SCORAV: Single-Camera Off-Road Autonomous Vehicle

Conservation, Machine Learning

Luke Rand, Isaac Olson

Final Report

New Mexico Supercomputing Challenge 2025-2026

Santa Fe Preparatory School

Santa Fe, NM, US

April 1 2026

Contents

1 Introduction

1.1 Executive Summary

The purpose of SCORAV (Single-Camera Off-Road Autonomous Vehicle) is to develop a low cost autonimous rover to assist in environmental monitoring. In manual driving mode, SCORAV collects image and servo data to train a machine learning model capable of autonomous navigation. Our system provides a scalable framework for off road navigation without relying on expensive sensors, making it possible for environmental organizations to include SCORAV data collection as part of their process.

In addition to autonomous navigation, SCORAV includes a proof of concept litter detection application that demonstrates the significant environmental preservation potential that is included in autonomous ground based data acquisition.

While prototype, hardware and dataset limitations prevented SCORAV from achieving autonomous navigation in the field, the project successfully validated key concepts and methodology in vehicle control, sensor integration, and machine learning model training. The current SCORAV prototype provides a strong foundation for future off road vehicle development and expanded environmental applications.

1.2 Problem Statement

The Forest Service struggles to adequately maintain trail, with a 22% decrease in maintained trail this past year and a $8.6 Billion backlog of necessary maintenance.[6][12] Similarly, litter and invasive species become ever-increasing problems on public trail, harming the pristine environment the outdoors can be.

[7][2] With these problems at hand, the benefit of automation becomes evident.

Our project aims to create a low-cost autonomous rover, by building a simple framework requiring only one camera sensor. The rover would have the ability to connect with auxiliary apps for litter detection, trail quality survey, and maintenance needs understanding. Doing so could aid organizations like the US Forest Service in performing their duties and ensuring the wilderness remains available to play and learn in.

1.3 Tools Used

Programming on board the vehicle was done with python in the Raspberry Pi and a C++ variant on the Arduino. Libraries were used to streamline the process including standard Python libraries, OpenCV, Math, PyTorch, Numpy, and MatPlotLib. Multiple web-downloadable datasets, discussed later in 6.1.1 were used to train auxiliary models.

The vehicle itself used an Arduino, Raspberry Pi, motor, H-bridge, USB battery pack, USB Camera, remote control and Receiver, and various structural elements and wiring.

1.4 Current State of the Field

Existing off-road autonomous vehicles often employ more complicated algorithms and perform better than our model would, even if completed to the full extent of our vision. However, they utilize multiple expensive sensors and machinery that drive prices. For example, the vehicle built by the AirLab at Carnegie Mellon University is self-supervised and performs at high speeds on variable terrain.[1] However, it aggregates LIDAR, IMU, and camera input.

Similarly, the military-focused vehicles at Overland AI utilize sensor fusion to provide reliable military off-road solutions.[9]

Figure 1: AirLab's vehicle.

While our approach would inevitably be less robust and encounter more issues at corner cases, it would be much less expensive to build and simpler due to requiring only a single camera. Thus, our solution, while less robust, would be easier for large fleets at a low budget, such as in the solutions discussed in 7.3 SCORAV Applications.

2 Methodology

Because this report represents a project that has not yet reached anticipated progress, italicized bullets represent future plans. Note that training a litter detection model was completed, despite being non-sequential.

- An electric rover suitable for concrete navigation was built and programmed to respond to remote control. A camera was affixed to the front.
- The rover was programmed to collect and correlate camera input and servo angle at one second intervals and save them to an aggregated data set on local storage.

- The rover was driven at a consistent speed (see 3.2.2) on concrete sidewalk in parks, collecting data.
- Most of the dataset was used to train a supervised machine learning model to derive servo angle from image information.
- A segment of the dataset not used for training was used to verify the model.
- *The rover is reprogrammed to determine steering from the outputs of the machine learning model, rather than remote control input.*
- *A more robust rover is constructed with off-road capabilities, and similarly trained and tested on local trails, most likely with clear distinctions of trail edge such as grass*
- A litter detection model was trained and tested from an aggregate dataset found online.
- *The litter detection model is used to determine hot spots of trail litter based on image and GPS data collected by the robust rover*

3 SCORAV: The Vehicle

3.1 Raspberry Pi

The main computing is done with an onboard Raspberry Pi, connected to all the other core electronics. The primary python script on board the Raspberry Pi runs continuously, taking an image from the servo and the servo angle at 1 second intervals. Both of these values are collected using communication with external devices. The Pi is configured using a systemd service to run the python script immediately on startup. The Raspberry Pi is loaded with a headless

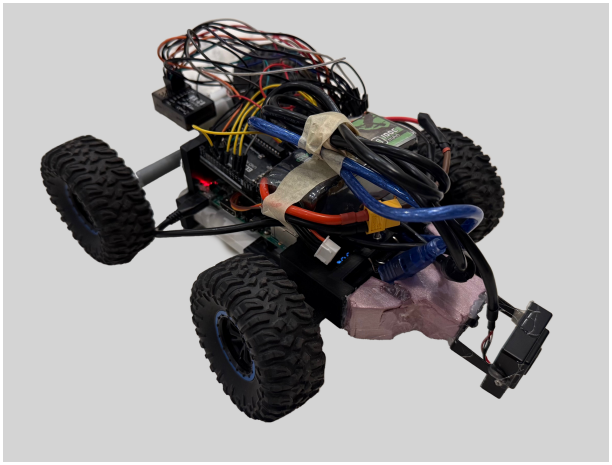

Figure 2: The vehicle.

operating system to ensure fast processing. All information is stored on a 64 GB SD card.

3.1.1 Camera

An external camera is plugged into the Raspberry Pi's USB port and used to collect image data.

Figure 3: Image collected by USB Camera.

3.2 Arduino

An Arduino Uno R3 was used for our micro controller. We choose an Arduino because we already owned one, and its, easy to use and open source, IDE allowed us to easily receive sensor and remote signals and control servos and motors.

3.2.1 Motor Control & Motors

The SCORAV contains two different motors, a servo and a direct current brushed motor. In data collection driving mode, both motors were controlled with a common remote control sensor and receiver through the Arduino. The signal from the remote control was a Pulse Width Modulation (PWM) signal, that rapidly switches between a high and low signal. To send information the controller modulates the speed of rapid switches between one and two microseconds.

The PWM signal input allowed for easy implementation with our steering servo as most standard servos are configured to receive PWM signals.

We choose to use a six volt direct current brushed motor that was geared down to increase torque. We choose to use this motor as apposed to a larger brush less motor because we already owned it and it was of sufficient size for our prototype. We used an L293D H bridge as a motor controller. Our input PWM signal was then mapped to the range accepted by the L293D of [-225 to 225]. We choose to use this motor controller because we already had one but its older architecture was somewhat limiting due it its inefficiency and large voltage drop. If a larger SCORAV were to be built, this setup would be easily replaced by either a larger brush less motor and motor controller that could directly accept the PWM signal.

3.2.2 Speed Consistency and Hall Effect

Maintaining a consistent speed is important for data collection, but would become even more vital to drive an autonomous vehicle. Constant motor power supply is insufficient, as it represents an accelerational force, rather than a constant velocity. The velocity of the rover would be subject to slope, terrain, and acceleration time.

Thus, a small magnet is affixed to the axle of the rover, and a hall sensor is placed such that the magnet passes it once a rotation. Hall sensors detect the presence of a magnetic field due to the potential difference the field causes across a current-carrying strip.[4] Thus, we can detect the time between pulses and adjust motor power to achieve a consistent pulse length.

This is accomplished using a "PD Controller." The P term creates a negative feedback loop effect, adjusting the power given to the motor based on the interval seen by the hall sensor. And, a secondary D term counters a high derivative. If the change in wheel speed is too great, the D term pushes it down, and vise versa. This regulates oscillation and overshoot. Experimental constants K_p and K_d regulate how much each term changes the motor speed.[5] Because hall sensors tend to be imperfect and mechanical jitter can create spikes, the interval is taken as a running average rather than an absolute last interval.

```
1  #using a PD controller to adjust motor power
2  void adjust_speed() {
3
4      #keep track of interval length to monitor change
5      static long last_interval = pulse_interval;
6
7      #calculate deviation from target_interval
```

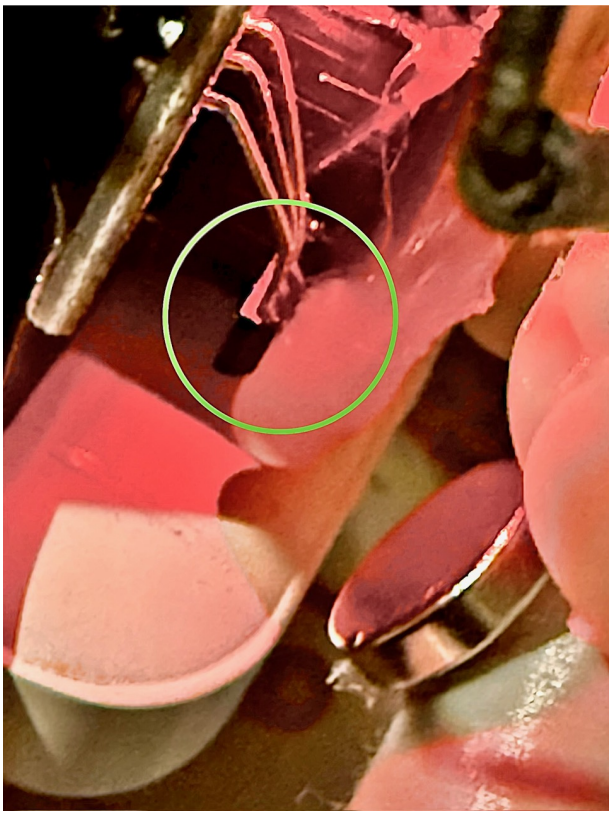

Figure 4: Hall sensor and magnet shown on vehicle. The red light indicates a magnetic signal.

```
    long error = (long)pulse_interval - (long)target_interval;

    #determine how much the interval has changed to apply damping
    long interval_change = pulse_interval - last_interval;

    #calculate how much to adjust motor
    float delta = Kp * error + Kd * interval_change;

    #apply changes
    motor_power += delta;

    #set-up for next function call
    last_interval = pulse_interval;
}

```

3.3 Serial Connection

Communication between the Arduino and Raspberry Pi is done through a serial connection. The Arduino sends information over the wire between the two electronics, and the Pi uses this information to perform processes, notably recording servo angle. A similar process would be used in reverse to supply the Arduino with servo angle provided by the machine learning model.

4 Data Collection

On startup, the main python script of the Raspberry Pi creates a new folder, titled the date and time of startup. From then, collected images and data are stored in that folder until the Pi is powered off. These images constitute a

training dataset which will be used to train a machine learning model, which will correlate servo angle with image, allowing it to generate new servo angles from unseen images.

The image and servo pairs are stored as images in a folder, and a csv file correlating each image with the corresponding servo angle. Motor value capability exists in case it is deemed necessary in the future, but is currently not stored, as speed is kept constant through the Hall sensor and PD controller. The servo angle is stored as the signal received from the remote control. This allows for consistent conversion to servo power.

```
image,servo,motor
img001.jpg,1500,120
img002.jpg,1600,140
img003.jpg,1700,180
```

Figure 5: Example CSV snippet

4.1 Visualizing Data

A small python script was created to visualize the correlation between image and servo data. The script plots a red arrow on top the image, indicating the direction of turn specified by the servo angle.

5 Model Training

5.1 Training Data

Training data was collected by driving the vehicle through Alto Park in Santa Fe. We felt that the clearly delineated difference between the grass and the sidewalk would help the model recognize direction with our limited training

Figure 6: A visual display of a servo angle correlated with an image.

data. Our data set was made up of 375 images with corresponding servo values for each image.

We ran into some trouble gathering this training data due to the limited capability of our prototype and limited sidewalk distance. To gather a more comprehensive dataset, it would be ideal to construct a more capable vehicle that could be used on long stretches of single track trail.

Figure 7: Example training images.

5.2 Image Processing

Each image was then preprocessed and cropped to remove noise and reduce image complexity.

```
def preprocess_image(img):
    img = img[60:400, :, :]
    img = cv2.resize(img, (200, 66))
    img = img.astype(np.float32)
    img = (img / 127.5) - 1.0
    return img

def denormalize_image(img):
    img = ((img + 1.0) * 127.5).clip(0, 255).astype(np.uint8)
    return img
```

Figure 8: Example preprocessed image.

The appropriately formatted data was then segmented into training, verification, and testing datasets with a 70/15/15 split.

5.3 Training

Our model used a convolutional neural network (CNN) as a regression model to predict a continuous servo steering value for any input image. The CNN was built using three layers with different characteristics to help the model recognize

complex patterns withing the images. As an image moves through the network, the number of feature channels increases from 16 to 32 and then to 64, allowing the model to learn more detailed and abstract information. After these layers, the output is flattened and passed through three fully connected layers, which gradually reduces the data down to a single value representing the predicted steering value.

This model was trained on the Training split of our data witch consisted of about 250 samples, and was trained over a total of fifty epochs.

5.4 Verification

As training occurred, accuracy and loss was monitored on the validation dataset. See fig 9. This demonstrated that model performance plateaued around 30 epochs and that training should be terminated at that point to avoid overfitting.

Figure 9: Validation and training loss over epochs.

5.5 Testing

After model training was complete, the model was tested on the testing dataset of approximately 50 samples.

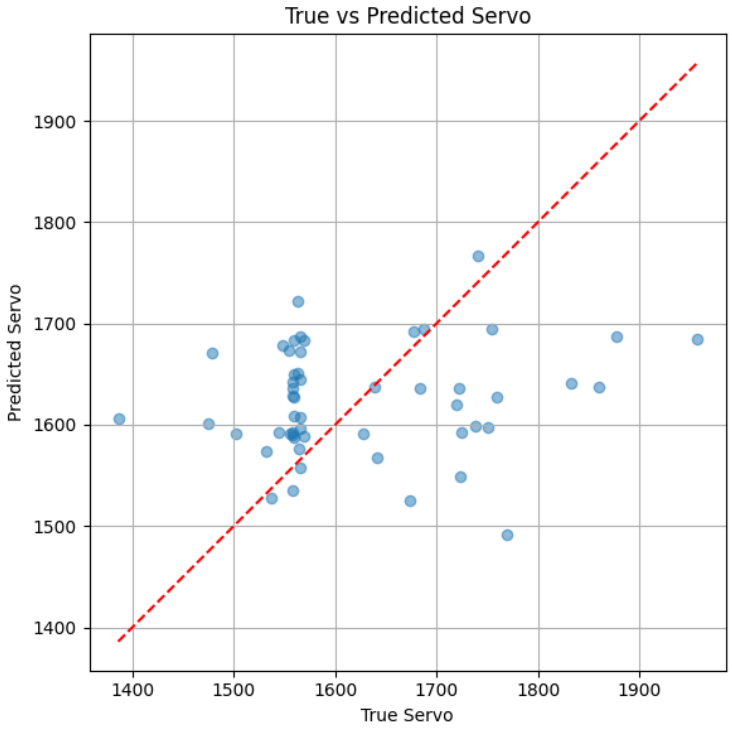

Figure 10: True vs predicted servo values for testing dataset. (Large spike of servo values around 1550 on the true servo values is due to that servo location being strait.)

As seen in figure 10, there was no statistically significant relationship between the predicted servo value and true servo value. However this poor result is not a reflection of the strength of the model architecture and underlying methodology of the project. This unfortunate result is almost certainly due to a small sample size of training data. Our sample size of 250 training images is nowhere near enough for the CNN to recognize patters within the image. As mentioned in

5.1, our limited dataset was due to limitations with our vehicle prototype and so the dataset weakness could be remedied with a more capable vehicle.

6 Auxiliary Apps

6.1 Litter Detection

An obvious yet nontrivial application of the vehicle is litter detection. Intended to follow trails, the vehicle would be apt at determining litter hot spots and flagging them for clean up. By correlating image data with GPS, the rover could create a dataset which could later be used to identify hot spots of litter.

A large part of this implementation would require a working litter detection model, which was developed using a composite training dataset and a pretrained model.

6.1.1 Training Data

No explicit dataset exists segmenting area with litter and without. While many datasets indicate where litter is on an image, none explicitly sort for a binary classification model. Thus, a composite dataset was created. For littered images, the PlastOpol dataset [10] was used, and for nonlittered images, the DeepWeeds dataset [3] was combined with a generic sidewalk dataset [11] and rover data from a park. Combined, this meant that images with and without litter in a variety of locations and from a variety of angles was present.

6.1.2 Formatting

In order to appropriately format data and get the best results, the entire dataset was combed through, applying small random changes to saturation, blur, hue, and brightness. These values were determined experimentally based

Figure 11: Example litter images.

Figure 12: Example nonlitter images.

on yielding "real-looking" images, and ensured that the dataset was more varied so the model could perform on images from a moving rover in different lighting conditions.

```
def augment_image(image):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV).astype(np.float32)

    # hue
    hue_shift = random.randint(*hue_shift_range)
    hsv[:, :, 0] = (hsv[:, :, 0] + hue_shift) % 180

    # sat
    sat_factor = random.uniform(*saturation_range)
    hsv[:, :, 1] = np.clip(hsv[:, :, 1] * sat_factor, 0, 255)

    # brightness
    brightness_factor = random.uniform(*brightness_range)
    hsv[:, :, 2] = np.clip(hsv[:, :, 2] * brightness_factor, 0, 255)

    augmented = cv2.cvtColor(hsv.astype(np.uint8), cv2.COLOR_HSV2BGR)

    # blur
    blur_kernel = random.choice([k for k in range(*blur_range) if k % 2 == 1] + [3, 5])
```

```
20      augmented = cv2.GaussianBlur(augmented, (blur_kernel,
        ↪  blur_kernel), 0)
21
22      return augmented
```

Then, the dataset was segmented into training, verification, and testing data in an 80/10/10 split.

6.1.3 Training

A pre-trained model, MobileNetV3-Small [8] was used to train the litter detection model. This pre-built model already had edge detection and other important capabilities, so all that was required was modifying the head. The head of the model is a subsection, the last few layers involved in a classifier, which turns detected edges and shapes into the intended outputs. By training this head to result in a binary output (litter or no-litter), we used pre-existing architecture to support our development process.

The head was trained on the "training" section of our dataset, which included roughly 2000 images in either category. The model was trained over 10 epochs, meaning that it saw the entire training dataset 10 times, running a forward pass identify loss, and performing backpropagation to improve the model for each image, 10 times.

6.1.4 Verification

As training occurred, for each epoch, the model was tested on the verification segment. By monitoring loss and accuracy, it became clear that 10 epochs was sufficient training and did not risk overfitting. The model reached 100%

accuracy on the verification data, with consistently decreasing loss (error on training dataset used to influence model learning).

6.1.5 Testing

After the model was complete, testing ensured that the model could perform on completely new data. The model made a prediction for each image in the "testing" directory on whether or not litter was present, and this was compared against the actual value. The model yielded a 100% accuracy rate.

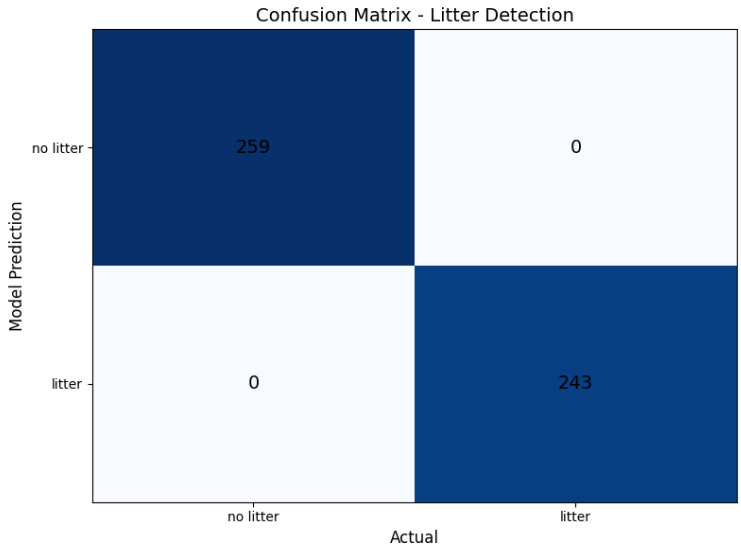

Figure 13: Confusion matrix of model prediction and actual litter status.

7 Conclusion

7.1 Success and Considerations

The goal of SCORAV was to develop a cost effective off-road vehicle that is capable of collecting ground level data to help with conservation and environmental efforts. Although our work fell short of this lofty goal it does represent a

considerable step toward a final product and is built upon a solid methodology that could lead the project to success with continued development.

Our vehicle prototype served well to proof our concepts for a final vehicle design and it represents significant learning in the engineering design process. With the insights gained from this prototype, given more time we would be able to construct a much more capable vehicle with potential for real application.

Although some of our software is still under development, the current SCORAV offers a significant step in hardware software integration as well as a solid methodology for full autonomy. Along with this the success of our proof of concept application for litter detection demonstrates the applicability of a fleet of SCORAVs for a variety of conservation purposes. With further development it is undeniable that SCORAV has enormous potential for conservation.

7.2 Next Steps

Clearly, the project remains in development. While significant progress has been made, and a working framework for training an off-road vehicle has been created, a driving model is not complete. Major limitations come from the hardware itself, so immediate next steps would be designing a new vehicle. The vehicle would have off-road capabilities to avoid another build required, but would more specifically have tighter assembly to ensure that steering angle is dependent solely on servo angle, whereas the current model is largely dependent on environmental factors and servo history (the wheels do not fully return to straight when released).

Additionally, collecting data on more varied terrain would be paramount to training an effective model to navigate on trail. This would require not only a

robust vehicle, but also a time investment in consistent driving.

7.2.1 Model Implementation

Additionally, the model could undergo further implementation. More auxiliary applications than just litter detection could be created, such as fallen tree detection, invasive species detection, and trail condition or forest health surveys. Beyond this, GPS set-up would ensure that images collected contain location data, rendering these auxiliary applications useful.

7.3 SCORAV Applications

Although our current vehicle and self driving model is lacking, the SCORAV offers significant potential applications for conservation and environmentalist groups. Our trash detection model is one potential application the SCORAV could be loaded with a myriad of applications to collect and process data for various environmental matters that could be aided by ground level data acquisition. I few possible application include:

- Trail health inspections
- Invasive species detection
- Forest health
- Illegal activity detection (Poaching, Unauthorized land use)
- Biomass density

To accomplish these goals, a conservation organization would ideally maintain a fleet of SCORAVS that could be deployed throughout a large area with charging docs or solar panels in order to continuously collect data. This system would be able to limit conservation costs that comes along with sending employees to

collect data manually. Along with this our system provides an advantage over many similar systems due to its relatively low cost that would allow for larger fleets on low budgets.

7.4 Learning Highlights

While the final result of this project was not what we initially desired, this remains our biggest learning accomplishment in our 4 and 2 years as part of the SuperComputing Challenge. We dove into hardware development, a new sector for both of us, and developed a ton of skill in regulating motor control and working with operating systems such as Raspberry Pi. Beyond that, we learned to work with the physical hardware, soldering connections and assembling and designing 3D printed parts. We learned the importance of careful fit when it comes to physical apparatus, especially when trying to train machine learning models off data collected.

From a software standpoint, we learned a lot about machine learning and image recognition, and how to write code for different machines and get them to talk to each other (Arduino and Pi). We discovered different equations and algorithms such as the PD controller. Additionally, we built on the backs of much work, including pre-trained image recognition models and existing crowd-sourced datasets.

7.4.1 Frying the H-Bridge

One of the most significant learning experiences happened about two weeks before the final report was due. Accidentally plugging the 14V lithium ion in backwards, we fried our H-bridge, rendering not only the component, but our entire Arduino set-up useless. We had to rebuild the entire connection to the Arduino, but this time learned from our mistakes. We color coded our wires

and didn't solder in connections that we didn't need to, ensuring that parts were replaceable if needed later. We learned that Murphy's Law isn't a cynical perspective, but rather a wise counsel.

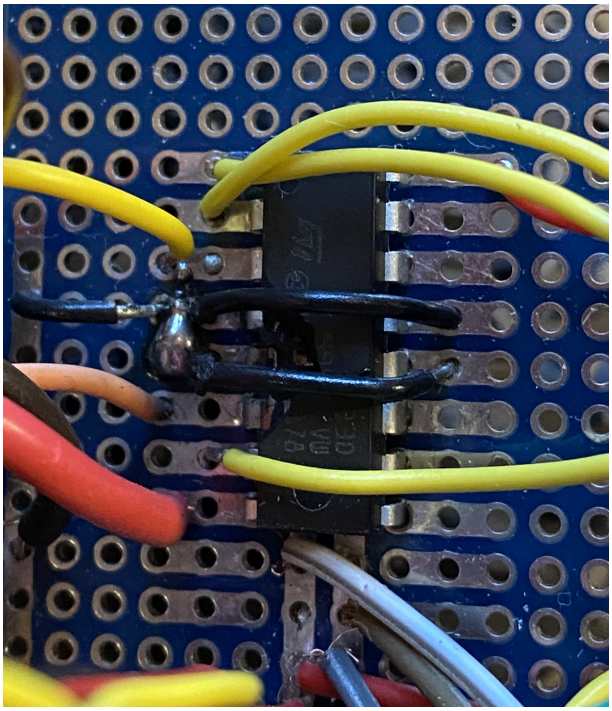

Figure 14: Aftermath of blown H Bridge motor controller.

7.5 Generative AI Use

In the interest of adhering to industry standards and utilizing available tools, the generative artificial intelligence Chat GPT was used in select cases to streamline the coding process and improve efficiency. The AI was used to write minimal code, but rather to debug and understand hardware specifications. Examples include determining why camera access was malfunctioning on the Raspberry PI and deciding which voltage to supply the H-Bridge to adequately power the motors.

8 Works Cited

References

[1] AirLab. "Offroad Navigation." *The AirLab*, https://theairlab.org/offroad/. Accessed 2 Apr. 2026.

[2] American Trails. "FAQ: Trash on Trails." *American Trails*, https://www.americantrails.org/resources/faq-trash-on-trails. Accessed 2 Apr. 2026.

[3] "DeepWeeds." *Kaggle*, https://www.kaggle.com/datasets/imsparsh/deepweeds. Accessed 2 Apr. 2026.

[4] Electronics Tutorials. "Hall Effect." *Electronics Tutorials*, https://www.electronics-tutorials.ws/electromagnetism/hall-effect.html. Accessed 2 Apr. 2026.

[5] Electronics Tutorials. "Hall Effect." *Electronics Tutorials*, https://www.electronics-tutorials.ws/electromagnetism/hall-effect.html. Accessed 2 Apr. 2026.

[6] Forest Service, U.S. Department of Agriculture. "Maintaining Trails." *U.S. Forest Service*, https://www.fs.usda.gov/science-technology/infrastructure/maintaining. Accessed 2 Apr. 2026.

[7] U.S. Department of the Interior. "Invasive Species: Finding Solutions to Stop Their Spread." *U.S. Department of the Interior*, https://www.doi.gov/blog/invasive-species-finding-solutions-stop-their-spread. Accessed 2 Apr. 2026.

[8] PyTorch. "MobileNet_V3_Small." *PyTorch Vision Documentation*, https://docs.pytorch.org/vision/main/models/generated/torchvision.models.mobilenet_v3_small.html. Accessed 2 Apr. 2026.

[9] Overland AI. "Overland AI." *Overland AI*, https://www.overland.ai/. Accessed 2 Apr. 2026.

[10] "PlastOpol: A Dataset for Litter Detection." *WUR*, https://research.wur.nl/en/datasets/plastopol-a-dataset-for-litter-detection/. Accessed 2 Apr. 2026.

[11] "Sidewalk-dz4ug." *Roboflow Universe*, https://universe.roboflow.com/school-stpl7/sidewalk-dz4ug/. Accessed 2 Apr. 2026.

[12] Washington Trails Association. "Report: Forest Service Trails Suffer from Lack of Maintenance after 15 Years." *Washington Trails Association*, https://www.wta.org/news/signpost/report-forest-service-trails-suffer-lack-maintenance-15-years. Accessed 2 Apr. 2026.