Few-shot Object Detection for Robotics

David Liu

Mike Callinan

AP Computer Science Principles

22 January 2022

Abstract

In the 2021-2022 FIRST® Tech Challenge (FTC) robotics competition, the match starts with a 30-second autonomous period in which robots operate using pre-programmed codes and sensory inputs. During the autonomous period, alliances earn points by delivering pre-loaded freight to a randomly selected level of the Alliance Shipping Hub (ASH). Also, there is an additional autonomous bonus: a robot that uses the Team Shipping Element (TSE) to detect the correct level and deliver the pre-loaded freight to the correct level of the ASH can earn twenty points. To recognize the randomly placed TSE and determine the corresponding ASH level by using an FTC-compatible onboard camera, automated video object detection is needed. However, automatic detection of a newly designed TSE from video stream is challenging, and the publicly available FTC Object Detection API based on Google TensorFlow does not offer an existing solution. To solve this problem, we found a solution using few-shot object detection. The basic idea of few-shot object detection is that it can detect new objects when there are only a few training samples with annotated bounding box information. Experimental results showed the excellent performance of the proposed few-shot method in robotics object detection. The implementation codes, data, and documents used in this work have been released on GitHub.

Keywords

Robotics, Object Detection, TensorFlow, Few-shot Learning

Introduction

During the autonomous period, as mentioned before, alliances earn points by delivering pre-loaded freight to a randomly selected level of the Alliance Shipping Hub (ASH). Also, there is an autonomous bonus, where if a robot uses the Team Shipping Element (TSE) to detect the correct level and delivers the pre-loaded freight to the correct level of the ASH, it earns twenty points. More specifically, the onboard camera needs to localize the TSE in the scene image and then determine which spot it is randomly placed. Using this information, the robot can decide its action plan to deliver the pre-loaded freight to the correct level of the ASH. Figure 1a provides an overall illustration of this challenging task of TSE localization and ASH level determination.

The FTC object detection API offers a TensorFlow Lite based solution to detect game elements (e.g., ducks and cubes) for the 2021-2022 Freight Frenzy challenge [2]. This FTC solution relies on Google's TensorFlow deep learning technology that is designed to run on mobile devices such as an Android smartphone or Rev Robotics control hub. Although the FTC object detection API has reasonably good performance in detecting those pre-designed game elements, as shown in Figure 1b, it cannot be readily applied to detect any user-designed TSE, as shown in Figure 1c. Instead, different TensorFlow Lite models should be defined and developed for user's specific TSEs. A major challenge in developing such user-specific TSE detection model is that there are very few example images available. Therefore, the traditional approach of training deep learning models for object detection that need many training samples will not be applicable in this robotics scenario.

In the computer vision (CV) and artificial intelligence (AI) fields, researchers have developed an innovative and effective few-shot learning approach for object detection [3-6]. The basic idea of few-shot object detection is that a backbone deep neural network model is leveraged and fine-tuned by learning from a few samples such that the last few layers of the model can be transferred and continually trained for the specific new category of object. Since its invention in [3], few-shot object detection has been extensively used in the CV and AI fields, and a variety of related open-source software packages are available [4-6]. In our FTC robotics project, we adopted the few-shot object detector based on the RetinaNet backbone pretrained on COCO checkpoint [13, 14] to detect the TSE (Figure 1c), which determines the ASH level in autonomous mode.

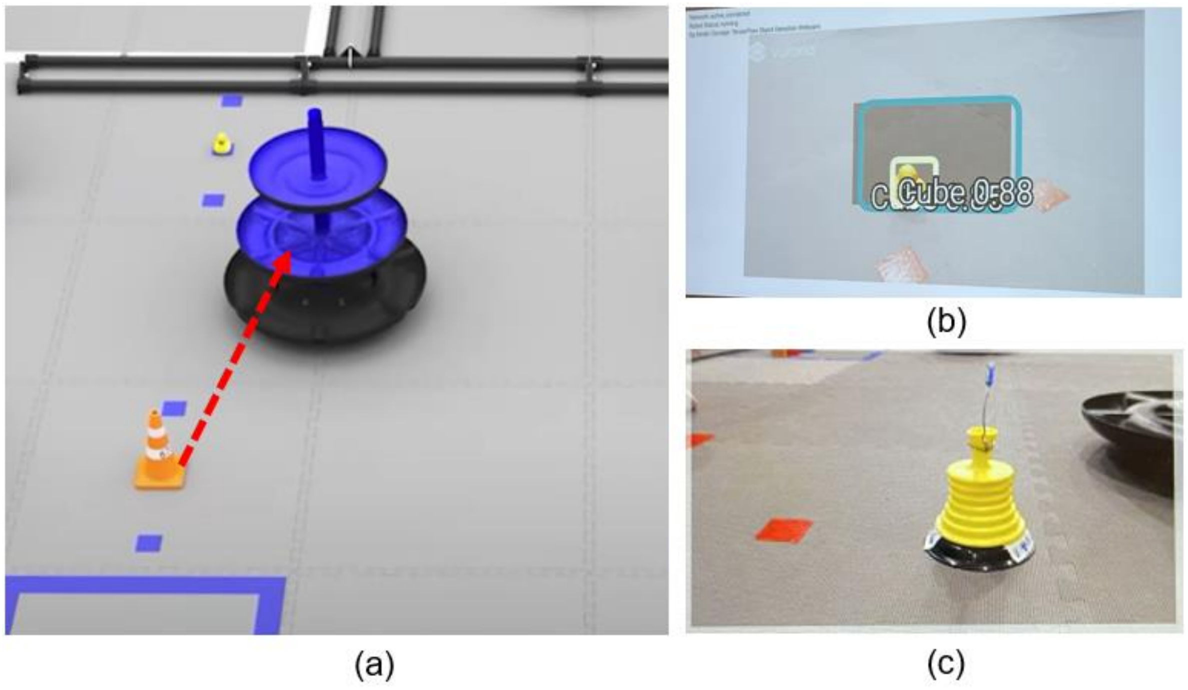


Figure 1. (a) Illustration of the challenging task of TSE localization and ASH level determination. For instance, if the TSE (the cone here) is randomly placed in the middle spot, the pre-loaded freight should be delivered to the second level of ASH. If the TSE is randomly placed in the left spot or right spot, the pre-loaded freight will then be delivered to the first or third level of the ASH. (b) An example of FTC TensorFlow object detection result. The duck object is detected by the TensorFlow Lite model provided by FTC. (c) The TSE (yellow hat) that we made to replace the duck.

Methods

Convolutional neural networks (CNN)

In the recent decade, deep learning has attracted much attention in the fields of machine learning and data mining [7-15], and it has been proven that the deep learning approach is excellent at learning high-level and mid-level features from low-level raw data. A deep learning architecture usually consists of deep network layers by stacking multiple similar building blocks [7]. The bottom layer receives input and then passes the transformed versions of the input to the next layer, all the way up to the top layer. As a result, the architecture of a deep learning model acts as a hierarchical feature extractor. A very popular network model in deep learning is convolutional neural network (CNN) [16], and plenty of recent studies in the deep learning field have demonstrated that CNN has superior spatial pattern representation ability, as shown in many image processing and computer vision tasks such as classification, recognition, detection, segmentation, and registration. In particular, the availability of open-source packages such as TensorFlow [17] and PyTorch [18] have dramatically advanced all aspects of development and application of deep learning and CNN in many domains, including robotics. Fortunately, FTC provided TensorFlow support and open-source software tools to integrate webcam, object detection, and autonomous navigation. This helped significantly in developing our own few-shot object detection model for TSE localization and ASH level determination.

CNN-based object detection

CNNs have been widely used for object detection [7]. The use of CNNs for object detection can be categorized into two groups: one-stage and two-stage CNN architectures. One-stage methods include region-based CNN (RCNN) families [8] such as Fast RCNN [9] and Faster RCNN [10], and two-stage methods include the you-only-look-once (YOLO) family [11] and the single-shot detector (SSD) framework [12]. A critical issue, however, has been widely recognized for these one-stage models: an imbalance in the number of object and background regions. In response, a focal loss function has been proposed in the RetinaNet [13, 14] to penalize inaccurately detected samples in the improved one-stage framework. It has been shown in many studies that the RetinaNet is highly effective [3, 4].

More specifically, RetinaNet [13, 14] is a single, unified network composed of a backbone network and two task-specific subnetworks. The backbone network is responsible for computing a convolutional feature map over an entire input image and is an off-the-shelf convolutional network, e.g., ResNet and Feature Pyramid Network (FPN) (ResNet-50-FPN), used in many literary works [4]. The first subnet performs convolutional object classification on the backbone's output, which is where the classification subnet predicts the probability of object presence (TSE in this work) at each spatial position. This subnet is a small, fully convolutional network (FCN) attached to each FPN level. The second subnet performs convolutional bounding box regression. The design of the box regression subnet is identical to the classification subnet except that it terminates in linear outputs per spatial location.

Few-shot object detection

Few-shot learning, such as few-shot image classification and few-shot object detection, has received increasing attention and witnessed significant advances in recent years [3, 4]. The basic idea of few-shot learning is that it can classify or detect new object when there are only a few training samples with annotated information. The main methodological innovation of few-shot learning is that it typically fine-tunes the last layer of existing CNN models (such as image classifier or object detector) on new datasets, as illustrated in Figure 2. This simple yet effective approach outperforms the traditional meta-learning methods by a significant margin on current benchmarks [3, 4].

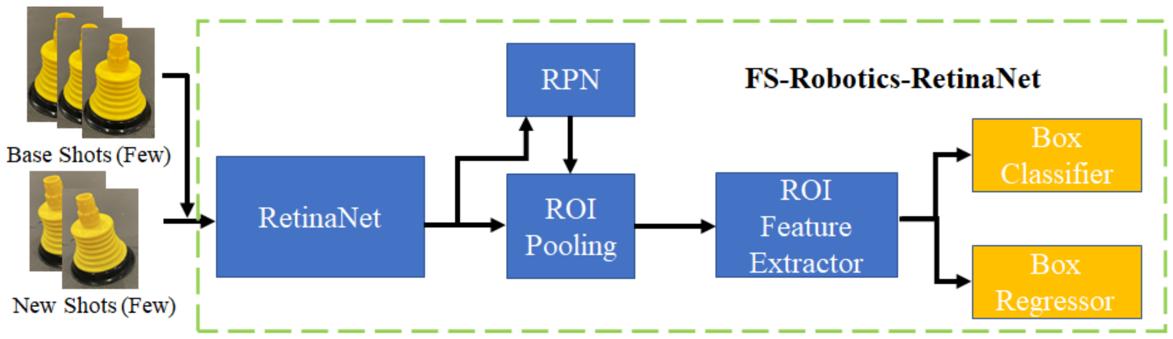


Figure 2. Illustration of the few-shot robotics object detection model of FS-Robotics-RetinaNet based on RetinaNet. Here, the RetinaNet is used as the backbone network, and the region proposal network (RPN), region of interest (ROI) pooling, and ROI feature extractors are used to extract features for bounding box classification and regression [3, 4].

Our FTC robot was designed to enable and facilitate both autonomous and manual driving modes, as shown in Figure 3a. For the autonomous mode, a webcam is installed in the front frame and connected to the Rev Robotics control hub. We employed the few-shot object detector based on the RetinaNet backbone [13, 14] pretrained on the COCO checkpoint to detect the TSE to determine the shipping hub level in autonomous mode, as shown in Figure 3b and Figure 3c. Here, 5 example images of TSE were taken from different views and used as the few-shot training samples (Figure 3b), and the trained model was then deployed using the TensorFlow object detector API on the Rev Robotics control hub. Afterwards, the detected TSE, illustrated in Figure 3c, will be measured by bounding box dimensions provided by the TensorFlow Object Detector (TFOD) API [21] and used to determine the ASH level. Afterwards, the determined ASH level is used to decide the robot's action plan to deliver the pre-loaded freight to the correct alliance shipping hub, as demonstrated in Figure 1a.

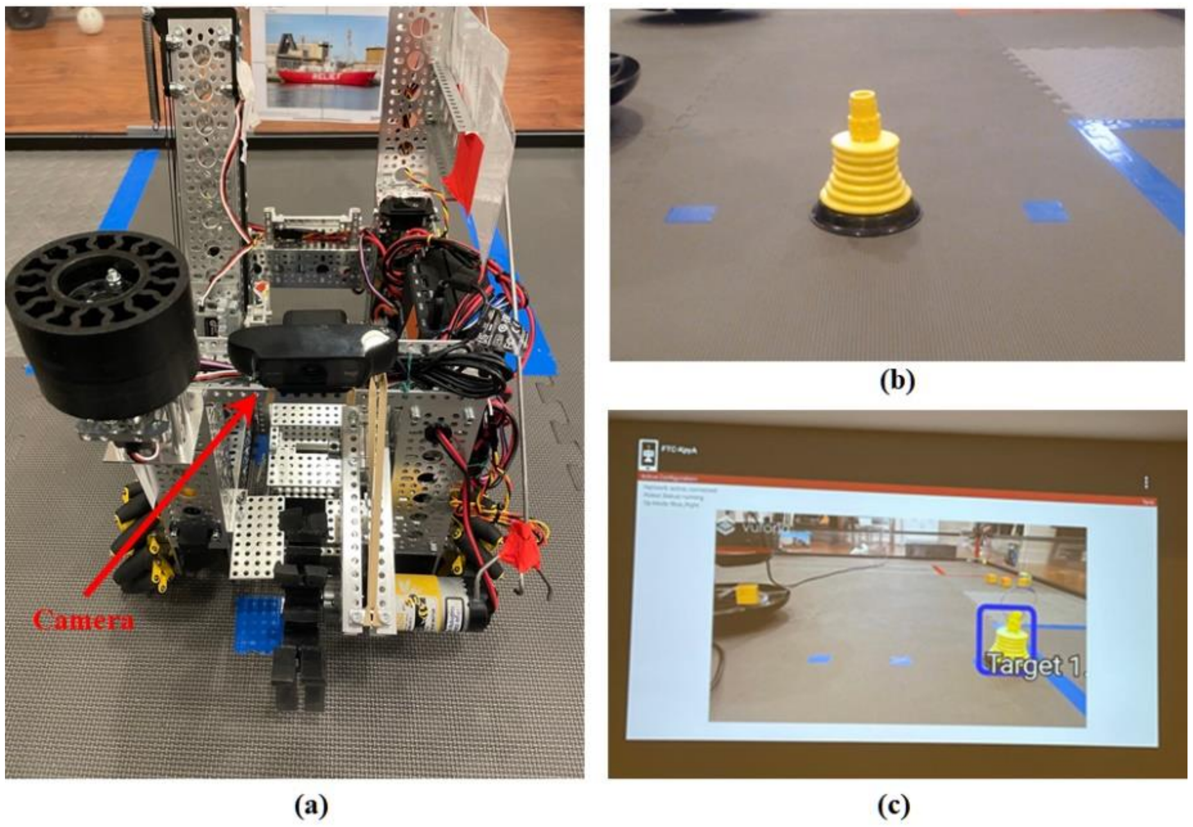


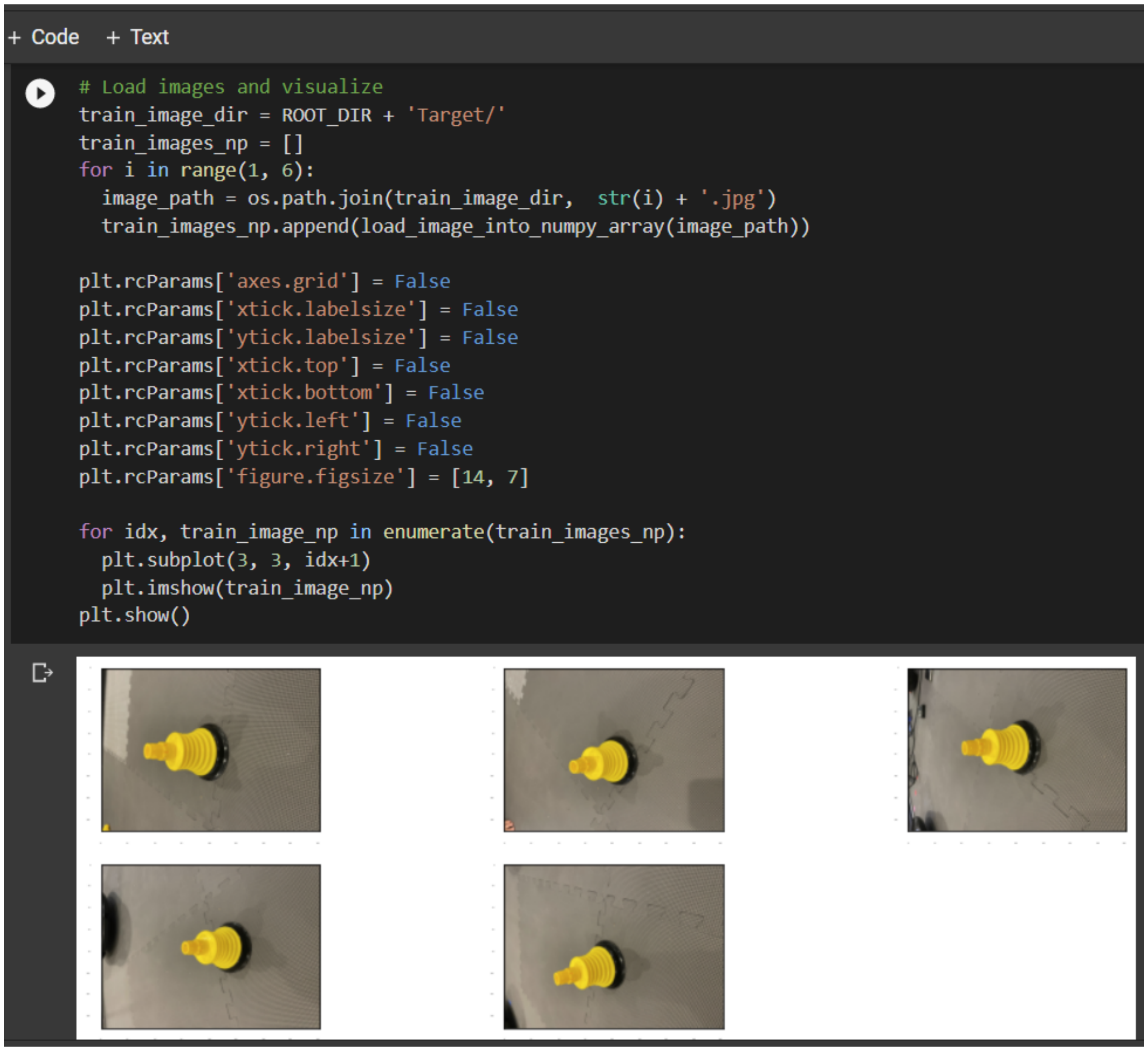
Figure 3. Illustration of few-shot TSE object detection used in our FTC robotics project. (a) Our FTC robot and its mounted Logitech camera for object detection. (b) An example of training image of TSE (the yellow hat). (c) An example of the detected yellow hat TSE through the onboard robot webcam.

Implementation

The implementation codes, data, and documents used in this work are released at GitHub [20]. In the Colab environment, all datasets and codes are stored and run on the Google cloud environment including Google drive, cloud GPU and TensorFlow. Conveniently, all of these tools and environments are free to people once they have a Gmail account. We had a very good learning experience in coding with Colab, as everything is in the Jupyter Notebook style and interactive. One of the most prominent features of Colab to us is that we do not need to worry about the complex and time-consuming configuration steps of setting up an environment for developing and using deep learning models such as those based on TensorFlow. The environment in Colab is also naturally compatible with the FTC

TensorFlow object detection API, meaning that all models developed in Colab can be easily converted to the FTC's TensorFlow Lite environment supported by Rev Robotics control hub.

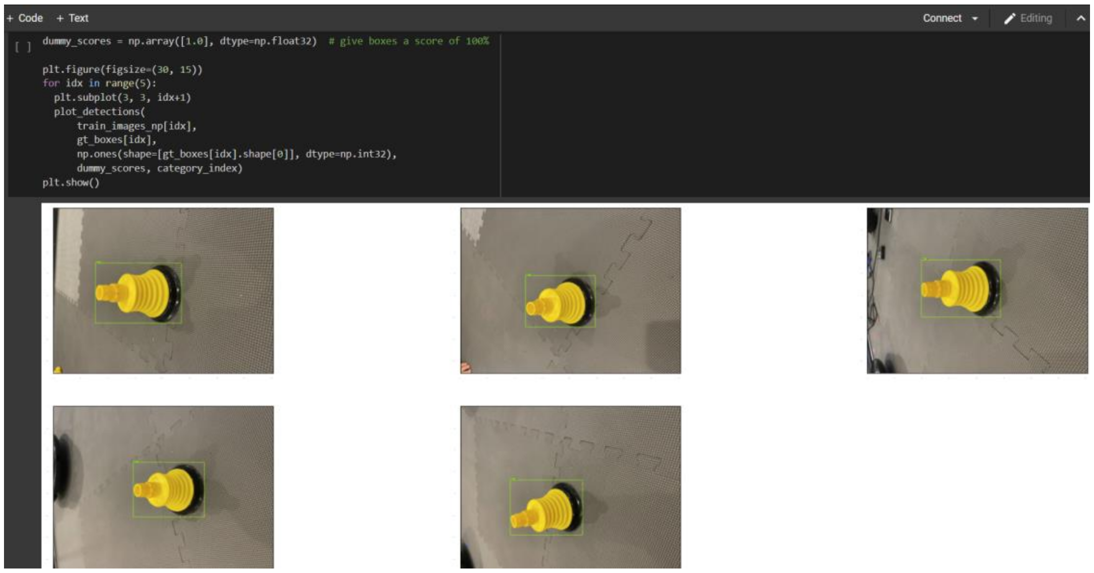
We would like to acknowledge that our few-shot object detection Colab was largely dependent on the RetinaNet [13, 14]. Our main work in this project is to add our own images of the TSE to the existing Colab and retrain the few-shot object detection model. Here, we included a few important steps we used. The codes that add our own images are below in Figure 4.



```
# Load images and visualize
train_image_dir = ROOT_DIR + 'Target/'
train_images_np = []
for i in range(1, 6):
  image_path = os.path.join(train_image_dir,  str(i) + '.jpg')
  train_images_np.append(load_image_into_numpy_array(image_path))

plt.rcParams['axes.grid'] = False
plt.rcParams['xtick.labelsize'] = False
plt.rcParams['ytick.labelsize'] = False
plt.rcParams['xtick.top'] = False
plt.rcParams['xtick.bottom'] = False
plt.rcParams['ytick.left'] = False
plt.rcParams['ytick.right'] = False
plt.rcParams['figure.figsize'] = [14, 7]

for idx, train_image_np in enumerate(train_images_np):
  plt.subplot(3, 3, idx+1)
  plt.imshow(train_image_np)
plt.show()
```

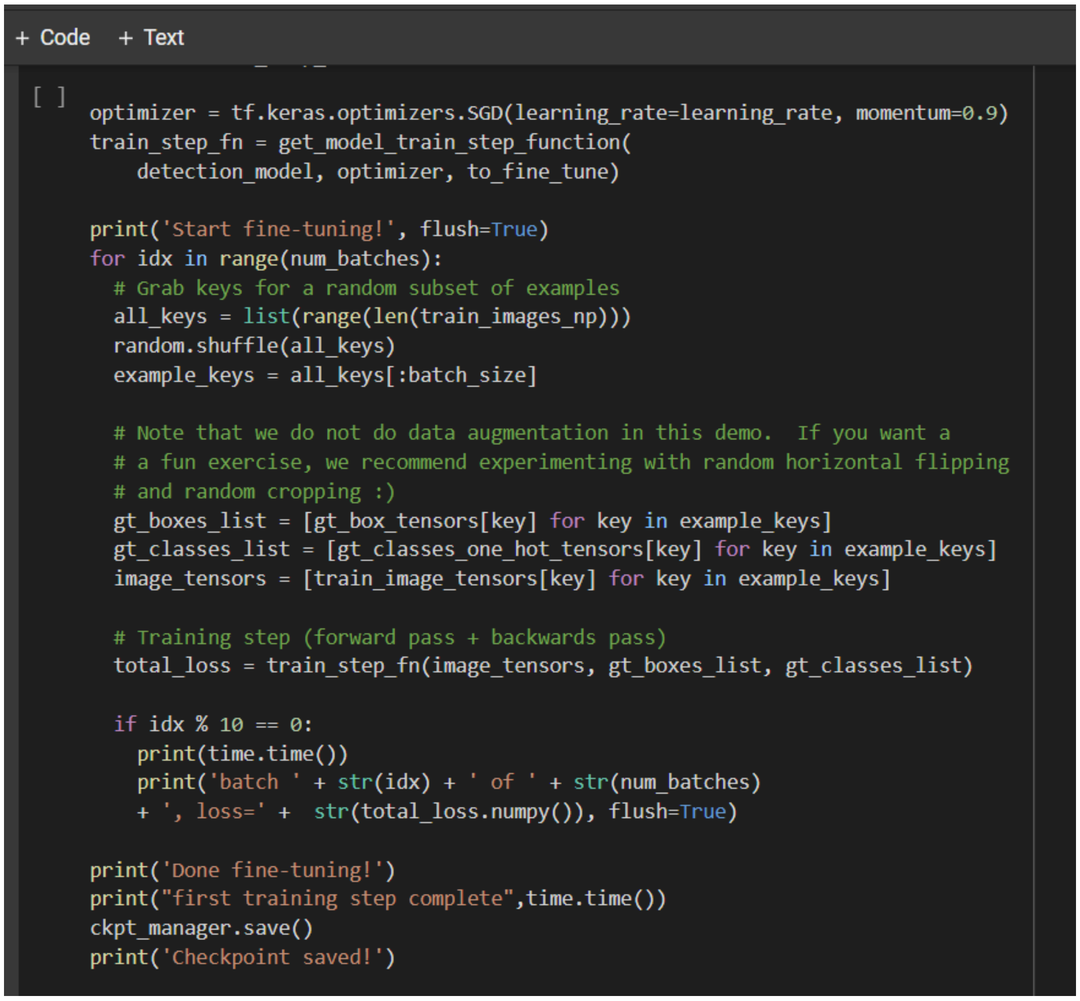
Figure 4. The Colab codes that are used to add our own TSE images (five ones at the bottom).

After the TSE images are added into the model, the next key step is to manually annotate the target bounding box in each image. This step is important for the TensorFlow object detection model to train the specific model for TSE. Therefore, this step is user-specific, and each FTC team can develop their own TSE and train their own TensorFlow object detection model, which will be deployed when it is uploaded to the Rev Robotics control hub. As examples, Figure 5 shows the code and annotated bounding boxes of the TSE images.



```
dummy_scores = np.array([1.0], dtype=np.float32)  # give boxes a score of 100%

plt.figure(figsize=(30, 15))
for idx in range(5):
  plt.subplot(3, 3, idx+1)
  plot_detections(
      train_images_np[idx],
      gt_boxes[idx],
      np.ones(shape=[gt_boxes[idx].shape[0]], dtype=np.int32),
      dummy_scores, category_index)
plt.show()
```

Figure 5. Illustration of the manually annotated bounding boxes of the TSE target (bottom images).



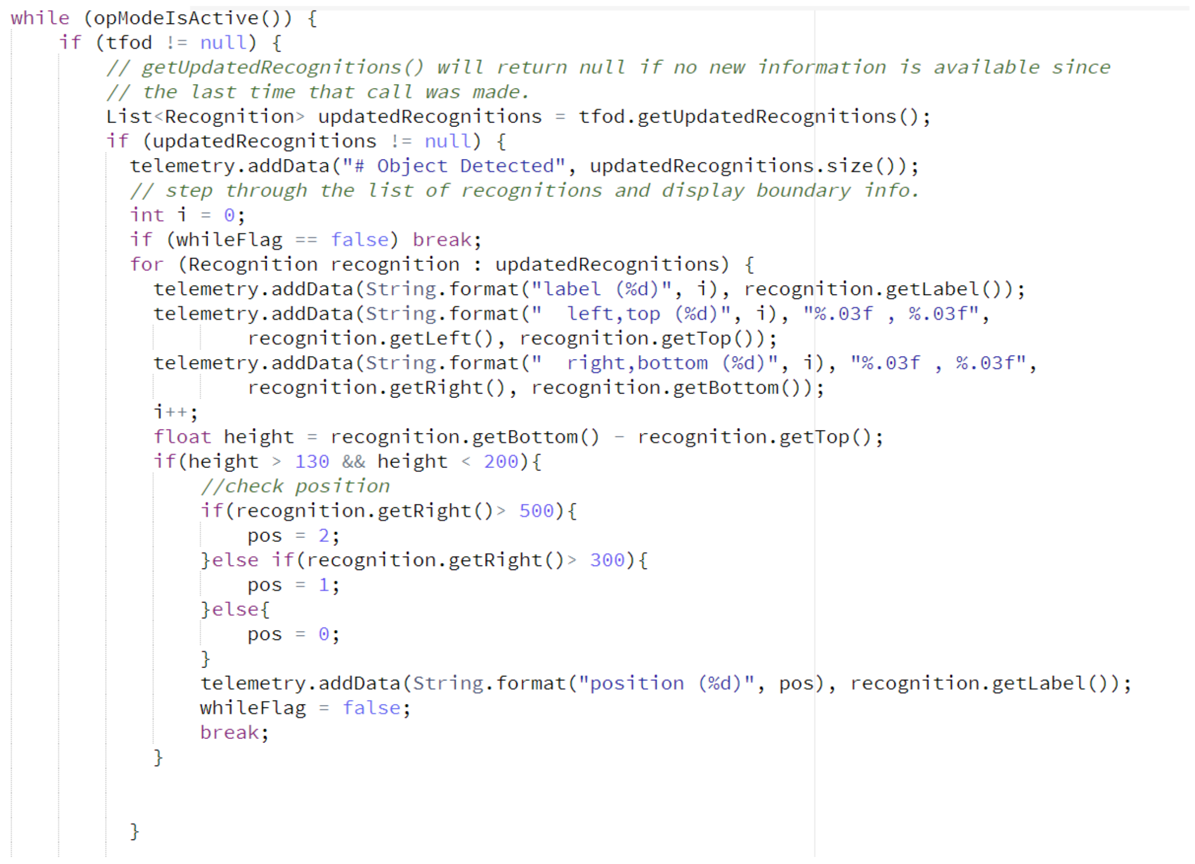
```
optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
train_step_fn = get_model_train_step_function(
    detection_model, optimizer, to_fine_tune)

print('Start fine-tuning!', flush=True)
for idx in range(num_batches):
  # Grab keys for a random subset of examples
  all_keys = list(range(len(train_images_np)))
  random.shuffle(all_keys)
  example_keys = all_keys[:batch_size]

  # Note that we do not do data augmentation in this demo.  If you want a
  # a fun exercise, we recommend experimenting with random horizontal flipping
  # and random cropping :)
  gt_boxes_list = [gt_box_tensors[key] for key in example_keys]
  gt_classes_list = [gt_classes_one_hot_tensors[key] for key in example_keys]
  image_tensors = [train_image_tensors[key] for key in example_keys]

  # Training step (forward pass + backwards pass)
  total_loss = train_step_fn(image_tensors, gt_boxes_list, gt_classes_list)

  if idx % 10 == 0:
    print(time.time())
    print('batch ' + str(idx) + ' of ' + str(num_batches)
    + ', loss=' +  str(total_loss.numpy()), flush=True)

print('Done fine-tuning!')
print("first training step complete",time.time())
ckpt_manager.save()
print('Checkpoint saved!')
```

Figure 6. The Colab codes used to fine-tune the general TensorFlow object detection model to specific TSE model by few-shot learning.

Once the training samples for object detection model training are available, the few-shot learning module will start to fine-tune pre-trained backbone models, e.g., RetinaNet for object detection [13, 14], to transfer the generic model to a specific model for TSE. Figure 6 shows the codes that perform the model fine-tuning step for TSE model generation. This is probably one of the most important pieces of codes for few-shot object detection in this FTC robotics project's autonomous mode. Once the few-shot TSE model was trained and generated, it will be converted into the TensorFlow Lite format and uploaded to the Rev Robotics control hub for deployment.

Robotics integration

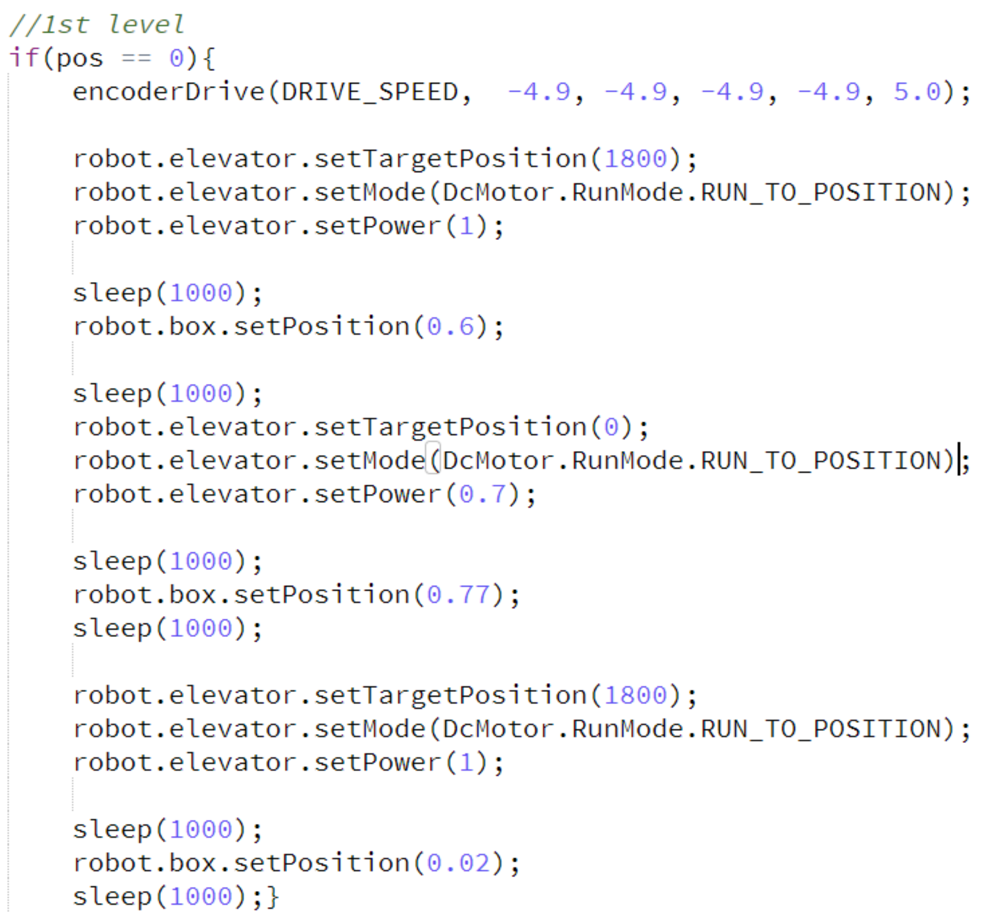
The FTC TFOD library [21] provides an excellent support for integrating webcam, TensorFlow object detection model, and other related modules. There is an abundance of online resources to learn TFOD library and its integration and application in FTC robotics, such as initialization of TFOD and detection of FTC game elements [21]. Here, we focus on the introduction of how our specific few-shot TSE detection model was employed and integrated into our robot (Figure 3a) to accomplish the task of delivering the pre-loaded freight to the correct level of the ASH (Figure 1a). The codes used to determine the ASH level based on the detected TSE bounding box are shown in Figure 7. It is important to point out that the key parameters used to make such determination will be highly dependent on the specific robot figuration, the webcam installation, and their geometric relationships. All the optimal parameters in our robotics project were empirically decided by extensive experiments in the robotics competition field.



```
while (opModeIsActive()) {
    if (tfod != null) {
        // getUpdatedRecognitions() will return null if no new information is available since
        // the last time that call was made.
        List<Recognition> updatedRecognitions = tfod.getUpdatedRecognitions();
        if (updatedRecognitions != null) {
          telemetry.addData("# Object Detected", updatedRecognitions.size());
          // step through the list of recognitions and display boundary info.
          int i = 0;
          if (whileFlag == false) break;
          for (Recognition recognition : updatedRecognitions) {
            telemetry.addData(String.format("label (%d)", i), recognition.getLabel());
            telemetry.addData(String.format("  left,top (%d)", i), "%.03f , %.03f",
                    recognition.getLeft(), recognition.getTop());
            telemetry.addData(String.format("  right,bottom (%d)", i), "%.03f , %.03f",
                    recognition.getRight(), recognition.getBottom());
            i++;
            float height = recognition.getBottom() - recognition.getTop();
            if(height > 130 && height < 200){
                //check position
                if(recognition.getRight()> 500){
                    pos = 2;
                }else if(recognition.getRight()> 300){
                    pos = 1;
                }else{
                    pos = 0;
                }
                telemetry.addData(String.format("position (%d)", pos), recognition.getLabel());
                whileFlag = false;
                break;
            }


          }
```

Figure 7. The Java codes for determining ASH level based on the bounding box of the detected TSE object in the webcam video scene.

Once the ASH level is determined by live webcam video stream, the robot needs to make an action plan to deliver the pre-loaded freight to the correct level of the ASH. The codes in Figure 8 show how the robot plans its motion to conduct such delivery. This piece of code is also very dependent on the robot's configuration, and more significantly, its robotic delivery arm. Additional details of the integration of webcam, few-shot TSE object detection, ASH level determination, pre-loaded freight delivery, and other related robot operations are referred to our open-source codes on GitHub [20].



```java
//1st level
if(pos == 0){
    encoderDrive(DRIVE_SPEED,  -4.9, -4.9, -4.9, -4.9, 5.0);

    robot.elevator.setTargetPosition(1800);
    robot.elevator.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    robot.elevator.setPower(1);

    sleep(1000);
    robot.box.setPosition(0.6);

    sleep(1000);
    robot.elevator.setTargetPosition(0);
    robot.elevator.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    robot.elevator.setPower(0.7);

    sleep(1000);
    robot.box.setPosition(0.77);
    sleep(1000);

    robot.elevator.setTargetPosition(1800);
    robot.elevator.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    robot.elevator.setPower(1);

    sleep(1000);
    robot.box.setPosition(0.02);
    sleep(1000);}
```

Figure 8. The Java codes for our robot's motion plan to deliver the pre-loaded freight to the correct level of the ASH [20]. Here, the situation of first ASH level is used as an example. The Java codes for other ASH levels are referred to our GitHub link.

Evaluation

To visualize and quantitatively evaluate the accuracy and performance of the developed few-shot TSE object detection model FS-Robotics-RetinaNet, we used a large-screen projector to see the results, as shown in Figure 9. The Rev Robotics control hub has a USB port to connect the projector to the hub via HDMI cable. In this way, the screen on the Android driver hub cell phone can be effectively visualized. Figure 9 shows an example of the visualization via large-screen projector. Many experiments were performed and visualized in this way, and our statistical result showed that the TSE detection accuracy is 100%, showing the effectiveness and accuracy of our developed FS-Robotics-RetinaNet model. Also, in the real robotics practices and competitions, the success rates of ASH level determination and pre-loaded freight delivery are very high, which is approximately 95%. Those unsuccessful cases were mainly due to the variation of the robot's initial location and the ASH's location before the autonomous mode competition in the field. In each robotics match, the locations of both the robot and the ASH are reset and thus are slightly different, and such variation results in the uncertainties to the hard-coded key parameters in the robot's motion plan, as shown in Figure 8 and other code segments in GitHub [20]. To solve these uncertainty problems in robotics, other advanced robotics and AI technologies, such as simultaneous localization and mapping (SLAM) [22] and integration of multimodal sensory data such as camera and LiDAR, are needed to have more accurate robot/environment localization and mapping in the future.

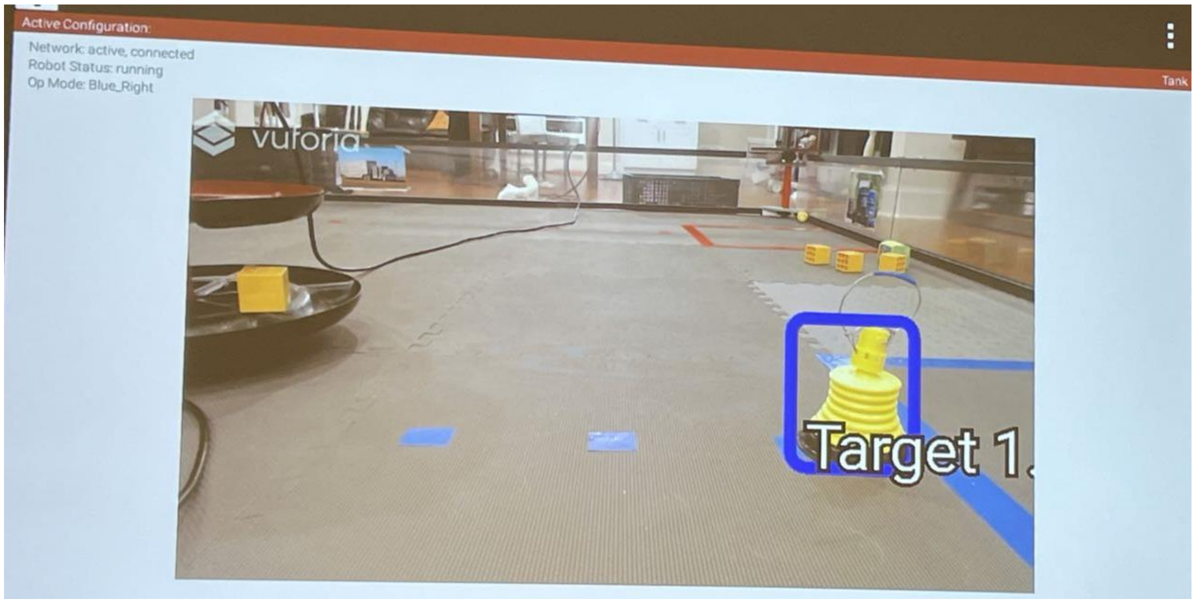


Figure 9. Visualization of few-shot TSE object detection result by a large-screen projector.

Conclusion and Discussion

In this work, we developed a few-shot object detection model FS-Robotics-RetinaNet to recognize the randomly placed TSE and determine the corresponding ASH level by using an FTC-compatible onboard webcam. Our work is built on top of the few-shot object detector based on the RetinaNet backbone [13, 14] and the FTC TensorFlow object detection library [21]. Experimental results demonstrated the excellent performance of the proposed few-shot object detection method for FTC robotics. We have released the few-shot object detection codes and data and the robot's autonomous mode codes via a GitHub link [20].

Notably, we explored various methods of video object detection during this FTC robotics project, such as using the FTC-compatible EasyOpenCV library [19] and training a TSE object detection TensorFlow model from scratch. However, due to the various factors of uncertainties and variations such robot-TSE distance, webcam/TSE viewing angles, room lighting, surrounding objects, among others, it has been challenging to develop a robust object detection model that works well in a real-world dynamic environment. Our experience is that the proposed few-shot object detection model FS-Robotics-RetinaNet works significantly better than other explored methods. We hope this FS-Robotics-RetinaNet model with open-sources [20] can be useful for other FTC teams to explore and evaluate in their specific FTC robotics application scenarios and beyond.

Acknowledgements

We would like to thank Professor Charlie Li, Dr. Javier Rodriguez, Daniel Petti, Haixing Dai, Lin Zhao, Zihao Wu, Zhengliang Liu, and Tianming Liu for their guidance and help on robotics, deep learning, image processing, TensorFlow, and Google Colab. We are grateful to the developers of FTC TensorFlow object detection library and RetinaNet based few-shot learning framework for sharing their great tools. David Liu worked on this few-shot object detection model for robotics as his AP Computer Science Principles course project under the guidance of Mike Callinan at Athens Academy.

References

1. https://www.firstinspires.org/sites/default/files/uploads/resource_library/ftc/game-manual-part-1-traditional-events.pdf.
2. https://github-wiki-see.page/m/FIRST-Tech-Challenge/FtcRobotController/wiki/Using-TensorFlow-in-Freight-Frenzy
3. Xin Wang, Thomas E. Huang, Trevor Darrell, Joseph E. Gonzalez, Fisher Yu, Frustratingly Simple Few-Shot Object Detection, Proceedings of ICML, Vienna, Austria, PMLR 119, 2020.
4. Wenbin Li, et al., LibFewShot: A Comprehensive Library for Few-shot Learning, https://arxiv.org/abs/2109.04898, 2021.
5. Bingyi Kang, Zhuang Liu, Xin Wang, Fisher Yu, Jiashi Feng, Trevor Darrell, Few-shot Object Detection via Feature Reweighting, ICCV 2019.
6. Qi Fan, Wei Zhuo, Chi-Keung Tang, Yu-Wing Tai, Few-Shot Object Detection with Attention-RPN and Multi-Relation Detector, CVPR 2020.
7. Yu Jiang, Changying Li, Convolutional Neural Networks for Image-Based High-Throughput Plant Phenotyping: A Review, Plant Phenomics, 4152816, 2020.
8. R. Girshick, J. Donahue, T. Darrell, and J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, in 2014 IEEE CVPR, Columbus, OH, USA, 2014.
9. R. Girshick, Fast R-CNN, in 2015 IEEE ICCV, Santiago, Chile, 2015.
10. S. Ren, K. He, R. Girshick, and J. Sun, Faster R-CNN: towards real-time object detection with region proposal networks, in Advances in Neural Information Processing Systems 28, Montréal, Canada, 2015.
11. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, You only look once: unified, real-time object detection, in 2016 IEEE CVPR, Las Vegas, NV, USA, 2016.
12. W. Liu, D. Anguelov, D. Erhan et al., SSD: Single shot multibox detector, in Computer Vision – ECCV 2016. Lecture Notes in Computer Science, vol 9905, Springer, Cham, 2016.
13. T. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, Focal Loss for Dense Object Detection, in IEEE PAME, vol. 42, no. 2, pp. 318-327, 1 Feb. 2020.
14. T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, Focal loss for dense object detection, in 2017 ICCV, Venice, Italy, 2017.
15. Bengio, Y., Courville, A., Vincent, P.: Representation Learning: A Review and New Perspectives. IEEE Trans. Pattern Anal. Mach. Intell. 35, 1798–1828 (2012).
16. Krizhevsky A, Sutskever I, and Hinton GE, “ImageNet Classification with Deep Convolutional Neural Networks,” in NIPS, 2012, pp. 1097–1105.
17. https://www.tensorflow.org/
18. https://pytorch.org/
19. https://github.com/OpenFTC/EasyOpenCV
20. https://github.com/davidweizhongliu/Robotics-Few-shot-Object-Detection
21. https://github.com/google/ftc-object-detection/tree/master/TFObjectDetector
22. https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

Authors

David W Liu is a 10th grader in Athens Academy, Athens, GA. He is interested in robotics, artificial intelligence, computer programming, and mathematics. He has been taking robotics courses in Athens Academy since 6th grade and has been actively involved in multiple robotics activities such as FLL and FTC challenges since 6th grade. He has served as the captain of robotics teams at Athens Academy since 7th grade. He has been a research intern with the Bio-Sensing and Instrumentation Lab, Phenomics and Plant Robotics Center, College of Engineering, University of Georgia (UGA) since June 2021 under the mentorship of Prof. Charlie Li.

Mike Callinan has taught at Athens Academy for 22 years serving as the Instructional Technology Coordinator and Robotics and Media Production Coordinator. He supports faculty and students in grades 1-12. Starting from 2002, Athens Academy Sparbots program has grown to 10 teams, 3 FLL

Jr. teams in 1st and 2nd grades, 4 FLL teams in grades 4-7, and 3 FTC teams in grades 8-12. Media Production includes establishing the first webcast of a high school sports event 20 years ago. He is part of the Georgia FLL Tournament Director's committee serving as a judge, judge advisor, and tournament director for FLL Regional, Super-Regional, and State Championships. Mr. Callinan also works with the Georgia FTC Planning Committee as a judge, league director of the Caterpillar League, and coach of 3 FTC teams. He received the Athens Award from the Athens Classic Center for bringing millions of dollars of revenue to host the 72-team, FTC South Super-Regional with teams from 14 states in the Southeast. Before becoming a teacher, Mr. Callinan obtained a B.S. in Computer Science and an M.Ed. in Mathematics Education. His previous career included being a Systems Analyst, Technology Director, and General Manager at the Central Intelligence Agency, serving in over 40 countries.