Spatialyze: A Geospatial Video Analytics System with Spatial-Aware Optimizations

EPIC DATA lab

Berkeley
UNIVERSITY OF CALIFORNIA

Chanwut (Mick) Kittivorawong | Yongming Ge | Yousef Helal | Alvin Cheung

EPIC Retreat — April 15, 2024

Overview

#1 Domain Challenges

- Motivation
- Usage Scenario
- Typical Workflow

#2 System Interface

- Data Model
- Programming Model

#3 System Optimization

- Video Processor
- Optimization Techniques

#4 Evaluation

- Experiment Setups
- Results

Overview

#1 Domain Challenges

- Motivation
- Usage Scenario
- Typical Workflow

#2 System Interface

- Data Model
- Programming Model

#3 System Optimization

- Video Processor
- Optimization Techniques

#4 Evaluation

- Experiment Setups
- Results

Overview

#1 Domain Challenges

- Motivation
- Usage Scenario
- Typical Workflow

#2 System Interface

- Data Model
- Programming Model

#3 System Optimization

- Video Processor
- Optimization Techniques

#4 Evaluation

- Experiment Setups
- Results

Overview

#1 Domain Challenges

* Motivation
* Usage Scenario
* Typical Workflow

#2 System Interface

* Data Model
* Programming Model

#3 System Optimization

* Video Processor
* Optimization Techniques

#4 Evaluation

* Experiment Setups
* Results

SPATIALYZE

#1 Domain Challenges

* Motivation
* Usage Scenario
* Typical Workflow

Motivation

What are Geospatial Videos?

Motivation

Geospatial Videos

Autonomous Vehicle

Traffic Camera

Drone Camera

Motivation

Geospatial Videos

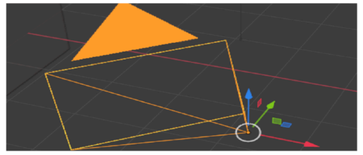
Autonomous Vehicle

Traffic Camera

Drone Camera

Road Network

Camera Location / Rotation



Comes with geospatial metadata

Motivation

Autonomous Vehicle

Traffic Camera

Drone Camera

A data journalist investigates the behaviors of autonomous driving vehicles when each of them sees cars crash at an intersection.

Motivation

Autonomous Vehicle

Traffic Camera

Drone Camera

A traffic analyst wants to determine when is the busiest hours of a traffic intersection based on recorded traffic camera footages.

Motivation

Autonomous Vehicle

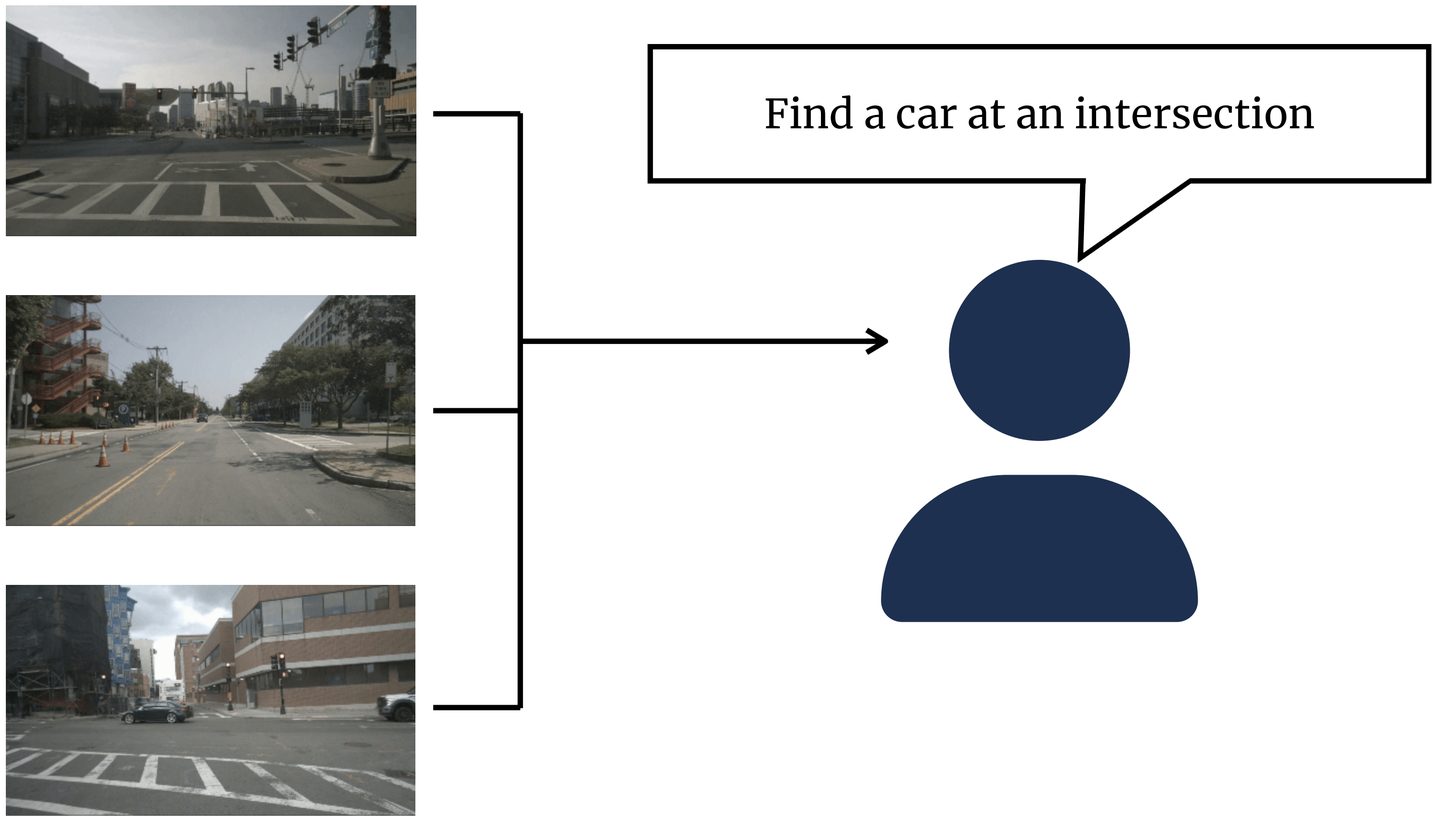
Traffic Camera

Drone Camera

An engineer wants to develop an application that reports realtime parking spots availability from drone camera footages

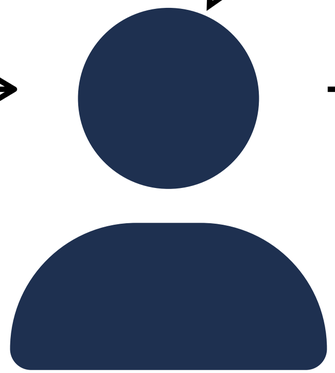
Usage Scenario

Usage Scenario

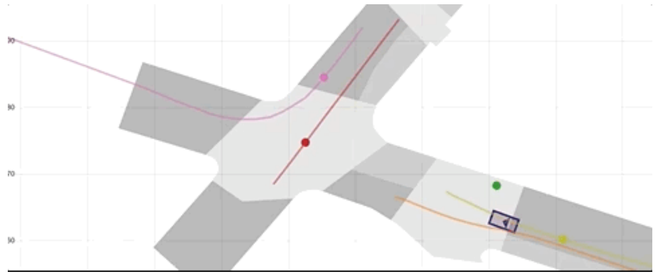


Usage Scenario

Find a car at an intersection

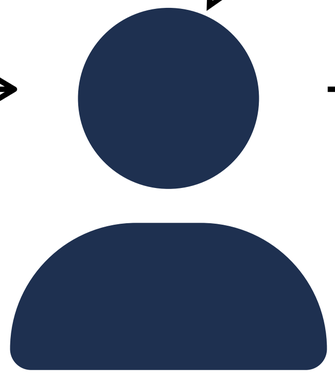


Analyze / Visualize

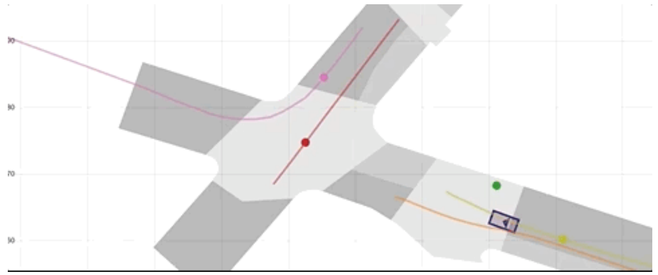


Usage Scenario

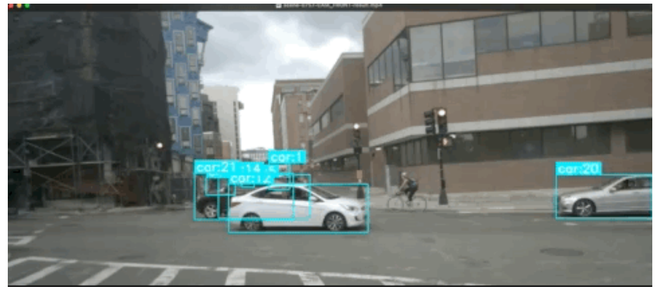
Find a car at an intersection



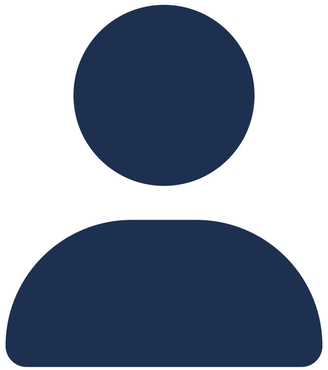
Analyze / Visualize



Annotate / Watch

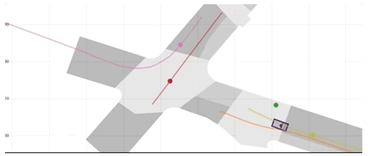


Typical Workflow (Manual)



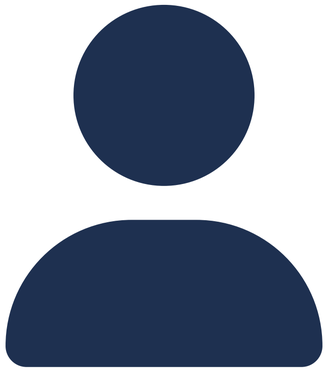
Manually watch and annotate

Analyze / Visualize



Annotate / Watch

Typical Workflow (Manual)

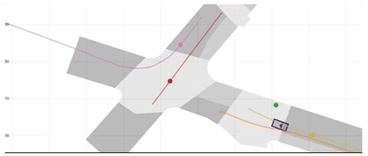


Manually watch and annotate

⏳

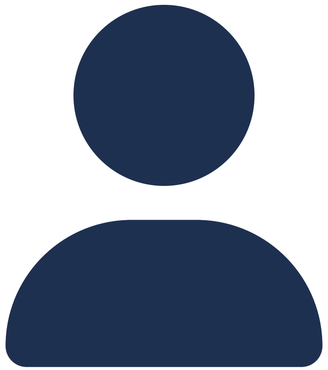
Time-consuming

Analyze / Visualize



Annotate / Watch

Typical Workflow (Manual)

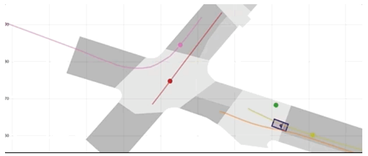


Manually watch and annotate

⏳ Time-consuming

😫 Tedious & Error-prone

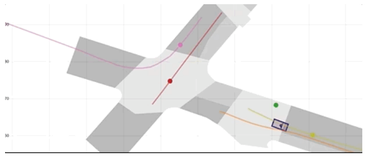
Analyze / Visualize



Annotate / Watch

Typical Workflow (Programming)

Analyze / Visualize



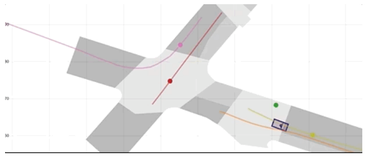
Annotate / Watch

Typical Workflow (Programming)

Object Detector

```
with torch.no_grad():
    _names = self.model.names

    assert isinstance(_names, dict), type(_names)
    names: list[str] = class_mapping_to_list(_names
    self.model.eval()

    assert isinstance(self.imgsz, list), type(self.
    self.model.warmup(imgsz=(1, 3, *self.imgsz))  #

    for frame_idx, im0 in enumerate(self.frames.str
        if isinstance(im0, Skip):
            yield skip
            continue

        im, _, _ = letterbox(im0, self.imgsz, strid
        im = im.transpose((2, 0, 1))[::-1]  # HWC to
        im = np.ascontiguousarray(im)  # contiguous

        # t1 = time_sync()
        im = torch.from_numpy(im).to(self.device)
        im = im.half() if self.half else im.float()
        im /= 255.0  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

        # Inference
        pred = self.model(im, augment=self.augment)

        # Apply NMS
        pred = non_max_suppression(
            pred,
            self.conf_thres,
            self.iou_thres,
            self.classes,
            self.agnostic_nms,
            max_det=self.max_det,
        )

        # Process detections
        assert isinstance(pred, list), type(pred)
        assert len(pred) == 1, len(pred)
        det = pred[0]
        assert isinstance(det, torch.Tensor), type(
        det[:, :4] = scale_boxes(im.shape[2:], det[
        yield Detection2D(
            det, names, [DetectionId(frame_idx, orde
            )
```

Analyze / Visualize



Annotate / Watch

Typical Workflow (Programming)

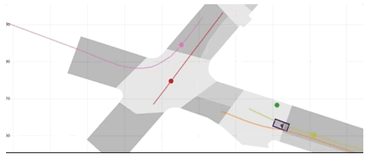
Object Detector

```
with torch.no_grad():
    _names = self.model.names

    assert isinstance(_names, dict), type(_names)
    names: list[str] = class_mapping_to_list(_names
    self.model.eval()

    assert isinstance(self.imgsz, list), type(self.
    self.model.warmup(imgsz=(1, 3, *self.imgsz))  #

    for frame_idx, im0 in enumerate(self.frames.str
        if isinstance(im0, Skip):
            yield skip
            continue

        im, _, _ = letterbox(im0, self.imgsz, stride
        im = im.transpose((2, 0, 1))[::-1]  # HWC to
        im = np.ascontiguousarray(im)  # contiguous

        # t1 = time_sync()
        im = torch.from_numpy(im).to(self.device)
        im = im.half() if self.half else im.float()
        im /= 255.0  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

        # Inference
        pred = self.model(im, augment=self.augment)

        # Apply NMS
        pred = non_max_suppression(
            pred,
            self.conf_thres,
            self.iou_thres,
            self.classes,
            self.agnostic_nms,
            max_det=self.max_det,
        )

        # Process detections
        assert isinstance(pred, list), type(pred)
        assert len(pred) == 1, len(pred)
        det = pred[0]
        assert isinstance(det, torch.Tensor), type(
        det[:, :4] = scale_boxes(im.shape[2:], det[
        yield Detection2D(
            det, names, [DetectionId(frame_idx, ord
        )
```

3D Location Estimator

```
class monodepth:
    def __init__(self, model_name=MODEL_NAME[2], no_cuda=False, pred_metric_depth=True) -> None:
        assert model_name in MODEL_NAME, "Invalid Model Name"

        if torch.cuda.is_available() and not no_cuda:
            self.device = torch.device("cuda")
            # print("GPU Visible")
        else:
            self.device = torch.device("cpu")
            print("GPU not visible. CPU mode")

        if pred_metric_depth and "stereo" not in model_name:
            print(
                "Warning: The pred_metric_depth flag only makes sense for stereo-trained KITTI "
                "models. For mono-trained models, output depths will not in metric space."
            )

        download_model_if_doesnt_exist(model_name=model_name)
        model_path = os.path.join(manydepth_models_path, model_name)

        # print("-> Loading model from ", model_path)
        encoder_path = os.path.join(model_path, "encoder.pth")
        self.depth_decoder_path = os.path.join(model_path, "depth.pth")

        # LOADING PRETRAINED MODEL
        # print("   Loading pretrained encoder")
        self.encoder = networks.ResnetEncoder(18, False)
        loaded_dict_enc = torch.load(encoder_path, map_location=self.device)

        # extract the height and width of image that this model was trained with
        self.feed_height = loaded_dict_enc['height']
        self.feed_width = loaded_dict_enc['width']
        filtered_dict_enc = {
            k: v for k, v in loaded_dict_enc.items() if k in self.encoder.state_dict()
        }
        self.encoder.load_state_dict(filtered_dict_enc)
        self.encoder.to(self.device)
        self.encoder.eval()

        # print("   Loading pretrained decoder")
        self.depth_decoder = networks.DepthDecoder(
            num_ch_enc=self.encoder.num_ch_enc, scales=range(4))

        loaded_dict = torch.load(self.depth_decoder_path, map_location=self.device)
        self.depth_decoder.load_state_dict(loaded_dict)

        self.depth_decoder.to(self.device)
        self.depth_decoder.eval()

    def eval_y(self, input_images: "List[np.ndArray | None]"):
        outputs: "List[np.ndArray | None]" = []
        with torch.no_grad():
            # Estimate depth
            for k, input_image in input_images:
                if im is None:
                    outputs.append(None)
                    continue
                # Load image and preprocess
                input_image = pil.fromarray(im)  # , [], 0, 0)
                original_width, original_height = input_image.size
                input_image = input_image.resize((self.feed_width, self.feed_height), pil.LANCZOS)
                input_image = transforms.ToTensor()(input_image).unsqueeze(0)

                # PREDICTION
                input_image = input_image.to(self.device)
                features = self.encoder(input_image)
                outputs = self.depth_decoder(features)

                disp = outputs[("disp", 0)]

                _, depth = disp_to_depth(disp, 0.1, 100)
                depth_resized = torch.nn.functional.interpolate(
                    depth, (original_height, original_width), mode="bilinear", align_corners=False
                )

                outputs.append(depth_resized.squeeze().cpu().detach().numpy() * 5.4)
        return outputs
m = monodepth()
for img in self.frames.stream(video):
    if isinstance(img, Skip):
        yield skip
        continue

    # Load image and preprocess
    input_image = Image.fromarray(img), [], 0, 0)
    original_width, original_height = input_image.size
    input_image = input_image.resize((m.feed_width, m.feed_height), Image.LANCZOS)
    input_image = transforms.ToTensor()(input_image).unsqueeze(0)

    # PREDICTION
    input_image = input_image.to(m.device)
    features = m.encoder(input_image)
    outputs = m.depth_decoder(features)

    disp = outputs[("disp", 0)]

    _, depth = disp_to_depth(disp, 0.1, 100)
    depth_resized = torch.nn.functional.interpolate(
        depth, original_height, original_width), mode="bilinear", align_corners=False
    )

    yield depth_resized.squeeze().cpu().detach().numpy() * 5.4
```

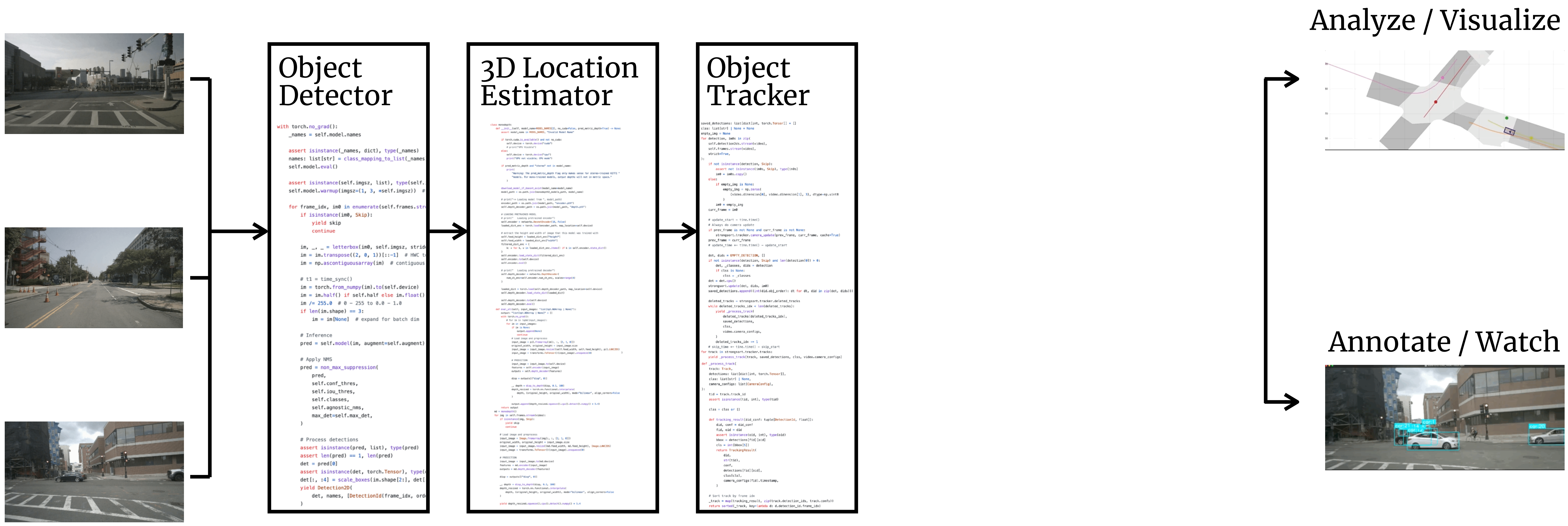
Analyze / Visualize



Annotate / Watch

#1 DOMAIN CHALLENGES

Typical Workflow (Programming)



Typical Workflow (Programming)

Object Detector

```
with torch.no_grad():
    _names = self.model.names

    assert isinstance(_names, dict), type(_names)
    names: list[str] = class_mapping_to_list(_names
    self.model.eval()

    assert isinstance(self.imgsz, list), type(self.
    self.model.warmup(imgsz=(1, 3, *self.imgsz))  #

    for frame_idx, im0 in enumerate(self.frames.str
        if isinstance(im0, Skip):
            yield skip
            continue

        im, _, _ = letterbox(im0, self.imgsz, stride
        im = im.transpose((2, 0, 1))[::-1]  # HWC to
        im = np.ascontiguousarray(im)  # contiguous

        # t1 = time_sync()
        im = torch.from_numpy(im).to(self.device)
        im = im.half() if self.half else im.float()
        im /= 255.0  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

        # Inference
        pred = self.model(im, augment=self.augment)

        # Apply NMS
        pred = non_max_suppression(
            pred,
            self.conf_thres,
            self.iou_thres,
            self.classes,
            self.agnostic_nms,
            max_det=self.max_det,
        )

        # Process detections
        assert isinstance(pred, list), type(pred)
        assert len(pred) == 1, len(pred)
        det = pred[0]
        assert isinstance(det, torch.Tensor), type(
        det[:, :4] = scale_boxes(im.shape[2:], det[
        yield Detection2D(
            det, names, [DetectionId(frame_idx, ord
        )
```

3D Location Estimator

```
class monodepth:
    def __init__(self, model_name=MODEL_NAME[2], no_cuda=False, pred_metric_depth=True) -> None:
        assert model_name in MODEL_NAME, "Invalid Model Name"

        if torch.cuda.is_available() and not no_cuda:
            self.device = torch.device("cuda")
            # print("GPU Visible")
        else:
            self.device = torch.device("cpu")
            print("GPU not visible. CPU mode")

        if pred_metric_depth and "zoedepth" not in model_name:
            print(
                "Warning: The pred_metric_depth flag only makes sense for zoedepth models. "
                "Results may not be metric depth for other models."
            )

        download_model_if_doesnt_exist(model_name=model_name)
        model_path = os.path.join(MonoDepthNN_models_path, model_name)

        # print("-> Loading model from ", model_path)
        encoder_path = os.path.join(model_path, "encoder.pth")
        self.depth_decoder_path = os.path.join(model_path, "depth.pth")

        # LOADING PRETRAINED MODEL
        # print("   Loading pretrained encoder")
        self.encoder = networks.ResnetEncoder(18, False)
        loaded_dict_enc = torch.load(encoder_path, map_location=self.device)

        # extract the height and width of image that this model was trained with
        self.feed_height = loaded_dict_enc["height"]
        self.feed_width = loaded_dict_enc["width"]
        filtered_dict_enc = {
            k: v for k, v in loaded_dict_enc.items() if k in self.encoder.state_dict()
        }
        self.encoder.load_state_dict(filtered_dict_enc)
        self.encoder.to(self.device)
        self.encoder.eval()

        # print("   Loading pretrained decoder")
        self.depth_decoder = networks.DepthDecoder(
            num_ch_enc=self.encoder.num_ch_enc, scales=range(4)
        )

        loaded_dict = torch.load(self.depth_decoder_path, map_location=self.device)
        self.depth_decoder.load_state_dict(loaded_dict)

        self.depth_decoder.to(self.device)
        self.depth_decoder.eval()

    def eval_y(self, input_images: "List[np.ndArray | None]"):
        outputs: "List[np.ndArray | None]" = []
        with torch.no_grad():
            # Run frame by frame inference
            for id in input_images:
                if id is None:
                    outputs.append(None)
                    continue
                # Load image and preprocess
                input_image = pil.fromarray(id)
                original_width, original_height = input_image.size
                input_image = input_image.resize((self.feed_width, self.feed_height), pil.LANCZOS)
                input_image = transforms.ToTensor()(input_image).unsqueeze(0)

                # PREDICTION
                input_image = input_image.to(self.device)
                features = self.encoder(input_image)
                outputs = self.depth_decoder(features)

                disp = outputs[("disp", 0)]

                _, depth = disp_to_depth(disp, 0.1, 100)
                depth_resized = torch.nn.functional.interpolate(
                    depth, (original_height, original_width), mode="bilinear", align_corners=False
                )

                output = depth_resized.squeeze().cpu().detach().numpy() * 5.4
                outputs.append(output)
        return outputs
m = monodepth()
for img in self.frames.stream(video):
    if isinstance(img, Skip):
        yield skip
        continue

    # Load image and preprocess
    input_image = pil.fromarray(img)
    original_width, original_height = input_image.size
    input_image = input_image.resize((self.feed_width, self.feed_height), pil.LANCZOS)
    input_image = transforms.ToTensor()(input_image).unsqueeze(0)

    # PREDICTION
    input_image = input_image.to(self.device)
    features = self.encoder(input_image)
    outputs = self.depth_decoder(features)

    disp = outputs[("disp", 0)]

    _, depth = disp_to_depth(disp, 0.1, 100)
    depth_resized = torch.nn.functional.interpolate(
        depth, (original_height, original_width), mode="bilinear", align_corners=False
    )

    yield depth_resized.squeeze().cpu().detach().numpy() * 5.4
```

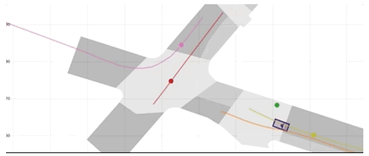
Object Tracker

```
saved_detections: list[dict[int, torch.Tensor]] = []
class: list[str] | None = None
empty_img = None
for detection, im0s in zip(
    self.detections(hls.stream(video),
    self.frames.stream(video),
    strict=True,
):
    if not isinstance(detection, Skip):
        assert not isinstance(im0s, Skip), type(im0s)
        im0 = im0s.copy()
    else:
        if empty_img is None:
            empty_img = np.zeros(
                (video.dimension[0], video.dimension[1], 3), dtype=np.uint8
            )
        im0 = empty_img
    curr_frame = im0

    # update_start = time.time()
    # Always do camera update
    if prev_frame is not None and curr_frame is not None:
        strongsort.tracker.camera_update(prev_frame, curr_frame, cache=True)
    prev_frame = curr_frame
    # update_time += time.time() - update_start

    det, dids = EMPTY_DETECTION, []
    if not isinstance(detection, Skip) and len(detection[0]) > 0:
        det, _classes, dids = detection
        if cls is None:
            cls = _classes
    det = det.cpu()
    strongsort.update(det, dids, im0)
    saved_detections.append(dict(zip(dids.obj_order)) if for d, did in zip(det, dids))

    deleted_tracks = strongsort.tracker.deleted_tracks
    while deleted_tracks_idx < len(deleted_tracks):
        yield _process_track(
            deleted_tracks[deleted_tracks_idx],
            saved_detections,
            cls,
            video.camera_configs,
        )
        deleted_tracks_idx += 1
    # skip_time += time.time() - skip_start
for track in strongsort.tracker.tracks:
    yield _process_track(track, saved_detections, cls, video.camera_configs)
def _process_track(
    track: Track,
    detections: list[dict[int, torch.Tensor]],
    class: list[str] | None,
    camera_configs: list[CameraConfig],
):
    tid = track.track_id
    assert isinstance(tid, int), type(tid)

    cls = cls or []

    def tracking_result(did_conf: tuple[DetectionId, float]):
        did, conf = did_conf
        fid, oid = did
        assert isinstance(oid, int), type(oid)
        bbox = detections[fid][oid]
        cls = int(bbox[5])
        return TrackingResult(
            did,
            str(cls),
            conf,
            detections[fid][oid],
            cls[cls],
            camera_configs[fid].timestamp,
        )

    # Sort track by frame idx
    _track = map(tracking_result, zip(track.detection_ids, track.confs))
    return sorted(_track, key=lambda d: d.detection_id.frame_idx)
```

Filter Objects

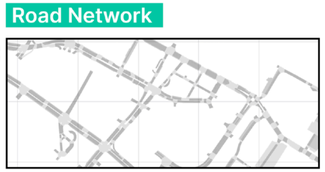
```
select c0.frameNum,
  c0.cameraId,
  c0.filename,
  t0.itemid
FROM Camera as c0
JOIN Item_Trajectory as t0
ON  c0.timestamp <@ t{i}.translations::period
AND c0.cameraId  =  t{i}.cameraId
JOIN SegmentPolygon
WHERE ST_Contains(
  SegmentPolygon.polygon,
  valueAtTimestamp(
    t0.translations,
    c0.timestamp
  )
)
```

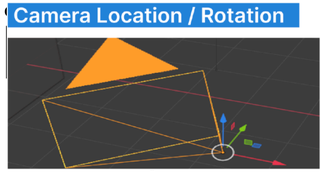
Analyze / Visualize



Annotate / Watch

Typical Workflow (Pro





Object Detector

```
with torch.no_grad():
    _names = self.model.names

    assert isinstance(_names, dict), type(_names)
    names: list[str] = class_mapping_to_list(_names
    self.model.eval()

    assert isinstance(self.imgsz, list), type(self.
    self.model.warmup(imgsz=(1, 3, *self.imgsz))  #

    for frame_idx, im0 in enumerate(self.frames.str
        if isinstance(im0, Skip):
            yield skip
            continue

        im, _, _ = letterbox(im0, self.imgsz, stride
        im = im.transpose((2, 0, 1))[::-1]  # HWC to
        im = np.ascontiguousarray(im)  # contiguous

        # t1 = time_sync()
        im = torch.from_numpy(im).to(self.device)
        im = im.half() if self.half else im.float()
        im /= 255.0  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

        # Inference
        pred = self.model(im, augment=self.augment)

        # Apply NMS
        pred = non_max_suppression(
            pred,
            self.conf_thres,
            self.iou_thres,
            self.classes,
            self.agnostic_nms,
            max_det=self.max_det,
        )

        # Process detections
        assert isinstance(pred, list), type(pred)
        assert len(pred) == 1, len(pred)
        det = pred[0]
        assert isinstance(det, torch.Tensor), type(
        det[:, :4] = scale_boxes(im.shape[2:], det[
        yield Detection2D(
            det, names, [DetectionId(frame_idx, ord
        )
```

3D Location Estimator

```
class MonoDepth:
    def __init__(self, model_name=MODEL_NAME[2], no_cuda=False, pred_metric_depth=True) -> None:
        assert model_name in MODEL_NAME, "Invalid Model Name"

        if torch.cuda.is_available() and not no_cuda:
            self.device = torch.device("cuda")
            # print("GPU Visible")
        else:
            self.device = torch.device("cpu")
            print("GPU not visible. CPU mode")

        if pred_metric_depth and "stereo" not in model_name:
            print(
                "Warning: The pred_metric_depth flag only makes sense for stereo-trained KITTI "
                "models. For mono-trained models, output depths will not in metric space."
            )

        download_model_if_doesnt_exist(model_name=model_name)
        model_path = os.path.join(MonoDepthNP_models_path, model_name)

        # print("-> Loading model from ", model_path)
        encoder_path = os.path.join(model_path, "encoder.pth")
        self.depth_decoder_path = os.path.join(model_path, "depth.pth")

        # LOADING PRETRAINED MODEL
        # print("   Loading pretrained encoder")
        self.encoder = networks.ResnetEncoder(18, False)
        loaded_dict_enc = torch.load(encoder_path, map_location=self.device)

        # extract the height and width of image that this model was trained with
        self.feed_height = loaded_dict_enc['height']
        self.feed_width = loaded_dict_enc['width']
        filtered_dict_enc = {
            k: v for k, v in loaded_dict_enc.items() if k in self.encoder.state_dict()
        }
        self.encoder.load_state_dict(filtered_dict_enc)
        self.encoder.to(self.device)
        self.encoder.eval()

        # print("   Loading pretrained decoder")
        self.depth_decoder = networks.DepthDecoder(
            num_ch_enc=self.encoder.num_ch_enc, scales=range(4)
        )

        loaded_dict = torch.load(self.depth_decoder_path, map_location=self.device)
        self.depth_decoder.load_state_dict(loaded_dict)

        self.depth_decoder.to(self.device)
        self.depth_decoder.eval()

    def eval_or(self, input_images: "List[np.ndArray | None]"):
        outputs: "List[np.ndArray | None]" = []
        with torch.no_grad():
            # Load image and preprocess
            for inv_idx, input_image in
                if in is None:
                    outputs.append(None)
                    continue
                # Load image and preprocess
                input_image = pil.fromarray(in0, 'r', [], 1, 0)[]
                original_width, original_height = input_image.size
                input_image = input_image.resize((self.feed_width, self.feed_height), pil.LANCZOS)
                input_image = transforms.ToTensor()(input_image).unsqueeze(0)

                # PREDICTION
                input_image = input_image.to(self.device)
                features = self.encoder(input_image)
                outputs = self.depth_decoder(features)

                disp = outputs[("disp", 0)]

                _, depth = disp_to_depth(disp, 0.1, 100)
                depth_resized = torch.nn.functional.interpolate(
                    depth, (original_height, original_width), mode="bilinear", align_corners=False
                )

                output.append(depth_resized.squeeze().cpu().detach().numpy() * 5.4)
        return outputs
in = MonoDepth()
for img in self.frames.stream(video):
    if isinstance(img, Skip):
        yield skip
        continue

    # Load image and preprocess
    input_image = Image.fromarray(img)[:, :, [2, 1, 0]]
    original_width, original_height = input_image.size
    input_image = input_image.resize((self.feed_width, self.feed_height), Image.LANCZOS)
    input_image = transforms.ToTensor()(input_image).unsqueeze(0)

    # PREDICTION
    input_image = input_image.to(self.device)
    features = self.encoder(input_image)
    outputs = self.depth_decoder(features)

    disp = outputs[("disp", 0)]

    _, depth = disp_to_depth(disp, 0.1, 100)
    depth_resized = torch.nn.functional.interpolate(
        depth, original_height, original_width, mode="bilinear", align_corners=False
    )

    yield depth_resized.squeeze().cpu().detach().numpy() * 5.4
```

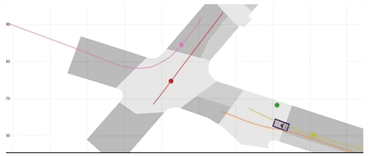
Object Tracker

```
saved_detections: List[dict[int, torch.Tensor]] = []
class: List[str] | None = None
empty_img = None
for detection, im0s in zip(
    self.detections(this.stream(video),
    self.frames.stream(video),
    strict=True,
):
    if not isinstance(detection, Skip):
        assert not isinstance(im0s, Skip), type(im0s)
        im0 = im0s.copy()
    else:
        if empty_img is None:
            empty_img = np.zeros(
                (video.dimension[0], video.dimension[1], 3), dtype=np.uint8
            )
        im0 = empty_img
    curr_frame = im0

    # update_start = time.time()
    # Always do camera update
    if prev_frame is not None and curr_frame is not None:
        strongsort.tracker.camera_update(prev_frame, curr_frame, cache=True)
    prev_frame = curr_frame
    # update_time += time.time() - update_start

    det, dids = EMPTY_DETECTION, []
    if not isinstance(detection, Skip) and len(detection[0]) > 0:
        det, _classes, dids = detection
        if cls is None:
            cls = _classes
    det = det.cpu()
    strongsort.update(det, dids, im0)
    saved_detections.append(dict(zip(did.obj_order)) if for dt, did in zip(det, dids))

    deleted_tracks = strongsort.tracker.deleted_tracks
    while deleted_tracks_idx < len(deleted_tracks):
        yield _process_track(
            deleted_tracks[deleted_tracks_idx],
            saved_detections,
            cls,
            video.camera_configs,
        )
        deleted_tracks_idx += 1
    # skip_time += time.time() - skip_start
for track in strongsort.tracker.tracks:
    yield _process_track(track, saved_detections, cls, video.camera_configs)
def _process_track(
    track: Track,
    detections: list[dict[int, torch.Tensor]],
    class: list[str] | None,
    camera_configs: list[CameraConfig],
):
    tid = track.track_id
    assert isinstance(tid, int), type(tid)

    ctxs = ctxs or []

    def tracking_result(did_conf: tuple[DetectionId, float]):
        did, conf = did_conf
        fid, oid = did
        assert isinstance(oid, int), type(oid)
        bbox = detections[fid][oid]
        cls = int(bbox[5])
        return TrackingResult(
            did,
            str(tid),
            conf,
            detections[fid][oid],
            cls[cls],
            camera_configs[fid].timestamp,
        )

    # Sort track by frame idx
    _track = map(tracking_result, zip(track.detection_ids, track.confs))
    return sorted(_track, key=lambda d: d.detection_id.frame_idx)
```

Filter Objects

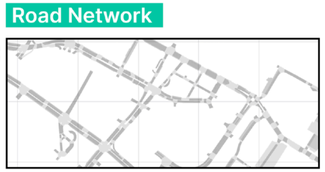
```
select c0.frameNum,
  c0.cameraId,
  c0.filename,
  t0.itemid
FROM Camera as c0
JOIN Item_Trajectory as t0
ON  c0.timestamp <@ t{i}.translations::period
AND c0.cameraId  =  t{i}.cameraId
JOIN SegmentPolygon
WHERE ST_Contains(
  SegmentPolygon.polygon,
  valueAtTimestamp(
    t0.translations,
    c0.timestamp
  )
)
```

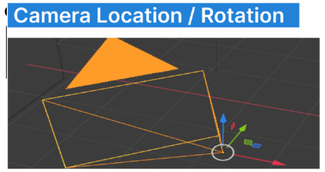
Analyze / Visualize



Annotate / Watch

Typical Workflow (Pro





Object Detector

```
with torch.no_grad():
    _names = self.model.names

    assert isinstance(_names, dict), type(_names)
    names: list[str] = class_mapping_to_list(_names
    self.model.eval()

    assert isinstance(self.imgsz, list), type(self.
    self.model.warmup(imgsz=(1, 3, *self.imgsz))  #

    for frame_idx, im0 in enumerate(self.frames.str
        if isinstance(im0, Skip):
            yield skip
            continue

        im, _, _ = letterbox(im0, self.imgsz, stride
        im = im.transpose((2, 0, 1))[::-1]  # HWC to
        im = np.ascontiguousarray(im)  # contiguous

        # t1 = time_sync()
        im = torch.from_numpy(im).to(self.device)
        im = im.half() if self.half else im.float()
        im /= 255.0  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

        # Inference
        pred = self.model(im, augment=self.augment)

        # Apply NMS
        pred = non_max_suppression(
            pred,
            self.conf_thres,
            self.iou_thres,
            self.classes,
            self.agnostic_nms,
            max_det=self.max_det,
        )

        # Process detections
        assert isinstance(pred, list), type(pred)
        assert len(pred) == 1, len(pred)
        det = pred[0]
        assert isinstance(det, torch.Tensor), type(
        det[:, :4] = scale_boxes(im.shape[2:], det[
        yield Detection2D(
            det, names, [DetectionId(frame_idx, ord
        )
```

3D Location Estimator

```
class monodepth:
    def __init__(self, model_name=MODEL_NAME[2], no_cuda=False, pred_metric_depth=True) -> None:
        assert model_name in MODEL_NAME, "Invalid Model Name"

        if torch.cuda.is_available() and not no_cuda:
            self.device = torch.device("cuda")
            # print("GPU Visible")
        else:
            self.device = torch.device("cpu")
            print("GPU not visible. CPU mode")

        if pred_metric_depth and "zoedepth" not in model_name:
            print(
                "Warning: The pred_metric_depth flag only makes sense for zoedepth models. "
                "Results may not be metric depth for other models."
            )

        download_model_if_doesnt_exist(model_name=model_name)
        model_path = os.path.join(MonoDepthNN_models_path, model_name)

        # print("-> Loading model from ", model_path)
        encoder_path = os.path.join(model_path, "encoder.pth")
        self.depth_decoder_path = os.path.join(model_path, "depth.pth")

        # LOADING PRETRAINED MODEL
        # print("   Loading pretrained encoder")
        self.encoder = networks.ResnetEncoder(18, False)
        loaded_dict_enc = torch.load(encoder_path, map_location=self.device)

        # extract the height and width of image that this model was trained with
        self.feed_height = loaded_dict_enc['height']
        self.feed_width = loaded_dict_enc['width']
        filtered_dict_enc = {
            k: v for k, v in loaded_dict_enc.items() if k in self.encoder.state_dict()
        }
        self.encoder.load_state_dict(filtered_dict_enc)
        self.encoder.to(self.device)
        self.encoder.eval()

        # print("   Loading pretrained decoder")
        self.depth_decoder = networks.DepthDecoder(
            num_ch_enc=self.encoder.num_ch_enc, scales=range(4)
        )

        loaded_dict = torch.load(self.depth_decoder_path, map_location=self.device)
        self.depth_decoder.load_state_dict(loaded_dict)

        self.depth_decoder.to(self.device)
        self.depth_decoder.eval()

    def eval_y(self, input_images: "List[np.ndArray | None]"):
        outputs: "List[np.ndArray | None]" = []
        with torch.no_grad():
            # Run frames one by one (batch_size=1)
            for id in input_images:
                if id is None:
                    outputs.append(None)
                    continue
                # Load image and preprocess
                input_image = pil.fromarray(id).convert('RGB')
                original_width, original_height = input_image.size
                input_image = input_image.resize((self.feed_width, self.feed_height), pil.LANCZOS)
                input_image = transforms.ToTensor()(input_image).unsqueeze(0)

                # PREDICTION
                input_image = input_image.to(self.device)
                features = self.encoder(input_image)
                outputs = self.depth_decoder(features)

                disp = outputs[("disp", 0)]

                _, depth = disp_to_depth(disp, 0.1, 100)
                depth_resized = torch.nn.functional.interpolate(
                    depth, (original_height, original_width), mode="bilinear", align_corners=False
                )

                output = depth_resized.squeeze().cpu().detach().numpy() * 5.4
                outputs.append(output)
        return outputs
m = monodepth()
for img in self.frames.stream(video):
    if isinstance(img, Skip):
        yield skip
        continue

    # Load image and preprocess
    input_image = pil.fromarray(img)
    original_width, original_height = input_image.size
    input_image = input_image.resize((self.feed_width, self.feed_height), Image.LANCZOS)
    input_image = transforms.ToTensor()(input_image).unsqueeze(0)

    # PREDICTION
    input_image = input_image.to(self.device)
    features = self.encoder(input_image)
    outputs = self.depth_decoder(features)

    disp = outputs[("disp", 0)]

    _, depth = disp_to_depth(disp, 0.1, 100)
    depth_resized = torch.nn.functional.interpolate(
        depth, (original_height, original_width), mode="bilinear", align_corners=False
    )

    yield depth_resized.squeeze().cpu().detach().numpy() * 5.4
```

Object Tracker

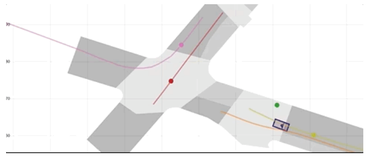
```
saved_detections: list[dict[int, torch.Tensor]] = []
class: list[str] | None = None
empty_img = None
for detection, im0s in zip(
    self.detections(hls,stream(video),
    self.frames.stream(video),
    strict=True,
):
    if not isinstance(detection, Skip):
        assert not isinstance(im0s, Skip), type(im0s)
        im0 = im0s.copy()
    else:
        if empty_img is None:
            empty_img = np.zeros(
                (video.dimension[0], video.dimension[1], 3), dtype=np.uint8
            )
        im0 = empty_img
    curr_frame = im0

    # update_start = time.time()
    # Always do camera update
    if prev_frame is not None and curr_frame is not None:
        strongsort.tracker.camera_update(prev_frame, curr_frame, cache=True)
    prev_frame = curr_frame
    # update_time += time.time() - update_start

    det, dids = EMPTY_DETECTION, []
    if not isinstance(detection, Skip) and len(detection[0]) > 0:
        det, _classes, dids = detection
        if cls is None:
            cls = _classes
    det = det.cpu()
    strongsort.update(det, dids, im0)
    saved_detections.append(dict(zip(dids.obj_order)) if for dt, did in zip(det, dids))

    deleted_tracks = strongsort.tracker.deleted_tracks
    while deleted_tracks_idx < len(deleted_tracks):
        yield _process_track(
            deleted_tracks[deleted_tracks_idx],
            saved_detections,
            cls,
            video.camera_configs,
        )
        deleted_tracks_idx += 1
    # skip_time += time.time() - skip_start
for track in strongsort.tracker.tracks:
    yield _process_track(track, saved_detections, cls, video.camera_configs)
def _process_track(
    track: Track,
    detections: list[dict[int, torch.Tensor]],
    class: list[str] | None,
    camera_configs: list[CameraConfig],
):
    tid = track.track_id
    assert isinstance(tid, int), type(tid)

    cls = cls or []

    def tracking_result(did_conf: tuple[DetectionId, float]):
        did, conf = did_conf
        fid, oid = did
        assert isinstance(oid, int), type(oid)
        bbox = detections[fid][oid]
        cls = int(bbox[5])
        return TrackingResult(
            did,
            str(cls),
            conf,
            detections[fid][oid],
            cls[cls],
            camera_configs[fid].timestamp,
        )

    # Sort track by frame idx
    _track = map(tracking_result, zip(track.detection_ids, track.confs))
    return sorted(_track, key=lambda d: d.detection_id.frame_idx)
```

Filter Objects

```
select c0.frameNum,
  c0.cameraId,
  c0.filename,
  t0.itemid
FROM Camera as c0
JOIN Item_Trajectory as t0
ON  c0.timestamp <@ t{i}.translations::period
AND c0.cameraId  =  t{i}.cameraId
JOIN SegmentPolygon
WHERE ST_Contains(
  SegmentPolygon.polygon,
  valueAtTimestamp(
    t0.translations,
    c0.timestamp
  )
)
```

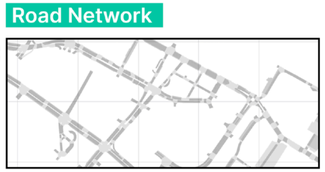
Format Outputs

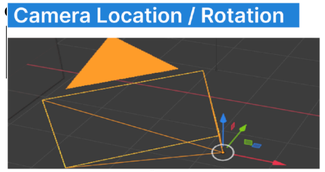
Analyze / Visualize



Annotate / Watch

Typical Workflow (Pro





Object Detector

```
with torch.no_grad():
    _names = self.model.names

    assert isinstance(_names, dict), type(_names)
    names: list[str] = class_mapping_to_list(_names
    self.model.eval()

    assert isinstance(self.imgsz, list), type(self.
    self.model.warmup(imgsz=(1, 3, *self.imgsz))  #

    for frame_idx, im0 in enumerate(self.frames.str
        if isinstance(im0, Skip):
            yield skip
            continue

        im, _, _ = letterbox(im0, self.imgsz, strid
        im = im.transpose((2, 0, 1))[::-1]  # HWC t
        im = np.ascontiguousarray(im)  # contiguous

        # t1 = time_sync()
        im = torch.from_numpy(im).to(self.device)
        im = im.half() if self.half else im.float()
        im /= 255.0  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

        # Inference
        pred = self.model(im, augment=self.augment)

        # Apply NMS
        pred = non_max_suppression(
            pred,
            self.conf_thres,
            self.iou_thres,
            self.classes,
            self.agnostic_nms,
            max_det=self.max_det,
        )

        # Process detections
        assert isinstance(pred, list), type(pred)
        assert len(pred) == 1, len(pred)
        det = pred[0]
        assert isinstance(det, torch.Tensor), type(
        det[:, :4] = scale_boxes(im.shape[2:], det[
        yield Detection2D(
            det, names, [Detection(*frame_idx, ord
        )
```

3D Location Estimator

```
class MonoDepth:
    def __init__(self, model_name=MODEL_NAME[2], no_cuda=False, pred_metric_depth=True) -> None:
        assert model_name in MODEL_NAME, "Invalid Model Name"

        if torch.cuda.is_available() and not no_cuda:
            self.device = torch.device("cuda")
            # print("GPU Visible")
        else:
            self.device = torch.device("cpu")
            print("GPU not visible. CPU mode")

        if pred_metric_depth and "shared" not in model_name:
            print(
                "Warning: The pred_metric_depth flag only makes sense for stereo-trained KITTI "
                "models. For mono-trained models, output depths will not be metric space."
            )

        download_model_if_doesnt_exist(model_name=model_name)
        model_path = os.path.join(MonoDepthNP_models_path, model_name)

        # print("-> Loading model from ", model_path)
        encoder_path = os.path.join(model_path, "encoder.pth")
        self.depth_decoder_path = os.path.join(model_path, "depth.pth")

        # LOADING PRETRAINED MODEL
        # print("   Loading pretrained encoder")
        self.encoder = networks.ResnetEncoder(18, False)
        loaded_dict_enc = torch.load(encoder_path, map_location=self.device)

        # extract the height and width of image that this model was trained with
        self.feed_height = loaded_dict_enc['height']
        self.feed_width = loaded_dict_enc['width']
        filtered_dict_enc = {
            k: v for k, v in loaded_dict_enc.items() if k in self.encoder.state_dict()
        }
        self.encoder.load_state_dict(filtered_dict_enc)
        self.encoder.to(self.device)
        self.encoder.eval()

        # print("   Loading pretrained decoder")
        self.depth_decoder = networks.DepthDecoder(
            num_ch_enc=self.encoder.num_ch_enc, scales=range(4)
        )

        loaded_dict = torch.load(self.depth_decoder_path, map_location=self.device)
        self.depth_decoder.load_state_dict(loaded_dict)

        self.depth_decoder.to(self.device)
        self.depth_decoder.eval()

    def pred_y(self, input_images: "List[np.ndArray | None]"):
        outputs: "List[np.ndArray | None]" = []
        with torch.no_grad():
            for i, input_image in enumerate(
                # Load image and preprocess
                if im is None:
                    outputs.append(None)
                    continue
                input_image = pil.fromarray(im)
                original_width, original_height = input_image.size
                input_image = input_image.resize((self.feed_width, self.feed_height), pil.LANCZOS)
                input_image = transforms.ToTensor()(input_image).unsqueeze(0)

                # PREDICTION
                input_image = input_image.to(self.device)
                features = self.encoder(input_image)
                outputs = self.depth_decoder(features)

                disp = outputs[("disp", 0)]

                _, depth = disp_to_depth(disp, 0.1, 100)
                depth_resized = torch.nn.functional.interpolate(
                    depth, (original_height, original_width), mode="bilinear", align_corners=False
                )

                outputs.append(depth_resized.squeeze().cpu().detach().numpy() * 5.4)
        return outputs
md = MonoDepthNP()
for img in self.frames.stream(video):
    if isinstance(img, Skip):
        yield skip
        continue

    # Load image and preprocess
    input_image = Image.fromarray(img)
    original_width, original_height = input_image.size
    input_image = input_image.resize((md.feed_width, md.feed_height), Image.LANCZOS)
    input_image = transforms.ToTensor()(input_image).unsqueeze(0)

    # PREDICTION
    input_image = input_image.to(md.device)
    features = md.encoder(input_image)
    outputs = md.depth_decoder(features)

    disp = outputs[("disp", 0)]

    _, depth = disp_to_depth(disp, 0.1, 100)
    depth_resized = torch.nn.functional.interpolate(
        depth, (original_height, original_width), mode="bilinear", align_corners=False
    )

    yield depth_resized.squeeze().cpu().detach().numpy() * 5.4
```

Object Tracker

```
saved_detections: list[dict[int, torch.Tensor]] = []
class: list[str] | None = None
empty_img = None
for detection, im0s in zip(
    self.detections(this.stream(video),
    self.frames.stream(video),
    strict=True,
):
    if not isinstance(detection, Skip):
        assert not isinstance(im0s, Skip), type(im0s)
        im0 = im0s.copy()
    else:
        if empty_img is None:
            empty_img = np.zeros(
                (video.dimension[0], video.dimension[1], 3), dtype=np.uint8
            )
        im0 = empty_img
    curr_frame = im0

    # update_start = time.time()
    # Always do camera update
    if prev_frame is not None and curr_frame is not None:
        strongsort.tracker.camera_update(prev_frame, curr_frame, cache=True)
    prev_frame = curr_frame
    # update_time += time.time() - update_start

    det, dids = EMPTY_DETECTION, []
    if not isinstance(detection, Skip) and len(detection[0]) > 0:
        det, _classes, dids = detection
        if cls is None:
            cls = _classes

    det = det.cpu()
    strongsort.update(det, dids, im0)
    saved_detections.append(dict(zip(dids.obj_order)) if for dt, did in zip(det, dids))

    deleted_tracks = strongsort.tracker.deleted_tracks
    while deleted_tracks_idx < len(deleted_tracks):
        yield _process_track(
            deleted_tracks[deleted_tracks_idx],
            saved_detections,
            cls,
            video.camera_configs,
        )
        deleted_tracks_idx += 1
    # skip_time += time.time() - skip_start
for track in strongsort.tracker.tracks:
    yield _process_track(track, saved_detections, cls, video.camera_configs)
def _process_track(
    track: Track,
    detections: list[dict[int, torch.Tensor]],
    cls: list[str] | None,
    camera_configs: list[CameraConfig],
):
    tid = track.track_id
    assert isinstance(tid, int), type(tid)

    cls = cls or []

    def tracking_result(did_conf: tuple[DetectionId, float]):
        did, conf = did_conf
        fid, oid = did
        assert isinstance(oid, int), type(oid)
        bbox = detections[fid][oid]
        cls = int(bbox[5])
        return TrackingResult(
            did,
            str(cls),
            conf,
            detections[fid][oid],
            cls[fid],
            camera_configs[fid].timestamp,
        )

    # Sort track by frame idx
    _track = map(tracking_result, zip(track.detection_ids,
    return sorted(_track, key=lambda d: d.detection_id.frame_idx)
```

Filter Objects

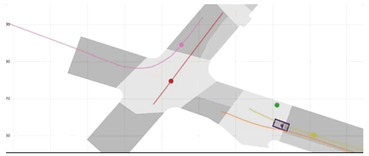
```
select c0.frameNum,
  c0.cameraId,
  c0.filename,
  t0.itemid
FROM Camera as c0
JOIN Item_Trajectory as t0
ON  c0.timestamp <@ t{i}.translations::period
AND c0.cameraId  =  t{i}.cameraId
JOIN SegmentPolygon
WHERE ST_Contains(
  SegmentPolygon.polygon,
  valueAtTimestamp(
    t0.translations,
    c0.timestamp
  )
)
```

Format Outputs

FFmpeg

OpenCV

Analyze / Visualize



Annotate / Watch

Typical Workflow (Pro

Object Detector

```
with torch.no_grad():
    _names = self.model.names

    assert isinstance(_names, dict), type(_names)
    names: list[str] = class_mapping_to_list(_names
    self.model.eval()

    assert isinstance(self.imgsz, list), type(self.
    self.model.warmup(imgsz=(1, 3, *self.imgsz))  #

    for frame_idx, im0 in enumerate(self.frames.str
        if isinstance(im0, Skip):
            yield skip
            continue

        im, _, _ = letterbox(im0, self.imgsz, stride
        im = im.transpose((2, 0, 1))[::-1]  # HWC to
        im = np.ascontiguousarray(im)  # contiguous

        # t1 = time_sync()
        im = torch.from_numpy(im).to(self.device)
        im = im.half() if self.half else im.float()
        im /= 255.0  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

        # Inference
        pred = self.model(im, augment=self.augment)

        # Apply NMS
        pred = non_max_suppression(
            pred,
            self.conf_thres,
            self.iou_thres,
            self.classes,
            self.agnostic_nms,
            max_det=self.max_det,
        )

        # Process detections
        assert isinstance(pred, list), type(pred)
        assert len(pred) == 1, len(pred)
        det = pred[0]
        assert isinstance(det, torch.Tensor), type(
        det[:, :4] = scale_boxes(im.shape[2:], det[
        yield Detection2D(
            det, names, [Detection
        )
```

3D Location Estimator

```
class MonoDepth:
    def __init__(self, model_name=MODEL_NAME[2], no_cuda=False, pred_metric_depth=True) -> None:
        assert model_name in MODEL_NAME, "Invalid Model Name"

        if torch.cuda.is_available() and not no_cuda:
            self.device = torch.device("cuda")
            # print("GPU Visible")
        else:
            self.device = torch.device("cpu")
            print("GPU not visible. CPU mode")

        if pred_metric_depth and "shared" not in model_name:
            print(
                "Warning: The pred_metric_depth flag only makes sense for stereo-trained KITTI "
                "models. For mono-trained models, output depths will not be metric space."
            )

        download_model_if_doesnt_exist(model_name=model_name)
        model_path = os.path.join(MonoDepthNP_models_path, model_name)

        # print("-> Loading model from ", model_path)
        encoder_path = os.path.join(model_path, "encoder.pth")
        self.depth_decoder_path = os.path.join(model_path, "depth.pth")

        # LOADING PRETRAINED MODEL
        # print("   Loading pretrained encoder")
        self.encoder = networks.ResnetEncoder(18, False)
        loaded_dict_enc = torch.load(encoder_path, map_location=self.device)

        # extract the height and width of image that this model was trained with
        self.feed_height = loaded_dict_enc["height"]
        self.feed_width = loaded_dict_enc["width"]
        filtered_dict_enc = {
            k: v for k, v in loaded_dict_enc.items() if k in self.encoder.state_dict()
        }
        self.encoder.load_state_dict(filtered_dict_enc)
        self.encoder.to(self.device)
        self.encoder.eval()

        # print("   Loading pretrained decoder")
        self.depth_decoder = networks.DepthDecoder(
            num_ch_enc=self.encoder.num_ch_enc, scales=range(4)
        )

        loaded_dict = torch.load(self.depth_decoder_path, map_location=self.device)
        self.depth_decoder.load_state_dict(loaded_dict)

        self.depth_decoder.to(self.device)
        self.depth_decoder.eval()

    def pred_y(self, input_images: "List[np.ndArray | None]"):
        outputs: "List[np.ndArray | None]" = []
        with torch.no_grad():
            # Prediction for each input_image:
            for im in input_images:
                if im is None:
                    outputs.append(None)
                    continue
                # Load image and preprocess
                input_image = pil.fromarray(im).convert('RGB')
                original_width, original_height = input_image.size
                input_image = input_image.resize((self.feed_width, self.feed_height), pil.LANCZOS)
                input_image = transforms.ToTensor()(input_image).unsqueeze(0)

                # PREDICTION
                input_image = input_image.to(self.device)
                features = self.encoder(input_image)
                outputs = self.depth_decoder(features)

                disp = outputs[("disp", 0)]

                _, depth = disp_to_depth(disp, 0.1, 100)
                depth_resized = torch.nn.functional.interpolate(
                    depth, (original_height, original_width), mode="bilinear", align_corners=False
                )

                outputs.append(depth_resized.squeeze().cpu().detach().numpy() * 5.4)
        return outputs
mm = MonoDepthNP()
for img in self.frames.stream(video):
    if isinstance(img, Skip):
        yield skip
        continue

    # Load image and preprocess
    input_image = Image.fromarray(img)._, (_, _), 0)
    original_width, original_height = input_image.size
    input_image = input_image.resize((mm.feed_width, mm.feed_height), Image.LANCZOS)
    input_image = transforms.ToTensor()(input_image).unsqueeze(0)

    # PREDICTION
    input_image = input_image.to(mm.device)
    features = mm.encoder(input_image)
    outputs = mm.depth_decoder(features)

    disp = outputs[("disp", 0)]

    _, depth = disp_to_depth(disp, 0.1, 100)
    depth_resized = torch.nn.functional.interpolate(
        depth, (original_height, original_width), mode="bilinear", align_corners=False
    )

    yield depth_resized.squeeze().cpu().detach().numpy() * 5.4
```

Object Tracker

```
saved_detections: list[dict[int, torch.Tensor]] = []
class: list[str] | None = None
empty_img = None
for detection, im0s in zip(
    self.detections(this.stream(video),
    self.frames.stream(video),
    strict=True,
):
    if not isinstance(detection, Skip):
        assert not isinstance(im0s, Skip), type(im0s)
        im0 = im0s.copy()
    else:
        if empty_img is None:
            empty_img = np.zeros(
                (video.dimension[0], video.dimension[1], 3), dtype=np.uint8
            )
        im0 = empty_img
    curr_frame = im0

    # update_start = time.time()
    # Always do camera update
    if prev_frame is not None and curr_frame is not None:
        strongsort.tracker.camera_update(prev_frame, curr_frame, cache=True)
    prev_frame = curr_frame
    # update_time += time.time() - update_start

    det, dids = EMPTY_DETECTION, []
    if not isinstance(detection, Skip) and len(detection[0]) > 0:
        det, _classes, dids = detection
        if cls is None:
            cls = _classes
    det = det.cpu()
    strongsort.update(det, dids, im0)
    saved_detections.append(dict(zip(did.obj_order)) if for dt, did in zip(det, dids))

    deleted_tracks = strongsort.tracker.deleted_tracks
    while deleted_tracks_idx < len(deleted_tracks):
        yield _process_track(
            deleted_tracks[deleted_tracks_idx],
            saved_detections,
            cls,
            video.camera_configs,
        )
        deleted_tracks_idx += 1
    # skip_time += time.time() - skip_start
for track in strongsort.tracker.tracks:
    yield _process_track(track, saved_detections, cls, video.camera_configs)
def _process_track(
    track: Track,
    detections: list[dict[int, torch.Tensor]],
    class: list[str] | None,
    camera_configs: list[CameraConfig],
):
    tid = track.track_id
    assert isinstance(tid, int), type(tid)

    cls = cls or []

    def tracking_result(did_conf: tuple[DetectionId, float]):
        did, conf = did_conf
        fid, oid = did
        assert isinstance(oid, int), type(oid)
        bbox = detections[fid][oid]
        cls = int(bbox[5])
        return TrackingResult(
            did,
            str(cls),
            conf,
            detections[fid][oid],
            cls[cls],
            camera_configs[fid].timestamp,
        )

    # Sort track by frame idx
    _track = map(tracking_result, zip(track.detection_ids, track.
    return sorted(_track, key=lambda d: d.detection_id.frame_idx)
```

Filter Objects

```
select c0.frameNum,
  c0.cameraId,
  c0.filename,
  t0.itemid
FROM Camera as c0
JOIN Item_Trajectory as t0
ON  c0.timestamp <@ t{i}.translations::period
AND c0.cameraId  =  t{i}.cameraId
JOIN SegmentPolygon
WHERE ST_Contains(
  SegmentPolygon.polygon,
  valueAtTimestamp(
    t0.translations,
    c0.timestamp
  )
)
```

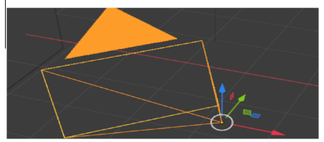
Format Outputs

FFmpeg

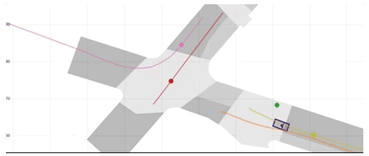
OpenCV

Road Network

Camera Location / Rotation



Analyze / Visualize



Annotate / Watch

Error-Prone

Typical Workflow (Pro

Object Detector

```
with torch.no_grad():
    _names = self.model.names

    assert isinstance(_names, dict), type(_names)
    names: list[str] = class_mapping_to_list(_names
    self.model.eval()

    assert isinstance(self.imgsz, list), type(self.
    self.model.warmup(imgsz=(1, 3, *self.imgsz))  #

    for frame_idx, im0 in enumerate(self.frames.str
        if isinstance(im0, Skip):
            yield skip
            continue

        im, _, _ = letterbox(im0, self.imgsz, stri
        im = im.transpose((2, 0, 1))[::-1]  # HWC t
        im = np.ascontiguousarray(im)  # contiguous

        # t1 = time_sync()
        im = torch.from_numpy(im).to(self.device)
        im = im.half() if self.half else im.float()
        im /= 255.0  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

        # Inference
        pred = self.model(im, augment=self.augment)

        # Apply NMS
        pred = non_max_suppression(
            pred,
            self.conf_thres,
            self.iou_thres,
            self.classes,
            self.agnostic_nms,
            max_det=self.max_det,
        )

        # Process detections
        assert isinstance(pred, list), type(pred)
        assert len(pred) == 1, len(pred)
        det = pred[0]
        assert isinstance(det, torch.Tensor), type(
        det[:, :4] = scale_boxes(im.shape[2:], det[
        yield Detection2D(
            det, names, [Detection
        )
```

3D Location Estimator

```
class MonoDepth:
    def __init__(self, model_name=MODEL_NAME[2], no_cuda=False, pred_metric_depth=True) -> None:
        assert model_name in MODEL_NAME, "Invalid Model Name"

        if torch.cuda.is_available() and not no_cuda:
            self.device = torch.device("cuda")
            # print("GPU Visible")
        else:
            self.device = torch.device("cpu")
            print("GPU not visible. CPU mode")

        if pred_metric_depth and "shared" not in model_name:
            print(
                "Warning: The pred_metric_depth flag only makes sense for stereo-trained KITTI "
                "models. For mono-trained models, output depths will not be metric space."
            )

        download_model_if_doesnt_exist(model_name=model_name)
        model_path = os.path.join(MonoDepthNP_models_path, model_name)

        # print("-> Loading model from ", model_path)
        encoder_path = os.path.join(model_path, "encoder.pth")
        self.depth_decoder_path = os.path.join(model_path, "depth.pth")

        # LOADING PRETRAINED MODEL
        # print("   Loading pretrained encoder")
        self.encoder = networks.ResnetEncoder(18, False)
        loaded_dict_enc = torch.load(encoder_path, map_location=self.device)

        # extract the height and width of image that this model was trained with
        self.feed_height = loaded_dict_enc["height"]
        self.feed_width = loaded_dict_enc["width"]
        filtered_dict_enc = {
            k: v for k, v in loaded_dict_enc.items() if k in self.encoder.state_dict()
        }
        self.encoder.load_state_dict(filtered_dict_enc)
        self.encoder.to(self.device)
        self.encoder.eval()

        # print("   Loading pretrained decoder")
        self.depth_decoder = networks.DepthDecoder(
            num_ch_enc=self.encoder.num_ch_enc, scales=range(4)
        )

        loaded_dict = torch.load(self.depth_decoder_path, map_location=self.device)
        self.depth_decoder.load_state_dict(loaded_dict)

        self.depth_decoder.to(self.device)
        self.depth_decoder.eval()

    def eval_pil(self, input_images: "List[pil.JpegImage] | None"):
        outputs: "List[np.JpegImage] | None" = []
        with torch.no_grad():
            for i, input_image in enumerate(input_images):
                if im is None:
                    outputs.append(None)
                    continue
                # Load image and preprocess
                input_image = pil.fromarray(im)
                original_width, original_height = input_image.size
                input_image = input_image.resize((self.feed_width, self.feed_height), pil.LANCZOS)
                input_image = transforms.ToTensor()(input_image).unsqueeze(0)

                # PREDICTION
                input_image = input_image.to(self.device)
                features = self.encoder(input_image)
                outputs = self.depth_decoder(features)

                disp = outputs[("disp", 0)]

                _, depth = disp_to_depth(disp, 0.1, 100)
                depth_resized = torch.nn.functional.interpolate(
                    depth, (original_height, original_width), mode="bilinear", align_corners=False
                )

                output.append(depth_resized.squeeze().cpu().detach().numpy() * 5.4)
        return output
m = MonoDepthNP()
for img in self.frames.stream(video):
    if isinstance(img, Skip):
        yield skip
        continue

    # Load image and preprocess
    input_image = Image.fromarray(img)
    original_width, original_height = input_image.size
    input_image = input_image.resize((self.feed_width, self.feed_height), Image.LANCZOS)
    input_image = transforms.ToTensor()(input_image).unsqueeze(0)

    # PREDICTION
    input_image = input_image.to(self.device)
    features = self.encoder(input_image)
    outputs = self.depth_decoder(features)

    disp = outputs[("disp", 0)]

    _, depth = disp_to_depth(disp, 0.1, 100)
    depth_resized = torch.nn.functional.interpolate(
        depth, (original_height, original_width), mode="bilinear", align_corners=False
    )

    yield depth_resized.squeeze().cpu().detach().numpy() * 5.4
```

Object Tracker

```
saved_detections: list[dict[int, torch.Tensor]] = []
class: list[str] | None = None
empty_img = None
for detection, im0s in zip(
    self.detections(h5_stream=video),
    self.frames.stream(video),
    strict=True,
):
    if not isinstance(detection, Skip):
        assert not isinstance(im0s, Skip), type(im0s)
        im0 = im0s.copy()
    else:
        if empty_img is None:
            empty_img = np.zeros(
                (video.dimension[0], video.dimension[1], 3), dtype=np.uint8
            )
        im0 = empty_img
    curr_frame = im0

    # update_start = time.time()
    # Always do camera update
    if prev_frame is not None and curr_frame is not None:
        strongsort.tracker.camera_update(prev_frame, curr_frame, cache=True)
    prev_frame = curr_frame
    # update_time += time.time() - update_start

    det, dids = EMPTY_DETECTION, []
    if not isinstance(detection, Skip) and len(detection[0]) > 0:
        det, _classes, dids = detection
        if cls is None:
            cls = _classes
    det = det.cpu()
    strongsort.update(det, dids, im0)
    saved_detections.append(dict(zip(dids.obj_order)) if for d, did in zip(det, dids))

    deleted_tracks = strongsort.tracker.deleted_tracks
    while deleted_tracks_idx < len(deleted_tracks):
        yield _process_track(
            deleted_tracks[deleted_tracks_idx],
            saved_detections,
            cls,
            video.camera_configs,
        )
        deleted_tracks_idx += 1
    # skip_time += time.time() - skip_start
for track in strongsort.tracker.tracks:
    yield _process_track(track, saved_detections, cls, video.camera_configs)
def _process_track(
    track: Track,
    detections: list[dict[int, torch.Tensor]],
    cls: list[str] | None,
    camera_configs: list[CameraConfig],
):
    tid = track.track_id
    assert isinstance(tid, int), type(tid)

    ctxs = ctxs or {}

    def tracking_result(did_conf: tuple[DetectionId, float]):
        did, conf = did_conf
        fid, oid = did
        assert isinstance(oid, int), type(oid)
        bbox = detections[fid][oid]
        cls = int(bbox[5])
        return TrackingResult(
            did,
            str(cls),
            conf,
            detections[fid][oid],
            cls[fid],
            camera_configs[fid].timestamp,
        )

    # Sort track by frame idx
    _track = map(tracking_result, zip(track.detection_idx,
    return sorted(_track, key=lambda d: d.detection_id.frame_idx)
```

Filter Objects

```
select c0.frameNum,
  c0.cameraId,
  c0.filename,
  t0.itemid
FROM Camera as c0
JOIN Item_Trajectory as t0
ON  c0.timestamp <@ t{i}.translations::period
AND c0.cameraId  =  t{i}.cameraId
JOIN SegmentPolygon
WHERE ST_Contains(
  SegmentPolygon.polygon,
  valueAtTimestamp(
    t0.translations,
    c0.timestamp
  )
)
```

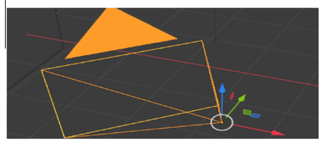
Format Outputs

FFmpeg

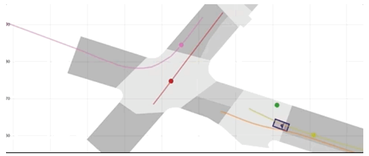
OpenCV

Road Network

Camera Location / Rotation



Analyze / Visualize

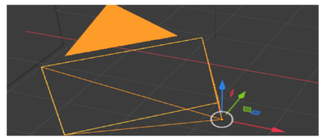


Annotate / Watch

Error-Prone

Expensive ⏳+💰

Spatialyze Workflow



Object Detector

```
with torch.no_grad():
    _names = self.model.names

    assert isinstance(_names, dict), type(_names)
    names: list[str] = class_mapping_to_list(_names
    self.model.eval()

    assert isinstance(self.imgsz, list), type(self.
    self.model.warmup(imgsz=(1, 3, *self.imgsz))  #

    for frame_idx, im0 in enumerate(self.frames.str
        if isinstance(im0, Skip):
            yield skip
            continue

        im, _, _ = letterbox(im0, self.imgsz, stride
        im = im.transpose((2, 0, 1))[::-1]  # HWC to
        im = np.ascontiguousarray(im)  # contiguous

        # t1 = time_sync()
        im = torch.from_numpy(im).to(self.device)
        im = im.half() if self.half else im.float()
        im /= 255.0  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

        # Inference
        pred = self.model(im, augment=self.augment)

        # Apply NMS
        pred = non_max_suppression(
            pred,
            self.conf_thres,
            self.iou_thres,
            self.classes,
            self.agnostic_nms,
            max_det=self.max_det,
        )

        # Process detections
        assert isinstance(pred, list), type(pred)
        assert len(pred) == 1, len(pred)
        det = pred[0]
        assert isinstance(det, torch.Tensor), type(
        det[:, :4] = scale_boxes(im.shape[2:], det[
        yield Detection2D(
            det, names, [Detection
        )
```

3D Location Estimator

```
class monodepth:
    def __init__(self, model_name=MODEL_NAME[2], no_cuda=False, pred_metric_depth=True) -> None:
        assert model_name in MODEL_NAME, "Invalid Model Name"

        if torch.cuda.is_available() and not no_cuda:
            self.device = torch.device("cuda")
            # print("GPU Visible")
        else:
            self.device = torch.device("cpu")
            print("GPU not visible. CPU mode")

        if pred_metric_depth and "stereo" not in model_name:
            print(
                "Warning: The pred_metric_depth flag only makes sense for stereo-trained KITTI "
                "models. For mono-trained models, output depths will not be metric space."
            )

        download_model_if_doesnt_exist(model_name=model_name)
        model_path = os.path.join(MONODEPTH2_MODEL_PATH, model_name)

        # print("-> Loading model from ", model_path)
        encoder_path = os.path.join(model_path, "encoder.pth")
        self.depth_decoder_path = os.path.join(model_path, "depth.pth")

        # LOADING PRETRAINED MODEL
        # print("   Loading pretrained encoder")
        self.encoder = networks.ResnetEncoder(18, False)
        loaded_dict_enc = torch.load(encoder_path, map_location=self.device)

        # extract the height and width of image that this model was trained with
        self.feed_height = loaded_dict_enc["height"]
        self.feed_width = loaded_dict_enc["width"]
        filtered_dict_enc = {
            k: v for k, v in loaded_dict_enc.items() if k in self.encoder.state_dict()
        }
        self.encoder.load_state_dict(filtered_dict_enc)
        self.encoder.to(self.device)
        self.encoder.eval()

        # print("   Loading pretrained decoder")
        self.depth_decoder = networks.DepthDecoder(
            num_ch_enc=self.encoder.num_ch_enc, scales=range(4)
        )

        loaded_dict = torch.load(self.depth_decoder_path, map_location=self.device)
        self.depth_decoder.load_state_dict(loaded_dict)

        self.depth_decoder.to(self.device)
        self.depth_decoder.eval()

    def pred_y(self, input_images: "List[np.ndArray | None]"):
        outputs: "List[np.ndArray | None]" = []
        with torch.no_grad():
            for i, input_image in enumerate(input_images):
                if im is None:
                    outputs.append(None)
                    continue
                # Load image and preprocess
                input_image = pil.fromarray(im0, 'r', [2, 1, 0])
                original_width, original_height = input_image.size
                input_image = input_image.resize((self.feed_width, self.feed_height), pil.LANCZOS)
                input_image = transforms.ToTensor()(input_image).unsqueeze(0)

                # PREDICTION
                input_image = input_image.to(self.device)
                features = self.encoder(input_image)
                outputs = self.depth_decoder(features)

                disp = outputs[("disp", 0)]

                _, depth = disp_to_depth(disp, 0.1, 100)
                depth_resized = torch.nn.functional.interpolate(
                    depth, (original_height, original_width), mode="bilinear", align_corners=False
                )

                output.append(depth_resized.squeeze().cpu().detach().numpy() * 5.4)
        return output
m = monodepth()
for img in self.frames.stream(video):
    if isinstance(img, Skip):
        yield skip
        continue

    # Load image and preprocess
    input_image = Image.fromarray(img[..., [2, 1, 0]])
    original_width, original_height = input_image.size
    input_image = input_image.resize((self.feed_width, self.feed_height), Image.LANCZOS)
    input_image = transforms.ToTensor()(input_image).unsqueeze(0)

    # PREDICTION
    input_image = input_image.to(self.device)
    features = self.encoder(input_image)
    outputs = self.depth_decoder(features)

    disp = outputs[("disp", 0)]

    _, depth = disp_to_depth(disp, 0.1, 100)
    depth_resized = torch.nn.functional.interpolate(
        depth, (original_height, original_width), mode="bilinear", align_corners=False
    )

    yield depth_resized.squeeze().cpu().detach().numpy() * 5.4
```

Object Tracker

```
saved_detections: List[dict[int, torch.Tensor]] = []
class: List[str] | None = None
empty_img = None
for detection, im0s in zip(
    self.detections(hls.stream(video),
    self.frames.stream(video),
    strict=True,
):
    if not isinstance(detection, Skip):
        assert not isinstance(im0s, Skip), type(im0s)
        im0 = im0s.copy()
    else:
        if empty_img is None:
            empty_img = np.zeros(
                (video.dimension[0], video.dimension[1], 3), dtype=np.uint8
            )
        im0 = empty_img
    curr_frame = im0

    # update_start = time.time()
    # Always do camera update
    if prev_frame is not None and curr_frame is not None:
        strongsort.tracker.camera_update(prev_frame, curr_frame, cache=True)
    prev_frame = curr_frame
    # update_time += time.time() - update_start

    det, dids = EMPTY_DETECTION, []
    if not isinstance(detection, Skip) and len(detection[0]) > 0:
        det, _classes, dids = detection
        if cls is None:
            cls = _classes
    det = det.cpu()
    strongsort.update(det, dids, im0)
    saved_detections.append(dict(zip(did.obj_order)) if for dt, did in zip(det, dids))

    deleted_tracks = strongsort.tracker.deleted_tracks
    while deleted_tracks_idx < len(deleted_tracks):
        yield _process_track(
            deleted_tracks[deleted_tracks_idx],
            saved_detections,
            cls,
            video.camera_configs,
        )
        deleted_tracks_idx += 1
    # skip_time += time.time() - skip_start
for track in strongsort.tracker.tracks:
    yield _process_track(track, saved_detections, cls, video.camera_configs)
def _process_track(
    track: Track,
    detections: list[dict[int, torch.Tensor]],
    class: list[str] | None,
    camera_configs: list[CameraConfig],
):
    tid = track.track_id
    assert isinstance(tid, int), type(tid)

    cls = cls or []

    def tracking_result(did_conf: tuple[DetectionId, float]):
        did, conf = did_conf
        fid, oid = did
        assert isinstance(oid, int), type(oid)
        bbox = detections[fid][oid]
        cls = int(bbox[5])
        return TrackingResult(
            did,
            str(cls),
            conf,
            detections[fid][oid],
            cls[cls],
            camera_configs[fid].timestamp,
        )

    # Sort track by frame idx
    _track = map(tracking_result, zip(track.detection_ids,
    return sorted(_track, key=lambda d: d.detection_id.frame_idx)
```

Filter Objects

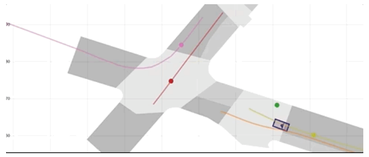
```
select c0.frameNum,
  c0.cameraId,
  c0.filename,
  t0.itemid
FROM Camera as c0
JOIN Item_Trajectory as t0
ON  c0.timestamp <@ t{i}.translations::period
AND c0.cameraId  =  t{i}.cameraId
JOIN SegmentPolygon
WHERE ST_Contains(
  SegmentPolygon.polygon,
  valueAtTimestamp(
    t0.translations,
    c0.timestamp
  )
)
```

Format Outputs

FFmpeg

OpenCV

Analyze / Visualize



Annotate / Watch

Error-Prone

Expensive ⏳+💰

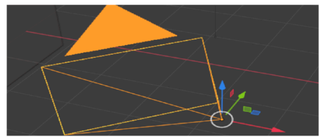
#1 DOMAIN CHALLENGES

Spatialyze Workflow

1 **The users only need to describe “what” their videos of interest look like instead of “how” to get them.**

2 **Spatialyze leverages users’ description of their videos of interest and the geospatial metadata to optimize for faster video processing.**

Spatialyze Workflow



Object Detector

```
with torch.no_grad():
    _names = self.model.names

    assert isinstance(_names, dict), type(_names)
    names: list[str] = class_mapping_to_list(_names
    self.model.eval()

    assert isinstance(self.imgsz, list), type(self.
    self.model.warmup(imgsz=(1, 3, *self.imgsz))  #

    for frame_idx, im0 in enumerate(self.frames.str
        if isinstance(im0, Skip):
            yield skip
            continue

        im, _, _ = letterbox(im0, self.imgsz, stride
        im = im.transpose((2, 0, 1))[::-1]  # HWC to
        im = np.ascontiguousarray(im)  # contiguous

        # t1 = time_sync()
        im = torch.from_numpy(im).to(self.device)
        im = im.half() if self.half else im.float()
        im /= 255.0  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

        # Inference
        pred = self.model(im, augment=self.augment)

        # Apply NMS
        pred = non_max_suppression(
            pred,
            self.conf_thres,
            self.iou_thres,
            self.classes,
            self.agnostic_nms,
            max_det=self.max_det,
        )

        # Process detections
        assert isinstance(pred, list), type(pred)
        assert len(pred) == 1, len(pred)
        det = pred[0]
        assert isinstance(det, torch.Tensor), type(
        det[:, :4] = scale_boxes(im.shape[2:], det[
        yield Detection2D(
            det, names, [Detection
        )
```

3D Location Estimator

```
class monodepth:
    def __init__(self, model_name=MODEL_NAME[2], no_cuda=False, pred_metric_depth=True) -> None:
        assert model_name in MODEL_NAME, "Invalid Model Name"

        if torch.cuda.is_available() and not no_cuda:
            self.device = torch.device("cuda")
            # print("GPU Visible")
        else:
            self.device = torch.device("cpu")
            print("GPU not visible. CPU mode")

        if pred_metric_depth and "stereo" not in model_name:
            print(
                "Warning: The pred_metric_depth flag only makes sense for stereo-trained KITTI "
                "models. For mono-trained models, output depths will not be metric space."
            )

        download_model_if_doesnt_exist(model_name=model_name)
        model_path = os.path.join(MONODEPTH2_MODEL_PATH, model_name)

        # print("-> Loading model from ", model_path)
        encoder_path = os.path.join(model_path, "encoder.pth")
        self.depth_decoder_path = os.path.join(model_path, "depth.pth")

        # LOADING PRETRAINED MODEL
        # print("   Loading pretrained encoder")
        self.encoder = networks.ResnetEncoder(18, False)
        loaded_dict_enc = torch.load(encoder_path, map_location=self.device)

        # extract the height and width of image that this model was trained with
        self.feed_height = loaded_dict_enc["height"]
        self.feed_width = loaded_dict_enc["width"]
        filtered_dict_enc = {
            k: v for k, v in loaded_dict_enc.items() if k in self.encoder.state_dict()
        }
        self.encoder.load_state_dict(filtered_dict_enc)
        self.encoder.to(self.device)
        self.encoder.eval()

        # print("   Loading pretrained decoder")
        self.depth_decoder = networks.DepthDecoder(
            num_ch_enc=self.encoder.num_ch_enc, scales=range(4)
        )

        loaded_dict = torch.load(self.depth_decoder_path, map_location=self.device)
        self.depth_decoder.load_state_dict(loaded_dict)

        self.depth_decoder.to(self.device)
        self.depth_decoder.eval()

    def pred_y(self, input_images: "List[np.ndArray | None]"):
        outputs: "List[np.ndArray | None]" = []
        with torch.no_grad():
            for i, input_image in enumerate(input_images):
                if im is None:
                    outputs.append(None)
                    continue
                # Load image and preprocess
                input_image = pil.fromarray(im0, 'r', [2, 1, 0])
                original_width, original_height = input_image.size
                input_image = input_image.resize((self.feed_width, self.feed_height), pil.LANCZOS)
                input_image = transforms.ToTensor()(input_image).unsqueeze(0)

                # PREDICTION
                input_image = input_image.to(self.device)
                features = self.encoder(input_image)
                outputs = self.depth_decoder(features)

                disp = outputs[("disp", 0)]

                _, depth = disp_to_depth(disp, 0.1, 100)
                depth_resized = torch.nn.functional.interpolate(
                    depth, (original_height, original_width), mode="bilinear", align_corners=False
                )

                output.append(depth_resized.squeeze().cpu().detach().numpy() * 5.4)
        return output
m = monodepth()
for img in self.frames.stream(video):
    if isinstance(img, Skip):
        yield skip
        continue

    # Load image and preprocess
    input_image = Image.fromarray(img[..., [2, 1, 0]])
    original_width, original_height = input_image.size
    input_image = input_image.resize((self.feed_width, self.feed_height), Image.LANCZOS)
    input_image = transforms.ToTensor()(input_image).unsqueeze(0)

    # PREDICTION
    input_image = input_image.to(self.device)
    features = self.encoder(input_image)
    outputs = self.depth_decoder(features)

    disp = outputs[("disp", 0)]

    _, depth = disp_to_depth(disp, 0.1, 100)
    depth_resized = torch.nn.functional.interpolate(
        depth, (original_height, original_width), mode="bilinear", align_corners=False
    )

    yield depth_resized.squeeze().cpu().detach().numpy() * 5.4
```

Object Tracker

```
saved_detections: List[dict[int, torch.Tensor]] = []
class: List[str] | None = None
empty_img = None
for detection, im0s in zip(
    self.detections(hls.stream(video),
    self.frames.stream(video),
    strict=True,
):
    if not isinstance(detection, Skip):
        assert not isinstance(im0s, Skip), type(im0s)
        im0 = im0s.copy()
    else:
        if empty_img is None:
            empty_img = np.zeros(
                (video.dimension[0], video.dimension[1], 3), dtype=np.uint8
            )
        im0 = empty_img
    curr_frame = im0

    # update_start = time.time()
    # Always do camera update
    if prev_frame is not None and curr_frame is not None:
        strongsort.tracker.camera_update(prev_frame, curr_frame, cache=True)
    prev_frame = curr_frame
    # update_time += time.time() - update_start

    det, dids = EMPTY_DETECTION, []
    if not isinstance(detection, Skip) and len(detection[0]) > 0:
        det, _classes, dids = detection
        if cls is None:
            cls = _classes
    det = det.cpu()
    strongsort.update(det, dids, im0)
    saved_detections.append(dict(zip(did.obj_order)) if for dt, did in zip(det, dids))

    deleted_tracks = strongsort.tracker.deleted_tracks
    while deleted_tracks_idx < len(deleted_tracks):
        yield _process_track(
            deleted_tracks[deleted_tracks_idx],
            saved_detections,
            cls,
            video.camera_configs,
        )
        deleted_tracks_idx += 1
    # skip_time += time.time() - skip_start
for track in strongsort.tracker.tracks:
    yield _process_track(track, saved_detections, cls, video.camera_configs)
def _process_track(
    track: Track,
    detections: list[dict[int, torch.Tensor]],
    class: list[str] | None,
    camera_configs: list[CameraConfig],
):
    tid = track.track_id
    assert isinstance(tid, int), type(tid)

    cls = cls or []

    def tracking_result(did_conf: tuple[DetectionId, float]):
        did, conf = did_conf
        fid, oid = did
        assert isinstance(oid, int), type(oid)
        bbox = detections[fid][oid]
        cls = int(bbox[5])
        return TrackingResult(
            did,
            str(cls),
            conf,
            detections[fid][oid],
            cls[cls],
            camera_configs[fid].timestamp,
        )

    # Sort track by frame idx
    _track = map(tracking_result, zip(track.detection_ids,
    return sorted(_track, key=lambda d: d.detection_id.frame_idx)
```

Filter Objects

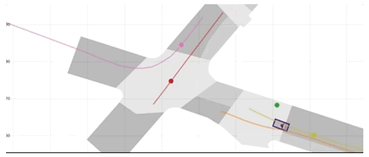
```
select c0.frameNum,
  c0.cameraId,
  c0.filename,
  t0.itemid
FROM Camera as c0
JOIN Item_Trajectory as t0
ON  c0.timestamp <@ t{i}.translations::period
AND c0.cameraId  =  t{i}.cameraId
JOIN SegmentPolygon
WHERE ST_Contains(
  SegmentPolygon.polygon,
  valueAtTimestamp(
    t0.translations,
    c0.timestamp
  )
)
```

Format Outputs

FFmpeg

OpenCV

Analyze / Visualize

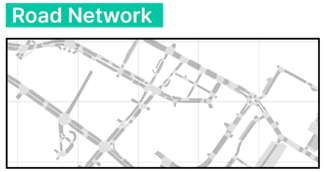


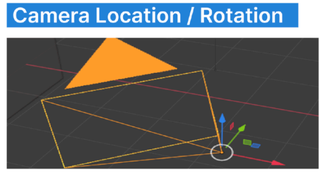
Annotate / Watch

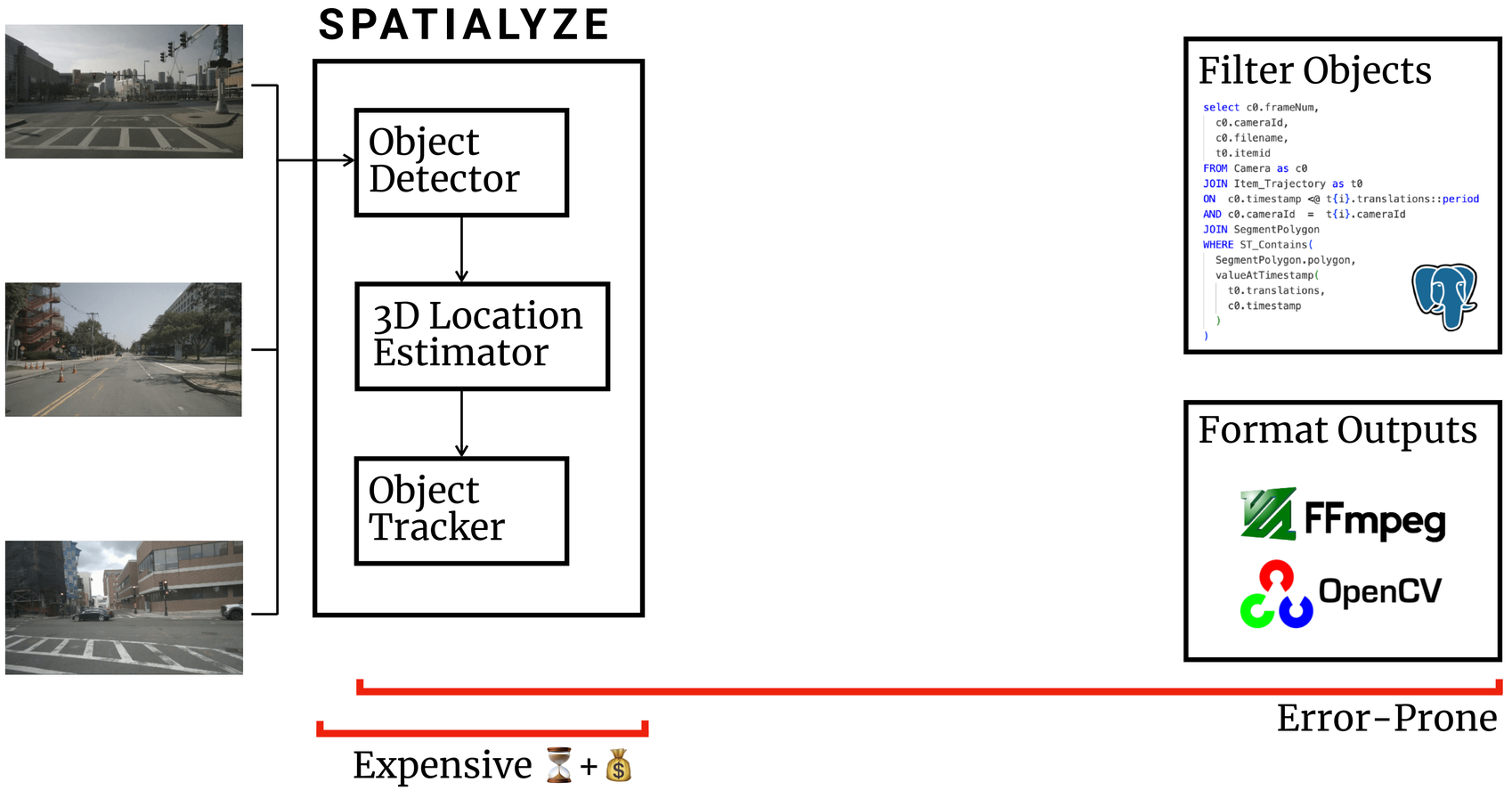
Error-Prone

Expensive ⏳+💰

Spatialyze Workflow



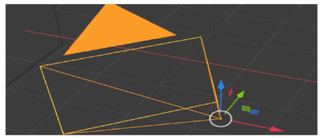




Spatialyze Workflow

Road Network

Camera Location / Rotation



SPATIALYZE

Object Detector

3D Location Estimator

Object Tracker

~~Expensive ⏳+💰~~

Filter Objects

```
select c0.frameNum,
  c0.cameraId,
  c0.filename,
  t0.itemid
FROM Camera as c0
JOIN Item_Trajectory as t0
ON  c0.timestamp <@ t{i}.translations::period
AND c0.cameraId  =  t{i}.cameraId
JOIN SegmentPolygon
WHERE ST_Contains(
  SegmentPolygon.polygon,
  valueAtTimestamp(
    t0.translations,
    c0.timestamp
  )
)
```

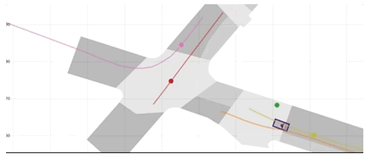
Format Outputs

FFmpeg

OpenCV

Error-Prone

Analyze / Visualize

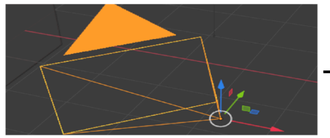


Annotate / Watch

Spatialyze Workflow

Road Network

Camera Location / Rotation



SPATIALYZE

Object Detector

3D Location Estimator

Object Tracker

Filter Objects

```
obj.type=='car' &
contains(
  intersection,
  obj)
```

Format Outputs

FFmpeg

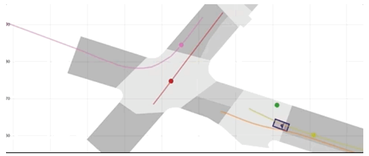
OpenCV

Error-Prone

~~Expensive ⏳+💰~~

5x speed up

Analyze / Visualize

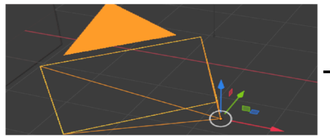


Annotate / Watch

Spatialyze Workflow

Road Network

Camera Location / Rotation



SPATIALYZE

Object Detector

3D Location Estimator

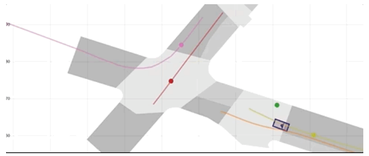
Object Tracker

Filter Objects

```
obj.type=='car' &
contains(
  intersection,
  obj)
```

Format Outputs

Analyze / Visualize



Annotate / Watch

Error-Prone

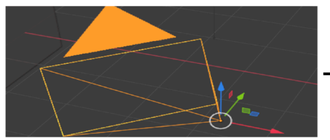
~~Expensive~~ ⏳ + ~~💰~~

5x speed up

Spatialyze Workflow

Road Network

Camera Location / Rotation



SPATIALYZE

Object Detector

3D Location Estimator

Object Tracker

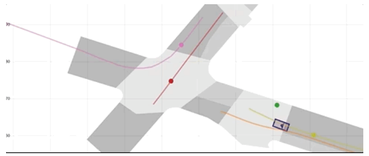
Filter Objects

```
obj.type=='car' &
contains(
  intersection,
  obj)
```

Format Outputs

Find a car at an intersection

Analyze / Visualize



Annotate / Watch

~~Error-Prone~~

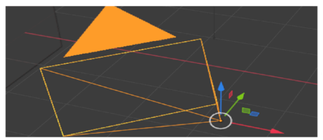
~~Expensive ⏳+💰~~

5x speed up

Spatialyze Workflow

Road Network

Camera Location / Rotation



SPATIALYZE

Object Detector

3D Location Estimator

Object Tracker

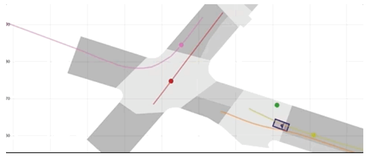
Filter Objects

```
obj.type=='car' &
contains(
  intersection,
  obj)
```

Format Outputs

Find a car at an intersection

Analyze / Visualize



Annotate / Watch

~~Error-Prone~~ → #2 SYSTEM INTERFACE

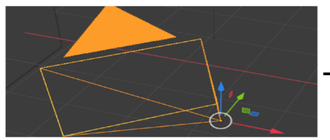
~~Expensive~~ ⏳+💰

5x speed up

Spatialyze Workflow

Road Network

Camera Location / Rotation



SPATIALYZE

Object Detector

3D Location Estimator

Object Tracker

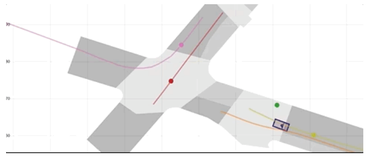
Filter Objects

```
obj.type=='car' &
contains(
  intersection,
  obj)
```

Format Outputs

Find a car at an intersection

Analyze / Visualize



Annotate / Watch

~~Error-Prone~~ → #2 SYSTEM INTERFACE

~~Expensive~~ ⏳+💰 → #3 SYSTEM OPTIMIZATION

5x speed up

SPATIALYZE

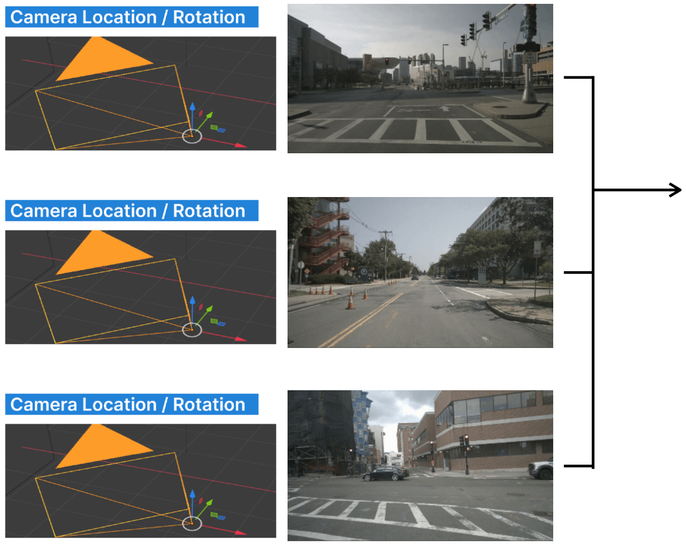
#2 System Interface

* Data Model
* Programming Model

Data Model

Data Model

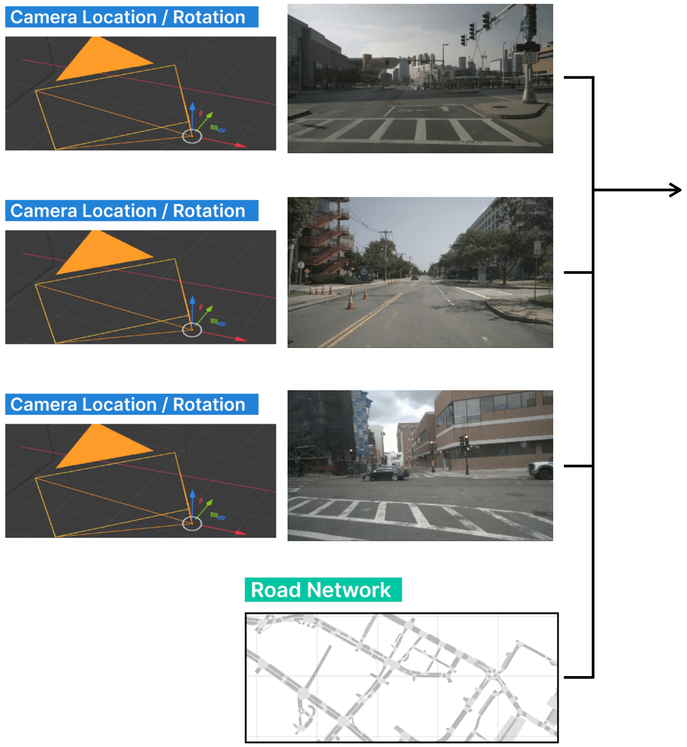
Data Model


Camera Location / Rotation
Camera Location / Rotation
Camera Location / Rotation

Data Model

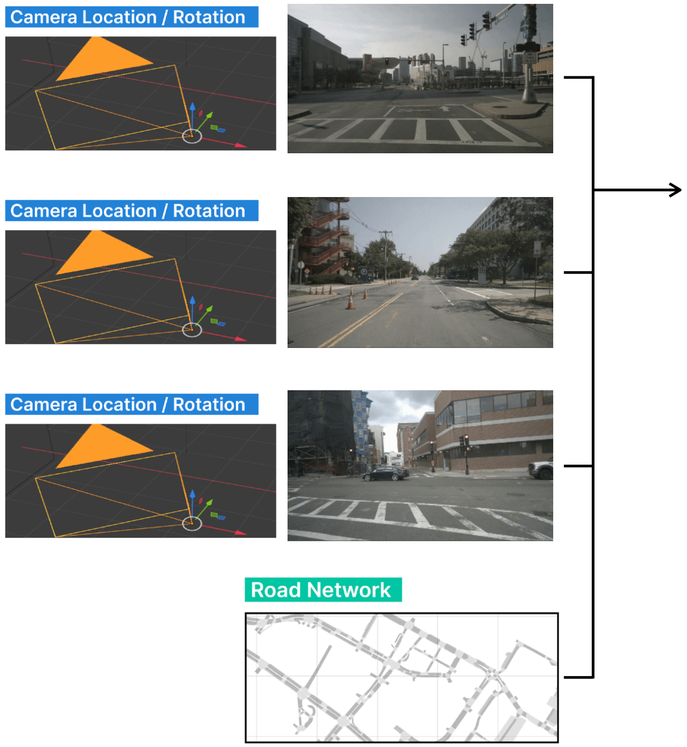

Camera Location / Rotation
Camera Location / Rotation
Camera Location / Rotation
Coca-Cola

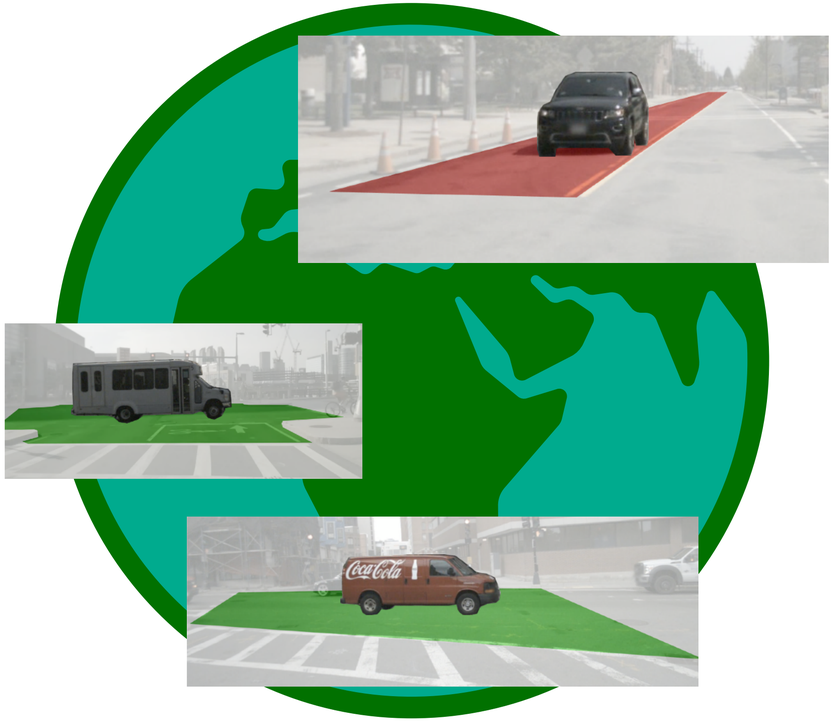
Data Model


Camera Location / Rotation
Camera Location / Rotation
Camera Location / Rotation
Road Network

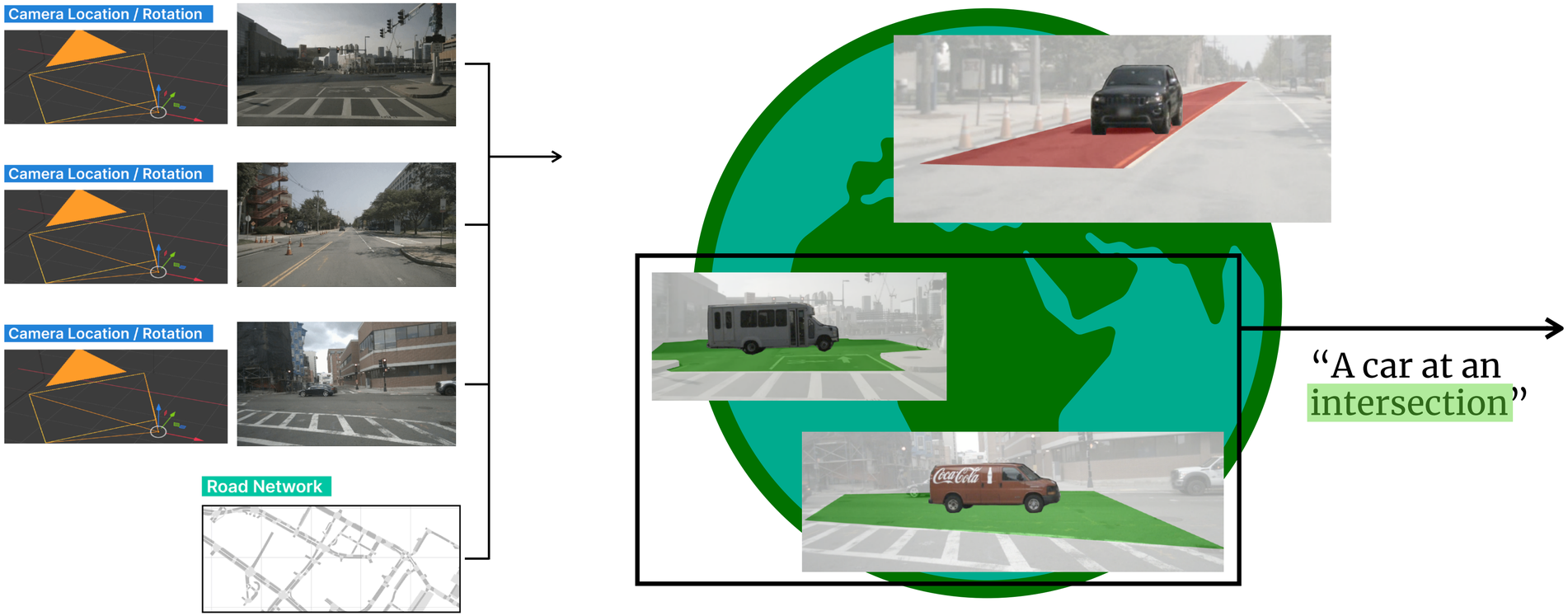

Coca-Cola

Data Model

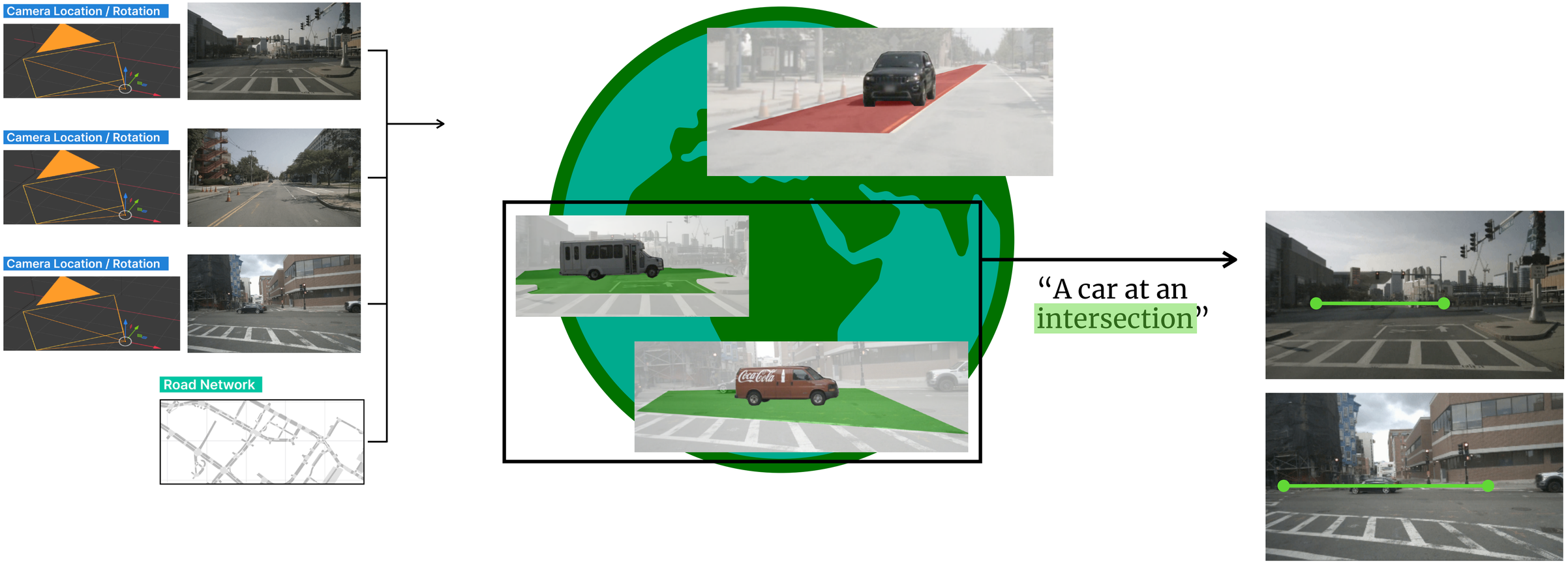

Camera Location / Rotation
Camera Location / Rotation
Camera Location / Rotation
Road Network


Coca-Cola

Data Model

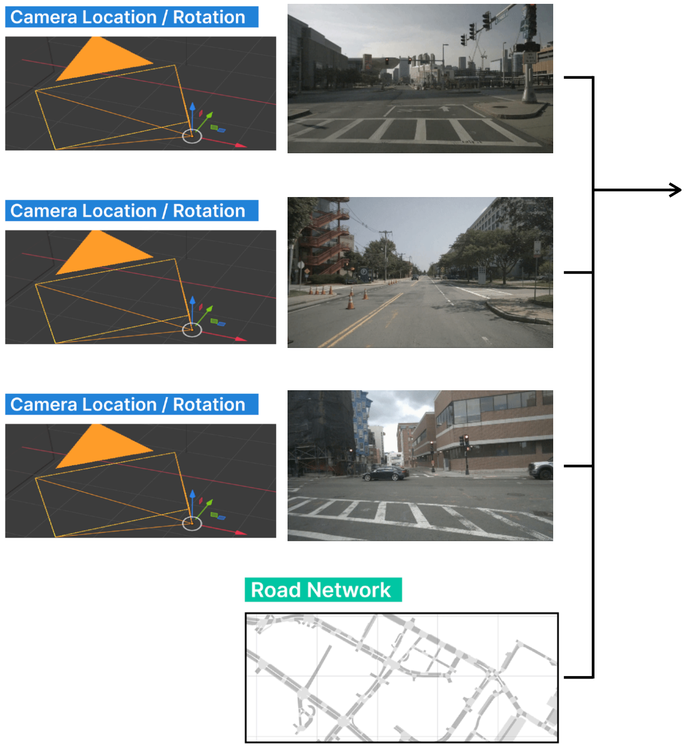

Camera Location / Rotation
Camera Location / Rotation
Camera Location / Rotation
Road Network
“A car at an intersection”

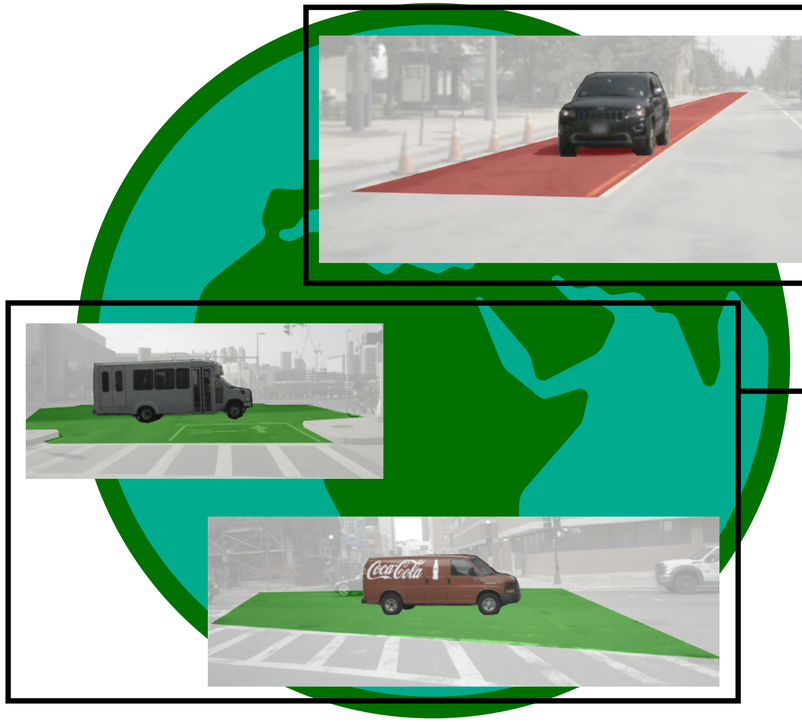
Data Model



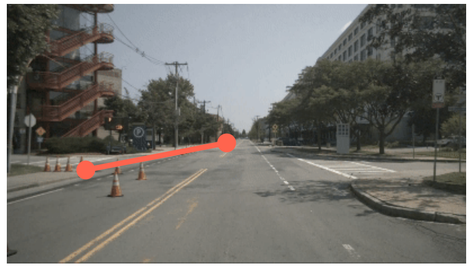
VisualWorldDB: A DBMS for the Visual World. Brandon Haynes, Maureen Daum, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. CIDR, 2020.

Data Model

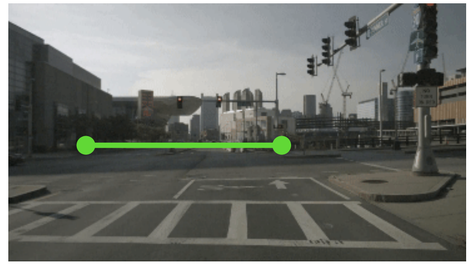




“A car on a lane”



“A car at an intersection”



VisualWorldDB: A DBMS for the Visual World. Brandon Haynes, Maureen Daum, Amrita Mazumdar, Magdalena Balazinska, Alvin Cheung, and Luis Ceze. CIDR, 2020.

Programming Model

Build → *Filter* → *Observe*

Programming Model

Build

```
w = World()
w.addVideo(video1, camera1)
w.addVideo(video2, camera2)
w.addVideo(video3, camera3)
w.addGeogConstructs(ROADNET_DIR)
```

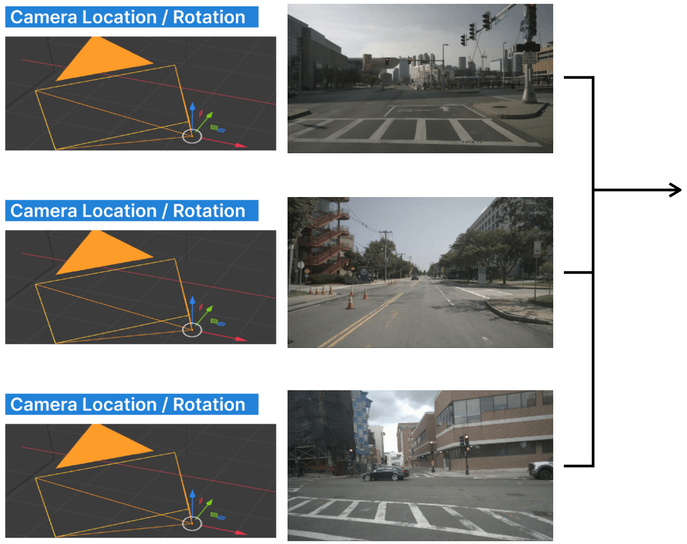
Filter

```
o = w.object()
i = w.geogConstruct('intersection')
w.filter((o.type=='car') &
         contains(i, o))
```

Observe

```
w.saveVideos('output-dir')
# or
objects = w.getObjects()
```

Programming Model



Build

```
w = World()
w.addVideo(video1, camera1)
w.addVideo(video2, camera2)
w.addVideo(video3, camera3)
w.addGeogConstructs(ROADNET_DIR)
```

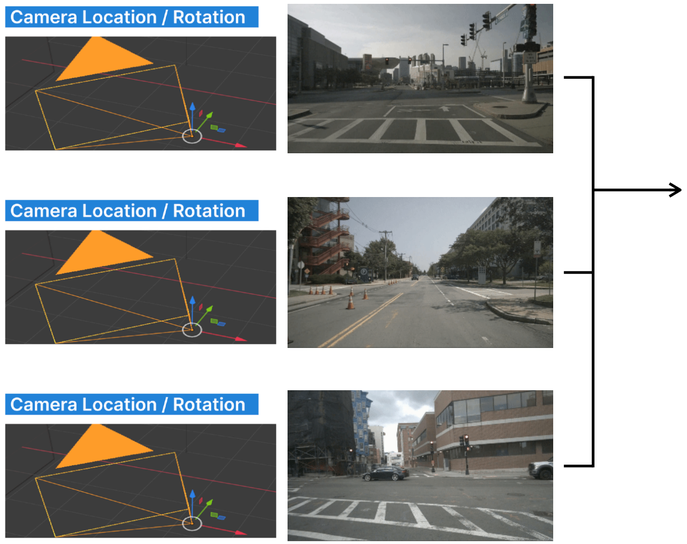
Filter

```
o = w.object()
i = w.geogConstruct('intersection')
w.filter((o.type=='car') &
         contains(i, o))
```

Observe

```
w.saveVideos('output-dir')
# or
objects = w.getObjects()
```

Programming Model



Build

```
w = World()
w.addVideo(video1, camera1)
w.addVideo(video2, camera2)
w.addVideo(video3, camera3)
w.addGeogConstructs(ROADNET_DIR)
```

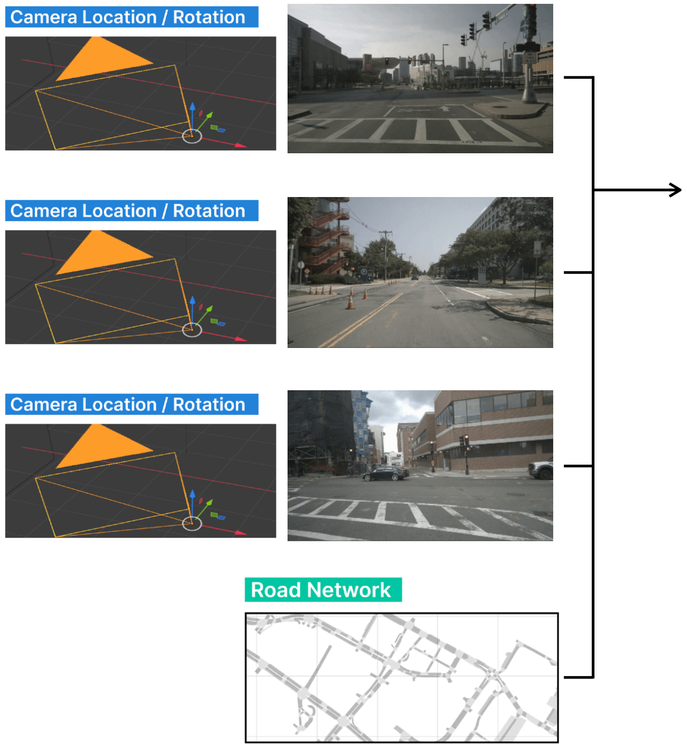
Filter

```
o = w.object()
i = w.geogConstruct('intersection')
w.filter((o.type=='car') &
         contains(i, o))
```

Observe

```
w.saveVideos('output-dir')
# or
objects = w.getObjects()
```

Programming Model



Build

```
w = World()
w.addVideo(video1, camera1)
w.addVideo(video2, camera2)
w.addVideo(video3, camera3)
w.addGeogConstructs(ROADNET_DIR)
```

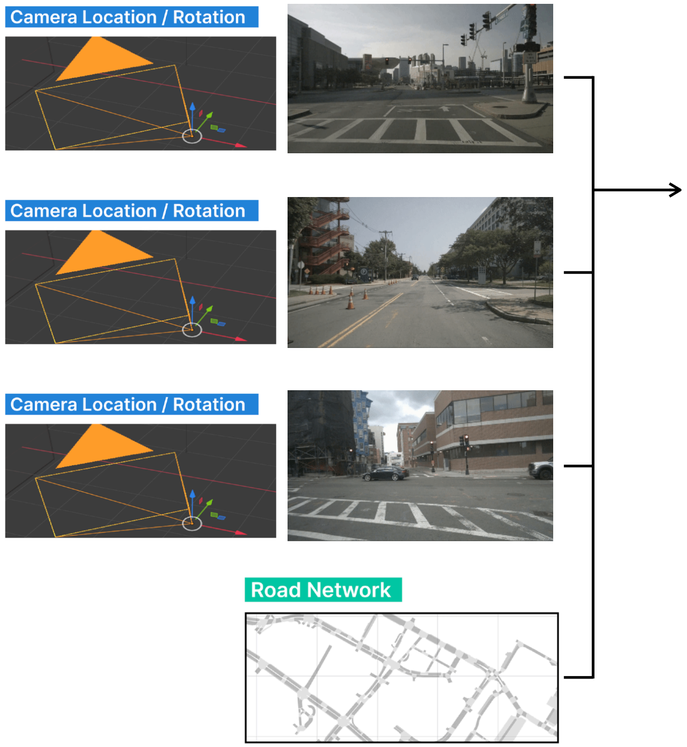
Filter

```
o = w.object()
i = w.geogConstruct('intersection')
w.filter((o.type=='car') &
         contains(i, o))
```

Observe

```
w.saveVideos('output-dir')
# or
objects = w.getObjects()
```

Programming Model



Build

```
w = World()
w.addVideo(video1, camera1)
w.addVideo(video2, camera2)
w.addVideo(video3, camera3)
w.addGeogConstructs(ROADNET_DIR)
```

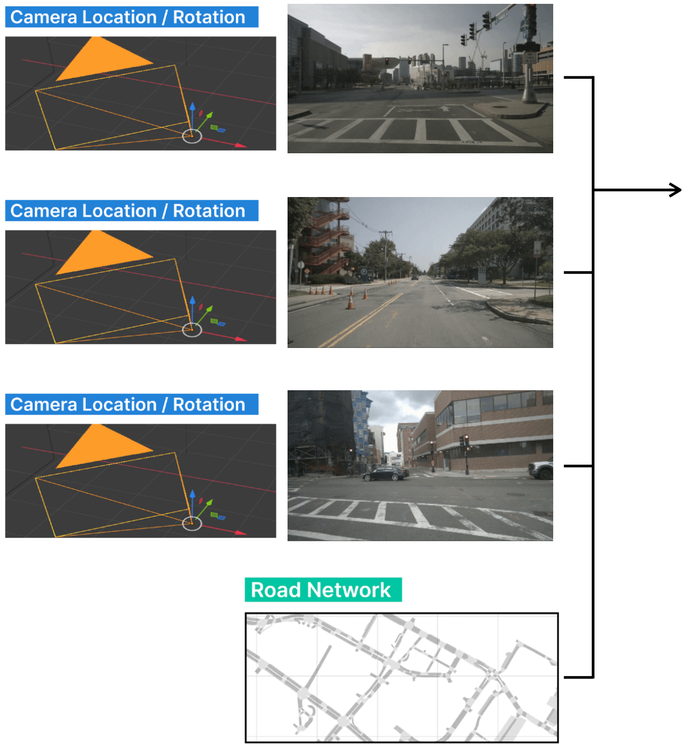
Filter

```
o = w.object()
i = w.geogConstruct('intersection')
w.filter((o.type=='car') &
         contains(i, o))
```

Observe

```
w.saveVideos('output-dir')
# or
objects = w.getObjects()
```

Programming Model



Build

```
w = World()
w.addVideo(video1, camera1)
w.addVideo(video2, camera2)
w.addVideo(video3, camera3)
w.addGeogConstructs(ROADNET_DIR)
```

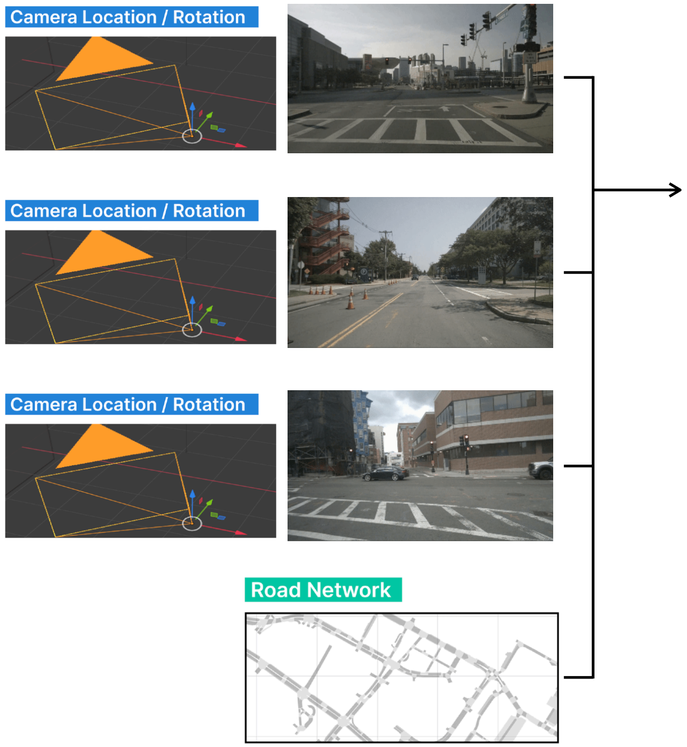
Filter

```
o = w.object()
i = w.geogConstruct('intersection')
w.filter((o.type=='car') &
         contains(i, o))
```

Observe

```
w.saveVideos('output-dir')
# or
objects = w.getObjects()
```

Programming Model



Build

```
w = World()
w.addVideo(video1, camera1)
w.addVideo(video2, camera2)
w.addVideo(video3, camera3)
w.addGeogConstructs(ROADNET_DIR)
```

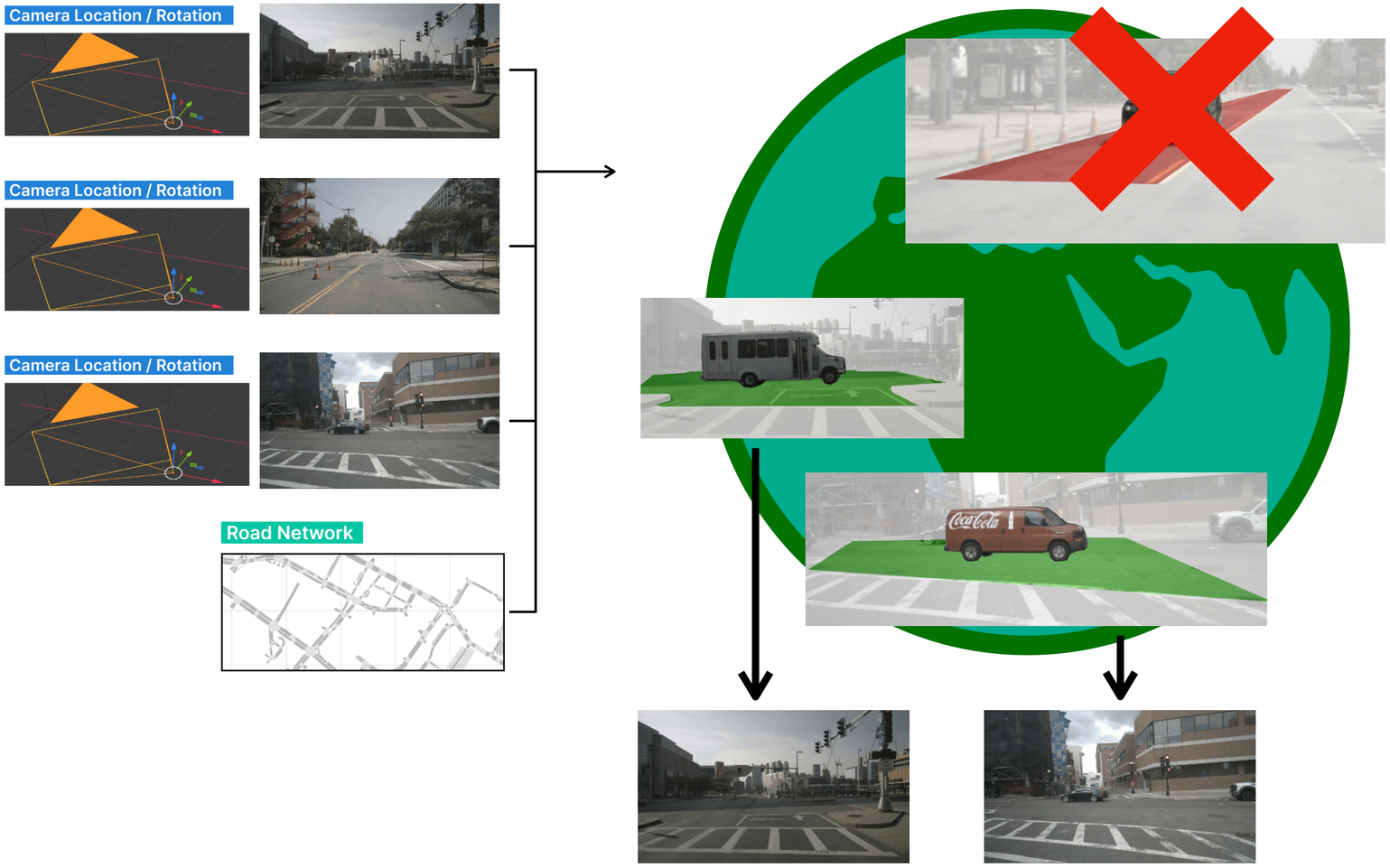
Filter

```
o = w.object()
i = w.geogConstruct('intersection')
w.filter((o.type=='car') &
         contains(i, o))
```

Observe

```
w.saveVideos('output-dir')
# or
objects = w.getObjects()
```

Programming Model



Build

```
w = World()
w.addVideo(video1, camera1)
w.addVideo(video2, camera2)
w.addVideo(video3, camera3)
w.addGeogConstructs(ROADNET_DIR)
```

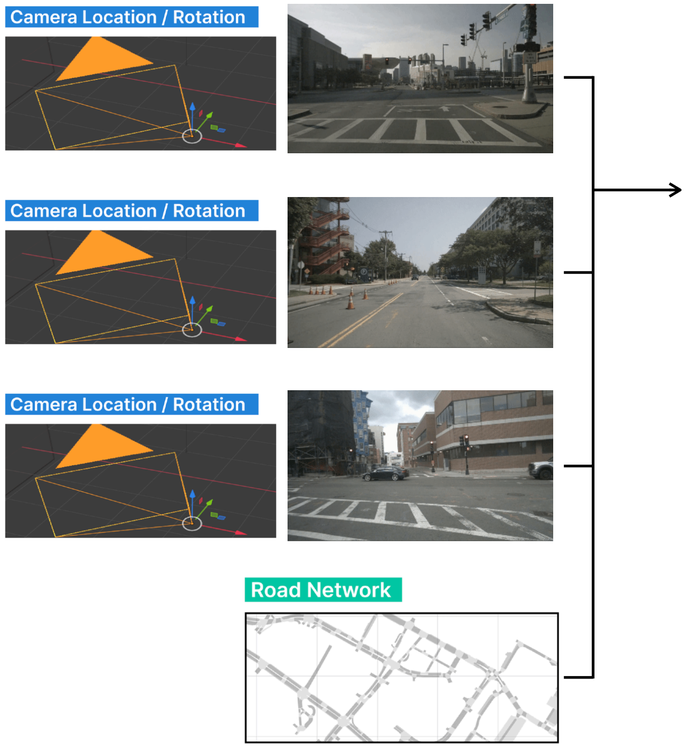
Filter

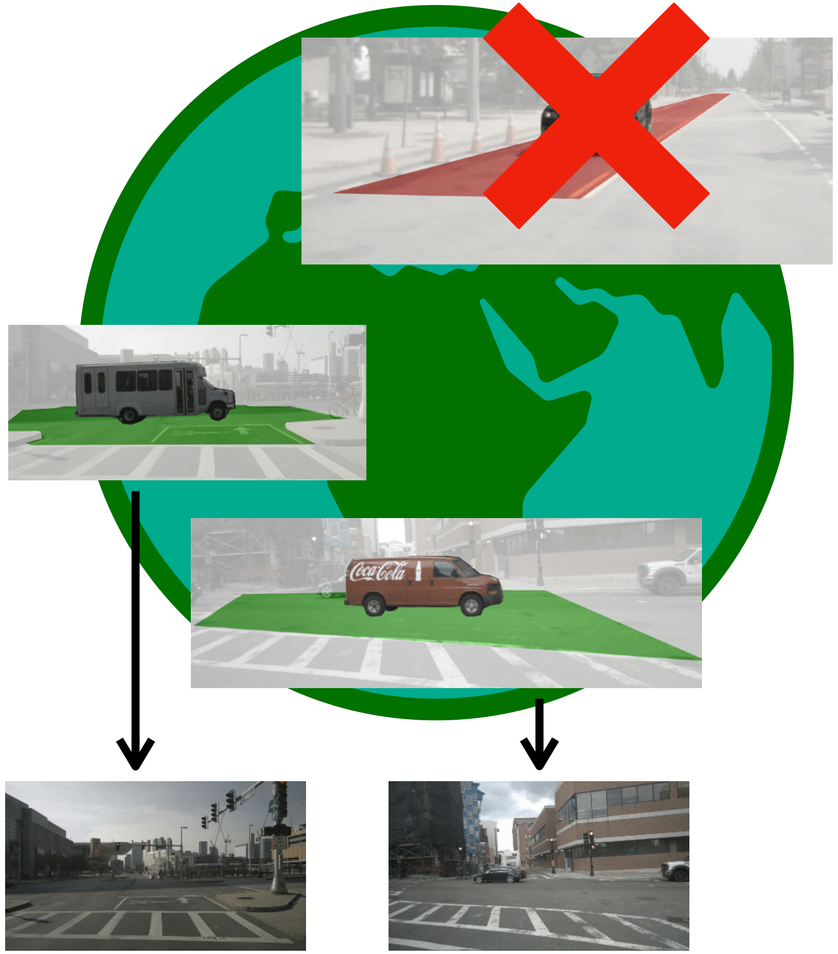
```
o = w.object()
i = w.geogConstruct('intersection')
w.filter((o.type=='car') &
         contains(i, o))
```

Observe

```
w.saveVideos('output-dir')
# or
objects = w.getObjects()
```

Programming Model





Build

```
w = World()
w.addVideo(video1, camera1)
w.addVideo(video2, camera2)
w.addVideo(video3, camera3)
w.addGeogConstructs(ROADNET_DIR)
```

Filter

```
o = w.object()
i = w.geogConstruct('intersection')
w.filter((o.type=='car') &
         contains(i, o))
```

Observe

```
w.saveVideos('output-dir')
# or
objects = w.getObjects()
```

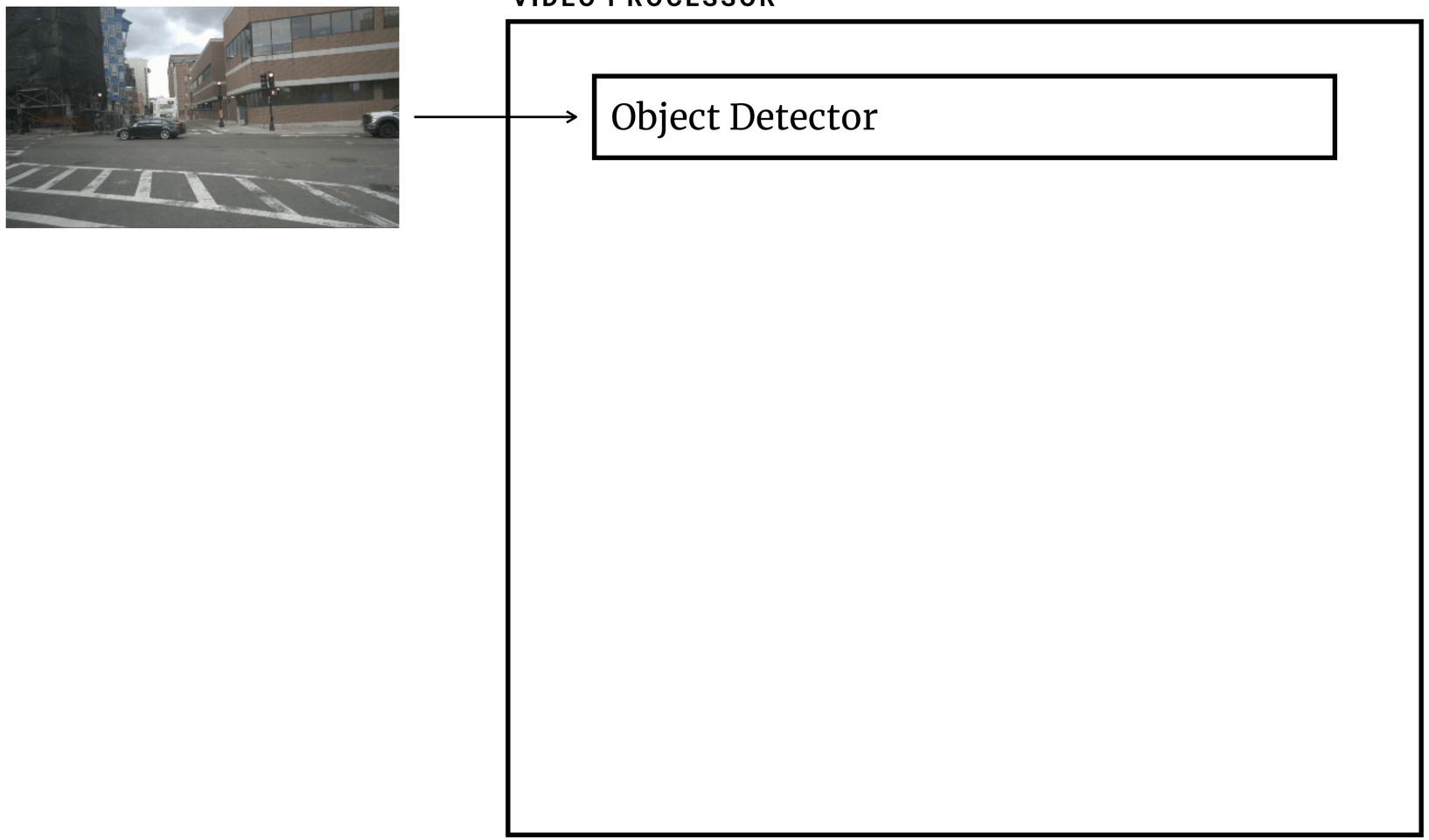
Video processing execution happens here!

SPATIALYZE

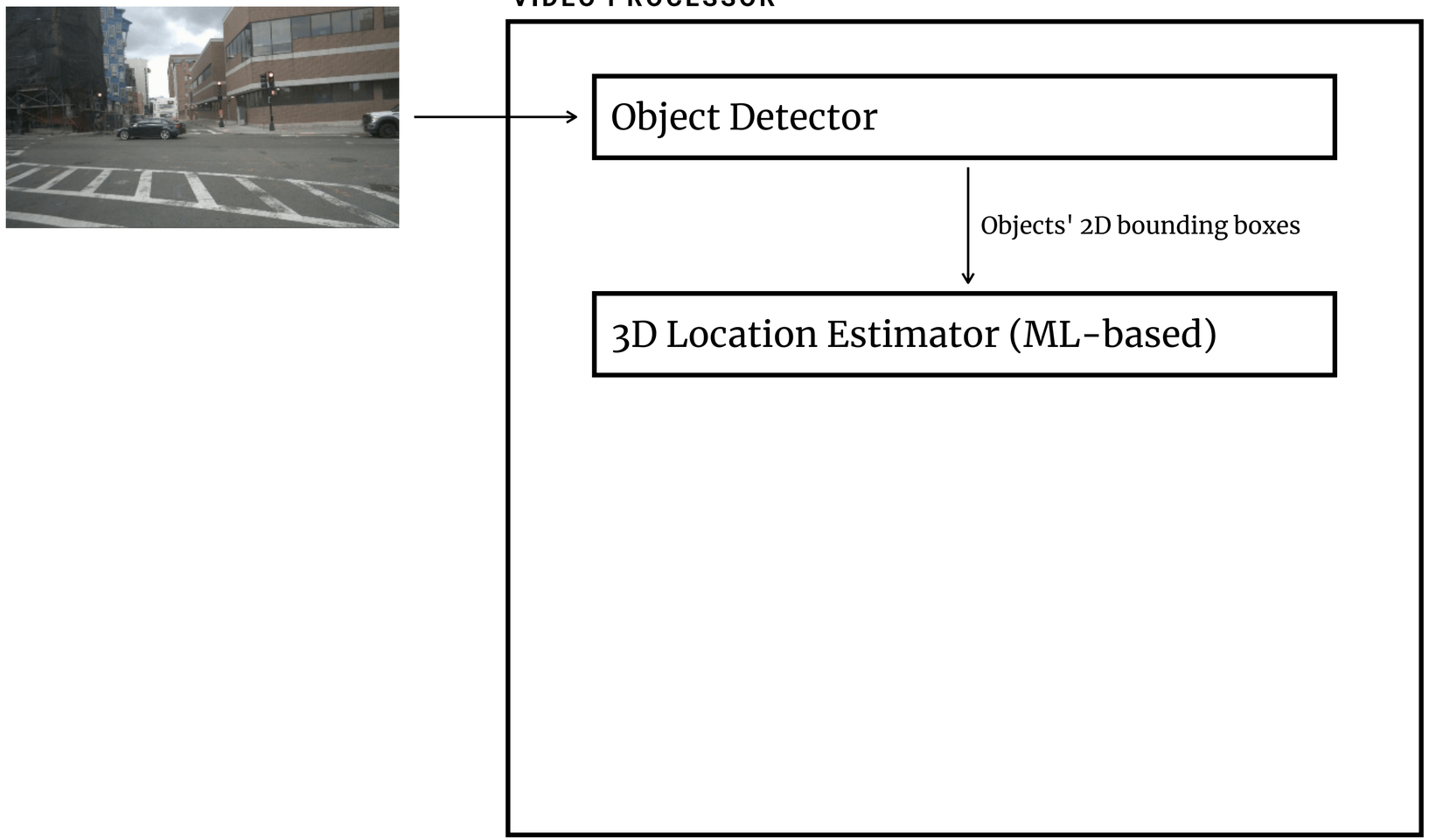
#3 System Optimization

* Video Processor
* Optimization Techniques

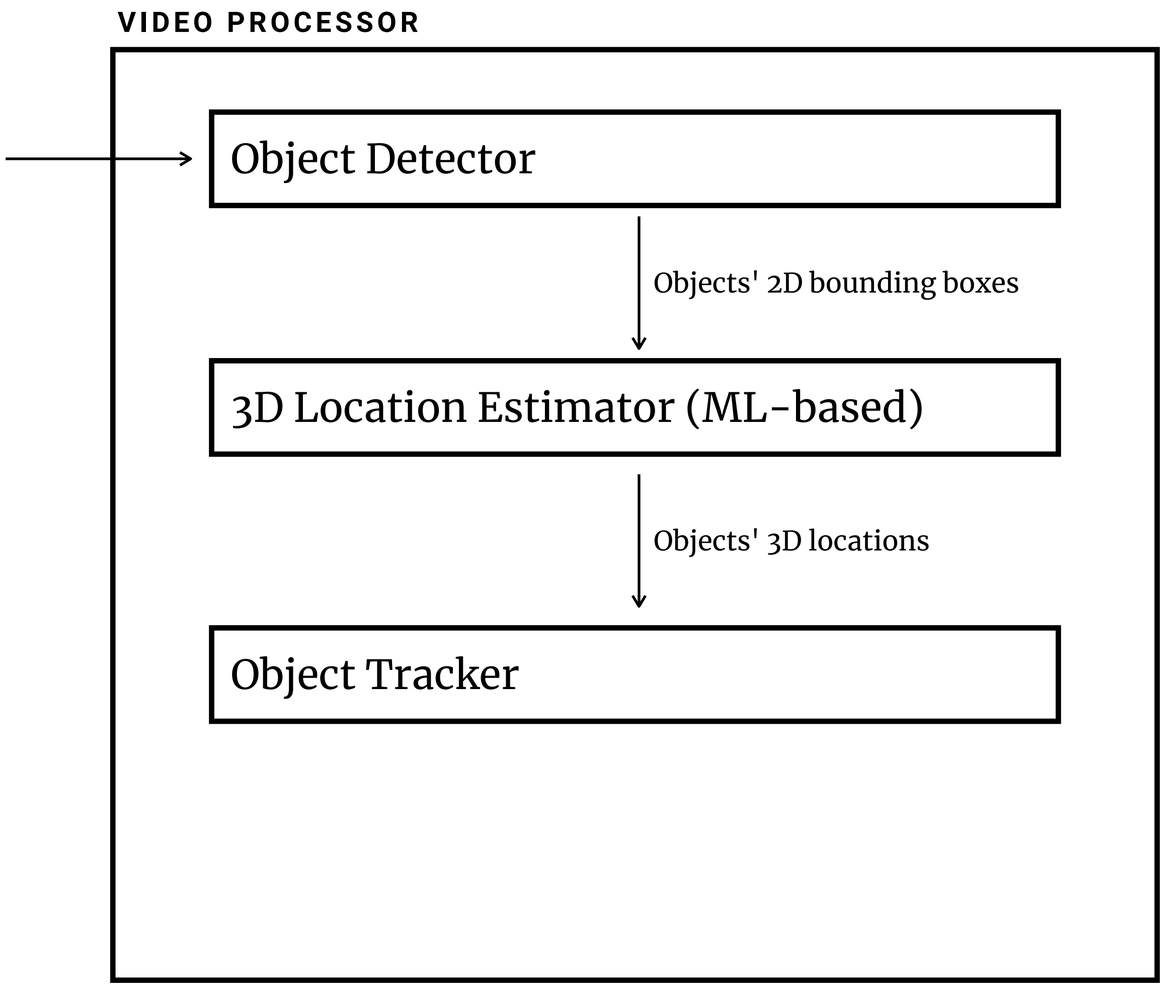
Video Processor



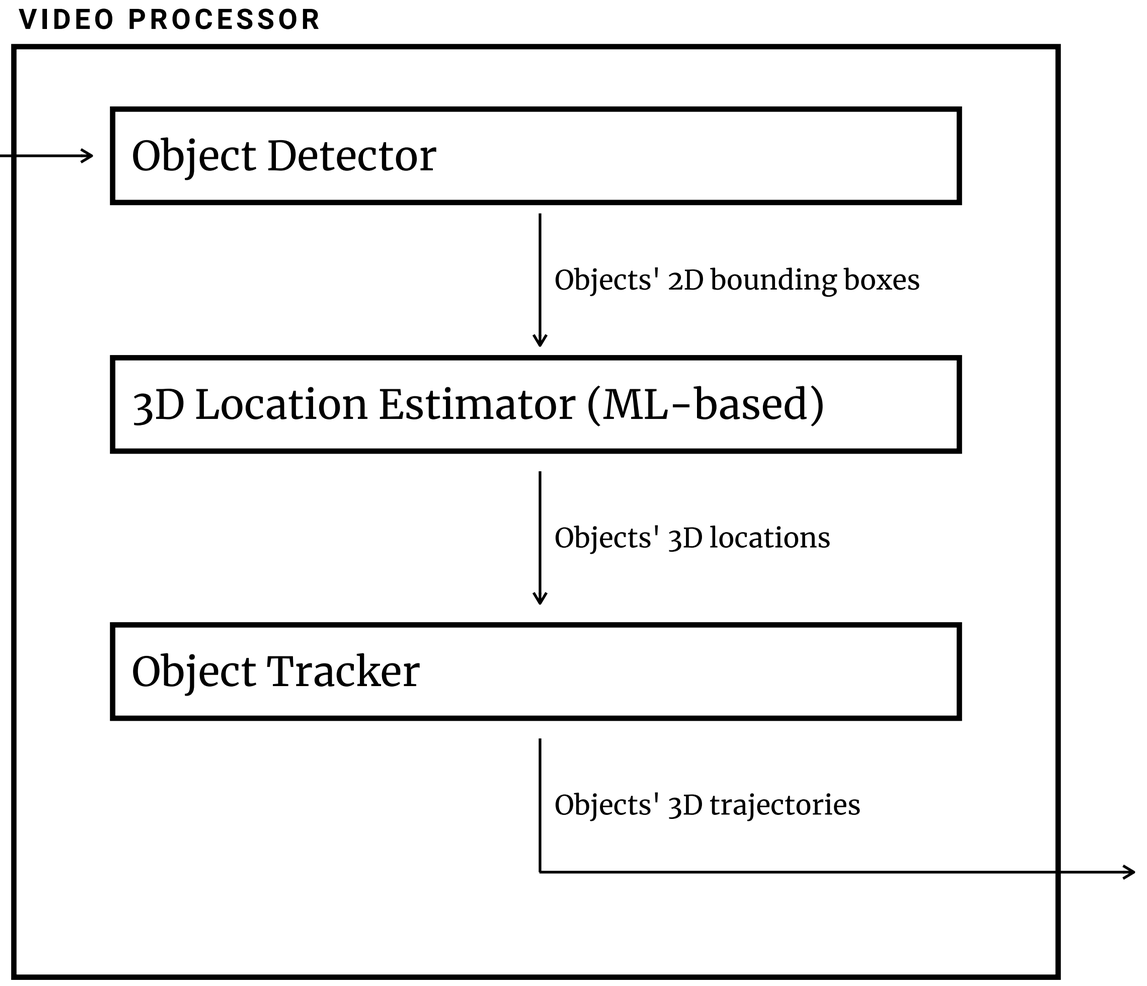
Video Processor

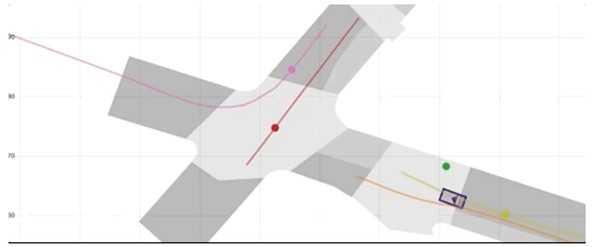


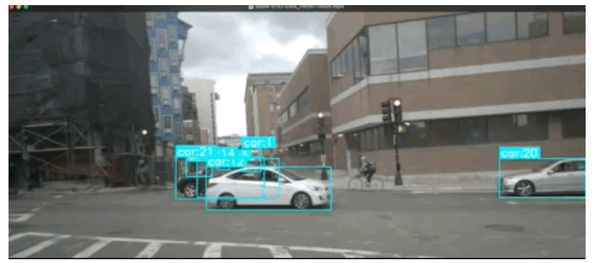
Video Processor



Video Processor







Video Processor

VIDEO PROCESSOR

Object Detector

Objects' 2D bounding boxes

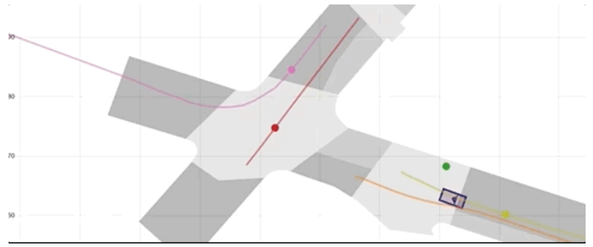
3D Location Estimator (ML-based)

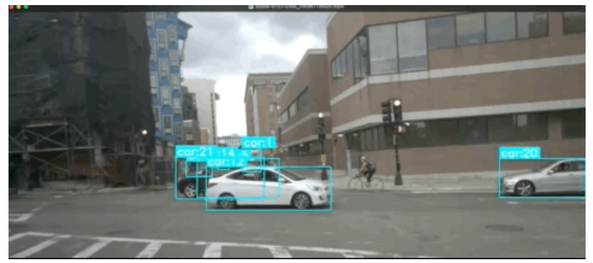
Objects' 3D locations

Object Tracker

Objects' 3D trajectories

Runtime depends on





Video Processor

VIDEO PROCESSOR

Object Detector

Objects' 2D bounding boxes

3D Location Estimator (ML-based)

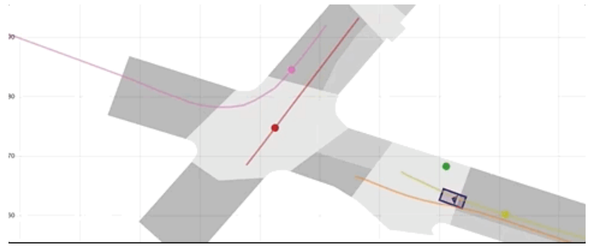
Objects' 3D locations

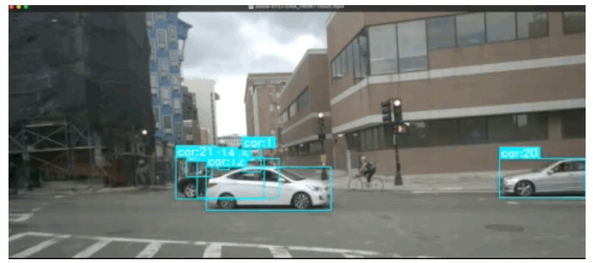
Object Tracker

Objects' 3D trajectories

Runtime depends on

* # of video frames





Video Processor

VIDEO PROCESSOR

Object Detector

Objects' 2D bounding boxes

3D Location Estimator (ML-based)

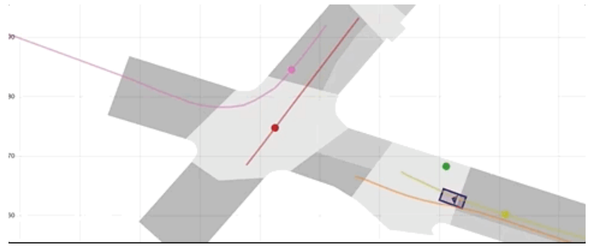
Objects' 3D locations

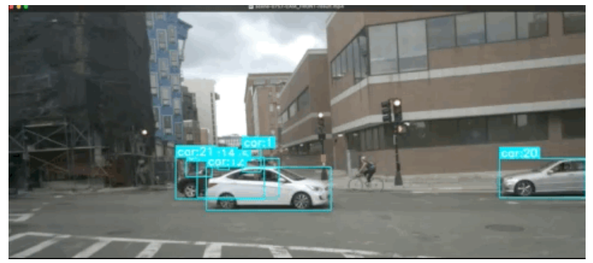
Object Tracker

Objects' 3D trajectories

Runtime depends on

* # of video frames
* ML inference time on each video





Video Processor

VIDEO PROCESSOR

01 Road Visibility Pruner

↓

Object Detector

↓

3D Location Estimator (ML-based)

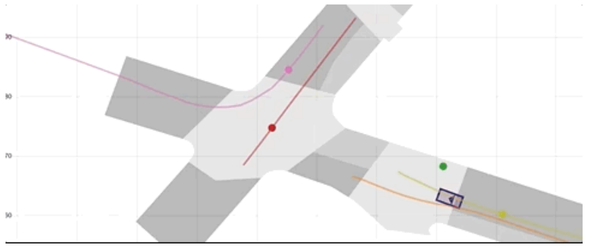
↓

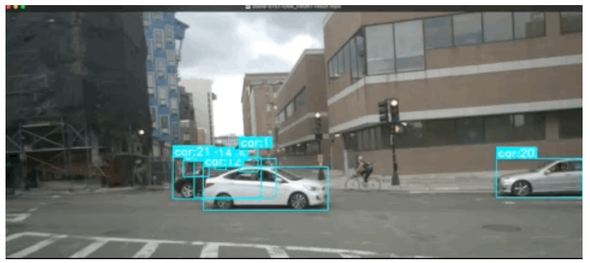
Object Tracker

→

Runtime depends on

* # of video frames 01
* ML inference time on each video





Video Processor

VIDEO PROCESSOR

01 Road Visibility Pruner

↓

Object Detector

↓

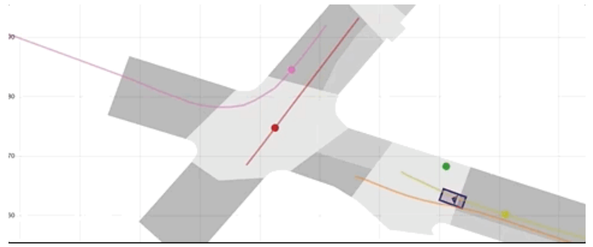
02 3D Location Estimator (with geospatial metadata)

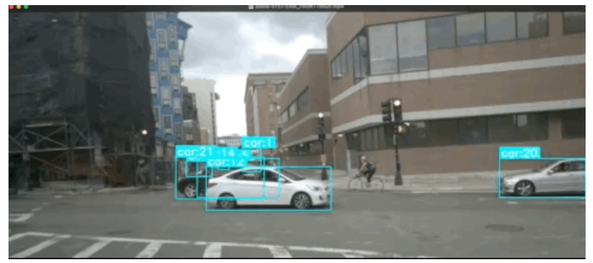
↓

Object Tracker

Runtime depends on

* # of video frames 01
* ML inference time on each video 02





Optimization Techniques

01 Road Visibility Pruner

QUERY

```
contains(intersection, car)
```


IMPLICATION

The car must be at an intersection → The intersection must be in the camera's view

Optimization Techniques

01 Road Visibility Pruner

QUERY

```
contains(intersection, car)
```

IMPLICATION

The car must be at an intersection → The intersection must be in the camera's view

Optimization Techniques

01 Road Visibility Pruner

QUERY

```
contains(intersection, car)
```

IMPLICATION

The car must be at an intersection → The intersection must be in the camera's view

Optimization Techniques

01 Road Visibility Pruner

QUERY

```
contains(intersection, car)
```

IMPLICATION

The car must be at an intersection → The intersection must be in the camera's view

Optimization Techniques

01 Road Visibility Pruner

QUERY

```
contains(intersection, car)
```

IMPLICATION

The car must be at an intersection → The intersection must be in the camera's view

Optimization Techniques

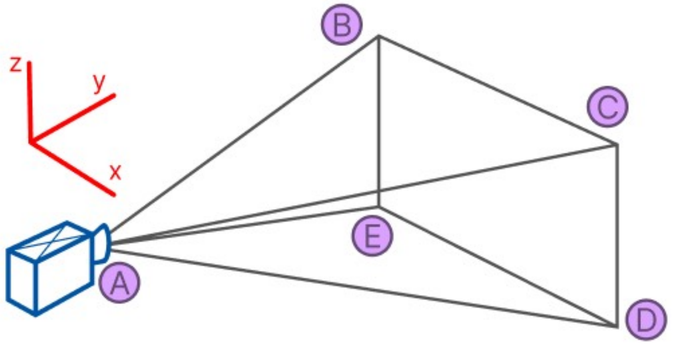
01 Road Visibility Pruner

"Geographic constructs' visibility is a proxy for objects' visibility"

Optimization Techniques

01 Road Visibility Pruner

"Geographic constructs' visibility is a proxy for objects' visibility"



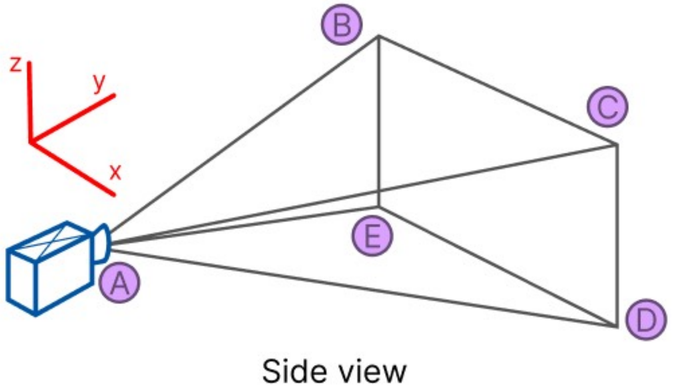
Side view

Top view

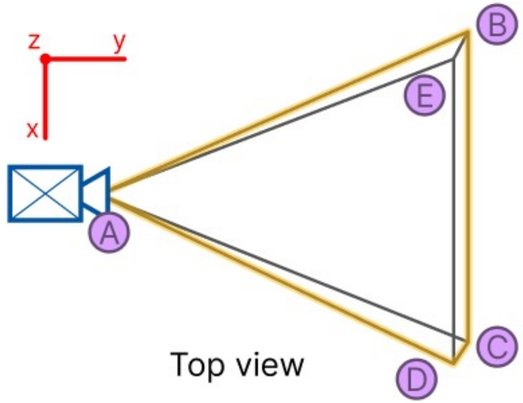
Optimization Techniques

01 Road Visibility Pruner

"Geographic constructs' visibility is a proxy for objects' visibility"



Side view

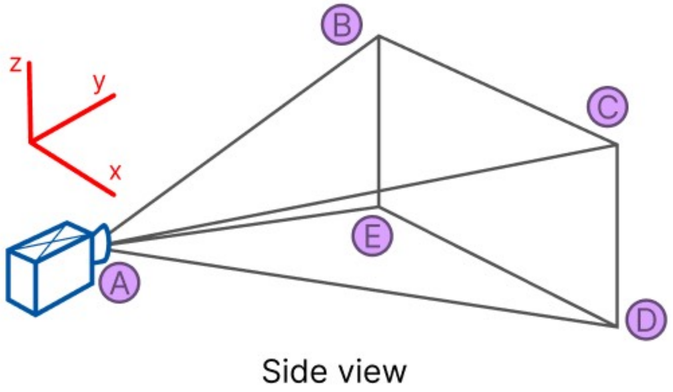


Top view

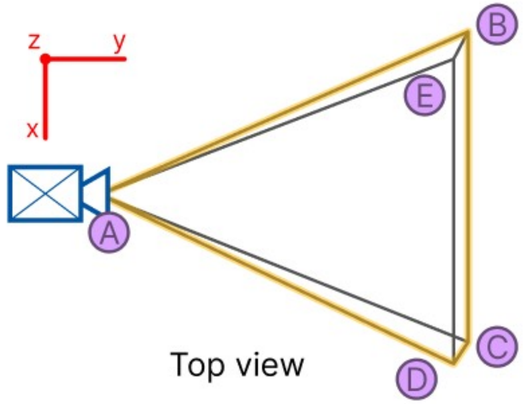
Optimization Techniques

01 Road Visibility Pruner

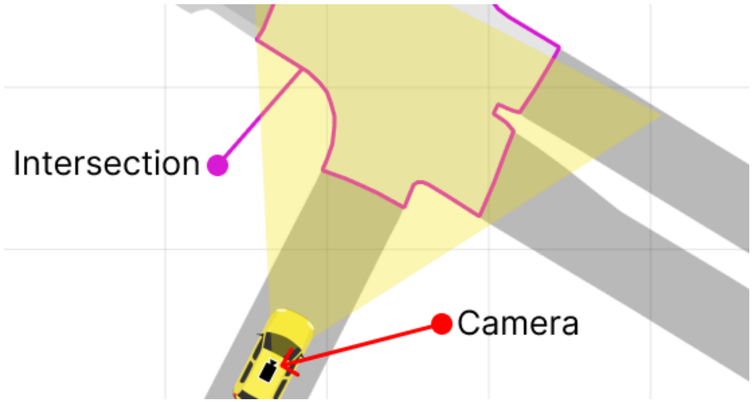
"Geographic constructs' visibility is a proxy for objects' visibility"



Side view



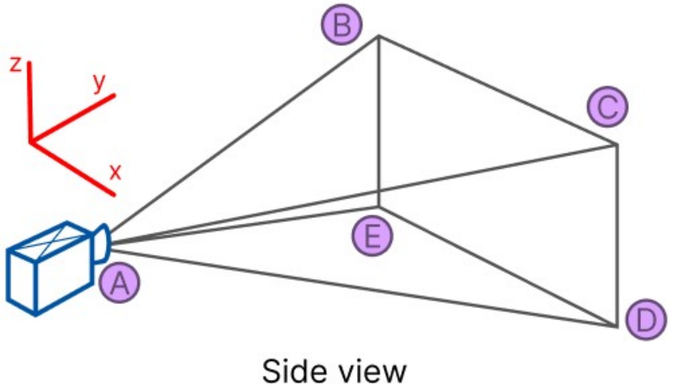
Top view



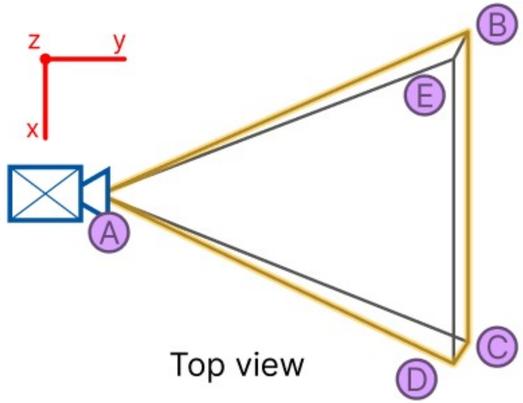
Optimization Techniques

01 Road Visibility Pruner

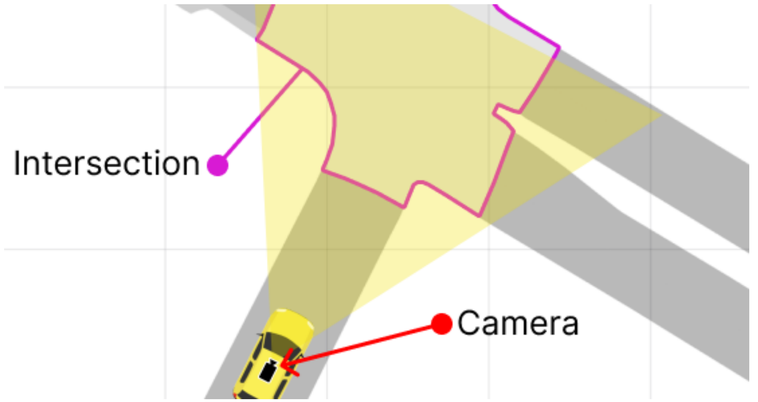
"Geographic constructs' visibility is a proxy for objects' visibility"

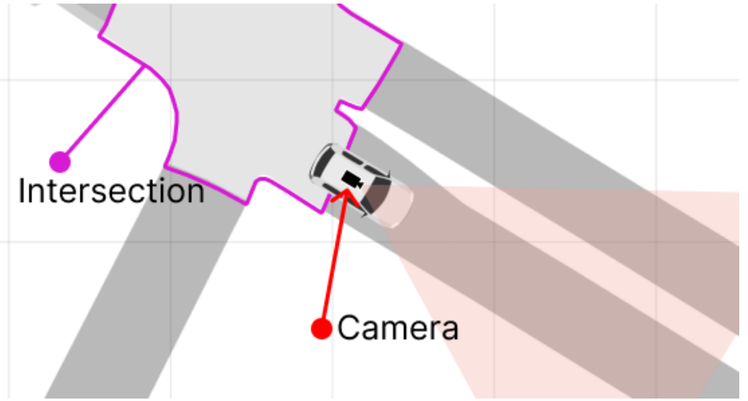


Side view



Top view

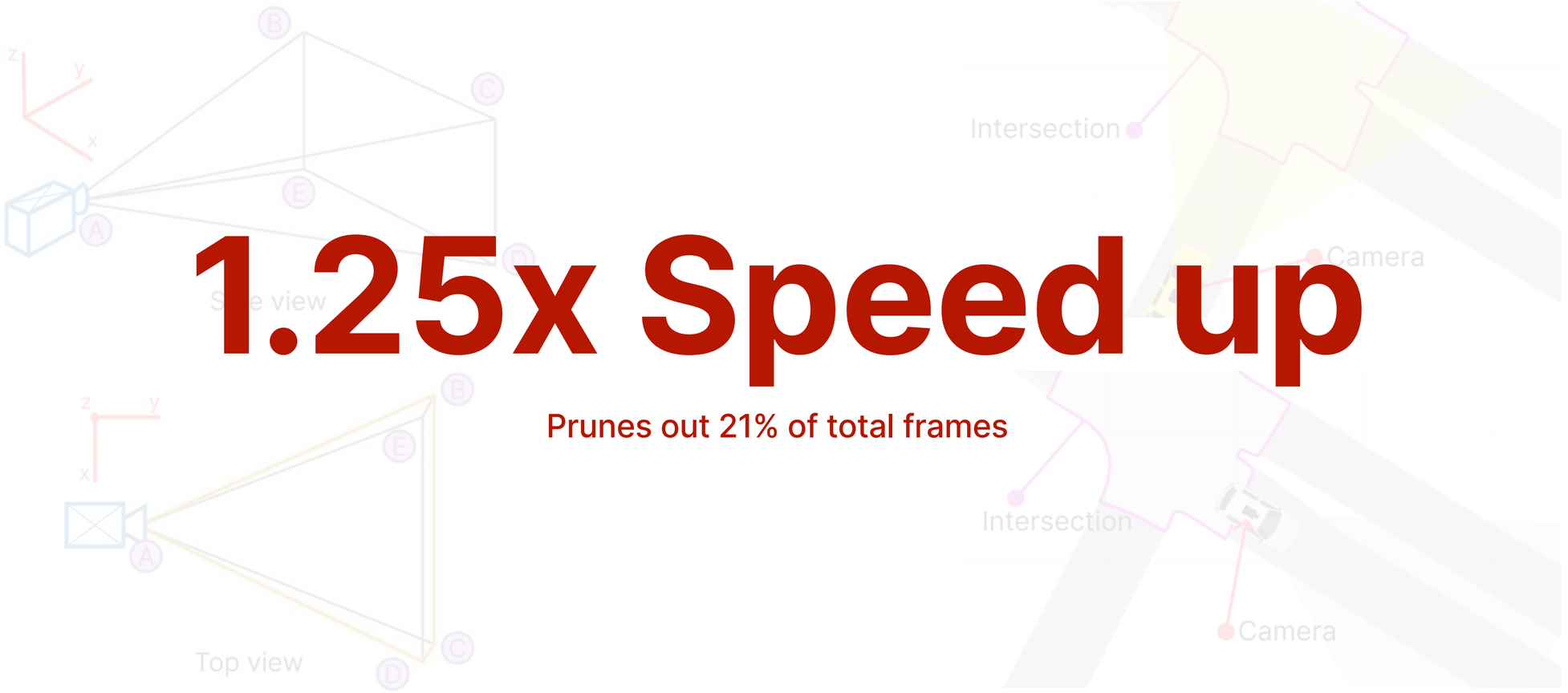




Optimization Techniques

01 Road Visibility Pruner

"Geographic constructs' visibility is a proxy for objects' visibility"



1.25x Speed up

Prunes out 21% of total frames

Optimization Techniques

03 3D Location Estimator (with geospatial metadata)

QUERY

```
(o.type=='car') | (o.type=='person')
```

"If objects of interest are on the ground, we can recover their 3D location based on their 2D pixels bounding box and camera parameters"

Optimization Techniques

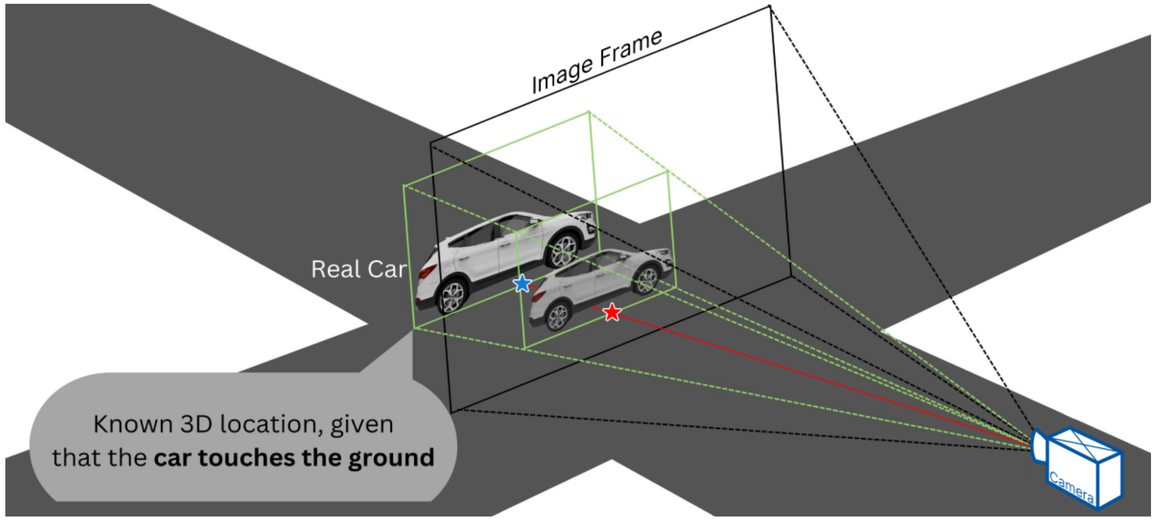
03 3D Location Estimator (with geospatial metadata)

"If objects of interest are on the ground, we can recover their 3D location based on their 2D pixels bounding box and camera parameters"

Optimization Techniques

03 3D Location Estimator (with geospatial metadata)

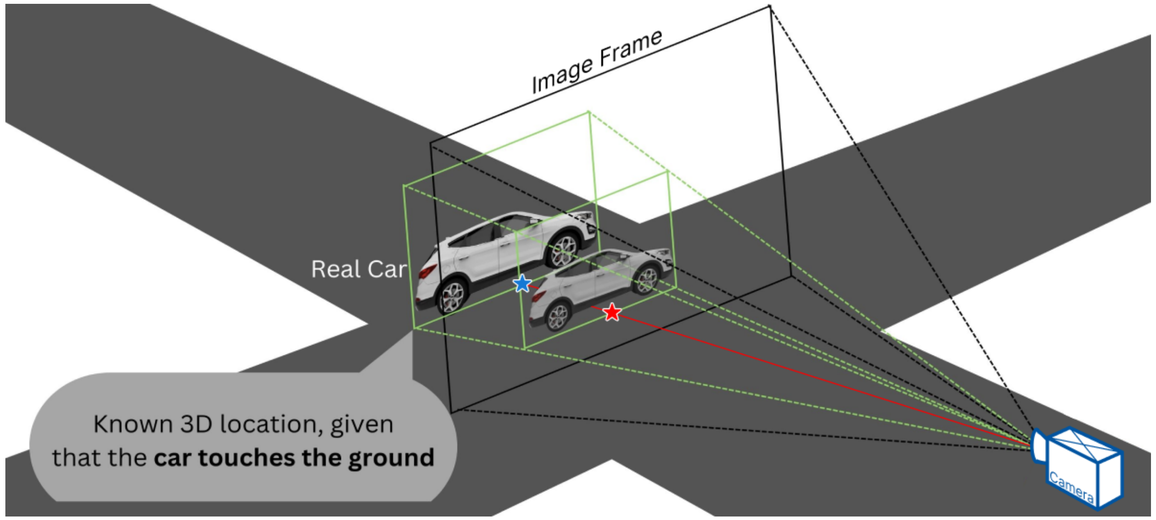
"If objects of interest are on the ground, we can recover their 3D location based on their 2D pixels bounding box and camera parameters"

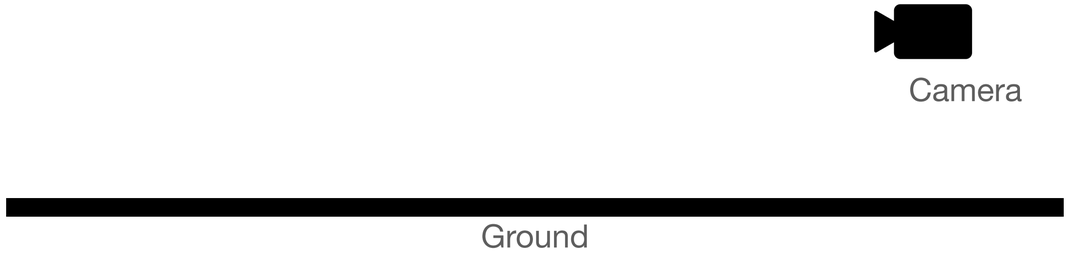


Optimization Techniques

03 3D Location Estimator (with geospatial metadata)

"If objects of interest are on the ground, we can recover their 3D location based on their 2D pixels bounding box and camera parameters"

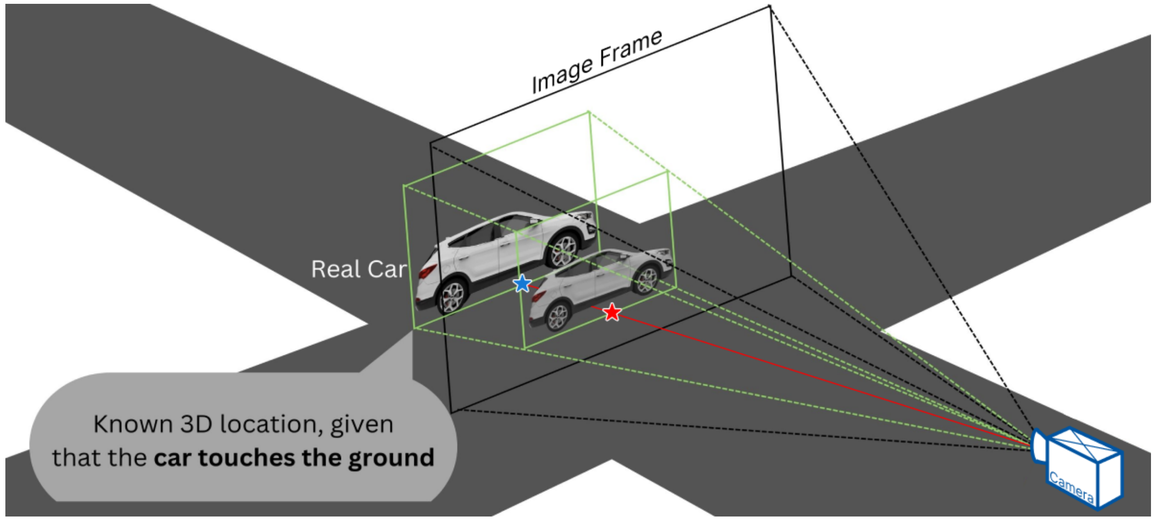


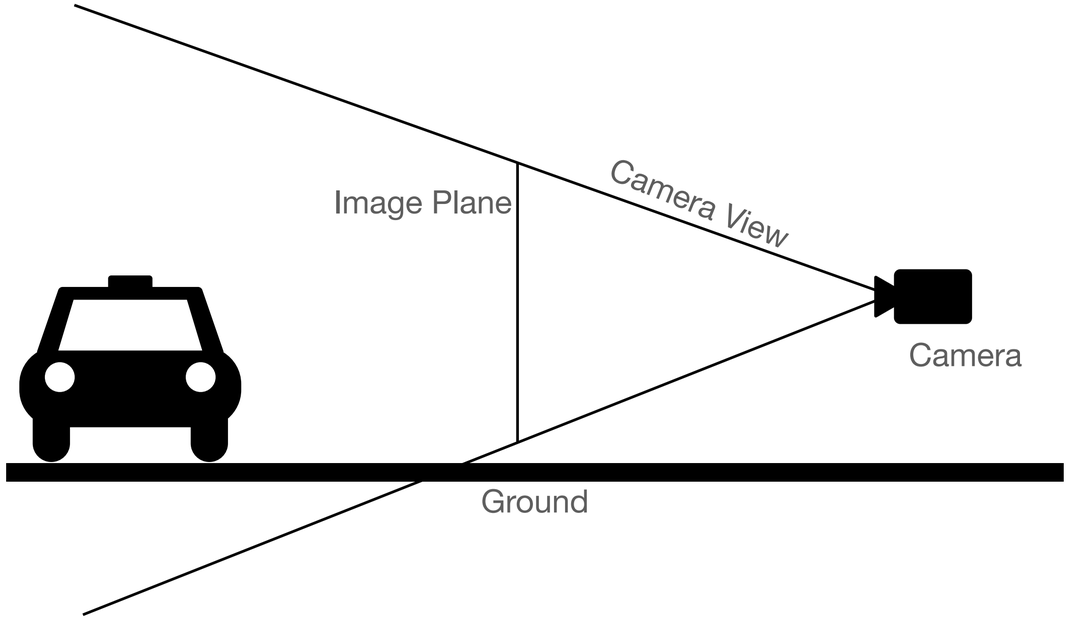


Optimization Techniques

03 3D Location Estimator (with geospatial metadata)

"If objects of interest are on the ground, we can recover their 3D location based on their 2D pixels bounding box and camera parameters"

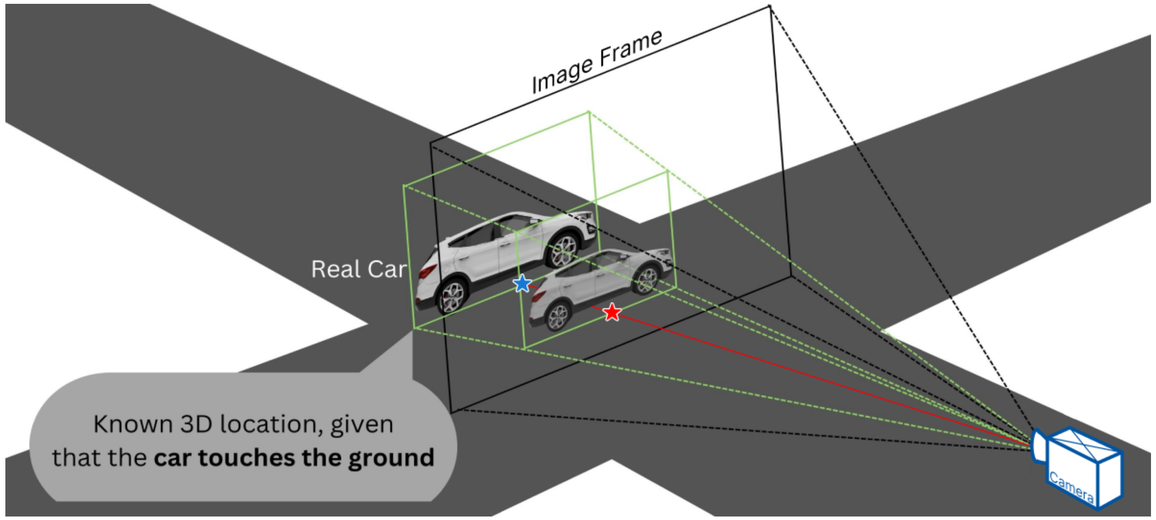


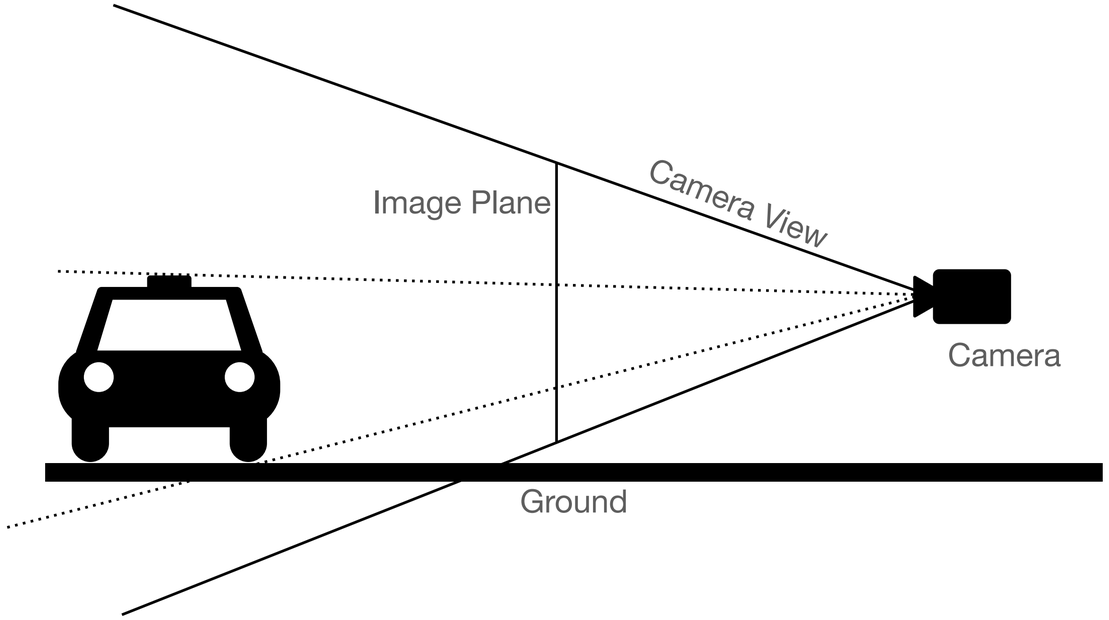


Optimization Techniques

03 3D Location Estimator (with geospatial metadata)

"If objects of interest are on the ground, we can recover their 3D location based on their 2D pixels bounding box and camera parameters"

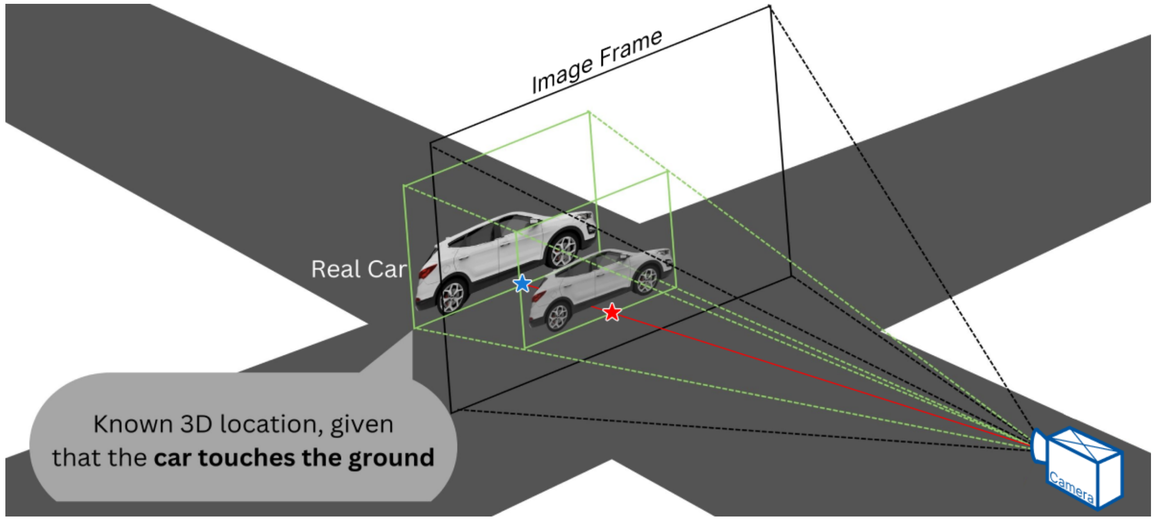


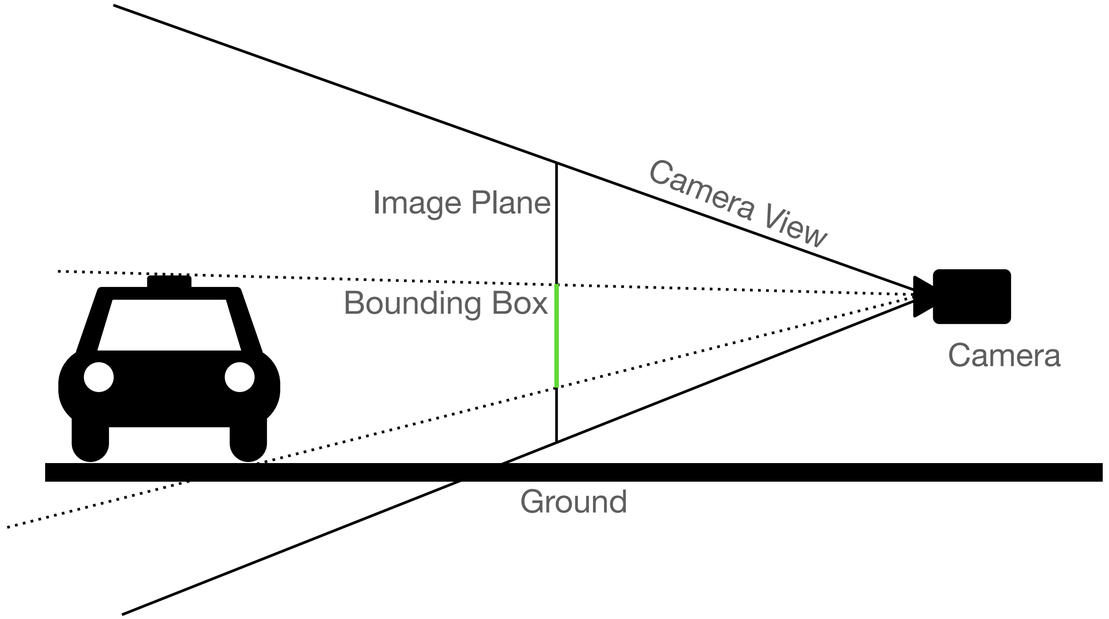


Optimization Techniques

03 3D Location Estimator (with geospatial metadata)

"If objects of interest are on the ground, we can recover their 3D location based on their 2D pixels bounding box and camera parameters"

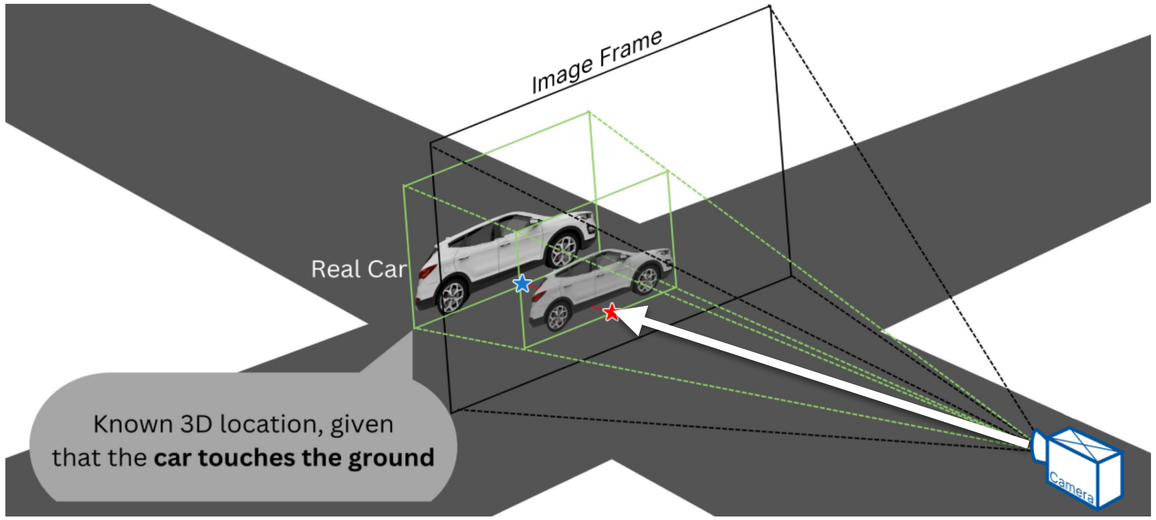


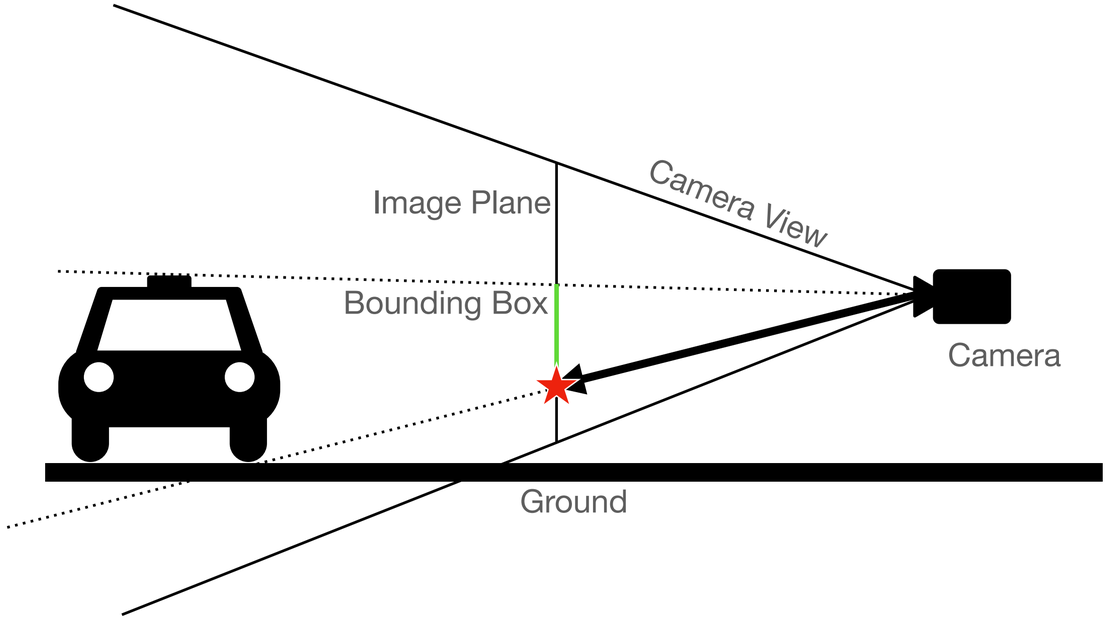


Optimization Techniques

03 3D Location Estimator (with geospatial metadata)

"If objects of interest are on the ground, we can recover their 3D location based on their 2D pixels bounding box and camera parameters"

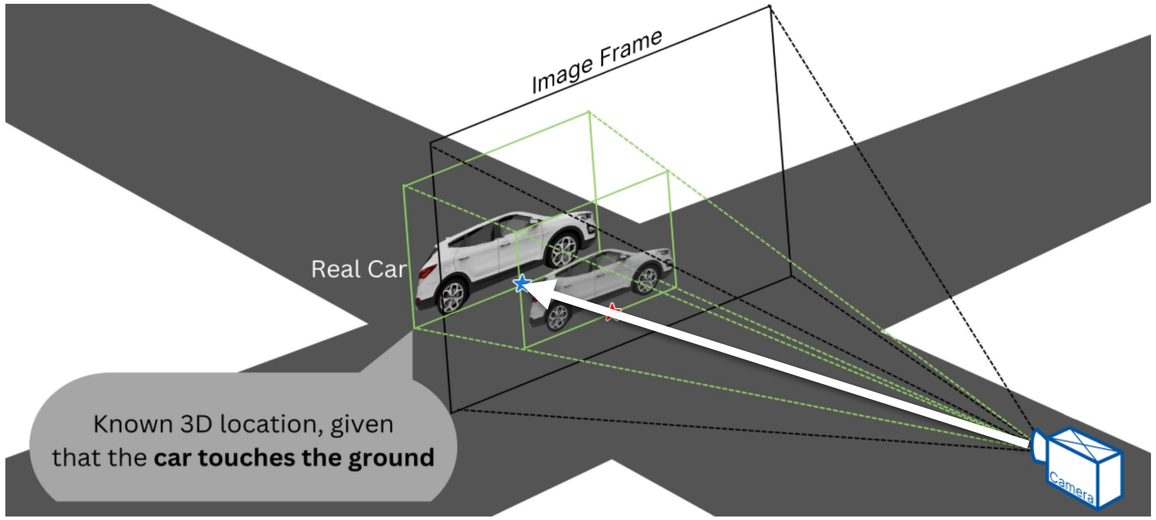


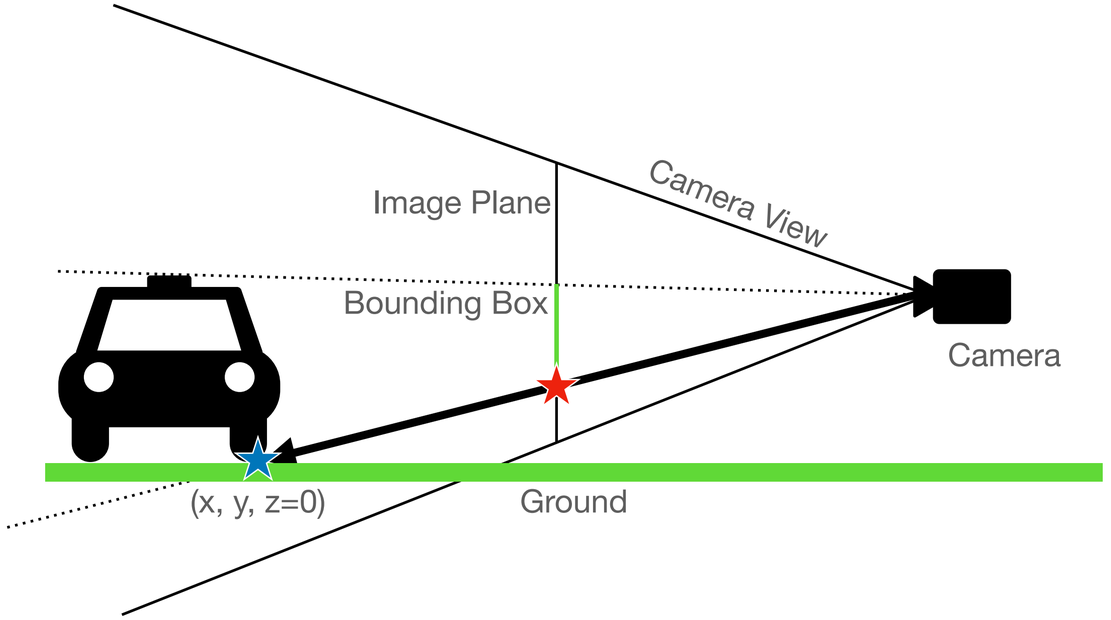


Optimization Techniques

03 3D Location Estimator (with geospatial metadata)

"If objects of interest are on the ground, we can recover their 3D location based on their 2D pixels bounding box and camera parameters"

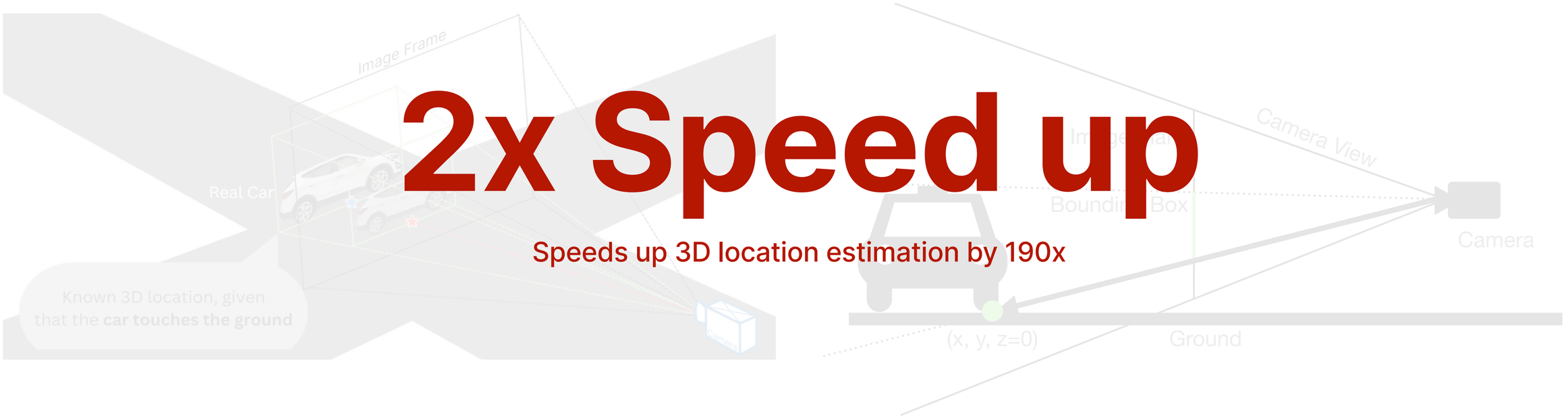




Optimization Techniques

03 3D Location Estimator (with geospatial metadata)

"If objects of interest are on the ground, we can recover their 3D location based on their 2D pixels bounding box and camera parameters"



2x Speed up

Speeds up 3D location estimation by 190x

SPATIALYZE

#4 Evaluation

* Experiment Setups
* Results

#4 EVALUATION

Experiment Setups

VIVA

End-to-End video analytic system

VIVA: An End-to-End System for Interactive Video Analytics

Daniel Kang*
ddkang@cs.stanford.edu
Stanford University

Francisco Romero*
faromero@stanford.edu
Stanford University

Peter Bailis
pbailis@cs.stanford.edu
Stanford University

Christos Kozyrakis
christos@cs.stanford.edu
Stanford University

Matei Zaharia
matei@cs.stanford.edu
Stanford University and Databricks

ABSTRACT

The growth of video volumes and increased DNN capabilities has led to a growing desire for video analytics. In response, the data analytics community has proposed multiple systems that optimize specific query types (e.g., selection queries) or a particular step in query execution (e.g., video retrieval from storage). However, none of these systems provide *end-to-end, practical* video analytics for users to iteratively and interactively engage with queries, as is the case with analytics systems for structured data.

In response, we are building VIVA: an end-to-end system for interactive video analytics. VIVA contains five novel components. First, VIVA uses *relational hints*, which allow users to express relationships between columns that are difficult to automatically infer (e.g., mentions of a person in a transcript can be used as a proxy for the person appearing in the video). Second, VIVA introduces a *mixed-data query optimizer* that optimizes queries across both structured and unstructured data. Third, VIVA features an *embedding cache* that decides which results/embeddings to store for future queries. Finally, VIVA co-optimizes storage and query execution with its *video file manager* and *accelerator-based execution engine*. The former decides how to pre-fetch/manage video, while the latter selects and manages heterogeneous hardware backends spanning the growing number of DNN accelerators. We describe the challenges and design requirements for VIVA's development and outline ongoing and future work for realizing VIVA.

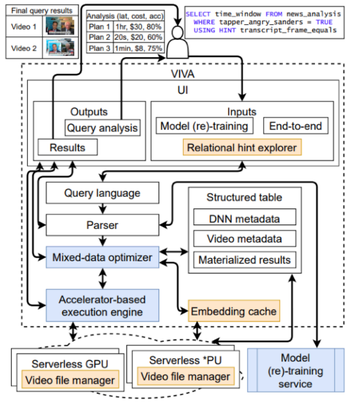


Figure 1: VIVA's architecture diagram. Blue components/interfaces are typically found in analytic systems for structured data that require rethinking. Orange components/interfaces are novel to VIVA.

SkyQuery

Geospatial video query processor

SkyQuery: An Aerial Drone Video Sensing Platform

Favyen Bastani
MIT CSAIL
US
favyen@csail.mit.edu

Songtao He
MIT CSAIL
US
songtao@csail.mit.edu

Ziwen Jiang
MIT CSAIL
US
ziwenj@csail.mit.edu

Osbert Bastani
University of Pennsylvania
US
obastani@seas.upenn.edu

Sam Madden
MIT CSAIL
US
madden@csail.mit.edu

Abstract

Video-based sensing from aerial drones, especially small multirotor drones, can provide rich data for numerous applications, including traffic analysis (computing traffic flow volumes), precision agriculture (periodically evaluating plant health), and wildlife population management (estimating population sizes). However, aerial drone video sensing applications must handle a surprisingly wide range of tasks: video frames must be aligned so that we can equate coordinates of objects that appear in different frames, video data must be analyzed to extract application-specific insights, and drone routes must be computed that maximize the value of newly captured video. To address these challenges, we built *SkyQuery*, a novel aerial drone video sensing platform that provides an expressive, high-level programming language to make it straightforward for users to develop complex long-running sensing applications. SkyQuery combines novel methods for fast video frame alignment and detection of small objects in top-down aerial drone video to efficiently execute applications with diverse video analysis workflows and data distributions, thereby allowing application developers to focus on the unique qualities of their particular application rather than general video processing, data analy-

ACM Reference Format:
Favyen Bastani, Songtao He, Ziwen Jiang, Osbert Bastani, and Sam Madden. 2021. SkyQuery: An Aerial Drone Video Sensing Platform. In *Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '21), October 20–22, 2021, Chicago, IL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3486607.3486750

1 Introduction

When equipped with sensors, computation, and communication capabilities, small multirotor aerial drones are an ideal mobile sensing platform for many applications. Indeed, video sensing on aerial drones has been proposed for a number of tasks, including *earth science*, for monitoring coastal sea ice and habitat loss [24]; *disaster emergency response*, for tracking floods, fires, and people in search-and-rescue [8]; *automated mapping*, for measuring traffic incidents, congestion, and parking [17]; *civil infrastructure monitoring*, for finding defects in bridges, buildings, and pipelines [25]; and *precision agriculture*, for micro-climate monitoring of plant growth [4].

However, realizing applications that involve extracting insights from aerial drone video remains difficult; in fact, every

#4 EVALUATION

Experiment Setups

VIVA

End-to-End video analytic system

VIVA: An End-to-End System for Interactive Video Analytics

Daniel Kang*
ddkang@cs.stanford.edu
Stanford University

Francisco Romero*
faromero@stanford.edu
Stanford University

Peter Bailis
pbailis@cs.stanford.edu
Stanford University

Christos Kozyrakis
christos@cs.stanford.edu
Stanford University

Matei Zaharia
matei@cs.stanford.edu
Stanford University and Databricks

ABSTRACT

The growth of video volumes and increased DNN capabilities has led to a growing desire for video analytics. In response, the data analytics community has proposed multiple systems that optimize specific query types (e.g., selection queries) or a particular step in query execution (e.g., video retrieval from storage). However, none of these systems provide *end-to-end, practical* video analytics for users to iteratively and interactively engage with queries, as is the case with analytics systems for structured data.

In response, we are building VIVA: an end-to-end system for interactive video analytics. VIVA contains five novel components. First, VIVA uses *relational hints*, which allow users to express relationships between columns that are difficult to automatically infer (e.g., mentions of a person in a transcript can be used as a proxy for the person appearing in the video). Second, VIVA introduces a *mixed-data query optimizer* that optimizes queries across both structured and unstructured data. Third, VIVA features an *embedding cache* that decides which results/embeddings to store for future queries. Finally, VIVA co-optimizes storage and query execution with its *video file manager* and *accelerator-based execution engine*. The former decides how to pre-fetch/manage video, while the latter selects and manages heterogeneous hardware backends spanning the growing number of DNN accelerators. We describe the challenges and design requirements for VIVA's development and outline ongoing and future work for realizing VIVA.

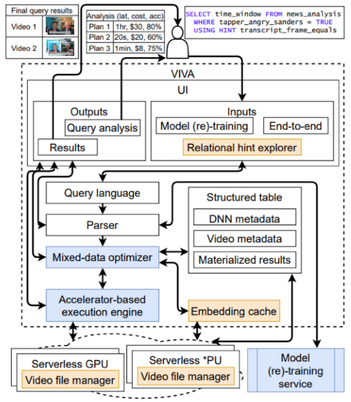


Figure 1: VIVA's architecture diagram. Blue components/interfaces are typically found in analytic systems for structured data that require rethinking. Orange components/interfaces are novel to VIVA.

SkyQuery

Geospatial video query processor

SkyQuery: An Aerial Drone Video Sensing Platform

Favyen Bastani
MIT CSAIL
US
favyen@csail.mit.edu

Songtao He
MIT CSAIL
US
songtao@csail.mit.edu

Ziwen Jiang
MIT CSAIL
US
ziwenj@csail.mit.edu

Osbert Bastani
University of Pennsylvania
US
obastani@seas.upenn.edu

Sam Madden
MIT CSAIL
US
madden@csail.mit.edu

Abstract

Video-based sensing from aerial drones, especially small multirotor drones, can provide rich data for numerous applications, including traffic analysis (computing traffic flow volumes), precision agriculture (periodically evaluating plant health), and wildlife population management (estimating population sizes). However, aerial drone video sensing applications must handle a surprisingly wide range of tasks: video frames must be aligned so that we can equate coordinates of objects that appear in different frames, video data must be analyzed to extract application-specific insights, and drone routes must be computed that maximize the value of newly captured video. To address these challenges, we built *SkyQuery*, a novel aerial drone video sensing platform that provides an expressive, high-level programming language to make it straightforward for users to develop complex long-running sensing applications. SkyQuery combines novel methods for fast video frame alignment and detection of small objects in top-down aerial drone video to efficiently execute applications with diverse video analysis workflows and data distributions, thereby allowing application developers to focus on the unique qualities of their particular application rather than general video processing, data analy-

ACM Reference Format:
Favyen Bastani, Songtao He, Ziwen Jiang, Osbert Bastani, and Sam Madden. 2021. SkyQuery: An Aerial Drone Video Sensing Platform. In *Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '21), October 20–22, 2021, Chicago, IL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3486607.3486750

1 Introduction

When equipped with sensors, computation, and communication capabilities, small multirotor aerial drones are an ideal mobile sensing platform for many applications. Indeed, video sensing on aerial drones has been proposed for a number of tasks, including *earth science*, for monitoring coastal sea ice and habitat loss [24]; *disaster emergency response*, for tracking floods, fires, and people in search-and-rescue [8]; *automated mapping*, for measuring traffic incidents, congestion, and parking [17]; *civil infrastructure monitoring*, for finding defects in bridges, buildings, and pipelines [25]; and *precision agriculture*, for micro-climate monitoring of plant growth [4].

However, realizing applications that involve extracting insights from aerial drone video remains difficult; in fact, every

#4 EVALUATION

Experiment Setups

VIVA

End-to-End video analytic system

VIVA: An End-to-End System for Interactive Video Analytics

Daniel Kang*
ddkang@cs.stanford.edu
Stanford University

Francisco Romero*
faromero@stanford.edu
Stanford University

Peter Bailis
pbailis@cs.stanford.edu
Stanford University

Christos Kozyrakis
christos@cs.stanford.edu
Stanford University

Matei Zaharia
matei@cs.stanford.edu
Stanford University and Databricks

ABSTRACT

The growth of video volumes and increased DNN capabilities has led to a growing desire for video analytics. In response, the data analytics community has proposed multiple systems that optimize specific query types (e.g., selection queries) or a particular step in query execution (e.g., video retrieval from storage). However, none of these systems provide *end-to-end, practical* video analytics for users to iteratively and interactively engage with queries, as is the case with analytics systems for structured data.

In response, we are building VIVA: an end-to-end system for interactive video analytics. VIVA contains five novel components. First, VIVA uses *relational hints*, which allow users to express relationships between columns that are difficult to automatically infer (e.g., mentions of a person in a transcript can be used as a proxy for the person appearing in the video). Second, VIVA introduces a *mixed-data query optimizer* that optimizes queries across both structured and unstructured data. Third, VIVA features an *embedding cache* that decides which results/embeddings to store for future queries. Finally, VIVA co-optimizes storage and query execution with its *video file manager* and *accelerator-based execution engine*. The former decides how to pre-fetch/manage video, while the latter selects and manages heterogeneous hardware backends spanning the growing number of DNN accelerators. We describe the challenges and design requirements for VIVA's development and outline ongoing and future work for realizing VIVA.

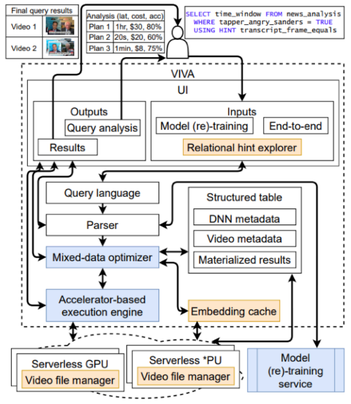


Figure 1: VIVA's architecture diagram. Blue components/interfaces are typically found in analytic systems for structured data that require rethinking. Orange components/interfaces are novel to VIVA.

SkyQuery

Geospatial video query processor

SkyQuery: An Aerial Drone Video Sensing Platform

Favyen Bastani
MIT CSAIL
US
favyen@csail.mit.edu

Songtao He
MIT CSAIL
US
songtao@csail.mit.edu

Ziwen Jiang
MIT CSAIL
US
ziwenj@csail.mit.edu

Osbert Bastani
University of Pennsylvania
US
obastani@seas.upenn.edu

Sam Madden
MIT CSAIL
US
madden@csail.mit.edu

Abstract

Video-based sensing from aerial drones, especially small multirotor drones, can provide rich data for numerous applications, including traffic analysis (computing traffic flow volumes), precision agriculture (periodically evaluating plant health), and wildlife population management (estimating population sizes). However, aerial drone video sensing applications must handle a surprisingly wide range of tasks: video frames must be aligned so that we can equate coordinates of objects that appear in different frames, video data must be analyzed to extract application-specific insights, and drone routes must be computed that maximize the value of newly captured video. To address these challenges, we built *SkyQuery*, a novel aerial drone video sensing platform that provides an expressive, high-level programming language to make it straightforward for users to develop complex long-running sensing applications. SkyQuery combines novel methods for fast video frame alignment and detection of small objects in top-down aerial drone video to efficiently execute applications with diverse video analysis workflows and data distributions, thereby allowing application developers to focus on the unique qualities of their particular application rather than general video processing, data analy-

ACM Reference Format:
Favyen Bastani, Songtao He, Ziwen Jiang, Osbert Bastani, and Sam Madden. 2021. SkyQuery: An Aerial Drone Video Sensing Platform. In *Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '21), October 20–22, 2021, Chicago, IL, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3486607.3486750

1 Introduction

When equipped with sensors, computation, and communication capabilities, small multirotor aerial drones are an ideal mobile sensing platform for many applications. Indeed, video sensing on aerial drones has been proposed for a number of tasks, including *earth science*, for monitoring coastal sea ice and habitat loss [24]; *disaster emergency response*, for tracking floods, fires, and people in search-and-rescue [8]; *automated mapping*, for measuring traffic incidents, congestion, and parking [17]; *civil infrastructure monitoring*, for finding defects in bridges, buildings, and pipelines [25]; and *precision agriculture*, for micro-climate monitoring of plant growth [4].

However, realizing applications that involve extracting insights from aerial drone video remains difficult; in fact, every

Experiment Setups

Ablation

Compare with no Optimization

Q1 A pedestrian at an intersection moving perpendicularly to the camera

```
p = w.object(); c = w.camera(); i = w.geoConstruct('intersection')
w.filter(p.type=='person' & perpendicular(p, c)
         & contains(i, p))
```

SkyQuery

Geospatial video query processor

Q3 A car stopped in a cycling lane.

```
c = w.camera(); b = w.geoConstruct('bike-lane')
w.filter(p.type=='car' & contains(b, c))
```

VIVA

End-to-End video analytic system

Q2 A car turning left with a pedestrian at an intersection.

```
c = w.object(); p = w.object(); i = w.geoConstruct('intersection')
w.filter(c.type=='car' & p.type='people' & turnLeft(c)
         & contains(i, [c, p]))
```

#4 EVALUATION

Experiment Setups

Ablation

Compare with no Optimization

Q1 A pedestrian at an intersection moving perpendicularly to the camera

```
p = w.object(); c = w.camera(); i = w.geoConstruct('intersection')
w.filter(p.type=='person' & perpendicular(p, c)
         & contains(i, p))
```

VIVA

End-to-End video analytic system

Q2 A car turning left with a pedestrian at an intersection.

```
c = w.object(); p = w.object(); i = w.geoConstruct('intersection')
w.filter(c.type=='car' & p.type='people' & turnLeft(c)
         & contains(i, [c, p]))
```

SkyQuery

Geospatial video query processor

Q3 A car stopped in a cycling lane.

```
c = w.camera(); b = w.geoConstruct('bike-lane')
w.filter(p.type=='car' & contains(b, c))
```

Experiment Setups

Ablation

Compare with no Optimization

Q1 A pedestrian at an intersection moving perpendicularly to the camera

```
p = w.object(); c = w.camera(); i = w.geoConstruct('intersection')
w.filter(p.type=='person' & perpendicular(p, c)
         & contains(i, p))
```

VIVA

End-to-End video analytic system

Q2 A car turning left with a pedestrian at an intersection.

```
c = w.object(); p = w.object(); i = w.geoConstruct('intersection')
w.filter(c.type=='car' & p.type='people' & turnLeft(c)
         & contains(i, [c, p]))
```

SkyQuery

Geospatial video query processor

Q3 A car stopped in a cycling lane.

```
c = w.camera(); b = w.geoConstruct('bike-lane')
w.filter(p.type=='car' & contains(b, c))
```

Experiment Setups

Ablation

Compare with no Optimization

Q1 A pedestrian at an intersection moving perpendicularly to the camera

```
p = w.object(); c = w.camera(); i = w.geoConstruct('intersection')
w.filter(p.type=='person' & perpendicular(p, c)
         & contains(i, p))
```

VIVA

End-to-End video analytic system

Q2 A car turning left with a pedestrian at an intersection.

```
c = w.object(); p = w.object(); i = w.geoConstruct('intersection')
w.filter(c.type=='car' & p.type='people' & turnLeft(c)
         & contains(i, [c, p]))
```

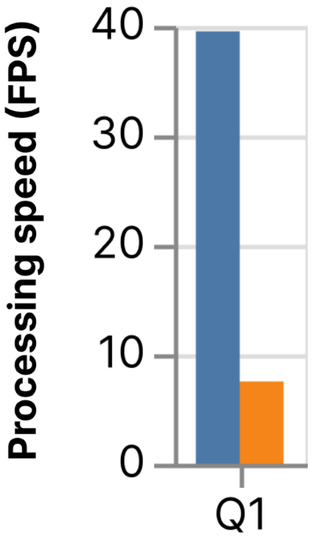
SkyQuery

Geospatial video query processor

Q3 A car stopped in a cycling lane.

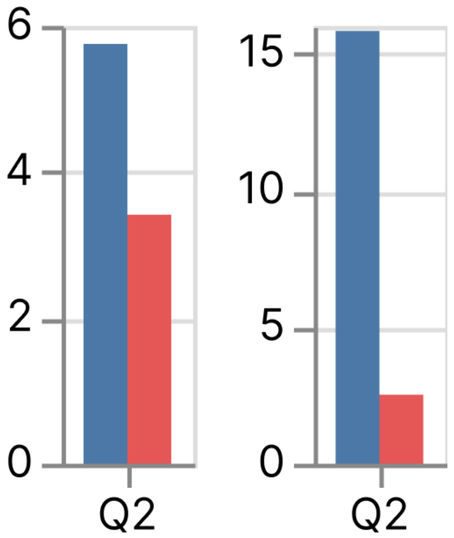
```
c = w.camera(); b = w.geoConstruct('bike-lane')
w.filter(p.type=='car' & contains(b, c))
```

Results



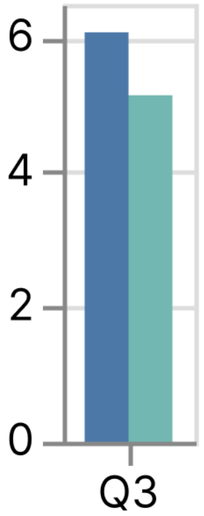
Ablation

Compare with no Optimization



VIVA

End-to-End video analytic system



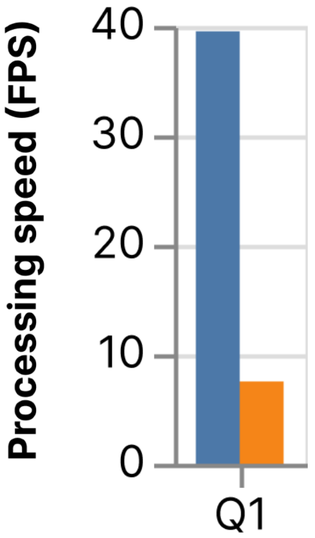
SkyQuery

Geospatial video query processor

System: Spatialyze No Optimization VIVA SkyQuery

Results

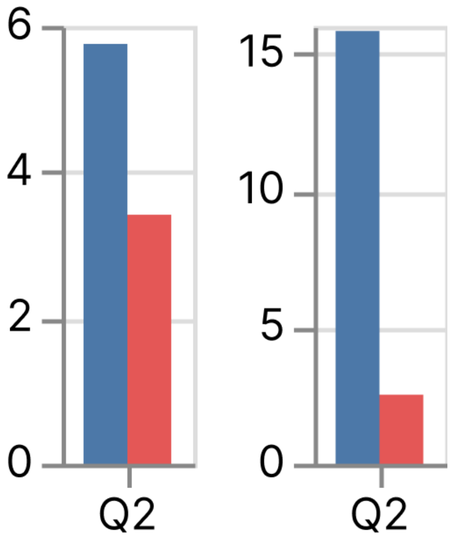
5x



Ablation

Compare with no Optimization

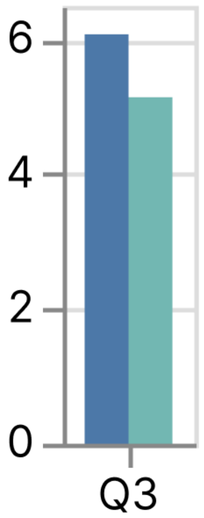
1.7-6.3x



VIVA

End-to-End video analytic system

1.17x



SkyQuery

Geospatial video query processor

System: Spatialyze · No Optimization · VIVA · SkyQuery

SPATIALYZE

In Summary

Constructing geospatial video analytics workflows is tedious and error-prone, let alone constructing them efficiently.

Spatialyze offers simplified data model and programming model; users analyze geospatial metadata together with videos.

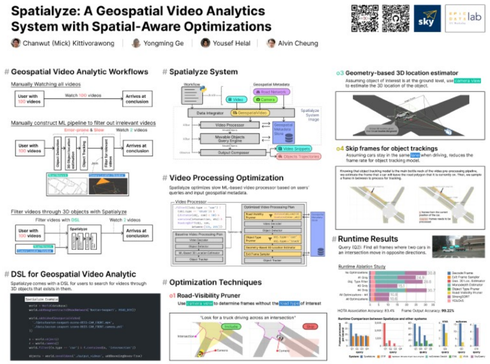
Based on users' queries, Spatialyze accelerate video processing using geospatial metadata.

SPATIALYZE

Come visit us!

Code

github.com/apperception-db/spatialyze



Poster

Tomorrow @ 10 AM

Paper

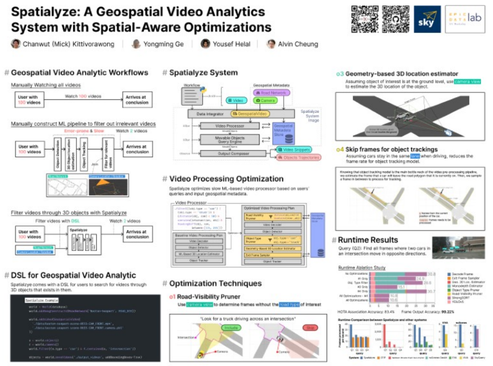
arxiv.org/abs/2308.03276

SPATIALYZE

Come visit us!

Code

github.com/apperception-db/spatialyze



Poster

Tomorrow @ 10 AM

Paper

arxiv.org/abs/2308.03276

Next Step: User-Centered Design for Video Analytics Tools

* We're starting a new research project!
* Studying real-world video analysis workflows
* Your insights will shape our design!