# JayRadar: A Zero-Code and Coprocessor Agnostic Robotics Vision System for the FIRST Robotics Competition

Steven Klinefelter, Nathaniel Brightup, Peilong Li
Department of Computer Science
*Elizabethtown College*
{klinefelters, brightupn, lip}@etown.edu

***Abstract*—Vision systems have become a common feature in robotics competitions, facilitating autonomous object interaction. While object detection using AI has been explored, the availability of open-source solutions remains limited. This paper introduces JayRadar, an open-source vision system designed to detect objects using AI. By leveraging machine learning techniques and fine-tuning a custom-trained YOLO model, JayRadar aims to provide a user-friendly platform for object detection. The emphasis is on democratizing AI by lowering the barriers to entry, enabling even novice programmers to customize and optimize the system. JayRadar fills a crucial gap in the robotics community, offering an open-source and widely applicable vision system for object detection. The paper details the development methodology, highlighting the open-source nature of JayRadar and its potential impact on robotics competitions, research, and innovation. Experimental results and the availability of the JayRadar code-base in the public domain further demonstrate the system's potential in advancing the field.**

***Index Terms*—Machine Learning; You Only Look Once (YOLO); Feature Engineering and Analysis; Spark Streaming; OneDNN**

## I. Introduction

Object detection plays a pivotal role in various computer vision applications, and its implementation has helped define the domain of robotics. In the context of the First Robotics Competition (FRC), where teams design, build, and program robots to compete in high-intensity challenges, object detection has become a critical component for robot perception and decision-making. YOLO (You Only Look Once) is the leading real-time object detection framework, known for its unprecedented speed and accuracy. This paper presents the development and features of a system specifically tailored for FRC teams utilizing YOLO object detection and aimed at enhancing user interaction and maximizing the potential of the underlying YOLO framework in the context of the FRC challenges.

Understanding the unique requirements and constraints faced by FRC teams, we sought to deliver a streamlined user experience while addressing the specific challenges users would encounter. Careful consideration of the ideal design for usability in an FRC context influenced many decisions; including the focus on unmarked object detection, hardware accessibility, and zero code interfacing in the final design.

Through the development of the software for YOLO object detection tailored to FRC teams, we aim to provide a user-friendly interface that enables teams to leverage the power of YOLO for enhanced robot perception and decision-making. The GUI serves as a valuable tool for FRC teams, simplifying the configuration process and facilitating real-time object detection with YOLO in the high-pressure environment of the FRC challenges. The backend code then ensures that the GUI functions in real-time and gives accurate results for the analysis of the camera feed to both the robot itself and the driver.

In this paper we will explore how the GUI enables teams to configure essential parameters such as confidence threshold, IoU threshold, maximum detections, and image size to suit the requirements of the FRC challenges. Additionally, we will examine the implementation of class filters, screenshot capabilities, and debugging options, all tailored to empower FRC teams in their robot perception and decision-making processes.

The organization of this paper is as follows. Section §II summarizes the current similar work on the market. Section §III depicts the overall design of JayRadar and the vision system. Section §IV evaluates the performance of a variety of hardware and software using vision detection and helps illustrate how such evaluations guided JayRadar's creation in the past, present, and future. Lastly, Section §V concludes the paper.

## II. Related Works

The conventional approach to vision processing in FRC has predominantly relied on the utilization of specific "vision targets," typically in the form of retro-reflective tape. These targets function by reflecting light directly back at the source when illuminated. To exploit this characteristic, popular coprocessors like the GloWorm incorporate bright green LEDs surrounding the camera lens [1]. By reducing the exposure while directing these lights towards the target, the camera can capture an image predominantly black in color, with distinct green vision

TABLE I
VISION PIPELINE FEATURES

| Name | Marked | Unmarked | Processor |
|---|---|---|---|
| Limelight | Yes | Yes | Limelight |
| PhotonVision | Yes | No | ARM Based |
| JayRadar | No | Yes | ARM Based |

targets. Subsequently, additional techniques such as color thresholding and contour filtering are applied to enhance detection accuracy [2].

More recently, FRC competitions have introduced AprilTags as an alternative form of "vision targets." Resembling simplified QR codes, these tags can be utilized in a similar manner to retro-reflective tape. With readily available libraries in popular programming languages, detecting these tags requires minimal code implementation. PhotonVision, an open-source solution written in Java, is particularly well-known in this regard [3].

However, both retro-reflective tape and AprilTags share a common limitation, namely the dependence on specific markers for object detection. Consequently, teams must rely on the game designers to mark any objects or locations they intend to autonomously interact with. In order to address the challenge of detecting unmarked objects autonomously, the Limelight system has recently incorporated a Neural Network pipeline, leveraging the Google Coral TPU for accelerated performance. With the capability to achieve impressive frame rates of up to 30 frames per second, albeit with limited tuning options, Limelight has remained one of the most popular coprocessors for computer vision. Nevertheless, the scarcity of available Google Coral TPUs has constrained its widespread adoption.[2]

To overcome these limitations, a more versatile Neural Network approach can be employed, such as YOLOv8 [4]. YOLOv8 offers the flexibility of exporting to various popular formats while still running natively, eliminating the requirement for a specific dedicated hardware accelerator like the Google Coral TPU. This allows the same code to be utilized across multiple coprocessors with distinct hardware configurations, providing a more flexible and accessible solution.

## III. DESIGN

In this section, we present the design of our project, outlining the vision pipeline, model training process, and pipeline tuning.

### A. Overview of the JayRadar System

The JayRadar system is designed to enhance the vision processing capabilities of FRC (FIRST Robotics Competition) teams, aiming to improve robot performance and control during competitions. The overall system architecture is illustrated in Fig. 1.

In line with typical FRC vision co-processors, JayRadar relies on a well-established process. The FRC Roborio assumes the crucial role of hosting a network table server where JayRadar, along with other vision processors, can transmit relevant data. This data serves as valuable input for the Roborio, which then employs it to govern the robot's actions and responses [5].

Additionally, the vision pipeline implemented by JayRadar offers the possibility of providing a live video feed to the driver station. This is accomplished through post the video stream to a URL. Alternative approaches involve posting the video stream using an RTSP (Real-Time Streaming Protocol) or utilize the camera server library, thoughtfully provided by WPILib, to streamline the integration of live video into the driver station interface[5].

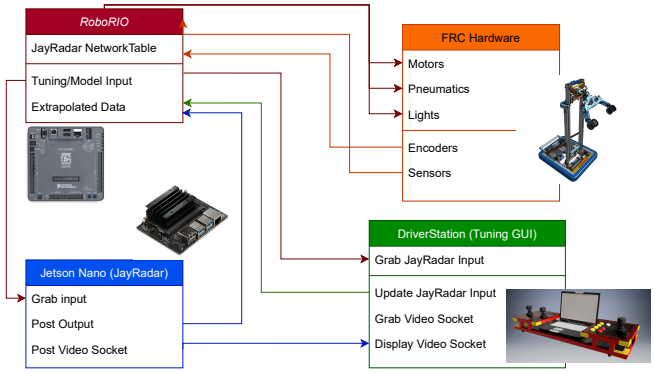


Fig. 1. JayRadar System Diagram

### B. Overview of the Vision Pipeline

The overall pipeline process is illustrated in Fig. 2. Our pipeline is inspired by PhotonVision's description of a pipeline:

> "A vision pipeline represents a series of steps that are used to acquire an image, process it, and analyzing it to find a target. In most FRC games, this means processing an image in order to detect a piece of retroreflective tape or an AprilTag." [3]

The data for the pipeline is sourced from a camera that captures frames and timestamps each frame. These data sources can be customized for individual devices to optimize native hardware capabilities.

The vision pipeline comprises a series of data pipes, each responsible for applying specific transformations to frames and corresponding data. For instance, filters are utilized to adjust the color balance of frames by modifying the red, green, and blue channels. Our project primarily focuses on a YOLOv8-based filter or the Ultralytics library, which leverages pretrained models in the yolov8 architecture. This filter effectively detects objects and includes only a single detection, subsequently adding the data back into the pipeline.[4]

As the data progresses through the various pipes, it ultimately reaches the output stage of the pipeline. Here, the data is posted to the network tables for FIRST Robotics Teams. Additionally, the frames can be locally displayed

or transmitted to a web server from this section of the pipeline.

The advantage of this approach lies in the flexibility and simplicity of the pipeline architecture. Creating a new source, pipe, or output is as straightforward as creating a child of the base class. This standardized template ensures clarity and ease for future implementations of sources, pipes, and outputs.

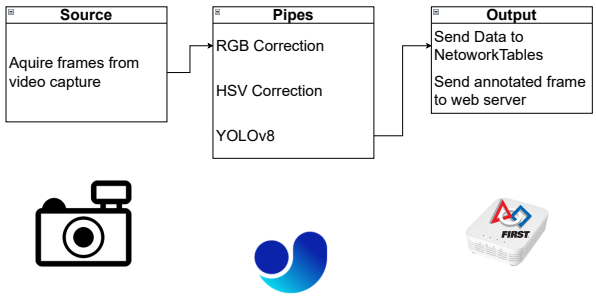


Fig. 2. Overall Vision Pipeline Design

### C. Custom Training

To tailor the object detection capabilities of our pipeline to specific requirements, we introduce a process for retraining a custom YOLOv8 model. Follow these steps:

*1) Data Collection:* Gather a substantial dataset of images for training the custom model. It is recommended to have a dataset with more than a thousand images to ensure robust training and accurate results.

*2) Data Annotation:* Upload the collected images to Roboflow or a similar annotation platform for efficient and accurate data annotation. Properly annotated data is crucial for training an effective model.

*3) Roboflow API Key:* Once the images are annotated, Roboflow will provide the user with an API key. This API key is essential for accessing the annotated data and facilitating model training.

*4) Model Training:* Utilize the annotated data and the provided API key to initiate model training. Google Colab or a similar platform can be used for this purpose. The custom YOLOv8 model will be trained using the annotated data, enabling it to recognize specific objects relevant to the project.

*5) Model Download and Usage:* After successful training, the custom YOLOv8 model can be downloaded and integrated into the vision pipeline. The model's enhanced object detection capabilities will contribute to the accuracy and efficiency of the project's overall functionality.d

### D. Live Pipeline Tuning

To offer users a simple and flexible way to fine-tune the pipeline, we have developed a web-based interface. Initially, the interface was a local Tkinter application, but it has evolved into a more user-friendly web-based version. This interface serves as a display and control center, enabling users to adjust the tuning values of each filter in the pipeline.

The web-based interface provides an intuitive and accessible platform for users to customize their pipeline according to their specific requirements. Users can easily access the interface through a web browser, making it convenient for remote tuning and configuration adjustments.

Each pipe in the pipeline, such as the YOLOv8 filter, comes with a set of configurable tuning values. These values include parameters like the confidence threshold, Intersection Over Union (IOU) threshold, and more. By modifying these values, users can fine-tune the behavior of individual filters, influencing the object detection and transformation outcomes.

The web-based interface streamlines the customization process, allowing users to make real-time adjustments and observe the effects on the pipeline's performance instantly. This ease of use empowers users to tailor the pipeline to suit the unique characteristics of their projects.

The interface facilitates the saving and loading of pipeline configurations through JSON files. Users can save their preferred tuning settings as JSON configurations, enabling quick and convenient switching between different configurations. This feature proves especially valuable when users need to switch between various pretuned setups or share specific configurations with others.

Additionally, the web-based interface grants users the ability to activate or deactivate specific filters within the pipeline. This dynamic control allows for rapid experimentation and comparison of different transformations and detections. Users can assess the impact of each filter on the pipeline's output and adjust the combination to achieve the desired results.

The incorporation of a web-based interface for live pipeline tuning significantly enhances the overall usability and flexibility of our system. By providing users with an intuitive platform to fine-tune filters and save/load configurations, we aim to empower teams to optimize their vision pipelines efficiently and effectively. This adaptive approach ensures that the system remains adaptable to various scenarios and requirements, ultimately contributing to improved performance and user satisfaction.

## IV. Evaluation

This section of the paper evaluates the importance of our research. To discuss the importance of our research, we will first discuss the strategies and controls we used to collect data throughout the course of our testing. We will then cover the data gathered and how it impacts future research.

### A. Data Collection

We took measurements with consistent parameters achieved via the following considerations: reducing reflections, maintaining consistent lighting, prioritizing consistent distance and angle from camera, and following identical parameters for initializing and measuring data from the testing program.

### B. YOLO Speeds

We collected our first set of data analyzing YOLO itself and the effects on accuracy incurred by distance from the camera using consistent hardware and software. We tested on a Jetson Nano running Ubuntu 18.04.6 and a Jetson Xavier NX running Ubuntu 20.04, their default operating systems. We then followed up with a test of the Jetson Nano running on Ubuntu 20.04 for controlled comparison.

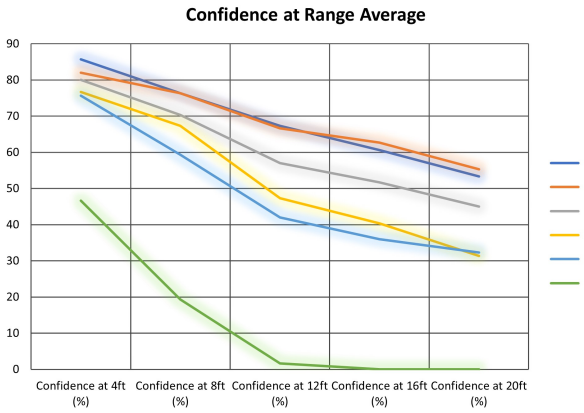


Fig. 3. Confidence values captured at specific ranges.

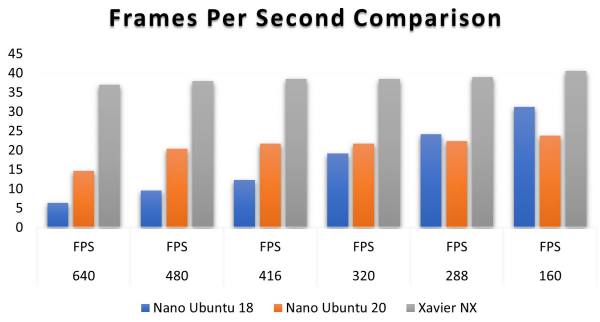


Fig. 4. Frames per second for varying hardware and software.

From this data we can draw two conclusions. The first is that the higher image sizes caused a reduction in frame rate but an increase in both confidence and range. The second is that both the hardware and the software have a significant impact on performance. While the Xavier NX performed better than the Nano on all image sizes (proving the impact of hardware), the Nano performed vastly better on larger image sizes when using a newer operating system. While it performed slightly worse at low image sizes while running Ubuntu 20, it more than doubled Ubuntu 18's performance when running the larger and more standard image sizes. This led us to conclude that it would be the best operating system to use moving forward.

### C. Model Comparison

In the process of our testing to determine the hardware accessibility goal of our project, we tested a variety of different boards using the models they were best suited for. We used a standard program and tested the average delay and standard deviation while it ran the same images through the respective models for each board.

| Device Name | Average Delay (ms) | Std. Deviation |
|---|---|---|
| Raspberry Pi 4 Pytorch | 955.14189 | 23.55688 |
| Raspberry Pi 4 Onnx | 660.84042 | 72.32858 |
| Raspberry Pi 4 NCNN | 559.34517 | 15.94227 |
| Orange Pi 5 Pytorch | 406.78961 | 20.73615 |
| Orange Pi 5 Onnx | 194.67651 | 20.5199 |
| Jetson Nano Pytorch | 185.45776 | 1.39596 |
| Jetson Nano U20 Pytorch | 95.61312 | 1.29458 |
| Jetson Nano U20 TensorRT | 91.69113 | 1.36545 |
| Jetson Xavier NX Pytorch | 36.90566 | 2.15438 |
| Jetson Xavier NX TensorRT | 31.75837 | 1.70889 |

TABLE II
DELAY AND STANDARD DEVIATION FOR DIFFERENT DEVICES

While these average delay times do not necessarily coincide with frame rate for the final product, they do indicate which boards and models perform best for object detection. This data led to many useful conclusions, such as the importance of choosing the best model type for each board.

### D. Model Training

To ensure we could test our program on the game objects used by FRC teams instead of the objects detected by default from YOLO's provided models, we photographed game objects and trained a custom model on Roboflow [6].

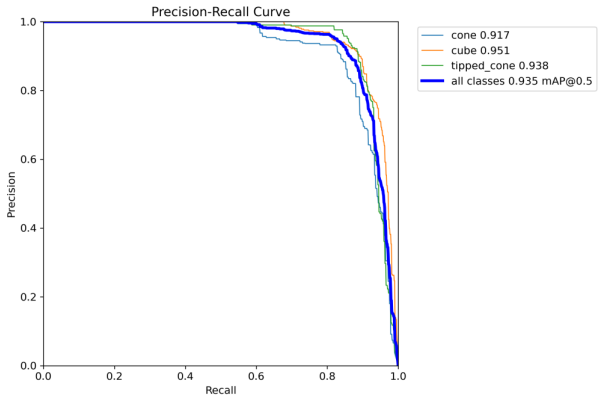


Fig. 5. Precision-recall curve for confidence.

This model helped us achieve high levels of both precision (ensuring accurate results) and recall (ensuring

mostly positive results). This means that the model successfully filters out both false positives and false negatives.

### E. Confidence Importance

During testing, one of the key factors for teams is the ability to alter their confidence thresholds to ensure precision in their detections. Our custom Roboflow-trained model can be analyzed to determine the best confidence threshold to use for the sake of detections.

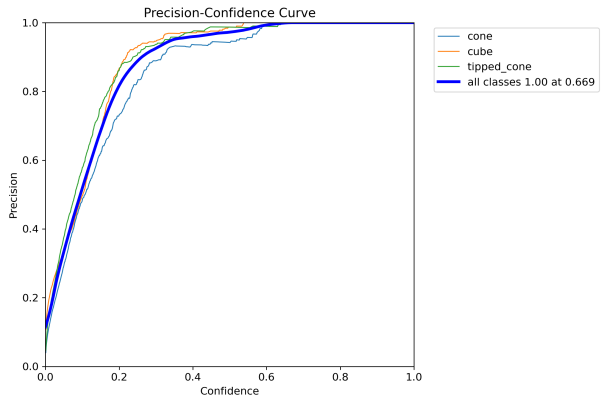


Fig. 6. Curve analyzing precision for each degree of confidence.

Based on this data, we can conclude that confidence thresholds below 0.2 will have significantly diminished precision compared to those above that. We can also determine that confidence thresholds above 0.6 have reduced impact on precision. Teams should be able to expect precise detections on any object close enough to reach at least .4 confidence.

## V. Conclusion

In this paper, we have successfully developed an open-source vision pipeline program for real-time detection of unmarked objects. Our Python-based vision pipeline, integrated with Ultralytics YOLOv8, offers teams a versatile solution for extracting valuable information about unmarked objects across diverse hardware platforms.

The open-source nature of our project fosters collaboration and encourages future research in the field of object detection. By sharing the source code on GitHub (https://github.com/BlueJayADAL/JayRadar), we aim to drive innovation and facilitate widespread adoption of our vision pipeline.

Overall, our vision pipeline presents a powerful tool that empowers teams with efficient and accurate object detection capabilities, holding promise for diverse real-world applications and inspiring further advancements in the realm of computer vision.

## Acknowledgment

This work is supported in part by a grant from Summer Scholarship, Creative Arts and Research Projects (SCARP) Program of Elizabethtown College.

## References

[1] Chief Delphi. (YYYY) Announcing gloworm: An inexpensive and open-source vision module. Chief Delphi. [Online]. Available: https://www.chiefdelphi.com/t/announcing-gloworm-an-inexpensive-and-open-source-vision-module/386370

[2] Limelight Vision. Limelight documentation. Limelight Vision. [Online]. Available: https://docs.limelightvision.io/en/latest/

[3] PhotonVision. Photonvision documentation. [Online]. Available: https://docs.photonvision.org/en/latest/index.html

[4] G. Jocher, A. Chaurasia, and J. Qiu, "YOLO by Ultralytics," Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[5] WPILib. (2023) Wpilib documentation. WPILib. Visited on 2023-07-26. [Online]. Available: https://docs.wpilib.org/en/stable/docs/software/vision-processing/wpilibpi/using-a-coprocessor-for-vision-processing.html

[6] lim Michael Jansen, "Frc charged up game pieces dataset," https://universe.roboflow.com/michael-jansen/frc-charged-up-game-pieces, jan 2023, visited on 2023-07-26. [Online]. Available: https://universe.roboflow.com/michael-jansen/frc-charged-up-game-pieces