Computer Vision & Machine Learning for FRC

Team 2635 Lake Monsters

March 2024

Introduction

This short paper describes FRC Team 2635's experience with computer vision (CV) and machine learning (ML) between 2020-2024. We begin with a discussion of potential applications for computer vision in gameplay, then describe our end-to-end computer vision workflow. We hope this resource will assist teams in utilizing computer vision and machine learning on their own robots.

Why computer vision? And why machine learning?

First of all: why not?

But also: It's Cool and Interesting and Fun™

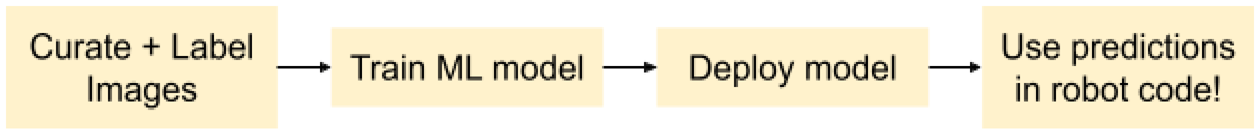
We have historically struggled with challenges including field visibility (2022 hangers, 2019 cargo ship) and reliably executing autonomous paths. With recent advances in semi-autonomous vehicles, among other things, we turned to machine-learning-driven computer vision to address some of these challenges. In contrast with non-machine learning computer vision systems like GRIP, our approach is robust for variability in lighting conditions, game element orientation, and more. Using a 3D depth camera, we are also able to extract accurate depth data to use in both autonomous and teleop.

Application: Rapid React (2022)

We primarily use computer vision in autonomous. With 2022's Rapid React game, instead of preplanning every path and rotation, we only needed to hard-code a few rotations to ensure the cargo was in the camera's field of view. Employing computer vision allowed us to perform closed-loop driving, where we zeroed in on the cargo and adjusted for changes due to gyro error, poor initial robot placement, etc. (Confession: we're just not the best at dead reckoning, and computer vision helps us do better.).

Against the will of our drivers, we also programmed a Button of Death™, which, when pressed, triggered the robot to autonomously drive to the nearest cargo that matched the alliance color (it was set at 60-70% maximum power, so, pretty fast/scary). Unfortunately, this feature was never utilized in the 2022 season, but we have hopes for it moving forward.

Process Overview

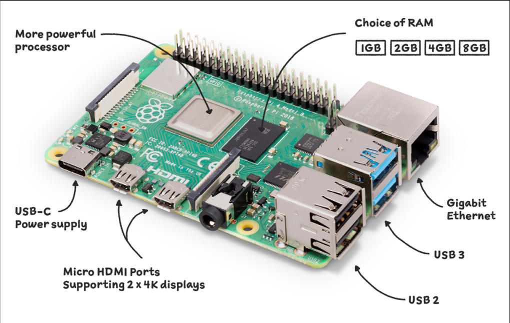


This paper will present an overview of each stage, along with links to additional resources and code examples for further study. We primarily focus on the *application* of ML to FRC---for a deeper dive into the inner workings of ML in computer vision, we recommend checking out the Roboflow blog and the many articles linked in this paper.

Hardware

If you've made it this far and are considering using ML/CV for your robot, yay! We used the following hardware on our robots in 2021-22.

Luxonis OAK depth camera	$249.00	We chose this camera because it runs AI models on the camera itself, not the co-processor, which reduces inference time. It also offers a simple one-step-deployment with our software provider, Roboflow.
Raspberry Pi 4 64GB microSD 8GB RAM	$35	We use the Pi to send detections over the NetworkTables, which can then be used by the roboRIO. *Note: These can be hard to get due to the semiconductor shortage, so make sure to*

 More powerful processor Choice of RAM 1GB 2GB 4GB 8GB Gigabit Ethernet USB 3 USB 2 USB-C Power supply Micro HDMI Ports Supporting 2 x 4K displays		*order early!*
External battery (to power the Pi and camera)	Here is a [$50 Anker one]() that lasts a long time before needing a recharge	Any portable phone charger will work, but it needs to have at least a 5V output with a minimum of 2 USB type A ports. It's nice to have two batteries, but we have been able to get away with using just one during a full day of competition.

Model Training

We utilize the [Roboflow]() platform to develop our model. It's an all-inclusive, easy-to-use solution that allows you to label images, preprocess/augment your dataset, and train your model on Roboflow servers. We're also fans of the collaboration features, Roboflow's extensive [user guides]() and blogs, and their large collection of pre-trained models and datasets.

But first, some background

(Or: Speedrunning ML)

Computer vision is a broad, yet specialized field concerned with some of these [general tasks](), among others:

- **[Classification]()** - What category of object is in the image? Is it a cat or dog? Apple or orange?
- **[Object Detection]()** - Where is the object of interest in the image? Is the cat five feet away or five inches away?
- **Landmark Detection** - What are the "important points" or identifying characteristics of the object in the image?

We are primarily interested in the task of object detection. We want our robot to be able to determine whether there are game objects in the field of view. If there are game objects in view, we want to figure out where they are located so the robot can drive to and pick up the object.

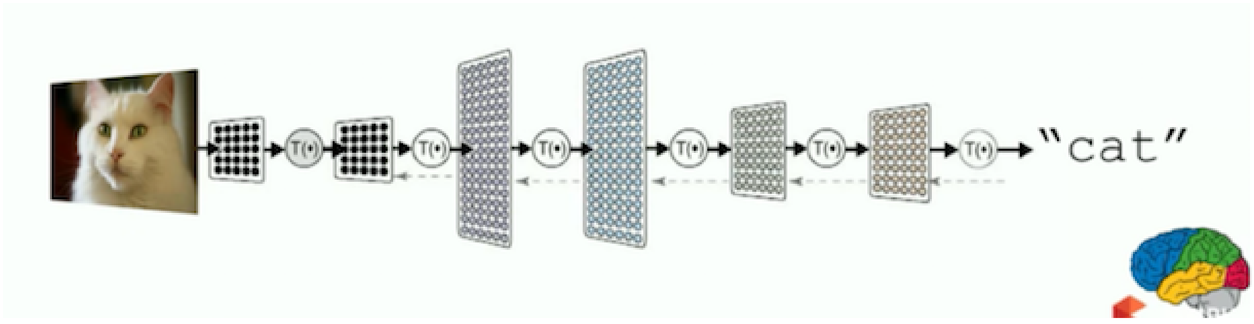
(This question also involves classification to the extent that we want to accurately distinguish different types of game elements from each other.)

Neural Networks: A Detour

The foundation of deep learning is neural networks, a type of AI algorithm modeled after the human brain that adaptively learns how to perform various tasks from many labeled examples or *training data*. Neural networks come in many flavors depending on the application (ex. text translation vs. image recognition). Computer vision is dominated by a special type of neural network called a *Convolutional Neural Network (CNN)*, which we will not describe in this paper. If you are interested, here are some great explainers on CNNs.

Once the neural network has been trained by viewing and processing enough examples, it will be able to generalize its predictions to new data. This excerpt from DeepAI sums it up well:

> *Imagine the "simple" problem of trying to determine whether or not an image contains a cat. While this is rather easy for a human to figure out, it is much more difficult to train a computer to identify a cat in an image using classical methods. Considering the diverse possibilities of how a cat may look in a picture, writing code to account for every scenario is almost impossible. But using machine learning, and more specifically neural networks, the program can use a generalized approach to understanding the content in an image. Using several layers of functions to decompose the image into data points and information that a computer can use, the neural network can start to identify trends that exist across the many, many examples that it processes and classify images by their similarities.*



(image is taken from a Google Tech Talk by Jeff Dean at Campus Seoul on March 7, 2016)

After processing many training examples of cat images, the algorithm has a model of what elements, and their respective relationships, in an image are important to consider

> *when deciding whether a cat is present in the picture or not. When evaluating a new image, the neural net compares the data points about the new image to its model, which is based off of all previous evaluations. It then uses some simple statistics to decide whether the image contains a cat or not based on how closely it matches the model.*

3Blue1Brown also has a really great video series on neural networks.

Creating + Labeling An Image Dataset: Tips for FRC

Now we see the importance of creating a large, robust training dataset. As they say, "garbage in equals garbage out." Without enough training data, the model may struggle to perform on the field.

> You can start creating a training dataset with the field images that are released alongside the game manual. We will also take videos of the game objects in our lab and sample the video frames every few seconds to get more training data. When creating the dataset, be aware of the characteristics (ex. shape and color) of game elements and include non-game objects of similar shape/color in the dataset. For the 2022 season, we took many pictures of the cargo next to the old robot bumper to make sure the model did not incorrectly categorize red bumpers as cargo. We also recommend taking pictures at an angle similar to how the camera will be mounted on the robot. If the camera is mounted close to the ground, it makes more sense to gather training images at a low angle rather than from a bird's-eye point of view.

After curating an image dataset, you can upload the files into Roboflow and annotate them. Annotating the dataset involves drawing *bounding boxes* around the game elements. These annotations are what teach the neural network to recognize the objects enclosed by the bounding boxes. This Roboflow tutorial walks through the process of uploading and labeling the dataset.

**Note about model-assisted labeling:* Training a model often requires multiple iterations, and model-assisted labeling can significantly speed up the somewhat tedious process of labeling the dataset. If you have already trained a model using Roboflow Train and later decide you want to add more images to your dataset, you can use that already-trained model to generate initial annotations that you can manually adjust and fine-tune. Here is the guide.

Image Preprocessing + Augmentation

Now that you've curated a nice dataset, it's time to preprocess and augment the dataset before training. This getting started guide walks through the entire process, from labeling to preprocessing/augmenting to training.

You'll need to generate a new version in Roboflow, which is where you add preprocessing steps and augmentation. The default preprocessing step involves cropping all the images so that they are the right size for model training. Make sure you choose the right type of crop for the dataset—we want to maintain the raw image's aspect ratio so objects do not become overly distorted.

Augmentation involves applying different filters and transformations (ex. rotating the image, lowering/increasing exposure, adding blur and noise, etc.) to expand the dataset. Applying augmentation makes the model more robust to real-world deployment. We encourage teams to experiment with different combinations of augmentations. Here are some that we use:

- **Grayscale** - Helps the model recognize shapes and object contours rather than just rely on the color of the game element.
  - On our first iteration of 2023's cone/cube model, we did not use grayscale and the model saw "cones" in yellow wood grain, carpet specks, and sorting bins. This told us the model was primarily using color to determine whether something was a cone or not.
- **Blur + Noise** - Accounts for lag and a fast-moving robot
- **Rotation + Reflection** - Non-symmetrical game elements like cones can be found in all sorts of orientations on the field.

Model Training

After applying the augmentations to your dataset, you can move on to training your model. In Roboflow, this consists of just pressing the train button and waiting. More information about the training process can be found on the Roboflow Docs and video.

We tried another method of training this year, where we used the Luxonis depthai-ml-training Yolov7 Jupyter notebook. All this process involves is following along with what it says, and it produces the same result, and sometimes it's faster.

After the model is trained, we convert the finished YOLO model to OpenVino format to deploy in MonsterVision and the Luxonis camera.

Model Deployment

With the model in OpenVino format, you download it onto the Raspberry Pi's SD card and change the config file to match the parameters of your setup. We describe a more technical, in-depth process on our GitHub for MonsterVision.

Acknowledgments

This project would not have been possible without Roboflow. We thank them for sponsoring our team with training credits and providing extensive technical support over the past few years.