Advanced Self Driving Quad

System Overview

Sieuwe Elferink

T41

Delta

Introduction

This document contains the system overview for the advanced self-driving quad project. This includes: an introduction, hardware schematics, mechanical systems, software components, image to road detection pipeline and ai architectures.

This project has been started by me 2 years ago by making a kid's quad (see picture) drive by itself using two infrared sensors for lane detection and an ultrasonic sensor for obstacle detection. While it did work it also had a lot of disadvantages. The main one being that because infrared based sensors where used the quad would not work outside since the infrared light from the sun makes the sensors unusable. Also, the system used white lane markings on the side to detect a road so if a road did not have white lane markings the system would not work.

Because of all these disadvantages I decided to give it another shot with some more advanced technology (hence the name "advanced" real time self-driving). For this new project I wrote down some requirements.

1- The quad can autonomously drive over a single lane road without the need of lane markings on the side.
2- The quad uses a Camera for road detection.
3- The road detection and control systems must be robust and work in all situations encountered at day. The quad does not have to work in the night.
4- The quad is completely independent, and all systems needed are on the vehicle.
5- The quad needs to be able to drive autonomous at a high speed.
6- The quad needs to be able to stop when a object is in it path. The quad also needs to adapt its speed when there is a slower moving object in its path.

While researching possible solutions for road detection and control I came across a couple of solutions. These could be categorized in vision-based control and Ai based control. While both have been tried, as can be seen on GitHub, This document will go over the current system being used on the quad which is Ai based.

Hardware system

The hardware of the quad can be categorized in a couple of separate sections. Each section is listed below with an explanation.

Quad base

The quad base is the original quad chassis with all original electrical components. This quad has been bought second hand from marktplaats.nl. The quad has the following specifications:

- 36 Volt 28A electric power train
- New 3x 12 Volt, 12Ah battery pack.
- 36-volt mosfet speed controller which accepts 5v PWM
- 8-inch wheels
- Full steel frame with independent suspension.
- Arm based steering (Not with rack and pinion)
- Top speed of 45km/h

The complete driving quad is used as a base. All systems listed below are integrated into the quad. For example, the original speed controller is used and only the source of the PWM signal has been changed from the throttle lever to a output pin of the embedded computer.

Road and object detection

For road detection a Xbox Kinect v1 is used. This sensor is chosen because it has both a 640p RGB camera and a IR matrix depth camera with a 12m range in a very affordable package. The sensor uses a usb cable to connect with the Main computer. The sensor is mounted to metal bracket using zip ties and in between the two parts is a small sponge to help with vibrations. The RGB and depth sensors do not have OIS and thus the sponge is needed to give a steady frame to the image pipeline discussed later.

Steering system

The steering system consists of two sprockets connected by a chain. The largest sprocket is mounted on top of the steering column. It is important that both the center of the sprocket is in the center of the steering column and that the sprocket angle is exactly perpendicular to the steering column. This is so that the sprocket wont change from its angle once the quad steers to a direction which makes the chain needing to be longer and then shorter again which is physically impossible.

The smaller sprocket is mounted on top of a windshield wiper motor from a car. This motor is chosen because it has a worm gearing inside which gives it a lot of torque. Also because it is a 12V DC motor it is easy to use since the battery pack already has 12V onboard.

The steering system also has a high quality 10K potentiometer mounted in line and in the center of the steering column. This potentiometer acts as a steering angle sensor which the software can use to check what the current angle of the steering system is.

Throttle system

The throttle system uses the original speed controller of the quad. This mosfet powered speed controller has a PWM input which ranges from 0 to 5v. This input is connected to a PWM output of the arduino based embedded computer. The arduino can then generate a PWM signal to control the speed controller which then controls the drive train.

Embedded computer

The autonomous quad has 2 computers. The first is the main computer which detects the road and turns that into throttle and steering commands. The second computer is a embedded computer which is a Arduino Nano. This setup is chosen because the main computer has no direct hardware outputs. It does have USB which the Arduino Nano also has. This makes the Arduino nano the translator between the hardware of the quad and software on the Main computer. The Arduino Nano is chosen because of its small size while still having all the necessary outputs and it is easy to use.

Training sensors

To train the AI the quad needs to have data from two sensors on the vehicle. These are the steering angle sensor explained in the section about the steering system and a throttle sensor. The throttle sensor used is the original throttle lever from the quad. In training mode, the sensor must be able to both drive the quad like originally intended but also needs to read its PWM value on an analog pin on the arduino. But in autonomous mode the arduino needs to be able to send certain PWM values to simulate the throttle lever being used on the same wire. So, the wire from the throttle lever has to be a input in one mode and a output in another mode. To make sure the that the arduino is not shorted a manual switch has been added which switches between an analog input pin or a PWM output pin according to what the user has selected.

Safety systems

The safety of an autonomous vehicle is very important. It is necessary to completely stop the vehicle if something goes wrong. The quad can do this using a couple of circuits. The quad has 2 main power buttons which cut off the power to all systems if one of the buttons is pressed. The quad also has a separate button for only the steering system and a separate button for the entire 220v system.

Main computer

To process everything a very powerful pc is needed. A embedded computer like a jetson nano is very nice but is just not powerful enough for this application. Because of that a full Desktop PC has been chosen. This PC has a GTX1080 graphics card for Ai processing and a I7 processor. The PC also has a SSD installed because of its better speed and also because of the vibrations from the power delivery system. These vibrations can give read and write errors on a spinning hard disk.

Power delivery

To supply the Desktop PC with its needed 220v power a powerful power delivery system needed to be made. One possibility is to use the 36 V available and transform it to 220V. This is possible for small loads but the PC can peak at over 600 Watt's which is to much for the 12Ah batteries to handle. They would go empty within 3 minutes and the batteries won't last a lot of charge cycles. Instead the decision was made to use a gas-powered generator. These can easily handle 600 watts and more. The only downside is that they have a lot of vibrations and are loud. A better solution for the power delivery would be nice but no better solution for a affordable price has been found.

Wireless connection

For debugging and controlling the software on the desktop pc it is necessary to have a wireless connection. This is done by using a wireless router mounted on the quad. The router is connected to the PC with a ethernet cable and the laptop of the user is connected with Wi-Fi to the router. The laptop can login to the Desktop computer using the x2go screen sharing protocol which works using SSH. X2go is much better in speed and reliability compared to VNC.

Wiring schematic

Needs to be made!!

Software system

The software for the autonomous quad has 3 main components constantly running besides each other in separate threads. Each component is explained below. Also, the quad must learn to drive which is done in a learning mode. This learning/training mode is also explained below.

Road Detection and control

One of the most important parts is the road detection and control thread. This thread gets the camera image, feeds this trough the segmentation model for road detection, then feeds the segmented image through an end to end drive model to generate steering and throttle commands and then sends these values to the control thread. Each step is explained in more detail below

Camera Image

The first step is to get the camera image from the Xbox Kinect. This is done by using the LibFreenect drivers and python wrapper. A simple get_video() function is called to get the image in the widely used OpenCV format. The image goes then through a Gamma color filter to make the image a little bit brighter. This helps the road detection AI. After the Gamma filter, the image is resized to a size of 512 by 256. This is because the input of the segmentation model is this size.

Segmentation model

The resized image is feeded into a Nvidia developed segmentation model. This model uses a Convolutional Neural Network (CNN) based a Deeplabv3-Plus and WideResNet38 architecture. This model has been trained by Nvidia on the city scapes dataset which has a lot of roads in the dataset. This makes the network detect roads and also other classes of objects very well. The model also uses a new technique to speed up segmentation between frames by combining frames for better real time performance which is needed for a autonomous vehicle. During testing the model can reach around 15fps with 99% GPU utilization.

The segmentation model is used to highlight certain features of the entire scene which includes the road. This is done so that the drive model can better learn which features are important and which features are not. If the normal camera image is feeded into the network, it will have a difficult time knowing which features are important and which are not. This is because the model does not know that the road is important and thus maybe starts to focus on the surrounding area like the sky. It for example can start to learn that a certain pattern in the sky corresponds to a certain steering angle. This is of course not correct and thus to help the model a segmentation of the scene is done. After the segmentation, the output is filtered where important classes are highlighted, and not important classes are removed from the image. How this exactly works is explained in the next section.

While the road detection accuracy is good it can be improved. The city scapes dataset only has roads in urban areas while the quad is being tested on roads in a agricultural area. By making a new dataset which also has these types of roads labeled, the accuracy of the detection model will increase. Also there are some faster models available but these do not have a pretrained models available online. To save time and because the city scapes data set is not publicly available, the decision was made to use a pretrained model. But in a future experiment it is worth looking at making a custom dataset.

Output filtering

The Segmentation model gives a NumPy array with the width and size of the input image. The only difference is that instead of having each element represent a RGB value for that pixel, every element represents the class for that pixel. The class is a number between 0 and 30. These classes are then colored in using predefined colors which gives a nice image like the image below.

This is very nice but the entire goal of the segmentation model is to filter out all not necessary classes and highlight the important ones. Because a convolutional layer just gets a RGB value per pixel it is easy to highlight a certain class by increasing its RGB value. This is the reason why the road is white. White in RGB is (2555,255,255) which is the highest possible values. This should activate the neurons at these pixels the best and thus making the model learn this feature very well. The other two important classes are vegetation and ground. This is because threes on the side of the road can show the width of the road and ground is highlighted because this is what is on the side of the road. These both get a value in which one RGB value is 255 and the other two are 0. This gives a very clear contrast between the two classes to the model and thus it can better learn edges. This is very important because the model needs to learn to detect road edges so that it knows where to drive and where not to drive. All the other classes which are not important are made black (0,0,0). This is because these pixels will not activate neurons and thus the model wont try to learn features from these classes.

The example image below has a very nice detection but this is not always the case. Sometimes there are holes in the road or some or there are some other problems with the result. To always give a smooth image to the drive model the output image of the segmentation model is smoothed with a morphology open technique. This uses a kernel of (15,15) to make the image smaller and then larger again to smooth out rough spots. This does mean that some detail is lost but this does not matter since the drive model does not need per pixel accuracy.

Before

After

Please not that the above images are not from the same image but should give a good enough representation on what the output filter is doing.

Drive model

The autonomous quad has two separate Ai models. The first segments the image and the second one which is the Drive model, predicts the steering and throttle commands. The drive model is a CNN based on the end-to-end driving paper from Nvidia.

https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

This model uses different convolutional and fully connected layers to learn patterns between an input image and a given set of throttle value and steering angle as output. Once trained the model can give with only an image as input a corresponding throttle and steering value. It can do this because it can learn complex features like road edges like shown in the image below.

Add image of features in layers

The drive model does require an extensive and well balanced training method to work at its best. This training method is explained later.

Quad control

The control thread gets the steering and throttle commando from the drive Ai. This is a value between -1 and 1 for steering and 0 to 1 for throttle. Every loop the quad controller reads out the current throttle and steering angle using the arduino. (The Arduino runs the standard firmata code which makes it possible to control the outputs and inputs of a Arduino over USB in python using the standard fermata protocol.). The controller then compares the current readed steering and throttle values with the latest desired values which come from the drive model. The error between the current and desired values is calculated and then fed into a PID controller which is tuned to minimize this error. The PID controller calculates the needed steering and throttle command which is then send to the arduino to perform.

This controller thread runs much faster at around 100hz while the main thread only runs at 15hz. This makes the controller able to better control the steering and throttle.

Obstacle Detection

For obstacle detection the Depth camera inside the Xbox Kinect v1 is used. This camera works by projecting a matrix of infrared dots. The Kinect knows the distance between each dot in the matrix. If a obstacle is in front of the camera, the distance between some dots changes. The Kinect can use this difference to calculate the depth.

The obstacle detection system works by getting the depth map of the Kinect at around 30fps. The depth map is a NumPy array where each element is a value between 0 and 2048. A obstacle is detected by getting the 100000 lowest values and checking weather the average of these lowest values is higher than a certain threshold. If this is the case the average is taken and mapped to a range between 0 and 1 which is the suggested throttle command. This suggested throttle command is compared to the throttle command of the Ai. The lowest throttle command is then chosen and send to the control system. The value of 100000 can be tweaked so that it can better detect smaller or larger objects. Also the depth map has to be cropped in such a way that the road is not also detected as a object.

One thing to consider is that the xbox kinect v1 uses IR for its detection. This may cause interference in bright sunlight. So a different solution may have to be found if the xbox kinect is unreliable in bright sunlight.

Visual overview

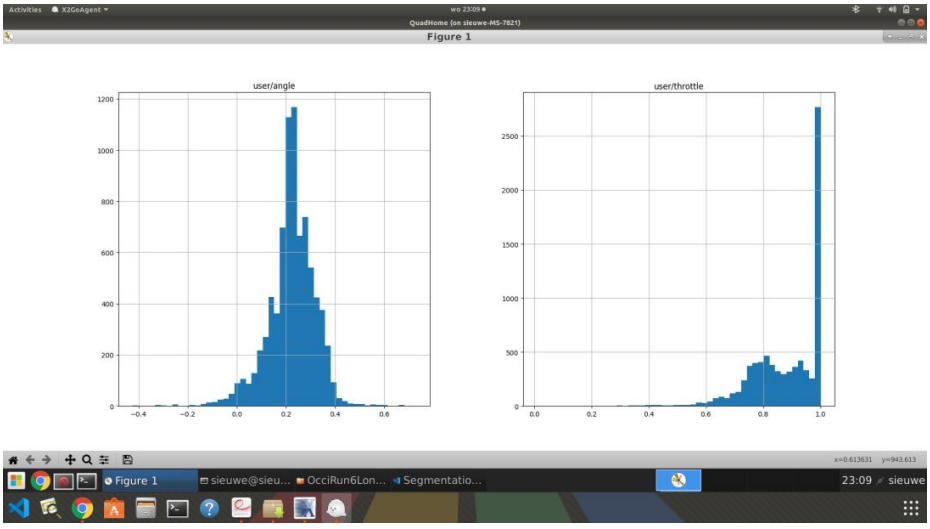
Overview needs to be made !

Training

To make the drive model predict the correct steering and throttle value it has to be trained. This is done by using a technique called behavior cloning. To get started I drive the quad manually over the road. For every image frame I save image, throttle and steering value. After a couple of hours of manually driving a big dataset is made which shows how to drive the quad. Now the end to end drive model is trained to predict the values corresponding to the input image. The Ai has learned to drive the quad by learning how a human has driven the quad and thus the behavior of the human is cloned by the Ai model.

To make the Ai model be able to correctly learn the human behavior it is important that the data is from good quality and most importantly well balanced. This means that the Ai has both a lot of data on steering left as on steering right as on driving straight. To make sure both steering left as steering right is represented well a couple of training (next to centerline runs) runs have to be done with oscillations. This means that the user must drive the quad from the center to the side of the road and then to the center again for both left and right sides. This will also help with edge detection because the drive model will learn where the edges of the road are because the user is going to the edge of the road and then back again. The oscillations also help with learning how to correct if the quad is going to the side of the road. If oscillations where not done and the quad was only driven on the centerline of the road, the Ai never learned to correct in that situation causing it to fail. It is important to only do a couple of oscillation runs because otherwise the quad will also oscillate when driving autonomous.

A image of a well-balanced training set can be seen below. This shows a training set of an almost straight road. This can be seen because steering straight is clearly done the most in the training run and there is a nice balance between steering left and right.



Calibration

A couple of calibration tools are available for calibration. These are:

A tool for calibrating steering and throttle position. This is very important because every vehicle is different. The throttle range or steering range may be different, so a user must be able to calibrate this. This can be done using the actuator calibration test script which asks the user to move the steering wheel and throttle. The program than saves the minimum and maximum values which are used for control.

A tool for calibrating the PID values by making the steering system and throttle go to min and max values. The user can then change the PID values so that the response is as intended.

Implementation

An implementation of the above-mentioned systems can be found online since the entire project is OpenSource. Follow the link below for instructions to run the system yourself and to contribute to the project by sharing your results.

https://github.com/sieuwe1/Advanced-Real-Time-Self-Driving