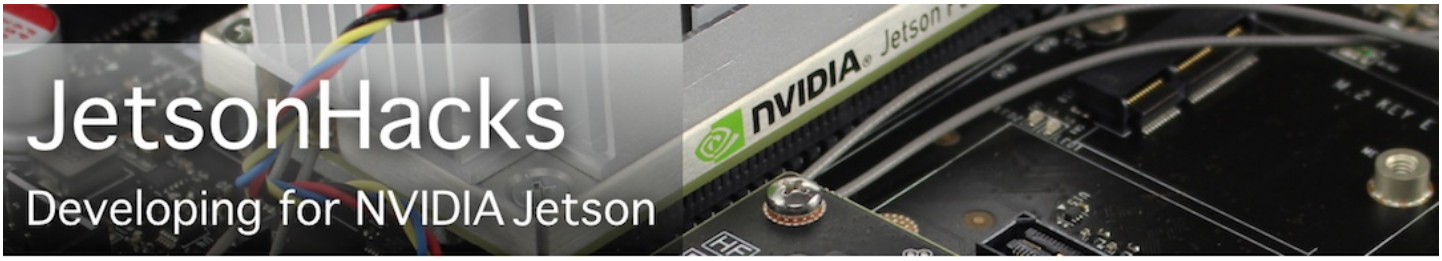

HOME CATEGORIES ARTICLES RESOURCES ABOUT CONTACT US

SITEMAP

SEARCH …

HOME › ROBOTICS › JETSON RACECAR › Daniel Tobias' CAR – Cherry Autonomous Racecar

Daniel Tobias' CAR – Cherry Autonomous Racecar

March 2, 2017 kangalow Jetson RACECAR 0

Daniel Tobias created a Jetson based autonomous vehicle named the Cherry Autonomous Racecar (**CAR**) for his senior design project at North Carolina A&T State University. Daniel was gracious enough to share the in-depth details of his project in this interview.

Daniel Tobias with the Cherry Autonomous Racecar

NCAT Cherry Autonomous Racecar – Tear down

Interview

JetsonHacks (JH): What is your background?

Daniel Tobias (Daniel): I am from Greensboro, North Carolina. My first exposure to robotics was back in high-school where I was the founding president of the school's FIRST based robotics club. After graduation I originally pursued a BS in chemistry but later switched to computer engineering.

Currently I am a senior in the Computer Engineering Undergraduate program at North Carolina A&T State University, where I also help to oversee and run the day-to-day operations of the school's Robotics Club/Makerspace. My current interests include the application of machine learning to robotics and music.

JH: Why did you choose to make the "Cherry Autonomous Racecar"?

Daniel: The idea came to me in March 2016 while I was walking through Cherry Hall on the way to the University's robotics club. The hallways of Cherry have parallel blue and gold stripes, that look like lane markers. So I thought it would be cool to see if a scale car could drive down them like a road. With a bit of Googling and YouTubing I found the MIT RACECAR. Their car was able to navigate the tunnels of MIT using a LIDAR and SLAM. I liked their project but the Hokuyo LIDAR was way out of my budget at $1800.

After more research I figured a vision based approach could get similar results without spending a lot. So the plan was to recreate their platform but use OpenCV with GPU support to do lane recognition, and eventually bring other sensors into the fold to help it navigate. That was how my project started out and where the name Cherry Autonomous Racecar comes from.

While researching techniques for lane centering, I became more interested in how current approaches to self driving cars work. So I figured I would try reproducing them at scale. I also wanted to get experience with sensors similar to those being used on autonomous cars in the industry; which is why I chose to add the Neato LIDAR and the RealSense depth camera.

JH: Why did you choose to use the Jetson for the CAR. What advantage does using the Jetson platform give the CAR?

Daniel: I originally went with the Jetson TX1 because I wanted to do real time lane recognition and centering by using OpenCV. Since the board was advertised as world's most advanced system for embedded visual computing I figured it would be a perfect fit.

My original plan was to offload image processing to the CUDA cores and onboard HW decoders to handle high res frames in real time. But the main benefit was that everything could be done locally without relying on a wireless connection to a server.

There was also online resources and active communities, like JetsonHacks, centered around the board which were helpful.

Currently I am using TensorFlow on the Jetson TX1 to run the End to End Learning for Self-Driving Cars CNN model that NVIDIA released a while back. The Jetson allowed me to use the GPU version of Tensorflow to run the trained model roughly in real time.

JH: What were the major obstacles you encountered when building your CAR?

Daniel: The biggest obstacle I faced was compiling OpenCV 3.1 with CUDA support. I spent a lot of Summer 2016 in dependency hell, involving over 50 attempts to get OpenCV compiled right. You can see that, around that time, there were multiple posts on the official Jetson TX1 forums regarding how to install it.

It took some time to diagnose why my data recording node was saving images out of order. Eventually I realized that the Jetson couldn't handle saving 65 FPS (30RGB,30Depth,5LIDAR) with the software encoding of cv.imwrite. So I tried reducing this number to (10,10,5) respectively and it worked; I suspect it was a threading issue.

In the paper NVIDIA implemented the CNN using Torch and their Drive PX system. Reproducing the same model in Tensorflow resulted in a trained model that was over 1Gb in size. This was problematic since the supporting ROS network and OS were already using 2.5/4 Gb of RAM. When we tried loading the model into RAM the Jetson threw OOM errors. We tried different solutions like black-and-white images and reducing the resolution of the images further. These helped bring the size of the model down, but the main issue was the 1164 Neurons on the first fully connected layer. By cutting it down to 512 neurons we were able to reduce the model to 150Mbs without any noticeable loss of performance in our validation set. There are other improvements that could reduce the size of the model, like using half-precision floating point which the Maxwell based CUDA cores could take advantage of. Or using TensorFlow quantize functions to reduce the precision to 8bit.

I was originally commanding the Traxxas Electronic Speed Control (ESC) using a 5 volt Arduino, but getting strange behavior. Turned out the ESC uses 3.3V logic, so switching to the 3.3V Teensy solved the issue.

IMUs are very noisey and calibrating the magnetometer of the razor 9dof IMU is near impossible when mounted on my car.

Had to upgrade the shocks and springs of the Traxxas Slash to support the weight of all the extra components.

Xbox controller transceiver module caused interference with the wifi I was using to remote access the car. Both apparently use 2.4GHz band. One could use the Jetson's 5GHz band to get around this, but my current router doesn't support it.

Another obstacle was the lack of storage space. In July I found a guide on how to make the filesystem on the SSD and point the Jetson to look at the SSD when it boots up. This also allowed compiling TensorFlow from source which requires at least 12Gbs of swap space. But now you can just use a .whl file to install Tensorflow. *(Install TensorFlow on NVIDIA Jetson TX1 Development Kit)*

I started to realize pretty early on that development would be difficult on a bleeding edge device like the Jetson TX1. But in its current state the support and community for the Jetson TX1 has grown exponentially where most of the problems and their solutions can be found, if you search for them.

JH: It'd be great to have a description about the idea behind doing deep learning/training for the CAR. I know that it's a big subject over at NVIDIA.

Daniel: I eventually stumbled upon the the video of NVIDIA's DAVE 2 system driving a Lincoln near the end of September. Looking at the paper that accompanied the video, I saw they were using the NVIDIA Drive PX to run their CNN, so I googled it. It seemed the Drive PX and the Jetson were similar in hardware, which led me to believe that it was possible to implement their net on the Jetson.

An end to end system that could handle input to actuation went completely against traditional approaches to autonomous driving. There were no explicit equations, variables or software modules that were designed for a specific task. It seemed like a new paradigm to use a purely data driven approach that could mimic the behavior of a human driver.

The training data was generated by manually driving the car around and saving every third image to disk as a jpg with the corresponding steering value in the filename. Each steering angle was normalized by mapping to a range of values where 0.0 represents full-left, and 1.0 full-right.

Each data-collection run lasted about 35 minutes producing over 30,000 labeled images. These were then offloaded to a computer with a GTX 1080 and sufficient RAM for training. The resulting trained model was copied to the Jetson and fed realtime images from the same camera; with the steering angles output by the model used to direct the car. Since the model only produced steering angles, a simple cruise control feature was added to set the throttle so the controller didn't have to be used to accelerate.

In the paper they augmented the images so the car would learn to recover from mistakes. Our take on this was to drive the CAR normally for 80% of the time and for last 20% to drive it somewhat chaotically. Chaotic driving was letting the car drift periodically on the straightaways and correcting when it got close to a wall. Also, additional human drivers were used so that the car had a variety of data. We noticed driving patterns emerge when the car ran the model that could be linked back to a specific driver. As in the paper, we tried to increase the relative number of turning images for training. This was done by flooring the throttle of the CAR when it was going straight and taking the turns very slowly to generate more images.

JH: What are your plans going forward with the CAR?

Daniel: Currently my team and I are creating hardware instructions on how to physically recreate the CAR, which go with the software already posted on Github. The Jetson and most of the sensors are not being utilized to the fullest, so we plan to release the project as an Open Source/Hardware platform so that others can modify or branch the project in a different direction like what I did with the MIT RACECAR.

As a senior design project, we are also working on a larger version of the Cherry Autonomous Racecar. Think Barbie Jeep but with a few thousand dollars worth of sensors thanks to Chrysler.

If time permits this semester I plan to try a 3D convolutional neural network to work with the point cloud from the RealSense. The net would try to detect other 1/10th scale cars and maybe some other smaller objects like a toy dog. I might also try a 3D CNN through time(FlowNet) and maybe a RNN approach.

I might also try running the CAR in a walking/bike trail setting and see if it can train on the first few miles and run on the last few miles assuming I can find a trail that doesn't change too much scenery wise.

On a more personal note, I wouldn't mind continuing development of the CAR project or a similar project in the capacity of a graduate student or researcher.

NCAT Cherry Autonomous Racecar – Trial Runs

Github Repositories

Tensorflow

https://github.com/DJTobias/Cherry-Autonomous-Racecar/tree/master/Tensorflow
https://github.com/DJTobias/Cherry-Autonomous-Racecar/blob/master/car/scripts/runModel.py

CAR package

https://github.com/DJTobias/Cherry-Autonomous-Racecar/

Conclusion

I want to thank Daniel for sharing his work with the JetsonHacks community, hopefully this will provide some inspiration for other Makers out there. If you have a Jetson based build you would like to share, send in an email!

Share this:

Email Print Reddit Facebook Twitter LinkedIn Tumblr More

Like this:

★ Like

Be the first to like this.

Related

Jetson RACECAR Build - Part 1 - Traxxas Rally Unboxing
January 18, 2016
In "Jetson RACECAR"

MIT RACECAR Walkthrough - NVIDIA Jetson TK1
October 6, 2015
In "ROS"

Jetson RACECAR 10 - Motor Control and Battery Discussion
July 5, 2016
In "Jetson RACECAR"

« PREVIOUS
Speech Recognition – Smart Microphone – Jetson Development Kits

NEXT »
NVIDIA Jetson TX2 Development Kit

BE THE FIRST TO COMMENT

Leave a Reply

Your email address will not be published.

Comment

Name *

Email *

Website

POST COMMENT

□ Notify me of follow-up comments by email.

□ Notify me of new posts by email.