Pattern Matching for Perception Streams

Jacob Anderson[0000−0002−4491−9399], Georgios Fainekos[0000−0002−0456−2129], Bardh Hoxha[0000−0001−6255−7566], Hideki Okamoto[0009−0009−2533−9581], and Danil Prokhorov[0000−0002−6208−4233]

Toyota Motor North America, Research & Development, Ann Arbor, MI, USA
{<firstname>.<lastname>}@toyota.com

Abstract. We introduce Spatial Regular Expressions (SpREs) as a novel querying language for pattern matching over perception streams containing spatial and temporal data. To highlight the capabilities of SpREs, we developed the STREM tool as a matching framework that works in both the offline and online domain. We demonstrate the tool through an offline example with an AV dataset, an online example through an integration with the ROS and CARLA simulators, and an initial set of performance benchmarks on various SpRE queries. From our designed matching framework, we are able to find over 20,000 matches within 296 ms making it highly usable in runtime monitoring applications.

Keywords: Pattern matching · Regular expressions · Spatial logic · Computer vision · Runtime monitoring

1 Introduction

Perception systems are utilized across a wide range of applications—from Autonomous Vehicles (AVs) [20,43,29], to sports media analysis [35,40], to Closed-Circuit Televisions (CCTVs) [36], and more [14,37,21,27]. These systems may be composed of various sensor and sensor fusion technologies such as Light Detecting and Radar (LiDAR), radar, cameras, etc. to support complex Computer Vision (CV) tasks in both the *offline* and *online* domain that generate and require a significant amount of data to effectively operate [5]. To improve upon these perception systems and further Machine Learning (ML) activities, large datasets are released for CV tasks in hopes of providing more exposure to these systems before deployment [26,13]. These perception-based datasets are further extended to Autonomous Driving System (ADS) applications where the perception system consists of a suite of sensors. Examples of such datasets include the popular Waymo Open [34], Woven Planet ("L5") Perception [22], and NuScenes [9] along with several others [30,42,39]. Therefore, as these perception systems become more comprehensive, methods and tools that enable querying of such stream data for specific scenarios in *testing*, *training*, and *monitoring* become increasingly important.

For a given perception stream, however, filtering and searching for scenarios of interest is not well-supported nor a ubiquitous process as the size of data, se-

lected schema, and present sensor suite varies. Within the offline setting, perception streams produced by AV companies provide minimal frameworks to interface and filter data according to a pre-defined schema. As for the online setting, perception systems streaming data in real-time are not traditionally responsible for identifying scenarios. Therefore, this work aims to address the problem of querying complex and dynamic perception streams comprised of spatially- and temporally-aligned data offline and online.

In the work presented in this paper, we introduce Spatial Regular Expressions (SpREs): a novel querying language for efficient and flexible matching of perception streams. SpREs combine Regular Expressions (REs) [1] with the modal logic of topology $\mathcal{S}4_u$ [25]. The language is designed with ease-of-use in mind by following syntactic similiarities of classic RE tools such as *grep* and *egrep* [16] while enabling reasoning over topological relations. The querying language has been implemented in the Spatio-Temporal Regular Expression Matcher (STREM) tool to support offline and online searching capabilities. The SpRE queries can be efficiently solved due to the reduction of the pattern matching problem to the one for REs, which in turn allows us to utilize well-established libraries [18] for fast processing. Furthermore, its modular design and compatibility with Linux, Bash, and any Command Line Interface (CLI) tools make the system more versatile and extendable for use in *verification* and *validation* process pipelines. Our formulation of the SpRE language (Section 3) is also general enough to utilize other branches of logic without major restructuring.

Contributions. From this work, the set of contributions are as follows:

- The RE $\times$ $\mathcal{S}4_u$ querying language (and associated semantics) for pattern matching spatio-temporal sequences of events in perception streams.
- The STREM tool for offline and online pattern matching of perception streams with the novel querying language.
- An offline demonstration of the STREM tool using the Woven Planet ("L5") Perception AV dataset.
- An online demonstration of STREM through integration with the Robot Operating System (ROS) and Car Learning to Act (CARLA) simulator stack.

2 Preliminaries

We let $\mathbb{Z}$ be the set of all integers, $\mathbb{N}$, $\mathbb{N}_0$ be the set of natural numbers with and without 0, and $\mathbb{R}$ be the set of real numbers. Furthermore, $\mathbb{B}$ represents the set of booleans $\{\top, \bot\}$ where $\top$ and $\bot$ are the boolean constants true and false, respectively. Furthermore, given a set A, $\mathcal{P}(A)$ denotes the powerset of A, and $|A|$ represents the cardinality of A.

2.1 Perception Stream

We consider a perception stream $\mathcal{S} = \mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \ldots$ to be a discrete sequence of frames $\mathcal{F}_i$ where $i \in \mathbb{N}_0$ represents the i^{th} frame of the stream. We use $\preceq$

to denote the subsequence relation, i.e., $\mathcal{S}' \preceq \mathcal{S}$ where $\mathcal{S}' = (\mathcal{F}_i, \ldots, \mathcal{F}_j)$ is a subsequence of $\mathcal{S}$. For readability, we henceforth refer to a subsequence $\mathcal{S}'$ of $\mathcal{S}$ as a range of frames using the shorthand notation $\mathcal{F}_{i,j}$ where $i \leq j$. For example, an instance of the subsequence of frames $\mathcal{F}_{0,2}$ from some perception stream $\mathcal{S}$ is provided in Fig. 1.

Each frame $\mathcal{F}_i$ in the perception stream is considered a *key frame* that contains data generated from one or more sensor channels $\mathcal{C}$. A key frame is a frame where the timestamp difference between each sample from a sensor in $\mathcal{C}$ is within some threshold $\epsilon_t \in \mathbb{R}$. For example, a frame of an AV affixed with a front, front-left, front-right, and rear camera sensor channels is a key frame if and only if the timestamp difference between all samples occur within a period of 0.001s. Furthermore, each frame contains a finite set of object annotations (henceforth, "objects") O from the entire stream $\mathcal{S}$ of objects $\mathcal{O}$ consisting of query-able information. That is, each object $o \in \mathcal{O}$ may be annotated with a label (e.g., *car*, *pedestrian*, *sign*, *bus*, *animal*, etc.), a unique identifier ("ID"), 2D/3D bounding box, confidence score, segmentation map, LiDAR points, etc.

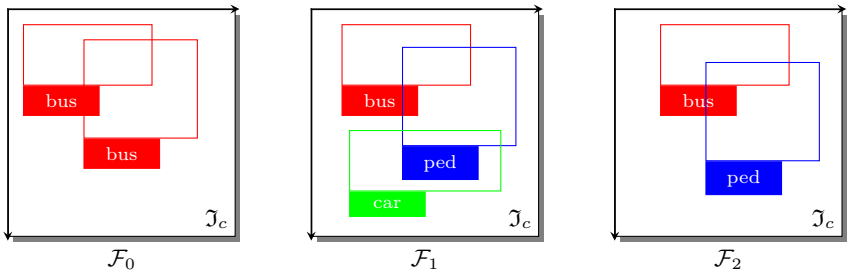

Fig. 1. An example perception stream $\mathcal{S}$ containing the frames $\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2$ of a camera sensor channel c with pixel space $\mathfrak{I}_c$. For each object in a given frame, a classification and bounding box is minimally assumed to be annotated. In addition, each frame may be augmented with other sensor data relevant to the system to provide further context such as GPS, IMU, etc.

In the following, we assume that given an object $o \in \mathcal{O}$, we have defined functions that return the required annotation and/or auxiliary data. For example, for retrieving qualitative attributes, we define the function $\mathfrak{A} : \mathcal{O} \times \mathcal{K} \to \mathcal{A}$ where $\mathcal{K}$ is a set of keys and $\mathcal{A}$ is a set of attributes. We will not formally define the sets $\mathcal{K}$ and $\mathcal{A}$ since they are dataset dependent, but as an example, we could have `class` and `color` as keys, and `bus` and `green` as the corresponding attributes. When formulating a query, we are searching to find in a frame an object that satisfies certain attributes, e.g., a *green bus*. In addition, for retrieving the 2D (axis-aligned) bounding box information of objects, we define the function $\mathfrak{B}_c : \mathcal{O} \to \mathcal{P}(\mathfrak{I}_c)$ where $\mathfrak{I}_c \subseteq \mathbb{N}^2$ is the set of pixels for a camera sensor channel $c \in \mathcal{C}$; and the function $\mathfrak{B} :\to \mathcal{P}(\mathcal{E})$ returns the 3D rectangular bounding box of any object $o \in \mathcal{O}$ in the working environment $\mathcal{E} \subseteq \mathbb{R}^3$ of a perception

Table 1. An example of annotated data from a perception stream. Each object has some attributes along with some bounding box ("BB") data. Depending on the application, the object IDs may be unique across the stream, unique only in each frame, or unique to each frame for each class of objects. In addition, each object may be annotated with additional data such as LiDAR points, color (if applicable), etc.

$\mathcal{F}_0$	$\mathcal{F}_1$	$\mathcal{F}_2$
(bus, red, ID: 1, BB)	(bus, red, ID: 1, BB)	(bus, red, ID: 1, BB)
(bus, yellow, ID: 2, BB)	(pedestrian, child, ID: 2, BB)	(pedestrian, child, ID: 2, BB)
	(car, sedan, ID: 3, BB)	

system. Table 1 presents a detailed layout of Fig. 1 with annotated information for each object.

3 Spatial Regular Expressions

SpRE (pronounced /spri:/) is a querying language designed to capture scenarios of perception streams. The RE $\times$ $\mathcal{S}4_u$ language leverages the power of pattern matching from REs [1] with the topological reasoning of $\mathcal{S}4_u$ logic [25]. The practice of merging a formal logic with an RE-based language to produce an extended, more expressive version has previously been studied in [38,7]. However, these efforts primarily focus on extending temporal-based logics such as Linear Temporal Logic (LTL) [31] with REs, whereas current extensions to spatial-based logics with REs is unheard of to the best of our knowledge—particularly, the $\mathcal{S}4_u$ branch of logic. Thus, the SpRE language aims to provide a formal approach in extending pattern-based constructs with spatial-based formulas.

Within this section, we first introduce the formal syntax of the SpRE language followed by its semantics interpreted over perception streams.

Syntax. The SpRE syntax consists of three interdependent grammars joined to form the complete querying language. The first two grammars (Defs. 1 and 2) are inspired by the $\mathcal{S}4$ and $\mathcal{S}4_u$ modal logics for topological spaces, respectively; and the last grammar (Def. 3) is inspired by classic REs traditionally used in *string matching* problems.

Spatial Formulas. The syntactic makeup of the spatial logic component of a SpRE is divided into two grammars: (1) $\mathcal{S}4^+$ ("spatial terms") inspired by $\mathcal{S}4$ [25], and (2) $\mathcal{S}4_u^+$ ("spatial formulas") inspired by $\mathcal{S}4_u$ [25]. Although the two logics ($\mathcal{S}4^+$, $\mathcal{S}4_u^+$) use an identical syntax to their counterparts ($\mathcal{S}4$, $\mathcal{S}4_u$), we re-introduce them here as new semantics for these logics will be defined later.

The spatial terms enable set operations over set based attributes of objects such as bounding boxes. The motivation behind spatial terms is to enable representation of the topological relations between different objects, e.g., the intersection of the bounding boxes of a *green bus* and a *yellow car*.

Definition 1 ($\mathcal{S}4^+$ Syntax). *The structure of an $\mathcal{S}4^+$ formula τ is inductively defined by the following grammar:*

$$\tau ::= \alpha \mid \bar{\tau} \mid \tau_1 \sqcap \tau_2 \mid \tau_1 \sqcup \tau_2 \mid \mathbf{I}\,\tau \mid \mathbf{C}\,\tau$$

where α is an atomic proposition ranging over sets of attributes $\mathcal{P}(\mathcal{A})$; $^-$ is the unary operator for complement; $\sqcap$ and $\sqcup$ are the binary operators for intersection and union; and $\mathbf{I}$ and $\mathbf{C}$ are the interior and closure operators.

Spatial formulas extend spatial terms by enabling emptiness checks and subset relations. For example, we will be able to answer a query which requires an non-empty intersection between the *green bus* and the *yellow car*.

Definition 2 ($\mathcal{S}4_u^+$ Syntax). *Given the spatial terms τ, τ_1, and τ_2, the structure of an $\mathcal{S}4_u^+$ formula ϕ is inductively defined by the following grammar:*

$$\phi ::= \alpha \mid \boxed{\exists}\,\tau \mid \tau_1 \sqsubseteq \tau_2 \mid \neg\,\phi \mid \phi \wedge \phi \mid \phi \vee \phi$$

where α is an atomic proposition ranging over sets of attributes $\mathcal{P}(\mathcal{A})$; $\boxed{\exists}$ and $\sqsubseteq$ are the boolean operators for set emptiness and set inclusion; and $\neg$, $\wedge$, and $\vee$ are the standard propositional logic operators.

Pattern Constructs. The syntax of a SpRE combines the RE elements with the previously defined $\mathcal{S}4_u^+$ elements to form a spatial-capable RE. Apart from the classic RE operators, SpRE patterns operate over symbols from the alphabet Σ that are resolved through spatial formulas.

Remark 1 (Alphabet). We consider the alphabet $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_n\} = \mathcal{P}(\mathcal{O})$ where each symbol represents a possible combination of the objects from the perception stream $\mathcal{S}$. The main intuition is that for every frame of the stream of perception data, we would like to match a symbol with only the relevant objects for the given SpRE query—see Ex. 1 below.

Example 1. Consider the data presented in Table 1. The alphabet Σ will contain 64 symbols in total. Some examples of symbols from Σ could be:

$\sigma_1 = \{$(bus, red, ID: 1, BB)$\}$
$\sigma_2 = \{$(pedestrian, child, ID: 2, BB), (bus, yellow, ID: 2, BB)$\}$
$\sigma_3 = \{$(bus, red, ID: 1, BB), (car, sedan, ID: 3, BB)$\}$

When we query for *bus*, we would like our pattern matching algorithm to return σ_1 in the matching strings, but not σ_2 or σ_3.

Definition 3 (SpRE Syntax). *Given the spatial formula ϕ, the structure of a SpRE query is inductively defined by the following grammar:*

$$Q ::= \phi \mid Q_1 \mid Q_2 \mid Q_1 \cdot Q_2 \mid Q^*$$

where the operators $\mid$, $\cdot$, and * are the standard RE operations alternation, concatenation, and Kleene-star, respectively.

Semantics. Pattern matching on perception streams differs from the standard string pattern matching. When querying perception data, the goal is to identify annotations represented as abstract objects affixed with some attributes. For example, a query could be "*Find all sequences where a car appears in at least three consecutive frames*". In such a query, we do not ask for a specific car with a unique identity (which is not known in advance), but rather for any car. In addition, there may be multiple cars in a frame which implies that all of them should be candidates for a pattern match. In other words, even though our patterns in SpREs are over object attributes, our queries should return strings where each symbol is a set of corresponding specific objects.

We define the semantics of $\mathcal{S}4^+$ expressions through a valuation function $[\![\,]\!] : \mathcal{P}(\mathcal{O}) \to \mathcal{P}(\mathcal{P}(W))$ where W is the spatial reasoning space that resolves to either the camera sensor's image space $\mathfrak{I}_c$ (i.e., pixels) or the working environment $\mathcal{E}$, $\mathcal{P}(W)$ is all possible bounding boxes from the reasoning space, and $\mathcal{P}(\mathcal{P}(W))$ is the set of all possible sets of all bounding boxes. The spatial terms of $\mathcal{S}4^+$ specify set-theoretic operations over bounding boxes of objects from the perception stream $\mathcal{S}$. We define the semantics of $\mathcal{S}4^+_u$ using a boolean satisfaction relation since our goal is to determine whether certain relations are true or not over bounding boxes and other object annotations.

Definition 4 ($\mathcal{S}4^+$ Semantics). *Given a set of objects $O \subseteq \mathcal{O}$, the semantics of an $\mathcal{S}4^+$ formula is inductively defined as follows:*

$$
\begin{aligned}
[\![\alpha]\!](O) &= \{\mathfrak{B}(o) \mid o \in O \;.\; \forall a \in \alpha \;.\; \exists k \in \mathcal{K} \;.\; \mathfrak{A}(o,k) = a\} \\
[\![\bar{\tau}]\!](O) &= \{\bar{S} \mid S \in [\![\tau]\!](O)\} \\
[\![\tau_1 \sqcap \tau_2]\!](O) &= \{S_1 \cap S_2 \mid S_i \in [\![\tau_i]\!](O)\}
\end{aligned}
$$

Informally, given a set of objects O from a frame, the valuation of the spatial term α is the set of bounding boxes of all the objects which satisfy all the attributes in α.

Definition 5 ($\mathcal{S}4^+_u$ Semantics). *Given a set of objects $O \subseteq \mathcal{O}$, the semantics of an $\mathcal{S}4^+_u$ formula is inductively defined as follows:*

$$
\begin{aligned}
O \vDash \alpha \quad &\text{iff} \quad \exists o \in O \;.\; \forall a \in \alpha \;.\; \exists k \in \mathcal{K} \;.\; \mathfrak{A}(o,k) = a \\
O \vDash \neg\, \phi \quad &\text{iff} \quad O \nvDash \phi \\
O \vDash \phi_1 \wedge \phi_2 \quad &\text{iff} \quad O \vDash \phi_1 \text{ and } O \vDash \phi_2 \\
O \vDash \boxed{\exists}\, \tau \quad &\text{iff} \quad \exists A \in [\![\tau]\!](O).\; A \neq \emptyset \\
O \vDash \tau_1 \sqsubseteq \tau_2 \quad &\text{iff} \quad \exists A_1 \in [\![\tau_1]\!](O).\; \exists A_2 \in [\![\tau_2]\!](O).\; A_1 \subseteq A_2
\end{aligned}
$$

Notice that the models (i.e., sets of objects) that satisfy a spatial formula ϕ are not minimal. For instance, using the symbols from Ex. 1, we have $\sigma_1 \vDash \mathit{bus}$, but also $\sigma_2 \vDash \mathit{bus}$ and $\sigma_3 \vDash \mathit{bus}$. In the *pattern matching* problem, we typically care more so about the sequence of frames $\mathcal{F}_{i,j}$ that satisfy the pattern Q rather than which exact objects are part of the pattern.

Definition 6 (SpRE Semantics). *Given the alphabet Σ, the language described by a SpRE query Q is inductively defined as follows:*

$$\begin{aligned}\mathcal{L}(\phi) &= \{\sigma \in \Sigma \mid \sigma \vDash \phi\} \\ \mathcal{L}(Q_1 \mid Q_2) &= \mathcal{L}(Q_1) \cup \mathcal{L}(Q_2) \\ \mathcal{L}(Q_1 \cdot Q_2) &= \mathcal{L}(Q_1)\mathcal{L}(Q_2) \\ \mathcal{L}(Q^*) &= \bigcup_{i=0}^{\infty} \mathcal{L}(Q^i)\end{aligned}$$

where Q^i denotes the concatenation of pattern Q a total of i times.

One notable difference from the standard language definition for a RE is that now our base case, i.e., the spatial formulas ϕ evaluate to sets of symbols as demonstrated in Ex. 1. This reflects the observation that at each frame we may have several matching objects for our query.

4 Perception Stream Matching

In this section, we provide a formulation of the problem of pattern matching against perception streams in both the offline and online domain. Furthermore, we introduce the STREM tool as our matching framework that implements the semantics of SpREs introduced in Sect. 3 to search over perception streams.

4.1 Problem Formulation

Informally, the traditional problem of pattern matching considers a finite word w and a pattern p from some finite alphabet Σ such that the goal is to find all non-overlapping subsets of w that contain an exact match of p. From Boyer-Moore (BM) [8] to Knuth-Morris-Pratt (KMP) [24], many algorithms have been developed to solve this problem of searching through strings [3]. In this work, we extend upon this idea with the modification that our search pattern p is symbolically a SpRE query Q, and our word w is a perception stream $\mathcal{S}$. We consider this problem in the both offline and online domain.

Offline Matching. We consider pattern matching in the offline domain primarily motivated by the presence of publicly available AV-based perception datasets [34,9,22,30,42]. These datasets provide a large suite of perception data collected for and used by ADS applications. However, to our knowledge, the capabilities and frameworks to search through these datasets for said applications are not well-supported or require a significant effort to do so. The offline pattern matching problem for perception streams is formalized below in Prob. 1.

Problem 1 (Offline Perception Stream Matching). Given a *finite* perception stream $\mathcal{S}$ and a SpRE query Q, then starting from frame 0, find the set of all non-overlapping leftmost longest frame subsequences $\{\mathcal{F}_{i_1,j_1}, \ldots, \mathcal{F}_{i_n,j_n}\}$ in $\mathcal{S}$ such that $i_k \leq j_k$, $j_k \leq i_{k+1}$ and $j_n \leq |\mathcal{S}|$ and $\mathcal{F}_{i_k,j_k} \in \mathcal{L}(Q)$ for all $k \leq n$.

Online Matching. We also consider pattern matching in the online domain to perform filtering and querying of perception streams generated in realtime. Applications of such use cases include monitoring of AVs deployed, CCTV camera alerts, and any perception-based systems generating data where detection of scenarios in realtime are of importance.

Regarding the procedure of matching online, the framework is re-run at every time instance l when a new frame $\mathcal{F}_l$ is received and returns the maximal query matched up to that point $\mathcal{F}_{i_k,j_k}$ with $j_k = l$ (or none if no match). The online pattern matching problem for perception streams is formalized below in Prob. 2.

Problem 2 (Online Perception Stream Matching). Given a perception stream $\mathcal{S}$ and a SpRE query Q, then at every incoming frame $\mathcal{F}_j$ of $\mathcal{S}$, find the longest subsequence of frames $\mathcal{F}_{i,j}$ such that $0 \leq i \leq j$ and $\mathcal{F}_{i,j} \in \mathcal{L}(Q)$.

4.2 Spatio-Temporal Regular Expression Matcher

Our SpRE matching framework follows the same principles as the classic RE matching frameworks [2]. Standard string matching approaches translate an RE to a Deterministic Finite Automata (DFA) D which is then used to process the strings. Our framework deviates from the established approaches using DFAs since each frame contains multiple objects which may satisfy different $\mathcal{S}4_u^+$ formulas and all potential matches need to be tracked simultaneously.

Example 2. Consider the data stream in Table 1 and assume that we only care about the classes and properties, e.g., we want to find two frames where a *red bus* appears. If we treat each object in each frame as a symbol of the form `(class, property)`, then the data stream represents $2 \times 3 \times 2 = 12$ strings. Two example strings from Table 1 are `(bus,red)(bus,red)(bus,red)` and `(bus,red)(car, sedan)(bus,red)`. As the length of the data stream increases, the number of strings that we need to consider increases exponentially in the worst case.

Especially in the case of online query matching, an approach that extracts strings from a perception stream to match against a DFA quickly becomes unmanageable. In this work, we take a more pragmatic approach which in practice works well. We treat each syntactically equivalent $\mathcal{S}4_u^+$ formula as a unique symbol and translate the SpRE into an RE. Even though we can now use the standard RE to DFA algorithms, the resulting automaton in execution becomes nondeterministic. This process can be easily visualized through Ex. 3 below.

Example 3. In the following, we use the convention that an atomic proposition (i.e., a set of attributes) is represented as an augmented *character class* familiar to `grep`. For readability, $\mathcal{S}4_u^+$ formulas are also surrounded by brackets. Using this notation, the formula `[<nonempty>([:car:]&[:ped:])]` is an $\mathcal{S}4_u^+$ formula that is only true when a frame contains a *car* and a *pedestrian* with interesecting bounding boxes. Since the operator `<nonempty>` applies only to spatial terms, we know that `[:car:]&[:ped:]` is an $\mathcal{S}4^+$ subformula where `&` is the operator for set intersection.

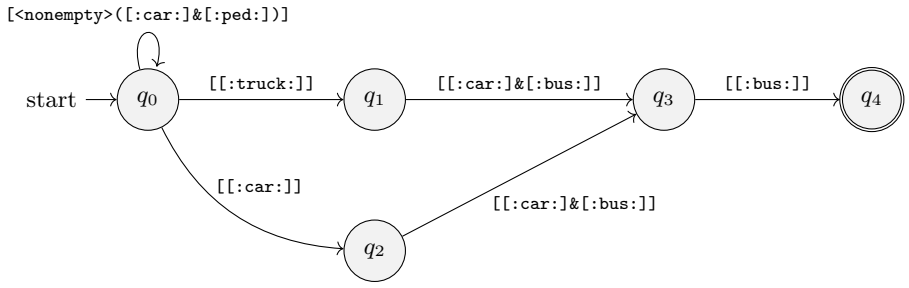

Fig. 2. SpRE to DFA.

From the previously introduced notation, we provide the following SpRE pattern written below

```
[<nonempty>([:car:]&[:ped:])]*([[:truck:]]|[[:car:]])
[[:car:]&[:bus:]][[:bus:]]
```

that matches zero or more (*) frames where a *car* and *pedestrian* overlap, followed by (·) a frame with either (|) a *truck* or a *car*, followed by (·) one frame that contains a *car* and a *bus*, and ending with (·) a frame with a *bus*.

The resulting automaton that accepts perception streams that match the SpRE above is presented in Fig. 2. Notice that in the resulting automaton, the transitions between states are labeled by $\mathcal{S}4_u^+$ formulas and, hence, the execution semantics is that of a Nondeterministic Finite Automata (NFA). That is, in each state, multiple transitions may be activated. For example, in state q_0 if the current frame contains a *truck* and a *car*, then both transitions to q_1 and to q_2 are activated. In principle, tracking multiple states for an NFA execution scales better than constructing single-symbol strings from a stream for tracking with a DFA. In the worst case, the total number of states of the DFA that we need to keep track off is order of magnitudes smaller than the number of all possible single symbol strings that we need to consider.

Software Tool. STREM is a CLI tool[1] developed with *Rust* [28] to find scenarios of interest in perception streams that match a given SpRE query. It functions in both the offline and online domain to search over perception-based datasets or realtime streams, respectively. An illustration of its core components is provided in Fig. 3. As input, the tool accepts a SpRE query and a perception stream. As output, it incrementally returns the set of matches where each match is a range of frames from the provided perception stream that matched the pattern. The five constituent components of the tool are grouped into two functionalities: the *frontend* and the *backend*. The frontend handles all input-/output-related activities pertinent to the usability of the tool; and the backend is concerned only with the core matching framework and procedures. Henceforth, we focus

[1] https://crates.io/crates/strem

on the backend components that support the main contributions of this work: the *Compiler*, *Matcher*, and *Monitor* modules.

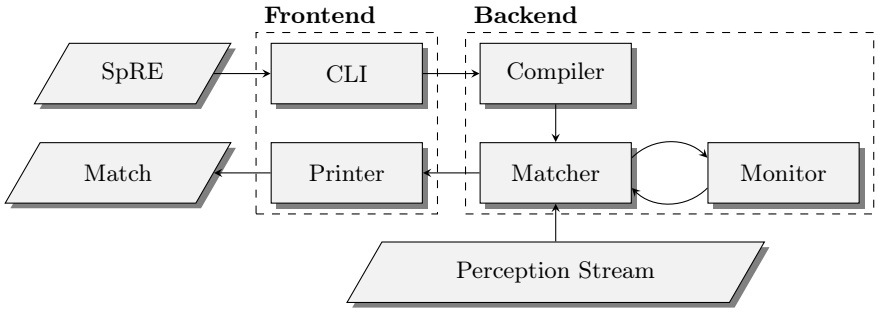

Fig. 3. The architectural design of STREM.

The *Compiler* is responsible for translating a SpRE into a symbolic-Abstract Syntax Tree (s-AST)—an Intermediate Representation (IR) form interpretable by the *Monitor* and *Matcher* modules. The *Monitor* is responsible for evaluating $\mathcal{S}4_u^+$ formulas against perception stream frames. The *Matcher* is responsible for constructing DFAs from s-ASTs and running the matching algorithms.

Spatial Matching Algorithms. The pattern matching procedure involves both the *Matcher* and the *Monitor*—as depicted in Fig. 3. The *Matcher* receives from the *Monitor* which $\mathcal{S}4_u^+$ formulas were satisfied and, then, it takes the appropriate transitions to the next states. Recall that in our framework multiple transitions on the DFA may become active. Therefore, the execution semantics of the DFA in the *Matcher* are effectively the execution semantics of an NFA. Nevertheless, our constructions are syntactically DFA and, in the following, we will still refer to them as DFA.

The algorithm to match against a perception stream given some DFA that recognizes a valid SpRE query is shown in Alg. 1. This algorithm is generalized for both offline and online applications and any differences in the assumptions and procedures are highlighted in the sections that follow.

Offline Algorithm. The offline matching procedure matches over a finite perception stream from frame $\mathcal{F}_0$ up to frame $\mathcal{F}_l$ by utilizing a *forward* DFA. Notably, the offline variant assumes that all frames within $\mathcal{S}$ are present at the beginning of the execution of the matching algorithm.

Online Algorithm. The online matching procedure matches over a perception stream by utilizing a *reverse* DFA. For each new frame received, the online algorithm variant is ran. We use a reverse DFA in the online problem as matching backwards (i.e., from frame $\mathcal{F}_l$ down to frame $\mathcal{F}_0$) ensures that the matching

procedure terminates (in the worst case at frame $\mathcal{F}_0$). However, in practice, it is recommended that the termination of the match be triggered by some finite horizon (i.e., maximum length) for which the SpRE query will match up to. For certain queries, we can compute the finite length needed to determine if a match is possible. The length of an online SpRE query can be computed as follows:

Definition 7 (SpRE Horizon). *The horizon $\mathcal{H}$ of a SpRE query Q is inductively defined as follows:*

$$\begin{aligned} \mathcal{H}(\phi) &= 1 & \mathcal{H}(Q_1 \mid Q_2) &= \max(\mathcal{H}(Q_2), \mathcal{H}(Q_2)) \\ \mathcal{H}(Q^*) &= \infty & \mathcal{H}(Q_1 \cdot Q_2) &= \mathcal{H}(Q_1) + \mathcal{H}(Q_2) \end{aligned}$$

where ϕ is a spatial formula.

When $\mathcal{H}(Q)$ is finite, i.e., there is no Kleene-star operator, then we know that the online algorithm will only need to use up to $\mathcal{H}(Q)$ frames in the past. As an alternative to the Kleene-star operator, in our implementation, we provide the *range* operator to capture bounded-ness (see Sec. 5). If a Kleene-star operator must be used, then a hard bound on the maximum length should be used to keep the monitoring time predictable, in the worst case.

Algorithm 1: SpatialMatching
This algorithm represents the offline variant. For the online variant, replace each line with its corresponding comment to the right.

Input: A perception stream $\mathcal{S}$, an initial frame index $i \in \mathcal{S}$.
Output: A range $[start, end)$ corresponding to the indices from $\mathcal{S}$.
Data: A set A of distinct active states from the DFA.

```
1  start ← i;                                   // end ← |S|
2  end ← start;                                 // start ← end − 1
3  foreach F ∈ S do                             // S = (S)
4  |  foreach (σ_s, φ) do
5  |  |  if F ⊨ φ then
6  |  |  |  symbols.push(σ_s)
7  |  |  end
8  |  end
9  |  foreach σ_s ∈ symbols do
10 |  |  A.insert(δ(σ_s))
11 |  end
12 |  if A contains accepting then
13 |  |  end ← F.index ;                        // start ← F.index
14 |  else if A all dead then
15 |  |  break
16 |  end
17 end
18 return (start, end)
```

Complexity. The time complexity depends on the data stream $\mathcal{S}$ and the query Q. Let $|\mathcal{S}|$ be the total number of frames in the perception stream, and $|O_i|$ be the number of objects in the i^{th} frame where O_i is the set of objects in the frame $\mathcal{F}_i \in \mathcal{S}$. The query Q is translated into a DFA D with set of states D_S and transition relation D_Δ with $|D_S|$ denoting the number of states, and $|D_\Delta|$ denoting the total number of transitions, respectively. Recall that the transitions in D are labeled by spatial formulas from Q. That is, the transitions have the form $(s, \phi_k, s') \in D_\Delta$. We denote by $|\phi_k|$ the size of the parse tree (number of nodes) of ϕ_k since evaluating ϕ_k will require traversing its parse tree.

We first evaluate the time complexity of the *Monitor*, followed by the *Matcher*, followed by the combination of the two. For the *Monitor* to evaluate a spatial formula ϕ_k against a frame, if the $\mathcal{S}4^+_u$ formula ϕ_k contains no spatial operations, then its evaluation takes linear time in the tree traversal of ϕ_k (number of internal nodes $(|\phi_k| - 1)/2$) and linear time in the number of objects $|O_i|$ in a frame for each leaf (number of leaves $(|\phi_k| + 1)/2$) in order to find objects with specific attributes. Thus, the complexity of running the monitor is $\mathcal{O}(|\phi_k| \times |O_i|)$. As a special case, if Q only contains queries about class labels, then we can use a hash table storing whether an object of some class appears in a frame or not, giving us $\mathcal{O}(1)$ evaluation of the leaves of ϕ_k.

The *Matcher* keeps track of the active states in the DFA and, for each state, checks all the formulas in the outgoing transitions by calling the *Monitor*. In the worst case, $|D_S|$ states will be active, which implies that all the transitions in D_Δ must be checked. Thus, there will be $\mathcal{O}(|D_\Delta|)$ calls to the *Monitor*. Since the *Matcher* will be called $|\mathcal{S}|$ times, the complexity of the offline algorithm is

$$\mathcal{O}\big(|\mathcal{S}| \times |D_\Delta| \times \max_k(|\phi_k|) \times \max_i(|O_i|)\big)$$

whereas the online algorithm pays this cost for every new frame that appears within the perception stream.

If spatial operations are present in the spatial formulas, then the worst-case time complexity increases. Spatial terms in Def. 4 evaluate to collections of bounding sets in 2D or 3D. Therefore, the leaf nodes of ϕ_k represent collections of sets, and the internal nodes apply set operations such as union, intersection, complementation, and set difference. The computational cost of the set operations depends on the set representation (e.g., orthogonal polyhedra, vertex representation, polytopes, zonotopes, etc). Here, we will not consider the representation of the sets explicitly, and refer the reader to [19].

Remark 2 (Best-Case Scenario). Our querying problem can be reduced to standard regular expression matching when: (1) all $\mathcal{S}4^+_u$ formulas are strictly atomic propositions, (2) the number of object attributes to search over are few in numbers, and (3) the attributes are not quantitative (e.g., no bounding box).

5 Examples and Benchmarks

To demonstrate the application of STREM, we provide two use cases of the tool: (A) an offline example of searching through the Woven Planet ("L5") Perception

dataset [22] and (B) an online example of monitoring an AV's perception system through the CARLA simulator [12] with ROS [32]. Furthermore, we provide an initial set of performance benchmarks of the tool. For all queries, we use the STREM implementation-level syntax equivalents in Table 2.

Table 2. SpRE implementation equivalencies

Notation	$\mid$	$\cdot$	*	$\boxed{\exists}$	$\neg$	$\wedge$	$\vee$	$\sqcap$	$\sqcup$
Symbol	`|`		`*`	`<nonempty>`	`~`	`&`	`|`	`&`	`|`

Furthermore, the range meta-operator ($\{m, n\}$) is used to support constraint concatenations. The operational equivalence is shown below:

$$Q\{m,n\} \equiv \overbrace{Q \cdot \ldots \cdot Q}^{m \text{ concatenations}} \mid \overbrace{Q \cdot \ldots \cdot Q}^{m+1 \text{ concatenations}} \mid \ldots \mid \overbrace{Q \cdot \ldots \cdot Q}^{n \text{ concatenations}}$$

where $0 \leq m \leq n$. In addition, the range operator support two other functions: (1) $Q\{\mathrm{m}\}$ matches Q exactly m times; and (2) $Q\{\mathrm{m},\} = Q\{\mathrm{m},\infty\}$ matches Q m or more times.

5.1 Example A: Offline Matching Examples

We demonstrate the offline searching capabilities of the STREM tool on the Woven Planet ("L5") Perception dataset: a collection of sensor and ground-truth labels used in training and evaluation of AV perception systems. The dataset is comprised of 10 sensor channels, 360 scenes, 9 object classifications, over 300K frames, and over 1.2M object annotations yielding slightly over 186 GBs of data to search from.

Example A.1. In ADS applications, it is important to distinguish between cyclists and pedestrians as the intent and behavior of both differ. Therefore, to improve the resilience of a perception system against mis-identification, filtering for scenarios where the two classifications overlap (i.e., potential cases of ambiguity) strengthens Deep Neural Networks (DNNs) on such edge cases.

Query 1. Find all longest sequences of frames where a detected *pedestrian* overlaps with a detected *cyclist*.

```
[<nonempty>([:pedestrian:]&[:bicycle:])]*
```

where the SpRE matches zero or more frames (*) where the intersection of a *pedestrian* and *bicycle* bounding box is non-empty.

From the results, a total of 62 unique matches were found where each match contains a sequence of frames with a pedestrian and cyclist overlapping.

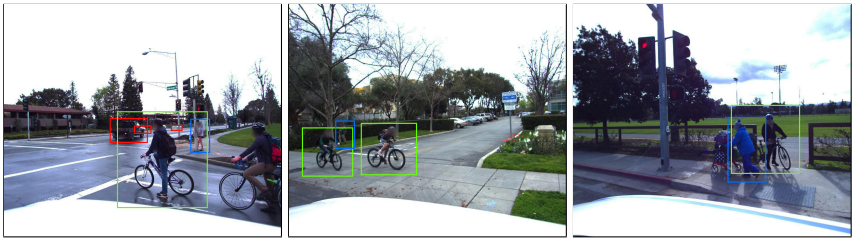

Fig. 4. A selection of three separately matching frames from the Woven Planet ("L5") Perception dataset where a *pedestrian* (blue) overlaps with a *cyclist* (green).

Example A.2. While queries targeting individual scenarios are useful for simple matching, more complex queries are needed to capture scenarios that can not be consolidated to a single frame and represent an evolution of events. For instance, consider the scenario where a pedestrian is initially occluded by a vehicle, unobstructed, and then occluded again by a vehicle.

Query 2. Find a sequence of frames where a *pedestrian* and *car* occlusion occurs for one or more frames, followed by an unobstructed *pedestrian* for one or more frames, followed by an occlusion of a *pedestrian* and a *car* for one or more frames.

```
[<nonempty>([:pedestrian:]&[:car:])]{1,}
[[:pedestrian:]&~<nonempty>([:pedestrian:]&[:car:])]{1,}
[<nonempty>([:pedestrian:]&[:car:])]{1,}
```

where the SpRE matches a sequence of scenarios (i.e., sub-scenario) where each sub-scenario must be at least one frame ({1, }) long. The first sub-scenario matches the intersection of a *pedestrian* and *car* bounding box is non-empty. The second sub-scenario matches the case where a *pedestrian* exists and the intersection of a *pedestrian* with a *car* is empty. The last sub-scenario matches the same as the first sub-scenario.

F: 009 F: 013 F: 034

Fig. 5. The results of running Query 2 through STREM on the Woven Planet ("L5") Perception dataset. From left to right, the matching frames include an instance of a *car* (red) occluding a *pedestrian* (blue), a *pedestrian*, and a *car* occluding a *pedestrian*.

From the results, a total of 336 uniques matches of three or more frames were found that matched the evolution of scenarios as described in the SpRE query.

5.2 Example B: Online Matching

To demonstrate the online searching capabilities of STREM, we developed a ROS package that bridges the CARLA simulator with the STREM tool by using the standard *topics* infrastructure provided by ROS. This design allows additional ROS applications (e.g., robots, AVs, etc.) to easily integrate and subscribe to the match results published by the STREM tool.

Simulator Setup. For each example, the CARLA server was populated with 50 vehicles (e.g., trucks, sedans, etc.), 20 walkers (i.e., pedestrians), and a single *ego* vehicle affixed with a one front-facing camera sensor. From the set of labels provided by CARLA, we capture bounding box information for the following actor types: (1) traffic signs, (2) traffic lights, (3) vehicles, and (4) walkers.

For all examples, the experiments were run on a Linux workstation running Ubuntu 20.04.6 with an AMD Ryzen 7 5800X, an NVIDIA GeForce RTX 3070, and 16 GBs of RAM with CARLA v0.9.13 at 60Hz and ROS Noetic (Focal).

Example B.1. Within the deployment of AVs, monitoring the perception stream for critical scenarios is a runtime-centric activity that requires a continuous analysis of the results of the perception system in order to take decisive actions quickly. An example of such a critical scenario common to AVs is in the occlusion of people by other vehicles in the scene. Within this situation, limited information is available to the system and naturally additional caution should be taken. However, reporting this information is not an inherent responsibility of the perception system. As such, the STREM tool provides the capability to instantaneously report frames in realtime where a pedestrian is occluded by some other object detected within a scene to allow the ADS to take action.

In this example, we consider the scenario in CARLA where a bounding box of a pedestrian and a vehicle annotation overlap one another. The formalization of this pattern is presented in Query 3 below.

Query 3. Report every frame where a *pedestrian* and *vehicle* overlap.

```
[<nonempty>([:pedestrian:]&[:vehicle])]
```

where the SpRE matches a single frame such that the intersection of the bounding boxes of a *pedestrian* and a *vehicle* classification is non-empty.

A illustrative example of some frames reported by STREM during the simulation are showcased in Fig. 6.

Example B.2. Another critical scenario that a perception system may experience is in an eventual case that information becomes missing (i.e., the presence of an object disappears from sight).

In this example, we consider the scenario in CARLA where perceived traffic signs are detected followed by an occlusion of some sign within the frame.

F: 9354 F: 15407 F: 16869

Fig. 6. A series of matching frames with object detections within the CARLA simulator where a *pedestrian* and *vehicle* intersect as expressed in Query 3.

Query 4. Find a traffic *sign* within the last 200 frames that is eventually occluded by a *vehicle* or *pedestrian*.

```
[[:sign:]]{1,200}
[<nonempty>(([:vehicle:]|[:pedestrian:])&[:sign:])]
```

where the SpRE matches at least 1 and at most 200 frames ({1, 200}) initially that contain a *sign* annotation, and ends with one frame where the resulting intersection of the bounding box of a *sign* with the bounding box of either a *vehicle* or *pedestrian* is non-empty.

A illustrative example of some frames reported by STREM during the simulation are showcased in Fig. 7.

F: 22215 F: 22341 F: 22358

Fig. 7. A series of matching frames with object detections within the CARLA simulator where a vehicle (red) eventually occludes a traffic sign (pink) in the green circle.

5.3 Benchmarks

To evaluate the performance of the STREM tool, we ran several different queries against the Woven Planet ("L5") Perception dataset. For each query, the average running time of 10 samples of the matching algorithm was evaluated against 0 to 150K frames. The results are summarized in Fig. 8. The benchmarks were

ran on a Linux workstation running Fedora 37 (6.2.14-200.fc37.x86_64) with an AMD Ryzen 7 Pro 4750U processor with Radeon Graphics, 32 GBs of RAM, and a *wall clock* time of 30 seconds.

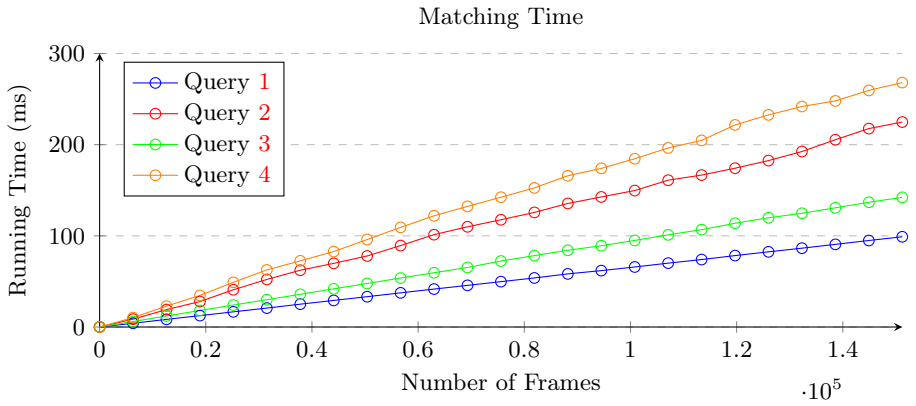

Fig. 8. Running time performance of STREM.

6 Related Work

The problem of querying video/multimedia datasets has a long history. Among the earliest works, [10] presents a spatio-temporal logic that can encode relationships among objects within image sequences. More recently, the Video Event Query Language (VEQL) was proposed in [41] (the paper also contains an exhaustive review of other video query languages). VEQL is a declarative language similar to SQL and it is used for monitoring of video data streams. It supports some ad hoc spatial and topological operators and some basic temporal relations through the Allen Interval Algebra (AIA) [4]. Besides the obvious differences of monitoring AIA (AIA can be encoded in LTL [33]) versus RE pattern matching, SpRE fully incorporates $\mathcal{S}4_u$ and it can foundationally support other modal logics of space (and time). Beyond queries, Timed Quality Temporal Logic (TQTL) was proposed in [11] to enable basic sanity checks over video feeds of automotive systems using object annotations. Furthermore, an online monitoring algorithm for TQTL was presented in [6].

Typically, perception data streams from automotive applications contain not only image sequences, but also data from a range of other sensing modalities, e.g., radar, lidar, infrared, etc. In [19], Spatio-Temporal Perception Logic (STPL) was introduced which combines TQTL [11] with an extension of the spatial-temporal logic PTL $\times$ $\mathcal{S}4_u$ [17] to support reasoning over spatial conditions such as intersection and distances between bounding boxes. In principle, temporal logics could be used for pattern matching after some modifications to their monitoring algorithms, but in practice, their syntax is not well suited for describing patterns. In another line of work, a querying method for sim-to-real applications

is presented in [23] which uses the Scenic probabilistic programming language [15]. Abstract static scenarios of interest are expressed in Scenic which are then queried over labeled datasets through a conversion into a Satisfiability Modulo Theory (SMT) problem. Even though one can envision that the method in [23] can eventually be extended to temporal queries, right now it is restricted to static scenes.

7 Conclusion and Future Work

In this paper, we proposed SpRE as a novel querying language for searching over perception streams using an RE $\times$ $\mathcal{S}4_u$ design. We demonstrated the application of SpREs in the offline and online domain through the development of the STREM tool alongside examples of matching over an AV dataset and the ROS and CARLA simulators, respectively. From this, we are able to find up to 20K+ matches in under 296 ms. As future work, we plan to include support for existential and universal operators in order to support a richer set of behaviors.

References

1. Aho, A.V.: Pattern matching in strings. In: Formal Language Theory, pp. 325–347. Elsevier (1980)
2. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: principles, techniques and tools. Pearson: Addison-Wesley (2020)
3. Alfred, V.: Algorithms for finding patterns in strings. Algorithms and Complexity **1**, 255 (2014)
4. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM **26**(11), 832–843 (1983)
5. Bai, Z., Nayak, S.P., Zhao, X., Wu, G., Barth, M.J., Qi, X., Liu, Y., Sisbot, E.A., Oguchi, K.: Cyber mobility mirror: A deep learning-based real-world object perception platform using roadside lidar. IEEE Transactions on Intelligent Transportation Systems (2023)
6. Balakrishnan, A., Deshmukh, J., Hoxha, B., Yamaguchi, T., Fainekos, G.: Percemon: online monitoring for perception systems. In: Runtime Verification: 21st International Conference, RV 2021, Virtual Event, October 11–14, 2021, Proceedings 21. pp. 297–308. Springer (2021)
7. Beer, I., Ben-David, S., Eisner, C., Fisman, D., Gringauze, A., Rodeh, Y.: The temporal logic sugar. In: Computer Aided Verification: 13th International Conference, CAV 2001 Paris, France, July 18–22, 2001 Proceedings 13. pp. 363–367. Springer (2001)
8. Boyer, R.S., Moore, J.S.: A fast string searching algorithm. Communications of the ACM **20**(10), 762–772 (1977)
9. Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Liong, V.E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O.: nuscenes: A multimodal dataset for autonomous driving. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 11621–11631 (2020)
10. Del Bimbo, A., Vicario, E., Zingoni, D.: Symbolic description and visual querying of image sequences using spatio-temporal logic. IEEE Transactions on Knowledge and Data Engineering **7**(4), 609–622 (1995)

11. Dokhanchi, A., Amor, H.B., Deshmukh, J.V., Fainekos, G.: Evaluating perception systems for autonomous vehicles using quality temporal logic. In: Runtime Verification (RV). LNCS, vol. 11237. Springer (2018)
12. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: Carla: An open urban driving simulator. In: Conference on robot learning. pp. 1–16. PMLR (2017)
13. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. International journal of computer vision **88**, 303–338 (2010)
14. Fang, W., Ding, L., Love, P.E., Luo, H., Li, H., Pena-Mora, F., Zhong, B., Zhou, C.: Computer vision applications in construction safety assurance. Automation in Construction **110**, 103013 (2020)
15. Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: a language for scenario specification and scene generation. In: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 63–78 (2019)
16. Friedl, J.E.: Mastering regular expressions. " O'Reilly Media, Inc." (2006)
17. Gabelaia, D., Kontchakov, R., Kurucz, A., Wolter, F., Zakharyaschev, M.: Combining spatial and temporal logics: expressiveness vs. complexity. Journal of artificial intelligence research **23**, 167–243 (2005)
18. Gallant, A.: regex-automata. https://github.com/rust-lang/regex (2023)
19. Hekmatnejad, M., Hoxha, B., Deshmukh, J.V., Yang, Y., Fainekos, G.: Formalizing and evaluating requirements of perception systems for automated vehicles using spatio-temporal perception logic. arXiv preprint arXiv:2206.14372 (2022)
20. Janai, J., Güney, F., Behl, A., Geiger, A., et al.: Computer vision for autonomous vehicles: Problems, datasets and state of the art. Foundations and Trends® in Computer Graphics and Vision **12**(1–3), 1–308 (2020)
21. Kapach, K., Barnea, E., Mairon, R., Edan, Y., Ben-Shahar, O.: Computer vision for fruit harvesting robots–state of the art and challenges ahead. International Journal of Computational Vision and Robotics **3**(1-2), 4–34 (2012)
22. Kesten, R., Usman, M., Houston, J., Pandya, T., Nadhamuni, K., Ferreira, A., Yuan, M., Low, B., Jain, A., Ondruska, P., Omari, S., Shah, S., Kulkarni, A., Kazakova, A., Tao, C., Platinsky, L., Jiang, W., Shet, V.: Woven planet perception dataset 2020. https://woven.toyota/en/perception-dataset (2019)
23. Kim, E., Shenoy, J., Junges, S., Fremont, D.J., Sangiovanni-Vincentelli, A., Seshia, S.A.: Querying labelled data with scenario programs for sim-to-real validation. In: 2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS). pp. 34–45. IEEE (2022)
24. Knuth, D.E., Morris, Jr, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM journal on computing **6**(2), 323–350 (1977)
25. Kontchakov, R., Kurucz, A., Wolter, F., Zakharyaschev, M.: Spatial logic+ temporal logic=? Handbook of spatial logics pp. 497–564 (2007)
26. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13. pp. 740–755. Springer (2014)
27. Lu, D., Eaton, E., Weg, M., Wang, W., Como, S., Wishart, J., Yu, H., Yang, Y.: Carom air–vehicle localization and traffic scene reconstruction from aerial videos. arXiv preprint arXiv:2306.00075 (2023)
28. Matsakis, N.D., Klock, F.S.: The rust language. ACM SIGAda Ada Letters **34**(3), 103–104 (2014)

29. Meng, T., Huang, J., Chew, C.M., Yang, D., Zhong, Z.: Configuration and design schemes of environmental sensing and vehicle computing systems for automated driving: A review. IEEE Sensors Journal (2023)
30. Pitropov, M., Garcia, D.E., Rebello, J., Smart, M., Wang, C., Czarnecki, K., Waslander, S.: Canadian adverse driving conditions dataset. The International Journal of Robotics Research **40**(4-5), 681–690 (2021)
31. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977). pp. 46–57. ieee (1977)
32. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y., et al.: Ros: an open-source robot operating system. In: ICRA workshop on open source software. p. 5. No. 3.2 in 3, Kobe, Japan (2009)
33. Roşu, G., Bensalem, S.: Allen linear (interval) temporal logic–translation to ltl and monitor synthesis. In: Computer Aided Verification: 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006. Proceedings 18. pp. 263–277. Springer (2006)
34. Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., et al.: Scalability in perception for autonomous driving: Waymo open dataset. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 2446–2454 (2020)
35. Thomas, G., Gade, R., Moeslund, T.B., Carr, P., Hilton, A.: Computer vision for sports: Current applications and research topics. Computer Vision and Image Understanding **159**, 3–18 (2017)
36. Turtiainen, H., Costin, A., Lahtinen, T., Sintonen, L., Hamalainen, T.: Towards large-scale, automated, accurate detection of cctv camera objects using computer vision. applications and implications for privacy, safety, and cybersecurity.(preprint). arXiv preprint arXiv:2006.03870 (2020)
37. Ward, T.M., Mascagni, P., Ban, Y., Rosman, G., Padoy, N., Meireles, O., Hashimoto, D.A.: Computer vision in surgery. Surgery **169**(5), 1253–1256 (2021)
38. Wolper, P.: Temporal logic can be more expressive. Information and control **56**(1-2), 72–99 (1983)
39. Xiao, P., Shao, Z., Hao, S., Zhang, Z., Chai, X., Jiao, J., Li, Z., Wu, J., Sun, K., Jiang, K., et al.: Pandaset: Advanced sensor suite dataset for autonomous driving. In: 2021 IEEE International Intelligent Transportation Systems Conference (ITSC). pp. 3095–3101. IEEE (2021)
40. Xu, Z., Julius, A.A.: Census signal temporal logic inference for multiagent group behavior analysis. IEEE Transactions on Automation Science and Engineering **15**(1), 264–277 (2016)
41. Yadav, P., Curry, E.: Vidcep: Complex event processing framework to detect spatiotemporal patterns in video streams. In: 2019 IEEE International conference on big data (big data). pp. 2513–2522. IEEE (2019)
42. Yu, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., Madhavan, V., Darrell, T.: Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 2636–2645 (2020)
43. Zhang, Y., Carballo, A., Yang, H., Takeda, K.: Perception and sensing for autonomous vehicles under adverse weather conditions: A survey. ISPRS Journal of Photogrammetry and Remote Sensing **196**, 146–177 (2023)