Visualization - Used in the LCC

Visualization messages can be sent to the LCC to draw basic shapes and text in the map view, where the vehicles are shown as well. An example use case for this would be to draw the desired path that a vehicle is supposed to take on command, to check if it actually follows this path. The messages can also be used to show additional information to the running program or simply to debug the sent trajectory by checking the created path visually.

Data structure

Each visualization message thus consists of seven data fields which need to be filled accordingly to draw the desired information in the LCC:

- ID
- visualization type
- time_to_live
- points
- size
- string_message
- string_message_anchor
- color.

ID

Each visualization command is identified by a **unique** ID. Choose different IDs for commands that should be displayed at the same time, or use the **same ID to override** an older visualization message.

The **highest ID is always drawn last**, so, if you want to draw symbols on top of each other, always make sure that you use the right order of IDs.

Visualization Type

Currently, four command types are supported:

- Line
- Polygon
- Filled circle
- String

Lines and polygons are defined by the points which are set in the structure data as well. In both cases, lines are drawn between subsequent points in the array. The first and the last point are also connected by a line if you want to draw a polygon.

If you choose a string, you need to specify a single point as anchor. The drawn string will be attached to this anchor as specified in the field `string_message_anchor`. A string message will only be displayed, if you set the field _String Message_.

A **filled circle** requires a single point which will be its center.

Time to Live

Each visualization command is **removed automatically** after a given time. Here, you can set the time to live in nanoseconds.

Points

This field defines the set of points between which lines should be drawn, or the point where the string message should be shown. The coordinate system is the same as the one used for the vehicles.

Points is a custom IDL data type which is a struct `Point2D { double x; double y}`. Points are set according to the coordinate system.

Size

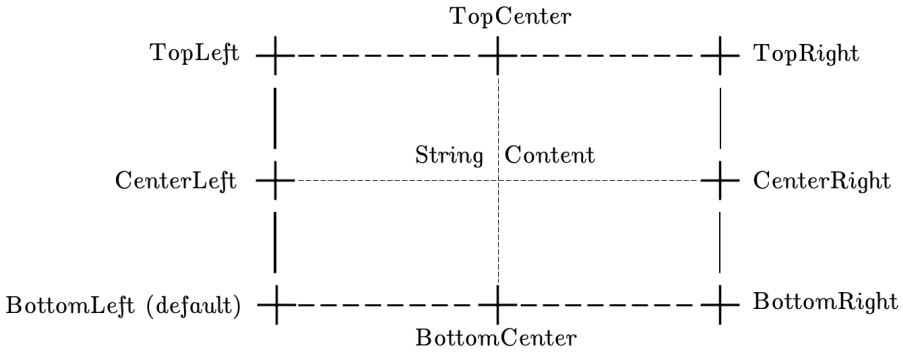
Polygons and lines interpret the size as **line width**, string messages as **font size**, and circles as **radius**.

String Message

Just set a string for this field (as RTI uses its own string format, be aware that setting this value with a std::string data type might lead to an error message, but give it a try before you use RTI's own implementation).

String Message Anchor

Defines where a string message will be aligned, relative to the anchor point (the first entry of `points`). You only have to specify this field manually, if it should be different from the **default**: **BottomLeft** corner.



Center is also a possible value.

Color

Lines, polygons and strings can be given any color within the RGB spectrum.

Color is a custom IDL data type which is a struct `Color {octet a; octet r; octet g; octet b}`. Color values are just **regular RGB** values. The **transparency** value should be **ignored**.

Usage example

https://github.com/embedded-software-laboratory/cpm_lab/blob/master/lab_control_center/test/VisualizationTest.cpp