Implementing Edge Computer Vision for Vex Over-Under Robotics Competition

Nathanael Oliver, Zachary St. Thomas, Zhymir Thompson

August 27, 2024

Abstract

The goal of the project was to implement a computer vision model on a small Vex robot for autonomous navigation and movement of "Triballs" to a predetermined goal zone for the Vex Robotics Over-Under competition. The location of the Tri-Balls is not predetermined, requiring real-time recognition of the Triballs and appropriate navigation to obtain and then move the game objects into the scoring zone. The rules for this competition allocate a 1 minute time limit and prevent the use of external communications devices meaning that this problem requires an edge device to perform real-time recognition. We have collected and detailed the literature we have used to learn about edge implementations of object detection models and the training and loading of these models onto a Jetson Nano. After initially developing a simple object detection model to find and deliver tri-balls to a predetermined goal zone, we then used tinyYOLO to implement object tracking to ensure the robot could handle multiple tri-balls simultaneously.

Keywords:
Edge Computing, Vex Robotics, Autonomous Navigation, Object Detection Dataset

1 Introduction

Vex Robotics has been providing competition opportunities for students and researchers to develop autonomous and manual driving robots for their competitions since 2007, all concerning objectives to be accomplished via manipulating their environment. Our project aims to apply an autonomous vehicle to be able to compete alongside human-driven devices. The unique challenges of Vex Robotics competitions are the short time limits on how long the robot may operate and the considerable amount of objects to move and manipulate.

This year, the goal of the game is to score Triballs into a goal on the opposing side of the field. This paper and the work associated are focused on the autonomous portion of the competition, in which the robot is required to perform without any human input or outside influence. For this reason, our robot must be able to identify the relative location of triballs on the field and move them into the goal area independently. The robot will start by launching the triballs onto the other side of the field at the beginning of the match and navigate to the other side of the field where it will locate triballs, navigate to the triballs, locate the goal, and score the triballs into the goal.

We are working alongside a small robotics team to build and assist in configuring the robot to be able to run and use our model, while the bulk of our work will be focused on creating the dataset for the model, training our model, and ensuring it can perform the required functionality in competitive time on the Jetson Nano. Some of the biggest challenges unique to this competition that we will have to overcome include the different colors of the Tri-balls, the random arrival and placement of the tri-balls to the playspace, and perhaps primarily that the cameras we are using will have to manually estimate the distances of objects in their frame. We deployed both a conventional CNN model and a tinyYOLO model for object tracking.

2 Literature Review

2.1 Real-World Robot Applications of Foundation Models

"Real-World Robot Applications of Foundation Models: A Review" discusses the workflow that robotic algorithms follow from sensor data to performance and evaluates the efficacy of these approaches. Typically, these algorithms take in sensor data, process this data in order to map the environment, find a path using this map and then use this map to output the motor values that are needed to follow the path. [1] This paper discusses how some models have skipped either the high level planning of the path and convert directly from the map to motor values or skipp mapping and create a path based on the raw sensor data. This paper proposes training a model that takes in the raw sensor values and output the motor values, skipping all of the higher level complexities in order to decrease latency in the robot. This paper proposes more unity among frameworks for robotics so that training models to take data as input and motor values as output becomes more viable for widespread use.

2.2 Enhancement of a VEX Robot with an Onboard Vision System

Enhancement of a VEX Robot with an Onboard Vision System is a direct implementation of a Vex robot with image processing capabilities via a Raspberry PI mounted to the robot. The goal of the paper was to demonstrate how to combine a PI with a VEX robot to allow the robot to perform more complex tasks without special equipment [2]. The PI is able to process images with its camera module, use a color-based feature extraction vision algorithm, and send instructions to the VEX microcontroller to enable autonomous movement and basic target tracking. Furthermore, the speed of the Raspberry PI makes the robot capable of real-time image processing. The report details step by step how to configure the Vex Robot, the Raspberry PI, and all required software packages to achieve autonomous functionality and communication between the Robot and the PI. This report focused on the educational applications of introducing computer vision to Vex robots, drafting three hypothetical project plans that would use the introduced features to help students learn for themselves how the vex robot achieves vision and familiarizing themselves with the tools used, such as C, Open CV, and the PI itself. The paper further demonstrates how a VEX robot can be designed to perform more complex tasks in a cost-effective manner [2].

2.3 A Method for Optimizing Deep Learning Object Detection in Edge Computing

Kim et al. discuss methods for optimizing the Tensor Virtual Machine for object detection on edge devices. To optimize inference on edge devices, the paper discusses breaking the process of reading data, pre-processing data, making the inference, post-processing, and writing into five subprocesses. This allows the machine to parallelize the work and perform operations on the CPU and GPU simultaneously. [3] Additionally, this paper discusses using compilers specific to TVM to optimize the code for the device that the inference function will be running on.

2.4 Edge-based Street Object Detection

Nagaraj et al. detailed a real-world implementation of edge computing in smart city cameras, running an image recognition model on a Jetson TX2 to detect city objects such as traffic and pedestrians. Edge computing was implemented to address concerns of limited city bandwidth hindering real-time model processing of images for their smart city cameras. They implemented two models and compared their performances. They first used a DetectNet model, which performs object classification and estimates bounding boxes, modified to ensure classification of 14 classes and compatibility of their dataset. [4] They also used a YOLO model and found the YOLO model beat out the DetectNet model. [4] Despite low accuracy on even the YOLO model, this paper was an early, comprehensive implementation of image recognition, object classification, and object tracking on an edge device. This was all tested on the emerging real-world problem of detecting traffic objects.

2.5 Pac-Man Pete: An Extensible Framework for Building AI in VEX Robotics

Zietek et al. wrote a comprehensive report detailing the use of computer vision models and deploying them on Vex robots. Their competition involved the robot picking up small purple rings and stacking them on a pole at the center of their play area. They trained their model via unity simulation, modeling the play area, rings and goal pole into their own areas. Additionally, the Purdue group implemented image preprocessing tools such as GRIP in the framework to create contrasts between objects of interest and the background to allow for the model to learn and detect objects from the environment significantly easier[5]. With this framework established the group had a competitive robot for the competition that was entirely autonomously run.

3 Proposed Work

3.1 Design object detection model

The model chosen to perform object detection was tiny YOLO. Initially, we planned to adapt a simple CNN model to perform the object detection and bounding-box application. However, due to the sheer number of distinct objects and the large variety of potential bounding boxes, it was impractical to manually modify such a model to work for the given problem. Tiny YOLO was ultimately a better fit for our problem.

The initial CNN model was developed as a simple object detection solution to ensure the robot could detect tri-balls and helped us establish a working model-to-robot pipeline. The model has 26 layers including activation and batch normalization layers, where the layers can be divided into three blocks of two sets, each set consisting of a 2x2 convolution followed by batch normalization, activation, max pooling, and dropout. Each block's convolutions use twice as many kernels as the previous layer (32, 64, 128), and they use ReLU activation. Finally, the network flattens the output and runs it through a 128-wide Dense layer, then a binary softmax layer to represent the presence or absence of a tri-ball.

Our initial plan was to then augment this model to then produce bounding boxes on tri-balls, but attempting to convert the relatively simple CNN setup to a model capable of drawing bounding boxes proved unfeasible due to the complexity of the task and numerous challenges that came about during our conversion process. We did not immediately attempt to use YOLO due to concerns that our limited dataset of 151 images was insufficient compared to the recommended requirement of 1,500 images for a robust model[6].

After realizing the limitations of the model we trained initially, we decided to train a YOLO model. We used the tiny architecture linked here to configure the model. We changed some of the hyperparameters and layers to fit the needs of our model and give the output for 3 classes, those being tri-balls, the goal, and goal barriers.

3.2 Creating the Dataset

The dataset[7] used to train the model was created manually. As of writing this report, there is no other dataset for the Triball competition. Since there was not a similar dataset available for the given problem, we chose to collect data on our own. For the dataset, we took pictures of the field along with the objects required for the competition. Specifically, we took pictures of the goals, the barrier, the field, and the Triballs. The Triballs were strewn about in various patterns in different amounts to simulate the in-game field. All of these pictures were taken from different angles and from the perspective of the VEX robot. Once all of the pictures were taken, they needed to be annotated. This was done manually with the aid of Label Studio **label-studio**. We drew boxes around the goals, barriers, and Triballs in every image taken by the robot. We created a fully annotated dataset of 151 images used for training the model.

3.3 Deploy model to Jetson Nano

Deployment was straightforward, though some obstacles were encountered. After training the model, we used the API from TensorFlow to convert the model to a TFLite model. The Jetson Nano, however, did not have up-to-date versioning of the HDF5 library, which was required for h5py, which is a Python package required for TensorFlow. Installing the newest version of HDF5 took a substantial amount of time, as well as configuring the Jetson Nano to only recognize the new version of HDF5 rather than the pre-installed version. After this had been configured, the rest of deployment went as anticipated, and the model was able to be run on the Jetson Nano and the robot.

3.4 Integrate object detection model with route planning

We programmed the robot to move based on a "Point and Turn" method, where the robot would continuously turn until a Tri-ball was detected, move towards the tri-ball, then turn towards the goal with the tri-ball once arriving and delivering the tri-ball to the goal. The actual effectiveness of this method varied due to the robot not always bringing along the tri-ball when it went to deliver it to the goal.

Algorithm 1 Turn and Push

```
while prediction == 'no triball in frame' do
    turnLeft()
    prediction = model.predict(camera.snapshot())
end while
while prediction == 'triball in frame' do
    driveForward()
    prediction = model.predict(camera.snapshot())
end while
while prediction == 'no goal in frame' do
    turnRight()
    prediction = model.predict(camera.snapshot())
end while
    driveForward()
while movingForward do
    driveForward()
end while
    driveBackwards()
    wait(2)
    stop()
```

4 Experimental Results

4.1 Detection Model performance

Our detection model [7] was capable of minimizing the mean squared error loss function to a value of around 0.17 to 0.18. This may has partially been a consequence of the model overfitting to training data, however we were able to perform tri-ball detection with decent consistancy when the triball was isolated from other triballs. Our model was exclusively able to detect triballs, however, meaning that our robot had to have the location of the goal zone pre-programmed into it to ensure it was capable of delivering the tri-balls it detected to the goal zone.

The limited capability of our detection model required the pre-programmed logic for our robot to deliver the tri-balls to the goal zone. This was in contrast to our initial objective of detecting and tracking the goal to ensure the robot could detect the goal and recognize it as an object. The detection model overall was incapable of detecting triballs while moving at competitive speeds.

When multiple Triballs were present on the field, the robot would continue navigating to the other triballs when it came into possesion of a triball rather than scoring the triball it was in possession of. This was due to the fact that this algorithm did not have any object tracking or state memory. Additionally, due to the limited path planning algorithm implemented, the robot was not capable of getting triballs that were on the edge of the field to score them.

4.2 YOLO Model Training

Detailed below is our YOLO model's loss during training[7]. Training proved a comprehensive task due to dependency issues with the Jetson Nano and the limited data we were able to create that was compatible.

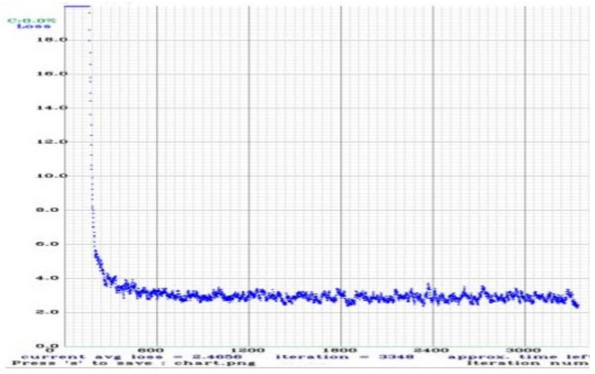

Figure 1: YOLO Loss Function during training

5 Conclusion and Future Work

We were able to train a model which could autonomously locate a triball, navigate to the triball, find the goal, and score the triball into the goal. We were able to deploy this model to the edge so that it could run on the robot. Additionally, we were able to train a tinyYOLO model that would provide more accurate identification of Triballs and allow for more advanced path planning algorithms.

Going forward, we would like to be able to implement advanced planning algorithms. Utilize tracking of game objects in order to not get distracted by other elements on the field. We would also like to find a way to account for the blur that occurs when pictures are taken at high speeds allowing the robot to act faster.

References

[1] K. Kawaharazuka, T. Matsushima, A. Gambardella, J. Guo, C. Paxton, and A. Zeng, "Real-world robot applications of foundation models: A review," *arXiv preprint arXiv:2402.05741*, 2024. DOI: 10.1007/s10462-021-10059-3. [Online]. Available: http://arxiv.org/abs/2402.05741.

[2] L. Ma and M. Alborati, "Enhancement of a vex robot with an onboard vision system," in *2018 IEEE 10th International Conference on Engineering Education (ICEED)*, Kuala Lumpur, Malaysia, 2018, pp. 117–121. DOI: 10.1109/ICEED.2018.8626918.

[3] R. Kim, J. Kim, H. Yoo, and S. C. Kim, "Implementation of deep learning based intelligent image analysis on an edge ai platform using heterogeneous ai accelerators," in *2023 14th International Conference on Information and Communication Technology Convergence (ICTC)*, 2023, pp. 1347–1349.

[4] S. Nagaraj, B. Muthiyan, S. Ravi, V. Menezes, K. Kapoor, and H. Jeon, "Edge-based street object detection," in *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, San Francisco, CA, USA, 2017, pp. 1–4. DOI: 10.1109/UIC-ATC.2017.8397675.

[5] J. Zietek *et al.*, *Pac-man pete: An extensible framework for building ai in vex robotics*, Purdue, West Lafayette, Indiana, 2022.

[6] L. Gur and Arie, "A practical guide for object detection with yolo algorithm," *Medium, Towards Data Science*, 2023.

[7] N. Oliver and et al., *Vex serial*, GitHub Repository, University of South Carolina, 2024. [Online]. Available: https://github.com/NathanaelOliver/Vex-Serial.