Code Base Documentation

```
import cv2

import argparse

from ultralytics import YOLO

import parameter
```

Explanation- Here, the code begins by importing necessary libraries and modules, including OpenCV (cv2) for image and video processing, argparse for command line argument parsing, YOLO from the ultralytics library for object detection, and a custom module 'parameter' store some configuration values.

```
parser = argparse.ArgumentParser(description="Object Detection on Video")

parser.add_argument("--path", required=True, help="Path to the video file")

args = parser.parse_args()
```

Explanation- An argument parser is set up to accept a command line argument --path, which is required and is expected to be the path to the video file. The argument is then parsed, and the result is stored in the args variable.

```
yolo_model = YOLO(parameter.model_name)
```

A YOLO object detection model is initialized using the model name defined in the 'parameter' module.

```
class_names = yolo_model.names
```

Explanation- The names of the classes that the YOLO model can detect are extracted and stored in the class_names variable.

```
video_capture = cv2.VideoCapture(args.path)

if not video_capture.isOpened():

    print(f"Error opening video file: {args.path}")

    exit(1)
```

Explanation- Here, a video capture object is created using OpenCV's VideoCapture class, with the video file path provided as an argument. It checks whether the video file was opened successfully and prints an error message if it wasn't, then exits the program with an error code.

```
while True:
    ret, frame = video_capture.read()
    if not ret:
        break
```

Explanation- A loop is started to read frames from the video file using video_capture.read(). If there are no more frames to read (ret is False), the loop is terminated.

```
    height, width, _ = frame.shape
```

Explanation- The height and width of the current frame are extracted using the .shape attribute of the frame.

```
    results = yolo_model.predict(
        source=frame,
        imgsz=parameter.image_size,
        conf=parameter.confidence,
        save=False,
        classes=parameter.selected_class
    )
```

Explanation- The YOLO model is used to predict objects in the current frame. The model takes several parameters, including the frame as the source, image size, confidence threshold, and the selected classes for detection.

```
    detection_boxes = results[0].boxes.data.tolist()
```

Explanation- The bounding boxes of detected objects are extracted from the prediction results and converted to a list.

```
    for box in detection_boxes:
        x1, y1, x2, y2, confidence, cls = int(box[0]), int(box[1]), int(box[2]), int(box[3]), int(box[4]), int(box[5])
```

Explanation- The loop iterates over each detected bounding box and extracts the coordinates (x1, y1, x2, y2), confidence score (confidence), and class index (cls) for each detected object.

```
normalized_width = (x2 - x1) / width
distance = parameter.distance_calculator(normalized_width)
```

Explanation- The normalized width of the object is calculated as the ratio of the object's width to the frame's width. Then, the distance_calculator function is called to calculate a distance value based on the normalized width.

```
label = f'**warning**' if distance <= parameter.distance_parameter else f'{class_names[int(cls)]}'
```

Explanation- A label is determined for the object. If the calculated distance is less than or equal to a predefined distance parameter, the label is 'warning', indicating a potential collision. Otherwise, it's set to the class name of the detected object.

```
cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255) if distance <= parameter.distance_parameter else (255, 125, 120), 2)
```

Explanation- A rectangle is drawn around the detected object on the frame. The color of the rectangle depends on the distance: red for a warning and another color (orange) for other objects. The last argument (2) specifies the thickness of the rectangle.

```
cv2.putText(frame, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255) if distance <= parameter.distance_parameter else (255, 125, 120), 1)
```

Explanation- A text label is added above the rectangle, displaying either 'warning' or the class name. The color and font size vary based on the distance, similar to the rectangle color.

```
cv2.imshow("Collision Detection", frame)
if cv2.waitKey(25) & 0xFF == ord('q'):
    break
```

Explanation- The current frame with object detection results is displayed in a window with the title "Collision Detection." The program waits for 25 milliseconds, and if the 'q' key is pressed, the loop is terminated.

```
video_capture.release()

cv2.destroyAllWindows()
```

Explanation- After processing all frames, the video capture is released and the OpenCV window is closed. This concludes the object detection on the video.

Distance Calculation -

For each detected object, the code calculates the normalized width of the object within the frame. This normalized width is used to estimate the distance of the object from the reference point. The closer an object is, the smaller its normalized width, and vice versa.

Collision Warning and Labeling:

Based on the calculated distance, the code determines whether to issue a collision warning. If the distance is less than or equal to a predefined distance parameter (specified in the parameter module), a warning label ("warning") is assigned to the object.