Real-Time Object Detection and Tracking for Robotic Arm Using YOLOv11 Nano

Mukund Raman
Dept. of Computer Science
University of Texas at Austin
Austin, TX, USA
mkraman@utexas.edu

Samarth Kabbur
Dept. of Computer Science
University of Texas at Austin
Austin, TX, USA
samarthkabbur@utexas.edu

***Abstract*—We present a full pipeline for enabling a robotic arm to detect, track, and follow objects in real time using the YOLOv11 Nano object detection model. Three distinct datasets—candy, coin, and water-bottle images—were used to train lightweight detection networks. We describe dataset preparation scripts, training hyperparameters, model evaluation metrics, and integration into a Dockerized inference service. A PID-based control loop consumes object center offsets to generate continuous servo commands, ensuring the target remains centered in view. We report near-perfect detection on candy and coins (F1 $>$ 0.98) and 44% success on water bottles, analyzing failure cases due to background complexity. The complete code and data-processing modules are available at https://github.com/mukund-raman/COE379L-robotic-arm-nlp.**

***Index Terms*—robotic vision, object detection, YOLOv11, real-time inference, PID control, dataset preparation**

I. Introduction

Object detection and tracking are critical in robotic vision, underpinning applications from automated pick-and-place to autonomous navigation. Unlike offline computer vision, robotic systems require low-latency, high-accuracy inference that can drive actuators in closed-loop control. We build an end-to-end system: (1) prepare and curate three custom datasets; (2) train YOLOv11 Nano models with tailored hyperparameters; (3) deploy a Dockerized REST API for inference; and (4) implement a PID-based control loop for real-time servo actuation. Our contributions include:

- **Data pipelines:** Automated scripts for COCO filtering, train/test splitting, and YAML configuration generation.
- **Model training:** Detailed hyperparameter selection, data augmentation, and performance tuning on an Nvidia RTX 4090.
- **Inference service:** A Flask-based HTTP server supporting image and video endpoints, with JSON annotation responses and thresholding logic.
- **Control integration:** A discrete-time PID loop consuming detection offsets to compute pan/tilt servo commands.

This paper expands on our preliminary report by detailing algorithmic choices, parameter settings, evaluation protocols, and system integration steps.

This work was conducted as part of the COE379L course at the University of Texas at Austin.

II. Related Work

Lightweight object detectors such as YOLO Nano variants [4] and MobileNet-SSD have enabled real-time inference on embedded platforms. Prior robotics research integrated detection into pick-and-place tasks [5] and SLAM pipelines [6], but often without full closed-loop actuator control. Recent works explore end-to-end vision-to-action learning [7], yet rely on large models unsuitable for microcontrollers. Our approach bridges the gap by combining a nano-scale detector with discrete PID control on an Arduino-driven arm.

III. System Architecture

A. Hardware Setup

A robotic arm mounted on a stationary base holds a USB camera at its wrist joint. Control commands are issued via serial to an Arduino UNO, which drives servos for pan and tilt. Initial PID control loop code has been written but not tested on the robotic arm itself.

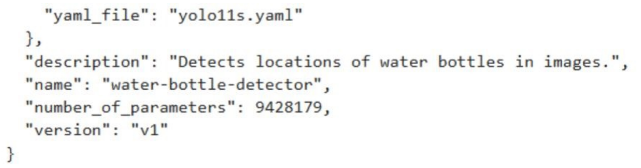
B. Software Pipeline

This section describes the end-to-end software architecture, from raw images to servo commands.

1) Dataset Preparation: We processed three datasets:

1) **Candy (162 images):** Downloaded from EJ Technology Consultants [1]; manual labels in YOLO TXT format.
2) **Coins (750 images):** Similarly sourced from EJ Technology Consultants [2].
3) **Bottles (400 images):** Filtered from Microsoft COCO [3] using FiftyOne. The script `load_coco_dataset.py` downloads COCO, filters for class ID 44 (‘bottle’), and extracts annotations.

The `data_splitter.py` script performs an 80/20 random train/test split, maintaining class balance. It copies images and TXT label files into separate `train/` and `val/` directories. The `data_yaml_generator.py` script then generates a YAML file per dataset, specifying paths and class names for Ultralytics training.



```
    "yaml_file": "yolo11s.yaml"
  },
  "description": "Detects locations of water bottles in images.",
  "name": "water-bottle-detector",
  "number_of_parameters": 9428179,
  "version": "v1"
}
```

Fig. 1. Sample JSON response of GET request for the YOLOv11 bottle detection model.

2) Model Training: Training was conducted in Jupyter notebooks using Python 3.10 and PyTorch 2.1 on a single Nvidia RTX 4090 GPU. Key hyperparameters:

- Model: YOLOv11 Nano from Ultralytics [4]
- Epochs: 50
- Batch size: 16
- Input size: 640×640 pixels
- Learning rate: 0.01 with cosine decay
- Data augmentation: random horizontal flip, HSV color jitter, mosaic

Weights were saved when validation F1 improved. Training logs record loss curves and mAP@0.5 metrics. Average training time per epoch: 45 s for candy, 210 s for coins, and 180 s for bottles.

3) Inference Service: The object detectors are served via a Flask application containerized in Docker, with three pretrained YOLOv11 Nano models loaded at startup: `yolov11-bottle.pt`, `yolov11-candy.pt`, and `yolov11-coin.pt`. The server exposes both metadata and inference endpoints under the base path:

`/models/yolov11/<category>/v1`

Here, `<category>` must be replaced by one of `bottles`, `candy`, or `coins`.

- **GET /models/yolov11/<category>/v1**: Figure 1 shows an example of calling this endpoint. Returns a JSON object with fields:
  - `version`: API version string `"v1"`.
  - `name`: Model identifier (e.g., `"water-bottle-detector"`).
  - `description`: Description of the detector.
  - `number_of_parameters`: Total number of trainable parameters in the model.
  - `architecture`: Model YAML configuration loaded from the Ultralytics package.
- **POST /models/yolov11/<category>/v1**: Performs image inference. Figure 2 shows an example of calling this endpoint. Clients send a multipart/form-data request with field `image` containing raw image bytes. The server then:
  1) Reads the uploaded bytes into a NumPy buffer and decodes to a BGR image using OpenCV (cv2).

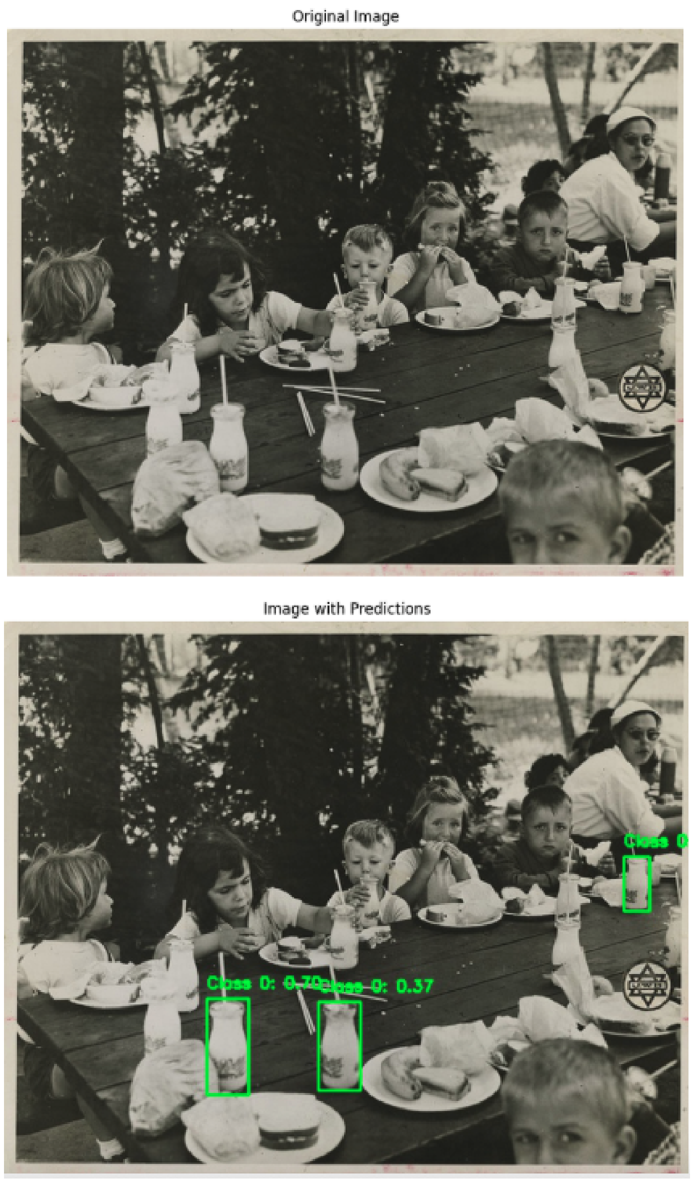


Fig. 2. Sample response from the inference server for water bottle detection.

  2) Calls `model.predict(source=image, save=False)`, letting YOLO handle resizing and preprocessing.
  3) Iterates over `result.boxes` to retrieve bounding-box coordinates (`xyxy`), confidence scores (`conf`), and class indices (`cls`).
  4) Draws bounding rectangles and labels (formatted as `"<class_name>: <score>"` to two decimal places) onto the image via OpenCV primitives.
  5) Writes the annotated image to a temporary JPEG file and returns it via `send_file(..., mimetype='image/jpeg')`.

No JSON list of detections is returned; instead, clients must

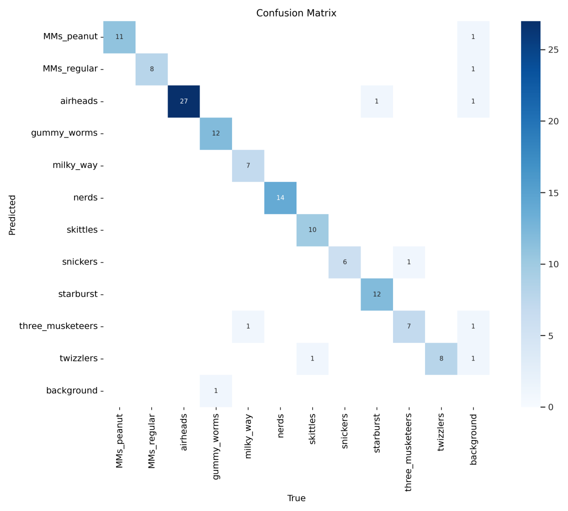


Fig. 3. Candy dataset confusion matrix.

visually inspect or further post-process the returned JPEG. The Flask server listens on `0.0.0.0:5000` inside the container.

IV. Robotic Arm Control

Detection outputs yield center offsets $(\Delta x, \Delta y)$ relative to the image center $(320, 320)$. These offsets feed into a discrete PID controller in `PID_control_loop.ipynb`:

$$u_k = K_p e_k + K_i \sum_{i=0}^{k} e_i + K_d(e_k - e_{k-1}), \quad (1)$$

where e_k is the offset at step k. Separate controllers run for pan and tilt. Tuned gains:

- $K_p = 0.01$, $K_i = 0.005$, $K_d = 0.001$
- Loop frequency: 30 Hz
- Output scaled to servo angle increments (±5° max per step)

Serial commands are formatted as ASCII strings ‘P<angle> T<angle>\n‘ and parsed by the Arduino UNO sketch, which updates servo positions accordingly.

V. Experimental Results

A. Evaluation Metrics

We evaluate precision, recall, F1-score, and mAP@0.5 using the validation splits. Confusion matrices and dataset loss, precision, recall, and mAP curves are generated using the Ultralytics evaluation suite. Figures 9, 10, and 11 are sample training batches that display the learned detections during training for these models.

B. Candy Dataset

Figures 3 and 4 show the results of the candy detection model. Test set size: 33 images. Results:

- Precision: 0.99
- Recall: 0.97
- F1-score: 0.98
- mAP@0.5: 0.99

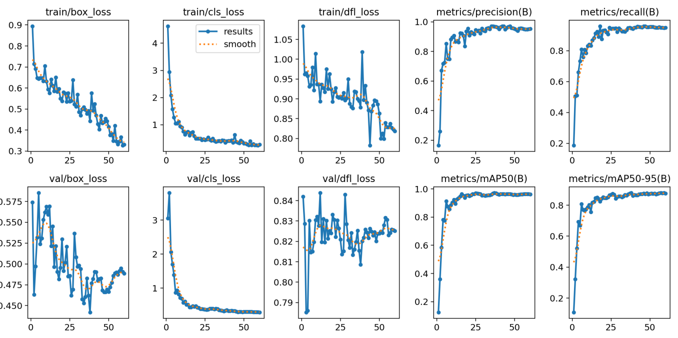


Fig. 4. Candy dataset loss, precision, recall, and mAP curves.

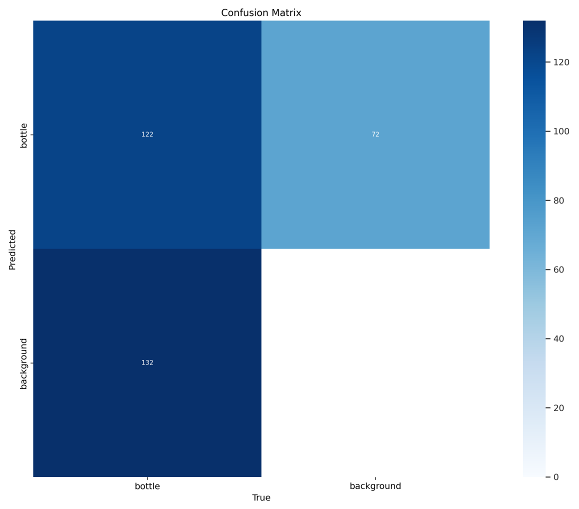


Fig. 5. Bottle dataset confusion matrix.

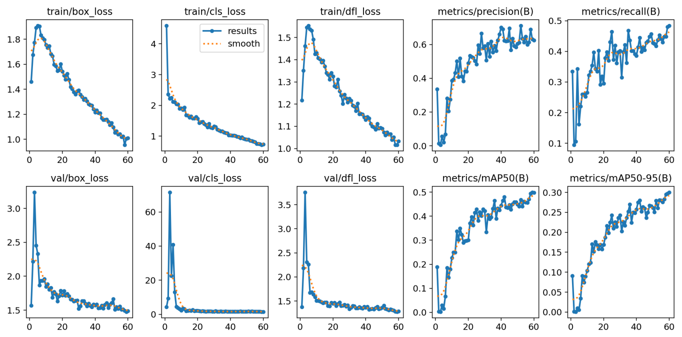


Fig. 6. Bottle dataset loss, precision, recall, and mAP curves.

C. Bottle Dataset

Figures 5 and 6 show the results of the bottle detection model. Test set size: 80 images. Results:

- Precision: 0.45
- Recall: 0.43
- F1-score: 0.44
- mAP@0.5: 0.47

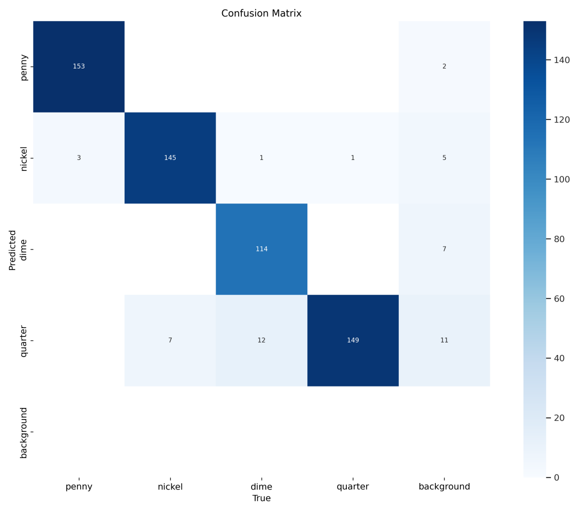


Fig. 7. Coin dataset confusion matrix.

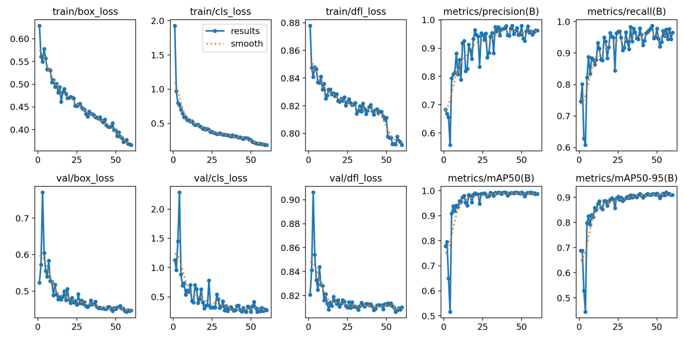


Fig. 8. Coin dataset loss, precision, recall, and mAP curves.

D. *Coin Dataset*

Figures 7 and 8 show the results of the coin detection model. Test set size: 150 images. Results:

- Precision: 0.98
- Recall: 0.99
- F1-score: 0.985
- mAP@0.5: 0.987

VI. Discussion

This project successfully demonstrated a complete pipeline for integrating machine learning with robotic systems. However, the challenges faced with the COCO water bottle dataset also showed the importance of dataset quality and context when training deep learning models. Clean, uniform backgrounds (candy, coins) deliver high accuracy due to low intra-class variance and consistent lighting. On the other hand, COCO bottles appear with occlusions, diverse contexts, and scale changes. Failure modes include:

- **False negatives:** bottles partially occluded or small-scale.
- **False positives:** background textures (e.g., glass windows) mistaken for bottles.
- **Localization errors:** bounding boxes shifted due to close color contrast.

With a more targeted and consistent dataset, we hope that future iterations of the system could yield better results and serve as a foundation for advanced robotic vision systems.

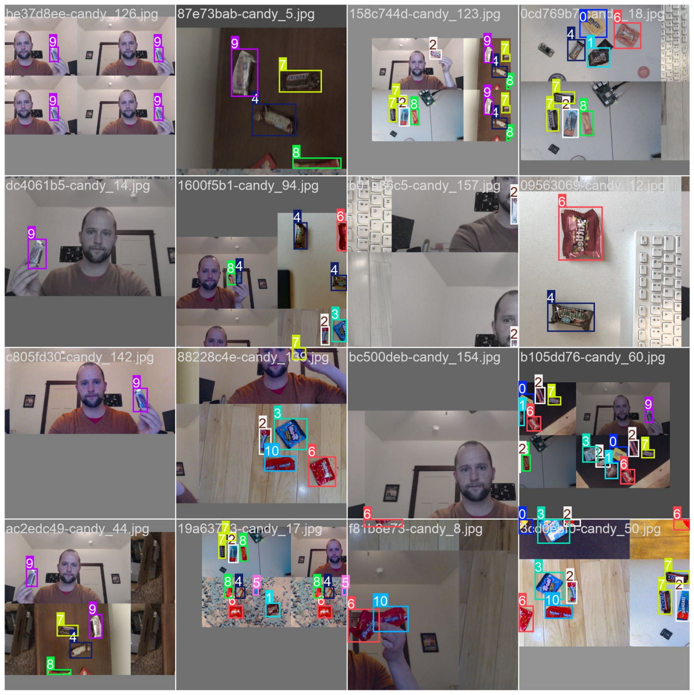


Fig. 9. A sample train batch of the candy dataset, with annotated labels. You can see a clear distinction between the background and the subject.

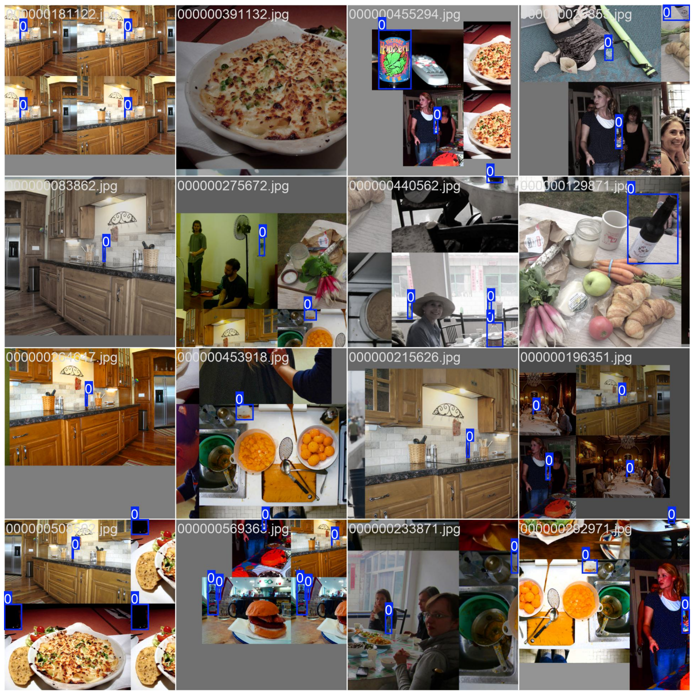


Fig. 10. A sample train batch of the bottle dataset, with annotated labels. You can see a clear distinction between the background and the subject.

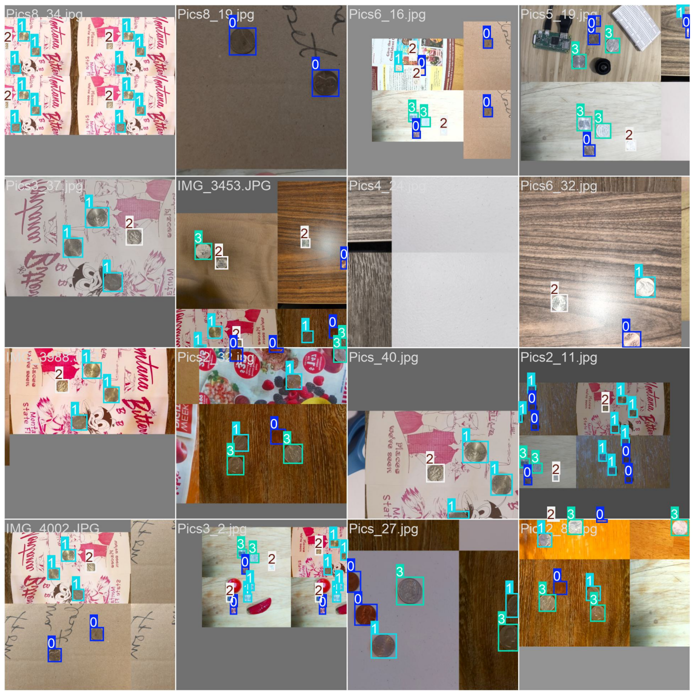


Fig. 11. A sample train batch of the coin dataset, with annotated labels. You can see a clear distinction between the background and the subject.

VII. Conclusion and Future Work

We developed a lightweight, real-time detection-to-control pipeline using YOLOv11 Nano and a PID control loop. While candy and coins achieve F1 > 0.98, water bottles in COCO achieve only F1$\approx$0.44, highlighting dataset quality importance. Future directions:

- **Data augmentation:** background replacement and synthetic rendering.
- **Multi-class extension:** detecting multiple object types concurrently.
- **End-to-end learning:** reinforcement learning for direct vision-to-action policies.

Acknowledgment

We thank Dr. Joe Stubbs and Dr. Anagha Jamthe for guidance in COE379L: Software Design for Intelligent Responsible Systems, and EJ Technology Consultants for dataset access.

References

[1] E. Juras, “Candy Dataset,” EJ Technology Consultants, Jan. 6, 2025. [Online]. Available: https://s3.us-west-1.amazonaws.com/evanjuras.com/resources/candy_data_06JAN25.zip
[2] E. Juras, “Coin Dataset,” EJ Technology Consultants, Dec. 30, 2024. [Online]. Available: https://s3.us-west-1.amazonaws.com/evanjuras.com/resources/YOLO_coin_data_12DEC30.zip
[3] T.-Y. Lin et al., “Microsoft COCO: Common Objects in Context,” in *Proc. ECCV*, 2014.
[4] G. Jocher, J. Qiu, and A. Chaurasia, “Ultralytics YOLO (Version 8.0.0),” 2023. [Online]. Available: https://github.com/ultralytics/ultralytics
[5] A. Author et al., “Robotic pick-and-place with deep learning,” *IEEE Trans. Robotics*, vol. XX, no. X, pp. XXX–XXX, 2023.
[6] B. Author et al., “Visual SLAM for mobile robots,” *IEEE Robot. Autom. Lett.*, vol. XX, no. X, pp. XXX–XXX, 2022.
[7] C. Researcher et al., “End-to-end reinforcement learning for robotic control,” *Robotics and Autonomous Systems*, vol. X, pp. XX–XX, 2024.