RL-Based Traffic Light Timing Control

Buket Özbay - Istanbul Technical University

October 3, 2025

Abstract

This report describes a system that measures vehicle density (queue lengths) from a video source and controls traffic light phases using both a simple proportional policy and reinforcement learning (*Q-learning*). Detection is performed using the Ultralytics YOLOv5m network, tracking is provided via centroid-based matching, and inbound queues are extracted with ROI (lane regions) and direction filtering. The system can also publish counts via MQTT to a Pico 2 W client.

1 Introduction

Fixed-time traffic signal plans cannot adapt to varying traffic conditions. The goal of this study is to adjust traffic light phases based on *real-time* queue lengths measured from a single video source. **Contributions:** (i) Vehicle detection with YOLOv5m, (ii) Centroid tracking and hybrid cost (distance + 1-IoU) matching, (iii) Reliable inbound queue extraction via ROI + direction filter and two-frame confirmation, (iv) Phase switching decision using Q-learning, (v) MQTT integration with an embedded device (Pico 2 W).

2 Method

2.1 Preprocessing (Optional)

For some videos, a simple `dehaze` function based on the *Dark Channel Prior* can be applied to reduce fog/smoke effects (disabled by default due to computational cost).

2.2 Vehicle Detection

The Ultralytics YOLO (v8 library) loads the `yolov5m.pt` weights and extracts `car, truck, bus, motorcycle` classes. The `imgsz` input size, `conf`, and `iou` thresholds balance runtime and accuracy. Track window is shown in Figure 1.

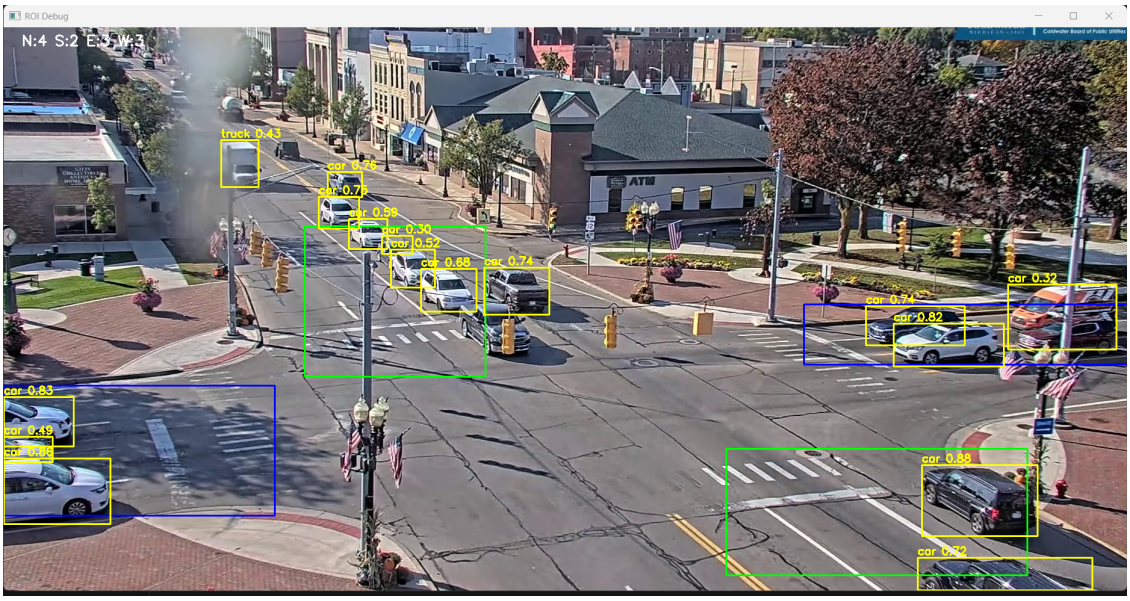

Figure 1: Track Window

2.3 Tracking and Matching

Detections in each frame are deduplicated (highest confidence kept). Track updates are made with a *hybrid cost*:

$$\text{cost} = w_{\text{dist}} \cdot \frac{d((c_x, c_y), (t_x, t_y))}{\text{diag}} + w_{\text{IoU}} \cdot (1 - \text{IoU}).$$

For tracks without a box, a small default box is used. Assignments below threshold are linked to the track, otherwise its `ttl` is reduced to manage its lifetime.

IoU & Deduplication

IoU:

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Among overlapping boxes in the same frame, only the one with the highest confidence is kept (IoU ≥ 0.7).

Hybrid Cost Parameters

Typical values are $w_{\text{IoU}} = 0.6$, $w_{\text{dist}} = 0.4$. diag is the image diagonal, used for distance normalization.

2.4 ROI, Direction Filter and Queue Generation

Each approach has user-defined rectangular ROIs (N,S,E,W). If a track's center lies within ROI and moves in the *correct direction*, it is considered a candidate. Candidates seen in *two consecutive frames* are added to the inbound queue (eliminating one-frame noise). Distance ordering is based on positive distance to the ROI edge. This step aims to interpret vehicles *approaching to stop at the gate* as *waiting vehicles*.

ROI-Based Direction Filter

For N, downward $(+y)$; for S, upward $(-y)$; for E, left $(-x)$; for W, right $(+x)$ motion is expected. A speed threshold (ε) prevents misclassification due to small jitter. The *Pending* set is updated with the intersection of $(lane, id)$ pairs for two-frame confirmation.

2.5 RL Environment and Agent

The reinforcement learning (RL) environment is modeled as a discrete-time Markov Decision Process (MDP), where the state, action, and reward components are explicitly defined.

State Space: The environment state is encoded as a four-dimensional tuple:

$$(\text{NS\_bin}, \text{WE\_bin}, \text{NS\_green}, \text{dur\_bin})$$

where NS_bin and WE_bin represent the discretized queue lengths for the North-South and East-West approaches, respectively. NS_green is a binary indicator of the current traffic light phase (1 for North-South green, 0 for East-West green). dur_bin denotes the discretized duration for which the current phase has been active. This state design allows the agent to capture both the instantaneous traffic density and the temporal persistence of a phase.

Action Space: The agent has two discrete actions: {0 : continue the current phase, 1 : switch to the opposite phase}. This simple action set captures the essential decision-making problem of traffic light control: whether to prolong the current phase or initiate a transition.

Reward Function: The reward at time t is defined as

$$r_t = (\text{waiting}_{t-1} - \text{waiting}_t) - 0.05 \cdot \mathbb{1}[\text{switch}] + 0.01 \cdot \text{moved},$$

where waiting_t denotes the total number of inbound vehicles at time t. The first term encourages a reduction in the queue length between consecutive steps. The second term penalizes unnecessary phase switching to discourage oscillatory behavior. The final term provides a small positive reward for throughput, incentivizing continuous traffic flow.

Learning Algorithm: The agent is trained with the `Q-learning` algorithm, an off-policy temporal-difference (TD) method. Action selection follows an ϵ-greedy exploration policy, gradually annealing ϵ to balance exploration and exploitation. The Q-values are updated iteratively based on observed state transitions.

Visualization and Simulation: To facilitate qualitative evaluation and debugging, the environment is coupled with a two-dimensional visualization built in `matplotlib`. The simulation displays:

- the intersection layout (lanes and road markings),
- the current traffic light states (red/green indicators),
- inbound and outbound vehicle queues represented as colored rectangles,
- textual overlays showing the current step, cumulative reward, exploration rate, and queue statistics.

This visual interface allows direct monitoring of the agent's decision-making process and provides an intuitive understanding of how traffic queues evolve over time under the learned control policy. An example simulation snapshot is presented in Figure 2.

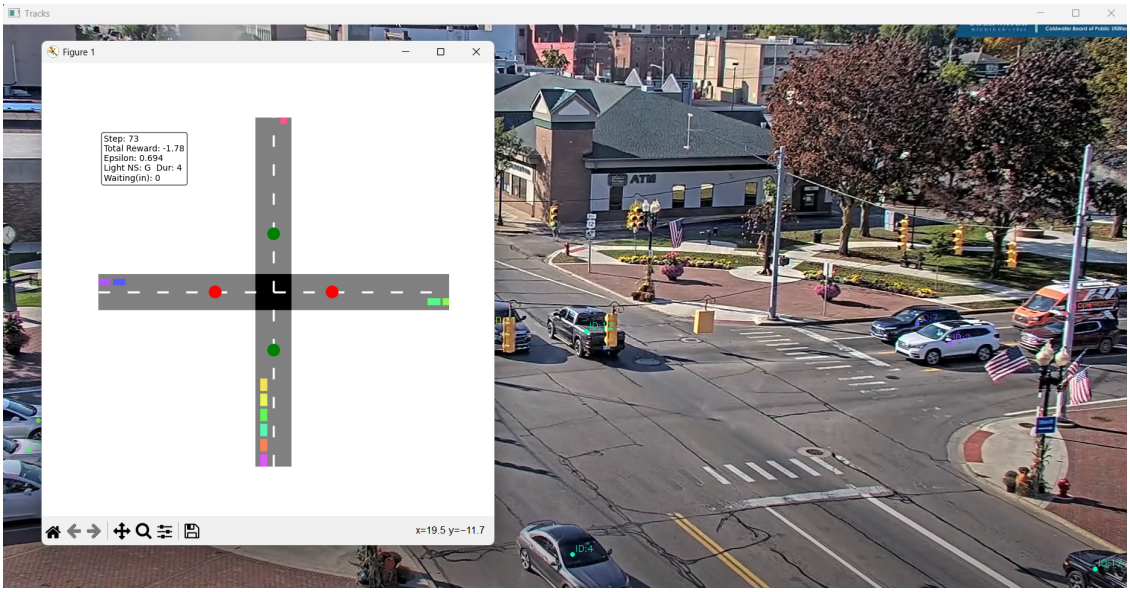

Figure 2: RL Parameters and Simulation

2.6 MQTT Publishing (Optional)

At each step, inbound counts of intersection directions (N,S,E,W) are published as JSON to the broker (mosquitto in this case); Pico 2 W subscribes via `umqtt.simple` and processes the counts. This is shown in Figure 3.

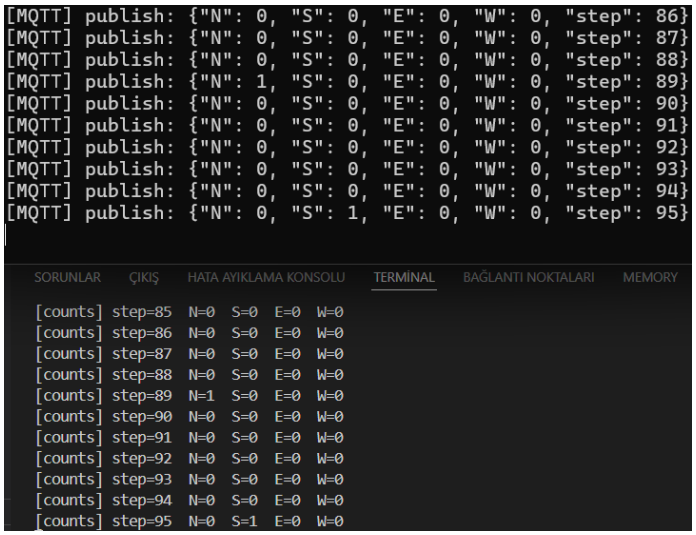

Figure 3: MQTT protocol between Pico 2 W (subscriber) and PC (publisher).

3 Experiments and Observations

Data: A single intersection camera video.

Method: The following steps were applied:

- **Vehicle Detection:** Classes `car, truck, bus, motorcycle` were extracted using YOLOv5m.
- **Tracking and Matching:** Vehicles were assigned persistent IDs using centroid-based tracking; hybrid cost (distance + IoU) was applied for frame-to-frame matching.
- **ROI and Direction Filter:** Only vehicles moving in the correct direction within user-defined ROIs (N,S,E,W) were considered; inbound queues were reliably generated with two-frame confirmation.
- **RL Environment:** Vehicle queues were taken as state, and the Q-learning agent learned traffic light phase switching. The reward function focused on reducing waiting time and improving flow.
- **MQTT Integration:** Obtained inbound queue counts were published in JSON format and subscribed by Raspberry Pi Pico 2W to be forwarded to external hardware.

Limitations: Single camera/parallax, detection may fail under harsh weather; DCP is costly; FPS limited on low hardware.

4 Conclusion and Future Work

This study demonstrates that queue-aware traffic signalization is possible using only a single camera, without requiring expensive sensor infrastructure. The reinforcement learning agent adapts to real-time traffic density by analyzing vehicle queues directly from video streams, showing that vision-based intelligent control is feasible in practice.

Future extensions are foreseen:

- Multi-camera fusion: Covering larger intersections and reducing occlusions by merging views from different angles.
- Advanced tracking with Kalman filter or DeepSORT: Combining appearance and motion models for more robust tracking against missed detections, overlaps, and occlusions.
- Adaptive inference resolution (imgsz): Dynamically adjusting YOLO input size depending on system load and traffic density to balance accuracy and latency.
- Edge deployment: Running the full system on embedded devices (e.g., Jetson Nano, Coral TPU, or Raspberry Pi) for real-time and decentralized smart traffic management.

Replication and Execution

Requirements

- Python 3.10 or higher
- Required libraries:
  - `ultralytics` (for YOLOv5 model)
  - `opencv-python` (for video processing)
  - `matplotlib` (for visualization and animation)
  - `numpy`, `collections` (numerical operations and data structures)
  - `paho-mqtt` (for MQTT publishing)
  - `micropython-umqtt.simple` (for MQTT subscription, will be uploaded to Pico 2 W)
- YOLO weight file: `yolov5m.pt`
- MQTT broker (optional): `mosquitto` recommended.

Installation

```
pip install ultralytics opencv-python matplotlib numpy paho-mqtt
```

If a CUDA-enabled GPU is available, YOLO inference will run on GPU, otherwise CPU will be used.

Steps

1. Download YOLO weight: `yolov5m.pt`. Obtain from Ultralytics repo or official sources.
2. Place the traffic video in the project directory and update the `VIDEO` path in the code.
3. To test the RL environment independently:

   ```
   python traffic_light_control_RL_OpenCV.py
   ```

   This launches YOLO detection, Q-learning agent, and visualization.
4. To send data to Pico 2 W via MQTT:

   ```
   python mqtt_publisher.py
   ```

   This publishes inbound queue counts in JSON format to the `traffic/counts` topic at each step.
5. On Pico 2 W, run `mqtt_client_pico.py`, which connects to WiFi, subscribes to `traffic/counts`, and prints incoming counts over USB serial with the help of `umqtt.simple` library of `Micropython`.

Notes

- ROI coordinates (`roi_N, roi_S, roi_E, roi_W`) must be manually adjusted to match the video scene.
- Parameters `imgsz`, `conf`, `iou` can be tuned for speed/accuracy trade-off.
- If the MQTT broker runs on the local machine (`localhost`), ensure Pico can access the broker's IP.