Autocar

Patrick Huang

August 2024

1 Introduction

We design and build an intelligent self driving robot car.

This paper describes the technical methods we used.

2 Mechanical

2.1 Frame

We use v-slot aluminum rails and MDF wood to form a 300 by 250mm base. We attach the motors to the aluminum rails via 3D printed mounts. We attach electronics onto the MDF. See Figure 1.

2.2 Motors and drivetrain

We use 370 sized brushed DC motors with a worm gearbox attached. We attach 125mm diameter wheels via 3D printed mounts. The motors are rated for 120RPM at 12V, which translates to 0.78m/s or 1.77mph.

Steering is achieved by powering the motors on one side more than the other (tank steer). For sharp and in-place turns, this causes skidding, as the wheels are moving perpendicularly across the ground. For shallow turns, it is able to maintain constant traction. Regardless, it is a reliable and simple steering system.

The four motors are controlled by only two motor drivers. The two motors on each side are shorted together and controlled from a single driver.

3 Electrical

Figure 2 labels the electrical components we use. They are as follows:

1. DepthAI OAK-D camera [5].
2. 5V USB-C power supply.
3. Raspberry Pi 5.

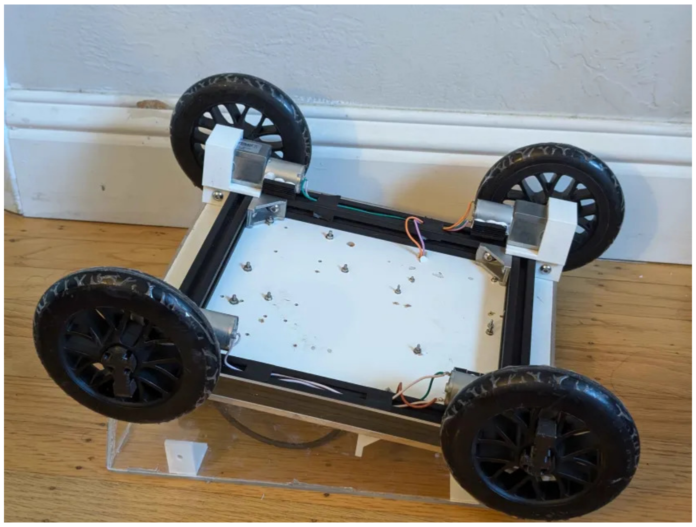
Figure 1: Underside of the car, showing the frame

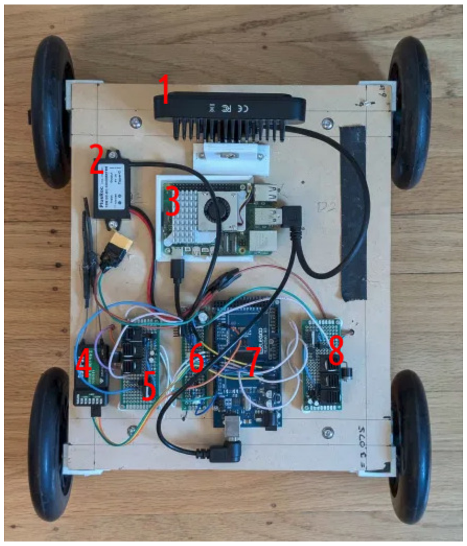

Figure 2: Enumeration of electrical components.

4. RC receiver.
5. Motor driver 1.
6. Power distribution board.
7. Arduino Mega.
8. Motor driver 2.

3.1 Power

We power the whole robot from a 4S lithium-ion polymer (lipo) battery. The power is distributed in a few ways, through multiple voltage regulators.

The motors receive the full battery voltage, switched by the motor drivers. The Arduino and Raspberry Pi are powered from 5V regulators; the Arduino has one built-in. The motor drivers require an additional 5V regulator for power to switch the relay.

The motor driver 5V regulator we use (L7805CV) generates much heat during normal operating conditions. We therefore add a cooling fan directed toward the regulator. In the next iteration, we would choose a more efficient voltage regulator.

3.2 Motor drivers

We design and build a H-Bridge and MOSFET motor driver, as show in Figure 3. This allows power modulation and polarity (motor direction) control. We use relays to switch the polarity and power, and a high current MOSFET to do PWM.

The first relay enables and disables the driver by switching power on and off. We add this for safety, in the event that the MOSFET is unable to cut power due to a failure.

The next two relays switch simultaneously, changing the polarity across the motor.

To each relay coil, we add flyback diodes to absorb the inductive spike; and a capacitor to filter high frequency interference and produce reliable switching.

We use a large 20A diode as the flyback for the motor.

We use a configuration of BJTs to drive the MOSFET gate [7]. The BJTs provide two purposes: 1) to charge the gate to 12V, turning the MOSFET fully on, from only a 5V signal; and 2) to rapidly charge and discharge the gate with high current to minimize power loss over the MOSFET.

4 Software

4.1 Firmware

The firmware runs on the Arduino and directly controls the motors and reads from the RC receiver.

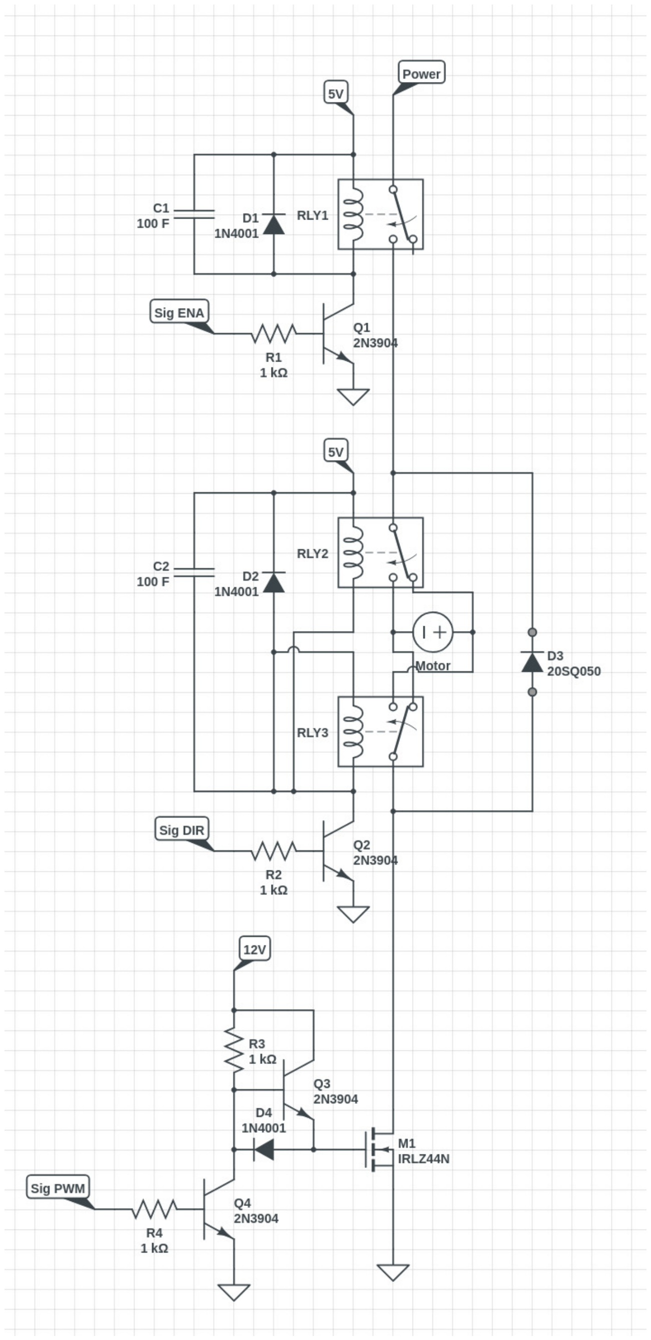

Figure 3: Motor driver schematic

We use built-in hardware PWM to adjust the motor power.

We use a community library, IBusBM [6], to interface with the receiver's protocol.

The Arduino constantly communicates, at 100Hz, with the Raspberry Pi via the serial port. The Arduino follows commands given by the Raspberry Pi: The Raspberry Pi sends requested motor speeds (i.e. PWM duty cycles), which the Arduino executes. The Arduino, in return, sends the received RC values.

4.2 High level software

Steering control: The software represents control as a speed-steer control system. Two continuous values — speed (0 to 1) and steer (-1 to 1) — are mapped to motor speeds.

We use a PID loop to monitor the turning rate. We interpret the steering input as a target angular velocity. We use the OAK-D camera's built-in IMU to measure the car's orientation and feed it into the control loop. We find that a PID loop is necessary in order to get reliable turning rates.

Neural network: The neural network outputs the aforementioned steering control — a single continuous value. We use a Resnet 18 [2] with a modified head. The input is the color and depth images from the OAK-D camera, concatenated. We use $tanh$ with a low temperature of 0.1 to allow finer adjustments: $y = tanh(0.1 * x)$. We find that the low temperature significantly improves performance.

Training the NN: We generate data by manually driving the car with the same speed-steer system. At periodic intervals, we save the camera's view, as well as the manual steering input that was provided. We find that training on such a dataset allows the car to both learn general driving patterns like obstacle avoidance, and also be biased to follow the specific paths taken while generating data.

Figures 4 and 5 show the RGB and depth data, including augmentations.

We apply 3 crucial "3D transform augmentations" during training, using ideas from [1]. As the data only includes "good" driving, we must purposely create some bad examples along with the correct steering adjustment to produce reliable driving.

- Z Rotation: We crop the left or right section of the image to simulate rotation along the Z axis in 3D. The steering label is adjusted in the opposite direction.
- Y Translation: We zoom in on the image to simulate moving forward in 3D space. We increase the magnitude of the label.
- Obstacle: We increase the values of the depth map on one side to simulate an obstacle. The label is adjusted in the opposite direction.

We use MSE loss and the Adam optimizer. We train for 50k steps across 100 epochs. The final validation loss is around 0.02, or an average of $\sqrt{0.02} = .141$ out of (-1, 1) steering error. See Figure 6 and 7.

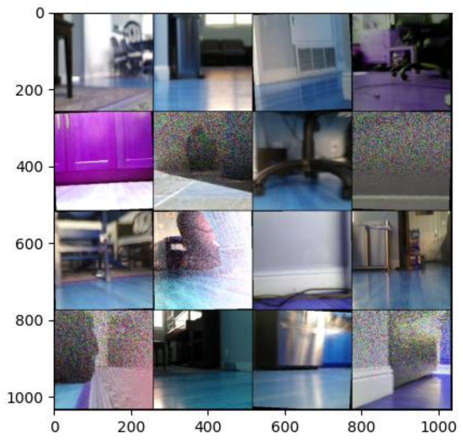

Figure 4: RGB pass

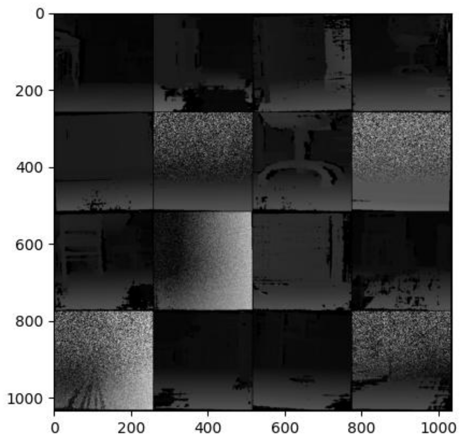

Figure 5: Depth pass

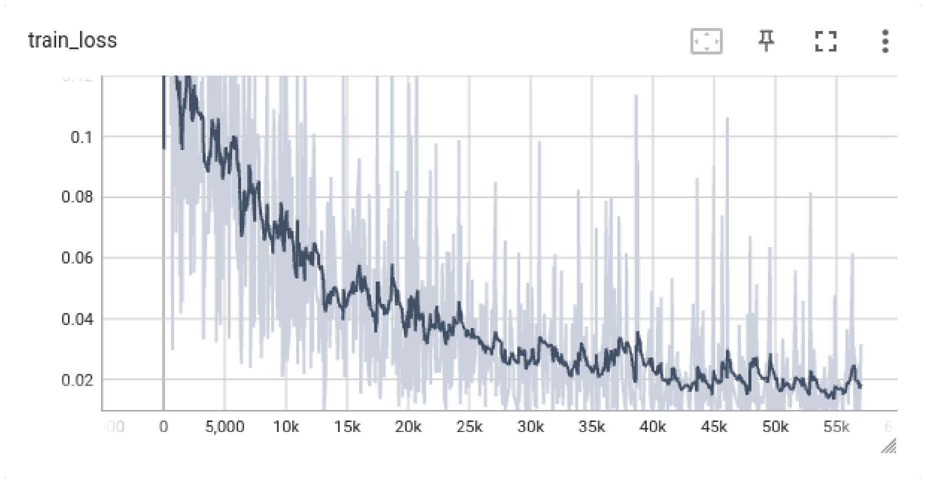

Figure 6: Train loss

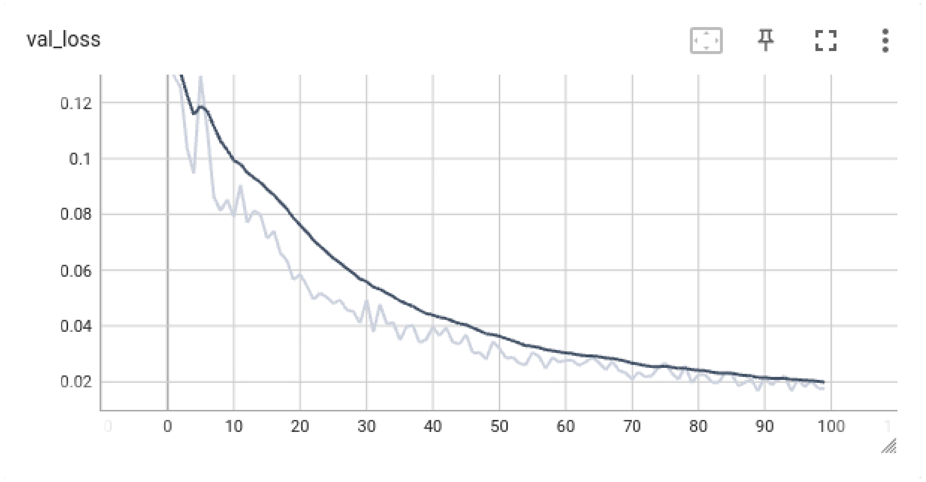

Figure 7: Validation loss

5 Next steps

5.1 3D mapping

3D mapping, as opposed to operating on a single camera view, is a natural choice for navigation.

3D mapping also allows more robust data augmentation, replacing the current "3D transform augmentations". Given the images from a single training route, we can build a 3D model of the environment and render views from off-center or askew positions for augmentation.

We propose using Mast3r [4] and 3D Gaussian Splatting (3DGS) [3] for 3D reconstruction. Mast3r solves camera poses and creates an initial dense reconstruction extremely quickly. 3DGS fine tunes and renders the point cloud to produce photorealistic views.

3D mapping provides long term memory to navigation. After scanning an environment, the robot can navigate to a specified destination.

6 References

References

[1] Bojarski et al. "End to End Learning for Self-Driving Cars". In: (2016).

[2] He et al. "Deep Residual Learning for Image Recognition". In: (2015).

[3] Kerbl et al. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: (2023).

[4] Leroy et al. "Grounding Image Matching in 3D with MASt3R". In: (2024).

[5] Luxonis. *OAK-D*. URL: `https://shop.luxonis.com/products/oak-d`.

[6] Bart Mellink. *IBusBM*. URL: `https://github.com/bmellink/IBusBM`.

[7] Neil UK. *Drive a MOSFET via BJT*. URL: `https://electronics.stackexchange.com/a/264153/353163`.