PROBLEM STATEMENT

And use Google earth to find example of people parking in front of fire hydrants in the city of Somerville MA and Boston MA submit a small example to demonstrate you understand the task

OVERVIEW

We have used this script to get the fire hydrant in both areas of interest what we have used is OverpassAPI

```
!pip install requests simplekml

import requests, simplekml

def get_hydrants_from_overpass(bbox):
    south, west, north, east = bbox
    query = f"""
    [out:json];
    node
      ["emergency"="fire_hydrant"]
      ({south},{west},{north},{east});
    out;
    """
    url = "https://overpass-api.de/api/interpreter"
    r = requests.get(url, params={"data": query})
    data = r.json()
    return [
        {"lat": el["lat"], "lon": el["lon"], "tags": el.get("tags", {})}
        for el in data.get("elements", [])
    ]

# Bounding boxes
somerville_bbox = (42.38, -71.12, 42.41, -71.08)
boston_bbox     = (42.33, -71.12, 42.36, -71.05)  # Back Bay/Downtown

# Query hydrants
somerville_hydrants = get_hydrants_from_overpass(somerville_bbox)
boston_hydrants     = get_hydrants_from_overpass(boston_bbox)

print(f"Somerville: {len(somerville_hydrants)} hydrants")
print(f"Boston: {len(boston_hydrants)} hydrants")

# Create single KML
kml = simplekml.Kml()

# Add Somerville hydrants
som_folder = kml.newfolder(name="Somerville Hydrants")
for h in somerville_hydrants:
    desc = ", ".join([f"{k}={v}" for k,v in h["tags"].items()])
    som_folder.newpoint(
        name="Hydrant",
        coords=[(h["lon"], h["lat"])],
        description=desc
    )

# Add Boston hydrants
bos_folder = kml.newfolder(name="Boston Hydrants")
for h in boston_hydrants:
    desc = ", ".join([f"{k}={v}" for k,v in h["tags"].items()])
    bos_folder.newpoint(
        name="Hydrant",
        coords=[(h["lon"], h["lat"])],
        description=desc
    )
```

Now as we have the KML file ready.
we have imported the file into google earth pro.
checked the accuracy by zooming to all the fire hydrants.

Next Step

Using the Street View Static API from google and enable it
Once we will have the API key, we can use that to:

- Take each hydrant's coordinates from your KML.
- Query the Street View API.
- Get back an image file of the view from ground level.

This way we can even batch-download all hydrant views and later check if cars are blocking them.

Instead of using the API key to fetch all the Streetview images, We have taken the

screen shots of street view as a sample and applied Yolo Pre-trained data which can detect the hydrants, cars etc. Now we are extracting the bounding boxes of the given classes for example car=2 and Hydrant = 10.

Purpose of the Code

The script detects **fire hydrants and cars** in images and determines if a hydrant is **blocked by a car**.
It uses **YOLO (Ultralytics YOLOv8)** for object detection and **OpenCV / NumPy** for processing masks, bounding boxes, and overlaying results.

Object Extraction

def get_box_and_mask(result, class_id):

- **Input:** YOLO detection results & a class ID (hydrant or car).
- **Process:**
    - Loops through YOLO detections.
    - Filters objects of the given class.
    - Extracts **bounding box coordinates** (x1, y1, x2, y2).
    - Computes **centroid** for each object.
    - Retrieves **segmentation mask** if available.
- **Output:** List of objects with box, mask, and centroid.

Determine if Hydrant is Blocked

def hydrant_blocked(hydrants, cars, ...)

- **Input:** Hydrant and car objects.
- **Logic:**
    1. For each hydrant, check against **all cars**.

2. **Vertical alignment check:** The car's bottom must be below the hydrant centroid.
3. **Mask overlap (if available):**
    - Convert hydrant and car masks to binary.
    - Calculate **intersection** of masks.
    - If overlap_ratio >= cover_thresh, mark as blocked.
4. **Bounding box fallback:**
    - Calculate horizontal overlap fraction.
    - Check proximity threshold.
    - If either condition is satisfied and vertically aligned, mark as blocked.

- **Output:** List of booleans: True if blocked, False if clear.

Draw Results

def draw_overlay(img, hydrants, cars, blocked_status)

- Draws **bounding boxes** on the image:
    - **Green boxes** for cars.
    - **Red boxes** for blocked hydrants.
    - **Blue boxes** for clear hydrants.
- Adds text labels: "Hydrant 0 BLOCKED" or "Hydrant 1 CLEAR".
- Returns the annotated image.

Main Processing

def process_image(model, img_path)

- Reads the image.
- Runs **YOLO model prediction**.
- Extracts hydrants and cars.
- Checks blocked status.
- Draws overlay.
- Returns the annotated image and blocked status.

Script Execution

```
if __name__ == "__main__":
    model = YOLO("yolov8s.pt")
    for img_file in os.listdir(INPUT_FOLDER):
        ...
```

- Loads the YOLOv8 model.
- Loops through all images in the input folder.
- Calls process_image() for detection & annotation.
- Prints which hydrants are blocked:
- 🚨 Blocked: image.jpg — hydrants [0, 2]
- ✅ Clear: image2.jpg
- Saves annotated images to OUTPUT_FOLDER.

Summary of the Logic

1. **Detect objects** → YOLO finds hydrants and cars.
2. **Extract boxes & masks** → Get geometric info for analysis.
3. **Check blockage** → Use mask overlap + bounding box rules.
4. **Draw results** → Annotate image with clear/block info.
5. **Save & report** → Output images & print which hydrants are blocked.

```python
def get_box_and_mask(result, class_id):
    """Extract bounding boxes & masks for a given class_id."""
    objects = []
    masks = result.masks.data.cpu().numpy().astype(bool) if result.masks is not None else [None] * len(result.boxes)

    for i, box in enumerate(result.boxes):
        cls = int(box.cls)
        if cls != class_id:
            continue
        x1, y1, x2, y2 = map(int, box.xyxy[0].cpu().numpy())
        centroid = ((y1 + y2) // 2, (x1 + x2) // 2)
        mask = masks[i] if masks[i] is not None else None
        objects.append({
            "box": (x1, y1, x2, y2),
            "mask": mask,
            "centroid": centroid
        })
    return objects
```

```python
def hydrant_blocked(hydrants, cars, use_masks=True, cover_thresh=0.15, horiz_thresh=0.3, prox_thresh=30):
    """Determine if hydrants are blocked by cars."""
    results = [False] * len(hydrants)

    for hi, h in enumerate(hydrants):
        hx1, hy1, hx2, hy2 = h["box"]
        h_w, h_h = hx2 - hx1, hy2 - hy1
        h_cy = (hy1 + hy2) / 2
        pad = 20
        hx1e, hy1e, hx2e, hy2e = hx1 - pad, hy1 - pad, hx2 + pad, hy2 + pad

        for c in cars:
            cx1, cy1, cx2, cy2 = c["box"]
            blocked = False
            vertical_ok = cy2 >= h_cy

            # Mask overlap check
            if use_masks and h.get("mask") is not None and c.get("mask") is not None:
                h_mask = h["mask"].astype(np.uint8)
                c_mask = c["mask"].astype(np.uint8)
                inter = cv2.bitwise_and(h_mask, c_mask)
                overlap_ratio = inter.sum() / (h_mask.sum() + 1e-6)
                if overlap_ratio >= cover_thresh:
                    blocked = True

            # Bounding box overlap fallback
            if not blocked:
                overlap_w = max(0, min(hx2e, cx2) - max(hx1e, cx1))
                horiz_frac = overlap_w / (h_w + 1e-6)
                horiz_gap = min(abs(cx1 - hx2e), abs(hx1e - cx2))
                if (horiz_frac >= horiz_thresh and vertical_ok) or (horiz_gap <= prox_thresh and vertical_ok):
                    blocked = True

            if blocked:
                results[hi] = True
                break

    return results
```

```python
def draw_overlay(img, hydrants, cars, blocked_status):
    """Draw results on image."""
    out = img.copy()

    # Draw cars
    for c in cars:
        x1, y1, x2, y2 = c["box"]
        cv2.rectangle(out, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.putText(out, "Car", (x1, y1 - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Draw hydrants
    for idx, (h, blocked) in enumerate(zip(hydrants, blocked_status)):
        x1, y1, x2, y2 = h["box"]
        color = (0, 0, 255) if blocked else (255, 0, 0)
        label = f"Hydrant {idx} BLOCKED" if blocked else f"Hydrant {idx} CLEAR"
        cv2.rectangle(out, (x1, y1), (x2, y2), color, 2)
        cv2.putText(out, label, (x1, y1 - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    return out
```

```python
def process_image(model, img_path):
    """Run detection and overlay for a single image."""
    img = cv2.imread(img_path)
    if img is None:
        raise ValueError(f"Could not load {img_path}")

    results = model.predict(img, verbose=False)[0]

    hydrants = get_box_and_mask(results, HYDRANT_CLASS)
    cars = get_box_and_mask(results, CAR_CLASS)

    blocked_status = hydrant_blocked(hydrants, cars)
    out = draw_overlay(img, hydrants, cars, blocked_status)

    return out, blocked_status
```

Input Images

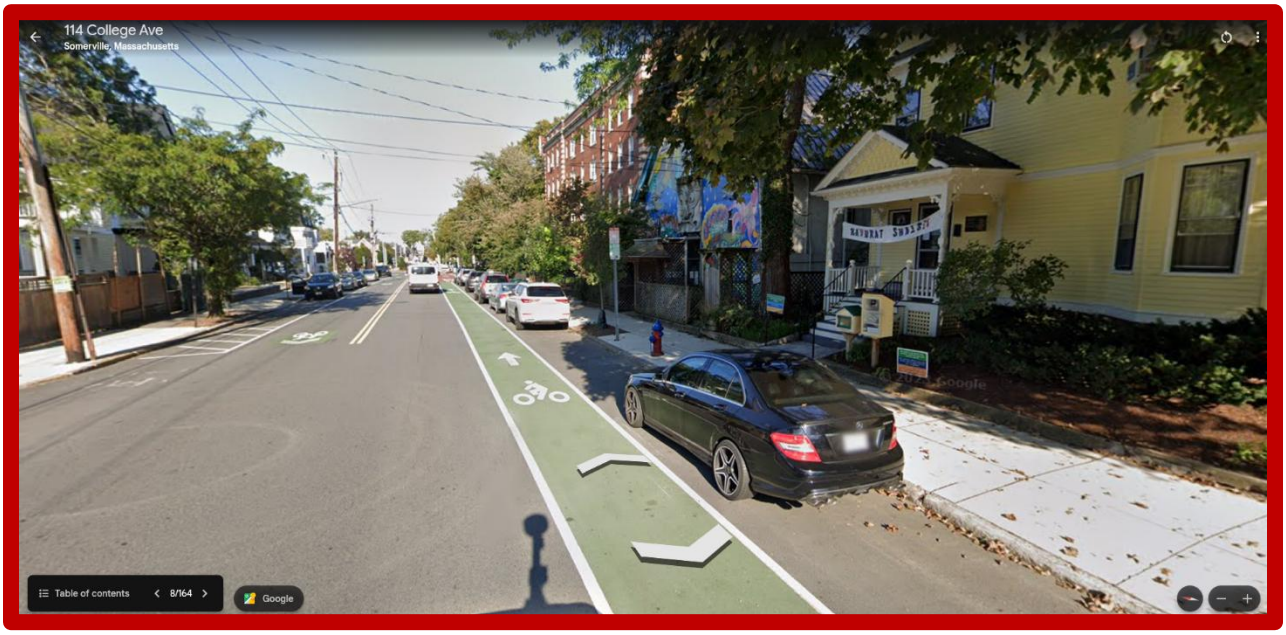

12 Mallet St
Somerville, Massachusetts
Lowden Ave
Table of contents
13/164
Google


8 Lowden Ave
Somerville, Massachusetts
Table of contents
12/164
Google


78 Albion St
Medford, Massachusetts
Albion St
© 2022 Google
Table of contents
11/164
Google


80 College Ave
Somerville, Massachusetts
20
Table of contents
10/164
Google


98 College Ave
Somerville, Massachusetts
College Ave
Table of contents
9/164
Google


114 College Ave
Somerville, Massachusetts
Table of contents
8/164
Google


138 College Ave
Somerville, Massachusetts
Table of contents
7/164
Google


Upper Campus Rd
Medford, Massachusetts
Table of contents
6/164
Google


Tufts University
Table of contents
5/164
Brandon Toms


N Hill Rd
Medford, Massachusetts
Table of contents
4/164
Google


2 Liberty Ave
Somerville, Massachusetts
Liberty Ave
Table of contents
1/164
Google


12 Craigie St
Somerville, Massachusetts


287 Summer St
Somerville, Massachusetts
Table of contents
25/164
Google


138 Willow Ave
Somerville, Massachusetts
Willow Ave
Table of contents
24/164
Google


55 Morrison Ave
Somerville, Massachusetts
Exit Street View
LOW ZONE
NO
PARKING
ANY TIME
Table of contents
23/164
Google

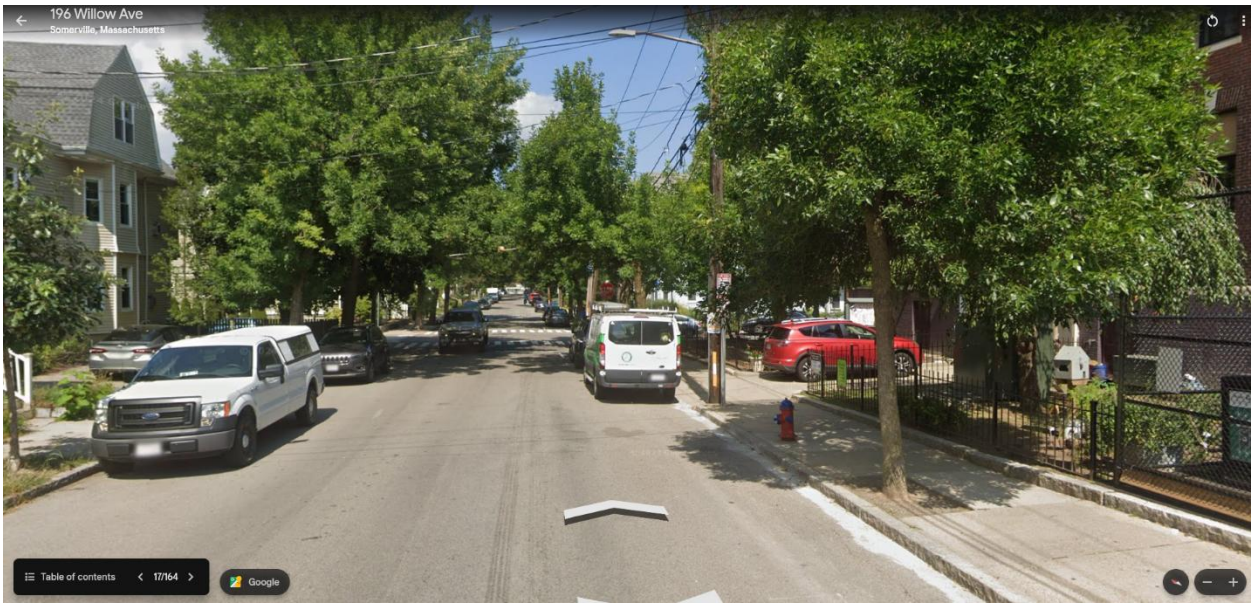

92 Liberty Ave
Somerville, Massachusetts
Table of contents
22/164
Google


112 Liberty Ave
Somerville, Massachusetts
Exit Street View
Hall Ave
Table of contents
21/164
Google


19 Foskett St
Somerville, Massachusetts
Foskett St
Table of contents
20/164
Google

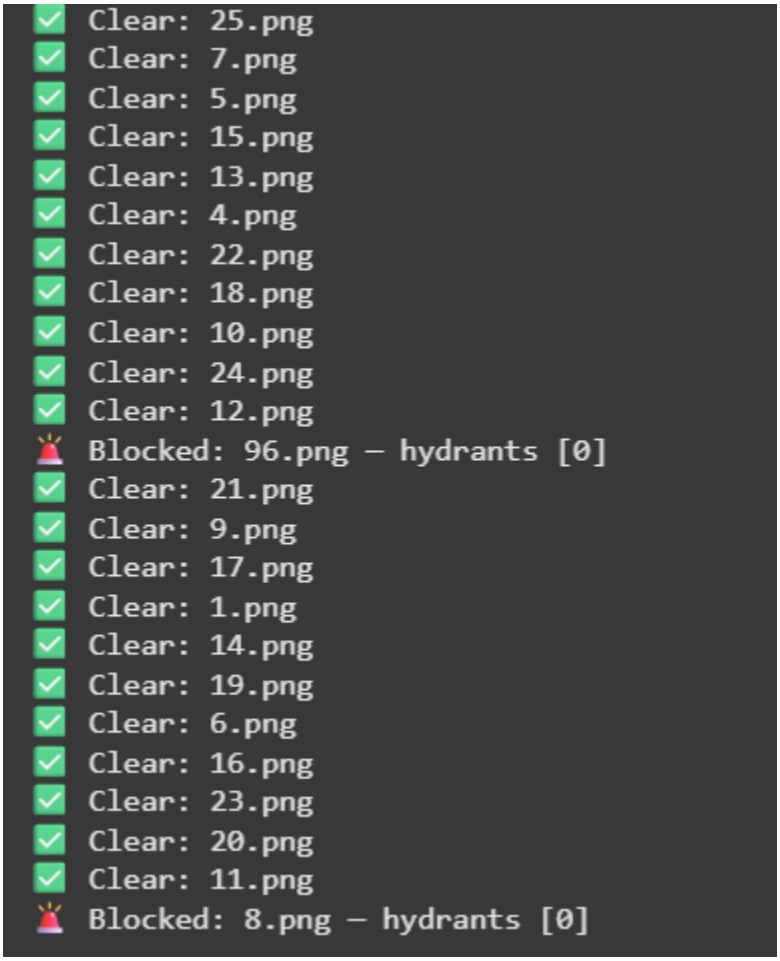

179 Willow Ave
Somerville, Massachusetts
Exit Street View
Willow Ave
Table of contents
19/164
Google


35 Josephine Ave
Somerville, Massachusetts
Exit Street View
Josephine Ave
Table of contents
18/164
Google


196 Willow Ave
Somerville, Massachusetts
Table of contents
17/164
Google


18 Powder House Terrace
Somerville, Massachusetts
Table of contents
16/164
Google

Results


✅ Clear: 25.png
✅ Clear: 7.png
✅ Clear: 5.png
✅ Clear: 15.png
✅ Clear: 13.png
✅ Clear: 4.png
✅ Clear: 22.png
✅ Clear: 18.png
✅ Clear: 10.png
✅ Clear: 24.png
✅ Clear: 12.png
🚨 Blocked: 96.png — hydrants [0]
✅ Clear: 21.png
✅ Clear: 9.png
✅ Clear: 17.png
✅ Clear: 1.png
✅ Clear: 14.png
✅ Clear: 19.png
✅ Clear: 6.png
✅ Clear: 16.png
✅ Clear: 23.png
✅ Clear: 20.png
✅ Clear: 11.png
🚨 Blocked: 8.png — hydrants [0]