2000004855

Emrehan Gökçay

Programming 3 Project Design Document:

Implementation Details:

Programming Language:

The code uses Python programming language and uses these libraries below:

Ultralytics YOLO: The Ultralytics YOLO library is used for real-time object detection. It provides pre-trained YOLO models that can be easily integrated into the application. The YOLOv8 model is instantiated in the ObjectTracker class.

OpenCV: OpenCV is employed for camera input and image processing. The Camera class initializes a video capture object using OpenCV, and frames are processed using OpenCV functions.

Data Description:

Input Data:

Video frames obtained from the camera using OpenCV.

Pre-trained YOLO model weights file (best.pt) used for object tracking.

Output Data:

Detected object bounding boxes and associated information.

Algorithms and Libraries:

Object Tracking Algorithm:

This Python code is designed for real-time object tracking using a custom-built dataset trained by Ultralytics' YOLO model. This code allows us to actively try to find a flying UAV type object, and if it finds images that are similar to the average of the 3000+ images on which the dataset was trained (in this case, the “best.pt” file), it flags them, also showing the percentage of confidence in 2 decimal point fraction. To make a custom dataset file with YOLOV8 training you need to make these steps:

* Gather around 3000+ images for selected object to track(in this case only flying unmanned aerial vehicle objects, it won’t detect anthing unless their average coincides with live footage) and open a drive file, upload those images on a drive folder and create a .yaml file for dataset configuration.
* Open a Google colab notebook on this page: <https://colab.research.google.com/>
* Select new notebook.
* On this notebook click folder icon and click on add drive icon.
* Find .yaml file and click on “copy path”.
* Open 2 code cells and write this code on the first one:
* !git clone <https://github.com/ultralytics/ultralytics.git>

%pip install ultralytics

import ultralytics

ultralytics.checks()

* And this to the second one:
* !yolo train model=yolov8x.pt data=/content/pathto/data.yaml epochs=75 imgsz=640 (This code tells us to train data.yaml content with yolov8x(most accurate yolov8 model) and train it through 75 loops and size of imput images as integer.
* Wait until training finishes and after that download best.pt file and add it into your code.

Here's an explanation of the provided Python code, including details on the functions performed, user interface functions, software architecture layers (presentation, business, data access, data), and module interactions, along with hardware structure details:

**Functions:**

1. **Capture Video Input (Camera Class):**
   * **Function:** The **Camera** class is responsible for capturing video input using OpenCV.
   * **Detail:** It uses **cv2.VideoCapture** to obtain frames from the specified source (default is 0, the primary camera). Additionally, it sets the frames per second (FPS) to 120.
2. **Object Tracking (ObjectTracker Class):**
   * **Function:** The **ObjectTracker** class performs object tracking using the Ultralytics YOLO model.
   * **Detail:** During initialization, it loads the YOLO model from the specified file ("best.pt"). The **track** method uses the YOLO model to detect flying plane like objects only in the input data and returns the results.
3. **Drawing Bounding Boxes (draw\_boxes Function):**
   * **Function:** Draws bounding boxes around detected objects.
   * **Detail:** The **draw\_boxes** function takes bounding box coordinates from the object tracking results and prints them. It creates a list containing the coordinates for each detected object.
4. **Main Program (main Function):**
   * **Function:** Captures video input, performs object tracking, and draws bounding boxes.
   * **Detail:** The main program runs in a continuous loop using instances of the **Camera** and **ObjectTracker** classes. In each iteration, it captures a frame, performs object tracking, and draws bounding boxes. The loop exits when the 'q' key is pressed.

**Layers and Module Interactions:**

1. **Presentation Layer (User Interface):**
   * The user interface is not explicitly defined in the code. However, visual results are displayed by drawing bounding boxes on the screen.
2. **Business Layer (Business Logic):**
   * Object tracking using the YOLO model is implemented in the **ObjectTracker** class, representing the business logic layer.
3. **Data Access Layer (Data Access):**
   * The acquisition of camera images and the utilization of the YOLO model occur through the **Camera** and **ObjectTracker** classes, representing data access operations.
4. **Data Layer (Data):**
   * Object detection results, specifically bounding box coordinates, are displayed on the screen through the **draw\_boxes** function, representing the data layer.

Hardware Structure:

GPU: Nvidia GeForce RTX 3080

Utilization of CUDA cores for image processing.

The presence of a powerful GPU like RTX 3080 enhances the speed of computations, especially for tasks involving deep learning models like YOLO. It reduces latency rates of inference and speed.

Other Hardware Considerations:

The computer running this code is equipped with a CUDA-enabled environment and the necessary dependencies to leverage the GPU for accelerated processing.