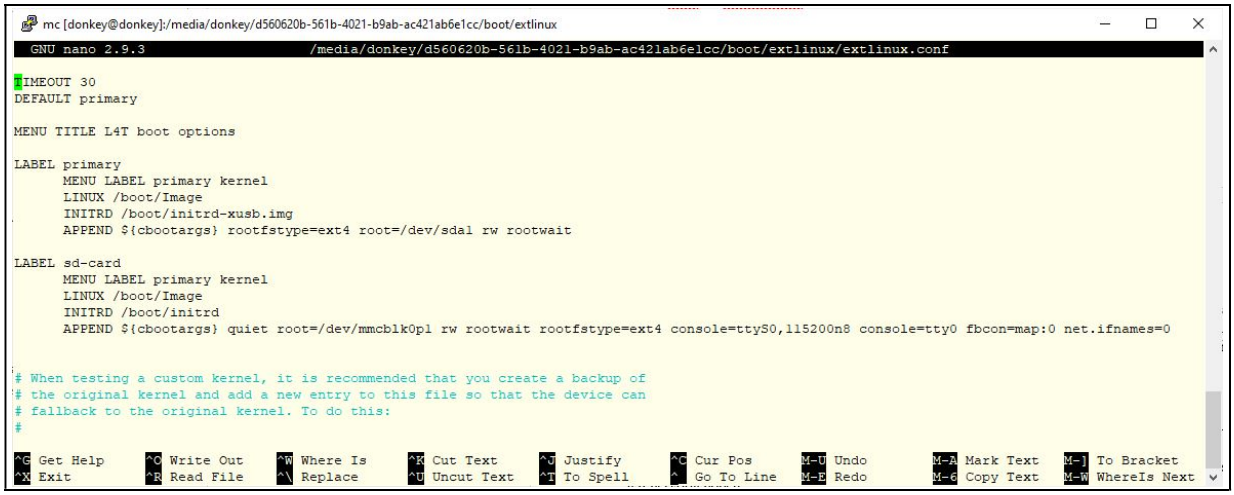


Figure 6.5: extlinux.conf customization

Here the adjustment once again in text form. The important points to which attention must be paid are highlighted in bold.

```
TIMEOUT 30
DEFAULT primary

MENU TITLE L4T boot options

LABEL primary
        MENU LABEL primary kernel
        LINUX /boot/Image
        INITRD /boot/initrd-xusb.img
        APPEND ${cbootargs} rootfstype=ext4 root=/dev/sda1 rw rootwait

LABEL sd-card
        MENU LABEL primary kernel
        LINUX /boot/Image
        INITRD /boot/initrd
        APPEND ${cbootargs} quiet root=/dev/mmcblk0p1 rw rootwait rootfstype=ext4
console=ttyS0,115200n8 console=tty0 fbcon=map:0 net.ifnames=0
```

Now save the changes in extlinux.conf and restart the Jetson Nano.

When you log back into the graphical user interface, you should see the symbol for an inserted mirco SD card in the lower left corner. If this is the case, your Jetson Nano has booted the operating system from the SSD hard disk without errors. If not then maybe something went wrong in the extlinux.conf file. Check here again the adjustments you have made.

6.3 What you have achieved up to here.

You have connected all components from the servo controller to the camera and prepared the micro SD card with the Jetson Nano image. You have also powered up the Jetson Nano and set up the SWAP file. Your Jetson Nano will boot flawlessly from the micro SD card or possibly from the SSD hard drive if you are using one for this project.

I very much appreciate your feedback on this chapter. Write me your thoughts, questions and suggestions to the following e-mail address: *ebook@custom-build-robots.com*

7 Installation of the Donkey Car framework and calibration of the robot car

I'm glad you're moving forward and haven't driven out to the right. This chapter is finally about bringing the robot car to life. From installing the Donkey Car framework to calibrating it to driving it for the first time.

The Donkey Car Framework forms the software basis of the robot car. Installing the software packages listed in this chapter together with the Donkey Car Framework turns the former RC car into an autonomously driving model car based on the Jetson Nano. The Donkey Car Framework is also available for the Jetson Nano since version 3.x and can be installed on it without any problems. You should take some time for the installation. You don't always have to sit in front of the computer and watch how the software installation progresses.

The first two commands are used to update the Jetson Nano to the latest software version, since some time may have passed since the last update. Therefore, you should always execute these two commands before you enter the remaining commands one after the other as usual and install the software step by step.

Command: sudo apt-get update

Command: sudo apt-get upgrade

Now, before you install the other libraries and tools for the Donkey Car Framework, please check that the SWAP file is also active. You can do this easily with the following command.

Command: free mem

If you do not see the SWAP file, then check again whether you still need to configure it or just activate it. The exact procedure has already been described in chapter 6.2

Note: Currently, the Donkey Car Framework is under very active development and changes are being made all the time. This e-book is always being updated, so check the following URL to see if a new version is available.

URL: <please add>

Now proceed with the installation and install the other packages on your Jetson Nano. This is now quite a long list you have to work through.

Befehl: sudo apt-get install -y libhdf5-serial-dev hdf5-tools libhdf5-dev zlib1g-dev zip libjpeg8-dev liblapack-dev libblas-dev gfortran

Command: sudo apt-get install -y python3-dev python3-pip

Command: sudo apt-get install -y libxslt1-dev libxml2-dev libffi-dev libcurl4-openssl-dev libssl-dev libpng-dev libopenblas-dev

Command: sudo apt-get install -y git

Command: sudo apt-get install -y openmpi-doc openmpi-bin libopenmpi-dev libopenblas-dev

Befehl: sudo -H pip3 install -U pip testresources setuptools

Command: sudo -H pip3 install -U futures==3.1.1 protobuf==3.12.2 pybind11==2.5.0

Command: sudo -H pip3 install -U cython==0.29.21

Command: sudo -H pip3 install -U numpy==1.19.0

Command: sudo -H pip3 install -U future==0.18.2 mock==4.0.2 h5py==2.10.0 keras_preprocessing==1.1.2 keras_applications==1.0.8 gast==0.3.3

Befehl: sudo -H pip3 install -U grpcio==1.30.0 absl-py==0.9.0 py-cpuinfo==7.0.0 psutil==5.7.2 portpicker==1.3.1 six requests==2.24.0 astor==0.8.1 termcolor==1.1.0 wrapt==1.12.1 google-pasta==0.2.0

Command: sudo -H pip3 install -U scipy==1.4.1

Command: sudo -H pip3 install -U pandas==1.0.5

Command: sudo -H pip3 install -U gdown

The following command will install TensorFlow version 2.2.0 on your Jetson Nano.

Command: sudo -H pip3 install --pre --extra-index-url https://developer.download.nvidia.com/compute/redist/jp/v44 tensorflow==2.2.0+nv20.6

The following two commands install PyTorch v1.7.

Command: wget https://nvidia.box.com/shared/static/wa34qwrwtk9njtyarwt5nvo6imenfy26.whl -O torch-1.7.0-cp36-cp36m-linux_aarch64.whl

Command: sudo -H pip3 install ./torch-1.7.0-cp36-cp36m-linux_aarch64.whl

Next, PyTorch Vision will be installed, but a few packages are needed beforehand.

Command: sudo apt-get install libjpeg-dev zlib1g-dev libpython3-dev libavcodec-dev libavformat-dev libswscale-dev

Now download torchvision v0.8.1 to the projects folder and install tourchvision.

Command: mkdir -p ~/projects; cd ~/projects

Command: git clone --branch v0.8.1 https://github.com/pytorch/vision torchvision

Command: cd torchvision

Command: export BUILD_VERSION=0.8.1

Command: sudo python3 setup.py install

Command: echo "export LD_PRELOAD=/usr/lib/aarch64-linux-gnu/libgomp.so.1" >> ~/.bashrc

Now you should restart the Jeston Nano before continuing with the software installation.

Command: sudo reboot

With the following command you can check which Tensorflow version you have installed now.

Befehl: python -c 'import tensorflow as tf; print(tf.__version__)'

7.1 Set up the virtual environment

The Donkey Car Framework is installed in a virtual environment. This gives you more flexibility later on if you want to install the software for face recognition in a second virtual environment, for example, because the two installations do not influence each other. In case of problems, you can, for example, install the software for the robot car again in a third virtual environment and can thus switch between the installations.

Command: pip3 install virtualenv

The following command creates a virtual environment named env.

Command: python3 -m virtualenv -p python3 env --system-site-packages

For much more convenience, the following command adds the entry *source env/bin/activate* into the .bashrc file in the home directory at the end of it. This entry ensures that whenever you log on to the Jetson Nano, the virtual environment is started and you are then located in it.

Command: echo "source env/bin/activate" >> ~/.bashrc

Now manually start the virtual environment named env for the first time with the following command.

Command: source ~/.bashrc

You can recognize in the terminal window in the input line at the very beginning by the "env" that you are in the virtual environment with the name "env". In the following picture the place in the terminal window is highlighted with an orange circle.

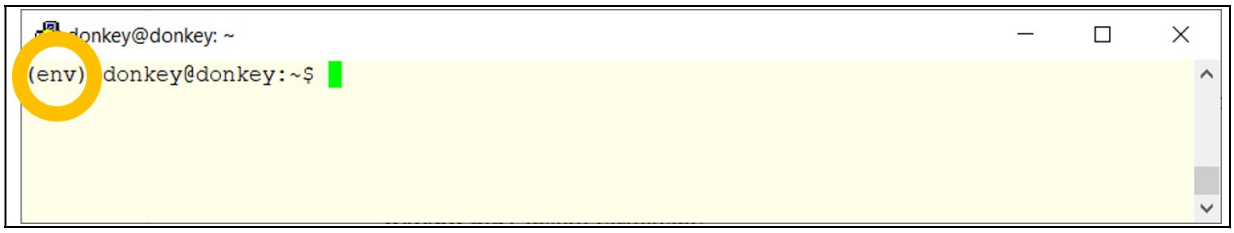


Figure 7.1: Active virtual environment

If you want to deactivate or leave the virtual environment you can do this with the following command.

Command: deactivate

Now that the virtual environment is set up, the installation can continue, but you might want to follow the advice below.

The installation of the Jetson Nano has taken me about 3 to 4 hours so far. Therefore, now would be the right time to create an image of the micro SD card as a backup.

Example command: dd if=/dev/sdc of=~/image_nano.img status=progress

Shut down the Jetson Nano using the following command.

Command: sudo shutdown -h now

7.2 Install Donkey Car Framework

Now all preparations are done and the Donkey Car framework for the Jetson Nano can be downloaded from GitHub. It is best to save the framework in a folder named "projects" in your home directory.

Note: Please make sure that you perform the following software installation steps in the active virtual environment "env".

Use the following command to create the folder named projects.

Command: mkdir ~/projects

Now switch to the folder projects.

Command: cd ~/projects

From GitHub you now download the current Donkey Car framework. This works with the following commands.

Command: git clone *https://github.com/autorope/donkeycar*

Please change now to the folder "donkeycar" under the folder "projects".

Command: cd donkeycar

Now you still need to check out the project.

Command: git checkout master

After so many preparations now follows the installation of the Donkey Car Framework. The framework is also available for the Raspberry Pi and is installed for it by default. Therefore you have to add the parameter [nano] to the following installation command.

Command: pip install -e .[nano]

Note: If you get an error message that the "pip install command" was not found then restart your Jetson Nano. After the restart the command should be executable.

The installation of the Donkey Car framework now takes only a few minutes.

Now follows the creation of your own Donkey Car instance on your Jetson Nano. To do this, create a folder named mycar in your home directory and then change to this folder.

Command: mkdir ~/mycar

Command: cd ~/mycar

The following command creates a Donkey Car instance in the mycar folder.

Command: donkey createcar --path ~/mycar

If an error message like the one below appears, then the file permissions of the folder are not correct.

Error message: PermissionError: [Errno 13] Permission denied: '/home/robo-Auto/mycar/models'.

The following command will set the file permissions accordingly and the error message will no longer appear. Then repeat the createcar command to create the Donkey Car instance.

Command: sudo chmod 777 ~/mycar

Now all the software needed for a working robot car is installed on the Jetson Nano. The next step is to configure the newly created Donkey Car instance with the technical parameters of the robot car hardware you are using, such as chassis, servo motor and speed controller. Only if the parameters are configured it is later possible to accelerate, steer and use the camera on the Jetson Nano.

7.2.1 I2C Bus and Servo Controller PCA9685 Adaptation

It can happen that the servo controller PCA9685 which is connected to the I2C bus is not found because we do not have access to the I2C bus of the Jetson Nano with our user.

You can test whether the user you are using can access the I2C bus with the following command.

Command: sudo i2cdetect -r -y 1

After you have executed the command, the PCA9685 servo controller is found with the I2C addresses 40 and 70. With the I2C address 3c the already connected OLED display. The display in the terminal window looks like this.

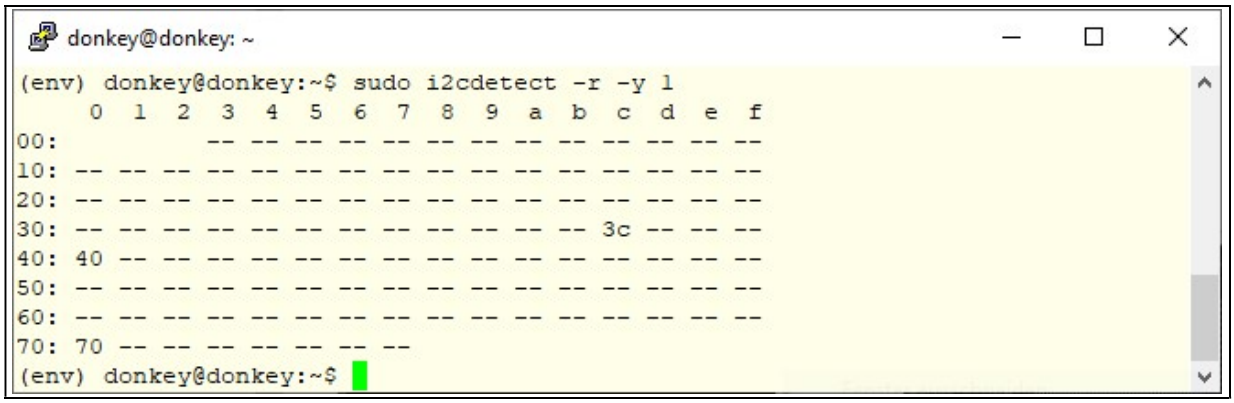


Figure 7.2: I2C bus display of all connected devices

If an error message appears or the servo controller is not recognized, then this is most likely due to the fact that the I2C bus may not be read with your user. To change this please execute the following command to change the group assignment of the I2C bus.

Command: sudo usermod -aG i2c $USER

Then restart the Jetson Nano and test again if you can find the servo controller connected to the I2C bus with the i2cdetect command.

7.3 Configuration of the robot car

Now it's time to start configuring the robot car. You have built the hardware so far and installed all software on the Jetson Nano as described. The goal of this section is to configure the steering and the speed controller so that you can control the robot car with the gamepad without any problems. The straight run should be correct as well as the steering angle to the left and to the right should be possible to the same extent. The configuration of the speed controller is important so that the robot car accelerates and brakes properly as is typical for a car.

Note: Since you will now not only switch on the Jetson Nano but also connect the speed controller to the battery, place the robot car on a cardboard box, for example, so that the wheels hang freely in the air. It already happened to me that one of my robot cars drove off unexpectedly. Once one fell off the table and another time a car scratched my monitor.

7.3.1 Prerequisites for calibration

For further calibration, you are connected to the Donkey Car via SSH and have disconnected the cables such as monitor, keyboard and mouse. The wheels of the robot car are in the air and the Jetson Nano gets its energy via the Power Bank. The RC battery for the drive motor is fully charged and connected as well.

Then switch on the power supply from the speed controller to the servo controller (BEC function) using the small switch on the underside of the Tamiya chassis, if present.

7.3.2 Calibrate steering:

If you have not yet connected the steering servo to the servo controller, you must now connect it to channel 1. Use the terminal window to switch to the mycar folder.

Command: cd ~/mycar

Now start the Donkey Car framework for calibrating the steering with the following command.

Command: donkey calibrate --channel 1 --bus=1

You will then be prompted to enter a PWM value. Start with a value around 400, which should be about the straight run. Then find the maximum value for the left and right steering angle. On one of my models the values are in the following range. You find the maximum value by letting the servo motor hit the steering so far that it mechanically hits a resistance and hums. Now change the value until no hum is heard.

	Example values:	**Your values:**
Full left:	250	
Straight run:	400	
Full impact on the right:	550	

Once you have found the appropriate values for the steering of your robot car, note these values in the table above.

7.3.3 Calibrate speed controller:

If you have not yet connected the speed controller to the servo controller, you must now connect it to channel 0.

As with the calibration of the steering, you must now find the value at which your motor is at a standstill. To do this, start the following command again, but now with channel 0 for the speed controller.

Note: The tires of the Robote car are in the air.

Command: donkey calibrate --channel 0 --bus=1

When you have found the value for stop of the motor on your speed controller by trial and error then increase this value in steps of approx. 20. This way you can determine when the robot car starts to move forward. Keep increasing the PWM value until the wheels start turning at top speed.

If you are looking for the value for reverse gear then you always have to start with the value for engine stop and then decrease this value in steps of about 20 until your model car goes fast in reverse.

For my model, the values are in the following range.

	Example values:	**Your values:**
Positive acceleration:	500	
Engine stop:	400	
Negative acceleration:	300	

Once you have found the appropriate values for the speed controller of your robot car, note them down in the table above.

Note: It is possible that the value for the positive acceleration is smaller than the value for the negative acceleration. If this is the case, the direction with which you accelerate or decelerate your robot car is later reversed on the right joystick of the gamepad.

In the following section I will explain how you can swap the driving direction via configuration in the software without possibly rewiring the drive motor.

7.3.4 Reversed direction during acceleration

If your robot car should unfortunately drive backwards instead of forwards and upside down then you still have to make a small adjustment in the part actuator.py.

Open the file actuator.py in the folder ~/projects/donkeycar/parts again with the text editor of your choice and search for the class "PWMThrottle". In this you only have to change the sign of the passed value for the two variables MIN_THROTTLE and MAX_THROTTLE.

Modify the PWMThrottle class as follows:

class PWMThrottle:

- …
- MIN_THROTTLE = 1
- MAX_THROTTLE = -1

From now on, your robot car should move forward when you move the right joystick forward and backward when you pull the joystick towards you.

7.3.5 Configuration file myconfig.py

The central configuration file of your robot car is myconfig.py. This file is not overwritten even when upgrading the Donkey Car framework and is located in the ~/mycar folder.

You will now make the essential adjustments in myconfig.py. I will guide you step by step through the configuration. Open the file e.g. over a SSH connection in the text editor Nano or an editor of your choice.

Command: nano ~/mycar/myconfig.py

The first change is made in the item #PATHS. Here the paths are defined in which the training data and later the trained neural network or the autopilot are stored.

- #PATHS
- CAR_PATH = PACKAGE_PATH = os.path.dirname(os.path.realpath(__file__))
- DATA_PATH = os.path.join(CAR_PATH, 'data')
- MODELS_PATH = os.path.join(CAR_PATH, 'models')

This is followed by the setting for the drive loop. The Drive Loop determines how fast the loop is run through which saves the current image of the camera + steering angle + speed as *.jpg and *.JSON file.

- #VEHICLE
- DRIVE_LOOP_HZ = 20
- MAX_LOOPS = None

The following settings for the camera are important. Here you have to change the entry from PICAM to CSIC for the Jetson Nano. Depending on your camera model and how the camera is mounted, it could be that the image is displayed upside down. You can still rotate and adjust the image if you want to. With the parameter "CSIC_CAM_GSTREAMER_FLIP_PARM" you can rotate the camera image. Even if the image is upside down, for example, this does not disturb the neural network later. Only for us as humans the image is then a bit more difficult to interpret.

- #CAMERA

- CAMERA_TYPE = " CSIC "
- IMAGE_W = 160
- IMAGE_H = 120
- IMAGE_DEPTH = 3
- CAMERA_FRAMERATE = DRIVE_LOOP_HZ
- CSIC_CAM_GSTREAMER_FLIP_PARM = 0

The Jetson Nano expects the I²C bus on bus number 1 instead of 0. Therefore the line PCA9685_I2C_BUSNUM must be adapted accordingly.

- #9865, over rides only if needed, ie. TX2..
- …
- PCA9685_I2C_BUSNUM = 1

Since the Donkey Car Framework supports different drive types and steering systems you have to define which of the different possibilities you use. If you have installed a classic model ESC electronic speed controller and steering servo, you must now select "SERVO_ESC".

- #DRIVETRAIN
- ….
- DRIVE_TRAIN_TYPE = "SERVO_ESC"

Now please open your notes a few pages before and transfer the PWM values for the steering and the speed controller into myconfig.py below.

- #STEERING
- STEERING_CHANNEL = 1
- STEERING_LEFT_PWM = 460
- STEERING_RIGHT_PWM = 290
-
- #THROTTLE
- THROTTLE_CHANNEL = 0
- THROTTLE_FORWARD_PWM = 500
- THROTTLE_STOPPED_PWM = 370
- THROTTLE_REVERSE_PWM = 220

Since you will later train the neural network on the Jetson Nano you have to make the adjustments for this part of the myconfig.py file as well. I always train my models with the default settings as follows.

- #TRAINING
- DEFAULT_MODEL_TYPE = 'linear
- BATCH_SIZE = 128
- TRAIN_TEST_SPLIT = 0.8
- MAX_EPOCHS = 100
- SHOW_PLOT = True
- VEBOSE_TRAIN = True
- USE_EARLY_STOP = True
- EARLY_STOP_PATIENCE = 5
- MIN_DELTA = .0005
- PRINT_MODEL_SUMMARY = True
- OPTIMIZER = None

- LEARNING_RATE = 0.001
- LEARNING_RATE_DECAY = 0.0

Two parameters from the TRAINING section of the configuration should be understood a little better so that when you train the neural net later, you will better understand the output during training.

MAX_EPOCHS: During a training epoch, the entire training dataset is presented to the neural net once during the training and the model is adapted. Thus, the variable MAX_EPOCHS = 100 determines how many training epochs are run through until the training is completed.

EARLY_STOP_PATIENCE: During the training of the neural net it will often happen that after approx. 25 training epochs the message Early Stopping is displayed and the training of the neural net is stopped. The variable EARLY_STOP_PATIENCE = 5 specifies that the neural net stops training early if it has not improved in five training epochs in a row.

Since I recommended at the beginning to always create the training data with a gamepad, the gamepad must also be permanently activated at the following point in myconfig.py.

Note: If you set the setting to "True" for the use of the gamepad, the web interface of the Donkey Car Framework will no longer be started. If you want to start and access the web interface, you have to set the setting to "False" again. However, you will then not be able to use the joystick or gamepad.

- #JOYSTICK
- USE_JOYSTICK_AS_DEFAULT = True

Then follow around the control of the robot car with the gamepad three settings that you can adjust as follows. I use a PS4 controller with Sony USB receiver and have selected ps4 as type in the configuration. Depending on your controller, you can select a variety of types here (see inline documentation). The EasySMX controller also works very well as type ps4.

- JOYSTICK_MAX_THROTTLE = 0.8
- AUTO_RECORD_ON_THROTTLE = True
- CONTROLLER_TYPE='ps4'

Now save the changes to the myconfig.py file.

Now you have made all configurations and can start the Donkey Car Framework in drive mode for the first time. In the active drive mode you control your robot car manually with the gamepad and record the first training data. The recorded images and json files can be found in the folder ~/mycar/data and there in individual subfolders that always start with tub_* in the name.

Start the drive mode with the following command.

Command: python ~/mycar/manage.py drive

7.4 ATTENTION: Error in part camera.py

It can happen that the Donkey Car Framework now crashes when loading. The error message indicates, for example, that something is not working properly in the camery.py part with the video stream from the camera. While searching for the cause of the error, I found out that the initial resolution for the GStreamer that processes the video image is physically too high for the wide-angle camera I installed.

The solution I found was to significantly reduce the resolution in the file "camera.py" and there in the class "class CSICamera(BaseCamera):".

The following image shows the error message that was issued in my case.

```
GST_ARGUS: Cleaning up
CONSUMER: Done Success
GST_ARGUS: Done Success
[ WARN:0] global /home/nvidia/host/build_opencv/nv_opencv/modules/videoio/src/cap_gstreamer.cpp (886) open OpenCV | GStreamer warning: unable to start pipeline
[ WARN:0] global /home/nvidia/host/build_opencv/nv_opencv/modules/videoio/src/cap_gstreamer.cpp (480) isPipelinePlaying OpenCV | GStreamer warning: GStreamer: pipeline have not been created
Exception in thread Thread-1:
Traceback (most recent call last):
  File "/usr/lib/python3.6/threading.py", line 916, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.6/threading.py", line 864, in run
    self._target(*self._args, **self._kwargs)
  File "/home/donkey/projects/donkeycar/donkeycar/parts/camera.py", line 174, in update
    self.init_camera()
  File "/home/donkey/projects/donkeycar/donkeycar/parts/camera.py", line 169, in init_camera
    self.poll_camera()
  File "/home/donkey/projects/donkeycar/donkeycar/parts/camera.py", line 181, in poll_camera
    self.frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
cv2.error: OpenCV(4.1.1) /home/nvidia/host/build_opencv/nv_opencv/modules/imgproc/src/color.cpp:182: error: (-215:Assertion failed) !_src.empty() in function 'cvtColor'
```

Figure 7.3: Error message camery.px file

You can find the camera.py file in the ~/projects/donkeycar/donkeycar/parts folder. You had previously downloaded the GitHub project to this folder.

Now open the file camery.py and search for the class "class CSICamera(BaseCamera)":

There you will find the following line which you have to change in the parameters "capture_width" and "capture_height".

- Original line:
- def __init__(self, image_w=160, image_h=120, image_d=3, capture_width=3280, capture_height=2464, framerate=60, gstreamer_flip=0):

- Adjusted line:
- def __init__(self, image_w=160, image_h=120, image_d=3, capture_width=160, capture_height=120, framerate=60, gstreamer_flip=0):

After you have made this adjustment, save the camery.py file and restart the Donkey Car Framework. The error should no longer occur.

Restart the drive mode with the following command and you should now be able to control the donkey car.

Command: python ~/mycar/manage.py drive

Note: It may be that you can steer the donkey car with the PS4 controller but you cannot accelerate it. Then please read in the chapter "Gamepad configuration" which settings you can adjust so that your PS4 controller or similar model works. Here you have to read and test depending on the model.

If everything has worked so far, i.e. you can steer and accelerate with the gamepad, then place the donkey car on the ground. Now turn a few laps and familiarize yourself with the control of the robot car via the gamepad.

7.5 Camera with faulty color display

It can happen that the camera used is not exactly supported by the operating system and the drivers or configurations available there. For example, I use a wide-angle camera in my Donkey Car models, where the edges of the image on the left and right are colored red.

There are always configuration files from the various camera manufacturers that set the parameters for the operating system in such a way that the image looks much better.

The following picture shows the problem quite well. On the left side you can see a picture of the camera with the wrong colors. On the right side of the image you can see the camera image after a *.isp file, which was specially created for the camera, was imported.

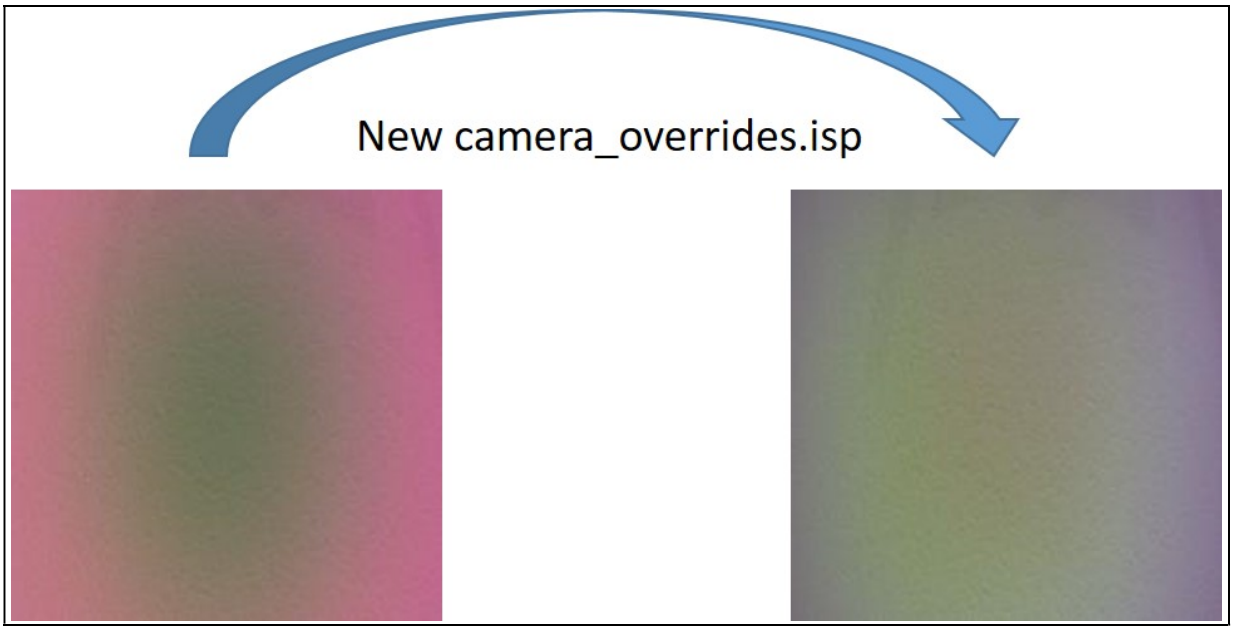


Figure 7.4: camery_overrides.isp configuration

The camera_overrides.isp file I used is available for download on my blog at the following URL.

Download: *https://custom-build-robots.com/jetson-nano-download-de*

After you have downloaded and unzipped the *.zip file you can continue. First of all you have to delete two files before using the new camera_overrides.isp file. Otherwise it will not work with the correct display of the colors.

Note: Before deleting or overwriting the files, remember to create a backup in a backup folder, for example.

To do this, execute the following two commands:

Command: sudo rm /var/nvidia/nvcam/settings/nvcam_cache_*

Command: sudo rm /var/nvidia/nvcam/settings/serial_no_*

Now change to the folder in the terminal window where you copied the previously downloaded and unzipped camera_overrides.isp file.

Then execute the following command which copies the camera_overrides.isp file to the "/var/nvidia/nvcam/settings/" folder.

Command: sudo cp camera_overrides.isp /var/nvidia/nvcam/settings/

After you have copied the file "camera_overrides.isp" you have to adjust the file permissions. You do this with the following command:

Command: sudo chmod 664 /var/nvidia/nvcam/settings/camera_overrides.isp

Afterwards the file camera_overrides.isp must be assigned to the user root so that he can access the file when starting the Jetson Nano. Therefore, execute the following command now.

Command: sudo chown root:root /var/nvidia/nvcam/settings/camera_overrides.isp

The neural network will not care about the not quite matching color in the images and will work just as well as with the corrected color representation. But when you, as a human, later create videos from the thousands of image files for quality checks of the training data, it is nice to see the correct colors.

7.6 Practice driving

Now practice driving your robot car together with the gamepad. You may notice that your chassis does not track straight and always pulls in one direction. Then correct this by adjusting and testing the PWM values for the steering in the myconfig.py file.

Note: In case the steering left and right should be swapped, the easiest solution is to swap the steering angle in the myconfig.py file. More about this in the chapter "Run your autopilot for the first time ".

For the recording of error-free training data for the later training of the neural network, it is very important that you are familiar with the steering, the speed, in short, with the control of your robot car.

7.7 Activate OLED display

Maybe you have installed an OLED display on your robot car, as recommended by me. Then I would like to explain in this section how to install the necessary software to get the OLED display working. Adafruit offers a very good library that can be used together with the Jetson Nano and the I²C bus.

First, you need to download the software from GitHub. To do this, run the following two commands.

Command: cd ~/

Command: git clone *https://github.com/adafruit/Adafruit_Python_SSD1306.git*

Now change to the folder "Adafruit_Python_SSD1306".

Command: cd Adafruit_Python_SSD1306

Then install the software on your Jetson Nano using the command below.

Command: sudo python3 setup.py install

It could be that the image library is still missing which is needed so that the display on the OLED display really works. You can install it with the following command.

Command: pip install image

Now restart the Jetson Nano and then execute the following command to get all connected devices on the I²C bus displayed.

Command: sudo i2cdetect -y -r 1

You should see the servo controller with the addresses 40/70 and the OLED display with the address 3c. For me the output looks like this and highlighted in orange is the OLED display with its I²C address.

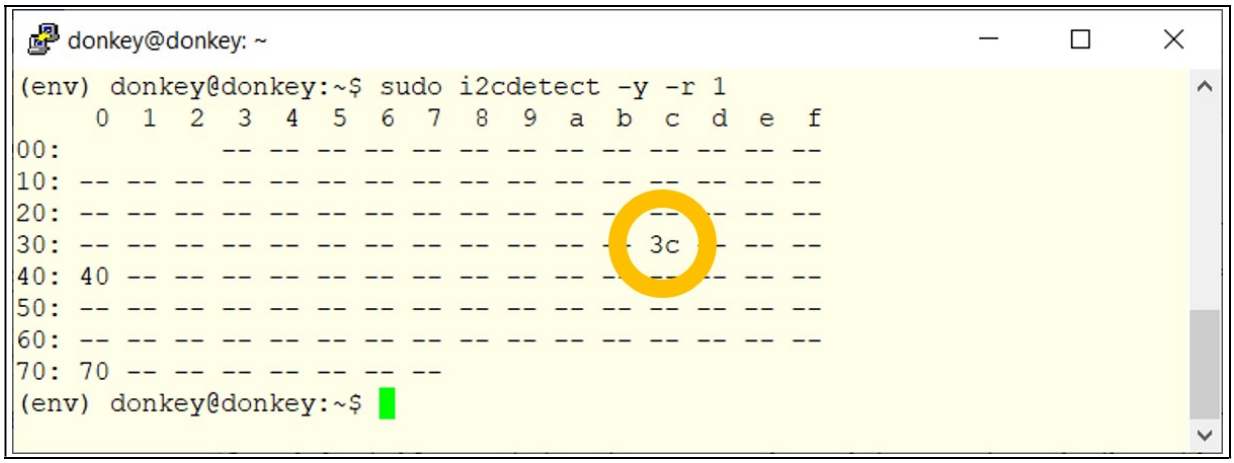


Figure 7.5: I2C bus display

Since the small example programs that Adafruit provides are not yet adapted for the Jetson Nano, I provide you with the adapted stats_nano.py program without going into the details of the adaptation here. In the source code of the Python program stats_nano.py you can see the small adjustments in comparison to the original stats.py program from Adafruit.

Download the corresponding *.zip file from the following link and unzip it on your Jetson Nano.

URL: *https://custom-build-robots.com/jetson-nano-download-de*

Place the two programs "start-oled.sh" and "stats.py" from the previously downloaded OLED_Display_Software.zip file in the folder ~/projects in which you have already downloaded the Donkey Car Framework.

So that the Python program for the display is always started automatically you must still deposit the script with the file name start-oled.sh in the /etc/crontab. The crontab I used is included as an example in the "OLED_Display_Software.zip" file.

Note: Make absolutely sure that the paths used in the start-oled.sh script and subsequent ones in the crontab match your installation. Otherwise, errors will occur.

After you have saved the script in the projects folder, you need to change the file permissions so that it can be executed from /etc/crontab.

Command: sudo chmod 777 start-oled.sh

The start-oled.sh script itself is very short and also starts the OLED display program in a virtual session. Below you can see the structure of the start-oled.sh script file:

```
#!/bin/bash
source ~/env/bin/activate
python /home/donkey/projects/stats.py >> /home/donkey/projects/oled-sh.log 2>&1 &
```

7.7.1 Create the start-oled.sh script yourself

If you want to create the script start-oled.sh yourself then execute e.g. the following command to create the script with the text editor nano.

Command: nano start-oled.sh

After you have saved the script in the home folder you have to change the file permissions so that it can be executed from /etc/crontab.

Command: sudo chmod 777 start-oled.sh

If an error occurs within the start-oled.sh script while executing it, it writes a log file named *oled-sh.log to the* /home/donkey/ folder or, if you have customized the path, to the folder you specified.

Now you have to add the following line to /etc/crontab to execute the script start-oled.sh automatically at every reboot. Again, please check the paths.

```
python /home/donkey/projects/stats.py >> /home/donkey/projects/oled-sh.log 2>&1 &
```

If you now restart the Jetson Nano, you should see a message on the OLED display. If you don't see anything then you can have a look at the file *oled-sh.log* or *oled-crontab.log in the* projects folder. These two log files are always written when the script is executed via the crontab.

Your /etc/crontab should now look like the following.

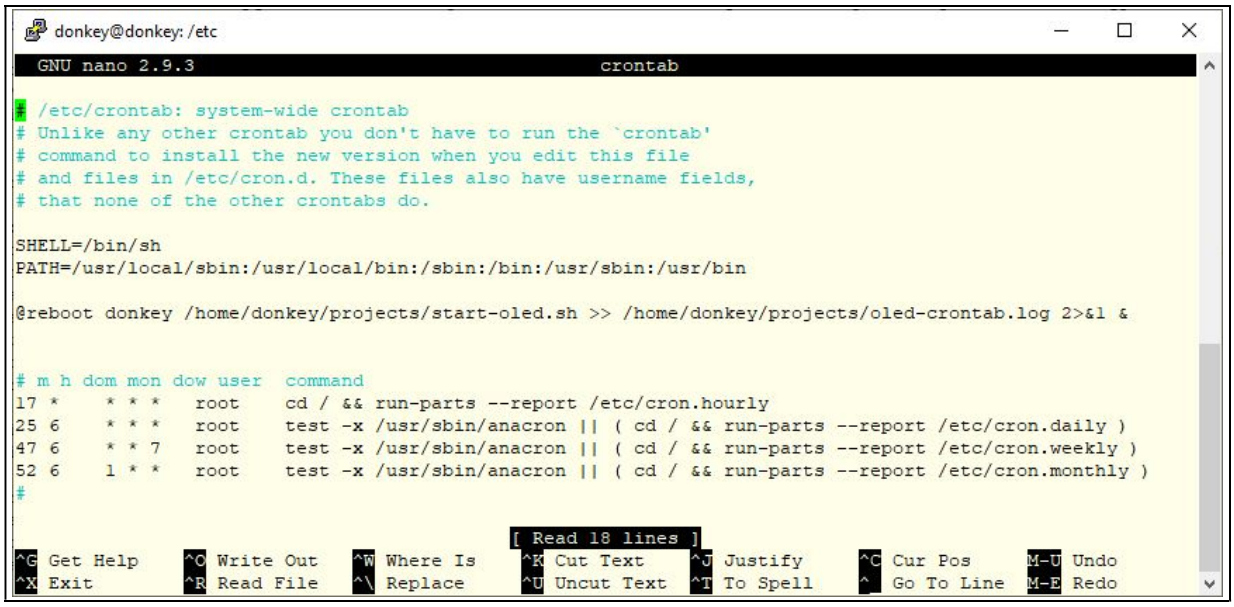


Figure 7.6: OLED display customization /etc/crontab

7.8 What you have achieved up to here.

You have installed all necessary programs and libraries, configured the steering and the speed controller and you have already driven the first laps. The OLED display shows the IP address and other information and you are now well prepared for the next step - training the neural network with your own training data.

I very much appreciate your feedback on this chapter. Write me your thoughts, questions and suggestions to the following e-mail address: *ebook@custom-build-robots.com*

8 Preparation for the training data recording and trainings of the autopilot

You have made the first rounds with your robot car and are eager to learn how to train the neural network and drive autonomously. Then let's get into the fun of autonomous driving model cars.

In the previous chapter, you built the robot car, installed the software and drove a few laps to test if everything works. You are familiar with the controls and the driving behavior of your robot car and will now learn how to record training data. But before we really get started, here is a short section with recommendations on track construction and the further process until the trained neural network is ready.

8.1 Build routes for the robot car

The tracks for recording the training data can be set up individually. These do not have to be standardized as long as you are not training for a competition where every second counts. You can quickly and easily create a track on the floor using inexpensive adhesive tape from a paint store.

> **Note:** But be careful that it does not happen to you like a colleague of mine who was full of enthusiasm at home on the weekend on the freshly laid parquet had stuck a tape. In the evening when he tore the tape off the floor, he removed not only the tape but also parts of the parquet sealant. The great advantage of this really stupid action was that the roadway was thus permanently available until the next sanding and painting.

The disadvantage with the tape is the cost and the large amount of waste that is generated each time the track is removed. This has moved me to think a little more sustainable to save the environment and the cost. Also, I no longer felt like having to stick tape on my knees.

If you work with a tape or thick rope, a simple line that you stick on is sufficient or even two tracks at a distance of about 50cm so that your robot car can drive well between the adhesive tracks. You should not make the radius of the curves too tight so that you can get through the curves well.

A route that I always use from its principle looks like this.

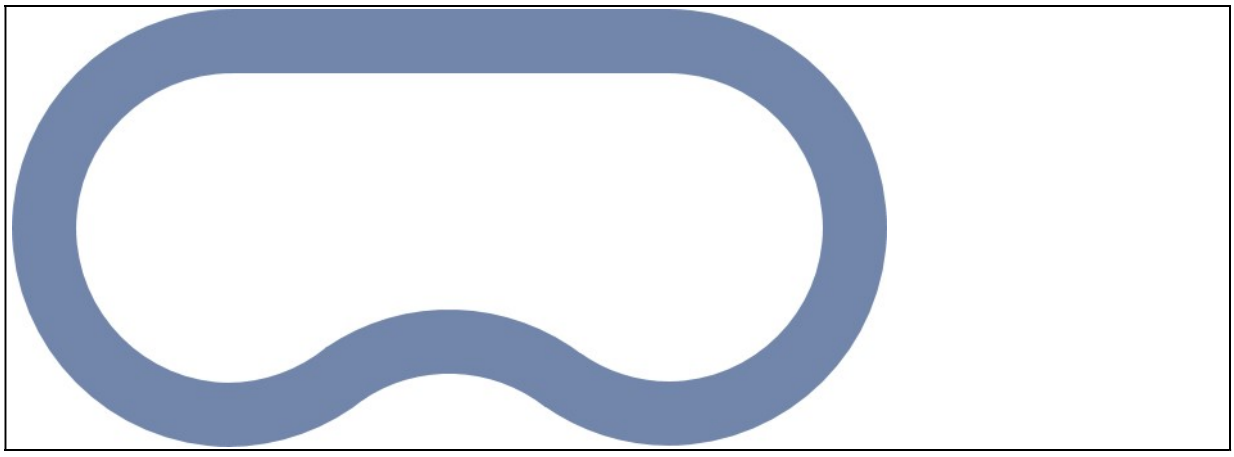


Figure 8.1: Race track idea

This path avoids the basic errors, such as that only one steering angle always prevails in one direction of travel.

I bought 100m seam tape as B-goods with a width of about 3 cm to 5 cm on the Internet for reasons of cost and waste for the track construction. Where I recommend the 3cm variant because this throws less wrinkles when modeling curves. On this seam tape I have again sewn Beleiband with which actually curtains are weighted. So I can lay beautiful track with curves and straight lines very quickly while standing by unwinding the tape from a large roll. Due to the lead tape, the seam tape lies nicely on the

ground without throwing big wrinkles with the 3cm version. When the robot car drives over the tape, it does not warp but remains in place. In the open air, it cannot be blown away by the wind. I also built an automatic winding mechanism with a small electric motor to protect my back.

Figure 8.2: Sewing the race track marking

A track can also be built quite simply, for example, outdoors with leaves weighted down with chestnuts, post-IT papers in the office or coffee mugs. It is important that the course of the track stands out well from the surroundings so that you as a human can steer the robot car around the track without driving errors and thus receive error-free training data. Later on the autopilot will be able to follow the track after it has been trained. It does not have to be that the trained neural network really follows the seam tape or coffee cups. Because a neural network is also "lazy" or optimized and, thanks to Deep Learning, looks for its own waymarks on which it orients itself.

In an open-plan office with ceiling lighting, it is also possible to point the camera of the robot car vertically upwards. It is important that the ceiling in this setup has a pattern that is not too symmetrical but offers some variety. The track marking on the floor is then only important for the human who keeps the robot car on course while recording the training data. The robot car then follows the track based on the position of the lamps and patterns in the ceiling. The route markings on the floor then play no role for the autopilot. If you switch off the lamps, the neural network will no longer be able to control the car because the marking on the ceiling has changed or is missing completely.

In general, changes in the environment of the draw frame play a major role in its success or failure. For example, changing lighting due to the sun shining into the room with the track, i.e. changes in the direct environment of the track, play a decisive role. So I always recommend to record the training data also in different lighting conditions to make the later trained autopilot as robust as possible against changing environmental factors. For example, if you go to meetups with your robot car or to a competition, keep in mind that there will be many spectators dynamically changing the setup around the track.

The track should not only be a circle but should contain curves, straights and turns. If you only drive in circles, the neural network will not learn to steer correctly, since you will most likely always drive the track with approximately the same steering angle.

In a later chapter I will give you a script which changes the brightness of the images and saves them together with the *.json files. This way you get more training data faster to make the autopilot as robust as possible.

I had once set up a track in the garden on the lawn with my seam tape. When I had wound up the seam tape again in the evening, I started the autonomous mode of my robot car again. It still followed the route that was actually no longer apparent. From this experience, I conclude that the trained autopilot had used, for example, the pattern and shading of the lawn as a reference for the steering angle and speed.

You can also teach the autopilot to avoid obstacles by placing coffee cups, for example. To do this, keep changing the position of the coffee mugs on the track and record training data again. Do this about 10 times per coffee cup and record three laps each time. If you train the neural network with this data, it will most likely have learned to avoid coffee cups at any position on the track. As you can see, even with such a small task as dodging coffee cups, you have to put a lot of effort into generating the training data. Therefore, structured and labeled data is the oil of our future economy.

8.2 Record good training data

It is hard to say what is good and what is bad training data. It depends on what exactly you are trying to do. But one thing will happen for sure, that you give your training data to your human BIAs, so your individual behavior. If you drive very conservatively around the track, the neural network will do the same later. If you drive at a constant speed with all recorded training data and do not accelerate or brake then it can happen that your autopilot has not learned to accelerate and does not drive off at the start. In such a case, if you push the car briefly with your foot, for example, it will suddenly drive its lap autonomously at a constant, unchanging speed.

Therefore, I always recommend adding as much variety to the recorded training data as possible. Accelerate and brake again and again while creating the training data. Drive beyond the track marker and immediately back again so that the neural network learns what to do in such a case. Above all, drive the track not only in one direction but also in the other. If you always drive the course clockwise, for example, then the pilot cannot autonomously drive the course counterclockwise later on. But just try it out for yourself.

With these tips from my experience, you should be able to generate good training data for training your autopilot with which your neural network or autopilot learns autonomous driving.

8.3 Gamepad button assignment

If you use a gamepad to control the robot car as recommended, then you still need to find out how the button mapping is configured exactly for your gamepad. You have already learned about the configuration in the ~/mycar/myconfig. py file and seen the different supported gamepads there.

Interesting is with which keys you can change the operating mode in the autonomous operation of your robot car and which functions there are besides that which offer you the one or other relief with the operation of the Donkey Car frameworks. You can check the key assignment in the controller.py file yourself and also change it there for your controller if you like.

A detailed description of how this works can be found in chapter 3.2.5

8.4 Alternative: The web control

If you don't use a gamepad but want to control and operate the robot car first via the web interface then you have to run the Donkey Car framework again with the drive parameter.

> Note**:** Please note that you have to set USE_JOYSTICK_AS_DEFAULT for the joystick to "False" in ~/mycar/myconfig. py to start the web interface.

Now start the Donkey Car Framework with the following command.

> **Command:** python ~/mycar/manage.py drive

After the framework is started, call the following URL in the browser.

> **URL:** <IP address robot car>:8887

The web interface that shows a live video image from the robot car looks like the image below.

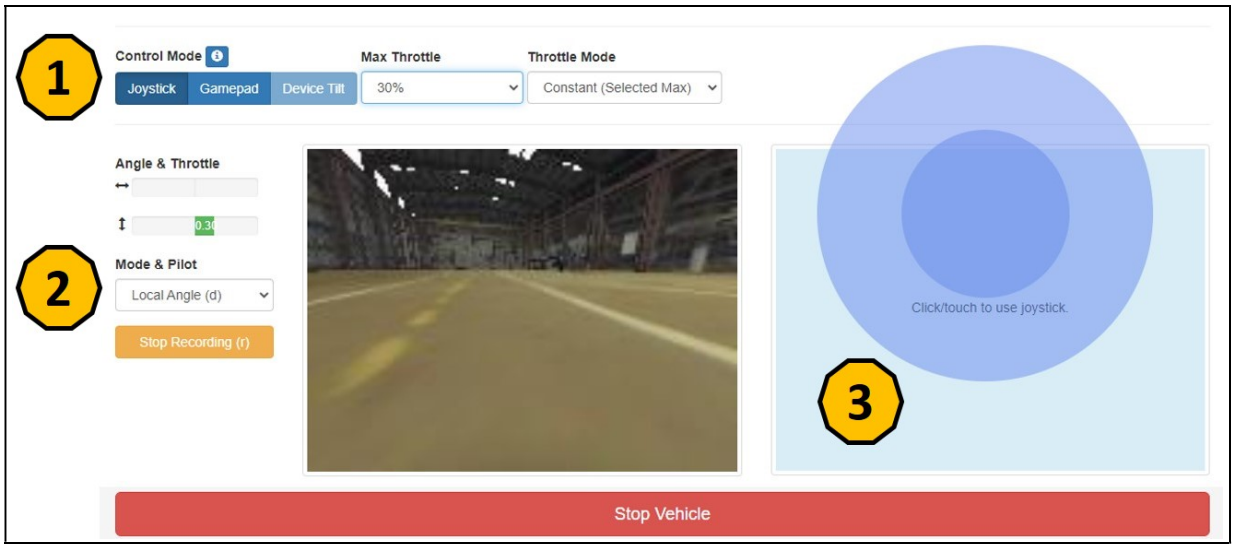


Figure 8.3: Donkey Car web interface

The illustration shows the individual functions well, such as the different control modes with the joystick, gamepad or by tilting a smartphone (device tilt).

The three most important functions have been numbered in the image.

1. control mode

Here you can change the mode how you want to control your robot car. If you are using a current Chrome browser on your PC and have a joystick or gamepad connected to your PC, you can now use it to control the robot car remotely via the browser. With the "Device Tilt" function you can control the robot car with your smartphone and its built-in position sensor by tilting the smartphone accordingly.

2. fashion & pilot

With the dropdown box "Select Mode" you can switch between the three modes User, Local Pilot and Local Angle. Later I will go into the meaning of the individual modes.

3. Click/touch to use joystic.

On the right you see a field within which you can control the car by holding down the left mouse button on the PC. This function also works quite well on a smartphone or tablet with the finger. Possibly this is also more of a generational issue.

Furthermore, you have the Max Throttle function on the web interface with an associated dropdown box. Here you can set the maximum speed that the robot car should drive. With the dropdown box Throttle Mode you determine whether you control the speed yourself (in User Mode) or whether the robot car should drive constantly fast with the speed selected under Max Throttle.

8.5 Record training data

Now that you've read a lot of theory, you're probably eager to record your first training data. Set up your track and do a few practice laps. Before that, delete all test and game data in the ~/mycar/data directory that was recorded there when you were still practicing driving.

The command to start the drive mode of the robot car is as follows.

Command: python ~/mycar/manage.py drive

Now drive approx. 7 clockwise and 7 counterclockwise laps on your racetrack. It is important that you have created approx. 8,000 data sets. A dataset always consists of an image and the corresponding JSON file. So in total you should then have about 16,000 files with a meta.json file.

The recorded training data consisting of images and at least one catalog file associated with the run.

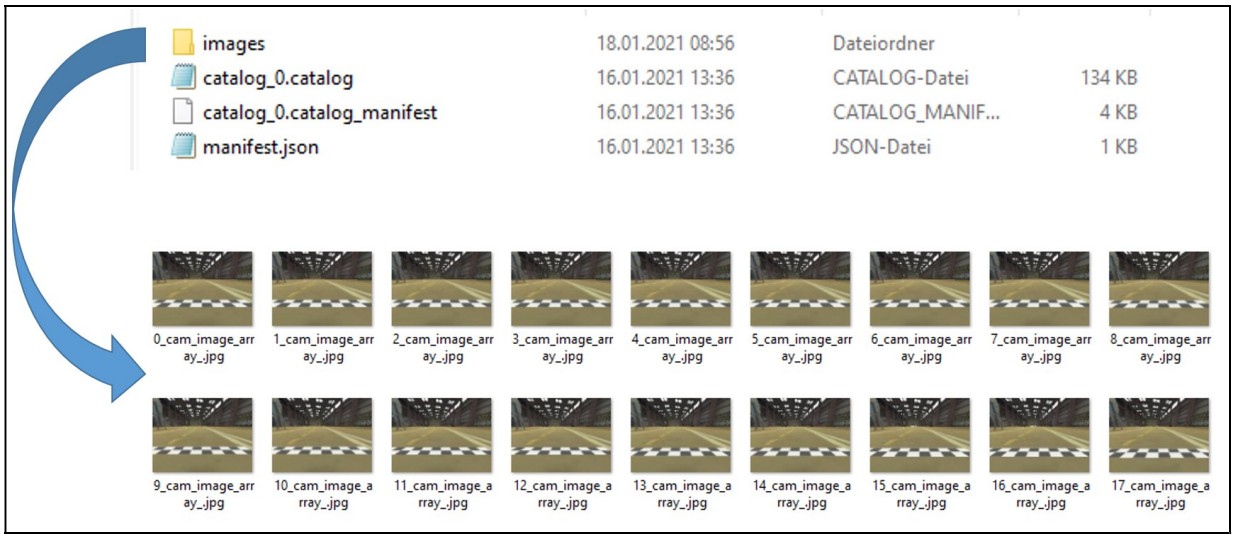


Figure 8.4: Donkey Car training data

If you have not made any gross driving errors when generating the training data and have only driven beyond the course marker from time to time, then in my experience you do not need to clean up any data. However, in the following section I would like to provide you with a small script that makes it easier for you to quality assure your training data.

8.5.1 Data quality assurance

You can use the auto_ffmpeg.sh script to create a quality assurance (QA) video from the individual images in the Tub folders. With the video, you can perform QA of your recorded training data faster than trying to view thousands of images individually.

The script creates a video from all subfolders in the ~/mycar/data/ directory and the images stored there with the file name of the respective folder. The videos are stored in the ~/mycar/data/ directory.

You can download the script from the following page. Place it in the ~/mycar/ folder.

Download: *https://custom-build-robots.com/jetson-nano-download-de*

You may need to change the file permissions of the script to run it.

Command: sudo chmod 777 ~/mycar/ auto_ffmpeg.sh

Start the script to generate the videos with the following command:

Command: sh auto_ffmpeg.sh

After the script has run through, you should be able to play the videos with the VLC player, for example. Transfer the videos to your PC with e.g. WinSCP a free SFTP client software if you are a Windows user.

Delete training data that show gross driving errors such as accidents or long driving off the track markings. If you are satisfied with your remaining training data and you have about 8,000 records of good training data in the folder ~/mycar/data/ then you can now train the neural network. You will learn how exactly this works in the following chapter.

8.6 Train your autopilot

You now have at least one Tub folder in the ~/mycar/data/ folder that contains your training data. All folders with training data or parts of them that did not pass your quality assurance have been deleted by you.

Note: Please remember to run the Jetson Nano with the external power supply now so that the maximum GPU power is available for the training of the neural network. Otherwise, the training may take significantly longer.

The Jetson Nano is not ideal for training neural networks. A PC with an appropriate GPU or a strong CPU and a RAM of more than 12GB would be much better suited. To make matters worse, not all libraries for the Jetson Nano are available that are currently used by the Donkey Car Framework or the community.

Therefore you still have to download from my website once the train_nano.py file and the training_nano.py file. On the following page you will find the corresponding ***Nano_training.zip*** file.

URL: *https://custom-build-robots.com/donkey-car/donkey-car-framework-download-seite/13982*

Unzip the ZIP file e.g. in the home folder of your user.

train_nano.py

Put the file ***train_nano.py*** into the folder **~/mycar.**

training_nano.py

Place the ***training_nano.py*** file in the **~/projects/donkeycar/donkeycar/pipeline** folder.

If you have stored both files accordingly then it should work with the training pipeline adapted for the Jetson Nano to train the own neural network.

You start the training of the autopilot with the following command. All tub folders in the ~/mycar/data/ directory are used for the training with this command. The autopilot created at the end of the training is named mypilot.h5 and is located in the path ~/mycar/models/.

Command: python train_nano.py --tubs=data/tub_2_21-01-16/,data/tub_1_21-01-16/,data/tub_3_21-01-16/ --model models/mypilot.h5

You should now see the following output in the console when the training is started.

```
(env) donkey@donkey:~$ cd mycar/
(env) donkey@donkey:~/mycar$ python train_nano.py --tubs=data/tub_test/,data/tub_2_21-01-16/,data/tub_1_21-01-16/,data/tub_3_21-01-16/ --model models/big.h5
 _____                  _____                    _____
___  /_ _____ ___ ____ ___  /_ ___  ____ ____ ___ /_  /_ ___  ____
__  __ \_  __ \_  __ \_  //_/  _ \_  / / /    _  /    _ \__ \_  / / /
_  /_/ /  /_/ /  / / /  ,<  /  __/  /_/ /     / /___  / /_/ /_  /_/ /
/_____/ \____//_/ /_//_/|_| \___/\__, /      \____/  \__,_/ /_/
                                /____/

using donkey v4.1.0-dev ...
2021-01-18 07:41:56.695689: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcudart.so.10.2
loading config file: /home/donkey/mycar/config.py
loading personal config over-rides from myconfig.py
"get_model_by_type" model Type is: linear
Created KerasLinear
2021-01-18 07:42:03.829779: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcuda.so.1
2021-01-18 07:42:03.849410: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:948] ARM64 does not support NUMA - returning NUMA node zero
2021-01-18 07:42:03.849548: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1561] Found device 0 with properties:
pciBusID: 0000:00:00.0 name: NVIDIA Tegra X1 computeCapability: 5.3
coreClock: 0.9216GHz coreCount: 1 deviceMemorySize: 3.86GiB deviceMemoryBandwidth: 194.55MiB/s
2021-01-18 07:42:03.849622: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcudart.so.10.2
2021-01-18 07:42:03.865919: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcublas.so.10
2021-01-18 07:42:03.881960: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcufft.so.10
2021-01-18 07:42:03.888081: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcurand.so.10
2021-01-18 07:42:03.904888: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcusolver.so.10
2021-01-18 07:42:03.915645: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcusparse.so.10
2021-01-18 07:42:03.917765: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcudnn.so.8
```

Figure 8.5: Jetson Nano TensorFlow Training

If the autopilot does not improve during five training epochs in a row, the training is automatically aborted and an "early stopping" message is displayed. You have already learned about the parameter EARLY_STOP_PATIENCE = 5 and you can set it e.g. to 7 and try if you get better results of your autopilot. In my experience the training data and here again the number of data sets as well as the diversity in the data play a decisive role to get a better autopilot.

At the end of the training you should get an output like this.

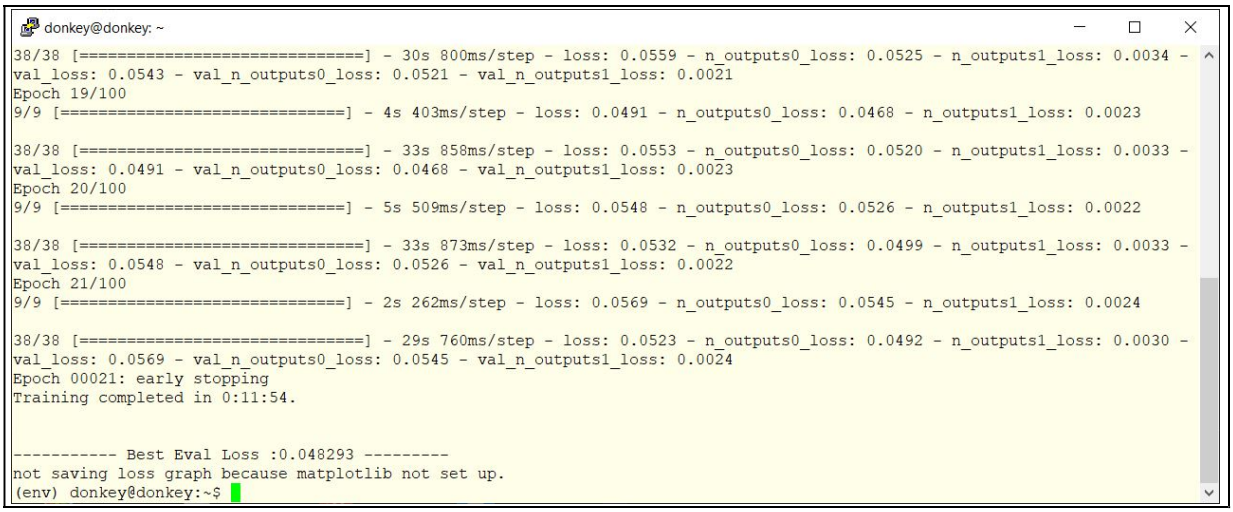


Figure 8.6: Autopilot successfully trained

8.7 Run your autopilot for the first time

Now that you have successfully trained your first autopilot, run it. It is important that you have activated the use of a gamepad in the myconfig. py file.

With the following command you now start the drive mode with the autopilot you have trained which here is named mypilot.h5.

Command: python manage.py drive --model ~/mycar/models/mypilot.h5

Now, for the first time, nothing will happen and the car will stand still. The drive mode knows three modes between which you can switch with the "Share" button on your PS4 controller or with the "Back" button of the "EasySMX Controller".

The drive mode knows the following three modes

User: You can manually control the robot car.

Local Angle: The autopilot steers and you control the speed.

Local Pilot: The autopilot steers and controls the speed autonomously.

Now switch to the "Local" mode and let your robot car drive autonomously. In the terminal window you will always see the currently selected mode. If the autopilot makes a driving error during autonomous driving, you can switch to User mode by pressing the Share key or Back key and thus take back control of the robot car.

Alternative web control: If you use the web control, you have to change the mode in the dropdown box "Mode & Pilot" to "Local Pilot" so that your robot car turns its rounds autonomously.

If you are not satisfied with the driving characteristics of the autopilot, then record training data again and train your model again. You may need to delete some of the existing training data.

Note: A lot of data does not always help much. Therefore, start with 8,000 to a maximum of 10,000 data sets to first get a feel for how a neural network learns.

8.8 Help my robot car does not start automatically

In several robot cars I had already built, I noticed the phenomenon that in "Local" mode the robot car did not start. The Donkey Car Framework version 2.5.x was used for these vehicles. If I gave the robot car a short push, it could continue to drive on its own. If you also have this problem, you can activate and increase a constant speed with e.g. the key "L1" and reduce it again with the key "R1" and your robot car should be able to start.

I have not yet been able to isolate the problem, but with the latest Donkey Car version 3.x I have not observed this problem.

8.9 What you have achieved so far

You have now learned a lot about the Donkey Car framework and how to build a track for the robot car, for example. What mistakes to avoid when generating good training data and how training the autopilot works. Your robot car has made its first autonomous driving attempts. Further, you may have done one or two tests with the different power settings of the Jetson Nano to ensure stable operation of the robot car.

I very much appreciate your feedback on this chapter. Write me your thoughts, questions and suggestions to the following e-mail address: *ebook@custom-build-robots.com*

9 Tips, tricks and further information about the robot car

Your robot car is already driving autonomously and you would like to hear a few tips. Then you've come to the right place and I'll tell you how I optimized my robot car and how I generate training data without actually recording it lap by lap.

9.1 Release additional working memory

The Jetson Nano with its 4GB Ram works very fast regarding the memory usage at the limit. Since you probably administer the robot car over an open SSH session with commands most of the time like I do, you don't really need the graphical user interface. Therefore you can deactivate it and start the operating system in text mode only. This way you have a few hundred MB more RAM free and achieve a slightly better performance of your robot car with little effort. How exactly this works I describe below.

To disable the graphical mode, the following command must be executed in the terminal window.

Command: sudo systemctl set-default multi-user.target

After rebooting the Jetson Nano, if it is connected to a monitor on the HDMI output, you should see the Jetson Nano boot up in text mode.

Note: I had connected my Jetson Nano to a monitor via the display port and after executing this command, I no longer saw a picture. Also the SSH login did not work. I then reinstalled the Jetson Nano. The next time I tried it, I found that the picture would have been available via the HDMI output.

With the following command you can now check how much memory is free.

Command: free mem

Now, about 3.42 GB of RAM should be free in the working memory. In comparison, with the graphical user interface active, only 2.73 GB of RAM was free in the working memory.

The following image shows the free memory without an active graphical user interface directly after the system start.

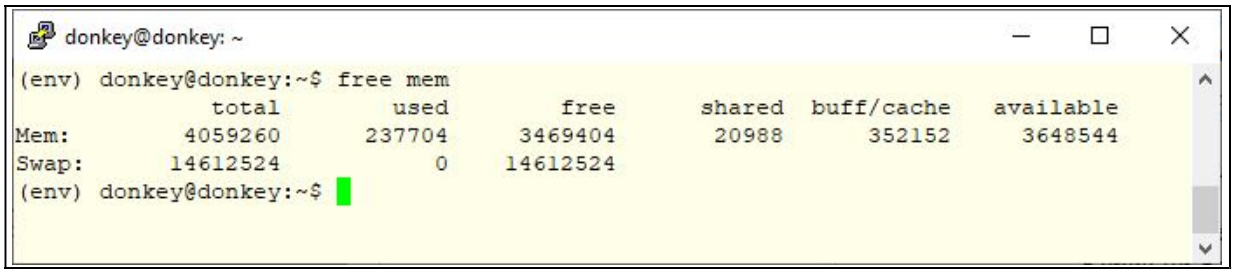


Figure 9.1: Display of the free working memory

If you want to reactivate the graphical mode, please execute the following command.

Befehl: sudo systemctl set-default graphical.target

If you have booted your Jetson Nano in text mode but want to start the graphical user interface manually, you can start it again with the following command.

Command: sudo systemctl start gdm3.service

Now, with this small adjustment, you have a little more memory available and so you may have achieved a little better performance.

9.2 Limiting the power consumption of the Jetson Nano

You can limit the power consumption of the Jetson Nano to 5W or 10W if you power the Jetson Nano via the micro USB interface. If you connect the power supply via the power connector, the Jetson Nano will run at full power and will need about 20W, i.e. 5V at 4A. Depending on how deep you are into self-

driving model cars and how much power you need for mobile operation, power the Jetson Nano via one of the two power connectors. If you use the power bank recommended by me, you can cut a USB cable and solder a plug for the DC socket of the Jetson Nano and thus get the maximum power for e.g. fast autonomous drives of the robot car from the Jetson Nano.

When training the autopilot on the Jetson Nano, always use full power and power the Jetson Nano through the extra power supply and DC jack. Remember to set the J48 jumper.

In mobile mode, I now assume that you have connected the Jetson Nano to the power bank via the micro USB interface. To limit the power consumption to 5W execute the following command in the console. With this you activate the POWER_MODEL with the ID=1.

Command: sudo nvpmodel -m1

If you want to have the maximum power available that you can demand from the micro USB port, then you can set the power consumption back up to the 10W. Execute the following command which activates the POWER_MODEL with ID=0.

Command: sudo nvpmodel -m0

In the file /etc/nvpmodel/nvpmodel_t210_jetson-nano.conf you can find the detailed power settings of the Jetson Nano and read the difference between the two modes m1 and m0 i.e. POWER_MODEL 1 and 0 respectively. The main difference between the modes is the number of CPUs used and the maximum allowed frequency of the CPUs. Exactly here you can also define your own mode if you are familiar with it and want to try it out.

If you want to see which power settings the Jetson Nano is currently working with, you can use the following command to display them.

Command: sudo jetson_clocks -show

Figure 9.2: Jetson Nano clock mode

9.3 Screen of the terminal multiplexer

Perhaps you already know Screen and are already using it. If not, then I would like to briefly explain the advantages of Screen when used in the robot car. Screen is a terminal multiplexer that allows you to start a terminal session that continues to run even if the SSH connection you are logged in to the robot car has been interrupted. You can always return to this session started in the Screen program. This has the decisive advantage that in case of a network problem and interruption of the SSH connection the robot car does not stop because the session does not crash so easily but continues to run and with it all started programs. Screen you install with the following command on your Jetson Nano.

Command: sudo apt-get install -y screen

If you are logged in to your robot car via SSH then use the following command to start a screen session with the name "session1".

Command: screen -S session1

After you have executed this command, you are in the screen session named "session1".

You can leave this session with the following command without terminating the session itself and the programs running in it.

Command: Ctrl + A + D

If you want to display all active screen sessions, execute the following command.

Command: screen -ls

You will get a list with the active screen sessions. If you now want to return to a specific session, such as the session "session1", you can do this with the following command.

Command: screen -r session1

Thus, in case of any connection problems, screen will help you not to have to restart the Donkey-Car framework all the time.

9.4 Manipulate training images

You have recorded the training data in bright light and now, after the autopilot has finished training, you find that the lighting conditions have changed. This is a typical problem that happened to me again and again with the result that the autopilot could no longer steer the robot car around the racetrack without errors. So that you don't have to wait or create different light conditions when creating training data I wrote a script that changes the already recorded images in their brightness and color saturation. If I train the autopilot with the original pictures and additionally with the changed pictures then my robot car drives very stable also with changing light conditions the Racetrack.

In the small script convert_images.sh you have to specify the folder in which your tub folders with the training images are located. The script copies the *.json files per found tub folder into a new tub folder and saves the manipulated images in these folders according to the *.json files.

If you look at the script then you will see that it is quite simple and it just uses the program "ImageMagick convert" and calls it with different parameters. By manually customizing it like deleting, changing or duplicating the code you can create more training images with properties you want.

The convert_images.sh script is available for download at the following URL.

URL: *https://custom-build-robots.com/jetson-nano-download-de*

A good description of the possible options with which you can call convert can be found at the following URL.

URL: *https://imagemagick.org/script/convert.php*

9.5 Help my fan does not start

In case you have installed a fan on your Jetson Nano and it does not start or rotates too fast and therefore too loud etc. then you can influence the rotation speed of the fan with the following command.

If you execute the following command in the terminal window, the fan will rotate at maximum speed. The number 255 in this command sets the speed and can be between 0 and 255.

Command: sudo sh -c 'echo 255 > /sys/devices/pwm-fan/target_pwm'

With the following command you stop the fan because the rotation speed is set to 0.

Command: sudo sh -c 'echo 0 > /sys/devices/pwm-fan/target_pwm'

If you execute the following command, the fan will rotate slightly and is hardly audible. I have also had good experience with this setting when training neural networks on the Jetson Nano.

Command: sudo sh -c 'echo 130 > /sys/devices/pwm-fan/target_pwm'

In the screenshot below, the speed of 130 that was previously set is highlighted in orange.

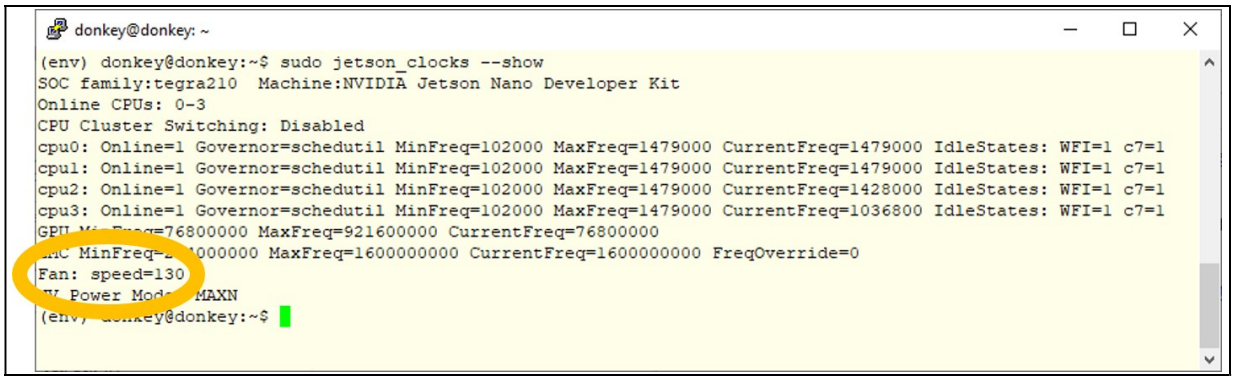


Figure 9.3: Fan speed

9.6 Robot car with WIFI access point

Perhaps you have already been outdoors with the robot car and also had the problem, for example, that no WIFI was available. In these cases, you can start an access point on your own smartphone or take your own router with you. But you can also start your own access point on your robot car and thus establish the connection between laptop and robot car.

How to start an access point with the NetworkManager already present in the operating system on your robot car is explained below. It is actually quite simple and if you terminate the access point in the robot car then the Jetson Nano also dials directly again into an available and known network.

Use the following commands to set up an access point with the name or SSID "robot-car". The password for this access point is "robot_123" and you should adapt this for your robot car. You don't have to pay much attention to the commands one after the other.

Befehl: sudo nmcli con add type wifi ifname wlan0 mode ap con-name Roboter-Auto ssid Roboter-Auto

Command: sudo nmcli con modify robot-auto 802-11-wireless.band bg

Command: sudo nmcli con modify robot-auto 802-11-wireless.channel 1

Command: sudo nmcli con modify robot-auto 802-11-wireless-security.key-mgmt wpa-psk

Command: sudo nmcli con modify robot-auto 802-11-wireless-security.psk robot_123

Command: sudo nmcli con modify robot-auto ipv4.method shared

With the last command of this small list you start the accesspoint on your robot car. Then you have to connect to it. You should see the IP address 10.42.0.1 in the OLED display. You can use this to reach your robot car from your smartphone or laptop, for example.

Command: sudo nmcli con up robot car

If you want to see which access points have already been set up on your Jetson Nano, you can display a small overview with the following command.

Command: nmcli con show

If you want to start the access point manually, use the following command. If the access point has been started once and you restart the Jetson Nano, the access point will also be started again automatically.

Command: sudo nmcli con up robot car

If you have switched off the access point manually and restart the Jetson Nano, the access point will not be started again. The Jetson Nano will then dial into the next known and accessible WLAN. Use the following command to deactivate the access point on your robot car

Command: sudo nmcli con down robot car

If you want to delete the access point you have configured, you can do this with the following command.

Command: sudo nmcli con delete robot car

9.7 What you have achieved so far

Now you have received a few minor tips that will hopefully help you work around a few problems that have been giving me trouble. With the script to generate additional training data from existing ones you should be able to train a much more stable autopilot without having to record additional training data. With the configuration of the access point on the robot car, you are free of a router or WIFI that you would have to bring with you outdoors in e.g. large parking lots.

I very much appreciate your feedback on this chapter. Write me your thoughts, questions and suggestions to the following e-mail address: *ebook@custom-build-robots.com*