International Journal of Latest Research in Engineering and Technology (IJLRET)
ISSN: 2454-5031
www.ijlret.com || Volume 12 - Issue 10 || October 2026 || PP. 23-30

Dynamic Traffic-Aware Route Optimization Using Road-Type Priority, Edge-Based Map Analysis, and Google Maps APIs

Chintha Lokeswara Reddy[1], Vajrala Divya[2], Ankitha S[3],
Bhavana M Radder[4], Krupaprasad KR[5]
[1,2,3,4,5]*Dept. of Electronics & Communication Engineering*
Acharya Institute of Technology
Bengaluru, India

Abstract: Choosing a good route through a city is not a one-time decision, because traffic, travel time and the usefulness of side streets keep shifting while the trip is under way. This paper presents a dynamic routing system that pulls route and traffic data from Google Maps and layers a road-capacity priority model and an optional computer-vision step on top of it. The user enters a start and an end point, and the system then requests candidate driving routes together with their traffic-aware duration, distance, geometry and traffic information. Road segments are ranked with four-lane or high-capacity roads first, two-lane roads next, and one-way or narrow roads last. That ranking is blended with live congestion and estimated travel time, so a wide road that is badly jammed cannot win on its class alone. Where road-class information is missing, a map-image module estimates rough road geometry through grayscale conversion, Canny edge detection, Hough-line extraction, road-boundary separation and direction analysis. As the vehicle moves, its position is sent at intervals so that the routing data can be refreshed. Each route is scored by a weighted cost function covering traffic-aware travel time, congestion, distance, road priority and the cost of switching routes, and the active route is replaced only when an alternative beats it by a minimum margin, which keeps the route from flipping back and forth. The design brings real-time API use, computer vision, graph-based optimization, GPS tracking and ideas from intelligent transportation together in one system.

Index Terms: dynamic routing, intelligent transportation, Google Maps Routes API, traffic-aware polyline, road classification, Canny edge detection, Hough transform, route optimization.

I. Introduction

Cities are getting harder to drive in. Congestion is unpredictable, signals make traffic stop and wait, and roadworks, accidents, weather and rush-hour demand all add to the mix. The route that looks best when a driver sets off can turn out to be a poor one a few minutes later. A navigation system therefore has to keep watching the road network and change course whenever the expected gain justifies it.

Our system takes map and traffic data from Google Maps and builds its own optimization layer on top. Distance alone is not what it looks at; it also prefers roads with more capacity, putting wide four-lane roads first, two-lane roads second and narrow or one-way roads last. Current traffic still carries the greatest weight, so capacity is taken into account without sending every trip onto a highway.

We also include an image-processing idea, in which a map image is checked for edges and line directions so that road width and orientation can be estimated. A rendered map is only a drawing and not a surveyed measurement of the road, so this module is treated as supporting evidence only. Whenever structured route and traffic data are available, they take priority.

A. Contributions

This work offers six contributions: (1) a hybrid design that joins Google Maps API data with our own optimization; (2) a road-type priority model; (3) continuous refreshing of routes from traffic data; (4) an optional vision module for rough road analysis, whose reasoning can be inspected; (5) a weighted route cost with a hysteresis rule for rerouting; and (6) an implementation and evaluation plan suited to a real-time application.

II. Problem Definition, Scope, and Objectives

We are given an origin S, a destination D, a set of candidate routes R, the position P(t) of the moving vehicle, and traffic that varies with time. The aim is to pick the route r*(t) with the lowest expected journey cost, leaning toward higher-capacity roads when routes are close. Since traffic never stops changing, this choice must be made repeatedly during the trip.

A. Functional Objectives

1) Accept the source and destination from the user and read the current GPS location.
2) Request route alternatives, distance, traffic-aware duration, route polylines and any available traffic information from Google Maps services.
3) Work out a category or capacity score for each road from route attributes and, where needed, from edge and direction analysis of map imagery.
4) Rank road types as four-lane/highway > two-lane > narrow/one-way.
5) Combine live congestion, distance and ETA into a single overall route score.
6) Refresh route conditions at regular intervals, or when an event occurs, while the vehicle is moving.
7) Detect when the current route has become worse and switch to a better alternative only if the gain is meaningful.
8) Display the chosen route on the project's own interface and repeat until the vehicle arrives.

B. System Requirements

Category	Requirement
Input	Source, destination, current coordinates, travel mode
External data	Candidate routes, traffic- aware ETA, distance, polylines, traffic intervals
Vision input	Map image/snapshot at controlled zoom, if image analysis is enabled
Output	Optimized route, ETA, distance, traffic state, reroute notification
Real-time behavior	Periodic/event-driven refresh and rerouting
Reliability	Fallback to API route when visual classification confidence is low
Security	Restricted API keys, HTTPS, minimal storage of location history

III. Related Work

Shortest-path finding is usually posed on a weighted graph G=(V,E), where intersections are vertices and road segments are edges. Dijkstra's algorithm returns an optimal path when edge weights are fixed and nonnegative, and A* speeds up the search with an admissible heuristic that points toward the destination. In live navigation, however, the weight w(e,t) changes with time and represents the expected time to traverse a segment instead of its length.

Commercial routing services expose this time-varying information through APIs. Google's Routes API provides the Compute Routes and Compute Route Matrix operations, and it returns alternate routes, distance, duration, traffic-aware routing options and route polylines. Its traffic-aware polylines can label stretches of a route as normal, slow or jammed. With such features a system can evaluate routes over and over without maintaining a full road graph of its own.

Traditional lane and road-boundary detection depends on image gradients and simple geometric shapes. Canny edge detection is good at locating sharp intensity changes, and the Hough transform extracts lines from edge points. On a map image these methods can find long road boundaries and their direction, but labels, icons, zoom level and map styling can all create false edges. Because of this, our method pairs every visual result with a confidence value.

IV. Proposed System Architecture

The architecture separates sensing, communication with the map service, perception, optimization and presentation into distinct parts. This modular layout means the road classifier or the route provider can be replaced without redesigning the whole application.

Module	Responsibilities	Output
User/Map Interface	Collect origin/destination; show current and optimized routes	Trip request, map visualization
Location Manager	Read device GPS and detect movement/deviation	P(t), speed, heading
Maps API Connector	Compute routes and request traffic-aware fields	Candidate routes and traffic data
Route Parser	Decode polylines; segment route; normalize metrics	Route feature vectors
Image Analyzer	Canny edges, Hough lines, width/direction cues	Visual road-class confidence
Road Priority Engine	Assign capacity preference	Road penalty R(r)

Optimization Engine	Fuse ETA, congestion, distance, road class	Best route r*
Rerouting Controller	Apply improvement threshold/hysteresis	Keep/switch decision
Trip Logger	Store anonymized evaluation metrics	Experiment records

A. Data Flow

Step 1: the user supplies S and D. Step 2: the application requests traffic-aware driving routes. Step 3: the response is parsed into candidate routes. Step 4: each candidate is given traffic, distance and road-priority features. Step 5: the optimizer selects the candidate with the lowest cost. Step 6: that route is displayed. Step 7: as P(t) changes, the route is refreshed. Step 8: if congestion or ETA worsens, the active route is compared against new alternatives. Step 9: the active route is replaced only when the improvement threshold is met.

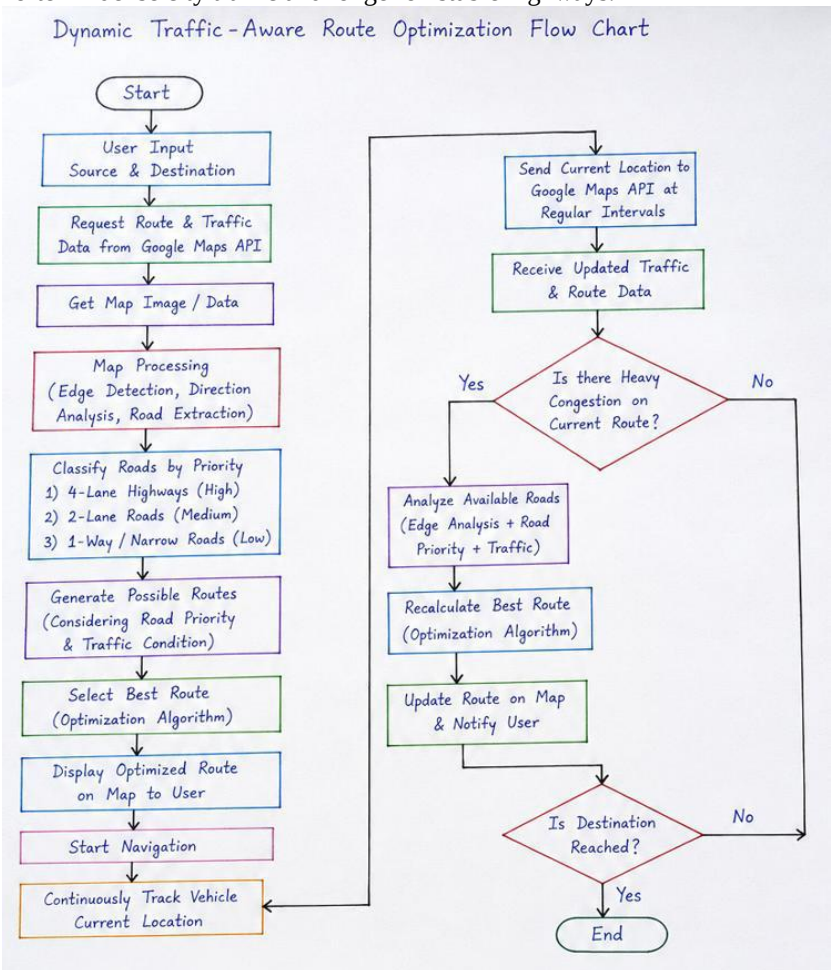
B. Google Maps Integration

The Google Maps Routes API currently provides three routing preferences: traffic-unaware, traffic-aware and traffic-aware-optimal. The optimal setting favors route quality and uses both live and historical traffic, whereas the plain traffic-aware setting gives up some quality in return for lower latency. A field mask lets the system request just the fields the optimizer needs: distance, duration, polyline and traffic intervals.

Traffic-aware polylines are useful because they link congestion to specific parts of the route geometry rather than giving a single ETA for the entire route. The application can then calculate what share of the route is in the NORMAL, SLOW and TRAFFIC_JAM states, and give more weight to congestion that still lies ahead of the vehicle.

C. Refresh Strategy

A fixed refresh interval is simple to build, but it wastes requests when the vehicle is standing still or traffic is steady. Our controller therefore combines two policies: a moderate periodic refresh and an immediate refresh when the vehicle leaves the route, the ETA rises sharply or a traffic jam appears ahead. The interval can also be adapted, shorter in dense city traffic and longer on stable highways.



V. Map Image Processing and Road-Type Estimation

The computer-vision part is intended to show how road geometry can be read from the visual structure of a map. We do not assume that a map screenshot gives a trustworthy lane count.

A. Preprocessing

Let I be the captured map image. It is converted to grayscale Ig and smoothed with a Gaussian filter Gσ to suppress small labels and pixel noise: Is = Gσ * Ig. Contrast normalization can be added if the map theme produces weak boundaries.

B. Edge Detection

Canny edge detection finds intensity gradients, applies nonmaximum suppression and uses two thresholds to produce a binary edge image E. The thresholds are tuned so that road outlines survive while text and small symbols are mostly removed. A region-of-interest mask can restrict processing to the corridor around a candidate route polyline.

C. Direction Analysis

The probabilistic Hough transform converts edge pixels into line candidates li=(x1,y1,x2,y2), and the orientation of each line is θi=atan2(y2−y1,x2−x1). Lines that have similar orientation and lie close together are grouped. The main direction of the route can also be taken from consecutive polyline coordinates and used to discard map edges unrelated to it.

D. Width Estimation

We examine pairs of roughly parallel boundary lines and measure their perpendicular separation di, converting it to a normalized width after correcting for map zoom. Using the median of the consistent separations makes the result less sensitive to outliers. A rough visual class of wide, medium or narrow is then assigned.

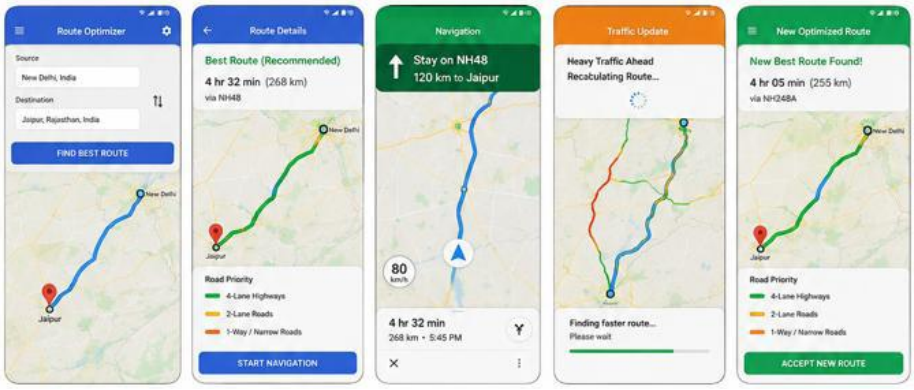
Wv = median(di), Classv = {Narrow if Wv<T1; Two-lane if T1≤Wv<T2; Wide if Wv≥T2}.

E. Confidence Fusion

The visual classifier returns a class along with a confidence cv. When cv is high, its road penalty counts fully; when cv is low, it is mixed with a neutral prior: Rfused = cv·Rvisual + (1−cv)·Rneutral. This stops a noisy screenshot from pushing the system toward a bad route choice.

F. Practical Challenges

Challenge	Effect	Mitigation
Map labels/icons	False edges and Hough lines	Route-corridor mask; morphology; length filtering
Zoom variation	Pixel width no longer equals road width	Fix zoom or calibrate thresholds by zoom
Curved roads	Straight Hough lines fragment geometry	Use short segments or contour/skeleton analysis
Intersections	Multiple orientations overlap	Segment route locally and use heading context
Overpasses	2-D map ambiguity	Prefer structured route data
Theme/style changes	Different contrast and colors	Use grayscale/gradient features and recalibration



VI. Dynamic Route Optimization Model

A. Feature Construction

For each candidate route r we compute five normalized features: T(r), the traffic-aware duration; C(r), the congestion severity; D(r), the distance; R(r), the road-type penalty; and S(r), a switching penalty. Normalization places these different units on a common [0,1] scale.

Congestion can be estimated from the traffic-aware polyline intervals. For example, $C(r)=0\cdot pn+0.5\cdot ps+1.0\cdot pj$, where pn, ps and pj are the fractions of the relevant route corridor that are normal, slow and jammed.

B. Road Priority

Road class	Priority	Illustrative penalty
Four-lane / high-capacity highway	1st	0.00
Two-lane / medium-capacity road	2nd	0.35
One-way / narrow local road	3rd	0.75
Unknown/low-confidence	Neutral	0.50

C. Objective Function

$$J(r)=\alpha T(r)+\beta C(r)+\gamma D(r)+\delta R(r)+\lambda S(r)$$

The best route is $r^*=\arg\min_{r\in R} J(r)$. The coefficients should be tuned through experiments. As a starting point we use $\alpha=0.45$, $\beta=0.25$, $\gamma=0.10$, $\delta=0.15$ and $\lambda=0.05$. This gives traffic-aware time the largest influence, while road class contributes a real but not overriding preference.

D. Rerouting Hysteresis

Without something to stabilize the decision, two routes with almost equal ETAs can swap places at every refresh. Let Jc be the cost of the current route and Jb the cost of the best alternative. A switch is accepted only if $Jc-Jb>\theta$ and/or the predicted time saving exceeds τ minutes. A lower, emergency threshold can be applied for serious traffic-jam events.

E. Pseudocode

Input S, D, refresh interval Δt, thresholds θ and τ.
Request traffic-aware alternatives R from S to D.
For each r in R: parse route features; estimate road class; compute J(r).
Set active route A to arg min J(r) and display A.
While destination not reached:
Read current position P(t).
If periodic refresh, deviation, ETA spike, or jam event occurs:
Request refreshed routes from P(t) to D.
Recompute all feature values and costs.
Let B be the minimum-cost alternative.
If improvement(A,B)>θ and timeSaving(A,B)>τ, set A←B.
Update route display, ETA, and traffic indicators.
Stop when arrival condition is satisfied.

VII. Implementation Framework

Layer	Suggested implementation
Client	Flutter mobile app or HTML/CSS/JavaScript web app
Map UI	Google Maps SDK / Maps JavaScript API
Backend	Python Flask or Django REST endpoints
Vision	OpenCV with Canny, Hough transform, morphology
Optimization	Python weighted scorer; optional NetworkX graph experiments
GPS	Mobile location service or browser Geolocation API
Database	SQLite/MySQL/MongoDB for trips, metrics, and settings
Deployment	HTTPS server with environment-based API key configuration

A. API Request/Response Handling

A route request contains the origin, destination, driving mode and a suitable traffic routing preference, and alternative routes are requested where the service supports them. The response parser extracts only the fields

that are needed. Encoded polylines are decoded so that they can be drawn and divided into local segments. Unrestricted server keys should never be exposed in a public client.

B. Application State

A navigation session tracks the current coordinates, the destination, the ID of the active route, the current ETA, the time of the last refresh, the time of the last accepted reroute, the route feature vector and the recent traffic state. A cooldown can be applied after a switch so that a small fluctuation does not immediately reverse the decision.

C. Failure Handling

If the network drops briefly, the application continues to show the last valid route and informs the user that traffic information may be out of date. If the vision module fails or returns low confidence, the optimizer falls back on metrics from the API alone. If no alternative route is returned, the current route stays active unless a new one can be computed from the vehicle's updated position.

D. Security and Privacy

Location data is sensitive. A production deployment should request location permission only while navigation is running, use TLS, retain data for as short a time as possible, avoid storing raw traces unless necessary, and state the reason for any data it keeps. API keys should be restricted to the allowed APIs and application identifiers, and server-side secrets belong in environment variables rather than in source code.

E. Complexity

Suppose k candidate routes are returned and each has m traffic or geometry intervals. Scoring on the application side then costs roughly O(km), which is small compared with network and API latency. Image processing costs between O(N) and O(N log N), depending on the operations used and the image size. The expensive network search is handled by the external routing service.

VIII. Test Plan

Testing is divided into unit, integration, vision, routing and field evaluation. Unit tests verify normalization and the cost calculations, while integration tests verify API parsing and error handling. Vision tests use manually labeled map samples. Routing tests replay GPS traces and traffic changes. Field tests compare the ordinary map route with our capacity-aware dynamic route.

IX. Experimental Evaluation and Expected Analysis

No accuracy or time-saving figures are reported here, since none have been measured yet. Once the system is implemented, it should be run on several source-destination pairs at different times of day, with each scenario repeated for a baseline configuration and for the proposed one.

Metric	**Formula / interpretation**
Travel time	Arrival time − departure time
ETA error	|ETA − observed remaining time| / observed remaining time ×100
Reroute gain	ETA immediately before reroute − ETA immediately after reroute
Congested-route share	Jam/slow distance divided by total route distance
Wide-road share	Distance on high-capacity road class / total distance
Reroute frequency	Accepted route switches per trip
Route stability	Mean time between accepted reroutes
Vision accuracy	Correct road classes / manually labeled samples
API latency	Response-ready timestamp − request timestamp

A. Suggested Experiment Matrix

Scenario	**Road environment**	**Traffic state**	**Comparison**
E1	Highway + arterial	Low	Baseline vs proposed
E2	Urban arterial + local	Moderate	Baseline vs proposed
E3	Urban arterial	Peak	Baseline vs proposed
E4	Current-route incident	Jam introduced	Reroute response
E5	Two similar alternatives	Fluctuating ETA	Stability/hysteresis
E6	Map-image samples	Mixed road widths	Vision classification

B. Results Table to Complete After Testing

Scenario	Baseline ETA	Proposed ETA	Actual Time	Reroutes	Saving
E1	—	—	—	—	—
E2	—	—	—	—	—
E3	—	—	—	—	—
E4	—	—	—	—	—
E5	—	—	—	—	—

C. Interpretation

A good system should not be judged by how often it picks a four-lane road, since the real goal is to arrive efficiently under current traffic. The proposed approach counts as an improvement when its route reduces actual or predicted travel time, avoids long stretches of congestion, uses suitable high-capacity roads where that helps, and does not switch routes too often.

Vision results should be reported separately from routing results. A road-width classifier may be less accurate at intersections or at zoom levels where road boundaries are stylized. Since confidence-weighted fusion is used, such errors have only a limited effect on the navigation decision.

X. Advantages, Limitations, and Applications

The main advantages are continuous adaptation, route scores that can be explained, a modular design, suitability for mobile and web applications, and a clear link between computer vision and intelligent transportation. Possible applications include ambulance navigation, logistics, delivery fleets, school and employee transport, tourist navigation and smart-city traffic dashboards.

The limitations are dependence on external map services, API quotas and billing, mobile-network quality and GPS error, and the fact that a generic map image cannot guarantee the actual lane count. Any lane or road type derived from an image should therefore be described as an estimate unless it has been checked against authoritative data.

XI. Future Enhancements

1) Learn the weights α, β, γ, δ and λ from historical trips instead of setting them by hand.
2) Add weather, toll cost, road-safety score, fuel consumption and vehicle type as further optimization objectives.
3) Apply reinforcement learning or contextual bandits so that route preference adapts to what trips actually produce.
4) Combine municipal GIS data or open road-network attributes with the visual road-width estimate.
5) Forecast congestion a few minutes ahead using time-series models or graph neural networks, rather than reacting only to present traffic.
6) Add emergency-vehicle priority and hospital-aware routing for ambulance use.
7) Offer an offline fallback route for periods when connectivity is lost.
8) Provide explainable notifications, for example 'rerouted: 8 min faster, heavy congestion ahead'.

XII. Conclusion

A navigation system that travels with the vehicle should keep re-examining how good its route is. This paper proposes a practical design that takes traffic-aware route data from Google Maps and adds our own optimization layer based on road capacity. The algorithm prefers four-lane or high-capacity roads, then two-lane roads, then narrow or one-way roads, while traffic-aware travel time and congestion still decide the outcome. An optional image-processing module based on Canny and Hough methods estimates rough road geometry from map imagery, and its influence depends on its confidence.

At every refresh, the vehicle's current position becomes the new origin, fresh alternatives are evaluated, and the best is chosen using the weighted objective function. Hysteresis and a minimum-saving threshold prevent needless oscillation between routes. The design can be built with a mobile or web interface, a Python backend, OpenCV, GPS and Google Maps Platform, and it offers a solid starting point for later work on predictive traffic modeling, multi-objective optimization and intelligent transportation.

References

[1]. Google for Developers, “Routes API,” Google Maps Platform, 2026.
[2]. Google for Developers, “Available traffic options,” Routes API, updated Jul. 20, 2026.
[3]. Google for Developers, “Set the level of traffic data,” Routes API, updated Jul. 20, 2026.
[4]. Google for Developers, “Request route polylines,” Routes API, updated Jul. 20, 2026.

[5]. Google for Developers, “Work with polylines,” Maps JavaScript API, updated Jul. 20, 2026.
[6]. Google for Developers, “Routes API Usage and Billing,” Google Maps Platform, 2026.
[7]. E. W. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Mathematik, vol. 1, pp. 269–271, 1959.
[8]. P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” IEEE Trans. Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.
[9]. J. Canny, “A computational approach to edge detection,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679–698, 1986.
[10]. R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures,” Communications of the ACM, vol. 15, no. 1, pp. 11–15, 1972.
[11]. D. B. Johnson, “Efficient algorithms for shortest paths in sparse networks,” Journal of the ACM, vol. 24, no. 1, pp. 1–13, 1977.
[12]. A. V. Goldberg and C. Harrelson, “Computing the shortest path: A* search meets graph theory,” Proc. ACM-SIAM SODA, 2005.
[13]. Google for Developers, “Get a route matrix,” Maps JavaScript API, 2026.
[14]. Google for Developers, “Available route options,” Routes API, updated Jul. 20, 2026.