Trajectory Behavior Language

Thorsten Edelhäusser  
University of Erlangen-Nuremberg  
CS Dept., Progr. Systems Group  
Erlangen, Germany  
thorsten.edelhaeusser@cs.fau.de

Michael Philippsen  
University of Erlangen-Nuremberg  
CS Dept., Progr. Systems Group  
Erlangen, Germany  
philippsen@cs.fau.de

Christopher Mutschler  
Fraunhofer Institute for Integrated Circuits  
RF and Microwave Design Department  
Erlangen, Germany  
mutschcr@iis.fraunhofer.de

Abstract—The traditional representation of object trajectories as spatio-temporal data is not suitable for all applications. We present a trajectory representation called Trajectory Behavior Language (TBL) for three dimensional data that better fits trajectory comparison or analyzing the behavior of moving objects. Our representation is based on a Chomsky-2 formal language and can be generated online. We explain the grammar and present an algorithm for translating spatio-temporal data into the novel representation at runtime. This presented algorithm only needs one sample per time step and introduces no latency. Further, we present a sliding window based online TBL-to-TBL transformation that combines repeated elements of a trajectory, expresses them with additional grammar elements, and thus increases the readability and further compresses a trajectory. On a benchmark, we achieve a compression rate of 66.5%.

Index Terms—RTLS, trajectory description, trajectory compression, online translation

I. INTRODUCTION

Traditional location systems provide spatio-temporal data of located objects. This is well suited for visualizing scenes, e.g. for an operator directing bus drivers to free parking slots. Or for a monitoring system checking that objects are within certain boundaries and do not approach a dangerous area, for example a person approaching a pressing plant. But spatio-temporal representations reach their limits if more information on the object and its behavior is necessary. Think of a soccer match, where we want to automatically detect that a player performs a sprint or that the ball is kicked. From spatio-temporal data such events cannot be easily detected.

Spatio-temporal data is not amenable for matching of trajectories, since similarity detection requires rotating, displacing, and scaling. Also, a set of coordinates with added time stamps is not very intuitive. A more human-friendly approach would be a description language, e.g. as it is used to give directions to tourists. Such a description is also advantageous for recognition of movement patterns and unusual behavior.

The requirements to overcome the deficiencies of spatio-temporal representations are: A better representation to describe the path and the motion of an object must not just be intuitively readable and writable by humans, but also interpretable and analyzable by machines. The description must allow for a detailed reconstruction of the trajectory and for its (online) generation at runtime, i.e. without knowledge about the past and future of a sampling point. Additionally, this generation process should have a minimal latency, i.e. as little delay as possible when converting the object’s movement into the representation.

This paper presents such a representation: the Trajectory Behavior Language (TBL) is a formal language to describe object behavior in a human-readable and machine-useable way. A formal language opens up new possibilities, like matching trajectories, finding patterns, or analyzing movements. And state-of-the-art text processing algorithms can be applied. With TBL comes a latency-free online conversion algorithm for spatio-temporal data. Since the TBL can encode repeated motions, its representation is both easier to read and even more compact. From a TBL representation, the original trajectory can be reconstructed with no (or little) error.

After covering related work in the next section, Section III introduces the TBL representation of trajectories. Section IV then explains the conversion of spatio-temporal data into the formal language representation. Section V presents the TBL-to-TBL translation that detects and encodes repeated segments of a trajectory.

II. RELATED WORK

Current research on trajectory representation is mainly targeted towards querying databases, i.e. to search for similar trajectories or for time series with some specific characteristics. Pelekis et al. [8] introduce metrics for querying trajectories that not only find similar trajectories but also support trajectory clustering and mining tasks. Their distance operator “Locality In-between Polylines” calculates the crossing area of two trajectories. Operators for time-aware spatio-temporal trajectory similarity are also defined and are based on a classic spatio-temporal database. In comparison, it is less complex to find similarity among TBL trajectories, since string algorithms like the Levenshtein distance [7] can be used.

For Li et al. [4] a trajectory is a sequence of motions (displacements, moving directions, and time intervals). They use a moving spatio-temporal (mst) relationship which is a set of topological relations, directional relations, and a time interval for their comparison. This results in a better abstraction and readability of the spatio-temporal data, but from their motion sequence the trajectory cannot be reconstructed as with TBL.

Meratnia and de By [6] group trajectory regions with constant velocity or acceleration and separate groups by break points. Although their representation is compact, it does not allow direct online processing in Real-Time Location Systems.
(RTLSs), as it needs a set of several samples for grouping. TBL avoids such a latency.

Li et al. [5] characterize a motion by means of movement features (motifs). An object’s path is a sequence of motif expressions which are an abstraction of the underlying trajectory for a given time and location, e.g. “Right-Turn 3am, 17”. This is amenable for classification tasks. But in contrast to TBL, trajectories cannot be reconstructed in detail.

Similarly, Kojima et al. [3] use expressions that are inspired by a human language and that aim at explaining the object’s behavior, which is estimated by evaluating the pose and position in relation to surrounding objects. Since their main focus is to automatically generate text with true readability for humans, the result is unsuitable for a detailed and accurate trajectory analysis, that can be done with TBL.

Chen et al. [1] use a Movement Pattern String (MPS) to represent trajectories. Their symbols are selected by calculating the movement direction and movement distance ratio for every sampling point and by choosing the corresponding symbol from a quantization map. Similarity of trajectories relies on their Edit Distance on Real Sequence (EDR), a derivation of the Levenshtein distance. Since the distance ratio needs to know the whole trajectory, there is no online translation. Also the symbol string is less readable than our TBL representation.

There are at least two XML standards, describing trajectories: MPEG-7 (see [2]) and SVG. The first only defines trajectories as a spatio-temporal textual description, whereas the latter includes paths as a string of commands with absolute or relative displacements. In spite of our approach, a comparison of trajectories cannot be reconstructed in detail.

Chen et al. [1] use a Movement Pattern String (MPS) to represent trajectories. Their symbols are selected by calculating the movement direction and movement distance ratio for every sampling point and by choosing the corresponding symbol from a quantization map. Similarity of trajectories relies on their Edit Distance on Real Sequence (EDR), a derivation of the Levenshtein distance. Since the distance ratio needs to know the whole trajectory, there is no online translation. Also the symbol string is less readable than our TBL representation.

There are at least two XML standards, describing trajectories: MPEG-7 (see [2]) and SVG. The first only defines trajectories as a spatio-temporal textual description, whereas the latter includes paths as a string of commands with absolute or relative displacements. In spite of our approach, a comparison of trajectories cannot be reconstructed in detail.

li et al. [5] characterize a motion by means of movement features (motifs). An object’s path is a sequence of motif expressions which are an abstraction of the underlying trajectory for a given time and location, e.g. “Right-Turn 3am, 17”. This is amenable for classification tasks. But in contrast to TBL, trajectories cannot be reconstructed in detail.

Similarly, Kojima et al. [3] use expressions that are inspired by a human language and that aim at explaining the object’s behavior, which is estimated by evaluating the pose and position in relation to surrounding objects. Since their main focus is to automatically generate text with true readability for humans, the result is unsuitable for a detailed and accurate trajectory analysis, that can be done with TBL.

Chen et al. [1] use a Movement Pattern String (MPS) to represent trajectories. Their symbols are selected by calculating the movement direction and movement distance ratio for every sampling point and by choosing the corresponding symbol from a quantization map. Similarity of trajectories relies on their Edit Distance on Real Sequence (EDR), a derivation of the Levenshtein distance. Since the distance ratio needs to know the whole trajectory, there is no online translation. Also the symbol string is less readable than our TBL representation.

There are at least two XML standards, describing trajectories: MPEG-7 (see [2]) and SVG. The first only defines trajectories as a spatio-temporal textual description, whereas the latter includes paths as a string of commands with absolute or relative displacements. In spite of our approach, a comparison of trajectories cannot be reconstructed in detail.

Chen et al. [1] use a Movement Pattern String (MPS) to represent trajectories. Their symbols are selected by calculating the movement direction and movement distance ratio for every sampling point and by choosing the corresponding symbol from a quantization map. Similarity of trajectories relies on their Edit Distance on Real Sequence (EDR), a derivation of the Levenshtein distance. Since the distance ratio needs to know the whole trajectory, there is no online translation. Also the symbol string is less readable than our TBL representation.

There are at least two XML standards, describing trajectories: MPEG-7 (see [2]) and SVG. The first only defines trajectories as a spatio-temporal textual description, whereas the latter includes paths as a string of commands with absolute or relative displacements. In spite of our approach, a comparison of trajectories cannot be reconstructed in detail.

Chen et al. [1] use a Movement Pattern String (MPS) to represent trajectories. Their symbols are selected by calculating the movement direction and movement distance ratio for every sampling point and by choosing the corresponding symbol from a quantization map. Similarity of trajectories relies on their Edit Distance on Real Sequence (EDR), a derivation of the Levenshtein distance. Since the distance ratio needs to know the whole trajectory, there is no online translation. Also the symbol string is less readable than our TBL representation.

There are at least two XML standards, describing trajectories: MPEG-7 (see [2]) and SVG. The first only defines trajectories as a spatio-temporal textual description, whereas the latter includes paths as a string of commands with absolute or relative displacements. In spite of our approach, a comparison of trajectories cannot be reconstructed in detail.

Chen et al. [1] use a Movement Pattern String (MPS) to represent trajectories. Their symbols are selected by calculating the movement direction and movement distance ratio for every sampling point and by choosing the corresponding symbol from a quantization map. Similarity of trajectories relies on their Edit Distance on Real Sequence (EDR), a derivation of the Levenshtein distance. Since the distance ratio needs to know the whole trajectory, there is no online translation. Also the symbol string is less readable than our TBL representation.

There are at least two XML standards, describing trajectories: MPEG-7 (see [2]) and SVG. The first only defines trajectories as a spatio-temporal textual description, whereas the latter includes paths as a string of commands with absolute or relative displacements. In spite of our approach, a comparison of trajectories cannot be reconstructed in detail.

Chen et al. [1] use a Movement Pattern String (MPS) to represent trajectories. Their symbols are selected by calculating the movement direction and movement distance ratio for every sampling point and by choosing the corresponding symbol from a quantization map. Similarity of trajectories relies on their Edit Distance on Real Sequence (EDR), a derivation of the Levenshtein distance. Since the distance ratio needs to know the whole trajectory, there is no online translation. Also the symbol string is less readable than our TBL representation.

There are at least two XML standards, describing trajectories: MPEG-7 (see [2]) and SVG. The first only defines trajectories as a spatio-temporal textual description, whereas the latter includes paths as a string of commands with absolute or relative displacements. In spite of our approach, a comparison of trajectories cannot be reconstructed in detail.
on the right hand side which can be nonterminal or terminal symbols (Chomsky-2). We discuss TBL’s productions now.

1) General productions:
\[ S \rightarrow C | CS \]
\[ C \rightarrow M | A | I | "\langle\text{string}\rangle" \]

The general productions describe the constructs of the TBL. Every trajectory starts with the symbol S that is replaced by one or more symbols C. The nonterminal symbol C can be transformed into a symbol for motion M, acceleration A, state information I, or it can become a comment enclosed in quotation marks.

2) Quantifier productions:
\[ C \rightarrow \{\langle q\rangle\} | \langle q\rangle Q \]
\[ Q \rightarrow * | + | - | \# \]

Quantifier productions are used to increase the readability for humans and are important for compression. A quantifier consists of parenthesized symbols S and a repetition indicator. The latter defines the number of repetitions that can either be a numeric constant \( <q> \) or a frequency descriptor \( Q \), where \( Q \) is one of the frequency descriptors from Table II. The numeric constant is used for precise trajectories, whereas the frequency descriptors are better for generic trajectory patterns.

3) Motion productions:
\[ M \rightarrow f | T | T : Q | T : \langle q\rangle \]
\[ T \rightarrow H | H \langle d\rangle \]
\[ H \rightarrow r | l | t | b \]

Motion productions describe physical motions of an object, either the primitive motion \( \text{forward} f \) or several types of turns \( T \). Note, that in every time step the object always performs a movement with its current velocity. A turn \( T \) is a heading change \( H \) with or without the attribute \( <d> \) that specifies the angular degree of the heading change. If \( <d> \) is missing, a default value is assumed. \( H \) can be one of the following: \( \text{right} r, \text{left} l, \text{top} t, \) or \( \text{bottom} b \).

Also, \( T \) can be followed by a colon plus a frequency descriptor \( Q \) or a numeric constant \( <q> \). In either case, the heading change is the quotient of the angular degree divided by the quantifier \( Q \) or by \( <q> \). For example \( r180:2 \) describes half a 180° right turn, i.e. it is equivalent to \( r90 \). The format is more useful to express curves when the exact turning rate is unknown. For example a hippodrome is written as \( ((f)-(r180:..))\{2\} \). This trajectory starts with a few forward moves \( f \) – followed by a 180° degree turn that is split into a normal number of partial heading changes that are repeated a normal number of times to form a semi-circle \( (r180:..) \). The whole movement is repeated two times \( \{2\} \). Note, that it is not \( \{<2>\} \). The \( <> \) in the productions of the grammar indicate that a value is inserted.

4) Acceleration productions:
\[ A \rightarrow / | \backslash \]

Acceleration productions control the velocity of an object. As we describe the language in terms of the object’s behavior, there are velocity operations: \( \text{speed up} / \) and \( \text{slow down} \backslash \). The former increases the current velocity by multiplying it by the constant velocity factor \( \hat{v} \). The latter decreases the velocity by dividing it by \( \hat{v} \). The velocity change affects subsequent motions only. For example, in \( r/f \) the right and forward moves are done at the increased velocity, whereas in \( r/f \) the forward move is done faster than the right move.

5) State information productions:
\[ I \rightarrow p \langle x \rangle, \langle y \rangle, \langle z \rangle | h \langle x \rangle, \langle y \rangle, \langle z \rangle | v \langle s \rangle \]

These productions change the internal state of an object. They are only necessary for reconstructing the original trajectory from TBL and are negligible for comparing trajectories. The current coordinates \( p<x>,<y>,<z> \) of the object on its trajectory are usually specified at the beginning of the trajectory and are updated whenever the reconstructed coordinates differ from the real position. Such a synchronization may be needed if symbols are lost in the transmission over a network or if an object reenters an observed area at a position that is different from where it has left. The same holds for the actual heading \( h<x>,<y>,<z> \) of an object and its initial velocity \( v<s> \).

D. Expressiveness

Some aspects of TBL need special consideration.

1) Valid trajectories: The grammar \( G_{TBL} \) is too general. It is possible to construct words \( \omega \) in \( G_{TBL} \) that do not describe valid trajectories. For example, a trajectory \( \omega \) without a motion descriptor \( M \) does not have a spatio-temporal correspondent, although according to \( G_{TBL} \) the word \( \omega \) is valid. In general, the valid trajectories \( L_{valid}(G_{TBL}) \) are a subset of \( L(G_{TBL}) \). We did not provide \( G_{TBL}^v \) that only allows valid trajectories.
since $G'^{TBL}$ is no longer context free and cannot be used for a latency-free encoding at runtime that does not need (large segments of) the whole trajectory as its input.

2) Standstill problem: Up to now, velocity changes are modeled by multiplying/dividing the velocity by $\hat{\nu}$. With this approach, a standstill cannot be expressed, since the velocity cannot become zero. To express the standstill (without introducing an extra symbol), the velocity can be set to zero explicitly by means of a $v0$. This however, will render subsequent / symbols useless since $0 \cdot \hat{\nu} = 0$. To solve that problem, after multiplication/division with $\hat{\nu}$ the velocity is compared to a threshold value. If it is below that threshold, it is set to the threshold.

For example in $\langle f \rangle \langle f \rangle v0 \langle f \rangle \langle f \rangle \langle f \rangle$ the object slows down ($\langle f \rangle \langle f \rangle$) to standstill ($v0$). The object remains standing for two time steps that are modeled as forward moves at zero speed. Instead of the forward moves, heading changes can of course be used if the sensors detect any turning. With the first acceleration symbol the velocity is set to the threshold and the object moves forward /$\langle f \rangle$, finally reaching a speed of $\hat{\nu}^2$.

3) Turning on place: To keep the number of TBL symbols low, we do not use special symbols for turning. A turn is considered as a part of a motion. This does not exclude a turn on place as it can be expressed as a motion with zero speed.

4) Double move problem: For enhanced readability and to keep the number of symbols small, TBL cannot express that an object moves both horizontally and vertically in a single time step. Instead, we state the motion with the larger direction change first, e.g. if the horizontal change is larger than the vertical change, only the horizontal change is encoded. The neglected vertical shift is postponed and added to the next time step where again the larger change is expressed. The comparison can also be weighted (by means of a fourth TBL configuration parameter) to focus on relevant motion axes.

E. Examples

Figs. 2-4 show TBL trajectories and their corresponding reconstructed spatio-temporal appearance. The plots consist of the xy-plane projection on the left plus smaller graphs for a velocity profile and separated x-, y-, z-plots over time.

Fig. 2 shows an idealized track of a miniature train that is performing two loops. This trajectory is an ideal reconstruction of a track recorded by the Fraunhofer Institute for Integrated Circuits IIS (Fraunhofer IIS) with the RTLS Witrack system [9] (see Fig. 4 for the original spatio-temporal trajectory). It consists of only forward moves and right turns at a constant velocity. Fig. 3 depicts a more complex trajectory.

IV. TRANSLATION INTO TBL

RTLSs usually provide spatio-temporal data streams of object coordinates with a constant or variable sampling frequency. Formally, $T_A = (L_0, L_1, \ldots, L_{n-1})$ is the trajectory of object $A$ with length $n$, $L_k$ is the object’s position in 3D space, i.e. $L_k = (x_k, y_k, z_k)$ for sample $k \in [0; n]_{\mathbb{N}}$ at the sampling time $t_k$.

We now present an online transformation to generate the trajectory description in TBL from such a spatio-temporal data stream. The TBL encoding uses a strict time discrete way, where $\Delta t^{TBL}_i$ is the constant move time in the TBL domain. Formally, $\omega_A$ is a word in $L(G^{TBL})$ that describes the trajectory of the object $A$. The word $\omega_A = (c_0, c_1, \ldots, c_{m-1})$ is the series of symbols that express the object’s trajectory for $i \in [0; m]_{\mathbb{N}}$ and the corresponding TBL time $t^{TBL}_i$. Each $c_i$ can be a sequence of several terminals from $\Sigma^{TBL}$. The transformation $F^{TBL}_T : T_A \rightarrow \omega_A$ transforms the trajectory $T_A$ into the word $\omega_A$. Note, that $t_i$ refers to the time of the symbol $c_i$ in the TBL domain whereas subscript $k$ refers to time stamps of the original trajectory.

The transformation $F^{TBL}_T$ internally uses several variables.
to store the state of the trajectory as it could be reconstructed from the TBL domain. $L_i^{TBL}$ is the reconstructed location after a move $c_i$ has been performed. The velocity $v_i^{TBL}$ and the headings $\varphi_i^{TBL}$ and $\sigma_i^{TBL}$ are similarly treated.

Our $F_{TBL}$ algorithm comprises the following steps, the first two of which will be discussed in detail below:

1) Calculate initial symbols.
2) Determine velocity changes plus motions and recalculate the internal variables.
3) Increment $t_i^{TBL}$ by $\Delta t^{TBL}$. The time stamp $t_i^{TBL}$ is in the TBL domain for which a symbol will be emitted. Hence, earlier locations $L_k$ of the original trajectory ($t_k < t_i^{TBL}$) are skipped.
4) As long as the end of $T_A$ has not been reached, continue with step 2. Note that $t_k \geq t_i^{TBL}$ holds due to step 3.

Since in general the spatio-temporal sampling period is not equal to the move time in TBL (i.e. $t_k - t_{k-1} \neq \Delta t^{TBL}$), we introduce a time scaling factor $\lambda_i$ for symbol $c_i$, that is a multiplication factor for $\Delta t^{TBL}$. The factor $\lambda_i$ is the difference between the current sample time $t_k$ and the previous time $t_i^{TBL}$ in the TBL domain in relation to the TBL's move time $\Delta t^{TBL}$.

$$\lambda_i = \frac{t_k - t_{k-1}}{\Delta t^{TBL}}$$

### A. Initialization (step 1)

At the beginning of the trajectory, $F_{TBL}$ constructs the symbols $c_0$ that encode the starting location, $c_0 = p \langle L_{0,x}, h_{0,y}, h_{0,z} \rangle$. The internal TBL location $L_0^{TBL}$ is set to $L_0$, the first location of the given trajectory.

As soon as the second location $L_1$ becomes available, $F_{TBL}$ works on the symbols for $c_1$ which are the concatenation of three parts: a velocity description $c_1^V$, a heading description $c_1^H$, and a subsequent $t$ to specify a forward move.

For $c_i^V$ the velocity $\vec{v}_i$ and heading $\vec{h}_i$ are calculated (for $k = 1$ and $i = 1$) according to:

$$\vec{v}_i = \frac{L_k - L_{i-1}^{TBL}}{\Delta t^{TBL}} \cdot \lambda_i \quad \vec{h}_i = \frac{\vec{v}_i}{|\vec{v}_i|}$$

With that, $c_i^V$ is $v < \vec{v}_i >$. The initial heading $c_0^H$ is $h < h_{0,x} >, h_{0,y} >, h_{0,z} >$.

### B. Velocity change and motion type (step 2)

For every location $L_k$ that follows in the trajectory, $F_{TBL}$ increments $i$ and determines the symbols that encode the velocity change $c_i^A$ and the motion type $c_i^M$. The velocity change $c_i^A$ depends on $vmult$:

$$vmult = \log_2 \frac{|\vec{v}_i|}{|\vec{v}_{i-1}|}$$

Based on $vmult$, $c_i^A$ is constructed as follows:

$$c_i^A = \begin{cases} / & vmult > 1 \\ \varepsilon & else \end{cases}$$

If there was a significant change of speed ($|vmult| \geq 2$), a single / or \ is insufficient. Therefore, $c_i^A$ is extended with a quantifier: $c_i^A = (v < c_i^A >) (\varepsilon < vmult >)$.

For the motion symbol $c_i^M$, $F_{TBL}$ then calculates the current horizontal $\varphi_i$ and vertical heading $\sigma_i$:

$$\varphi_i = \arctan \frac{h_{i,y}}{h_{i,x}} \quad \sigma_i = \arctan \frac{h_{i,z}}{|h_{i,y}|}$$

Next, $F_{TBL}$ calculates the horizontal heading change $\Delta \varphi_i^{TBL}$ and the vertical change $\Delta \sigma_i^{TBL}$ based on the internal heading values $\varphi_{i-1}^{TBL}$ and $\sigma_{i-1}^{TBL}$. For $\lambda_i > 1$, i.e. if the sampling time is above the TBL move time, $F_{TBL}$ scales the heading change accordingly. Note, that since the heading change cannot be scaled linearly with respect to $\lambda_i$, $\lambda_i$ is raised to the power of $\varphi_i$ for symbol $c_i$. The velocity $\vec{v}_i$ and determines the symbols that encode the starting location, $\vec{h}_i$ and $\vec{h}_{i-1}$ are similarly treated.

$$\Delta \varphi_i^{TBL} = \frac{\varphi_i - \varphi_{i-1}^{TBL}}{\lambda_i} \quad \Delta \sigma_i^{TBL} = \frac{\sigma_i - \sigma_{i-1}^{TBL}}{\lambda_i}$$

After these preparations, $F_{TBL}$ constructs $c_i^M$ as:

$$\begin{cases} \tau \langle \Delta \varphi_i^{TBL} \rangle \Delta \varphi_i^{TBL} < 0 \lor |\Delta \sigma_i^{TBL}| < |\Delta \varphi_i^{TBL}| \\ \lambda \langle \Delta \varphi_i^{TBL} \rangle \Delta \varphi_i^{TBL} > 0 \lor |\Delta \sigma_i^{TBL}| < |\Delta \varphi_i^{TBL}| \\ \xi \langle \Delta \sigma_i^{TBL} \rangle \Delta \sigma_i^{TBL} = 0 \\ \beta \langle \Delta \varphi_i^{TBL} \rangle \Delta \varphi_i^{TBL} < 0 \lor |\Delta \sigma_i^{TBL}| > |\Delta \varphi_i^{TBL}| \\ \gamma \langle \Delta \sigma_i^{TBL} \rangle \Delta \sigma_i^{TBL} > 0 \lor |\Delta \sigma_i^{TBL}| > |\Delta \varphi_i^{TBL}| \end{cases}$$

Note, that the comparison of $\Delta \varphi_i^{TBL}$ and $\Delta \sigma_i^{TBL}$ can be weighted to prefer one direction, e.g. reducing the translation error in this direction. Finally, after emitting $c_i^A$ (if not $\varepsilon$) and $c_i^M$, $F_{TBL}$ recalculates $\varphi_i^{TBL}$, $\sigma_i^{TBL}$, $v_i^{TBL}$, and $L_i^{TBL}$.

It is obvious that $F_{TBL}$ can only construct valid words $\omega_T$ that are in $L(G_{TBL})$ and that also conform to the additional restrictions discussed in Section III-D.

### C. Example

Fig. 4 shows the first part of a TBL description generated from a real captured trajectory of a miniature train. It also shows the reconstructed and the original trajectory (dotted line) as a spatio-temporal visualization. The translated TBL trajectory has a turning resolution of 8 bit and horizontal movements are preferred by means of a weight factor of 2.

The reconstructed trajectory is a very close match to the original one. See inset that is a zoom in on the left lower corner. Only the velocity plot shows moderately higher deviation, since the velocity is quantified by TBL. Altogether the reconstructed positions have a standard deviation of 2 mm in relation to its originals.

Assume that the original trajectory is stored in double values for $x$ and $y$. The 885 samples of the example above thus take 13.8 kByte. The uncompressed TBL representation encoded
as a string needs 6.0 kByte (-56.5%). Another 10.0% can be saved (for the example above) if repeated segments in the trajectory are detected and encoded as it will be discussed in the next section.

V. DETECTING AND ENCODING OF REPETITIONS

The above transformation algorithm \( F_{TBL} \) does not detect repeated segments. This is done in a second stage, a TBL-to-TBL transformation. A repeated segment is a substring of a TBL trajectory that can be aggregated by means of a quantifier. For example in \( frl/lfrlll \) the three \( l \) symbols can be expressed as the aggregate \( (1)(3) \). A subrepetition occurs within another aggregate, i.e. the repetition is part of the repeated segment of the enclosing repetition, e.g. in \( frlllfrlll \) which can be expressed as \( (fr(1)(3))(2) \) the subrepetition \( (1)(3) \) is part of the enclosing repetition. Note, that \( ll \) is not a subrepetition of \( lll \), since it is not part of aggregate \( (1)(3) \).

Since in general, a TBL string may contain several repetitions that can overlap, there are several ways to aggregate that string. A repetition detecting and encoding algorithm has to find the best combination of aggregates that can either maximize the readability or yield the highest compression rate. Further, for RTLSs, only an online version is acceptable.

Our repetition detection and encoding algorithm uses a sliding window and thus introduces a limited latency (that depends on the size of the window) and restricts detection capability to repetitions that completely fit into the window. But for RTLSs these drawbacks are acceptable.

We perform repetition detection and encoding in three steps:
1) find all possible repetitions inside the sliding window,
2) determine the best combination of aggregates, and
3) push the oldest symbol or aggregation out of the window.

A. Finding repetitions

Whenever a new symbol \( c_i \) of a TBL data stream is queued into the sliding window \( sw \), Algorithm 1 searches for repetitions that become available due to the new symbol. For growing values of \( size \), the algorithm compares the rightmost symbols to the rest of the sliding window. Segments of length \( size \) are compared as long as they match (lines 1-3, 8). In Fig. 5 the rightmost segment of size 1 matches two times and fails on the third symbol. In the next iteration for size 2, already the first comparison fails. The same holds for larger values of \( size \). For every match, a corresponding repetition \( R \) is created (line 4) and stored in a global list (line 6).

![Figure 5. Finding repetitions.](https://example.com/figure5)

**Algorithm 1: Finding repetitions**

```
Input : Sliding window of TBL symbols \( sw \)
Result: List of repetitions \( \mathcal{R} \) with their subrepetitions \( \mathcal{S}_\mathcal{R} \)

for size ← 1 to sizeof\( sw/2 \) do
    for offset ← size to sizeof\( sw \) step size do
        if \( sw(l to size) = sw(offset + l to offset + size) \) then
            \( \mathcal{R} \leftarrow \text{Repetition}(start = l, \end = offset + \text{size} \text{ size} = \text{size}) \);
            \( \mathcal{S}_\mathcal{R} \leftarrow \text{SubrepsInRightmostSegment}(\mathcal{R}) \);
            store \( \mathcal{R} \) and its \( \mathcal{S}_\mathcal{R} \); 
        else
            continue with next size;
```

It is not shown in the pseudo-code that the position information of all the repetitions stored in the global list is updated (shifted to the left) whenever symbols are pushed into and out of the window. After a while the global list points to repetitions that are either buried somewhere within the window or to other repetitions that also start at the right hand side but that are shorter than the newly found \( R \). Hence, whenever a new \( R \) is inserted into the global list, the algorithm can inspect the global list to find all subrepetitions \( \mathcal{S}_\mathcal{R} \) that are within \( \mathcal{R} \).

In the example the sliding window started empty. When the first symbol \( f \) is pushed in, nothing can be compared. With the second symbol \( r \) the sliding window contains \( ---fr \) and the algorithm compares \( r \) to \( f \). The first match is found only after the two symbols \( ll \) have been inserted. The window then contains \( -frlll \), and the first repetition \( \mathcal{R}_1 \) is found for \( ll \), namely \( -fr[1|1]|\mathcal{R}_1 \) (square brackets indicate the start and end of the symbols that can be aggregated, vertical lines separate repeated segments). When the third \( l \) is pushed in, the previous repetition \( \mathcal{R}_1 \) is updated, i.e. shifted to the left by one position. Due to the new \( l \), two new repetitions become available: \( frl[1|1]|\mathcal{R}_2 \) and \( fr[1|1]|\mathcal{R}_3 \). \( \mathcal{R}_1 \) is not a subrepetition of \( \mathcal{R}_3 \), since it does not fit into the repeated
Determined the optimal arrangement two subrepetitions the gain is $1$. In the example of $R_3$, the gain is $\rho = 0.5$. If that is the case, then these subrepetitions can themselves be expressed as aggregates and further add to the combined gain $g_R$. Hence, the optimal combined gain of a repetition $R$ is

$$g_R = g_R + \max_{S \subseteq S_R \land S \text{ non overlapping}} \left( 0, \sum_{\rho \in S} g^*_\rho \right).$$

Of course, for computing the optimal gains of the subrepetitions, the same recursive idea applies.

Consider $R_7$ in Fig. 6 again. It has three subrepetitions $R_{4,6}$. To find the optimal gain, we have to consider the power set of all the subrepetitions, but restricted to those that do not overlap. In the example, from the 7 non-empty sets in the power set (all combinations of $R_{4,6}$) only the three one-element sets $\{R_4\} \ldots \{R_3\}$ are free of overlaps. Their respective gain is $g^*_{R_4} = g^*_{R_5} = 0.5$ and $g^*_{R_6} = 1.5$. Hence, the best gain for $R_7$ can be achieved if the maximal gain that can be achieved by aggregating all non-overlapping combinations of its subrepetitions is added. Since $g_R = 10 - 5 - 0.5 = 4.5$ ($o_q = 0.5$) and since the maximum of the $g^*$-values of the three sets is 1.5, we therefore get $g^*_{R_7} = 4.5 + 1.5 = 6$. Fig. 6 gives the $g^*$-values of all repetitions inside the brackets, the contribution of subrepetitions is mentioned where applicable. Note, that this is a special case. In general, a set could have more than one element, where its subrepetitions can be turned into aggregates at the same time. Consider a repetition with the repeated segment $frfrlll$. Obviously there are two non-overlapping subrepetitions $[fr|fr]$ and $[lll|lll]$ that both can be turned into aggregates, whose gains add up.

Algorithm 2 takes a repetition $R$ and its subrepetitions $S_R$ and computes the optimal combined gain $g_R$. The forloop loop computes the power set by a systematic sweep through the subrepetitions. It takes one of the subrepetitions, say $\rho$, and recursively computes the optimal gain (line 6) that can be achieved for it and its subrepetitions $S_\rho$. In the above $R = frfrlll$ example, the algorithm would first compute the optimal gain of the subrepetition $lll$. It has to add whatever can be contributed by the rest of $R$ (line 7). To do so, the algorithm restricts $R$ to what is left after $\rho$ is taken away. Here, $R|\rho$ is $rfrf$. Similarly, the set of subrepetitions is restricted to what is left after $\rho$ has been selected: these are the subrepetitions that do not overlap with $\rho$ and that are within $R|\rho$. Less formally, line 7 considers both the symbols and the subrepetitions that are on the left side of $\rho$. This of course is recursive again, since in general there might be more than just one subrepetition within that remainder of $R$. Since the forloop loop treats every subrepetition in $S_R$ in that way, the algorithm enumerates the power set and finds the best possible gain of all combinations of subrepetitions that cover $R$.

Let us consider $R_7$ first. The algorithm picks one of the three subrepetitions $R_{4,6}$ in turn. For each of the subrepetitions the optimal gain is computed in line 6. If $R_7$ is restricted with respect to one of these $R_i$, it will be either $fr$ or $frrR$. Restricting the set of subrepetitions always yields the empty set, so that the optimal gain is $g_{\max} = \max(0.5, 0.5, 1.5)$. 

---

### Table 1: Repetitions and their gains ($o_q = 0.5$).

<table>
<thead>
<tr>
<th>Segment</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f\ r\ l\ l\ l\ l\ l\ l\ l\ l$</td>
<td>$6.0 (R_6)$</td>
</tr>
<tr>
<td>$f\ r\ l\ l\ l\ l\ l\ l\ l\ l$</td>
<td>$4.5$</td>
</tr>
</tbody>
</table>

---

### Figure 6: Repetitions and their gains ($o_q = 0.5$).

- $R_1$
- $R_2$
- $R_3$
- $R_4$
- $R_5$
- $R_6$
- $R_7$

---

The goal of this step is to find an aggregation of repetitions that yields the best readability or maximal reduction of symbols. We determine the best arrangement by calculating a potential gain for each repetition and then searching for the highest possible combinations of repetitions. Consider a repetition $R$. It has a segment of length $l_R$ symbols that appears $x_R$ times in $R$ so that $R$ has an overall length of $s_R = x_R \cdot l_R$. When $R$ is aggregated, we can save $s_R - l_R$ symbols but we have to take the overhead $o_q = 5$ for the qualifier symbols (e.g., $(\ldots, \{<x_R>\})$) into account, since the parentheses, the curly brackets, and $x_R$ add five additional symbols (for simplicity let us assume that $x_R < 10$. The correct value is $o_q = 5 + \log_{10} x_R$ if all nonterminal symbols have an encoding length of 1). Hence, for repetitions without subrepetitions the gain is

$$g_R = s_R - l_R - o_q.$$
Algorithm 2: Recursive gain calculation of repetitions

Input : Repetition $\mathcal{R}$ and its subrepetitions $\mathcal{S}_R$
1 gain $(\mathcal{R}, \mathcal{S}_R) = s_{\mathcal{R}} - l_{\mathcal{R}} - o_{\mathcal{R}} + \text{subgain}(\mathcal{R}, \mathcal{S}_R)$;
2 subgain $(\mathcal{R}, \mathcal{S}_R)$;
3 begin
4 $g_{\text{max}} = 0$;
5 foreach $\rho$ in $\mathcal{S}_R$ do
6 $g = \text{gain}(\rho, \mathcal{S}_\rho)$
7 $\quad + \text{subgain}(\mathcal{R}|_\rho, \mathcal{S}_\rho|_\rho)$;
8 $g_{\text{max}} = \max (g, g_{\text{max}})$
9 return $g_{\text{max}}$;
10 end

But Algorithm 2 cannot only be used to compute the optimal gain of a repetition. It can also be used to determine the optimal gain available in the sliding window. For that purpose, it has to be started with $\mathcal{R} = \text{sw}$.

Let us assume that the foreach processes the repetitions backwards. Then the foreach loop considers $\mathcal{R}_{16}$ first (since $\mathcal{R}_{16}$ has a subrepetition $\mathcal{R}_{12}$, the total gain of $\mathcal{R}_{16}$ is 2). In line 7, the algorithm recursively dives into the rest of the window. It studies $\text{sw}[12:5]$ with the set of the three repetitions $\mathcal{R}_{1,3}$. The maximal value among those is 1.5. The total gain of this arrangement is 1.5 + 2 = 3.5.

The foreach tries many combinations. Among those are the two that are shown in the bottom lines of Fig. 6. For the first of the result lines, the foreach has considered $\mathcal{R}_{12}$. The rest of the window $\text{sw}[12:3]$ holds the repetitions $\mathcal{R}_{1,7}$. The recursive call in line 7 detects that $\mathcal{R}_7$ (with the appropriate sub-aggregate) returns the best gain, as we have discussed above. Hence, the final readability-gain for this case is 6.5. The best encoding is $(fr(l)\{3\})\{2\}\{1\}\{2\}$. The last line in Fig. 6 shows the 2nd best aggregation found: $\mathcal{R}_3$ and $\mathcal{R}_{15}$ are used for the aggregation $fr(l)\{3\}fr(l)\{5\}$.

C. Pushing out optimal repetitions

As soon as the sliding window is full and the optimal aggregation has been computed, the leftmost symbol is written to the output and replaced by a “-”. If the leftmost symbol is part of an aggregation found in step 2, the whole aggregate is written to the output and all positions that become vacant are replaced by dashes. In Fig. 6 the leftmost $f$ belongs to the optimal aggregation (gain 6.5). So the first 10 slots of the window are output. If the last line of Fig. 6 had been the optimal solution, only one $f$ had been emitted.

Afterwards we shift all slots of the sliding window (conceptually) to the left by one slot. We update the positions of all repetitions in the global list (and their subrepetitions) that still correspond to symbols in the window. And we purge repetitions from the list that contain emitted symbols.

Then the detection process starts over by pushing in the next input symbol. Note that for many applications, e.g. for trajectory similarity reasoning, the information symbols (p, h, and v) can be skipped and suppressed.

VI. Conclusion

We proposed a novel representation for object trajectories based on a formal language. The language is designed so that representations can be generated at runtime (online) with zero sampling latency. The same holds for the reconstruction of a trajectory. The representation describes the trajectory based on a separation of velocity and motion information. It is both human-readable and amenable for algorithmic processing.

We also presented a latency-free online algorithm for generating a TBL representation from spatio-temporal data. We proposed solutions for the standstill problem and the double move problem and showed that the translation can be applied to real data and furthermore can compress the data size.

Finally, an algorithm for detecting and encoding repetitions within the TBL is given. This yields a better readability or further increases the compression rate. On a benchmark, we have obtained a compression rate of 66.5%.

REFERENCES