FRAUNHOFER INSTITUTE FOR COMMUNICATION, INFORMATION PROCESSING AND ERGONOMICS, FKIE

FIELD TEST TOOL OPERATION MANUAL

Prepared by: Carlos Tampier Cotoras

Table of contents

1
Introduction

The Field Test Tool (FTT) is composed of five software modules: a database, an automatic report generator, a web server, a ROS interface, and a user interface. A diagram of the system's architecture is shown in Fig. 1.

This document focuses on explaining the usage of the interface components. That is, the user interface and the ROS interface. For an overview of the complete system, installation and execution instructions, please read the README.md file in the project repository.
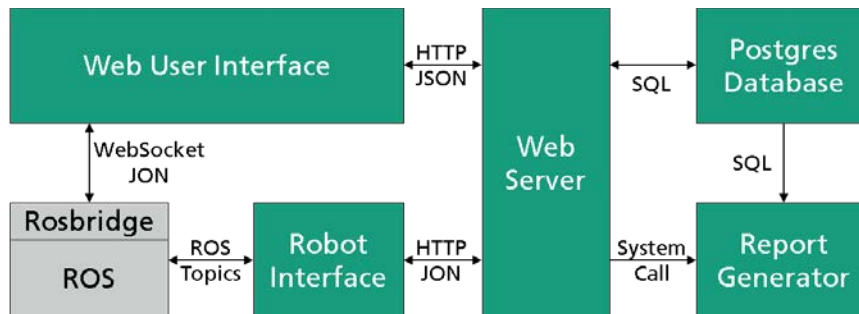

Fig. 1   System's architecture.

This system is designed to store the robot operation mode and its position, along with other context data that provides insight of the working conditions. It should be noted, that the operational data is stored in the database under a tree-layer logging structure, namely:

- Level 1: Test Event
- Level 2: Shift
- Level 3: Leg

The role of each one will become clearer with the explanation of the user interface further down in this document. Furthermore, each time the autonomous system's operation mode changes, a new entry, called "Segment", is created in the database. Segments define the portions of the robot path. Therefore, the complete robot trajectory of a system's test run will be divided into these segments. One for every mode change.

2
ROS Interface

The ROS interface is a component that needs to be set up and integrated with the autonomous system to be tested[1]. To properly carry out this integration, the following should be noted.

The "ros2api" ROS node subscribes to topics and listens to the robot TF in order to send HTTP POST requests to the database API. The node has Python and C++ implementations. Originally implemented in Python, the node was ported to C++ in order to improve its performance. The gathered data includes robot's operation mode (manual/autonomous), its position, and optionally, images from an onboard camera.

The FTT can work with either GNSS data or locally referenced position data. If working with the latter, a map of the environment is also required.

Tab. 1 summarizes the data acquired from the robot environment, along with some relevant comments. The parameters of the launcher file of this node (remap and parameter arguments) are described in the README.md file.

Topic message type	Topic default name	POST request	Notes
industrial_msgs/ RobotMode	robot_mode	segment	Mandatory. A new segment creation request is issued every time the message's "val" variable switches between 1 (manual) and 2 (auto).
sensor_msgs/ NavSatFix	robot_position	pose	Mandatory if working with global positions. POST requests sent on a timed-based schedule.
sensor_msgs/ Image **or** sensor_msgs/ CompressedImage	image **or** image_com- pressed	image	Optional. The stored buffer of images or compressed images are sent after the creation of a new segment. The **C++** implementation uses only the "image" topic with an ImageTransport subscriber.
nav_msgs/ OccupancyGrid	map	map_image	Mandatory if working with local positions. POST requests sent on a timed-based schedule (only if a new message arrives to the subscriber).
sensor_msgs/ CompressedImage	map_jpeg	map_image	Optional. If a map image is received between POST requests, this image is used instead of transforming the actual map from the map topic. Only in the **Python** implementation.
geometry_msgs/ TransformStamped[] **or** geometry_msgs/ PoseStamped	/tf ("map" to "base_link") **or** local_pose	local_pose	Mandatory if working with local positions. POST requests sent on a timed-based schedule. A parameter controls whether the position is obtained from the TFs or the topic.

Tab. 1 Data acquired from the ROS environment.

---

[1] As of version 2.0, the ROS configuration can be done through the web user interface.

3
Using the web GUI

3.1   Starting view

The web GUI allows the human operator to interact and manipulate the data stored in the database. The inclusion of relevant context information about the autonomous system's condition during testing, through the web GUI, aids posterior analysis. The GUI also controls the starting and finishing of new records. The initial view of the web application is shown in Fig. 2. Three log levels are used to store the robot' data: a "Test Event", a "Shift", and a "Leg".

Fig. 2   Starting view of the FTT web GUI.

3.2   Configuration

The first time using the tool, some configuration data will need to be inputted into the database through the web GUI. Clicking on the top left cog of the main window will open a new tab in your browser (Fig. 3), where data for the tables "performer", "personnel", "pose_source", and "vehicles" can be entered.

Fig. 3   Configuration view of the FTT web GUI.

Updating the parameters of the FTT ROS interface node running in the robot is also possible from the configuration window (alternatively from the config yaml file, as explained in the README.md file). The connection between the web GUI and the robot's ROS system is implemented using the "rosbridge_suite" ROS package and "roslibjs" JavaScript library.

A section of the configuration web page is dedicated to the updating of ROS parameters, as previously shown in Fig. 3 and shown again in extension in Fig. 4. An icon to the right of the section's title show the connection status between the web GUI and ROS. If not connected, clicking on the icon will attempt to establish a connection. A connection is required to show and edit the ROS parameters. The edition of ROS parameters is executed directly in the ROS parameter server. For permanent storage of edited parameters, a "save" button located at the end of the tables will call a service to write the parameters to the ROS node's configuration file.

Fig. 4   Configuration of ROS parameters through the FTT web GUI.

3.3   Creating the log entries

Once the configuration data has been saved, you can go back to the main page. In order to use the application, a user must first be selected. Clicking on the top right user icon will deploy a selection list, as shown in Fig. 5. Then a new entry for every logging level can be created by pressing on the "New" button. A dialog will open each time, as can be seen in Fig. 6, Fig. 7 and Fig. 8. The same dialog will show when the "edit" button is pressed. Each dialog allows the user to input relevant context information about the system's test.

A "Test Event" entry has fields for the test location, version and time zone, as well as a field to input any additional comment. A "Shift" entry allows the identification of the test performer, as well as the involved personnel. The test intent, a workspace description and the used vehicle can also be added. A field for any additional comments is also provided here. A "Leg" entry is the last log identifier for the recording instance. The current weather, configured position data source, and additional comments are entered here.

Fig. 5   User selection.

Fig. 6   Test event creation/editing dialog.

Fig. 7   Shift creation/editing dialog.

Fig. 8   Leg creation/editing dialog.

3.4　Fetching the field data

Once all log levels have been selected/created, the available field data in the database will load automatically. The appearance of the GUI for a newly created logging instance is depicted in Fig. 9. If a direct connection to the robot's ROS environment can be established, the connection icon next to the toolbar`s title will be displayed in green and the "Start Logging" button will become active. Should the connection to ROS be established later, a click on the icon will attempt a connection anew.
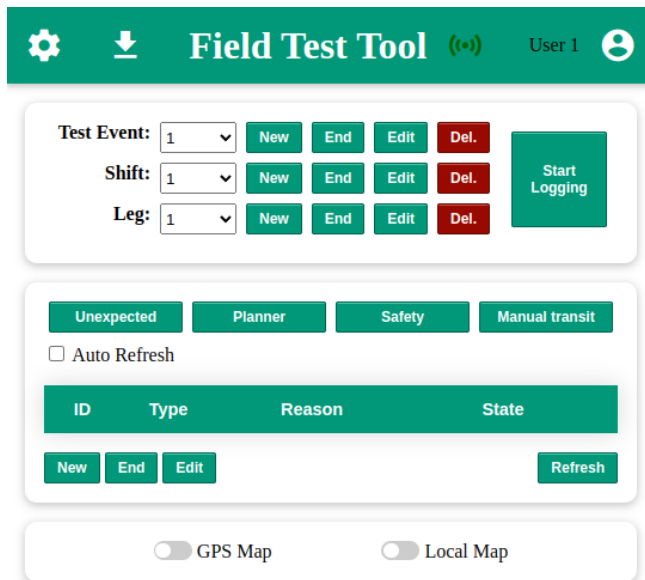
Fig. 9　View of the GUI ready to start a recording.

Before starting the autonomous system's test, it is recommended to check the "Auto Refresh" box and activate the map display (either global or local), as shown in Fig. 10.
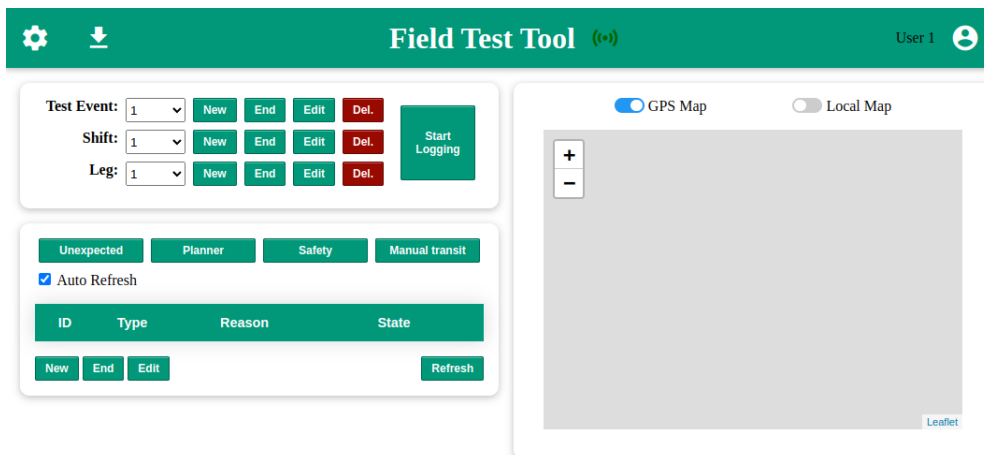

Fig. 10　View of the GUI with auto refresh and active map.

A click on the "Start Logging" button triggers a function call to the FTT ROS interface node to start its subscribers. The ROS node should start creating data entries in the database as the robot performs the intended tests. With the "Auto Refresh" box checked, the GUI will update every 2 seconds, showing the robot position in the map, and a list

of all segments. An example with both global and local data display is shown in Fig. 11. In this example, Clearpath's Husky ROS stack was used in simulation. In Fig. 11, the local map is also activated to be shown as a layer with transparency on top of the global map. This is an experimental feature, only available when both the global and local map are displayed simultaneously, and it can be shown or hidden using the top right "layers" control icon in the top right corner of the global map.
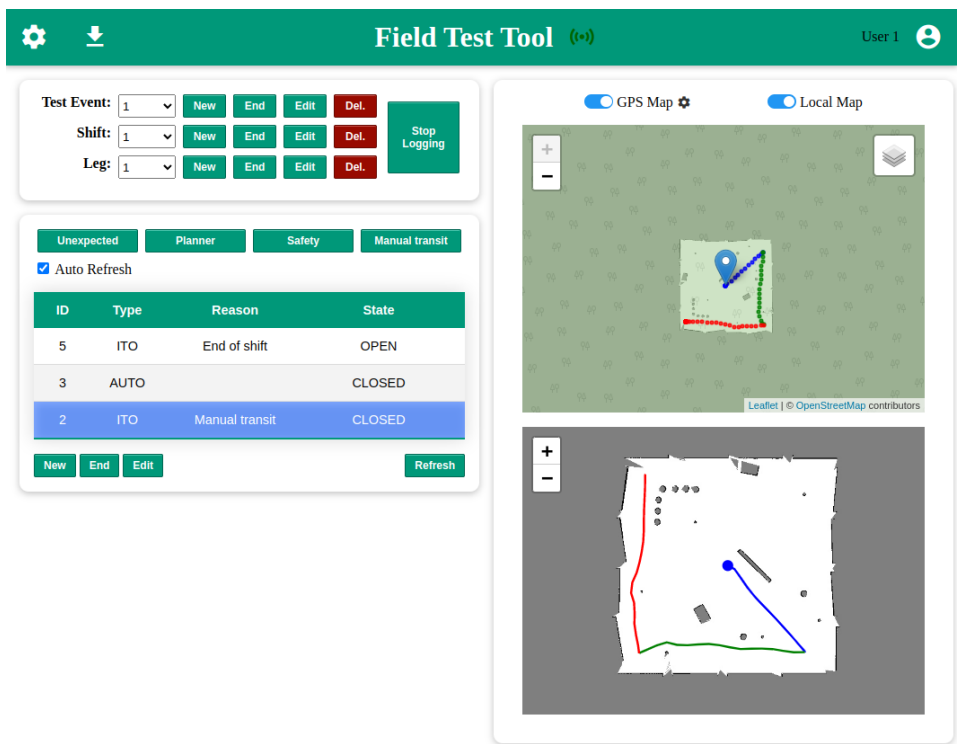

Fig. 11   View of the GUI during the autonomous system's test with global and local position data. The superimposition of the local map on top of the global map is an experimental feature and was activated for this example.

3.5 Creating and editing segments

Manual segments can also be added through the GUI by the user. It is recommended, though, that segment creation is mainly done by the ROS interface node. Nevertheless, once the robot is running in manual mode, the creation of a second manual segment might be desired, in order to register a new event which forces the robot to stay in manual mode for longer than expected for the original transition cause.

Transition causes (ITO Reason) can be quickly assigned to manual segments by selecting one from the list and then clicking on one of the four buttons over it: "Unexpected", "Planner", "Safety" and "Manual Transit". An extended list of autonomous to manual transition reasons can be found when clicking on the "Edit" button at the bottom of the segment list. Doing this will display the editing dialog for the selected manual segment. Fig. 12 shows the mentioned dialog. Further information, such as annotation of obstacles, lighting conditions or terrain slope can be added here. Several independent notes can also be added to the selected segment and pictures can be uploaded from the user's device.

Fig. 12   Segment creation/editing dialog.

3.6 Closing and re-opening the logs

Once the test is finished, the recording instance should be closed by the user. The proper procedure considers the user first stopping the FTT ROS interface node and then ending the corresponding log level. The ROS node is stopped when clicking on the "Stop Logging" button (which will then change back to "Start Logging" as shown in Fig. 13). Stopping the ROS node will automatically close any open segment.
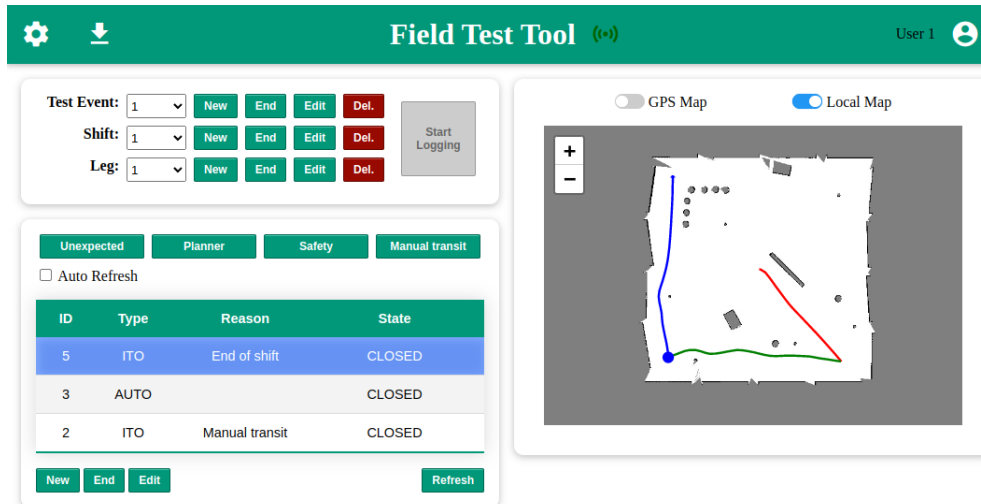

Fig. 13 View of the GUI after stopping the data recording.

If no further trial runs are planned, the user can then close the "Test Event" log level, by clicking on the corresponding "End" button. "Closing" a log level or a segment means assigning an "end time" to the corresponding entry in the database. This information is then used in the report generator. Closing any log level will automatically close any entries below it (e.g. closing the "Test Event" will close any open shift, leg and segments). Once closed, no new data coming from the robot can be added (regardless of whether the ROS node was stopped or not). Nonetheless, the user can still modify the additional context information.

Note that segments can also be closed manually from the GUI. This will only close the portion of the segment storing the "ITO Reason", the database will still keep the "main" manual segment open until a new autonomous segment is created, the ROS node is stopped, or the "Leg" log level is closed. Internally, this is implemented with manual segments having "child" manual segments. The parent segment stores the position data and images, while the children store the transition reason.
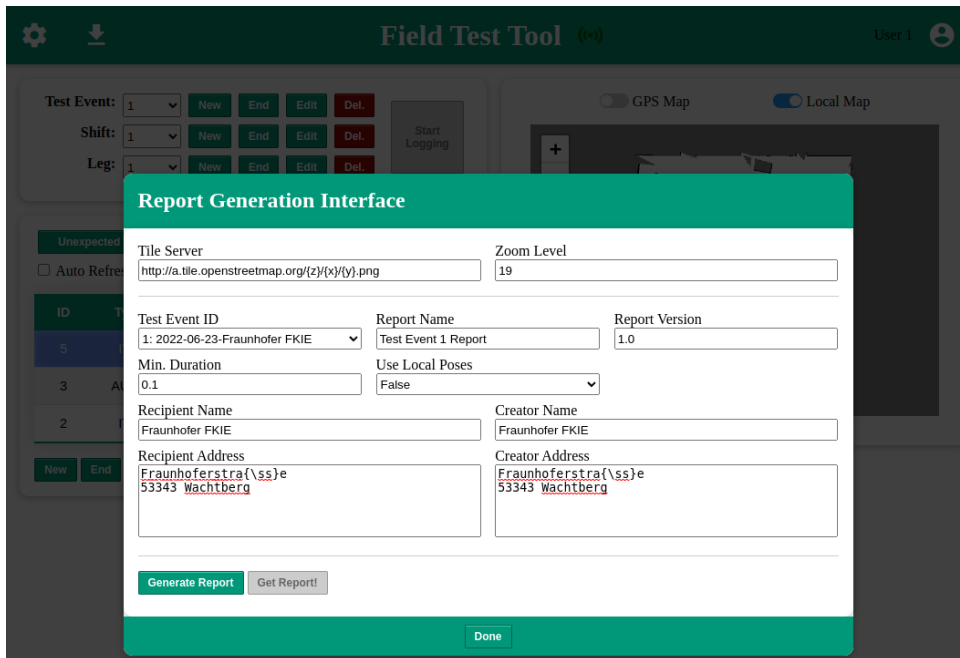
In order for the automatic report generator to work properly, the "Test Event" must be closed before invoking the report generator's script.

When a "Test Event" is closed, the corresponding "End" button will change its signature to "Open". Clicking this button will then delete the end time for the "Test Event", thus enabling the creation of new "Shifts" and consequently the recording of new data under a previously created log entry. This is particularly useful when conducting experiments over several days in many different locations. Please note that only "Test Events" can be re-opened.

3.7   Generating the report

The report generator can be called directly from the web GUI (alternatively from the command line, as explained in the README.md file). Clicking on the download icon, next to the cog, at the top left corner of the web page, will open the report generator dialog, as shown in Fig. 14. All input fields must be filled. Then, clicking on the "Generate Report" button will issue the call to the report generating script. Once finished, the "Get Report!" button will become available and clicking on it will open a new tab with the report in PDF format.
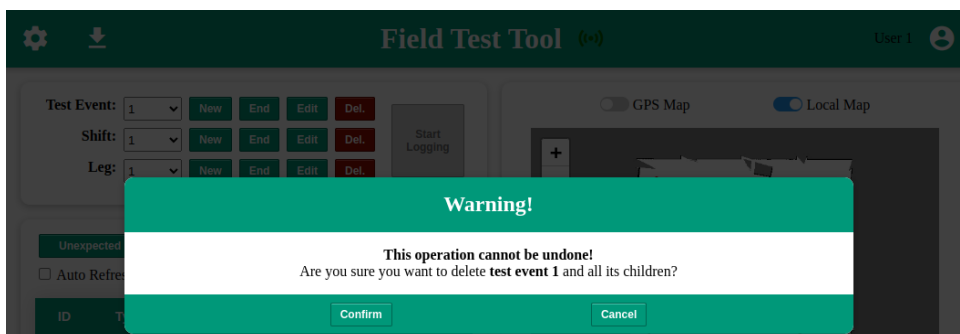
Fig. 14   Report generator dialog.

3.8   Deleting a log

If the data is no longer required or an entry has been created by mistake, it can be deleted with the "Del." button next to the entry. When pressed, a confirmation overlay will appear, after which the entry and all data depending on it will be deleted.

Fig. 15   Log entry deletion dialog.