The Streetscape Application Services Stack (SASS): Towards a Distributed Sensing Architecture for Urban Applications

Navid Salami Pargoo[1,*], Mahshid Ghasemi[2], Shuren Xia[1], Mehmet Kerem Turkcan[2], Taqiya Ehsan[1], Chengbo Zang[2], Yuan Sun[1], Javad Ghaderi[2], Gil Zussman[2], Zoran Kostic[2], Jorge Ortiz[1,*]

[1]WINLAB, Rutgers University, New Jersey, USA
[2]Columbia University, New York, USA
Email:{navid.salamipargoo,shuren.xia,taqiya.ehsan,ys820,jorge.ortiz}@rutgers.edu;
{mahshid.ghasemi,mkt2126,cz2678,jg3465,gil.zussman,zk2172}@columbia.edu

Abstract

As urban populations grow, cities are becoming more complex, driving the deployment of interconnected sensing systems to realize the vision of smart cities. These systems aim to improve safety, mobility, and quality of life through applications that integrate diverse sensors with real-time decision-making. Streetscape applications—focusing on challenges like pedestrian safety and adaptive traffic management—depend on managing distributed, heterogeneous sensor data, aligning information across time and space, and enabling real-time processing. These tasks are inherently complex and often difficult to scale. The **Streetscape Application Services Stack (SASS)** addresses these challenges with three core services: *multimodal data synchronization*, *spatiotemporal data fusion*, and *distributed edge computing*. By structuring these capabilities as clear, composable abstractions with clear semantics, SASS allows developers to scale streetscape applications efficiently while minimizing the complexity of multimodal integration.

We evaluated SASS in two real-world testbed environments: a controlled parking lot and an urban intersection in a major U.S. city. These testbeds allowed us to test SASS under diverse conditions, demonstrating its practical applicability. The Multimodal Data Synchronization service reduced temporal misalignment errors by *88%*, achieving synchronization accuracy within *50 milliseconds*. Spatiotemporal Data Fusion service improved detection accuracy for pedestrians and vehicles by over *10%*, leveraging multicamera integration. The Distributed Edge Computing service increased system throughput by *more than an order of magnitude*. Together, these results show how SASS provides the abstractions and performance needed to support real-time, scalable urban applications, bridging the gap between sensing infrastructure and actionable streetscape intelligence.

1 INTRODUCTION

With rapid urbanization, cities are evolving into complex, sensor-rich environments, equipped with a vast array of sensors, actuators, and communication infrastructure [5]. These urban settings are essentially large-scale distributed systems, generating massive data streams that present new opportunities for streetscape applications focused on enhancing safety, accessibility, and overall urban livability [44]. Streetscape applications address high-impact, real-time challenges, such as improving pedestrian safety at intersections—critical given that in the United States alone, intersections were the site of 1,705 pedestrian fatalities in 2022, representing 23% of all pedestrian deaths that year [8]. By leveraging advanced sensing and computing capabilities, these applications can respond to the immediate needs of urban environments, making interactions within cities safer and more context-aware. Yet, despite the increasing instrumentation available in smart city testbeds, creating effective urban applications remains challenging due to the lack of a unified framework that offers consistent abstractions and controlled access to shared resources [28].

Current urban systems tend to be isolated, vertically integrated stacks that are tightly bound to specific hardware and software configurations. They often fail to integrate diverse data sources, resulting in fragmented information and limited situational awareness. Applications tailored to one environment are difficult to adapt to others, which severely limits scalability and reusability across different urban settings. Additionally, privacy concerns are frequently overlooked, hindering user trust and adoption. These siloed systems often lack high-level programmability and enforceable access controls, making it hard to bridge the gap between application policies and low-level system operations. Without consistent abstractions and robust access controls, developers are left with ad-hoc solutions rather than a dependable,

* Corresponding authors.

standardized approach to harness the potential of smart city infrastructure [41].

To address these challenges, we introduce the **Streetscape Application Services Stack (SASS)**, a structured framework that supports the development of complex urban applications. SASS tackles the limitations of previous frameworks by providing a modular, distributed application stack designed specifically for smart cities. With an SDK and a set of APIs, SASS abstracts the complexity of underlying hardware and software configurations, enabling developers to work with diverse urban sensors and components as manageable, composable units.

SASS is built to meet the specific demands of urban applications, which require precise multimodal data synchronization, spatiotemporal data fusion, and low-latency edge computing. It synchronizes data streams from heterogeneous sensor types, integrates spatially distributed data over time, and performs edge-based processing to reduce latency and conserve bandwidth. By offering these core services as modular, composable components, SASS simplifies the creation and deployment of applications in dynamic urban environments, enabling flexible and robust solutions that adapt to varied infrastructure setups.

We evaluated SASS's performance across different real-world testbeds: a controlled parking lot and a city-scale mobile wireless testbed in New York City (COSMOS) [27]. These evaluations confirm SASS's ability to support scalable, high-performance urban applications by providing the necessary modular components and architectural support.

The key contributions of this work are as follows:

- **Introduction of the Streetscape Application Services Stack:** SASS provides a novel, modular framework with standardized abstractions and controlled resource access, addressing the programmability and portability gaps in existing urban systems. Through its SDK and APIs, SASS facilitates application development in complex urban settings.
- **Architectural Design for Core Urban Application Properties:** We identify three critical properties for urban applications—multimodal synchronization, spatiotemporal data fusion, and low-latency edge processing—and design specialized services to support them. These services provide a scalable foundation that simplifies data synchronization, integration, and real-time processing across diverse urban sensors.
- **Implementation of Advanced Application Services:** We present three specialized services: Multimodal Sensor Synchronization, Multicamera Detection, and Distributed Edge Processing. These services introduce new techniques, including a two-stage event-based synchronization algorithm, a neural network-based multicamera fusion model, and a decay-based dynamic scheduling algorithm.

While existing smart city frameworks have tackled aspects of urban sensing and processing, they typically lack the modularity and composable abstractions necessary for handling the complexity of diverse, distributed urban sensors at scale. SASS addresses these limitations by providing a structured, service-oriented framework tailored to the demands of real-time, multimodal streetscape applications. By bridging the gap between low-level sensor management and high-level application logic, SASS enables scalable and adaptable solutions across various urban testbeds. In the following section, we examine prior work on urban sensing frameworks and multimodal data processing, outlining the key challenges that have influenced SASS's design.

2 RELATED WORK

The Streetscape Application Services Stack (SASS) distinguishes itself as a novel system architecture that integrates multimodal data synchronization, spatiotemporal data fusion, and edge computing, addressing critical gaps in real-time, distributed urban applications. In this section, we contrast SASS capabilities with existing work across three principal application domains.

2.1 Multimodal Data Synchronization

In urban environments, effective data synchronization across diverse sensor modalities is essential for applications requiring coordinated interpretation of events. Early efforts include Spinello et al.'s work on pedestrian detection and tracking using 2D and 3D laser range finders and cameras [31], and Piadyk et. al's *StreetAware* dataset [25] which employed multimodal integration for pedestrian tracking, while *GruMon* leveraged smartphones to detect pedestrian clusters [29]. More recently, Sukel et al. advanced audio-visual data fusion for micro-event classification in urban spaces [32].

SASS builds on these foundations by offering a robust *Data Synchronization Service* that aligns diverse data streams with advanced timestamp alignment and buffering, crucial for high-precision applications such as assistive navigation and emergency response coordination. By providing a *Multimodal Data API*, SASS enables developers to access synchronized data streams seamlessly, meeting the timing requirements of urban intelligence systems.

2.2 Spatiotemporal Data Fusion

Spatiotemporal fusion across distributed sensors is pivotal for applications that monitor dynamic urban patterns over time and space. Studies by Brunetti et al. and Tian et al. illustrate the effectiveness of multi-sensor setups for tracking urban pedestrian movement [6, 36]. However, limitations in these systems, particularly regarding scalability and cross-location adaptability, remain.

SASS overcomes these challenges by employing a dedicated *Data Fusion Engine* to aggregate sensor data based on proximity and spatial parameters. These components enable continuous, high-resolution pedestrian and vehicle tracking necessary for adaptive traffic optimization and surveillance, allowing SASS to deliver robust monitoring with minimal latency across urban scales.

2.3 Edge Computing for Real-Time Data Processing

Edge computing has been explored for smart city applications to reduce latency and improve data efficiency, as demonstrated by Shi et al. and Yu et al. in traffic and event monitoring contexts [30, 43]. Nonetheless, scalability and resource management in edge infrastructures present ongoing technical challenges, particularly for urban deployments that require adaptive, low-latency responses.

SASS extends real-time processing capabilities for smart cities through its *Edge Computing and Distributed Processing Services*, which facilitate localized data analysis for applications such as adaptive signal control. The *Resource Management Layer* further optimizes computational resources across nodes, providing reliable real-time data handling. SASS' efficient *Communication Middleware* ensures high-speed data exchange, making it ideal for large-scale, latency-sensitive applications.

2.4 Privacy-Preserving Techniques in Urban Applications

As urban systems handle increasingly sensitive data, privacy-preserving mechanisms are critical. Liu et al. propose a privacy-preserving approach using multi-armed bandits for IoT [16], while Miao et al. use crowd-sensing techniques to protect user privacy [18]. Federated learning [13, 17] and blockchain [39] have also shown promise in decentralized data security.

SASS incorporates a privacy-focused architecture, combining edge-based anonymization and server-side measures, addressing privacy requirements in pedestrian tracking applications. This design allows SASS to meet regulatory privacy standards, which is critical for secure and compliant urban deployments.

2.5 Pedestrian Safety and Mobility Applications

There has been extensive research focusing on pedestrian safety, such as *WheelShare* for accessible routing [12] and *Ghost-Probe* for blind spot detection [45]. Barón et al. studied urban walkability factors [3, 4], which impact pedestrian experiences in urban areas.

Through components like the *Multimodal Synchronization Module*, SASS advances applications for real-time safety interventions, making it particularly valuable for pedestrian detection, health monitoring, and emergency response. This integration of safety and mobility features in a single framework represents a significant contribution to urban intelligence.

2.6 IoT and Big Sensor Data Systems for Smart Cities

General IoT frameworks, including *MACeIP* and *REIP*, have enhanced urban system management [20, 26], while the Smart City Framework promotes interoperability in transportation [37]. SASS surpasses traditional systems by focusing on smart streetscape with an emphasis on real-time pedestrian and traffic monitoring. Its modular architecture allows tailored, low-latency analytics for applications like adaptive traffic control and emergency response.

3 FRAMEWORK DESIGN

To clarify the design of the Streetscape Application Services Stack (SASS), we introduce three real-world applications that emerged from smart city testbed initiatives. These applications highlight the specific needs and challenges of streetscape applications, framing the requirements that SASS is designed to meet.

3.1 Motivating Applications

Waypoint Finding for Navigation Application: This application assists visually impaired individuals in navigating urban areas by providing real-time guidance. It combines data from multiple sources, such as GPS, cameras, and wearable devices, to offer accurate directions and obstacle avoidance. The application relies on multimodal data synchronization to ensure guidance is both accurate and responsive to the user's immediate surroundings [14].

Adaptive Traffic Signal for Extended Cross Time Application: This application aims to improve pedestrian safety by dynamically adjusting traffic signal timings to accommodate individuals with varying mobility needs, like the elderly or those with disabilities. It depends on spatiotemporal data fusion from cameras and wearables to detect pedestrians and estimate crossing times, requiring low-latency processing at the edge to adjust signals in real-time [9, 19].

Pedestrian and Vehicle Detection for Urban Analytics Application: This application monitors pedestrian and vehicle flows across multiple intersections to support urban planning and safety analysis. It integrates data from distributed sensors, requiring spatiotemporal fusion to accurately track movement. Edge computing is used to handle high data volumes and enable real-time analytics [10, 11, 33, 42].

Each of these applications relies on integrating and processing data from diverse, distributed sensors in real-time to deliver hyper-local, context-aware services. *They all require the ability to synchronize multimodal data, fuse spatiotemporal information, and process data at the edge to meet performance requirements.*

3.2 Architectural Implications

Our experience developing applications on COSMOS testbed highlighted the need for shared services and higher-level abstractions to streamline development and reduce repeated work. Building each application independently revealed recurring challenges, such as managing heterogeneous data, synchronizing data streams, and ensuring low-latency processing. These challenges, which we observed across urban applications in the literature, are foundational for streetscape environments [2, 28, 41].

From this analysis, we identified three properties that are essential to address the multimodal, distributed nature of urban environments:

- **Multimodal Data Synchronization:** Synchronizing data from different sensor types in time is essential for coherent analysis and processing. Without proper alignment, data fusion can produce misleading or incorrect results. Applications that rely on this property include assistive navigation for visually impaired users [14] and emergency response systems that need precise, real-time situational data [21, 22].
- **Spatiotemporal Data Fusion:** Integrating data across time and spatial locations is essential for capturing and analyzing dynamic events and patterns in the urban landscape. Applications like multi-camera pedestrian tracking [15] and traffic flow optimization [34] depend on this capability to interpret movement and detect changes across different areas.
- **Edge Computing and Distributed Processing:** Processing data close to the source reduces latency and lowers bandwidth usage, which is needed to support real-time applications. Distributed processing improves scalability by balancing computation across multiple devices. This property is central to real-time analytics in traffic management systems [10] and adaptive traffic signal control [9].

These core properties informed SASS's architectural design. To support scalable, efficient urban applications, SASS provides services and abstractions that address these common needs. By focusing on multimodal data synchronization, spatiotemporal data fusion, and edge computing, SASS enables developers to build applications that integrate data seamlessly, perform real-time analysis, and leverage edge resources, while reducing the inherent complexity of streetscape development.

4 SYSTEM ARCHITECTURE DESIGN AND IMPLEMENTATION

SASS is a modular framework designed to handle the unique demands of urban applications by providing scalable tools for development, deployment, and management. Structured into distinct layers and services, SASS abstracts core complexities and offers comprehensive support for building urban data-driven applications.

4.1 Overall System Architecture

The SASS architecture consists of six main parts (see Figure 1): (1) Hardware Layer, (2) API Suite, (3) Data Storage, (4) Runtime Environment, (5) APP Gateway, and (6) Smart Streetscape Applications. Each layer has a defined role that contributes to efficient, scalable, and secure data processing and application support.

The **Edge Hardware Layer** acts as the system's base, comprising testbed sensors, actuators, and edge nodes for localized data capture and command execution. Sensors and actuators communicate with medium nodes over MQTT and RTSP to relay raw data and receive commands, and medium nodes further link these devices with software services. Edge nodes execute high-compute tasks and process data near its source to reduce latency. This setup also includes mobile and wearable citizen sensors, such as IMUs and GPS, adding diverse data to the system without being restricted to proprietary infrastructure.

Data Storage manages both structured and unstructured data, supporting relational databases for structured sensor data and configurations, and blob storage for unstructured data such as videos and images. A cache server temporarily stores frequently accessed data to enhance I/O performance.

The **API Suite** in SASS consists of four key APIs that enable seamless interaction between applications and the system's underlying services. The **Multimodal Data API** is designed to synchronize and fuse data from a variety of sources, providing coherent and aligned inputs for applications. It also ensures privacy compliance by applying access-based data distillation mechanisms. This API is particularly critical for applications relying on diverse sensor modalities to deliver accurate and integrated outputs.

The **Control API** facilitates actuator interactions, ensuring transaction integrity through an action queue and rollback mechanisms. By managing command consistency and recovering from partial failures, this API maintains the stability and reliability of actuator operations, enabling robust system control.

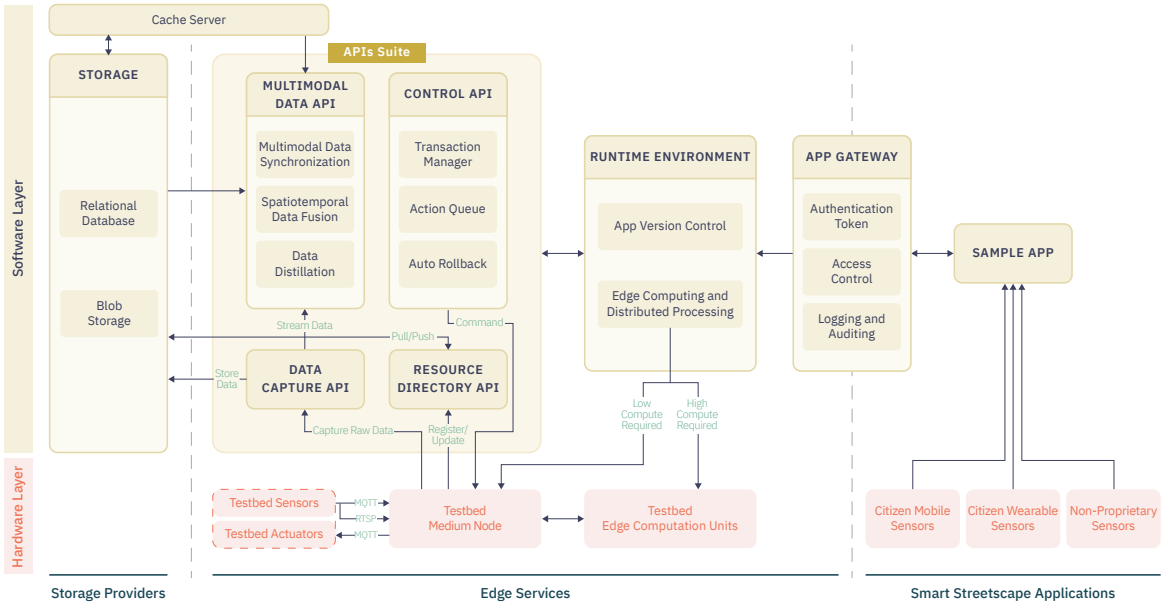

Figure 1: SASS system architecture with core services for multimodal data synchronization, spatiotemporal fusion, and edge computing. Key subsystems support device management, data routing, and distillation, while the Control API ensures transaction reliability with rollback and action queues. The App Gateway secures access through authentication and auditing, and the Runtime Environment manages resources across edge nodes and various sensor types.

The **Data Capture API** supports the acquisition and storage of both raw and processed data from various sensors and applications. It streams collected data into a centralized database, simplifying its integration into multimodal fusion workflows and analytical pipelines, which enhances system-wide data accessibility.

Finally, the **Resource Directory API** manages the registration and monitoring of IoT devices. It provides developers and users with a unified interface to track device status and efficiently manage resources. This API enhances operational oversight and ensures the effective deployment of connected devices across the system.

The **Runtime Environment** on edge nodes integrates with Git for app version control, ensuring nodes stay synchronized with the latest versions and allowing quick rollbacks if issues arise. It provides the necessary infrastructure for edge computing and distributed processing, handling resource management and network coordination to streamline task execution.

The **App Gateway** secures access to SASS's services, controlling entry points with authentication and access control. By segmenting the runtime environment, the Gateway prevents unauthorized data access and maintains user privacy. Comprehensive logging and auditing support traceability and system monitoring.

At the top, **Smart Streetscape Applications** make use of SASS services and data to address urban challenges. Applications can directly interact with citizens or support city agencies by securely sharing data for urban planning and public safety.

4.2 Core Services

SASS Core Services is the backbone for managing IoT devices in urban settings, with a modular framework that facilitates device registration, secure access, and real-time data flow management.

4.2.1 Key Components and Methodology. Core Services consists of several key components that together ensure secure and efficient urban IoT management.

The **User Management and Access Control** component manages access through a role-based system using JSON Web Tokens (JWT). Each token encodes user roles and permissions, enforcing access control across all requests. Permissions are assigned based on user roles and validated at each request to secure sensitive data.

The **Resource Directory** registers and monitors devices, storing attributes like device type, location, and access methods. Using SQLAlchemy ORM, a polymorphic base class manages diverse device types, allowing easy extension for future devices. Device-specific tokens authenticate status

```
{"device_id":"camera-001","type":"sensor","location":{
    ↪ "latitude":40.00,"longitude":-70.00,"description":
    ↪ "Alpha St."},"capabilities":["video_stream","
    ↪ detect","track"],"data_format":"H.264",
    ↪ "access_methods":{"api_endpoint":"https://generic-
    ↪ endpoint.org/","protocols":"RTSP"},"status":"
    ↪ online","last_sync_timestamp":"2024-11-07T13
    ↪ :24:02.0923","registration_timestamp":"2024-08-05
    ↪ T11:19:31.5754","owner":"Testbed"}
```

(a) Device Registration Information

```
{"timestamp":"2024-11-07T13
    ↪ :49:53.6500",
    ↪ "activity_type":"rollback",
    ↪ "details":{"device_id":"
    ↪ sensor-001",
    ↪ "old_version_id":"e4a7e88b-
    ↪ f32b4cf5-fca31f3f",
    ↪ "new_version_id":"c0aa938e
    ↪ -0c8d4c39-ca4c6681"}}
```

(b) Rollback Log (Version Control)

```
{"timestamp":"2024-10-05T21
    ↪ :19:45.3932",
    ↪ "activity_type":"update
    ↪ ","details":{
    ↪ "device_id":"sensor-001
    ↪ ","status":"maintenance
    ↪ ","last_sync_timestamp"
    ↪ :"2024-10-05T21
    ↪ :19:45.388"}}
```

(c) Device Status Log

Figure 2: Illustrative examples of the JSON-based data format used in SASS for device management and monitoring.

updates, while version histories provide tracking and operational visibility.

The **Data Capture** component acquires data from sensors, performing initial filtering, cleaning, and compression. Data is tagged with metadata such as location and timestamps, then directed to the Multimodal Data API for further processing or stored in SASS's database.

A **Data Distillation** pipeline balances data utility and privacy using anonymization, aggregation, and temporal delay techniques. Anonymization uses cryptographic hashing and location generalization, aggregation provides summary statistics, and temporal delay prevents real-time tracking by adding a time buffer. Future work includes exploring differential privacy and federated learning to enhance privacy protections.

The **Logging and Auditing** component tracks system activities like device registrations and data access, storing logs in JSON format with categorized timestamps and event types. This logging framework supports troubleshooting and compliance by providing a complete interaction history.

Version Control and Rollback leverages a UUID-based snapshot system that simplifies rollbacks by storing each configuration as a complete state, ensuring that administrators can easily restore previous configurations if needed.

The **Transaction Manager** acts as an intermediary between applications and actuators, ensuring control commands are accurately executed. The **Action Queue** sequences commands to prevent conflicts and includes an auto-rollback mechanism for error handling, restoring stable configurations when necessary. Integration testing ensures reliable operation across system components.

4.2.2 System Workflow. Core Services handle device registration, data flow, control operations, authentication, configuration management, and monitoring through a series of streamlined processes.

Applications interact with SASS through JWT-based authentication, obtaining temporary tokens that encode access permissions. When a device registers, it submits attributes like location and capabilities (see Figure 2a), and after validation, the system generates a device token for secure communication. Device status and configuration changes are logged, with rollback versions stored in the version control system. IoT devices transmit data and accept control commands through secure channels.

For data flow, devices send status updates and data through secure channels authenticated by device tokens. Data Capture manages incoming data, tagging it with metadata, synchronizing streams, and storing data in the system. For control operations, applications initiate commands through the Transaction Manager, which translates commands into actuator-friendly formats, sequences them in the Action Queue, and handles auto-rollback if errors occur (see Figure 2b). Control activities are logged, providing traceability (see Figure 2c).

4.3 Multimodal Data Synchronization Services

The *Multimodal Data Synchronization Service* aligns streams from sensors with varying rates, formats, and time accuracies, ensuring consistency across data sources, especially in real-time applications.

Incoming *edge sensor data* streams are timestamped using NTP to maintain accuracy across devices, with clock drift corrected using a Kalman filter. Timestamp corrections are calculated as: $T_{\text{corrected}} = T_{\text{local}} + \Delta T_{\text{offset}} + \Delta T_{\text{drift}} \cdot \Delta t$, where T_{local} is the device's local timestamp, ΔT_{offset} is the fixed time offset between clocks, ΔT_{drift} is the drift rate, and Δt is the elapsed time since the last synchronization. For *mobile and wearable data*, SASS supports both on-device NTP synchronization and alternative methods to reduce battery and bandwidth demands. Predefined and custom synchronization algorithms, such as Dynamic Time Warping (DTW), allow developers to tailor alignment methods to specific applications.

Adaptive buffering manages inconsistent latencies by adjusting buffer sizes based on observed data arrival times: $B = \max(B_{\min}, \beta \cdot \sigma_{\text{arrival}})$ where $B_{\min}$ is the minimum buffer

Figure 3: Perspective transformation workflow using satellite and camera imagery. Left to right: (1) Original satellite image, (2) Original camera image, (3) Extracted background, (4) Area segmentation, (5) Histogram-equalized satellite image, (6) Histogram-equalized camera image, (7) Feature matching between satellite and camera images.

Algorithm 1 Decay-Based Dynamic Scheduling Algorithm

Require: Task Queue $\{T_1, T_2, \ldots, T_n\}$, Decay factor α, Initial priority $P_{\text{initial}}(T)$

Initialize $P(T) = P_{\text{initial}}(T)$ and $W(T) = 0$ for each T in Task Queue
while true **do**
  for each T in Task Queue **do**
    $W(T) \leftarrow$ current time − entry time of T
    $P(T) \leftarrow P_{\text{initial}}(T) + \alpha \cdot \log(1 + W(T))$
  end for
  Sort Task Queue by $P(T)$ (descending)
  for each T in Task Queue **do**
    if resources available **then**
      Execute T
      Remove T from Task Queue
    end if
  end for
  Wait for the next cycle
end while

time, β is a scaling factor, and σ_{arrival} is the standard deviation of packet intervals.

4.4 Spatiotemporal Data Fusion Services

The *Spatiotemporal Data Fusion Service* integrates data from spatially distributed sensors, aligning data temporally and spatially without detailed calibration.

The *Fusion Workflow Engine* orchestrates data flow, mapping incoming data to a shared spatial and temporal framework, operating fusion at raw, feature, and decision levels. For visual sensors, *Automatic Inverse Perspective Mapping* aligns camera views with satellite imagery by computing a homography transformation matrix **T** (Figure 3). Using Affine Scale-Invariant Feature Transform (ASIFT) and RANSAC, SASS achieves accurate spatial alignment, even in complex environments. Preprocessing includes background extraction, histogram equalization, and area segmentation for enhanced alignment. Fused data is tagged with fusion metadata, enabling downstream analysis and storage.

4.5 Edge Computing and Distributed Processing Services

The Edge Computing and Distributed Processing service balances computational demands across nodes for efficiency. Tasks are broken into subtasks and classified by computational requirements. Low-compute tasks are routed to medium nodes for low latency, while high-compute tasks, like 3D pose detection, go to edge computation units. If medium nodes are overloaded, tasks are redirected to high-capacity units, maintaining uninterrupted processing.

Tasks on medium nodes are managed with our proposed **Decay-Based Dynamic Scheduling algorithm** (Algorithm 1), which prioritizes based on urgency and wait time: $P(T) = P_{\text{initial}}(T) + \alpha \cdot \log(1 + W(T))$ where $P(T)$ is task priority, α is a decay factor, and $W(T)$ is wait time. This approach balances rapid prioritization with gradual growth to prevent low-priority tasks from overtaking high-priority ones, ensuring fairness and responsiveness. A centralized resource monitor tracks utilization across all nodes, providing real-time data for efficient load balancing. Computation nodes process tasks in batches, optimized for high-throughput environments where immediate response is not critical.

5 APPLICATIONS METHODOLOGY

SASS enables a range of urban streetscape applications by streamlining data integration and reducing complexity across multimodal inputs. In Section 3.2, we outlined the core properties needed to support these applications, and Sections 4.3–4.5 detail the modular components of SASS that encapsulate these properties as scalable services. To validate SASS's capabilities, we present three real-time application services developed on SASS, each demonstrating how the stack handles the integration and processing demands unique to urban streetscapes. This section covers the methodology for each application. All experiments are conducted in a controlled parking lot and COSMOS testbed and are approved by institutional review boards (IRB), maintaining ethical standards.

5.1 Multimodal Data Synchronization

Synchronizing data from different sensor types in urban environments presents a substantial challenge given the diversity

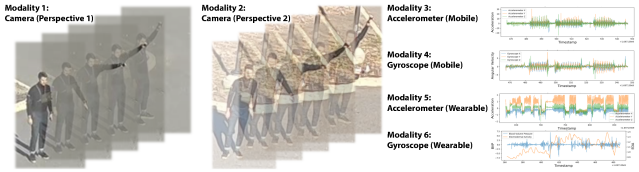

Figure 4: Multimodal Data Streams from various devices to be Synchronized based on Event Detection.

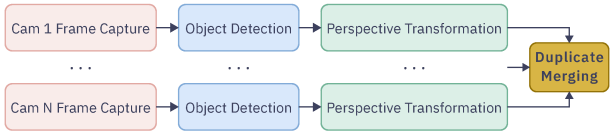

Figure 5: Multi-camera integration process. First, an object detection model is applied to each camera. Then, *CoordinateTransformNet* projects bounding boxes to a top-level view, using a Euclidean distance threshold to remove duplicates.

of data sources. Using SASS's Multimodal Data Synchronization Service, we developed a synchronization application that aligns data streams from various sensors with high efficiency. This application highlights how SASS abstracts and simplifies complex synchronization tasks, enabling consistent real-time integration without burdening the developer with low-level implementation details.

The synchronization setup included two cameras positioned around a parking lot, mobile IMU sensors, and wearable physiological sensors. Each sensor produced temporally linked data streams, as illustrated in Figure 4.

5.1.1 Synchronization Algorithm. We implemented a two-stage algorithm for optimal data alignment across multimodal streams. In the *Coarse Synchronization* stage, we detect predefined events across sensors to establish initial alignment. In *Fine-Tuning*, we minimize any remaining temporal misalignment between modalities.

5.1.2 Event-Based Synchronization. Our event for synchronization is a simple hand-raising gesture, detectable across camera feeds and IMU data streams. This simultaneous gesture, with consistent timestamps across sensors, provides a reliable synchronization point.

Gesture Detection in Video Streams Hand gestures in video streams are detected using a combination of 3D pose estimation and Dynamic Time Warping (DTW). The 3D pose estimation identifies keypoints of the hand, tracking its z-coordinate across frames. We then apply DTW-based Barycenter Averaging (DBA) to create a gesture template, which maintains key temporal dynamics [23, 24].

To detect matching gestures in new video input, we slide the gesture template over the z-coordinate time series of the hand movement. Detection is confirmed if the DTW distance falls below a preset threshold: $\mathrm{DTW}(S,T) = \min \sum_{(i,j)\in \mathrm{path}} d(s_i, t_j)$ where S and T represent the input signal and template, respectively.

Gesture Detection in IMU Streams For IMU data, we apply Hidden Markov Models (HMM) to identify hand gesture events. IMU data is segmented into sliding windows, extracting features such as mean, variance, SMA, and entropy from accelerometer and gyroscope readings. The HMM model is trained to maximize $P(O|\lambda)$, the probability of observing the feature sequence O given model λ. The Viterbi algorithm decodes the hidden state sequence to identify the gesture within each window.

5.1.3 Temporal Alignment. Once detected events are time-stamped in both video and IMU data streams, we align them within a tolerance of ±500 ms. If events fall within this window, they are considered synchronized.

5.1.4 Fine-Tuning Synchronization. The fine-tuning stage further refines alignment by pinpointing the start of the hand-drop gesture. The steps include:

(1) Applying a Butterworth low-pass filter to reduce noise.
(2) Calculating entropy in a sliding window to detect primary movement patterns.
(3) Identifying entropy peaks to locate the hand-drop start point.
(4) Backtracking to the initial rise in entropy to mark the event's initiation.

We use the visual stream's reference timestamp to synchronize the other timestamps, ensuring all streams align precisely.

5.2 Spatiotemporal Data Fusion

Spatiotemporal data fusion creates a unified view of urban intersections by combining data from spatially distributed sensors. In environments with overlapping camera coverage, fusion improves object detection and situational awareness. This application service integrates detections from multiple cameras to enhance accuracy and coverage, transforming coordinates into a unified top-down view for real-time scene interpretation. Our test setup included two high-resolution cameras overlooking an intersection from perpendicular angles, capturing video at 3840x2160 resolution.

5.2.1 Fusion Algorithm. The fusion process involves several sequential steps, shown in Figure 5. We begin by applying YOLOv9e [40] to detect pedestrians and vehicles in each video stream. For high-resolution input, slicing-aided

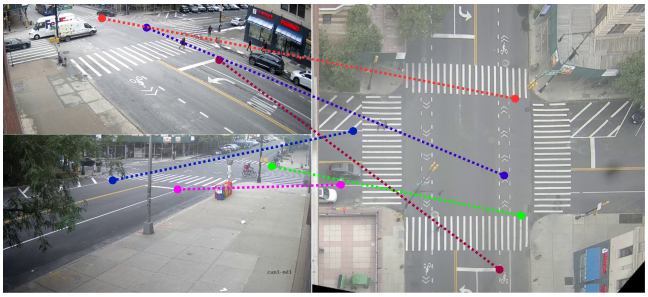

Figure 6: A set of corresponding points from the street-level cameras' perspective and the top-level view perspective is used to train *CoordinateTransformNet*.

hyper inference [1] reduces computational overhead. Bounding boxes and class probabilities are generated in each camera's coordinate system. Detected coordinates are then transformed using *CoordinateTransformNet*, a lightweight neural network trained using a set of point pairs to map each perspective into a unified top-down view (Figure 6) To further refine detections, we apply a Euclidean distance threshold for de-duplication. If the distance between detections from different cameras falls below this threshold, they are treated as duplicates, and we use confidence-weighted averaging to maintain data accuracy.

5.3 Distributed Edge Computing for Real-Time Data Processing

Edge processing is necessary for low-latency applications in urban environments. This application service uses SASS's Edge Computing and Distributed Processing Services to process data at the network edge with minimal latency. The objective is a real-time processing pipeline that detects and tracks pedestrians, estimates trajectories and poses, and visualizes results on a dashboard. Our setup included mobile IMU sensors and a high-resolution camera in two testbed environments: the controlled parking lot and a dynamic intersection in COSMOS testbed.

IMU readings and camera frames are captured and processed in parallel. For mobile sensor data, only IMU readings from individuals within the target area are used, determined by location metadata. Frames from edge cameras undergo object detection via YOLOv8 [35]. We enhance tracking continuity using the OC-SORT algorithm with shadow tracking for unmatched tracks [7]. Trajectories, 3D bounding box, and 2D poses are estimated, with results displayed on a dashboard for spatial analysis. Bounding boxes and 2D poses are then fed to MotionBERT [46] for 3D pose estimation, with a Kalman filter applied to smooth keypoint trajectories. Processed data streams are visualized on an interactive dashboard.

6 IMPLEMENTATION AND EVALUATION

In this section, we present the implementation of the mentioned application services developed using SASS. We further evaluate each service based on specific performance metrics demonstrating how SASS enables robust, high-performance urban applications across diverse, sensor-rich environments.

6.1 Multimodal Sensor Synchronization

The development of our synchronization application service was significantly facilitated by the Multimodal Data Synchronization Services provided by SASS. By leveraging these services, we abstracted away the complexities associated with precise timestamping, clock synchronization, data buffering, and alignment across heterogeneous sensor modalities. This allowed us to focus on high-level synchronization logic and algorithm development, resulting in an efficient and robust synchronization method integrated within the SASS architecture.

The Multimodal Data Synchronization Services within SASS offered core functionalities that streamlined the development process. SASS ensured consistent time references across edge sensors by implementing *precise timestamping mechanisms and clock synchronization protocols*. The services also applied clock drift correction to adjust local timestamps. The services managed variations in data arrival times and sampling rates through *adaptive buffering and resampling mechanisms*. By dynamically adjusting buffer sizes based on real-time monitoring of data arrival variability, the system accommodated network delays and processing latencies without significant impact on synchronization accuracy. SASS provided a *library of predefined algorithms* for data synchronization, including DTW and HMM, which allowed us to utilize these algorithms directly without implementing them from scratch. By utilizing these components, we efficiently implemented our novel, user-defined two-stage synchronization algorithm outlined in Section 5.1. SASS simplified the development by handling low-level data management tasks, providing reusable algorithms, supporting scalability for the addition of more sensors and modalities, and improving performance by adaptively buffering the data stream.

To further evaluate the effectiveness of our synchronization method, we conducted experiments in our controlled parking lot environment, collecting over an hour of data from various sensors. Eight individuals performed walking and hand-raising/dropping gestures while holding a mobile phone and wearing a wristband. We first evaluated the performance of our gesture detection algorithms in both the visual and IMU data streams.

In the vision domain, a gesture template was created using DTW-based Barycenter Averaging (DBA), which maintained the intrinsic temporal dynamics without distorting

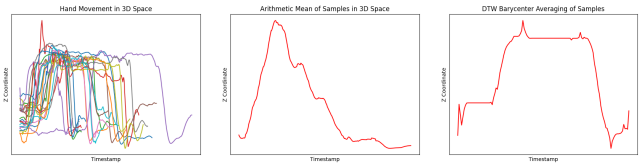

Figure 7: Gesture Samples (Left), their Arithmetic Average w/o Temporal Variations (Middle), and the DTW Barycenter Average w/ Temporal Dynamics used as the Template (Right)

key trends (Figure 7). The DTW distance threshold was set to 0.8, and the algorithm was applied using sliding windows of 4 seconds. In the IMU domain (mobile devices and wristbands), Features such as mean, variance, standard deviation, Signal Magnitude Area (SMA), and entropy were extracted from the accelerometer and gyroscope signals. The HMM was trained over multiple iterations to maximize the likelihood of observing the feature sequence given the model. The precision, recall and F1-score were calculated for both domains and reported in Table 1.

After detecting synchronization events in both modalities, we applied our synchronization method to align the data streams. We quantified the effectiveness of our synchronization using Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Time Offset (MTO) between the ground truth event start times and the predicted start times from the synchronized data streams. The initial discrepancies between sensor streams averaged approximately 400 ms due to unsynchronized clocks and network delays. After applying our synchronization method, these discrepancies decreased to less than 50 ms, representing an 88% reduction in temporal misalignment. The results are detailed in Table 2. The significant reduction in synchronization error demonstrates the effectiveness of our method and the critical role of SASS's Multimodal Data Synchronization Services in achieving precise alignment across heterogeneous sensor modalities.

6.2 Multicamera Detection

Our multicamera detection application service greatly benefited from the Spatiotemporal Data Fusion Services offered by SASS. These services simplified the process of data acquisition, processing, and integration, allowing us to concentrate on high-level data fusion algorithms and application logic. As a result, we achieved a streamlined and resilient detection system, seamlessly embedded within the SASS framework.

The Spatiotemporal Data Fusion Services within SASS provided core functionalities that streamlined the development process. Fusion Workflow Engine, at the core of these services, orchestrates the fusion process by managing data

Table 1: Event Detection Accuracy: Camera (IMU)

Event Type	Recall	Precision	F1
Gesture Event	0.64 (0.61)	1.00 (0.92)	**0.78 (0.73)**
Non-Event	1.00 (1.00)	0.91 (0.98)	0.95 (0.99)

Table 2: Performance Metrics (Seconds) by Modality

Modality	MAE	RMSE	MTO
Camera (IMU)	0.054 (0.032)	0.069 (0.039)	0.033 (0.006)
Average	0.04323	0.0545	0.0197

flow between each component, enabling real-time fusion. It first yields time-aligned frames from cameras overlooking the area of interest. This is done using the location data and synchronized timestamps embedded in the metadata of the input data stream provided by Data Capture API. The flexibility of SASS in supporting multiple abstraction levels of fusion (early, intermediate, and late), allowed us to apply an intermediate fusion workflow by utilizing bounding box and confidence features obtain from raw sensor data processed by object an detection model.

To transform the coordinates from each camera's perspective into a unified top-down view, we integrated our custom neural network, *CoordinateTransformNet*, into the Fusion Algorithms Library provided by the Spatiotemporal Data Fusion Services. Instead of relying on the built-in inverse perspective calibration, this choice demonstrated the modular architecture and adaptability of the framework by allowing developers to integrate custom fusion algorithms tailored to specific application needs. Finally, the Fusion Workflow Engine handled integrating the custom processes into its overall workflow as part of a larger, adaptable fusion pipeline. By abstracting these computational complexities, SASS enabled the efficient deployment of our multicamera detection service.

To further assess the impact of multi-camera integration on detection accuracy, we conducted experiments using a dataset of 900 images captured simultaneously from three cameras as illustrated in Figure 6 (two street-level cameras on first and second floors and one high-altitude view camera on 12^{th}). To further enhance detection accuracy, we fine-tuned the object detection model using a manually annotated dataset recorded from the same high-altitude camera [38]. As a result, the detection results from the 12^{th} floor camera can be considered a reliable ground truth. We performed object detection using the YOLOv9e model [40] on the street-level cameras. The detection results were transformed using

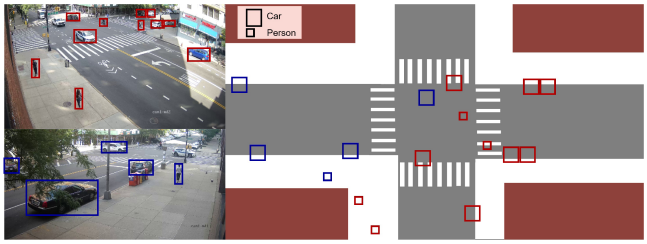

Figure 8: Detected bounding boxes from two perpendicular perspectives are integrated into the top-view perspective.

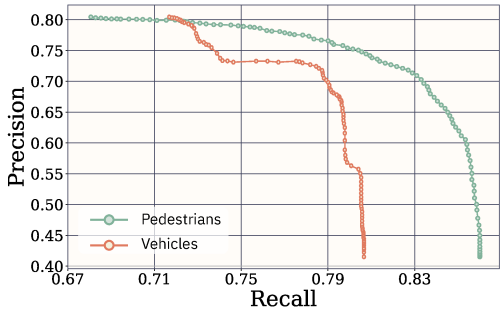

Figure 9: Precision-Recall Curves for Multi-Camera Integration, with the Duplicate Removal Threshold Reducing from 5.5 m to 0 m from Left to Right.

CoordinateTransformNet, integrated within the Fusion Algorithms Library, as illustrated in Figure 8. The transformed detections were then compared against the ground truth from the high-altitude camera on temporally aligned frames.

We present the precision, recall, and F1 score values for the pedestrian and vehicle classes separately in Table 3. The results indicate that integrating the detections from both cameras enhances accuracy by over 10% for both classes. We also analyzed the impact of varying the duplicate removal threshold on detection performance. Figure 9 shows the precision-recall curves for both classes as the threshold is reduced from 5.5 meters to 0 meters. For pedestrians, reducing the threshold led to a gradual decline in precision and a rise in recall, indicating fewer false positives but more false negatives. For vehicles, precision dropped more sharply, highlighting difficulties in accurately pinpointing vehicle centers. These results demonstrate the advantages of our method and the key role of SASS's Spatiotemporal Data Fusion Services in the seamless implementation of such multi-camera transformation models.

6.3 Distributed Edge Processing

The development and deployment of our real-time edge processing application service was significantly enhanced by

Table 3: Detection Metrics: Pedestrians (Vehicles)

Metric	1st Cam	2nd Cam	Multi-Cam
Recall	0.609 (0.681)	0.813 (0.558)	0.790 (0.720)
Precision	0.769 (0.510)	0.635 (0.684)	0.766 (0.617)
F1 Score	0.680 (0.583)	0.713 (0.615)	**0.778 (0.665)**

the Edge Computing and Distributed Processing Services in SASS. These services streamlined task decomposition, computation distribution, and resource allocation, enabling us to maintain responsiveness and focus on high-level processing logic. This resulted in the processing of high-volume, computationally intensive tasks with minimal latency.

To support this application, SASS decomposed the primary task into several subtasks: mobile sensor capture, video frame acquisition and preprocessing, object detection, object tracking, trajectory calculation, 3D bounding box estimation, 2D and 3D pose estimation, and visualization. Each subtask was evaluated for computational load, with lighter tasks—such as frame capture/preprocessing, detection, tracking, and trajectory estimation—processed on edge medium nodes for low-latency results. Resource-intensive tasks, including 3D bounding box construction, 2D/3D pose estimation, and visualization, were offloaded to the Edge Computation Unit. This hierarchical distribution allows parallel subtask processing at independent rates, optimizing resource use and processing efficiency for downstream tasks. A key feature of this service, the scheduling algorithm, efficiently manages tasks on edge medium nodes by dynamically prioritizing them based on urgency and wait time. This prevents low-priority tasks from delaying high-priority, low-latency operations, optimizing throughput while preserving responsiveness across the entire subtasks. Additionally, SASS's centralized resource monitor tracks node utilization in real time, enabling dynamic load management to reroute tasks as needed, preventing overloads, and ensuring consistent processing speeds.

To further evaluate the performance of tasks within this module, the Edge Computing and Distributed Processing Services were tested for efficiency under demanding conditions. With an input video rate of 30 fps, the system achieves preprocessing, detection, tracking, and trajectory calculation results with a *latency* of ≈30 milliseconds, enabling real-time operation of these subtasks. Computationally intensive stages, such as 3D bounding box construction and 2D/3D pose estimation, are seamlessly offloaded to the Edge Computation Unit, achieving a combined *latency* of ≈45 milliseconds. Additionally, mobile sensor capture was handled directly on the Edge Computation Unit and used by the visualization dashboard, as these inputs originate from the application rather than edge sensors. The visualization task,

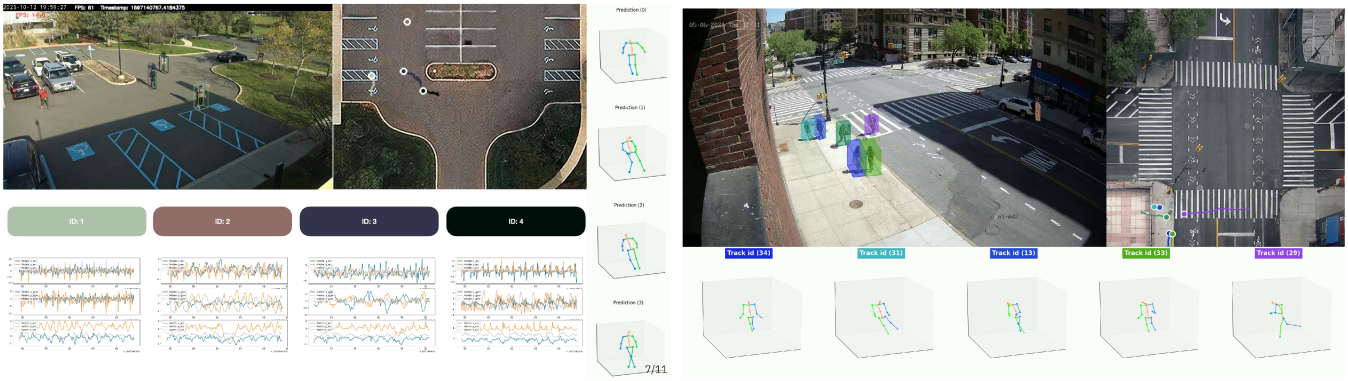

Figure 10: Analytics visualization dashboard showing live video feeds, IMU data streams, 3D bounding boxes, trajectories in both camera view and BEV, and 2D/3D pose estimations for pedestrians in urban and parking lot settings, processed with low-latency edge computing.

which is resource-heavy due to 3D plotting, is processed at a latency of ≈200 milliseconds (Figure 10).

By processing subtasks independently and allowing downstream tasks to access data streams at customized output rates, SASS ensures flexible support for diverse application needs. The system *throughput* for this pipeline reaches ≈57 tasks per second, facilitated by task segmentation and edge distribution. This is more than tenfold better than only 5 tasks per second without task decomposition, as the most resource-intensive task (visualization) operates at 5 fps. *Task offloading* to the Edge Computation Unit constitutes about 47% of total tasks, as these are more demanding processes.

We also evaluated the proposed scheduling algorithm across three key metrics. In our setup, we set initial priority value 0 and 1 for high- and low-priority tasks, and set α to 1. First, we measured the time taken by the scheduler to process, which revealed a low *scheduling overhead* of average 0.07 milliseconds, ensuring minimal processing delay. We also measured the average time and variance from when each task enters the queue to when it is selected. The algorithm maintained an *average latency* of 240 milliseconds and 4660 milliseconds and a *wait time variance* of 390 milliseconds and 2670 milliseconds for high- and low-priority tasks, respectively. Lastly, we calculated the percentage of instances where lower-priority tasks were processed before higher-priority tasks due to the dynamic re-prioritization. The *priority inversion rate* of 3% underscores the algorithm's ability to sustain fair and responsive task processing.

7 CONCLUSION

This paper presented the Streetscape Application Services Stack (SASS), a novel framework that addresses the complexities of developing real-time, multimodal applications in urban environments. By providing modular services for multimodal data synchronization, spatiotemporal data fusion, and distributed edge computing, SASS abstracts the underlying challenges associated with integrating heterogeneous sensor modalities, varying spatial and temporal data, and low-latency edge processing.

Our contributions include the design of SASS as a programmable, scalable framework that offers uniform abstractions and controlled access to shared urban resources. We identified three essential requirements for urban applications—multimodal synchronization, data fusion, and low-latency edge processing—and implemented specialized services to support these needs. SASS enables developers to build and deploy complex, responsive applications through an SDK and APIs. We demonstrated SASS's capabilities through three real-time application services: Multimodal Sensor Synchronization, Multicamera Detection, and Distributed Edge Processing. These services, supported by custom algorithms for event-based synchronization, multicamera fusion, and dynamic scheduling, showcase SASS's adaptability for sophisticated urban analytics.

Furthermore, the promising performance metrics observed in our evaluation of these developed applications substantiate SASS's ability to support the development of highly responsive and efficient urban applications. The Multimodal Data Synchronization service reduced temporal misalignment by 88%, the Spatiotemporal Data Fusion service improved detection accuracy by over 10%, and the Distributed Edge Computing service increased throughput tenfold, reaching 57 tasks per second. These results confirm that SASS's modular design and efficient resource allocation can successfully facilitate complex, real-time streetscape applications, supporting future advancements in smart city solutions.

Acknowledgments

This work was supported by the National Science Foundation (NSF) and Center for Smart Streetscapes (CS3) under NSF Cooperative Agreement No. EEC-2133516, NSF Grant CNS-2148128, NSF Grant CNS-2038984, and corresponding support from the Federal Highway Administration (FHWA).

References

[1] Fatih Cagatay Akyon, Sinan Onur Altinuc, and Alptekin Temizel. 2022. Slicing Aided Hyper Inference and Fine-tuning for Small Object Detection. *2022 IEEE International Conference on Image Processing (ICIP)* (2022), 966–970.

[2] Eiman Al Nuaimi, Hind Al Neyadi, Nader Mohamed, and Jameela Al-Jaroodi. 2015. Applications of big data to smart cities. *Journal of Internet Services and Applications* 6, 1 (2015), 1–15.

[3] Alejandro Barón, Cristina Aranda, and Antonio Estepa. 2018. Investigating the walking accessibility, usability, and UX for a blind user of michelin-starred restaurants' websites. *IEEE Access* 6 (2018), 30770–30781.

[4] Nicole Barón, Stephanie Romero, Christiane Schönfeld, and Vera Pavlakovich-Kochi. 2018. Walkability and safety around elementary schools: Economic and ethnic disparities. *Journal of transport & health* 10 (2018), 236–245.

[5] Simon Elias Bibri and John Krogstie. 2017. Smart sustainable cities of the future: An extensive interdisciplinary literature review. *Sustainable cities and society* 31 (2017), 183–212.

[6] Antonio Brunetti, Domenico Buongiorno, Gianpaolo F Trotta, and Vitoantonio Bevilacqua. 2018. Computer vision and deep learning techniques for pedestrian detection and tracking: A survey. *Neurocomputing* 300 (2018), 17–33.

[7] Jinkun Cao, Jiangmiao Pang, Xinshuo Weng, Rawal Khirodkar, and Kris Kitani. 2023. Observation-centric sort: Rethinking sort for robust multi-object tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9686–9696.

[8] Insurance Institute for Highway Safety (IIHS). [n. d.]. Pedestrian Fatality Statistics. https://www.iihs.org/topics/fatality-statistics/detail/pedestrians. Accessed: 2024-06-24.

[9] Yongjie Fu and Xuan Di. 2023. Federated Reinforcement Learning for Adaptive Traffic Signal Control: A Case Study in New York City. In *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 5738–5743.

[10] Mahshid Ghasemi, Sofia Kleisarchaki, Thomas Calmant, Levent Gürgen, Javad Ghaderi, Zoran Kostic, and Gil Zussman. 2022. Real-time camera analytics for enhancing traffic intersection safety. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 630–631.

[11] Mahshid Ghasemi, Zoran Kostic, Javad Ghaderi, and Gil Zussman. 2021. Auto-SDA: Automated video-based social distancing analyzer. In *Proceedings of the 3rd ACM workshop on hot topics in video analytics and intelligent edges*. 7–12.

[12] Anhong Guo, Andrew Payne, and Stacy Kuznetsov. 2018. WheelShare: Crowd-sourced surface classification for accessible routing. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*. 414–416.

[13] Hongyu Guo, Yaochu Ren, Shuai Liu, Qian Chen, Xiaodong Li, and Jing Xu. 2021. Multi-device federated learning: Challenges, methods, and future directions. *IEEE Wireless Communications* 28, 3 (2021), 144–152.

[14] Gaurav Jain, Basel Hindi, Zihao Zhang, Koushik Srinivasula, Mingyu Xie, Mahshid Ghasemi, Daniel Weiner, Sophie Ana Paris, Xin Yi Therese Xu, Michael Malcolm, et al. 2024. StreetNav: Leveraging street cameras to support precise outdoor navigation for blind pedestrians. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–21.

[15] Joao Paulo Lima, Rafael Roberto, Lucas Figueiredo, Francisco Simoes, and Veronica Teichrieb. 2021. Generalizable multi-camera 3d pedestrian detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 1232–1240.

[16] Yufeng Liu, Jiawen Chen, and Shan Huang. 2020. Privacy-preserving collaborative learning for multiarmed bandit in IoT. *IEEE Internet of Things Journal* 7, 11 (2020), 11175–11186.

[17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. *Artificial intelligence and statistics* (2017), 1273–1282.

[18] Chao Miao, Lu Su, Wenjun Jiang, Yaliang Li, and Mengdi Tian. 2019. Privacy-preserving truth discovery in crowd sensing systems. *ACM Transactions on Sensor Networks (TOSN)* 15, 4 (2019), 1–32.

[19] Zhaobin Mo, Wangzhi Li, Yongjie Fu, Kangrui Ruan, and Xuan Di. 2022. CVLight: Decentralized learning for adaptive traffic signal control with connected vehicles. *Transportation research part C: emerging technologies* 141 (2022), 103728.

[20] Truong Thanh Hung Nguyen, Phuc Truong Loc Nguyen, Monica Wachowicz, and Hung Cao. 2024. MACeIP: A Multimodal Ambient Context-enriched Intelligence Platform in Smart Cities. *arXiv preprint arXiv:2409.15243* (2024).

[21] Audrey Olivier, Matt Adams, Sevin Mohammadi, Andrew Smyth, Kathleen Thomson, Timothy Kepler, and Monish Dadlani. 2022. Data analytics for improved closest hospital suggestion for EMS operations in New York City. *Sustainable Cities and Society* 86 (2022), 104104.

[22] Audrey Olivier, Sevin Mohammadi, Andrew W Smyth, and Matt Adams. 2023. Bayesian neural networks with physics-aware regularization for probabilistic travel time modeling. *Computer-Aided Civil and Infrastructure Engineering* 38, 18 (2023), 2614–2631.

[23] François Petitjean, Germain Forestier, Geoffrey I Webb, Ann E Nicholson, Yanping Chen, and Eamonn Keogh. 2014. Dynamic time warping averaging of time series allows faster and more accurate classification. In *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE, 470–479.

[24] François Petitjean, Alain Ketterlin, and Pierre Gançarski. 2011. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition* 44, 3 (2011), 678–693.

[25] Yurii Piadyk, Joao Rulff, Ethan Brewer, Maryam Hosseini, Kaan Ozbay, Murugan Sankaradas, Srimat Chakradhar, and Claudio Silva. 2023. Streetaware: A high-resolution synchronized multimodal urban scene dataset. *Sensors* 23, 7 (2023), 3710.

[26] Yurii Piadyk, Bea Steers, Charlie Mydlarz, Mahin Salman, Magdalena Fuentes, Junaid Khan, Hong Jiang, Kaan Ozbay, Juan Pablo Bello, and Claudio Silva. 2022. REIP: A Reconfigurable Environmental Intelligence Platform and Software Framework for Fast Sensor Network Prototyping. *Sensors* 22, 10 (2022), 3809.

[27] Dipankar Raychaudhuri, Ivan Seskar, Gil Zussman, Thanasis Korakis, Dan Kilper, Tingjun Chen, Jakub Kolodziejski, Michael Sherman, Zoran Kostic, Xiaoxiong Gu, et al. 2020. Challenge: COSMOS: A city-scale programmable testbed for experimentation with advanced wireless. In *Proc. ACM MobiCom*.

[28] Eduardo Felipe Zambom Santana, Ana Paula Chaves, Marco Aurelio Gerosa, Fabio Kon, and Dejan S Milojicic. 2017. Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. *ACM Computing Surveys (Csur)* 50, 6 (2017), 1–37.

[29] Rijurekha Sen, Youngki Lee, Kasthuri Jayarajah, Archan Misra, and Rajesh Krishna Balan. 2018. GruMon: Fast and accurate group monitoring for heterogeneous urban spaces. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*. 51–63.

[30] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[31] Luciano Spinello, Kai O Arras, Rudolph Triebel, and Roland Siegwart. 2010. People detection in 3D point clouds using distant and height layered grids. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2010), 3545–3550.

[32] Maarten Sukel, Stevan Rudinac, and Marcel Worring. 2019. Multimodal classification of urban micro-events. *Proceedings of the 27th ACM International Conference on Multimedia* (2019), 1455–1463.

[33] Yuan Sun, Navid Salami Pargoo, Peter Jin, and Jorge Ortiz. 2024. Optimizing Autonomous Driving for Safety: A Human-Centric Approach with LLM-Enhanced RLHF. In *Companion of the 2024 on ACM International Joint Conference on Pervasive and Ubiquitous Computing*. 76–80.

[34] Zheng Tang, Milind Naphade, Ming-Yu Liu, Xiaodong Yang, Stan Birchfield, Shuo Wang, Ratnesh Kumar, David Anastasiu, and Jenq-Neng Hwang. 2019. Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8797–8806.

[35] Juan Terven and Diana Cordova-Esparza. 2023. A comprehensive review of YOLO: From YOLOv1 to YOLOv8 and beyond. *arXiv preprint arXiv:2304.00501* (2023).

[36] Yonglong Tian, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2019. Pedestrian detection aided by deep learning semantic tasks. *CVPR* (2019).

[37] Vlasios Tsiatsis, Pramod Anantharam, Payam Bamanga, Martin Fischer, Frederik Ganz, Muhammad Imran Ali, Gabi Nechifor, Daniel Keuper, Alexandra Muldoon, Konstantinos Kokkinos, Christian Sailer, Dan Paul, and Ralf Torjus. 2016. *Smart City Framework: Real-Time IoT Stream Processing and Large-Scale Data Analytics for Smart City Applications*. Technical Report. European Commission.

[38] Mehmet Kerem Turkcan, Sanjeev Narasimhan, Chengbo Zang, Gyung Hyun Je, Bo Yu, Mahshid Ghasemi, Javad Ghaderi, Gil Zussman, and Zoran Kostic. 2024. Constellation Dataset: Benchmarking High-Altitude Object Detection for an Urban Intersection. *arXiv preprint arXiv:2404.16944* (2024).

[39] Anil Vangala, Ashok Kumar Das, Neeraj Kumar, and Mamoun Alazab. 2021. Blockchain-enabled certificate-based authentication for smart energy systems. *IEEE Transactions on Industrial Informatics* 17, 6 (2021), 4450–4461.

[40] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. 2024. Yolov9: Learning what you want to learn using programmable gradient information. *arXiv preprint arXiv:2402.13616* (2024).

[41] Pengjun Wu, Zhanzhi Zhang, Xueyi Peng, and Ran Wang. 2024. Deep learning solutions for smart city challenges in urban development. *Scientific Reports* 14, 1 (2024), 5176.

[42] Tong Wu, Navid Salami Pargoo, and Jorge Ortiz. 2023. Multi-sensor Fusion for In-cabin Vehicular Sensing Applications. In *Proceedings of the 22nd International Conference on Information Processing in Sensor Networks*. 332–333.

[43] Wei Yu, Fan Liang, Xiaofei He, William Grant Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. 2017. A survey on the edge computing for the Internet of Things. *IEEE access* 6 (2017), 6900–6919.

[44] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. 2014. Internet of things for smart cities. *IEEE Internet of Things journal* 1, 1 (2014), 22–32.

[45] Yue Zhang, Jiaqi Gu, Zhiqian Cao, Kaigui Xu, Xuan Gao, and Prasant Mohapatra. 2020. Ghost-Probe: Detecting Vehicle Blind Spots for Pedestrian Safety. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 529–541.

[46] Wentao Zhu, Xiaoxuan Ma, Zhaoyang Liu, Libin Liu, Wayne Wu, and Yizhou Wang. 2023. Motionbert: A unified perspective on learning human motion representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 15085–15099.