****

**MINI - PROJECT REPORT ON**

**“Real time Car Trajectory”**

**BY**

***Name: Enrollment No’s. :***

***Raashi Lokhande MITU22BTCS0165***

***Atharv Jadhav MITU22BTCS0181***

***Anshita Nair MITU22BTCS0127***

***Vaibhav Kadam MITU21BTCS0953***

**Under the Guidance of**

***Guide Name: Prof. Deepak Naik***

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**MIT SCHOOL of Engineering**

Loni Kalbhor Pune

**M.I.T. SCHOOL OF COMPUTING**

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

LONI – KALBHOR PUNE

***CERTIFICATE***



This is to certify that the Mini- Project report entitled

“**Car Trajectory Prediction**”

Submitted by

***Name: Enrollment No’s. :***

***Raashi Lokhande MITU22BTCS0165***

***Atharv Jadhav MITU22BTCS0181***

***Anshita Nair MITU22BTCS0127***

***Vaibhav Kadam MITU21BTCS0953***

Is a record of bonafide work carried out by them, under my guidance, in partial fulfillment of the requirement for the Second Year of Engineering (Computer) at M.I.T. School of Engineering, Pune under MIT Art, Design & Technology University

Date: 22/11/2024

**Prof.Deepak Naik**

**Guide,**

**Department of CSE**

**M.I.T. School of Engineering**

**Loni Kalbhor, Pune**

**ACKNOWLEDGEMENT**

We are very glad to present you this project entitled “Real time Car Trajectory”. This project has been made possible through the direct and indirect contribution of various people to whom we wish to express our gratitude.

We are very thankful to our project guide Prof. Mr. Avinash Ingle for supporting and guiding us in the project. We would also like to thank them for their valuable inspiration and for their cooperation during this project.

Last but not least, we are very thankful to all colleagues for their immense encouragement for the successful completion of the project.

We hope that the project will serve its purpose for which it is developed

Thereby underlining the success of the process.

**CONTENTS**

|  |  |
| --- | --- |
| **Sr. No.** | **Contents** |
| **1** | Introduction |
| **2** | Problem Definition |
| **3** | Features of the project |
| **4** | Platform/Technology |
| **5** | Flowchart |
| **6** | Literature Survey |
| **7** | Result |
| **8** | Code |
| **9** | Advantages |
| **10** | Limitations |
| **11** | Applications |
| **12** | Conclusion |
| **13** | Future Enhancements |
| **14** | References |

**INTRODUCTION**

* Artificial Intelligence (AI) has become a buzzword in the tech industry and is slowly making its way into various sectors, including the veterinary industry. AI has the potential to transform the way veterinarians diagnose, treat and monitor animal health. With the use of AI in the veterinary industry, veterinarians can now detect illnesses earlier, offer more precise diagnoses, and provide better treatment options for their patients. Through this project, we will be able to briefly explore possible solutions on how AI could transform the veterinary industry and the benefits it provides for veterinarians and their animal patients.
* The benefits of AI in the veterinary industry are numerous. AI can improve the accuracy and speed of diagnoses, leading to better treatment outcomes for animal patients. It can also help veterinarians detect illnesses earlier and prevent serious health issues. AI can provide insights into animal behavior, allowing veterinarians to provide more personalized care. Additionally, AI can help to speed up the drug discovery process, leading to the development of more effective treatments for animal patients.

**PROBLEM STATEMENT**

* The challenge is to effectively monitor the motion of objects over time and conduct behavior analysis. This is crucial for various applications such as surveillance, security, and research, where understanding the trajectories of objects is essential for pattern recognition and anomaly detection.

**FEATURES OF THE PROJECT**

* Object Detection and Tracking:
* Develop a system to detect and track objects in real-time.
* Utilize computer vision algorithms to identify and monitor the motion of objects within a given area.
* Trajectory Analysis:
* Collect and analyze the trajectories of objects over time.
* Calculate the speed, direction, and acceleration of the moving objects to understand their behavior.
* Pattern Recognition:
* Identify recurring patterns in the motion and behavior of objects.
* Develop algorithms to recognize and categorize these patterns for predictive analysis.

**Software Requirements:**

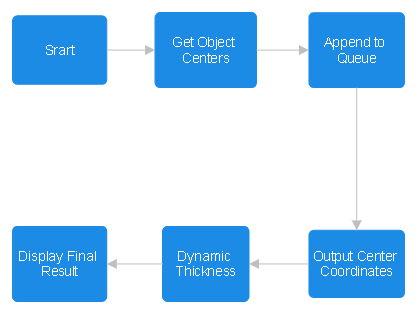
**Tools:-**

* VS Code, PyCharm
* TensorFlow, PyTorch
* OpenCV
* scikit-learn

**Language:-**

* Python
* Python is widely used in machine learning, computer vision, and data analysis, making it an excellent choice for implementing the algorithms and developing the monitoring system.

**FLOWCHART**



**LITERATURE SURVEY: Detail Survey Done**

* Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, 1998.
* This paper introduced the concept of using convolutional neural networks (CNN) for object detection and recognition, laying the foundation for modern deep learning-based object detection methods.
* Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-Learning-Detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2012.
* This paper presents the TLD (Tracking-Learning-Detection) algorithm, which combines object detection, tracking, and learning in a unified framework.

**Result:-**

****

**CODE**

import argparse

from typing import Dict, Iterable, List, Set

import cv2

import numpy as np

from tqdm import tqdm

from ultralytics import YOLO

import supervision as sv

COLORS = sv.ColorPalette.from\_hex(["#E6194B", "#3CB44B", "#FFE119", "#3C76D1"])

ZONE\_IN\_POLYGONS = [

np.array([[592, 282], [900, 282], [900, 82], [592, 82]]),

np.array([[950, 860], [1250, 860], [1250, 1060], [950, 1060]]),

np.array([[592, 582], [592, 860], [392, 860], [392, 582]]),

np.array([[1250, 282], [1250, 530], [1450, 530], [1450, 282]]),

]

ZONE\_OUT\_POLYGONS = [

np.array([[950, 282], [1250, 282], [1250, 82], [950, 82]]),

np.array([[592, 860], [900, 860], [900, 1060], [592, 1060]]),

np.array([[592, 282], [592, 550], [392, 550], [392, 282]]),

np.array([[1250, 860], [1250, 560], [1450, 560], [1450, 860]]),

]

class DetectionsManager:

def \_\_init\_\_(self) -> None:

self.tracker\_id\_to\_zone\_id: Dict[int, int] = {}

self.counts: Dict[int, Dict[int, Set[int]]] = {}

def update(

self,

detections\_all: sv.Detections,

detections\_in\_zones: List[sv.Detections],

detections\_out\_zones: List[sv.Detections],

) -> sv.Detections:

for zone\_in\_id, detections\_in\_zone in enumerate(detections\_in\_zones):

for tracker\_id in detections\_in\_zone.tracker\_id:

self.tracker\_id\_to\_zone\_id.setdefault(tracker\_id, zone\_in\_id)

for zone\_out\_id, detections\_out\_zone in enumerate(detections\_out\_zones):

for tracker\_id in detections\_out\_zone.tracker\_id:

if tracker\_id in self.tracker\_id\_to\_zone\_id:

zone\_in\_id = self.tracker\_id\_to\_zone\_id[tracker\_id]

self.counts.setdefault(zone\_out\_id, {})

self.counts[zone\_out\_id].setdefault(zone\_in\_id, set())

self.counts[zone\_out\_id][zone\_in\_id].add(tracker\_id)

if len(detections\_all) > 0:

detections\_all.class\_id = np.vectorize(

lambda x: self.tracker\_id\_to\_zone\_id.get(x, -1)

)(detections\_all.tracker\_id)

else:

detections\_all.class\_id = np.array([], dtype=int)

return detections\_all[detections\_all.class\_id != -1]

def initiate\_polygon\_zones(

polygons: List[np.ndarray],

triggering\_anchors: Iterable[sv.Position] = [sv.Position.CENTER],

) -> List[sv.PolygonZone]:

return [

sv.PolygonZone(

polygon=polygon,

triggering\_anchors=triggering\_anchors,

)

for polygon in polygons

]

class VideoProcessor:

def \_\_init\_\_(

self,

source\_weights\_path: str,

source\_video\_path: str,

target\_video\_path: str = None,

confidence\_threshold: float = 0.3,

iou\_threshold: float = 0.7,

) -> None:

self.source\_video\_path = source\_video\_path

self.target\_video\_path = target\_video\_path

self.conf\_threshold = confidence\_threshold

self.iou\_threshold = iou\_threshold

self.model = YOLO(source\_weights\_path)

self.tracker = sv.ByteTrack()

self.video\_info = sv.VideoInfo.from\_video\_path(source\_video\_path)

self.zones\_in = initiate\_polygon\_zones(ZONE\_IN\_POLYGONS, [sv.Position.CENTER])

self.zones\_out = initiate\_polygon\_zones(ZONE\_OUT\_POLYGONS, [sv.Position.CENTER])

self.bounding\_box\_annotator = sv.BoundingBoxAnnotator(color=COLORS)

self.label\_annotator = sv.LabelAnnotator(

color=COLORS, text\_color=sv.Color.BLACK

)

self.trace\_annotator = sv.TraceAnnotator(

color=COLORS, position=sv.Position.CENTER, trace\_length=100, thickness=2

)

self.detections\_manager = DetectionsManager()

def process\_video(self):

frame\_generator = sv.get\_video\_frames\_generator(

source\_path=self.source\_video\_path

)

if self.target\_video\_path:

with sv.VideoSink(self.target\_video\_path, self.video\_info) as sink:

for frame in tqdm(frame\_generator, total=self.video\_info.total\_frames):

annotated\_frame = self.process\_frame(frame)

sink.write\_frame(annotated\_frame)

else:

for frame in tqdm(frame\_generator, total=self.video\_info.total\_frames):

annotated\_frame = self.process\_frame(frame)

cv2.imshow("Processed Video", annotated\_frame)

if cv2.waitKey(1) & 0xFF == ord("q"):

break

cv2.destroyAllWindows()

def annotate\_frame(

self, frame: np.ndarray, detections: sv.Detections

) -> np.ndarray:

annotated\_frame = frame.copy()

for i, (zone\_in, zone\_out) in enumerate(zip(self.zones\_in, self.zones\_out)):

annotated\_frame = sv.draw\_polygon(

annotated\_frame, zone\_in.polygon, COLORS.colors[i]

)

annotated\_frame = sv.draw\_polygon(

annotated\_frame, zone\_out.polygon, COLORS.colors[i]

)

labels = [f"#{tracker\_id}" for tracker\_id in detections.tracker\_id]

annotated\_frame = self.trace\_annotator.annotate(annotated\_frame, detections)

annotated\_frame = self.bounding\_box\_annotator.annotate(

annotated\_frame, detections

)

annotated\_frame = self.label\_annotator.annotate(

annotated\_frame, detections, labels

)

for zone\_out\_id, zone\_out in enumerate(self.zones\_out):

zone\_center = sv.get\_polygon\_center(polygon=zone\_out.polygon)

if zone\_out\_id in self.detections\_manager.counts:

counts = self.detections\_manager.counts[zone\_out\_id]

for i, zone\_in\_id in enumerate(counts):

count = len(self.detections\_manager.counts[zone\_out\_id][zone\_in\_id])

text\_anchor = sv.Point(x=zone\_center.x, y=zone\_center.y + 40 \* i)

annotated\_frame = sv.draw\_text(

scene=annotated\_frame,

text=str(count),

text\_anchor=text\_anchor,

background\_color=COLORS.colors[zone\_in\_id],

)

return annotated\_frame

def process\_frame(self, frame: np.ndarray) -> np.ndarray:

results = self.model(

frame, verbose=False, conf=self.conf\_threshold, iou=self.iou\_threshold

)[0]

detections = sv.Detections.from\_ultralytics(results)

detections.class\_id = np.zeros(len(detections))

detections = self.tracker.update\_with\_detections(detections)

detections\_in\_zones = []

detections\_out\_zones = []

for zone\_in, zone\_out in zip(self.zones\_in, self.zones\_out):

detections\_in\_zone = detections[zone\_in.trigger(detections=detections)]

detections\_in\_zones.append(detections\_in\_zone)

detections\_out\_zone = detections[zone\_out.trigger(detections=detections)]

detections\_out\_zones.append(detections\_out\_zone)

detections = self.detections\_manager.update(

detections, detections\_in\_zones, detections\_out\_zones

)

return self.annotate\_frame(frame, detections)

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser(

description="Traffic Flow Analysis with YOLO and ByteTrack"

)

parser.add\_argument(

"--source\_weights\_path",

required=True,

help="Path to the source weights file",

type=str,

)

parser.add\_argument(

"--source\_video\_path",

required=True,

help="Path to the source video file",

type=str,

)

parser.add\_argument(

"--target\_video\_path",

default=None,

help="Path to the target video file (output)",

type=str,

)

parser.add\_argument(

"--confidence\_threshold",

default=0.3,

help="Confidence threshold for the model",

type=float,

)

parser.add\_argument(

"--iou\_threshold", default=0.7, help="IOU threshold for the model", type=float

)

args = parser.parse\_args()

processor = VideoProcessor(

source\_weights\_path=args.source\_weights\_path,

source\_video\_path=args.source\_video\_path,

target\_video\_path=args.target\_video\_path,

confidence\_threshold=args.confidence\_threshold,

iou\_threshold=args.iou\_threshold,

)

processor.process\_video()

**APPLICATIONS**

* Navigation Systems: Real-time trajectory systems are integral to GPS navigation in vehicles. They provide accurate positioning information and route guidance to drivers, helping them reach their destinations efficiently. These systems continuously update the vehicle's position and suggest the best routes based on real-time traffic conditions.
* Autonomous Vehicles: In autonomous vehicles, real-time trajectory systems play a crucial role in decision-making and path planning. These systems use sensor data, such as lidar, radar, and cameras, to perceive the vehicle's surroundings and generate safe trajectories to navigate through traffic and obstacles.
* Traffic Management: Real-time trajectory systems can be used by traffic management authorities to monitor and optimize traffic flow. By analyzing vehicle trajectories in real-time, authorities can identify congestion hotspots, adjust traffic signals, and implement dynamic route guidance to alleviate traffic congestion.
* Collision Avoidance Systems: Advanced driver assistance systems (ADAS) utilize real-time trajectory data to detect and prevent potential collisions. These systems use sensors and algorithms to analyze the relative trajectories of vehicles and other objects in the environment, issuing warnings or taking autonomous actions to avoid accidents.
* Fleet Management: In logistics and transportation companies, real-time trajectory systems are used to track the movements of vehicles in a fleet.
* Emergency Response: Real-time trajectory systems can assist emergency responders in navigating to accident or disaster sites quickly and efficiently. By analyzing real-time traffic conditions and generating optimal routes, emergency vehicles can reach their destinations faster, potentially saving lives.

**CONCLUSION**

In conclusion, real-time car trajectory systems play a pivotal role in modern transportation by providing accurate positioning information, optimizing route planning, enhancing safety, and improving overall efficiency. These systems are utilized in a wide range of applications, including navigation, autonomous vehicles, traffic management, collision avoidance, fleet management, emergency response, road condition monitoring, and urban planning.

By continuously analyzing vehicle trajectories and updating data in real-time, these systems enable drivers to navigate effectively, help authorities manage traffic flow, assist emergency responders in reaching destinations promptly, and support urban planners in designing sustainable transportation infrastructure.

As technology continues to advance, real-time car trajectory systems are expected to become even more sophisticated, integrating with emerging technologies such as artificial intelligence, machine learning, and vehicle-to-everything (V2X) communication systems to further enhance safety, efficiency, and convenience in the realm of transportation.Top of Form

**FUTURE ENHANCEMENTS**

The future of real-time car trajectory systems holds promising advancements that will revolutionize transportation in various ways. Here are some potential future enhancements and developments:

1. Integration with Autonomous Vehicles: As autonomous vehicle technology matures, real-time trajectory systems will become more tightly integrated into these vehicles' navigation and control systems. This integration will enable autonomous vehicles to make more informed decisions based on real-time traffic conditions, road hazards, and dynamic route changes.
2. Predictive Analytics: Future trajectory systems may incorporate predictive analytics algorithms to anticipate traffic patterns, congestion, and road conditions. By analyzing historical data and real-time inputs, these systems can provide more accurate predictions and proactive route recommendations to drivers and autonomous vehicles.
3. Vehicle-to-Infrastructure (V2I) Communication: Enhanced communication between vehicles and infrastructure, known as V2I communication, will enable real-time trajectory systems to receive data from roadside sensors, traffic signals, and other infrastructure elements. This bi-directional communication will provide more comprehensive information to vehicles, allowing for better traffic management and coordination.
4. Environmental Sensing: Real-time trajectory systems of the future may incorporate environmental sensing capabilities to detect air quality, weather conditions, and other environmental factors that affect driving conditions. This information can be used to optimize routes, reduce emissions, and enhance driver safety.
5. Dynamic Routing and Re-routing: Advanced algorithms will enable real-time trajectory systems to dynamically adjust routes in response to changing traffic conditions, accidents, or road closures. These systems will consider multiple factors, such as traffic flow, road capacity, and driver preferences, to provide optimal route recommendations in real-time.

REFERENCES

* <https://journals.sagepub.com/doi/abs/10.1177/0361198121990681?journalCode=trra>
* [https://www.hindawi.com/journals/jat/2021/8453726/#results-and-discussion](https://www.hindawi.com/journals/jat/2021/8453726/" \l "results-and-discussion)
* <https://ieeexplore.ieee.org/document/10005047>
* <https://github.com/ultralytics/ultralytics>