Republic of Tunisia

*****

Ministry of Higher Education and Scientific Research

*****

University of Monastir

*****

Higher Institute of Computer Science and Mathematics of Monastir

Department of Computer Science

Order N°:

End of Studies Project Report

Presented in order to obtain the

National Bachelor's Degree in Computer Science

Specialization:

Software Engineering and Information Systems

By:

Computer Vision Models Fit for NPUs

Presented on in front of the jury composed of:

	President
	Member
Ms. Nada Haj Messaoud	Academic Supervisor
Mr Ilyes Tlili	Professional Supervisor

Academic Year: 2025 / 2026

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1 Introduction

This introductory chapter presents the scope of the end-of-studies project conducted within **in2-Technologies** in the context of the **EYES (EYE-D)** solution. It introduces the hosting organization, clarifies the project context, and formalizes the problem statement and objectives. It also provides a high-level overview of the adopted approach, with particular emphasis on producing computer vision models that are compatible with deployment constraints on NPU-based systems.

2 Host Organization Presentation

2.1 in2-Technologies

in2-Technologies is an IT company that positions itself around digital innovation by delivering solutions that combine software engineering, creative design, and advanced technologies. In particular, the organization emphasizes applied Artificial Intelligence as a practical lever to deliver real-world value (Figure 1.1(a)).

According to the public communications provided for this report, in2-Technologies is actively involved in the innovation ecosystem through participation in events and technology programs. These activities help connect the company with partners and stakeholders, and they support continuous improvement by confronting products with real deployment needs.

In this context, in2-Technologies develops EYES, an AI-driven solution focused on video analytics and on-device intelligence.

2.2 EYES (EYE-D)

EYES is an AI-driven solution developed within in2-Technologies and centered on computer vision and video analytics. It is presented as an approach that transforms existing video surveillance infrastructures into a system capable of producing actionable insights (Figure 1.1(b)).

Based on the provided communication materials, EYES is designed to integrate with existing camera networks and perform on-site processing. This positioning aims to reduce dependence on cloud processing, improve responsiveness, and address operational constraints such as privacy considerations and deployment cost.

EYES is also described as modular and expandable, enabling the activation or integration of AI modules according to the targeted needs. The communication materials highlight use cases related to security monitoring and industrial safety, with real-time alerting and reporting.

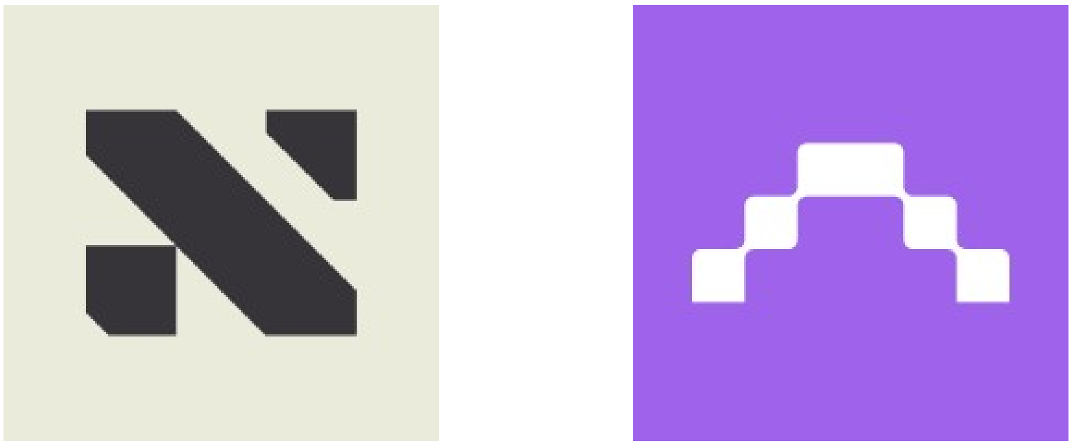
From an operational perspective, the provided materials emphasize the following aspects:

- **Integration**: EYES connects to an existing camera network and can be installed with minimal disruption.
- **On-device processing**: video analytics are executed locally to provide real-time insights and to reduce reliance on external connectivity.
- **Expandable modules**: the solution supports an evolving library of AI modules that can be enabled according to the deployment context.
- **Monitoring and reporting**: the solution provides alerts and reporting capabilities to support operational decision-making.

The brochure-style description provided for EYES points to a set of security and industrial safety features, including (non-exhaustively) access control and intrusion detection, people and vehicle detection, fire and emergency response related detection, and safety compliance monitoring. These capabilities motivate the need for robust, efficient, and deployable computer vision models under edge constraints, which is the focus of this project.

According to the public information provided for this report, EYES has been highlighted through multiple milestones and participations, including:

- Selection as a winner in an AI-focused innovation program (AI Garage Cohort 3 at Novation City).
- First place in the Tunisia IoT & AI Challenge 2025, followed by participation in the Arab IoT & AI Challenge in Dubai.
- Presence in technology events such as Big Tech Africa, STEP Dubai 2025, Security Expo London 2025, and GITEX Expand North Star.



(a) in2-Technologies (b) EYES (EYE-D)

Figure 1.1: in2-Technologies and EYES logos

3 Project Presentation

3.1 Project Context

The project addresses the engineering challenge of developing and deploying computer vision models on resource-constrained platforms equipped with specialized accelerators. The target device and its detailed specifications are **confidential under a signed NDA**. In this report, the target platform is described in a general manner as an embedded system equipped with an **NPU (Neural Processing Unit)**.

Within the EYES context, the project contributes to an applied video analytics solution where on-device inference is required for responsiveness, operational constraints, and deployment practicality. The scope of the work includes a set of computer vision tasks required by the solution, covering multiple model families. The overall goal is not limited to training models, but extends to building complete and reliable pipelines that support deployment and monitoring in realistic operational conditions.

3.2 Problem Statement

Although computer vision models can achieve strong accuracy in research settings, deploying them on NPU-based platforms introduces several constraints and practical difficulties. In particular, the project must address the following issues:

- **Deployment constraints**: limited memory and compute budgets, strict latency requirements, and limited operator support depending on the target accelerator.
- **Model portability**: ensuring that trained models can be exported to an interoperable format (e.g., ONNX) and then converted into optimized inference artifacts.
- **Performance trade-offs**: balancing accuracy, inference speed, and power consumption when selecting architectures and optimization strategies.
- **Pipeline robustness**: handling runtime failures and edge cases through systematic error handling and stable orchestration across multiple vision tasks.

3.3 Requirements and Constraints

In line with the project context and problem statement, the work is driven by a set of requirements and constraints that guide design decisions and validation activities:

- **On-device inference**: the deployed solution must support local execution on the target embedded platform equipped with an NPU.
- **Latency and resource limits**: inference must remain compatible with strict latency targets and limited compute and memory budgets.
- **Deployment compatibility**: trained models must be exportable to an interoperable format and convertible into optimized inference artifacts.
- **Reliability in operation**: inference pipelines must handle runtime failures and edge cases through systematic validation and error handling.
- **Confidentiality constraints**: hardware specifications and internal data collection tooling remain confidential under NDA, requiring careful abstraction in documentation.

3.4 Project Purpose

The main objective of this project is to design an end-to-end workflow that produces **computer vision models fit for NPU deployment** while maintaining reliable inference pipelines for EYES.

The following guiding questions and objectives summarize the expected contribution of the project.

Guiding Question	**Project Objective**
How can we deploy computer vision models under strict edge constraints?	Build an optimization and deployment pipeline compatible with NPU execution.
How can we maintain acceptable accuracy without exceeding latency and power budgets?	Select and tune suitable architectures, then benchmark trade-offs on representative workloads.
How can we ensure stable integration in a multi-task video analytics product?	Integrate inference into robust pipelines with monitoring, validation, and failure handling.

Table 1.1: Guiding questions and corresponding project objectives

3.5 EYES Web Application

In addition to the on-device inference components, EYES is accompanied by a web-based interface that supports operational use. This interface is used to visualize analytics outputs, consult alerts and reports, and support supervision activities around deployed AI modules. In this report, the web application is considered as the user-facing component that consumes and presents the outputs produced by the embedded inference pipelines.

The 5 W's

In order to clarify the scope of the project and align expectations, the project can be summarized using the **5 W's** framework.

Aspect	**Description**
Who?	• The project stakeholders include in2-Technologies, the EYES product team, and the operational users of video analytics solutions.
What?	• The project involves the development and integration of an end-to-end workflow for computer vision models within EYES. • The workflow covers training, export to ONNX, optimization, and integration for edge inference.
When?	• EYES is under continuous development and is updated in response to evolving market needs.
Where?	• Within the in2-Technologies environment, targeting an embedded deployment platform equipped with an NPU (details confidential under NDA).
Why?	• Reduce reliance on cloud processing by enabling reliable on-device analytics. • Improve responsiveness by meeting latency, compute, and power constraints on the target platform.

Table 1.2: The Project's 5 W's

4 Project Steps

This project follows an iterative and incremental approach to deliver deployable computer vision components for EYES.

At a high level, the work is organized into the following steps:

- Data collection and preparation using an internal data collector (confidential under NDA).
- Model selection and training using **PyTorch**, focusing on architectures suitable for edge deployment.
- Model export to **ONNX** and validation to ensure correctness after conversion.
- Optimization into deployment-ready inference artifacts suitable for the target platform.
- Integration into multi-task pipelines with systematic validation and failure handling.
- Benchmarking of runtime indicators (latency, memory, and power) under representative inference workloads.

4.1 Existing Solutions

Several solution strategies are commonly adopted to deliver computer vision capabilities under operational constraints:

- **Cloud-centric processing**: cameras stream data to a server or cloud environment that runs inference, then returns results to client applications.
- **Edge computing with accelerators**: inference is executed close to the cameras on embedded devices equipped with GPUs or NPUs, reducing latency and dependency on connectivity.
- **Standardized model exchange and deployment tooling**: workflows often rely on interoperable formats (e.g., ONNX) and on optimization toolchains (quantization, pruning, operator fusion) to obtain deployable artifacts.
- **Modular analytics pipelines**: products typically combine multiple specialized models (detection, tracking, classification) orchestrated to produce higher-level events.

4.2 Critique of Existing Solutions

Although these approaches are effective in many contexts, they present limitations that motivate the focus of this project:

- **Latency and bandwidth trade-offs**: cloud-centric approaches increase dependency on network connectivity and may not satisfy strict real-time constraints.
- **Conversion and portability issues**: model export and optimization may fail due to unsupported operators, numerical differences, or vendor-specific constraints.
- **Accuracy degradation**: deployment-oriented optimizations such as quantization may reduce accuracy if not carefully validated and calibrated.
- **Operational robustness**: multi-model pipelines can be sensitive to runtime failures and edge cases, requiring systematic monitoring and error handling.
- **Vendor lock-in risks**: NPU deployment toolchains may impose specific constraints that limit portability across platforms.

5 Adopted Project Management Methodology

The project is conducted using an iterative approach that allows incremental delivery and continuous validation. This approach is suitable for AI engineering workflows where model performance, deployment compatibility, and resource constraints must be checked throughout the development cycle.

For this project, we adopt the **Scrum** methodology as an Agile project management framework. The work is organized into time-boxed sprints, enabling regular planning, implementation, and review cycles. This organization supports continuous refinement of both model quality and deployment readiness.

Scrum Element	**Project Adoption**
Product Backlog	Maintain a prioritized list of tasks covering training, optimization, and integration activities.
Sprint Planning	Select sprint goals and define deliverables that can be validated on the target platform.
Sprint	Implement and integrate incremental improvements, with frequent checks on performance constraints.
Sprint Review	Demonstrate the sprint increment (models, pipelines, or benchmarks) and collect feedback.
Sprint Retrospective	Identify process improvements and adapt the next sprint organization accordingly.

Table 1.3: Scrum methodology and its adoption in the project

For project tracking and collaboration, **GitHub** is used as the central platform. It supports version control, task tracking through issues, and coordination of development activities throughout the project lifecycle.

6 Conclusion

This chapter presented the hosting organization (in2-Technologies) and the context of the EYES solution, then formalized the project overview, requirements, objectives, and the adopted project management approach. The next chapter will introduce the technical background required for this work, including computer vision modeling, ONNX export, and deployment-oriented optimization for NPU-based inference.

Chapter 2

Release 1: Speed and Angle Estimation

1 Introduction

This chapter presents the technical background and implementation work for speed and angle estimation within the EYES video analytics solution. The work described here corresponds to Sprint 1 of the project, which focused on developing the foundational components for measuring object speed and heading direction from video streams.

Accurate estimation of object speed and heading direction is essential for a variety of surveillance and safety applications, including traffic monitoring, anomaly detection, and predictive analysis. The speed and angle estimation module operates as part of the broader EYES pipeline, receiving tracking information from upstream detection and tracking modules.

Sprint 1 aimed to establish the core infrastructure for speed and angle estimation, including the state of the art study, mathematical foundations, pipeline architecture, and initial implementation with configuration parameters.

2 Sprint 1 Development

2.1 Sprint Backlog For Sprint 1

The Sprint 1 backlog focused on building the foundational components for speed and angle estimation. The main tasks included:

- Research and document state-of-the-art methods for speed estimation
- Research and document state-of-the-art methods for angle and heading estimation
- Design and implement the speed estimation pipeline architecture
- Develop mathematical foundations for speed, curvature, and angle calculations
- Implement EMA smoothing for temporal filtering
- Address implementation challenges including sparse trajectories and calibration dependency
- Define and document configuration parameters for pipeline tuning

2.2 State of the Art Study

Speed Estimation Methods

Pixel-Based Methods Pixel-based speed estimation operates directly in the image coordinate space without explicit geometric transformation. Given the centroid position (x_t, y_t) at frame t, the pixel displacement is computed as:

$$\Delta p = \sqrt{(x_{t+1} - x_t)^2 + (y_{t+1} - y_t)^2} \tag{2.1}$$

The speed in pixels per frame is then:

$$v_{pixel} = \frac{\Delta p}{\Delta t} \tag{2.2}$$

where $\Delta t = 1/fps$ is the time between consecutive frames. While computationally simple, pixel-based methods suffer from perspective distortion, as the relationship between pixel displacement and real-world distance varies with the object's position in the image. Objects farther from the camera appear smaller and move fewer pixels per unit of real-world distance compared to objects closer to the camera.

Homography-Based Methods Homography-based approaches address the perspective distortion problem by transforming image coordinates to a top-down view of the scene. A homography matrix $\mathbf{H}$ maps points from the image plane to a world coordinate system:

$$x_w y_w w = \mathbf{H} x_i y_i 1 \tag{2.3}$$

where (x_i, y_i) are image coordinates and the world coordinates are obtained as $(X, Y) = (x_w/w, y_w/w)$. The full homography matrix is:

$$\mathbf{H} = h_{11} h_{12} h_{13} h_{21} h_{22} h_{23} h_{31} h_{32} h_{33} \tag{2.4}$$

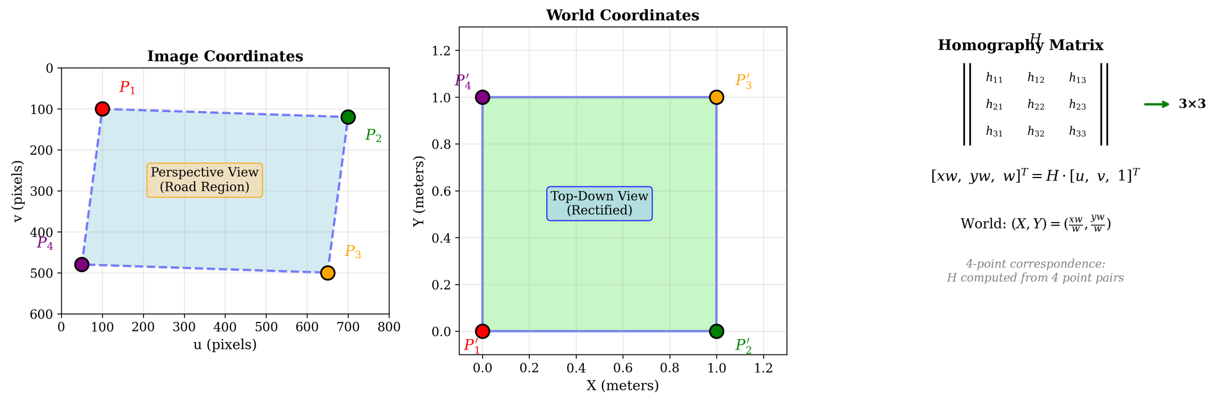


Figure 2.1: Homography transformation from image coordinates to world coordinates. Four point correspondences are used to compute the 3×3 homography matrix.

The homography matrix is typically estimated from at least 4 point correspondences using the Direct Linear Transform (DLT) algorithm with RANSAC for robustness against outliers. Once calibrated, the homography enables accurate distance and speed measurements in real-world units.

The primary advantage of homography-based methods is their ability to provide physically meaningful measurements. However, they require accurate calibration and assume a planar ground surface, which may not hold in all deployment scenarios.

Optical Flow Methods Optical flow estimates the apparent motion of pixels between consecutive frames by analyzing intensity patterns. The optical flow field $\mathbf{u}(x, y)$ at each pixel provides a dense representation of motion:

$$I(x, y, t) = I(x + u_x, y + u_y, t + \Delta t) \tag{2.5}$$

where I is the image intensity and (u_x, u_y) is the flow vector. Methods such as the Lucas-Kanade algorithm or Horn-Schunck approach compute flow by assuming brightness constancy and spatial coherence:

$$I_x u + I_y v + I_t = 0 \tag{2.6}$$

where I_x, I_y are spatial image gradients and I_t is the temporal gradient. For speed estimation, optical flow can be aggregated over object regions to estimate the average motion magnitude.

While optical flow provides rich motion information, it is sensitive to illumination changes, occlusions, and aperture problems. Additionally, converting optical flow to physical speed still requires knowledge of the camera geometry and scene scale.

Deep Learning-Based Methods Recent advances in deep learning have enabled end-to-end speed estimation from video frames. Convolutional neural networks (CNNs) and recurrent architectures can learn to predict speed directly from spatiotemporal features without explicit geometric modeling. 3D CNN architectures capture spatio-temporal features from sequences of consecutive frames, while hybrid approaches combine optical flow computation with CNN processing for speed regression.

Deep learning methods offer the potential to handle complex scenarios and implicit scene understanding. However, they introduce additional computational overhead and require substantial training data, which may not be practical for all deployment contexts. Furthermore, the black-box nature of neural networks can complicate error analysis and calibration.

Angle and Heading Estimation Methods

Trajectory-Based Methods Trajectory-based heading estimation computes the direction of motion from the object's trajectory over time. Given a sequence of positions $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, the heading angle θ can be estimated using the four-quadrant inverse tangent:

$$\theta = \text{arctan}\,2(\Delta y, \Delta x) \tag{2.7}$$

where Δx and Δy represent the displacement between trajectory points. In world coordinates (after homography transformation), this becomes:

$$\theta_t = \text{arctan}\,2(y_{w,t+1} - y_{w,t}, x_{w,t+1} - x_{w,t}) \tag{2.8}$$

The choice of points for computing displacement affects the estimation quality: using immediate neighbors provides responsiveness but is sensitive to noise, while using points further apart improves stability but reduces responsiveness to rapid directional changes.

Curvature-Based Methods For objects following curved paths, the instantaneous heading can be derived from the local curvature of the trajectory. The curvature κ at a point is defined as:

$$\kappa = \frac{|\mathbf{v} \times \mathbf{a}|}{|\mathbf{v}|^3} \tag{2.9}$$

where $\mathbf{v}$ is the velocity vector and $\mathbf{a}$ is the acceleration vector. In component form:

$$\kappa = \frac{|x'y'' - y'x''|}{(x'^2 + y'^2)^{3/2}} \tag{2.10}$$

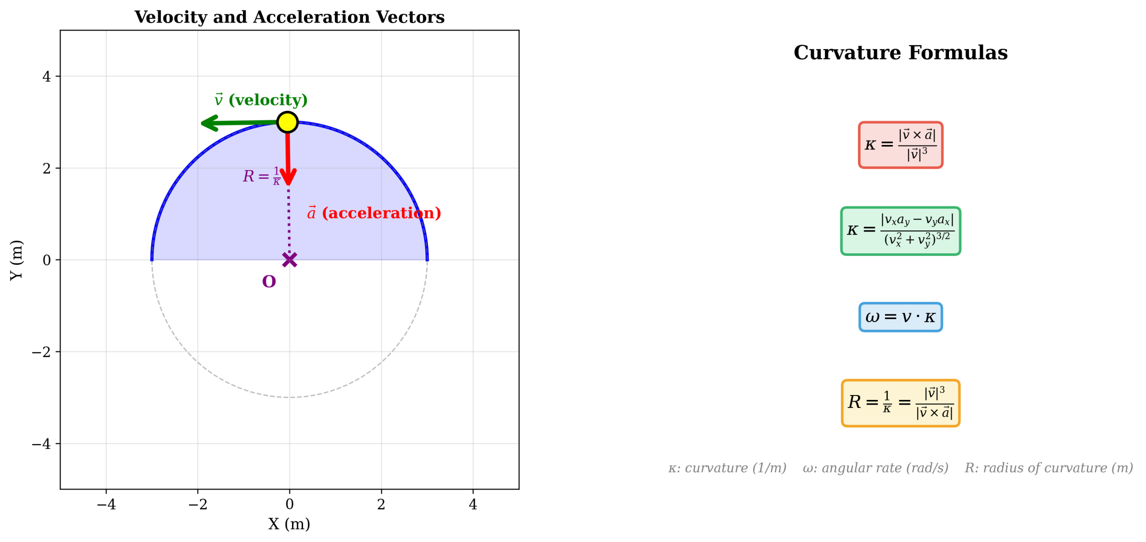

Velocity and Acceleration Vectors
$\vec{v}$ (velocity)
$R = \frac{1}{\kappa}$
$\vec{a}$ (acceleration)
O
Y (m)
X (m)
Curvature Formulas
$\kappa = \frac{|\vec{v} \times \vec{a}|}{|\vec{v}|^3}$
$\kappa = \frac{|v_x a_y - v_y a_x|}{(v_x^2 + v_y^2)^{3/2}}$
$\omega = v \cdot \kappa$
$R = \frac{1}{\kappa} = \frac{|\vec{v}|^3}{|\vec{v} \times \vec{a}|}$
κ: curvature (1/m) ω: angular rate (rad/s) R: radius of curvature (m)

Figure 2.2: Curvature calculation from trajectory points showing velocity and acceleration vectors.

The angular rate of change ω is then computed from the curvature and speed:

$$\omega = v \cdot \kappa \tag{2.11}$$

Curvature-based methods provide smooth heading estimates that naturally account for turning behavior. However, they require higher-order derivatives of position, which can amplify noise in the trajectory data.

Angular Rate Integration An alternative approach estimates the angular rate directly from trajectory changes. The angular rate can be computed from consecutive velocity vectors:

$$\omega = \frac{d\theta}{dt} = \frac{v_x a_y - v_y a_x}{v_x^2 + v_y^2} \tag{2.12}$$

where (v_x, v_y) and (a_x, a_y) are the velocity and acceleration components. This formulation provides a direct estimate of the turning rate without explicit curvature computation.

2.3 Design Diagrams For Sprint 1

Pipeline Architecture

The speed estimation pipeline was developed as part of Sprint 1 to provide real-time speed and angle measurements for detected objects. The architecture follows a modular design with well-defined interfaces between components.

Pipeline Modules The implementation consists of the following core modules:

Module	Lines	Responsibility
`math.py`	1045	Speed estimation, turning metrics, trajectory analysis
`pipeline.py`	2251	End-to-end processing, RTSP handling, event clipping
`calibration.py`	177	Homography-based coordinate transformation
`detection.py`	128	YOLOv8 object detection wrapper

Table 2.1: Core modules of the speed estimation pipeline

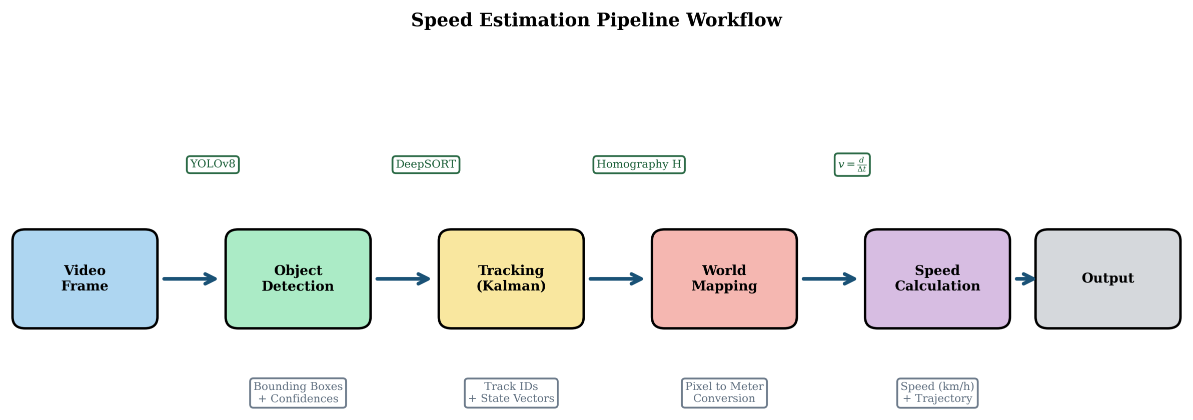

Speed Estimation Pipeline Workflow
YOLOv8
DeepSORT
Homography H
$v = \frac{d}{\Delta t}$
Video Frame
Object Detection
Tracking (Kalman)
World Mapping
Speed Calculation
Output
Bounding Boxes + Confidences
Track IDs + State Vectors
Pixel to Meter Conversion
Speed (km/h) + Trajectory

Figure 2.3: Speed estimation pipeline workflow showing data flow from video frames to speed output.

Data Flow The processing pipeline follows these stages:

1. **Detection**: YOLOv8 processes each video frame to produce bounding boxes with class labels and confidence scores.
2. **Tracking**: An IoU-based tracker assigns persistent IDs to detections across frames.
3. **World Mapping**: The bottom-center point of each bounding box is transformed from pixel to world coordinates using the homography matrix.
4. **Trajectory Accumulation**: World coordinates are accumulated as trajectory dots with timestamps.

5. **Speed Calculation**: Speed is computed from displacement between trajectory dots.
6. **Turning Metrics**: Curvature and angular rate are computed from trajectory derivatives.
7. **Smoothing Filter**: EMA smoothing is applied to produce stable estimates.
8. **Constraint Validation**: Physical constraints are enforced and implausible estimates are flagged.

Trajectory Dot System A key feature of the implemented pipeline is the trajectory dot system, which provides visual feedback for debugging and monitoring. The system maintains a buffer of recent trajectory points for each tracked object and renders them as a trail of dots on the display output.

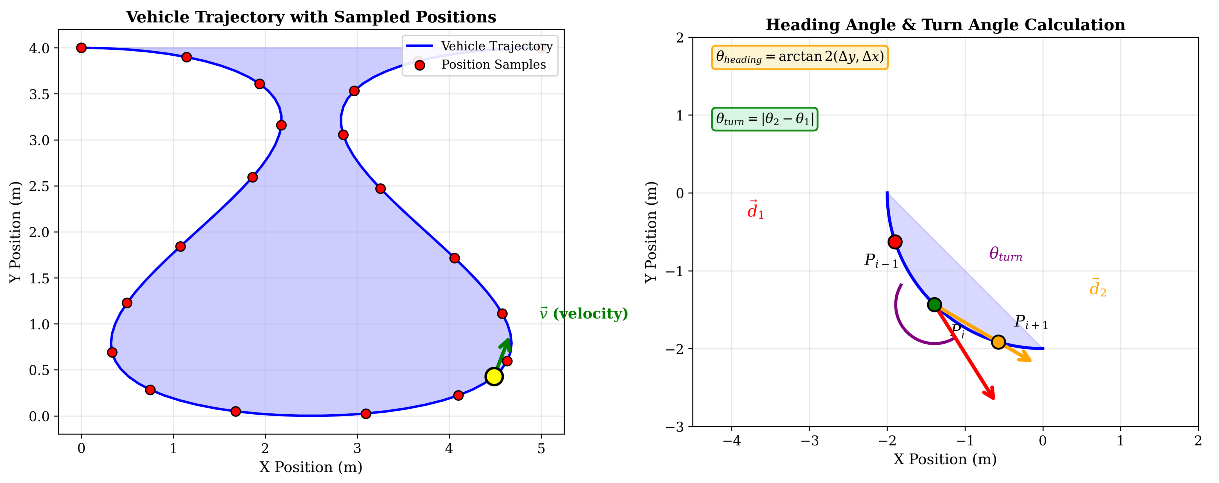


Figure 2.4: Trajectory analysis with dot visualization system showing motion history.

The system uses a trajectory dot approach where dots are added to each track's history when specific conditions are met:

$$Adddotif : |\Delta \mathbf{p}| \geq d_{min} and \Delta t \geq t_{min} \tag{2.13}$$

Default parameters are $d_{min} = 0.20$ meters and $t_{min} = 0.2$ seconds. This approach provides temporal smoothing by requiring minimum displacement before recording new trajectory points.

The trajectory dot system serves multiple purposes:

- Visual verification of tracking continuity
- Assessment of trajectory smoothness and noise characteristics
- Debugging support for speed and angle estimation issues
- Presentation of motion history to operators

Mathematical Foundations

Speed Calculation The fundamental speed calculation converts displacement between trajectory points into physical speed. Given two consecutive trajectory points in world coordinates (x_1, y_1) and (x_2, y_2) with timestamps t_1 and t_2, the speed v is computed as:

$$v = \frac{\Delta d}{\Delta t} = \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{t_2 - t_1} \tag{2.14}$$

The displacement Δd is computed in world units (typically meters), yielding speed in physical units (meters per second). The homography transformation ensures that perspective effects are properly accounted for in the distance computation.

Curvature Computation For heading estimation, the pipeline computes the local curvature of the trajectory. Given three consecutive points $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$, the velocity and acceleration vectors at $\mathbf{p}_2$ are approximated as:

$$\mathbf{v} = \frac{\mathbf{p}_3 - \mathbf{p}_1}{2\Delta t} \tag{2.15}$$

$$\mathbf{a} = \frac{\mathbf{p}_3 - 2\mathbf{p}_2 + \mathbf{p}_1}{\Delta t^2} \tag{2.16}$$

The curvature κ is then:

$$\kappa = \frac{|\mathbf{v} \times \mathbf{a}|}{|\mathbf{v}|^3} \tag{2.17}$$

The angular rate ω represents the rate of change of the heading angle:

$$\omega = v \cdot \kappa = \frac{|\mathbf{v} \times \mathbf{a}|}{|\mathbf{v}|^2} \tag{2.18}$$

EMA Smoothing Exponential Moving Average (EMA) smoothing is applied to reduce noise in speed and angle estimates. The EMA filter provides a simple yet effective temporal smoothing mechanism:

$$S_t = \alpha \cdot X_t + (1 - \alpha) \cdot S_{t-1} \tag{2.19}$$

where S_t is the smoothed value at time t, X_t is the raw input value, and $\alpha \in (0, 1)$ is the smoothing factor. Lower values of α provide stronger smoothing but introduce more lag, while higher values preserve responsiveness but allow more noise to pass through.

For angle estimation, special handling is required due to the circular nature of angles. The EMA filter is applied to the sine and cosine components separately:

$$\sin(S_t) = \alpha \cdot \sin(X_t) + (1 - \alpha) \cdot \sin(S_{t-1}) \tag{2.20}$$

$$\cos(S_t) = \alpha \cdot \cos(X_t) + (1 - \alpha) \cdot \cos(S_{t-1}) \tag{2.21}$$

The smoothed angle is then computed as:

$$S_t = \arctan 2(\sin(S_t), \cos(S_t)) \tag{2.22}$$

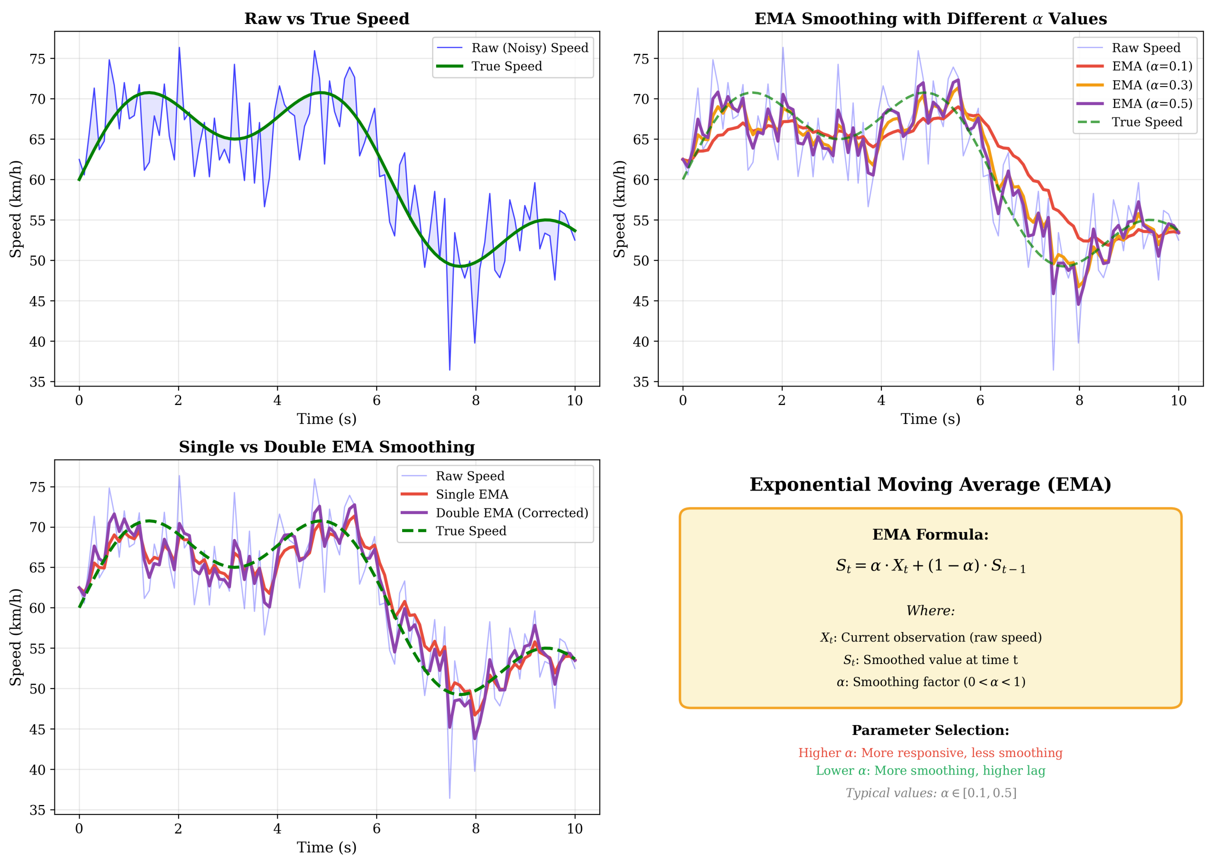


Figure 2.5: Effect of EMA smoothing on speed estimates comparing raw and filtered values.

Physical Constraints Physical constraints are enforced to reject implausible estimates and maintain consistency with real-world behavior:

- **Maximum speed**: Estimates exceeding a configurable maximum speed threshold are flagged as invalid.
- **Minimum displacement**: Very small displacements are treated as zero velocity to avoid noise amplification.
- **Acceleration limits**: Changes in speed exceeding physical acceleration limits trigger smoothing or invalidation.
- **Angular rate limits**: Heading changes are bounded by plausible turning rates for the object class.

2.4 Implementation Of Sprint 1

Implementation Challenges and Issues

During Sprint 1, several implementation challenges were encountered and addressed. This section documents the key issues and their mitigations.

Sparse Trajectory Dots In scenarios with low frame rates or fast-moving objects, trajectory points may be sparse, leading to inaccurate speed and angle estimates. Sparse dots result in larger time intervals Δt, which amplifies the effect of positional noise.

Mitigation: The pipeline implements adaptive interpolation for sparse trajectories and applies stronger smoothing when the inter-frame interval exceeds a threshold. Additionally, a minimum number of trajectory points is required before producing estimates.

Zero Velocity False Positives Stationary objects or objects with minimal motion may produce spurious velocity estimates due to detection jitter. Small positional variations from the detection module can be misinterpreted as motion, leading to non-zero speed estimates for stationary objects.

Mitigation: A minimum displacement threshold is applied. Displacements below this threshold are classified as zero velocity. The threshold is configurable and can be tuned based on detection noise characteristics.

Heading Uncertainty at Low Speeds At low speeds, the trajectory direction becomes increasingly unreliable due to the dominance of positional noise over actual motion. This leads to erratic heading estimates for slow-moving or nearly stationary objects.

Mitigation: The heading estimation module implements a confidence measure based on speed magnitude. Below a minimum speed threshold, heading estimates are either withheld or marked as low confidence. The smoothed heading is maintained from the last reliable estimate.

Calibration Dependency The homography-based approach depends critically on accurate camera calibration. Errors in the homography matrix propagate to speed and angle estimates, potentially introducing systematic biases. Furthermore, calibration must be performed for each camera viewpoint, adding to deployment overhead.

Mitigation: The pipeline includes calibration validation checks that detect gross calibration errors by verifying expected geometric properties. Documentation for the calibration process was developed to ensure consistent setup across deployments.

Configuration Parameters

The speed and angle estimation pipeline exposes several configuration parameters that control its behavior. Table 2.2 summarizes the key parameters and their default values.

Parameter	Default	Description
`smoothing_alpha`	0.3	EMA smoothing factor for speed estimates. Lower values provide stronger smoothing.
`angle_smoothing_alpha`	0.2	EMA smoothing factor for angle estimates. Applied to sine and cosine components separately.
`max_speed_mps`	50.0	Maximum valid speed in meters per second. Estimates exceeding this threshold are flagged.
`min_displacement_m`	0.1	Minimum displacement in meters for non-zero velocity detection.
`min_speed_for_heading`	0.5	Minimum speed in m/s required for reliable heading estimation.
`trajectory_buffer_size`	30	Number of trajectory points retained for visualization and smoothing.
`max_acceleration`	10.0	Maximum acceleration in m/s^2 for physical constraint validation.
`max_angular_rate`	2.0	Maximum angular rate in rad/s for heading change validation.
`min_trajectory_points`	3	Minimum trajectory points required before producing estimates.

Table 2.2: Configuration parameters for speed and angle estimation

3 Conclusion

This chapter presented the work accomplished during Sprint 1, focusing on speed and angle estimation for tracked objects within the EYES video analytics framework. The implemented pipeline combines homography-based coordinate transformation, trajectory-based speed computation, curvature-derived heading estimation, and EMA smoothing to produce robust and physically meaningful estimates.

The state of the art review highlighted the trade-offs between different approaches, with homography-based methods offering the best balance of accuracy and practicality for fixed-camera deployments. The mathematical foundations established the core equations governing speed calculation, curvature computation, and temporal smoothing.

Several implementation challenges were encountered, including sparse trajectory dots, zero velocity false positives, heading uncertainty at low speeds, and calibration dependency. Mitigations were implemented and documented to address these issues.

The configuration parameters table provides a reference for tuning the pipeline to specific deployment requirements. The next chapter will extend this work to address additional estimation tasks and pipeline enhancements developed in subsequent sprints.