Towards Natural Language Search and Exploration of Automotive Telemetry with QuoTS

João M. A. Rodrigues[†1,3], Hugo Gamboa[1], Seyhan Ucar[2], Eamonn Keogh[3]
[1]LIBPhys-UNL FCT – Nova University of Lisbon, Caparica, Portugal; [2] InfoTech Labs, Toyota Motor North America, R&D, Mountain View, CA, USA; [3]Computer Science Department, University of California, Riverside, USA
jmd.rodrigues@campus.fct.unl.pt, hgamboa@fct.unl.pt, seyhan.ucar@toyota.com, eamonn@cs.ucr.edu

ABSTRACT

Modern automobiles produce copious amounts of data reflecting the driver's behavior. This is true, even if the "driver" is an algorithm. This data can reveal atomic actions, such as lane changes and U-turns, and may potentially reveal behaviors at a higher semantic level, such as distracted driving, aggressive driving, etc. As such, there are many stakeholders that may wish to search and explore such datasets, including engineers, insurance companies, accident investigators, etc. However, there is a dearth of useful tools for this task. Existing algorithms for time series similarity search can be useful in some domains, but as we shall show, they have difficulty generalizing over the variability created by considering different drivers, different vehicles, and differing road conditions. In any case, such tools tend to be unintuitive and require a steep learning curve. To address this problem, we propose to create a search paradigm based on natural language, where a user can search automotive telemetry by tersely describing the events she wishes to discover. As we will show, even with the limited vocabulary we propose, our system is expressive enough to allow a user to find target events and behaviors with high specificity and sensitivity.

CCS CONCEPTS

Human computer interaction • Information retrieval • Ubiquitous and mobile computing

KEYWORDS

Time Series, Similarity Search, Query Languages

ACM Reference format:

FirstName Surname, FirstName Surname and FirstName Surname. 2018. Insert Your Title Here: Insert Subtitle Here. In *Proceedings of ACM Woodstock conference (WOODSTOCK'18). ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

[*]Article Title Footnote needs to be captured as Title Note
[†]Author Footnote to be captured as Author Note

WOODSTOCK'18, June, 2018, El Paso, Texas USA
 978-1-4503-0000-

1 Introduction

Automobiles are increasingly monitoring every aspect of the driver's behavior. Such data can be used for many direct and explicit purposes, such as optimizing fuel consumption or enhancing safety systems. However, there are also many potential indirect and offline uses of this data. The data may be of interest to engineers optimizing driver's comfort, study the behavior of automated vehicles, to insurance companies fine-tuning insurance rates, to accident investigators trying to understand the cause of an accident, etc. [8,10–12,14,19]. It is commonly understood that individuals with domain experience can often "read" such telemetry. For example, in academic and industrial labs it is common to hear engineers annotate such data "*This y-axis bump is where he hit the pothole, and then the sharp decline here in the x-axis is where he begins to apply the brakes"* [22]. However, this ability to *interpret* such data does not help in *searching* such data collections. Simply manually panning through the data does not scale beyond minutes of data, and we may wish to search massive data archives.

There is an extensive literature on time series similarity search. This involves a user finding or sketching a query template, and then using that template to search the data. This search is typically performed using a distance measure based on Euclidean distance (ED) or Dynamic Time Warping (DTW). However, while such algorithms are very useful in certain domains, they can have difficulty generalizing over the variability created by considering different drivers, different vehicles, and differing road conditions. In any case, such tools tend to be unintuitive and require a steep learning curve. To address this problem, we propose to create a search paradigm based on natural language (NL) keyword queries. The usage of such queries is particularly common in our daily chores and are a logical choice when trying to explain what we are looking for. Good examples of systems that use such query methods are search engines (such as *Google)* and chatbots (more recently, *ChatGPT* has stirred the community as an efficient system to answer any of our NL queries). It only makes sense that we move towards using such systems for data analysis. But how can we do it with *Quo*TS? Informally, if a user wishes to find examples of a particular behavior with our system, she can

Comentado [JMR1]: I have been thinking about this and am wondering if it truly is yet natural language, because we are using words+operators. It is near this but do you think some reviewers might think we are misleading the readers?

simply "*Google* it", by describing the data she expects to see, *given* that behavior. For example, if she wants to find data that is suggestive of a sudden stop, a simple query of `surge: high valley`, (recall that *surge* is the acceleration in the normal direction of travel) will return candidate regions that have a high probability of exhibiting that behavior.

As hinted at in Figure 1, the Euclidean distance can be too 'literal', and produce similarity judgements that do not reflect human judgements. In contrast, natural language keywords can be optimized to reflect similarity judgements of users. For example, users in the automotive domain often distinguish between U-shaped, and V-shape valleys, as they reflect different driving experiences.

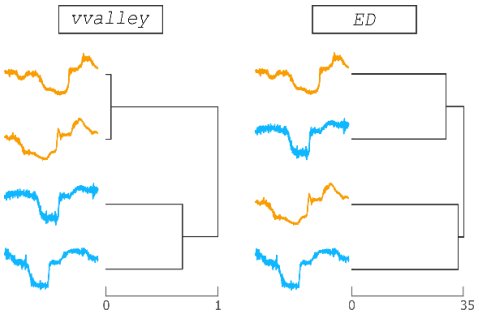

Figure 1 - *Four time series snippets in two classes clustered using two methods. left) Using the method introduced in this paper, here defining similarity based on how well the snippets embody the keyword* `v-valley` *(i.e. "V-shaped Valley"). right) Using the ubiquitous Euclidean distance.*

Concretely, in this work we will demonstrate the following:

- The proposed natural language representation is expressive enough to discriminate subtly different behaviors, for example a *full-legal-stop* vs. a *rolling-stop*. This may be surprising, as the raw data is high dimensional and often noisy, and it is not obvious that simple text queries are expressive enough to distinguish semantically different behaviors.
- As noted in [5], *expressiveness* is necessary but not sufficient for producing a useful classifier/query system (The original wording is "*Representable Does Not Imply Learnable*"). We will further show that our proposed representation is intuitive enough to allow novice users to express their information needs.

We will present forceful evidence of these claims in Section 4, but we will briefly preview some evidence here. We consider the four-class Trace dataset, which has been studied in more than 1,000 papers [1]. As Figure 2 shows, it is possible to use our system to create queries that can separate each of the four classes from the rest of the data. Note that while the results in this figure are not exactly commensurate with the classification task, the highly competitive nearest neighbor algorithm with Euclidean Distance is only 76% accurate for the classification task on this dataset, suggesting that this is a non-trivial discrimination problem.

The rest of this paper is organized as follows. In Section 2 we review related work. In Section 3 we introduce the definitions and notations need to introduce our proposed query system. Section 4 sees a detailed empirical evaluation of ideas on both real automotive telemetry and challenging proxy datasets. Finally in Section 5 we offer conclusions and directions for future work.

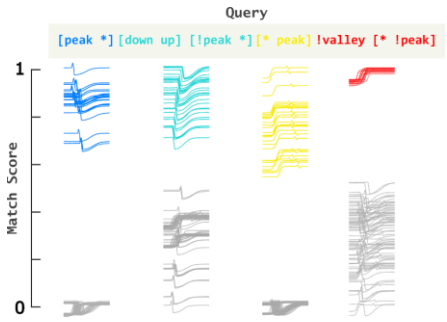

Figure 2 - *Using our proposed query language, we can create short, intuitive natural language queries to rank the 100 exemplars in Trace, separating our sought class from the remaining data. Here "*" is don't-care, "!" is not, and square brackets are a grouping operator (more details in Section 3).*

2 Related Work

There is a large literature on time series similarity search, see [26] and the references therein. However, in most cases it is assumed that the query comes from a downstream *algorithm*, not a *human*. As such, there has been relatively little attention paid to the ability of humans to formulate meaningful queries. In principle one could do "query-by-sketching" and invite the user to draw the pattern she is interested in finding [15,16]. The recent "Qetch" system is a prominent example of this approach [15]. However, there are two possible limitations to such an approach: First, it is not clear that most people have the ability to sketch their query. For example, many people cannot even draw an accurate circle [25]. Secondly, as Figure 1 hinted at, classic distance measures may be too literal and limited in expressiveness to retrieve the desired pattern. As a simple example, suppose that a user wishes to retrieve all highly symmetric patterns. There is simply no way to do this with Euclidean distance or similar distance measures.

Other researchers have noted these issues and proposed more flexible queries languages for time series. For example, the SDL (shape definition language) of [11]- allows the user to formulate "blurred" queries. However, we believe that most such systems are not accessible for the typical user. For

Comentado [JMR2]: Is this true? Considering that we are not actually asking to other subjects to write queries or study if other subjects are able to find what they are looking for in the data. Should we consider rephrase it so it does not give the impression that we are making a user study or something related?

example, in our proposed system, a 3-point-turn can be successfully queried by noting that the surge axis will exhibit three consecutive "bumps" and formulating the query `Surge: [peak peak peak]`. In contrast, SDL would require:

```
(Shape triplespeak (width ht1 ht2 ht3) (in
width (in order spike (ht1 ht2 ht3) spike
(ht1 ht2 ht3)))).
```

Several similar query systems based on regular expressions or SQL-like languages have been proposed, but none seem suitable for general use [17,20].

There have also been a handful of other attempts at natural language querying for time series [6,7]. None of these works seem to have been adopted by practitioners. We feel that this is because they probably suffer from too broad an ambition, proposing completely domain independent search. While domain independence is a worthy ambition (and our *eventual* research goal), it is clearly challenging. Even the word "spike" can have a different meaning for neuroscientists, economists, epidemiologists, and astronomers. In this work we take advantage of the fact that driving is a familiar, even quotidian, activity for most people, and therefore a domain for which most people have strong intuitions for. Moreover, this domain has a near unique property that allows a user to model the behavior they wish to find. We found that, in many cases we could glean intuition as to how a driver's behavior would reveal itself in telemetry by simply "puppeteering" a smartphone equipped with an app that shows its acceleration and gyroscope readings. For example, by modeling a 3-point turn by sliding the smartphone on a desk, we can see that this behavior best revels itself on the surge axis as three consecutive bumps.

3 Definitions and Methods

3.1 General Definitions

We begin by describing the necessary definitions and notation. The data type of interest is *time series:*

Definition 1 (Time Series): A time series T of length n is a sequence of real-valued numbers t_i: $T = t_1, t_2, \ldots, t_n$. Time series can occur as a *multidimensional* set.

Definition 2 (Multidimensional Time Series): A multidimensional time series (*MTS*) is represented by k time series T of length n, where $k > 1$. Such that:

$\boldsymbol{MTS} = \{T_1, T_2, \ldots, T_k\}$; $\boldsymbol{T_1} = t_{11}, t_{12}, \ldots, t_{1n}$; $\boldsymbol{T_2} = t_{21}, t_{22}, \ldots, t_{2n}$; $\boldsymbol{\ldots}$; $\boldsymbol{T_k} = t_{k1}, t_{k2}, \ldots, t_{kn}$

Some occurrences are multivariate and more easily (or even only) described by the dependencies over multiple dimensions of a time series. For example, the analysis of a driving behavior might require the usage of all three axis of an accelerometer, then: $\boldsymbol{TS} = \{ACC\_X, ACC\_Y, ACC\_Z\}$.

(*QuoTS)* provides a querying mechanism for multidimensional search by including a name identifier of the dimension (e.g $\boldsymbol{Name\_D_1:}$ $query_1$ $\boldsymbol{Name\_D_2:}$ $query_2$ … $\boldsymbol{Name\_D_k:}$ $query_k$*)* that is parsed to retrieve its corresponding query. The search process is then multidimensional but focused on identifying the local region of interest, maximized by the query (either *1D* or *kD*). This local region is called a *subsequence*.

Definition 3 (Subsequence): A subsequence $T_{i,m}$ of a time series T is a continuous subset of the values from T of length m starting from position i. $T_{i,m} = t_i, t_{i+1}, \ldots, t_{i+m-1}$, where $1 \le i \le n - m + 1$. Every subsequence is mapped and characterized by a set of properties. These are extracted features to which we associate intuitive and commonly used words to describe the dynamics of a *subsequence.* Each of these *subsequences* are selected with a sliding window, with the size m being defined by the user. A *word feature vector* of each property is then created.

Definition 4 (Word Feature vector): We define a *word feature vector* $\boldsymbol{W}$ as a mapping of feature $\boldsymbol{F}$ with a specific word. With feature, we intend to describe either a property, such as the *mean* or *standard deviation*, but also a *distance measure* to pre-defined examples. In linguistic terms, this *word feature vector* is an *adjective* of the *subsequence.* Every subsequence $T_{i,m}$ from each time series T is characterized by the selected set of words, being created a set of *word feature vectors* for each time series, further normalized between 0 and 1. The *word feature vector* has the size of T minus the subsequence length: $n\text{-}m+1$, and the feature value indicates how relevant is the word in the subsequence. Figure 3 shows an example of the *word feature vectors* for `up` (F_{up}) and `down` (F_{down}).

To allow interactive search, the *word feature vectors* are pre-

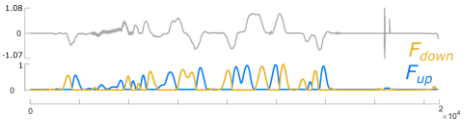

Figure 3 - *Example of two word feature vectors from a gyroscope signal. The words are* `up` *and* `down`*. Note that the feature vectors are normalized between 0 and 1. Further note that low values for* `up` *do not imply the presence of* `down`*, it simply implies that lack of* `up`*-ness. However, for some feature pairs X/Y, it is the case that X = 1-Y, for example symmetric /asymmetric.*

computed in an offline indexing stage. In addition to this, we also extract three dimensions of the same *word feature vector* with different window lengths, based on m: $W^1 \rightarrow m$; $W^2 \rightarrow \frac{m}{2}$; $W^3 \rightarrow \frac{m}{4}$; with the intent of matching ordered sequences of words inside a *subsequence* with the *grouped followed by operator*, as will be explained further. It is important to note that the ratio of the window used for the dimension W^2 and W^3 might have to be fine-tuned for the domain, as there might not be a "*one window ratio fits all*". However, empirically we observed that the exact value of this ratio is not critical to the success of search.

For each $\boldsymbol{W}$ is assigned a word $\boldsymbol{w}$. We use English words to make the process more intuitive, such as `noise`, `up` and `peak`. We recognize that the intuitive meaning of such words can vary from user to user depending on their domain, their experience

and on the current context. Either way, words can be mapped to features that are domain specific or *word feature vectors* can be given a domain specific vocable, providing a more appropriate mathematical thinking behind what is its meaning. In addition, we are aware that multiple words can be given the same meaning and for this reason, we associate several synonyms to each word. We also note that our proposed mechanism can benefit from the current advances in Natural Language Processing (NLP). For instance, synonyms could automatically be associated with the closest word listed in our vocabulary with word embeddings. We defer such considerations to future work. We define an initial subset of *features* that are mapped to *words*. When defining one *word feature vector* it would often come with a *negation pair*.

Definition 5 (Negation Pair): A *negation pair* (***!W***) represents the exact opposite of a defined *word feature vector* (***W***) following the rule:

$$!W = 1 - W$$

This indicates that when one increases the other one has to decrease proportionally. Examples of such *word feature vectors* are `symmetric` and `asymmetric`, or `complex` and `simple`. Note that some *words* might be the opposite of each other, but do not follow this rule, or even seem to be the opposite of each other, but are not. For instance, `up` and `down` are opposite of each other, but do not follow Equation 1. While one exists, the other can not, but it does not mean that when one is small, the other has to be high, since the subsequence might just be `flat`. Another case is the word `peak`. Intuitively, we would think that `valley` is the opposite of `peak`, but the consideration of `!peak = valley` is false.

In this work, we use the negation of a *word feature vector* for cases where there is no *negation pair*. This negation is realized using an *operator*.

Definition 6 (Operator): The same way we use word and sentence connectors in our language to create contrast or attribute a temporal sequence, in our proposed system we use *operators*. An *operator* is a metacharacter or a word that can be used to diversify the way *word feature vectors* are handled, either in the way the information is extracted or how these are combined. It contributes to a more versatile and expressive usage of this language. Currently, we have a simple list of four operators: `negation (!)`, `wild card (*)`, `followed by`, and `grouped followed by` (e.g., `[`W_1 W_2 … W_w This list can obviously be expanded and customized, but we want to demonstrate that with a minimal set of operators, most of the problems we present are solvable.

3.2 Mapping Features to Words

As previously noted, a set of features is used to extract several properties of all *subsequences* of a time series and attribute a semantic meaning to each one of them by mapping it to a specific *word*. It is our assumption that a *subsequence* can be mapped to a set of *words* that an analyst would use to describe it. Depending on the domain or vocabulary of the analyst, the set of *words* might have to be different and adjusted. Eventually, the *dictionary* can be expanded to other types of features and words. In any case, we want to demonstrate that this current set of *words* and *operators* can solve many transportation query search problems.

The initial subset of features is listed and described below. We divide the list of features in groups: *local, global,* and *special*.

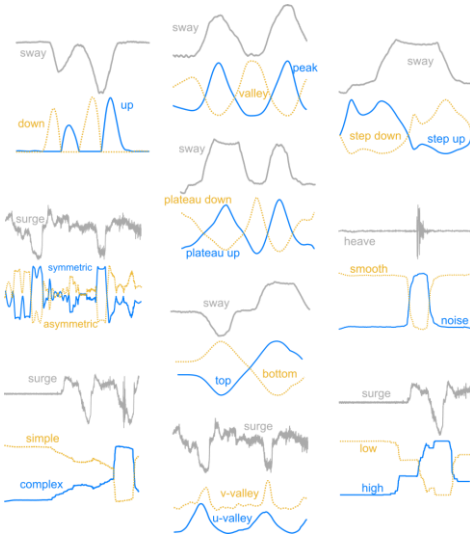

Figure 4 – *Examples of matches for most word feature vectors defined above, with subsequences from telemetry datasets. In gray are presented the subsequences that were used to generate the word feature vectors.*

- **Local Features**

`up (down)`: The slope estimation of a linear adjustment (*y= ax + b)* to the *subsequence*, being `up (down)` = a, if a_{up}>0 (a_{down} <0) or `up (down)` = 0, if a_{up}<=0 (a_{down} >=0).

`complex (simple)`: A complexity-invariant distance measure of the *subsequence* (`simple` = 1 - `complex`) [2].

`noise (smooth)`: The residual error when modeled by a moving average (`smooth` = 1 - `noise`).

`symmetric (asymmetric)`: The z-normalized euclidean distance (MASS algorithm), see [27]) to the *subsequence*'s horizontally flipped self.

`peak (valley)`: The logarithmic MASS distance to the template of a peak (valley), modulated by a gaussian function.

`stepup (stepdown)`: The logarithmic MASS distance to the template of a step-up(down) function;

plateauup (plateaudown): The logarithmic MASS distance to the template of a *plateau*-up(down) function;
uvalley (vvalley): The logarithmic MASS distance to the template of a *U-shaped* (*V-shaped)* valley function.

- **Global Features**

top (bottom): The moving average of the time series (`bottom` = 1 - `top`);
high (low): The difference between the maximum and minimum value of a *subsequence* (`low` = 1 - `high`);
middle: The inverse of the distance to the average of the signal for each *subsequence;*
uncommon (common): The matrix profile of the time series (`common` = 1 - `uncommon`).

- **Special Feature**

***shape*:** The MASS distance profile of the time series with a query given by the user as an example. A word must be given as well, so that the shape can be integrated into the query language.
Most of the word feature vectors are illustrated in Figure 4, with subsequences (in gray) from automotive telemetry data.

3.3 Operators

Web search engines have many operators at the user's disposal, but since a list of words is usually powerful enough to retrieve and correctly sort most of the desired results, very few (or none!) are often used. We believe that this is the case for this application as well but acknowledge that simple operators can make the query more natural and come in handy to perform conjunctions between features and multiple dimensions, such as temporal logic or negation. These operators are especially useful to close the gap between the query and human discourse, contributing to a more expressive mechanism when using the proposed language. Currently, four operators are available. Below is a list and description of each of them, starting with the negation operator (represented by the symbol `!`).
Negation Operator - *!W* : As mentioned above, most words come as an opposite pair, but some do not follow Equation 1. In these cases, or when the word has no direct opposite, it can be useful to penalize the presence of the word in a subsequence. This operator does that by applying Equation 1 to the word feature vector, *W*.
When describing time series, we inevitably use temporal logic in explaining the sequence of shapes we perceive. The next operator is `followed by`.
A followed by B: This operator rewards a subsequence represented by **A** followed by one subsequence that has a high score for **B**, within a distance of size *m*. `A` and `B` can be single words, multiple words or even queries for different dimensions of the time series.
With this operator, we look ahead of a subsequence in the time series. However, in some cases, it might be useful to describe the sequence inside the limits of the window we defined. For these we have a special case of `followed by`, which is the `grouped followed by` (represented as brackets - [ ]).
Grouped *followed by ([W_1 W_2 … W_N]):* Instead of looking ahead in the time series, we look *inside* the subsequence to reward an ordered sequence of words. In this special case, the subsequence is segmented into *N* sub-windows, with size int($\frac{N}{m}$), and the corresponding word is scored within this sub-window. For this, we use the other 2 dimensions of the word feature vectors ($\frac{W}{2}$ and $\frac{W}{4}$). If N<=3, $\frac{W}{2}$ is used, while if N>3, $\frac{W}{4}$ is used. Often, within a subsequence, the differentiating property occurs on the first half, last third or another sub-window of the subsequence, while the remaining sub-window(s) is(are) not relevant. We therefore introduce the `wildcard (*) operator`.
Wildcard - * - The sub-window where * is used is valued equally for all *subsequences*.
As with vocabulary, the reader could imagine expanding our dictionary of operators, but even with a limited set of them, we are able to successfully solve all the proposed search tasks, which cover dozens of examples.
After presenting the set of elements that can be used in *QuoTS* to query a pattern of interest, we are ready to explain how the query is turned into a score function and finally to a selection of the *k*-most relevant *subsequences*.

3.4 Querying with *QuoTS*

All words are stored in a vocabulary file, associated with a thesaurus file for synonym checks. When the user loads a signal to work on, all *word feature vectors* are extracted based on a specific window size and stored in memory. For each word, three sets of *word feature vectors* are stored (W, $\frac{W}{2}$ and $\frac{W}{4}$) based on the original window size. In case of having a *MTS*, all three sets of *word feature vectors* are extracted for each dimension. Having this set of information pre-computed helps make the search run at interactive speeds, even for large data collections.
When all data is pre-computed, *Quo*TS is ready to accept queries by the user. The query field accepts any word available in the vocabulary and thesaurus. When any of the words are not present in our vocabulary, we alert the user which is the closest word available, based on edit distance. The query can accept operators and works for multidimensional querying. These are relevant elements that are used as a reference to parse the query into individual scoring elements. This parsing process is made by looking into: (1) which dimension(s) of the time series is (are) included; (2) which operators are used; and (3) the single written words. The first two define how the score is calculated, that is, how word feature vectors are combined, as well as which dimensions of the word feature vectors are used. For instance, when including multiple signals, the query parses

which word(s) corresponds to which signal, to search for the correct index of *word feature vectors* in the pre-computed data. When the `followed by` operator is used, the query is parsed in which word(s) comes before and after it (this is applicable either if the operator is used for intra-signal or inter-signal search). Another element is the `grouped followed by` operator, which is parsed by identifying square brackets in the query.

When each of these elements are parsed, we end up with a single word or sequences of words, which are combined by summing their corresponding *word feature vectors* (this corresponds to an implicit OR). It is important to note that the score is calculated by adding together normalized scores for each parsed segment of the query. The reasoning is that each segment of the query should be weighted the same (e.g., if using the query `noise [up down]`, as `up` and `down` are combined, the range of this segment is [0-2], while noise is [0-1]. Therefore, `[up down]` is normalized between [0-1] before being added to `noise`). Finally, a score is given to each *subsequence*. The top k-*subsequences* are highlighted on the signal and sorted from highest to lowest. This process implies that trivial matches are not considered. As an example, if we want to search matches for the query `s1: [up down] s2: flat`, we parse it by signal, first computing the score function for `s1` and `s2` individually. Then, the score function of `s1` will be normalized between [0-1] to then be added to the score function of `s2`.

4 Experimental Evaluation

We conducted experiments with several datasets relevant to automotive scenarios. In these we demonstrate the representational power of the *Quo*TS by finding the desired patterns with meaningful queries. The datasets used are either public [21] or available in [3].

4.1 *QuoTS* captures gestures

We start by demonstrating the ability of *QuoTS* to sort how well individual shapes match a written query. For this, we used the *UWaveGestureLibrary* dataset from the *UCRArchive* as a proxy [13], which like telemetry data, relies on inertial time series. We use this proxy data simply because it is much larger than any publicly available *labeled* transportation data. However, we note that gestural interaction with the automobile interfaces is an area of active research [4,24]. With this example, we show that the system can recognize different gestures with or without intuitive queries, hence, if humans were able to learn and master this tool, this recognition problem would be largely solved.

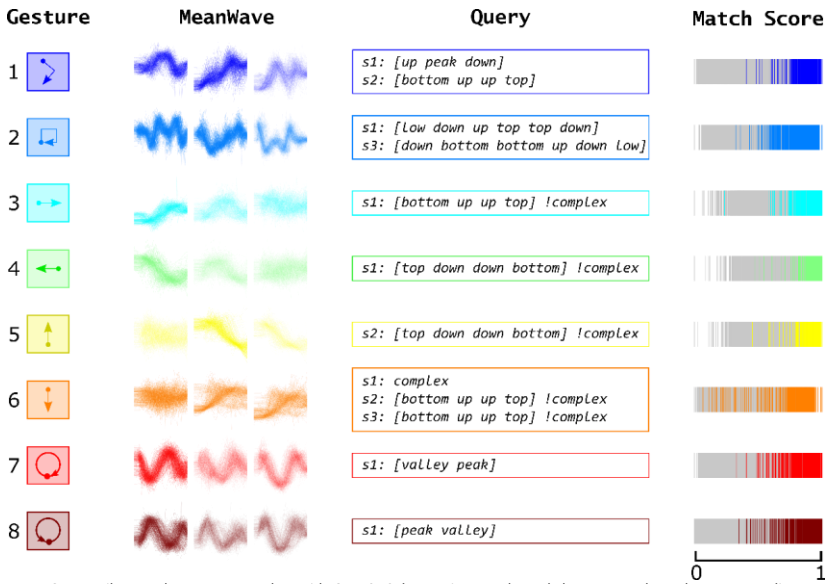

Figure 5 - *UWaveGestureLibrary subsequence matches with QuoTS. Column-wise are showed the gesture class, the corresponding mean wave for all subsequences, the query used to match the subsequence class and the corresponding match score for all subsequences, highlighting with the color of the specific class.*

We not only demonstrate that the system can significantly distinguish gesture patterns, but it does so with very simple and intuitive queries. The ability of the written queries to match the correct class is presented both visually (Figure 5) and quantitatively (Table 1). In Figure 5, the set of eight gestures is described row-wise, having a corresponding mean shape (*MeanWave*) and a query (*Query*) that should filter it. The visual intuition is demonstrated with the barplot (*MatchScore*), which for the sake of readability, highlights the normalized match score ([0-1]) of *subsequences* belonging to the row-wise specific gesture. The other classes are shown in gray. For each gesture, we show the shape for all the available axis (X, Y and Z). We attempt to create queries using only a single axis (X-axis) but used other dimensions when needed.

	G1	G 2	G3	G4	G5	G6	G 7	G8
TP/10	10	10	10	10	10	10	10	10
TP/100	96	100	94	95	74	100	100	99

Table 1 – Results for the top 10 and top 100 sorted gesture classes when using the queries from Figure 5. Here G means Gesture.

To quantify these results, we looked at the top-10 and top-100 matches for each class and counted how many gestures of the selected class were correctly sorted (TP/10 and TP/100). The results are presented in Table 1. These show that the top 10 matches always correspond to the correct class. The reader might think that the first 10 matches might be too easy considering a problem that has around 400 gesture samples per class but note that having more gesture samples also means more opportunities to make mistakes. Moreover, considering the analogy of searching for a webpage with the *Google search engine*, a user will probably be interested in examining at least the top 10 results. Nevertheless, the reader will appreciate that even if we consider the top 100 matches, *Quo*TS still achieved impressive results.

Although we simply wanted to demonstrate that with a set of meaningful words, we can correctly sort each of the classes of this dataset, we also want to highlight that the nature of the classes can be very well expressed by the queries. This is especially evident when we look at the query for gestures that are inverse to each other, such as gestures 7 and 8. Gesture 7 is well matched by the query `[valley peak]`, implicitly, the transposed gesture should have the exact opposite query, which it indeed does (`[peak valley]`). This also occurs for the two other sets of gestures that occur in natural pair; gestures 3 (`[bottom up up top] simple`) - 4 (`[top down down bottom] simple`) and gestures 5 (`[top down down bottom]`) - 6 (`[bottom up up top]`).

We also demonstrate that for the handful cases where this did not work, there was a semantic explanation for it. (e.g. Classes 5 and 6 have not a specific pattern and seem to be specially random in their X-axis or, classes 1 and 2 are very similar, and because of that, are mislabeled). But, as our tool can perform queries in multidimensional data, we can still discriminate these by using the Y-axis in conjunction with X-axis to sort them correctly.

Note that discriminating among these eight classes is not a trivial problem. Of the more than 1,000 papers to have worked with this dataset, the current best accuracy was obtained by COTE algorithm which achieved 76.56% using a single axis. In addition, this dataset was acquired from eight different subjects, which indicates that our system can account for the intra-subject and inter-subject variability in motion for this dataset [13].

4.2 *Quantitative User Behavioral Study*

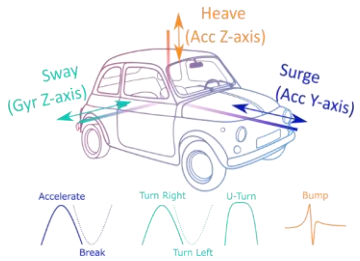

Figure 6 - *Apparatus to acquire inertial data while driving. We used a smartphone with the OpenSignalsApp installed and acquired the accelerometer and gyroscope data. The Y and Z axis of the accelerometer were used to analyze surge and heave motion, while the Z-axis of the gyroscope is used to track the sway motion. On the bottom, we show a reference of shapes for simple events in telemetry.*

4.2.1 Discrete Patterns in Telemetry Data. In order to have discrete labeled examples of typical shapes in the telemetry domain, such as breaking, accelerating, turning, or hitting a bump, we acquired telemetry data while the user was driving in a controlled environment. For the sake of brevity, we summarize the apparatus of the acquisition, made with a smartphone, which axis were oriented as presented in Figure 6. We provide additional resources in [3] that detail the acquisition script used and the driving map.

A selected set of examples are presented in Figure 7. The next section explains how we were able to match the desired patterns with the support of Figure 7.

We begin by considering different types of *breaking* patterns. All of them come with the same basic shape, which is a valley, but have subtle differences. For instance, we might want to search for *sudden breaks* (*emergency stops*). One can imagine they look like a sharp valley with a sudden drastic and high deceleration on the *surge* axis. In our vocabulary, if we simply use the word `valley`, it would possibly allow false positives (by including more gentle breaking patterns). But by simply combining it with the word `high` we can match the desired pattern with high specificity (*Figure 7.SB*).

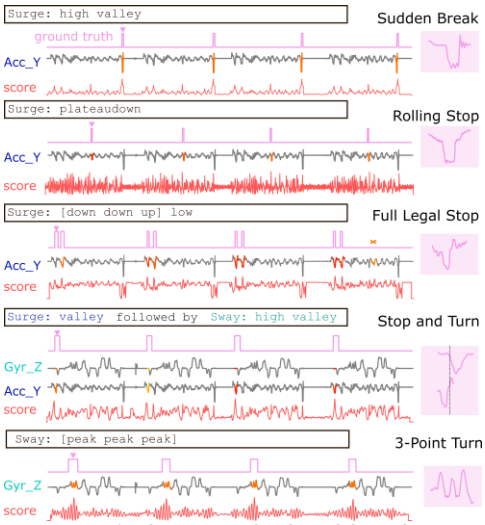

Figure 7 - *Specific subsequence match in driving behavior dataset. Five examples are shared, indicated by the title (Figure7.SB, Figure7.RS, Figure7.FLS, Figure7.ST, Figure7.3PT). The matches are highlighted on the signal (ACC_Y and Gyr_Z). The ground truth is presented on top with a pointer on the subsequence example on the right. The query is presented in a box at the top of each example. The score defined by the query is showed at the bottom of each example. On the side of each example, we enhance the shape we are trying to match with the query. The cross at Figure 7.FLS indicates a False Positive.*

Using the same time scale (5 seconds), we can match a similar breaking pattern but with a lower deceleration, the *rolling stop* (*Figure 7.RS*). For this pattern, we have to select a shorter timescale (2.5 seconds) and use the word `plateaudow`, which exactly matches the shape we are looking for, as it looks like a plateau.

This breaking pattern distinguishes itself from a *full legal stop* by its shape and duration in time (*Figure 7.FLS and 7.RS*). Therefore, to match a *full legal stop*, we used a larger time scale (10 seconds) with the query `[down down up] low`, indicating a `valley` that has a longer downward phase and with a lower amplitude to reject the high valley of the *sudden break*. In the scoring function, it comes clear that the `low` keyword reduces the score for the *sudden break* event.

Considering a *full stop* or any other breaking pattern, we might be interested in understanding the subsequent actions of the driver. For instance, we might ask "*did the driver stopped because of a red light, does she/he turned left, right or kept in front?*". In these cases, we need a time dependency operator, such as `followed by`. In addition, we need to consider multiple signals (breaking – *surge* axis; turning – *sway* axis). In this example, we show how *Quo*TS can sort subsequences that reflect a breaking pattern followed by a right turn with the query `Surge: valley followed by Sway: high valley`. To be clear, when using the query `Surge: valley`, we match any breaking pattern, whereas by including followed by `Sway: high valley`, further filters the signal by rewarding subsequences that show a breaking pattern but with a valley on the next subsequence window of the *sway* axis.

The final example shows how expressive *Quo*TS is. One might imagine what a three pointer might look like: we 1: forward sway left, 2: backward sway left and 3:forward sway left again. This sequence would be represented by 3 followed peaks. *Quo*TS matches it very well with the query `Sway:[peak peak peak]`, which translates into 3 followed peaks.

4.2.2 QuoTS also matches the driving behaviors. In addition to search for specific atomic patterns in telemetry data, *Quo*TS can be helpful to search for driving patterns associated with higher-level behavioral patterns, traffic congestion or road surface condition. One of the most traditional problems in telemetry is understanding the driver's aggressive behavior.

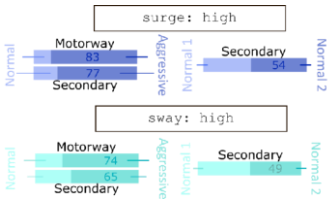

Figure 8 – *Higher level analysis with QuoTS for distinguishing between aggressive and normal driving. The signals compared come from the same driver, automobile and route. The comparison is made for surge and sway, while also showing different routes (Motorway and Secondary). As two normal behavior signals were available for each driver, normal1 and normal2 behaviors were also compared.*

In this example, we used a public dataset [21] to perform additional performance analysis of *Quo*TS. The main question was to use the labelled dataset to identify high level differences between behavioral classes: aggressive vs normal. *Aggressiveness* is described as having sudden and brusque changes in surge and sway [21]. We searched for the top 100 subsequences corresponding to the word `high` for a signal with *normal* and *aggressive* behavior, from the same subject, automobile, and route. The results are promising, since, as presented in Figure 8, the number of events detected with a single English word tends to be from the telemetry labeled as aggressive for both surge and sway, and in both motorway and secondary road condition. These results are also consistent for all six drivers that participated in the study. Results for *Normal1 vs Normal2* also show that close to half of the events are shared between signals when compared. In addition to this, we may understand how the change in aggressiveness occurred, if by breaking/accelerating more suddenly, or stirring more sharply.

4.2.3 QuoTS can be used for Puppetry

Figure 9 – *Creating query prompt data by puppeteering a car model. The puppet data for a 3-point turn (smoothed with 10 samples moving average) is presented on the left, while the real data from a real car is presented on the right.*

We believe that *Quo*TS is intuitive enough to allow most novice users to create simple effective queries for most information retrieval tasks. However, in some cases, the user's ability to formulate queries may be a limitation. In this section we show that, perhaps uniquely, this domain allows us to generate queries by "puppeteering" a model car. To demonstrate this, we attached a smartphone to a model car (as indicated in Figure 9) and acquired inertial data while mimicking a 3-point turn event on a table. The resulting shape is illustrated in *puppet data*. We believe this can be used in two ways: for the user to gain intuition as to how maneuvers map to telemetry or the inertial data from the model car can be used as a MASS template.

- *Shape Intuition*: A user might not know what the shape of a 3-point turn looks like. By using a model car, she can mimic the motion and gain intuition over what the query should be (in this case, `[peak peak peak]`).
- *MASS template:* As mentioned above, we have a special *shape* word, which corresponds to a shape template given by the user, to which a word can be assigned. We then can use the word in our language to match desired patterns with the MASS distance profile as a *word feature vector*.

We added the *puppet data* and the word *3pointturn* to our vocabulary. Searching for it in our dataset matched all the desired shapes. Note that the template has a different amplitude and time scale, but for MASS, the z-normalization makes the amplitude difference irrelevant, and the time scale was adjusted by resampling the template based on the selected window size for the query search. With this, the template is not *only* a template, it also becomes part of the language, being used in combination with all the other words available in our vocabulary.

4.2.1 Speed Performance

We tested the speed limits of *Quo*TS in calculating the score function for each of the used queries in Figure 7. We calculated the average time of 10 searches with each query. Figure 10 shows the results for the query search in increasing signal sizes (up to 1 million datapoints ~ 2.5 hours at 100 Hz) and increasing window sizes for the desired query (up to 1500 samples). The change in signal size shows that the query mechanism is almost linear in this range and always below 0.06 seconds, which in human response can be considered real-time. The change in the query window size was not significant.

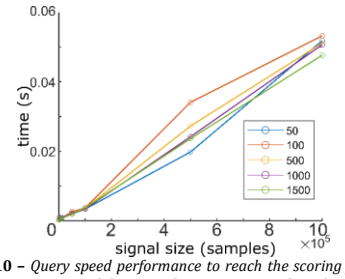

Figure 10 – *Query speed performance to reach the scoring function. The time was measured for several queries in signals with increasing length, for several query window sizes.*

5 Conclusion and Future Work

We introduced *Quo*TS, a system to search for patterns in time series with expressive natural language queries. We demonstrated its expressiveness in describing patterns with several datasets and domains, particularly for identifying discrete driving behaviors. We showed how *Quo*TS is more flexible and "forgiving" for searching driving patterns than traditional distance measures, such as Euclidean distance. In addition, we also demonstrated that *Quo*TS can be useful for higher level data exploration in telemetric data, discovering aggressive driving behavior, with a single word query. All this was achieved using a total of ten words of our current vocabulary of size twenty-two and the four operators. We also showed that there is the potential for a unique interaction, by puppeteering a model car to gain intuition over the query or use the example by adding it to the vocabulary. This expressiveness and flexibility does not incur significant computational overhead, even our unoptimized current version of the tool can handle one hour of driving datasets with subsecond queries, since the data is pre-computed for the posterior search process. This benefits the interactive power of the tool for exploratory analysis. Still, for every change in window size for the search, the word feature vectors set has to be recomputed, being a current limitation that should be addressed in the future.

The ability of the tool to work with MTS is very relevant, especially for our planed future work in datasets that include physiological data from the driver as well, for which analysis of relationships between the car, the driver and its surroundings [9,23], or for perceived risks of drivers in automated vehicles[12]. We intend to improve the interactive power of the tool, namely by adding more relevant words or operators, and create context-based words. The usage of the tool will be expanded to other datasets for other evaluations, such as driver's drowsiness, road surface conditions and other discrete driving patterns.

REFERENCES

1. Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. 2017. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery* 31, 3: 606–660. https://doi.org/10.1007/s10618-016-0483-9
2. Gustavo E. A. P. A. Batista, Eamonn J. Keogh, Oben Moses Tataw, and Vinícius M. A. de Souza. 2014. CID: an efficient complexity-invariant distance for time series. *Data Mining and Knowledge Discovery* 28, 3: 634–669. https://doi.org/10.1007/s10618-013-0312-3
3. BlindReview. 2022. QuoTS. *https://github.com/Anonymous14151/QuoTS*.
4. Zhitong Cui, Hebo Gong, Yanan Wang, Chengyi Shen, Wenyin Zou, and Shijian Luo. 2021. Enhancing Interactions for In-Car Voice User Interface with Gestural Input on the Steering Wheel. In *13th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 59–68. https://doi.org/10.1145/3409118.3475126
5. Pedro Domingos. 2012. A few useful things to know about machine learning. *Communications of the ACM* 55, 10: 78–87. https://doi.org/10.1145/2347736.2347755
6. Shima Imani, Sara Alaee, and Eamonn Keogh. 2019. Putting the human in the time series analytics loop. *The Web Conference 2019 - Companion of the World Wide Web Conference, WWW 2019*: 635–644. https://doi.org/10.1145/3308560.3317308
7. Shima Imani, Sara Alaee, and Eamonn Keogh. 2021. Qute: Query by Text Search for Time Series Data. . 412–427. https://doi.org/10.1007/978-3-030-63089-8_27
8. Christian Kaiser, Alexander Stocker, Gianluigi Viscusi, Michael Fellmann, and Alexander Richter. 2021. Conceptualising value creation in data-driven services: The case of vehicle data. *International Journal of Information Management* 59: 102335. https://doi.org/10.1016/j.ijinfomgt.2021.102335
9. Nataliya Kosmyna, Caitlin Morris, Thanh Nguyen, Sebastian Zepf, Javier Hernandez, and Pattie Maes. 2019. AttentivU. In *Proceedings of the 11th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 355–368. https://doi.org/10.1145/3342197.3344516
10. Thomas Kundinger, Andreas Riener, and Ramyashree Bhat. 2021. Performance and Acceptance Evaluation of a Driver Drowsiness Detection System based on Smart Wearables. In *13th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 49–58. https://doi.org/10.1145/3409118.3475141
11. Jingyi Li, Agnes Reda, and Andreas Butz. 2021. Queasy Rider: How Head Movements Influence Motion Sickness in Passenger Use of Head-Mounted Displays. In *13th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 28–38. https://doi.org/10.1145/3409118.3475137
12. Mengyao Li, Brittany E. Holthausen, Rachel E. Stuck, and Bruce N. Walker. 2019. No Risk No Trust. In *Proceedings of the 11th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 177–185. https://doi.org/10.1145/3342197.3344525
13. Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. 2009. uWave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing* 5, 6: 657–675. https://doi.org/10.1016/j.pmcj.2009.07.007
14. Yu-Luen Ma, Xiaoyu Zhu, Xianbiao Hu, and Yi-Chang Chiu. 2018. The use of context-sensitive insurance telematics data in auto insurance rate making. *Transportation Research Part A: Policy and Practice* 113: 243–258. https://doi.org/10.1016/j.tra.2018.04.013
15. Miro Mannino and Azza Abouzied. 2018. Qetch: Time series querying with expressive sketches. *Proceedings of the ACM SIGMOD International Conference on Management of Data*: 1741–1744. https://doi.org/10.1145/3183713.3193547
16. Prithiviraj K. Muthumanickam, Katerina Vrotsou, Matthew Cooper, and Jimmy Johansson. 2016. Shape grammar extraction for efficient query-by-sketch pattern matching in long time series. In *2016 IEEE Conference on Visual Analytics Science and Technology (VAST)*, 121–130. https://doi.org/10.1109/VAST.2016.7883518
17. Syeda N. Z. Naqvi and Sofia Yfantidou. 2017. Time Series Database and InfluxDB. *Advanced Databases*.
18. R. Agrawal, G. Psaila, E. Wimmers, and M. Zaït. 1995. Querying Shapes of Histories. In *Proceedings of the 21th International Conference on Very Large Data Bases*.
19. Imke Reimers and Benjamin R. Shiller. 2019. The Impacts of Telematics on Competition and Consumer Behavior in Insurance. *The Journal of Law and Economics* 62, 4: 613–632. https://doi.org/10.1086/705119
20. João Rodrigues, Duarte Folgado, David Belo, and Hugo Gamboa. 2019. SSTS: A syntactic tool for pattern search on time series. *Information Processing & Management* 56, 1: 61–76. https://doi.org/10.1016/j.ipm.2018.09.001
21. Eduardo Romera, Luis M. Bergasa, and Roberto Arroyo. 2016. Need data for driver behaviour analysis? Presenting the public UAH-DriveSet. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 387–392. https://doi.org/10.1109/ITSC.2016.7795584
22. Sergio M. Savaresi, Vincenzo Manzoni, Andrea Corti, and Pietro de Luca. 2010. Estimation of the Driver-Style Economy and Safety Via Inertial Measurements. In *Advanced Microsystems for Automotive Applications 2010*. Springer Berlin Heidelberg, Berlin, Heidelberg, 121–129. https://doi.org/10.1007/978-3-642-16362-3_13
23. Stefan Schneegass, Bastian Pfleging, Nora Broy, Albrecht Schmidt, and Frederik Heinrich. 2013. A data set of real world driving to assess driver workload. In *Proceedings of*

the 5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications - AutomotiveUI '13, 150–157. https://doi.org/10.1145/2516540.2516561

24. Gözel Shakeri, John H. Williamson, and Stephen Brewster. 2017. Novel Multimodal Feedback Techniques for In-Car Mid-Air Gesture Interaction. In *Proceedings of the 9th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 84–93. https://doi.org/10.1145/3122986.3123011
25. Thu-Huong Ha and Nikhil Sonnad. 2022. How do you draw a circle? We analyzed 100,000 drawings to show how culture shapes our instincts. *https://qz.com/994486/the-way-you-draw-circles-says-a-lot-about-you/*.
26. Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. 2013. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery* 26, 2: 275–309. https://doi.org/10.1007/s10618-012-0250-5
27. Abdullah Mueen, Sheng Zhong, Yan Zhu, Michael Yeh, Kaveh Kamgar, Krishnamurthy Viswanathan, Chetan Kumar Gupta and Eamonn Keogh (2022), The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance

A. Supplemental Material

We have taken great pains to ensure that our paper is self-contained. Here we simply present resources that help make our claims open-source and reproducible.

A.1 Code

In order to provide evidence of the results and make this work reproducible, we share the repository where you will find all the code used to compute all examples presented: https://github.com/Anonymous14151/QuoTS

The code has the Matlab interface that we designed to run the results, which the reviewers can load and try out.

A.2 Experiment it with your data

We also took the liberty to design a Python version of the same app and make it available online: https://jmdrodrigues-pynlts-streamlit-main-aou28r.streamlit.app/

With this app, the reviewers can upload their own data and try out the set of keywords and operators from our vocabulary.

You can have a better usage of the app if you install it following the instructions at:

https://github.com/JmdRodrigues/PyNLTS

A.3 Detail the Examples

All examples presented in this paper are reproducible. Here we show each one of them (which can be validated using the matlab app or the Live Scripts):

|*How to compute all word feature vectors (Figure 4)* https://github.com/Anonymous14151/QuoTS/blob/main/examples/wordfv_match.md

|*All Telemetry examples from (Figure 7)* https://github.com/Anonymous14151/QuoTS/blob/main/examples/telemetry_examples.md

|*Describing higher level behaviors (Figure 8)* https://github.com/Anonymous14151/QuoTS/blob/main/examples/higher_level_behavior.md

|*Check Time Performances (Figure 10)* https://github.com/Anonymous14151/QuoTS/blob/main/examples/time_performance.md

A4. Video demonstrating the Matlab App

In order to have a visual output that shows the usage of the proposed tool, we also share a short video with simple examples at:

https://github.com/Anonymous14151/QuoTS/tree/main/examples

under the file *demonstration_video.mp4*

B. Datasets

We reinforce our claim to make this work reproducible by sharing all the data we recorded or borrowed.

B1. UCR Benchmark

Several examples were made with data from the UCR time series classification benchmark, namely *Trace* and *UWaveGesture*, which can be found here: https://github.com/Anonymous14151/QuoTS/tree/main/datasets/UCR

and here: http://www.timeseriesclassification.com/dataset.php

B2. Own Telemetry Examples

You can also find the dataset we recorded for the telemetry examples (Figure 7) here: https://github.com/Anonymous14151/QuoTS/tree/main/datasets/example3/parking_lot_controlled_environment

The map with place and trajectory with the main driving events are also displayed in the following image: https://github.com/Anonymous14151/QuoTS/blob/main/datasets/example3/acquisition_script_sequence_of_events.png

B3. Behavioral Telemetry Examples

The dataset used for behavioral analysis (Figure 8) can be found here: http://www.robesafe.uah.es/personal/eduardo.romera/uah-driveset/