AutoShield: Real-Time Pedestrian-Intent Prediction with Safety-Filtered Autonomous Driving

Het Patel, Sunny Deshpande, Keisuke Ogawa, Ansh Bhansali
M.S. Autonomy and Robotics
University of Illinois Urbana-Champaign
Urbana, IL, USA

***Abstract*—Reactive pedestrian handling in autonomous driving often relies on proximity thresholds and late braking, which is brittle under uncertainty, latency, and partial observability. This report presents *AutoShield*, a modular ROS 2 autonomy stack deployed on the UIUC GEM platform that adapts vehicle behavior using pedestrian motion cues and time-to-collision (TTC) risk estimation. The stack combines LiDAR- and RGB-D-based pedestrian perception, weighted sensor fusion with time synchronization and data association, trajectory buffering and smoothing, pedestrian and ego-motion prediction, TTC estimation, and a high-level decision state machine that gates a safety controller and low-level controllers. Lateral control is implemented using a Stanley controller, while longitudinal control uses speed mapping, PID velocity control, and a hard-brake override path for emergency conditions. The resulting system cleanly separates perception, prediction, planning, and control, enabling earlier risk-aware speed adaptation compared to distance-only triggers and improving robustness to sensor disagreements and intermittent detections.**

***Index Terms*—Autonomous Driving, Pedestrian Prediction, Sensor Fusion, Time-to-Collision, Safety Controller, ROS 2, Stanley Controller, Drive-by-Wire.**

I. Introduction

Pedestrian interaction is a core failure mode for autonomy stacks that treat humans as static obstacles until they enter a collision corridor. In real deployments, pedestrian behavior is uncertain and often intention-driven: people hesitate, reverse direction, step into the roadway late, or remain near crosswalks and signs. These dynamics, combined with sensor latency and detection intermittency, make purely reactive braking policies brittle. This project addresses a practical version of this problem on a real vehicle platform: **adapt vehicle control online using behavior cues and TTC risk**, rather than distance thresholds alone.

A. Contributions

AutoShield contributes:

- A deployable, modular autonomy stack with explicit interfaces between **perception** $\rightarrow$ **fusion** $\rightarrow$ **prediction** $\rightarrow$ **decision** $\rightarrow$ **control**.
- A LiDAR pedestrian pipeline with preprocessing, DBSCAN clustering, centroid tracking with EMA smoothing, and human filtering.
- An RGB-D pipeline using object detection + depth extraction + pixel-to-ego transform to estimate pedestrian position (distance, bearing) and regulatory sign presence.
- Weighted fusion with approximate time synchronization (slop) and Euclidean data association.
- A pedestrian behavior predictor producing `/pedestrian_motion` and `/pedestrian_ttc`, enabling a TTC-driven high-level decision state machine.
- A safety-aware control layer: Stanley lateral control, PID longitudinal control, and hard-brake override when risk is critical.

II. System Overview

AutoShield is implemented as ROS 2 nodes connected by typed topics. Fig. 1 summarizes the functional pipeline.

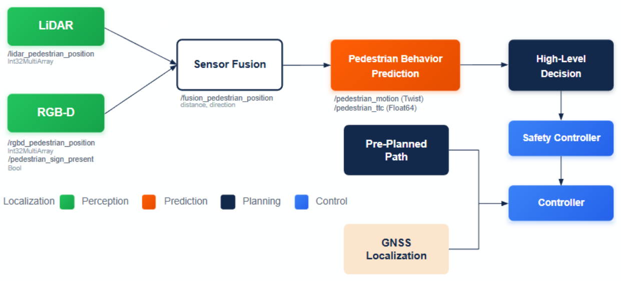

Fig. 1: AutoShield autonomy stack approach (Perception $\rightarrow$ Fusion $\rightarrow$ Prediction $\rightarrow$ High-Level Decision $\rightarrow$ Safety Controller $\rightarrow$ Controller).

A. Core ROS Topics

The stack uses simple, inspectable messages for modularity. Table I lists the primary interfaces used throughout the system.

TABLE I: Primary ROS 2 Interfaces

Topic	**Type / Payload**
`/lidar_pedestrian_position`	`Int32MultiArray` $[d,\ \theta]$
`/rgbd_pedestrian_position`	`Int32MultiArray` $[d,\ \theta]$
`/pedestrian_sign_present`	`Bool`
`/fusion_pedestrian_position`	fused $[d,\ \theta]$ (same schema)
`/pedestrian_motion`	`Twist` (predicted pedestrian motion)
`/pedestrian_ttc`	`Float64` (risk estimate)

III. Vehicle Platform and Deployment Constraints

A. *UIUC GEM Hardware Stack*

Experiments were conducted on the UIUC GEM vehicle equipped with complementary perception sensors, GNSS/INS localization, and drive-by-wire actuation:

- **Top LiDAR:** Ouster OS1-128 (Ethernet @ 1 Gbps), 128 channels, 360° horizontal FoV, 45° vertical FoV, 10–20 Hz rotation, max range ∼200 m.
- **Front LiDAR:** Livox HAP (Ethernet @ 100 Mbps), 120° × 25° FoV, ∼452k points/s.
- **Front Stereo RGB-D:** OAK-D LR (USB 3.1 @ 5 Gbps), global shutter, 1280×800 @ 23 FPS.
- **Corner Cameras:** Lucid cameras (PoE Ethernet @ 1 Gbps), 1920×1200 @ 48.3 FPS.
- **GNSS/INS:** Septentrio AsteRx SBi3 Pro+ (USB 2.0; includes RTK via internet access).
- **Radar:** Smartmicro 152 4D radar (Automotive Ethernet @ 100 Mbps), provides relative speed measurements.
- **Drive-by-wire:** PACMod2 via USB-to-CAN bridge (steering, throttle, brake commands).

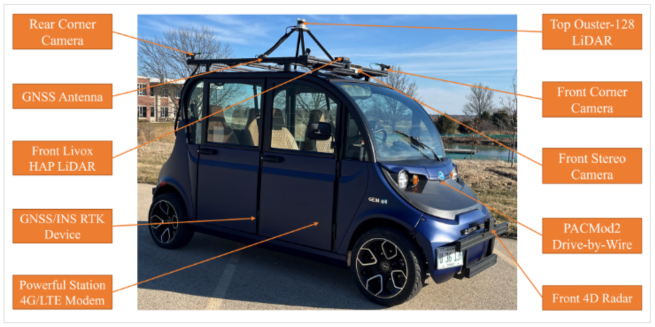

Fig. 2: GEM vehicle platform sensor and compute stack.

B. *Bring-up and Safety Interlocks*

The platform includes a Power Distribution System (PDS) controlling compute and sensor power (not the vehicle itself). A deployment constraint is that LiDAR is intentionally powered *off by default* during startup: the LiDAR cable and one GNSS antenna cable are routed together, and to avoid cross-talk the workflow waits for GNSS satellite lock before enabling LiDAR power. Additionally, physical safety interlocks (emergency button and a brake-pedal click button) *sever the computer-to-PACMod connection*. These interlocks prevent autonomous actuation commands from reaching the vehicle but do not automatically brake the vehicle; a safety driver maintains final stopping authority using the brake pedal.

IV. Methodology

This section describes each module in the autonomy stack: LiDAR perception, RGB-D perception, sensor fusion, pedestrian behavior prediction, high-level decision logic, and control.

A. *Perception: LiDAR Pedestrian Pipeline*

The LiDAR pipeline converts PointCloud2 into a stable pedestrian estimate in the ego frame. The processing stages are:

- **Preprocessing:** voxelization, ground filtering, and outlier removal.
- **Clustering:** DBSCAN to group points into object clusters and separate static/dynamic objects.
- **Tracking:** nearest-centroid matching; centroid smoothing using an exponential moving average (EMA).
- **Human Detection:** geometric filtering, motion filtering, and distance-based sorting to select the most relevant pedestrian candidate.

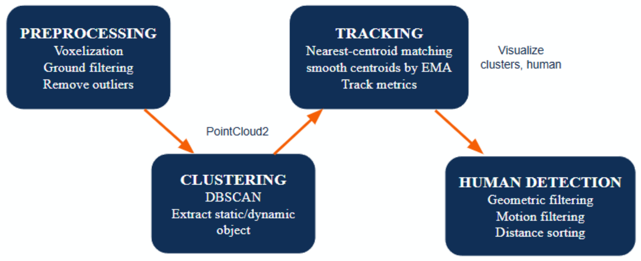

Fig. 3: LiDAR perception pipeline: preprocessing → clustering → tracking → human filtering.

Algorithm 1 LiDAR Pedestrian Detection and Tracking

```
1: Input: point cloud $P_t$ (PointCloud2)
2: $P'_t \leftarrow$ voxelize + ground-filter + remove outliers
3: $\mathcal{C}_t \leftarrow$ DBSCAN($P'_t$)
4: $\mathcal{H}_t \leftarrow$ apply human filters to candidate clusters
5: $c_t \leftarrow$ nearest-centroid match from $\mathcal{H}_t$ to previous track
6: $\bar{c}_t \leftarrow \alpha c_t + (1-\alpha)\bar{c}_{t-1}$ {EMA smoothing}
7: Convert $\bar{c}_t$ to $(d_t, \theta_t)$ in ego frame
8: Publish /lidar_pedestrian_position as
   Int32MultiArray $[d_t, \theta_t]$
```

B. *Perception: RGB-D Pedestrian Estimation*

The RGB-D module estimates pedestrian position using object detection and depth:

- **Detection:** YOLOv11 detects pedestrian bounding boxes in RGB.
- **Depth Extraction:** map the bounding box to depth to estimate the closest pedestrian depth.
- **Pose Transform:** convert pixel location + depth into an ego-frame position estimate (Euclidean distance and bearing).

The module also produces a regulatory sign detection signal used downstream by the high-level decision logic.

C. *Sensor Fusion*

Sensor fusion combines LiDAR and RGB-D pedestrian estimates into a single, robust fused estimate. Fusion includes:

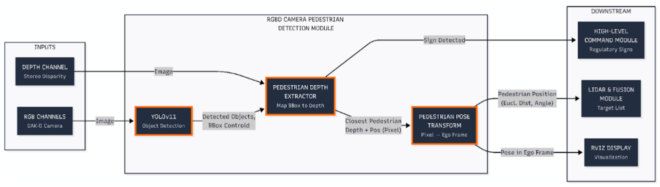

Fig. 4: RGB-D pipeline: object detection → depth extraction → pixel-to-ego transform; also provides sign presence.

1) **Approximate time sync:** slop ≈ 0.1s between LiDAR and RGB-D messages.
2) **Data association:** Euclidean match threshold ≈ 2.0m to associate estimates.
3) **Weighted fusion:** compute a weighted average for distance and bearing.

The fusion weights used in the system are summarized in Table II.

TABLE II: Fusion Weight Parameters

Quantity	LiDAR Weight	Camera Weight
Distance	0.8	0.2
Direction	0.3	0.7

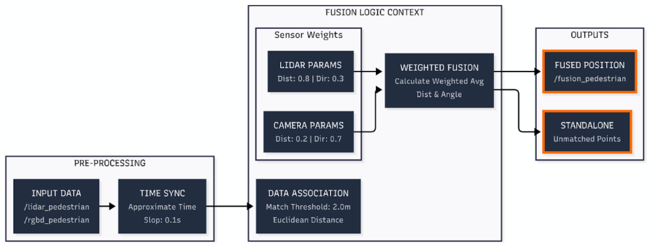

Fig. 5: Fusion logic: input streams → time sync → association → weighted fusion; outputs fused position and standalone unmatched detections.

Algorithm 2 Weighted Pedestrian Fusion with Association

```
1: Input: LiDAR estimate $(d_L, \theta_L)$, RGB-D estimate $(d_C, \theta_C)$
2: Time-align using approximate sync (slop $\delta t \leq 0.1$s)
3: if $\|p_L - p_C\| \leq 2.0$m then
4:     $d_F \leftarrow 0.8 d_L + 0.2 d_C$
5:     $\theta_F \leftarrow 0.3\theta_L + 0.7\theta_C$
6:     Publish fused $(d_F, \theta_F)$ on /fusion_pedestrian_position
7: else
8:     Publish standalone unmatched detection(s) for monitoring
9: end if
```

D. Pedestrian Behavior Prediction and TTC

The behavior predictor consumes fused pedestrian observations and outputs motion and TTC signals for downstream decision-making. The module includes:

- **Pose Transform:** represent pedestrian pose consistently in ego frame.
- **Trajectory Buffer:** maintain a bounded history of pedestrian positions.
- **Trajectory Smoothing:** smooth trajectory to mitigate jitter and intermittent detections.
- **Motion Prediction:** estimate pedestrian velocity and near-horizon trajectory.
- **Ego Motion Prediction:** use ego vehicle speed to predict future vehicle position over the horizon.
- **TTC Estimation:** forward-simulate the ego vehicle as a constant-speed point mass along $+x$, then compute the earliest predicted time when the pedestrian trajectory enters a collision radius around the ego rollout (discrete-time closest approach test).

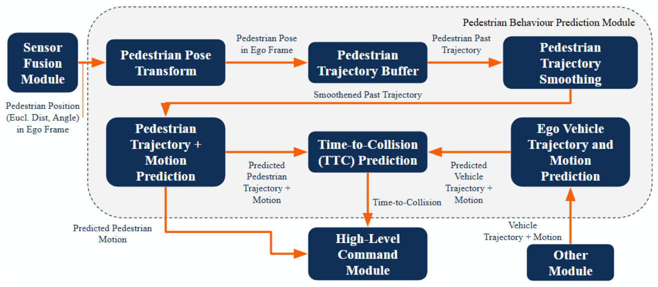
Pedestrian Behaviour Prediction Module
Sensor Fusion Module
Pedestrian Position (Eucl. Dist, Angle) in Ego Frame
Pedestrian Pose Transform
Pedestrian Pose in Ego Frame
Pedestrian Trajectory Buffer
Pedestrian Past Trajectory
Pedestrian Trajectory Smoothing
Smoothened Past Trajectory
Pedestrian Trajectory + Motion Prediction
Predicted Pedestrian Trajectory + Motion
Time-to-Collision (TTC) Prediction
Predicted Vehicle Trajectory + Motion
Ego Vehicle Trajectory and Motion Prediction
Time-to-Collision
Predicted Pedestrian Motion
High-Level Command Module
Vehicle Trajectory + Motion
Other Module

Fig. 6: Pedestrian behavior predictor module: buffering/smoothing, motion prediction, ego prediction, TTC computation, and publishing to the high-level command module.

Algorithm 3 Discrete TTC via Forward Simulation

```
1: Input: predicted pedestrian positions $\{p_i^{\text{ped}}\}_{i=1}^{N}$ in base_link, vehicle speed $v_{\text{car}}$, collision threshold $r$
2: Assumptions: ego car moves straight along $+x$ with constant speed; car lateral/vertical position is $(y, z) = (0, 0)$
3: if $N = 0$ or $v_{\text{car}} \leq 0.01$ then
4:     TTC $\leftarrow +\infty$, $d_{\min} \leftarrow +\infty$
5:     return
6: end if
7: $\Delta t \leftarrow \frac{T_{\text{pred}}}{N}$ $\{T_{\text{pred}} = \text{prediction horizon}\}$
8: TTC $\leftarrow +\infty$, $d_{\min} \leftarrow +\infty$
9: for $i = 1$ to $N$ do
10:     $t_i \leftarrow i\Delta t$
11:     $p_i^{\text{car}} \leftarrow [v_{\text{car}} t_i,\ 0,\ 0]^\top$
12:     $d_i \leftarrow \|p_i^{\text{ped}} - p_i^{\text{car}}\|_2$
13:     $d_{\min} \leftarrow \min(d_{\min}, d_i)$
14:     if $d_i < r$ and TTC $= +\infty$ then
15:         TTC $\leftarrow t_i$ {earliest threshold crossing}
16:     end if
17: end for
18: Publish /pedestrian_motion (Twist) and /pedestrian_ttc (Float64)
```

We model the ego vehicle as a point moving at constant longitudinal speed along the base_link $+x$ axis over the prediction horizon, and define TTC as the earliest discrete-time step when the Euclidean separation between the predicted pedestrian position and the ego rollout drops below a collision radius.

E. *High-Level Command: Safety State Machine*

The high-level command module is a structured state machine that gates vehicle behavior based on data validity, TTC risk, sign context, and whether the pedestrian is static or crossing. The principal states are:

- **CRUISE:** normal operation on the pre-planned path.
- **SLOW_CAUTION:** reduce speed while monitoring.
- **STOP_YIELD:** immediate halt when critical risk or sign context demands yielding.

The decision logic includes key checks consistent with the implementation diagram:

- **Data staleness:** treat data as stale if no update for > 0.5s.
- **Critical TTC:** critical region when $0 < \text{TTC} < 2.5$s.
- **Sign context:** if a regulatory sign is detected, enforce stop/yield behavior.
- **Closing vector:** determine whether the pedestrian is moving toward the vehicle / path (closing speed).
- **Static person:** classify static when speed < 0.1 m/s for context-dependent caution.
- **Recovery:** a wait buffer (2.0s) before returning to cruise once the path is clear.

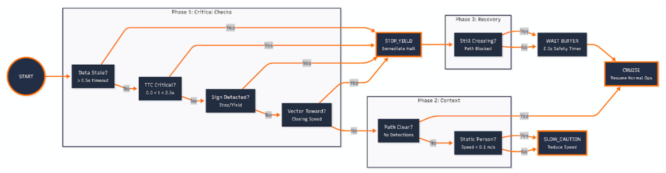

Fig. 7: High-level command state machine for CRUISE / SLOW_CAUTION / STOP_YIELD with stale-data, TTC, sign, and recovery buffering logic.

Algorithm 4 High-Level Decision Logic (Simplified)

```
1:  Inputs: TTC, sign_present, data_age, closing_speed, ped_speed, path_clear
2:  if data_age $> 0.5$s then
3:      state ← STOP_YIELD {fail-safe on stale risk data}
4:  else if $0 < \text{TTC} < 2.5$s then
5:      state ← STOP_YIELD
6:  else if sign_present then
7:      state ← STOP_YIELD
8:  else if not path_clear then
9:      if ped_speed $< 0.1$ m/s then
10:         state ← SLOW_CAUTION
11:     else
12:         state ← STOP_YIELD {dynamic crossing}
13:     end if
14: else
15:     state ← CRUISE (after 2.0s wait buffer)
16: end if
17: Output: safety_state to controller
```

F. *Control: Lateral (Steering) and Longitudinal (Speed)*

The controller consumes the safety state and sensor/estimation signals (GNSS, INS heading, vehicle speed). It is split into:

- **Lateral control:** Stanley controller minimizes heading and cross-track error to track a pre-planned path.
- **Longitudinal control:** speed mapping chooses target speed based on safety state; a PID velocity controller outputs throttle/brake; an emergency hard-brake override triggers under STOP_YIELD danger conditions.

The speed mapping used in the design is:

$$v_{\text{target}} = \begin{cases} 5.0 \text{ m/s}, & \text{CRUISE} \\ 2.5 \text{ m/s}, & \text{SLOW\_CAUTION} \\ 0, & \text{STOP\_YIELD} \end{cases}$$

The hard-brake override applies an emergency braking command (panic stop) with magnitude ≈ 0.6 when the safety state indicates danger.

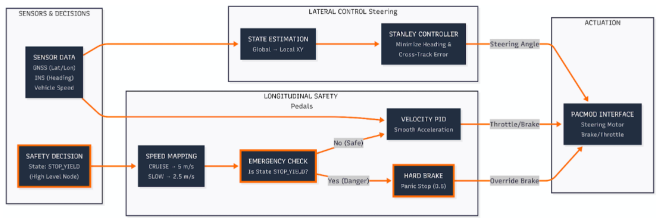

Fig. 8: Controller module: Stanley steering, velocity PID for pedals, and emergency hard-brake override, integrated via PACMod2 drive-by-wire.

V. Experiments and Evaluation Plan

This project emphasizes robust real-time integration on a real vehicle platform. Recommended metrics (and those naturally supported by the architecture) include:

- **Perception stability:** centroid jitter (LiDAR), detection intermittency (RGB-D), ID switches.
- **Fusion consistency:** association success rate; disagreement rates; fused estimate variance.
- **Prediction quality:** TTC smoothness and responsiveness; false critical TTC triggers.
- **Control response:** time to slow/stop, minimum TTC achieved, braking aggressiveness events.

A. *Operational Safety Notes*

All tests assume supervised operation with a safety driver. Physical interlocks that sever PACMod control are treated as safety boundaries, and STOP_YIELD is implemented as a control gating state rather than an assumption of automatic braking by the vehicle.

VI. Results (Qualitative)

AutoShield achieves a clean functional separation between perception, prediction, decision-making, and control, enabling interpretable debugging and safer operation. In scenarios where pedestrian detections appear intermittently or disagree across modalities, the fusion module stabilizes the estimate and the behavior predictor produces a smoother TTC signal. The high-level state machine enables conservative transitions to STOP_YIELD under critical TTC or sign contexts, while SLOW_CAUTION supports controlled speed reduction when the context is ambiguous.

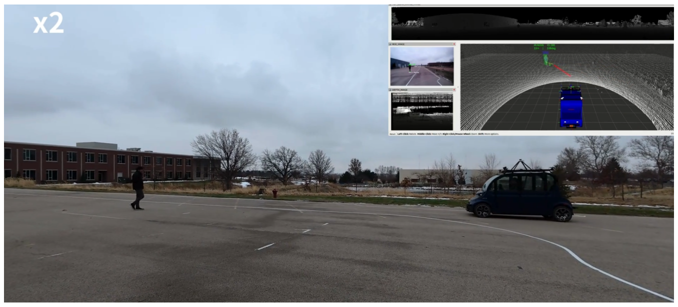

Fig. 9: Before: Pedestrian Detected

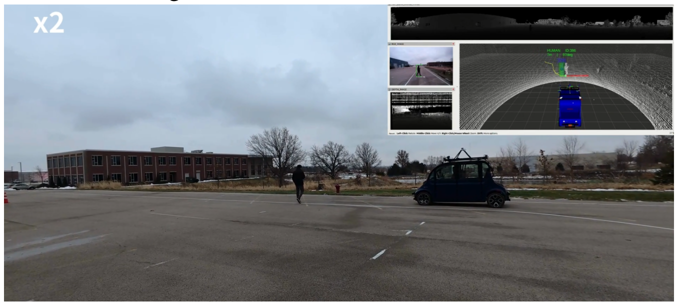

Fig. 10: After: Speed Adaptation / Stop-Yield

VII. Limitations

Key limitations observed/anticipated from the design:

- **Heuristic thresholds:** TTC critical range and state transitions require tuning across environments, pedestrian speeds, and sensor latencies.
- **Partial observability:** occlusions and limited depth returns can temporarily destabilize pedestrian estimates; the system mitigates but does not eliminate this.
- **Intent ambiguity:** a pedestrian near the path is not always a crossing intent; richer intent models could reduce false slowdowns/stops.
- **Actuation boundary:** emergency interlocks sever autonomy control but do not apply brakes; the design must assume a safety driver for hard safety guarantees.

VIII. Conclusion and Future Work

This report presented AutoShield, a real-vehicle ROS 2 autonomy stack that adapts control based on pedestrian behavior rather than distance-only triggers. The approach integrates multi-modal perception (LiDAR + RGB-D), weighted fusion with time sync and association, behavior prediction with TTC estimation, a structured high-level safety state machine, and safety-aware control using Stanley steering and PID speed control with emergency braking override.

Future work directions include:

- Replace short-horizon kinematic prediction with multi-modal learned intent/trajectory predictors.
- Extend to multi-pedestrian tracking and interaction-aware planning.
- Add formal safety filters (e.g., control barrier functions) to enforce provable constraints under modeling uncertainty.
- Improve calibration and synchronization to reduce association errors and TTC noise.

Acknowledgment

Thanks to the ECE 484 course staff and the UIUC GEM platform maintainers for providing the vehicle infrastructure, ROS drivers, and safety procedures that enabled on-platform experimentation.

Appendix

A. *Fusion Parameters*

- Approximate time sync slop: 0.1s
- Association threshold: 2.0m
- Weights: distance (LiDAR 0.8, camera 0.2), direction (LiDAR 0.3, camera 0.7)

B. *Decision Parameters*

- Data stale timeout: 0.5s
- TTC critical range: $0 <$ TTC < 2.5s
- Static person speed threshold: 0.1 m/s
- Recovery wait buffer: 2.0s

C. *Control Parameters*

- Speed mapping: CRUISE $= 5.0$ m/s, SLOW_CAUTION $= 2.5$ m/s, STOP_YIELD $= 0$ m/s
- Emergency hard brake magnitude: 0.6
- Lateral controller: Stanley controller (heading + cross-track minimization)

References

[1] Open Source Robotics Foundation, "ROS 2 Documentation," https://docs.ros.org/.

[2] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Proc. KDD*, 1996.

[3] S. Thrun *et al.*, "Stanley: The Robot that Won the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[4] J. Redmon *et al.*, "You Only Look Once: Unified, Real-Time Object Detection," in *Proc. CVPR*, 2016.

[5] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, OTexts, 2018 (EMA background).