ARI2201 - Individual Assigned Practical Task: Distance Estimation between Moving Cameras and Objects

Miguel Baldacchino | 0171205L
University of Malta
Dr. Kristian Guillaumier

CONTENTS

***Abstract*—This paper presents a real-time system for estimating the distance between a moving camera and surrounding vehicles using YOLO object detection and monocular vision geometry. The implementation is tailored for cycling footage and supports dynamic camera calibration.**

I. INTRODUCTION

In the past few years, advancement in fields such as computer vision and deep learning techniques led the development of systems capable of interpreting the real-world, via visual data. A critical area of application, is that of estimation of distances between moving cameras (such as bicycle mounted cameras and dash-cams) and objects in frame including other road users (such as vehicles, pedestrians and cyclists).

II. MOTIVATION

Cyclists and other vulnerable road users often face risks due to close proximity interactions with larger vehicles. In most countries, a legal mandate is present requiring vehicles to maintain a safe distance when overtaking cyclists. However, these rules are often violated, with enforcement remaining a challenge. A technology which monitors and records unsafe driving behaviour would be of benefit to many.

A lightweight, camera-based distance estimation system can serve multiple purposes: alerting users in real-time when another object is dangerously close.

From a technical perspective, estimating distance only using a monocular camera for video input - without depth sensors or stereo vision - brings up a challenge in computer vision.

III. AIMS AND OBJECTIVES

Aim

This project aims to design and implement a vision-based system that estimates the distance between a moving monocular camera and surrounding objects, leveraging object detection and geometric analysis. The system supports multiple use cases, including safety alerts for users (such as cyclists).

Objectives

To achieve this aim, the following S.M.A.R.T. (Specific, Measurable, Achievable, Relevant, Time-bound) objectives were defined:

- **Specific:** Develop a system that accepts video input, detects known object classes (e.g., cars, bicycles, pedestrians), and estimates their distance from the camera.
- **Measurable:** Implement visual feedback overlays (color-coded bounding boxes and warnings) when detected objects are closer than 1.5m (caution) or 0.8m (danger), and support exporting processed video with annotations.

- **Achievable:** Pre-trained YOLOv8 for object detection and classical triangle similarity for distance estimation, using user-assisted calibration to compute focal length of an unknown camera, or select a previously used calibration.
- **Relevant:** Address the growing need for cyclist and pedestrian safety tools, especially in countries where minimum overtaking distances are legally bound.
- **Time-bound:** This project was developed as part of the ARI2201 IAPT, by the official deadline.

IV. Literature Review

Cyclist Safety and EU Policy

Cyclist safety remains an issue across Europe, as according to the European Commission, 2,000 cyclists were killed on EU roads in 2019, with many more seriously injured [1] [2]. To mitigate this, several EU members, US states, and provinces in Canada have introduced legal provisions requiring vehicles to maintain a minimum overtaking distance from cyclists of 1.5m [3]. However, enforcement of these regulations remains limited due to the difficulty in collecting evidence.

Monocular Distance Estimation

Distance estimation from monocular cameras is an active area of research in computer vision. Unlike stereo vision systems which work by directly inferring depth through disparity, monocular systems rely on:

1. Geometric Assumptions: Traditional methods estimate distance based on known object dimensions and camera's focal length [4]. This approach is computationally efficient, but it is sensitive to calibration accuracy and assumes objects perpendicular to ground level.

2. Perspective and Contextual Cues: This approach leverages monocular depth cues such as relative object size, linear perspective (vanishing points), texture gradients, interposition (occlusion ordering), and motion parallax to infer relative depth [5]. While more flexible than purely geometric methods, these still lack precision in real-world dynamic scenes unless supported by additional context.

3. Learned Depth Priors: Recent advancements in deep learning have enabled monocular depth estimation through models that learn depth priors directly from visual input. These systems are trained end-to-end on large datasets to infer dense depth maps from a single frame. Its worth mentioning open-source implementations such as MonoDepth2 [6] and Apple's ML-Depth Pro [7], both of which approximate depth using deep neural networks with high accuracy. However, these models typically require substantial computational resources and unsuitable for real-time applications.

Object Detection Models

Accurate object detection is a requirement for distance estimation in this system. The YOLO (You Only Look Once) models offer real-time object detection with high accuracy. Trained on the COCO dataset, YOLOv8n (nano version) supports detection of a wide range of road users including cars, pedestrians, bicycles, and buses; making it suitable for cyclist safety scenarios [8]. Its high inference speed make it ideal for deployment in such systems.

Focal Length and Triangle Similarity

Focal length is a must in monocular distance estimation. The pinhole camera model defines the relationship between the object's real height H, its height in the image h, the focal length f, and the estimated distance D as [4]:

$$D = \frac{H \cdot f}{h}$$

To apply this formula, either intrinsic camera parameters must be known in advance, or focal length must be approximated through calibration. A semi-automatic calibration step which would allow users to select a known object in the video and input its actual distance, from which f is calculated could be utilized in my approach.

Similar Real-World Applications

Real-time proximity awareness common in Advanced Driver Assistance Systems (ADAS), with features such as forward collision warning and blind spot detection. Commercial systems by companies like Mobileye and Tesla Vision have shown the feasibility of vision-only safety systems [9] [10]. While such systems are integrated into vehicles with proprietary hardware, this project could explore the consumer-grade equivalent targeted at cyclists, using only monocular video input and a general-purpose object detection model.

Prior Work

Many datasets are used as benchmarks in the computer vision to evaluate detection and depth estimation systems, such as KITTI [11] and Cityscapes **cityscapes2016**. These datasets include stereo and monocular camera inputs, annotated bounding boxes, and sometimes LiDAR ground truth for depth. These resources illustrate the standard methodologies and data formats in the field, even if not to be used in this project.

V. Methodology

A. System Overview

The implemented system is a Python-based desktop application designed to estimate the distance of nearby road users from monocular video footage. It integrates object detection, geometric distance estimation, and a graphical user interface to provide a complete, user-interactive tool for evaluating proximity risks in real time.

Technology Stack: The application was built using the following technologies:

- **Python 3.10+** – Core programming language.
- **OpenCV** – Used for video frame capture, image processing, ROI selection, and visual annotation.
- **Tkinter** – Provides the GUI for video input, calibration, playback controls, export, and speed selection.
- **Ultralytics YOLOv8n** – Lightweight object detection model pre-trained on the COCO dataset, used to identify vehicles, pedestrians, and cyclists in video frames [8].

- **PIL (Pillow)** – Used to render frames into the GUI.
- **NumPy and JSON** – Handle image data manipulation and focal length calibration persistence.

Core Features: The system provides a number of integrated functions:

- **Real-Time Detection:** Each frame of the selected video is processed by the YOLOv8n model detecting only relevant object classes such as cars, buses, bicycles, motorcycles, pedestrians, and trucks.
- **Distance Estimation:** For each detected object of known class and height, the system estimates distance using triangle similarity based on the pixel height of the object and the calibrated camera focal length.
- **Calibration Interface:** Users can load a previous calibration, or calibrate a new camera into the system by selecting a detected object and providing its actual distance, allowing the system to compute and store the effective focal length in pixels.
- **Proximity Warnings:** Visual alerts are rendered on the video feed when detected objects are within critical proximity thresholds (1.5m = caution, 1.2m = danger).
- **Video Export:** Users can export a fully annotated video showing bounding boxes, estimated distances, and safety warnings.
- **Preview Mode:** Allows users to process and view video in real time without saving.
- **Preview Speed:** Users can select from multiple playback speeds (e.g., 1x, 4x, 7x, 12x, 15x) to speed up video analysis during preview.
- **Preview Stop/Resume:** The system supports stopping and resuming preview at any point for flexible inspection.
- **Ignore Bottom Third:** An optional toggle allows users to exclude pedestrians detected objects in the bottom third of the video frame, avoiding false warnings camera wearers.
- **Double-Decker Bus Toggle:** A user-configurable option lets users indicate whether double-decker buses are common in their region. If enabled, buses with large bounding box heights are treated as taller vehicles (4m).

The entire system runs locally and is optimized for use with consumer-grade hardware, requiring no internet access or external servers.

B. Dataset and Design

The system accepts video input through manual file selection, allowing users to process any `.mp4` footage. For the purposes of this project, real-world testing was conducted using footage sourced from YouTube [12] [13] [14] [15] [16], covering a diverse range of scenarios:

- **Dashcam recordings** from moving vehicles representing stable and consistent input.
- **Bicycle-mounted camera footage**, both stable and unstable, introducing natural movement and vibrations.
- **Footage containing partial occlusions** such as cyclist hands on handlebars, testing the model's ability to handle partial visibility.

This variety of footage allows for better test conditions later for evaluating the robustness of both object detection and distance estimation under realistic and challenging use cases.

C. Implementation Details

*1) **Application Startup:*** Upon execution, the application initializes the graphical user interface (GUI) using the `tkinter` library. The main window includes a video display area and a control panel for user interaction (see Figure 1).

During startup, the system also sets internal state variables, loads the object detection model (`YOLOv8n.pt`) using the `ultralytics` framework, and prepares to capture video frames from either a default video file (`footage1.mp4`) or a user-specified path entered via the interface.

This design ensures that the system is ready for immediate calibration or playback operations as soon as the user selects a video source.

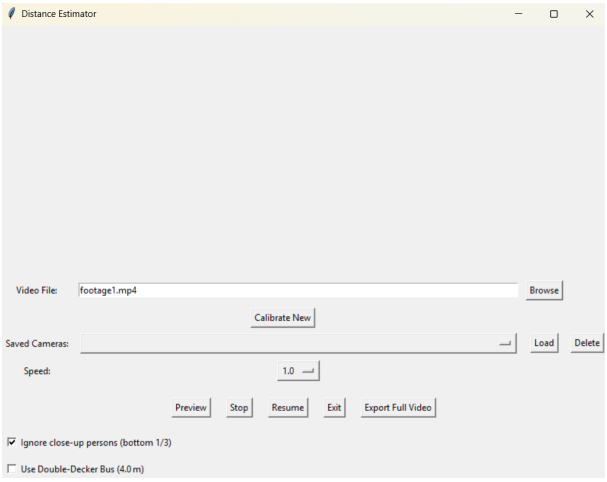
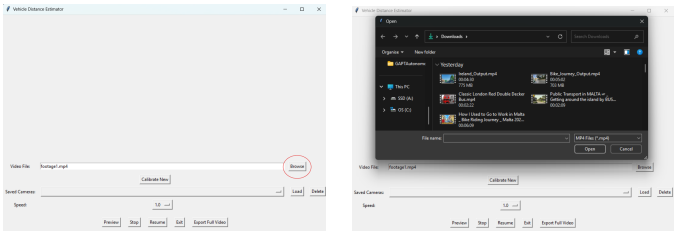


Figure 1: Main GUI layout on application startup showing the video preview area and control panel.

*2) **Camera Calibration:*** Before calibration begins, the user must select a video source. By default, the system loads a predefined file (`footage1.mp4`). Alternatively, users can:

- Browse and select a video file using the file dialog (see Figure 2).
- Manually enter the path to a video file in the input field.



(a) Browse button on GUI (b) File selection dialog

Figure 2: User selecting a video file via the Browse button or manual path input.

Once the application is launched, users may either load a previously saved focal length (camera calibration) or initiate a new calibration process. Calibration is necessary to enable accurate monocular distance estimation using triangle similarity.

a) Calibration Workflow:

1) Once a video is selected, the user initiates the calibration tool via `Calibrate New` button.
2) The system finds a frame with a known object class (e.g., `car`, `person`, `bicycle`) using the YOLOv8n model (see Figure 3).

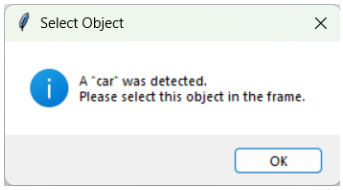


Figure 3: Frame with detected object found. User is to select that object.

3) A bounding box is manually selected by the user over the detected object using an OpenCV ROI selector (see Figure 4), and press Enter/Space to continue, or c to cancel selection process.

Figure 4: Selecting ROI

4) The user is prompted to enter the actual distance from the camera to the selected object (see Figure 5).

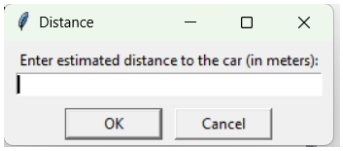


Figure 5: Input Object Distance from Camera

5) The focal length is computed using the formula:

$$f = \frac{h \cdot D}{H}$$

where h is the pixel height of the object, D is the real-world distance (entered by the user), and H is the known real-world height of the object class.

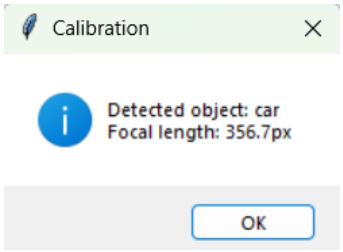


Figure 6: Focal Length Calculated

6) The computed focal length is saved locally under a user-defined camera name (see Figures 7 and 8).

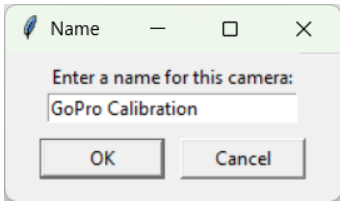


Figure 7: Naming Calibration to Save

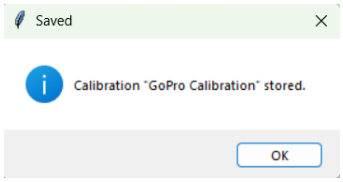


Figure 8: Calibration Saved

All saved calibrations are stored as JSON files in a dedicated `calibrations/` directory and can be reloaded or deleted as needed using the GUI (see Figure 9).

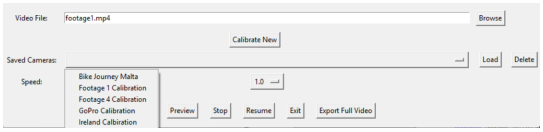


Figure 9: Camera selection, loading, and deletion options in the GUI.

*3) **Real-Time Video Processing**:* Once the focal length has been calibrated or loaded, users can initiate the main video processing routine via the `Preview`. The application reads the video frame-by-frame using OpenCV and applies object detection and distance estimation in real-time.

For each frame:

1) The YOLOv8n object detection model identifies relevant objects, such as cars, buses, bicycles, motorcycles, trucks, and pedestrians.
2) Bounding boxes are extracted from each detected object.

3) The pixel height of each bounding box is used along with the known object class height and focal length to estimate the real-world distance via triangle similarity:

$$D = \frac{H \cdot f}{h}$$

where D is the distance, H is the known real-world height of the object, f is the focal length, and h is the pixel height of the bounding box.

4) Based on the estimated distance, objects are classified into proximity levels:
   - **Green:** Safe distance
   - **Orange:** Close (below caution threshold of 1.5m, as per EU Laws on minimum overtaking distance [3])
   - **Red:** Too close (below danger threshold of 1m)
5) Visual overlays are applied to the video to display bounding boxes, distance estimates, and a warning banner if any object is too close (see Figure).

To support real-time distance estimation, a dictionary of known average object heights (e.g., cars, bicycles, trucks, pedestrians) is hardcoded in the application.

```
OBJECT_HEIGHTS = {
    'car': 1.5,
    'person': 1.6,
    'motorcycle': 1.2,
    'bicycle': 1.1,
    'truck': 2.2,
    'bus': 3.0
}
```

The flexible design allows for experimentation with different camera types, angles, and traffic environments, ensuring generalizability of results.

*4) **Preview**:* The preview feature allows users to review distance estimation results in real-time without exporting the video. This functionality is useful for quickly assessing system performance, testing calibration accuracy, or reviewing specific moments in the footage.

Upon loading a video and selecting a calibration profile, users can start the preview by pressing the `Preview` button on the control panel. The video feed is displayed inside the GUI, where detections and proximity warnings are rendered frame-by-frame (see Figure 10).

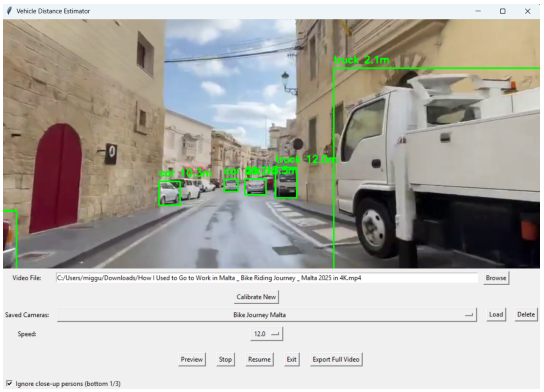


Figure 10: User triggering real-time preview from GUI.

Speed Control: A dropdown menu allows users to adjust the preview speed. Available options include `1x`, `4x`, `7x`, `12x`,and `15x`, corresponding to how many frames are skipped per cycle (Figure 11).

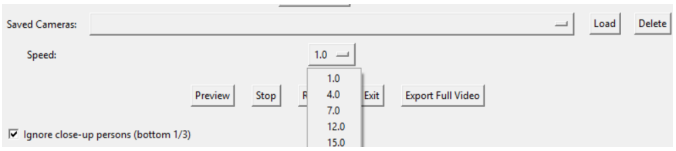


Figure 11: Speed selector used to adjust playback rate.

Stop and Resume: Users may stop the preview at any point using the `Stop` button and resume playback using the `Resume` button.

Ignore Bottom Third Option: To avoid false warnings triggered by cyclists hands in frame, an optional checkbox labeled `Ignore close-up persons (bottom 1/3)` is available (Figure 12). When enabled, the system excludes any person detections that appear in the bottom third of the video frame.

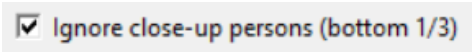


Figure 12: Option to ignore pedestrian detection appearing in the lower third of the frame.

These preview features allow users to quickly test their setup before exporting annotated video, optimizing both performance and result clarity.

Additional heuristics were introduced to improve the accuracy of large vehicle detection:

- **Bus Height Heuristic:**

```
if label.lower() == 'bus':
    real_h = 4.0 if self.use_double_decker
        .get() else 3.0
```

Taller buses, such as double-decker buses, are assigned a greater real-world height value when the user enables the Double-Decker Bus option. This toggle has been added to allow the user to specify which type of bus is more common in their country. For instance, in Malta, double-decker buses are relatively rare, so the option would typically remain disabled. In contrast, countries like the UK frequently use double-deckers, in which case enabling the option would be more appropriate.

By default, the option is left disabled, as this minimizes the risk of false negatives in proximity warnings. In safety-critical systems like this, it is preferable to slightly overestimate proximity (false positives) than to underestimate it.

This bus differentiation mechanism was necessary because YOLO classifies all buses under a single label ("bus"), without distinguishing between single and double-decker types. Given that double-decker buses are typically at least one metre taller than standard buses, accurate height assignment is critical for reliable distance estimation.

- **Truck Height Heuristic:**

```
if label.lower() == 'truck':
    if h > 170:
        real_h = 2.8  # large truck
    if h > 230:
        real_h = 3.2  # larger truck
```

  Truck height is adjusted according to the pixel height of the bounding box to reflect various truck types (e.g., delivery vans vs. articulated trucks).

To reduce false positives when the cyclists hand or body is in frame, the system includes an option to ignore persons detected in the lower third of the frame, included in the GUI (see Figure 1):

```
if label == 'person' and self.
    ignore_low_persons.get() and y2 > frame_h
    * (2/3):
    continue
```

This setting helps eliminate irrelevant proximity alerts from the cyclists hands on the handlebar.

*5) **Exporting Results**:* After choosing video source and loading a calibration, the system allows users to process the entire video, export it with all annotations. This functionality is particularly useful for documenting proximity violations.

The export process includes:

- Reading the selected video file frame by frame.
- Applying object detection, bounding box rendering, and distance overlays identical to the preview mode.
- Drawing coloured rectangles (green, orange, red) based on proximity classification.
- Adding warning overlays in frames where objects are detected within dangerous distance thresholds.

The fully annotated video is then saved as an `.mp4` file using OpenCV's `VideoWriter` class, with the destination path specified by the user through a file save dialog. During export, progress is printed to the terminal at regular intervals to indicate the number of frames processed (see Figure 13).

```
Processed frame 50/22138
Processed frame 100/22138
Processed frame 150/22138
Processed frame 200/22138
Processed frame 250/22138
Processed frame 300/22138
Processed frame 350/22138
```

Figure 13: Command-line output during video export showing progress updates.

D. Evaluation Strategy

To assess the accuracy of the developed system, multiple evaluation strategies were used:

- **Parallel Object Comparison:** Videos containing two different road users (e.g., a car and a bicycle) positioned beside each other were analysed to verify if the system estimated their distances consistently.
- **Detection Accuracy:** The correctness of object detection was evaluated by comparing the detected classes and bounding boxes against visual ground truth. This involved manual frame-by-frame review of multiple annotated videos to determine the rate of correct detections and false positives/negatives.
- **Truck Size Differentiation:** Special attention was given to trucks, using the implemented heuristics to compare the system's ability to distinguish between small and large trucks.
- **Bus Size Differentiation:** Accurate single and double decker bus distance estimations.
- **Obstruction Handling:** Footage where the cyclist's hands or handlebars partially obstruct the lower third of the frame was tested to confirm that such close-up detections were correctly ignored when the setting was enabled.
- **Stable vs. Shaky Footage:** The system was tested on both smooth dashcam videos and unstable, shaky footage recorded from bicycles in motion.

VI. Evaluation and Discussion

A. Accuracy of Detections

Overall, the object detection performance is highly accurate, owing to the robustness of the YOLOv8n model. Most objects are correctly identified and tracked, even under varying lighting and environmental conditions. However, some outliers were observed. For example, a garbage bag was incorrectly classified as a car, and a construction skip was similarly mistaken for a car. At longer distances, the model occasionally struggles to differentiate between cars and trucks due to reduced resolution and object size. Nonetheless, these cases were infrequent, and the model's real-time detection capabilities remain consistently reliable.

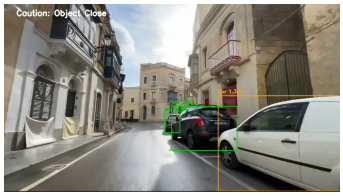


Figure 14: Object correctly identified at a close distance

Figure 15: Object marked as too close, triggering warning overlay

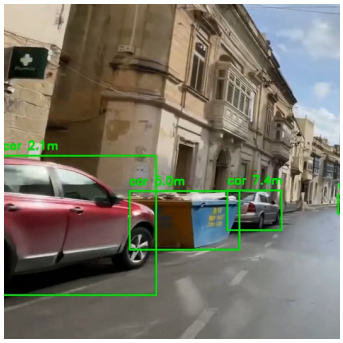


Figure 16: A skip incorrectly detected as a car

Figure 17: Garbage bag misclassified as a car

B. Parallel Object Detection and Distance Consistency

Detecting different object classes in the same frame and estimating their distances consistently. In scenarios where two distinct objects (such as a pedestrian and a vehicle) appear side by side, the system correctly assigns similar distance estimates.

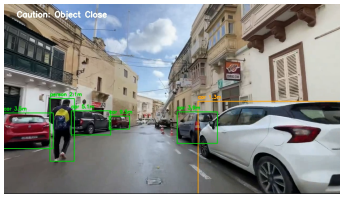


Figure 18: Pedestrians and vehicles at similar distances assigned consistent distance estimates

Figure 19: Motorcycle and car side by side with comparable estimated distances

C. Truck Size Differentiation

The system consistently estimates similar distances for trucks.

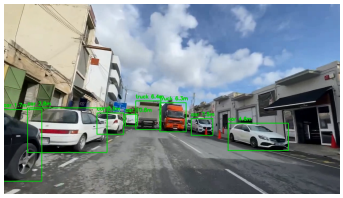


Figure 20: Consistent size estimation for trucks

Moreover, when a truck appears alongside another vehicle such as a car, the system assigns comparable distance estimates to both.

Figure 21: Relative size comparison between truck and car

D. Camera Stability

Both stable and slightly shaky videos were tested. The detection system maintained performance under both conditions, though excessive shake may occasionally shift bounding box positions slightly. Overall, no significant accuracy degradation was observed.

E. Bus Size Differentiation

Both scenarios in the figures below show the system's ability to estimate distance accurately. In the first image, a double-decker bus is correctly identified using the adjusted height setting, and its distance estimation closely matches that of a car positioned beside it. In the second image, a standard single-decker bus is shown with the default setting, and the estimated distance aligns well with its real-world position.

Figure 22: Double-decker bus with adjusted height estimation; distance matches adjacent car

Figure 23: Standard bus with default height estimate showing accurate distance

F. Obstruction Handling

The system provides an option to ignore persons detected in the lower third of the frame. This helps avoid false positives when a small part of a person (such as the cyclists hand on the handlebars). When enabled, this setting ensures detections are contextually meaningful and not triggered by partial occlusions.

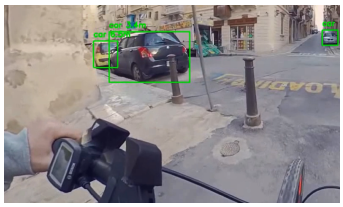


Figure 24: Hand ignored due to user setting (bottom third ignored)

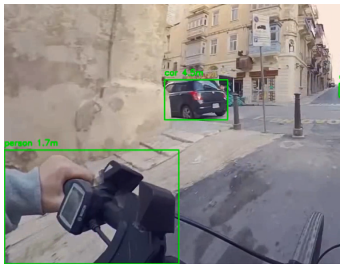


Figure 25: Hand detected when ignore setting is disabled

7. Conclusion and Future Work

This project was aimed at developing a monocular distance estimation system tailored for cyclist safety. The application successfully integrates object detection using YOLOv8n, focal length calibration through manual selection, and distance estimation via triangle similarity. A user-friendly GUI was built using Python and Tkinter, allowing video input, calibration storage, preview and dedicated speed, stop and resume functionality, and export functionality. The system was evaluated across a diverse set of real-world videos, demonstrating robustness in detecting proximity violations and correctly handling variable object types.

While the system achieves its primary objectives, future improvements could be:

- **Stereo Vision Integration:** Incorporating stereo camera input would eliminate the reliance on manual calibration and geometric assumption.
- **Monocular CNN Depth Estimation:** Integrating state-of-the-art monocular depth estimation models, such as MonoDepth2 [6] or Apple's ML-Depth Pro [7].
- **Real-Time Feedback for Cyclists:** The system could be extended to issue real-time alerts using audio or haptic feedback when objects come within critical proximity.
- **GPS-Enhanced Reporting:** Combining GPS data with video evidence could enable automatic generation of location-tagged proximity reports to police enforcement.
- **Enhanced Bus and Truck Differentiation:** While the current system uses user-configurable heuristics to differentiate between standard and double-decker buses and an automatic truck size adjustment is present, integrating a more advanced detection model capable of distinguishing these sub-classes automatically could further streamline the process and enhance overall precision without requiring user input.

References

[1] European Commission, *Road Safety Thematic Report: Cyclists*, https://road-safety.transport.ec.europa.eu/document/download/c82fa210-8707-4402-a9be-b70deded1d5e_en?filename=road_safety_thematic_report_cyclists.pdf, Accessed: 2025-05-30, 2023.

[2] European Commission, *Traffic rules and regulations for cyclists and their vehicles*, https://road-safety.transport.ec.europa.eu/eu-road-safety-policy/priorities/safe-road-use/cyclists/traffic-rules-and-regulations-cyclists-and-their-vehicles_en, Accessed: 2025-05-30, 2023.

[3] European Parliament, *Written question: Road safety – distance for overtaking cyclists*, https://www.europarl.europa.eu/doceo/document/E-8-2017-002358_EN.html, Accessed: 2025-05-30, 2017.

[4] The Green Alliance, *Introduction to Pinhole Cameras*, https://thegreenalliance.dev/cameras/intro-to-pinholes/, Accessed: 2025-05-30, 2023.

[5] Warren Forensics, *Binocular and Monocular Cues in Depth Perception*, https://www.warrenforensics.com/2025/01/20/binocular-and-monocular-cues-in-depth-perception/, Accessed: 2025-05-30, Jan. 2025.

[6] Niantic Labs, *monodepth2: PyTorch implementation of MonoDepth2*, https://github.com/nianticlabs/monodepth2, Accessed: 2025-05-30, 2019.

[7] Apple Inc., *ML-Depth: High-Quality Depth Estimation from Monocular Video*, https://github.com/apple/ml-depth-pro, Accessed: 2025-05-30, 2023.

[8] Ultralytics, *YOLOv8 by Ultralytics*, https://yolov8.com/, Accessed: 2025-05-30, 2023.

[9] Mobileye, *Mobileye Launches the First Camera-Only Intelligent Speed Assist to Meet New EU Standards*, https://www.mobileye.com/news/mobileye-launches-the-first-camera-only-intelligent-speed-assist-to-meet-new-eu-standards/, Accessed: 2025-05-30, 2022.

[10] Caradas, *Tesla Vision: How Tesla's Camera-Only ADAS Works*, https://caradas.com/tesla-vision-adas-features/, Accessed: 2025-05-30, 2023.

[11] A. Geiger, P. Lenz, and R. Urtasun, *Kitti vision benchmark suite*, https://www.cvlibs.net/datasets/kitti/, Accessed: 2025-05-30, 2012.

[12] 4. World. "How i used to go to work in malta | bike riding journey | malta 2025 in 4k." Accessed: 2025-05-30. (2025), [Online]. Available: https://www.youtube.com/watch?v=6fRjqpGu2ek.

[13] Aviationvlad. “Public transport in malta mt | getting around the island by buses.” Accessed: 2025-05-30. (2022), [Online]. Available: https://www.youtube.com/watch?v=pIND9QWMxs4.

[14] C. stroll. “4k dublin ireland 2025 | driving downtown - dashcam ireland |crumlin, terenure, milltown, donnybrook.” Accessed: 2025-05-30. (2025), [Online]. Available: https : / / www . youtube . com / watch ? v = 5GoZItUmeOE.

[15] D. Trends. “Contour roam2 sample footage: City bike ride.” Accessed: 2025-05-30. (2013), [Online]. Available: https://www.youtube.com/watch?v=x0AXNq2Fcc8.

[16] Unknown. “Cycling in malta: Exploring valletta.” Accessed: 2025-05-30. (2018), [Online]. Available: https://www.youtube.com/watch?v=wejKHRGls50.