Interface development for environment's monitoring and exploring in the system of RESCUER

Dimitar Mladenov Dimitrov
Interface development for environment’s monitoring and exploring in the system of RESCUER

Project written by Dimitar Mladenov Dimitrov

University of Växjö       June 2006
Abstract
This project presents a model of monitoring GUI, built for human search and rescue system, part of RESCUER system. This system is an innovative system with main task decreasing the human involvement in human search and rescue or another type of dangerous missions. In this project we try to make functional and easy to use interface that provide the most necessary information on one screen, such as map of the investigated area, human presence, environment status and status of the most important sensors of the robocar. We also present two already developed interfaces solving similar to ours problem and compare our interface with them and also we present the way of building our GUI on Matlab environment. The final result is an interface, combining the advantages of the other two investigated GUIs and visualises the risk zones, different types of risks that can be expected in the investigated area.

Keywords
Monitoring, GUI, Interface, RESCUER, ERMMAS, Visualisation, System.
1 Introduction

The prices of computer systems nowadays are decreasing every day. Every day our tasks are getting more and more complicated. We cannot manage them so easy or we need a lot of time to solve for example not that difficult problem. This is a good reason for the people to use a lot of computer systems.

There are a lot of different areas of using computer systems for example production (like automation), medicine, scientific researches, robotics and so on. Especially in the robotics area it is impossible to manage without using computer systems.

Robots can assist human teams in urban search and rescue (USAR) tasks by traveling into dangerous and small areas to search for survivors. Robots need to climb on and maneuver round rubble piles that are deemed too dangerous or hazardous for human or animal searchers to investigate. In difficult USAR environments, it is not yet possible to create a fully autonomous robot to completely take the place of a human rescue worker. In fact, most USAR robots that are sent into disaster zones are teleoperated.

1.1 Problem

This project has the mission to develop an interface for monitoring for a human search and rescue system. The main tasks for this interface are accuracy, fast reaction on the passed signals and good functionality. At this moment there are some similar projects, that are already developed and we will refer to them more detailed later in this report. All of them are specialized either for urban search and rescue or for bomb disposal. In this project we use the advantages of the already developed systems to make one more functional interface that can be used for civil protection rescue mission scenarios.
2 Background

This project, part of RESCUER system is part of 6th Framework program project, financed by the European Union.

In this chapter, the systems RESCUER and ERMMAS are presented, that the current project participates in.

2.1 RESCUER Overview

The RESCUER project focuses on the development of an intelligent Information and Communication Technology and mechatronic Emergency Risk Management tool and on testing it in five Improvised Explosive Device Disposal, and Civil Protection Rescue Mission scenarios [2]. One of the main tasks of RESCUER is to decrease the human involvement solving too risky and dangerous for humans tasks. The control will be accomplished by remote methods which doesn’t make humans to go to the dangerous areas. Using the remote control will significantly enhance the speed and the accuracy of range of solving hazardous situations.

As an intelligent mechatronic system, RESCUER combines electronic systems, computers, control and mechanical systems like shown on the figure 2.1.

![Figure 2.1 Mechatronic system structure](image)

Another presentation of mechatronic system is presented on the figure 2.2 below. The main difference is that on figure 2.2 includes the user as an end consumer of the system and the hierarchy.
Figure 2.2 Mechatronic system hierarchies

RESCUER mechatronic system will include:
- sensor for detection of CO2;
- acoustic sensor;
- sensor for detection and measurement of nuclear radiation;
- sensor for gas analysis; sensor for detection of flammable materials;
- sensor for detection of explosive materials;
- sensor for detection electronic components;
- tactile sensors;
- stereo vision system;
- a pair of intelligently controlled robots with dextrous grippers (combination of teleoperating [through exoskeleton, data glove and joystick] and autonomy [through intelligent control based on the vision and tactile sensory input]);
- emergency risk management monitoring and advising system (ERMMAS);
- shared control (autonomy and remote teleoperating) vehicle (a mobile platform rover type with high manoeuvrability); mobile control centre for control and monitoring of the mechatronic system and the risk process;
- and advanced information and communication facilities (secure multi-channel wireless communication between the mechatronic system and the mobile control centre).

All this will make the system better for human search and rescue in different environment and different hazards.

The existing remotely controlled Improvised Explosive Device Disposal (IEDD) and Explosive Ordnance Disposal (EOD) and rescue mechatronic systems have a limitation in their handling of non-rigid and complex shape materials - they are designed for a specific environment and have a simple vision and control systems which restricts their ability to observe and respond adequately (human like) to the changes in the environment [2]. This is another proof of the necessity of developing such an intelligent system as RESCUER.

2.1.1 RESCUER Objectives
The system of RESCUER has some major objectives in improvement of urban search and rescue systems:
• Improvement of risk management by using new mechatronic and intelligent methods for bomb disposal and rescue operations and IT techniques on management of rescue missions [2].
• To applies and combines advanced and intelligent techniques for the detection of explosives, chemical, biological and radioactive materials, and human bodies [2].
• To develop advanced rescue planning methods and human-machine interface techniques for secure IEDD/EOD and rescue operations [2].

This will help for replacement of the rescue worker in operations potentially dangerous for his life and health. This means increasing the safety of the rescue worker by reducing the need of human interaction at the dangerous area by using the RESCUER advice giving subsystem ERMMAS. This is possible because when a human is found in collapsed building for example it is automatically located on the map that RESCUER system provides.

2.1.2 Major Innovations

The RESCUER system will combine two separate streams – bomb disposal robots and rescue robots in one more powerful, more practical, more functional, more intelligent and more dextrous machine [2]. Currently the bomb disposal robots are focused mostly on the possibility to grasp simple shape rigid objects and to transport them to a disposal place, or to disrupt the threat on-site. They have single manipulator arm with a two jaw gripper, a vision system limited to monotype view of the target area and joystick control of the robots motion and handling. The rescue robots are generally of observation type. RESCUER’s challenge will be achieved by implementing in one intelligent mechatronic system innovative issues like: decision making based on smart sensing, 3D perceptions, two cooperating dextrous arms, exoskeleton type human-machine interface including data glove, tactile sensors and grasping of non-rigid materials, autonomous mobile platform. RESCUER together with ERMMAS combines the potential of the combatant and rescue specialists in an intelligent mechatronic mobile system making it an ideal engine for Civil Protection Agencies. RESCUER not only replaces the rescue specialist in rescue missions, but it also concentrates on the following:

• RESCUER will perform complex automatic estimation of the environment (visual, tactile, radioactive, chemical for the place and quantity of victims and survivors etc) in real time [2]. RESCUER will create an topical model (map) of the environment. Of great importance here is the duration of this process, which RESCUER will reduce several times in comparison with the existing estimation of the environment in catastrophic event;
• RESCUER will give the possibility for identification and recognition of objects and situations and for decision making (automatically) in real time, based on the sensor information combined with the enormous quantity of data and knowledge. At present this is not possible for one, or several co-operating rescue specialists [2];
• RESCUER’s ERMMAS will generate a decision which will be used either as an advice to the operator, or as a control command to the mechatronic system. This is of great importance when the time for reaction is critical because it will implement autonomy and reduce and monitor human error [2];
• RESCUER will be able to work in conditions and environments where it is impossible or dangerous for humans [2] (total darkness, very dusty areas, radioactive areas, absence of oxygen, etc.);

RESCUER is based on the reduction of the risk during detection, identification and disposal of terrorist and other threats in which the rescue and police professionals might come across in their work. The RESCUER system will cover a range of actions from
the physics of the threat and identification of the various kinds of explosive ordnance to the psychological aspects of accidents and first aid in rescue missions. RESCUER will reduce the most critical factor in the search for survivors after disasters – time – by eliminating unsuccessful support actions using the enhanced manipulation and perception capabilities embedded in the RESCUER system [2].

In conclusion the RESCUER system is an intelligent, dextrous, mobile two arm system equipped with sensors for field detection and location of human bodies under debris.

2.2 ERMMAS Overview

What is ERMMAS and what is its role in RESCUER?
The Emergency Risk Management Monitoring and Advising System (ERMMAS) [3] is one of the basic subsystems of RESCUER, integrating the main part of IT into RESCUER. ERMMAS should not be examined only as an expert/advising system. Its functions are considerably wider and complex. ERMMAS will use databases and a knowledge base both for identification of specific risk sources and for advices - giving to the teams of specialists and to the operator, who controls the mechatronic system; however this does not exhaust the functions of ERMMAS.

ERMMAS is designed to make a complex assessment of environment, to map the investigated environment, to perform monitoring of environment, rescue missions documenting, etc. These functions are not more insignificant than the advice-giving function.

To implement its functions, ERMMAS will use information both for the status of the environment and the status of the mechatronic system and of other components, embedded on it. In addition ERMMAS will use other information sources such as custom-made databases, secure web-based communication with the risk management centers, etc [3]. Therefore these modules, generating information necessary for ERMMAS, also should be included in the conceptual model of ERMMAS shown on Figure 2.3.

The manner of entering, storage and presentation of the sensor data in ERMMAS is of special significance. Therefore it is expedient to present the conceptual model as a structure of independent modules and links, implementing the basic ERMMAS functions. To this model it is expedient to associate other modules of RESCUER, which are directly or indirectly connected with ERMMAS functioning. The figure below [3] shows us all the connections between the different modules and systems in ERMMAS (Figure 2.3).
2.2.1 Basic functions of ERMMAS

As an important part of RESCUER, ERMMAS’ basic functions will include:

- To establish the context of any event causing the relevant situation (to be entered by the operator in accordance with the specific mission);
- To identify specific risk sources of the environment (local destructions, presence of explosive materials, hazardous gases, spilling of dangerous liquid fluids, hazardous solid substances, sources of radiation, improvised explosive devices, unexploded ordnance, local fires, etc.). For identification of risk sources, ERMMAS will use information from specialized sensors (sensors for explosives, gas analyzer, radiation sensor, acoustic sensor, etc.), information from the visual inspection, performed by means of stereo vision head, as well as the available custom-made databases (ERGO 2000; EOD IS, etc.);
- Quantitative evaluation of determining characteristics for specific risk sources based on the obtained information;
- Assessment of risk level, caused by specific risk sources according to a grading scale (low, average, high);
- Presentation of the descriptions of risk sources, including the level of risk, which they create;
- Searching and localization of survived people, located in the researched zone.

ERMMAS will not make statistical assessments, financial assessments and forecasts; ERMMAS will make only risk assessment with regard to risk sources found in the environment and only with regard to the specific conditions.

2.2.2 Main modules

The main modules of ERMMAS are:

- Mapping
- Monitoring
- Module of primary processing, identification and risk assessment
2.2.2.1 Mapping module

This ERMMAS function should be divided in two: local mapping of the investigated area and global mapping of the region affected by a specific event.

- Local mapping

The main task of this module is to present 2D map of the area step by step (incrementally) after every survey from concrete position of the mechatronic system. The map will include presentation of:
  - Risk sources – destruction, dangerous gases, dangerous fluid spills, dangerous substances, explosive materials, radiation sources, improvised explosives, local fires, etc. These object zones must be localized by 3D coordinates \((x, y, z)\) and to be represented on the map with standard markers;
  - Survivors – they must be localized by 3D coordinates \((x, y, z)\) and are represented with suitable marker;
  - The path of the robot.

In Figure 2.4 a model of mapping user interface is presented and it consists of:
  - Area map
  - Image from stereo head
  - Control buttons
  - Strong points (repers) – R1 and R2

![Figure 2.4 Mapping GUI (local)](image-url)
• Global mapping
  Presenting the map of the affected region and localize RESCUER on the map (in process of development). The following figure presents the global mapping screen

![Global mapping screen](image)

**Figure 2.5 Mapping screen (global)**

### 2.2.2.2 Monitoring module

The monitoring module is one of the most important modules in ERMMAS because it will be the connecting link between the intelligent machines and the operator.

The monitoring module has the following main functions:

- To form and present to the operator an information about the status of main RESCUER’s component during its stay in the investigated area (mobile platform, arms, multi-purpose sensing system, communication system, etc.)
- To present to the operator an information about main parameters and characteristics of the investigated environment (presence of different risk sources, risk level, etc.)
- To display an information about the presence of survivors and their location in the investigated area;
- To present advices for the rescue teams and for the operator controlling the mobile platform;
- To present to the operator a local map of the investigated area and global map of the region with localization of the investigated area and the dispersion model onto it;
- To present detailed information related to the characteristics of the risk sources and allow the operator to input parameters and characteristics of the
environment on the ground of the sensor data, view examination and custom-made databases;

- To allow the realization of different operation modes of ERMMAS and other components of RESCUER as initialization, corrections, tuning, etc.

Structure of the monitoring module:
It is foreseen the monitoring module to include:
- Hardware components (three monitors (screens) and corresponding interface to the PC), shown on the following figure;
- Software for designing the screens, for visualization and performance the dialogue with the rescue operator.

![Figure 2.6](image)

It is foreseen on the three monitors to be displayed the following:
- First monitor – main screen for monitoring the investigated area and will be active during the whole rescue mission;
- Second monitor – the operator will be able to call different screens needed for the performance of main ERMMAS functions from that monitor. Those screens are related to:
  - Operating modes of ERMMAS;
  - Visualization of the information from the sensing system;
  - Assessment of the characteristics of the risk sources;
  - Local mapping;
  - Global mapping;
  - Generation of advices to the rescue operator;
  - Other functions
- Third monitor – visual information about the investigated environment and RESCUER’s components based on the information from the stereo vision head and the CCD cameras will be displaying non-stop.
2.2.2.3 Module for primary processing, identification and risk assessment

The Module for primary processing, identification and risk assessment has the following functions:

- to transform and structure input data, formed by a sensing system, entered by an operator, or available at custom-made databases in a form compatible with the tools used for processing, identification and risk assessment;
- to process input data in order to improve the quality of information and to extract specific attributes (coordinates, characteristics, parameters) by means of tools connected with image processing, data fusion and AI;
- to identify individual risk sources of the environment (local destructions, presence of explosive materials, hazardous gases, spilling of dangerous liquid fluids, hazardous solid substances, sources of radiation, improvised explosive devices, unexploded ordnance, local fires, etc.), to make quantitative evaluation of their characteristics (composition, volume, power, dimensions, concentration, etc.), to grade them in accordance with the defined threshold values;
- to make quantitative evaluation of risk, caused by specific risk sources under specific conditions;
- to recognize individual risk sources (improvised explosive devices and unexploded ordnance) on the basis of a set of features, measured by specialized sensors or entered by an operator on the basis of visual inspection in dialog mode;
- to form and structure output data for other modules of ERMMAS (mapping, monitoring, decision support system, etc.);
- to realize dialog with the operator by means of appropriate interface.

Assessment of risk level, caused by specific risk sources according to a grading scale (low, average, high);

Presentation of the descriptions of risk sources, including the level of risk, which they create;

Searching and localization of survived people, located in the researched zone.

For identification of risk sources, ERMMAS will use information from specialized sensors (sensors for explosives, gas analyzer, radiation sensor, acoustic sensor, etc.), information from the visual inspection, performed by means of stereo vision head, as well as the available custom-made databases (ERGO 2000; EOD IS, etc.).

2.2.3 Contributions of ERMMAS for improvement of risk management.

ERMMAS is a multifunctional system, which summarizes the experience gained in risk management [3] and gives a number of new opportunities compared to existing risk management technologies (including by using rescuer robots). These new opportunities are mainly the following:

- ERMMAS provides opportunity for automatic performance of a complex assessment of dangerous environment. This assessment includes identification of risk sources, quantitative assessment of their characteristics, and assessment of risk level, caused by these risk sources. This opportunity significantly improves the conditions for risk management, since it provides certainty of environment and the required information for the work of the teams of rescue specialists and for the mechatronic system control.
ERMMAS provides opportunity for global monitoring, which includes: monitoring (status) of the components of RESCUER; monitoring (status) of investigated environment and of its characteristics. Existing risk management systems give partial solution to these tasks.

ERMMAS is a system, which provides opportunity for advice giving to the teams and for mechatronic system control. For that purpose it summarizes the gained experience in the form of databases and knowledge bases, as well as the existing experience of experts, involved in risk management.

ERMMAS provides opportunity for environment mapping, both for local mapping and for global mapping. For local mapping are implemented a number of new elements, which are not available in the existing mapping systems and which are extremely important from the point of view of risk management.

ERMMAS is an intelligent system. It uses methods and tools of artificial intelligence, which make it an open system. It may be learned and re-learned, as well as it may use the already gained experience under risk management;

as a subsystem of RESCUER, ERMMAS gives a number of other advantages, which may receive quantitative or qualitative assessment. Example: it provides opportunity to shorten the duration of reconnaissances missions (which is of great importance for risk management), it does not run a risk on life and health of the teams of rescue specialists, being involved in reconnaissance missions etc.

as an advice-giving system ERMMAS participates in the realization of missions, connected with neutralization of different risk sources. The known robotized systems for risk sources neutralization (mainly of improvised explosives) have much more limited opportunities;

as an advice-giving system ERMMAS participates in the realization of missions, related to carrying out of rescue operations (finding and localization of survived people, removal of structures under which there are people, supply of air, water, food and medicines, etc.). A big part of these options are not offered by any of the existing robotized systems designed for work in dangerous environments.
3 Main tasks and purposes

The main purpose of the project is developing a functional and easy to use human-machine interface (HMI) for investigated area and main components in the system of RESCUEER. It will provide information about:

- status of sensors
- environment status
- map of the area
- indicator for human presence
- the autonomous mobile platform status
- manipulators status
- lots of buttons that will help calling other screen and/or databases.

Sometimes saving human life is a matter of having the right decision in very short time interval. Therefore the information that the monitoring screen provides should be not superfluous, because then it will be harder for the people who operates with it to have the right decision in time.

The main tasks of the project include:

- Reviewing the scope of the field of similar existing resolves;
- main screen of the monitoring module developing;
- software development for main components status defining;
- software development for environment status visualizing;
- software development for simulation of Rescuer state and investigated area;
- interface simulations and rates;
- results, analyzing and conclusions.
4 Similar solutions

As we know this is not something completely new. There are solutions of similar problems of developing GUIs. This chapter illustrates similar solutions of similar problems. At the end of the chapter we include our interface and we compare the functionality of all the solutions, presented here in this chapter.

4.1 INEEL

At the 2002 AAAI Robotics Competition and Exhibition, the Idaho National Engineering and Environmental Laboratory (INEEL) [4] demonstrated a robot that can adjust its level of autonomy leveraging its own, intrinsic intelligence to meet whatever level of control was handed down from the user. The robot had the ability to actively protect itself and the environment as it navigated through the USAR environment. In addition, the robot continuously assessed and adapted to changes in its own perceptual capabilities. The INEEL also demonstrated an interface for supporting mixed-initiative interaction between the operator and human. The interface displays an abstracted representation of the robot’s experience and exploits sensor-suites and fusion algorithms that enhance capabilities for sensing, interpreting, and "understanding" environmental features.

![Figure 4.1 The original INEEL interface](image)
4.2 UMass Lowell

Urban Search and Rescue (USAR) robots work to make sense of dangerous post-disaster environments for other first responders. First responders are the initial rescue workers who arrive on a disaster scene such as an earthquake [5]. Oftentimes, USAR robots are at least partially tele-operated, meaning the user is not in the same location as the robot. Rather, the user closely monitors and operates the robot through the interface.

The goal of a USAR robot is twofold: find survivors and provide a map of the disaster site for rescue workers.

The goal of UMass Lowell project is to build and test an interface for a Search and Rescue Robot. One of the larger considerations of this project is that the robot and user work as a team to achieve a goal.

Figure 4.2 The UMass Lowell interface.
4.3 Our interface

The figure below shows us our interface.

![Monitoring GUI](image)

Figure 4.3 Monitoring GUI

As we can see on the pictures above both of these independent from each other GUIs (the INEEL and UMass Lowell) gives us information about power, different working conditions and picture from the investigated area and map of the area. Compared to them our interface provides information about:

- different working conditions;
- map of the investigated area;
- status information for the environment, power, manipulators, control of the robocar;
- sensors and robocar status at the moment;
- information about motion (velocity) and axis angles;
- rescue robot control – tele or auto;
- the ability to call different screens from it.

The main disadvantage of our interface might be that there is no place on it for current picture of the environment, like on the other two. This is solved with another screen where the operator will have visual information about the investigated environment and RESCUER’s components based on the information from the stereo vision head and the CCD cameras and this will be displaying non-stop.

The main advantage is that there is information of the investigated environment, such as status of the environment (low, medium and high level of risk) which is important in people search. Also an important advantage is the human presence indicator, which shows when a human is found in the area and automatically locates it on the presented map. This gives the ability to be able to use them in different situations like earthquakes, pollutions, terrorist assaults and so on.
5 Main screen building

This chapter presents some information about the chosen environment and some information about the tool for building GUIs – ‘Guide’ as tools, menus, properties etc.

5.1 GUI building

Windows applications can be divided in two big groups – windowed and consoled. Consoled applications look like old MS DOS programs. They use the whole RAM and screen. In contrast to Windows applications include one or more windows at the screen at the same time. They can change their sizes and placement, to be minimized and started new ones. They use a lot of components like menus, different process indicators.

MATLAB environment allows us to build Windows based applications in which are included 2D and 3D arrangement tools. MATLAB gives us not so many but enough tools and techniques to build good GUI.

In modern visual programming systems each object has individual properties. In some of these MATLAB properties are shown in the table below. Some of them are used only in concrete situation like for example SliderStep property is used only by Slider element.
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Allowed values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backgroundcolor</td>
<td>Background color</td>
<td>Three component vector</td>
</tr>
<tr>
<td>BeingDeleted</td>
<td>Possibility for deletion</td>
<td>On or Off</td>
</tr>
<tr>
<td>BusyAction</td>
<td>Reaction on new event arising</td>
<td>Queue or Cancel</td>
</tr>
<tr>
<td>ButtonDownFcn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CData</td>
<td>Array</td>
<td></td>
</tr>
<tr>
<td>Clipping</td>
<td>Object clipping property</td>
<td>On or Off</td>
</tr>
<tr>
<td>CreateFcn</td>
<td>String or function handle</td>
<td></td>
</tr>
<tr>
<td>DeleteFcn</td>
<td>String or function handle</td>
<td></td>
</tr>
<tr>
<td>Enable</td>
<td>Access to the object</td>
<td>On or Off</td>
</tr>
<tr>
<td>Extent</td>
<td>Rectangle dimensions</td>
<td></td>
</tr>
<tr>
<td>FontAngle</td>
<td>Font angle</td>
<td>Normal or Italic or Oblicue</td>
</tr>
<tr>
<td>FontName</td>
<td>Name of the font</td>
<td>All Windows fonts</td>
</tr>
<tr>
<td>FontSize</td>
<td>Size of the font</td>
<td></td>
</tr>
<tr>
<td>FontUnits</td>
<td>Units for font measurement</td>
<td>Inches, Cm, Pixels, Points</td>
</tr>
<tr>
<td>FontWeight</td>
<td>Weight of the font</td>
<td>Normal, Light, Demi, Bold</td>
</tr>
<tr>
<td>ForegroundColor</td>
<td>Color of the labels</td>
<td>Three component vector</td>
</tr>
<tr>
<td>HandleVisibility</td>
<td>Visibility sign of a function</td>
<td>On or Off</td>
</tr>
<tr>
<td>HitTest</td>
<td>Allowing sign</td>
<td>On or Off</td>
</tr>
<tr>
<td>HorizontalAlignment</td>
<td>Text alignment</td>
<td>Left, Right, Center</td>
</tr>
<tr>
<td>Interruptible</td>
<td>Allows interruption</td>
<td>On or Off</td>
</tr>
<tr>
<td>Max</td>
<td>Defines max value</td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>Defines min value</td>
<td></td>
</tr>
<tr>
<td>Position</td>
<td>Defines object position</td>
<td>Three component vector</td>
</tr>
<tr>
<td>SliderStep</td>
<td>Defines the slider step</td>
<td></td>
</tr>
<tr>
<td>Tag</td>
<td>Event processing function</td>
<td></td>
</tr>
<tr>
<td>_TOOLTIPSTRING</td>
<td>Hint text</td>
<td></td>
</tr>
<tr>
<td>UIContextMenu</td>
<td>Context menu creating</td>
<td></td>
</tr>
<tr>
<td>Units</td>
<td>Objects size</td>
<td>Inches, Cm, Points, Pixels</td>
</tr>
<tr>
<td>UserData</td>
<td>Data array</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Object value</td>
<td></td>
</tr>
<tr>
<td>Visible</td>
<td>Object visibility</td>
<td>On or Off</td>
</tr>
</tbody>
</table>

Table 5.1 Object properties

There are two different ways to create GUI in MATLAB. Dynamical and with the build in editor GUIDE. The dynamical way is while the program execution. For creating graphical objects we can use the UIControl function.

```matlab
handle = uicontrol(parent)
handle = uicontrol(...,'PropertyName',PropertyValue,...)
```

The object properties can be changed during the program execution with set function.

```matlab
set(h, 'propertyname', value[, 'propertyname2', value2, ...])
```
5.1.1. Building GUI with GUIDE
GUIDE is the main tool for developing GUIs in MATLAB. The editor can be accessed by typing guide in the command line or file\new\GUI, with the following choices:
- Open existing GUI;
- Create new GUI;
  - Blanc GUI – empty GUI where we can add menus, axis, uicontrols and so on shown on the picture below:

![GUIDE Quick Start](image)

Figure 5.1 Guide’s quick start
- GUI with UIControls – opens the figure shown below

![Figure 5.2 UIControls GUI](image1.png)

There are object like Static Text, Edit, Frame, Radio Button, Push Button on it. We can start it but first we have to save it in the current directory.

- GUI with Axes and Menus – consists axis, menus and popup menus – Figure 5.3

![Figure 5.3 GUI with axis and menus](image2.png)

- Modal question dialog – window with question and two answers in – Figure 5.4

![Figure 5.4 Modal question dialog](image3.png)
The beginning of the work with GUIDE is illustrated on Figure 5.5

![Image of Blank GUI](image_url)

Figure 5.5 Blank GUI

All the files created by GUIDE have *.fig extension. On the left side there is the tools section. On the work field there is a grid which helps the developer to situate objects on the screen. The objects themselves are arranged by the left mouse button and the upper left corner responds to the mouse cursor.

GUIDE menus:

- **File**
  - New;
  - Open;
  - Save;
  - Save as;
  - Print;
  - Close;
  - Preferences;
  - Export;

- **Edit**
  - Undo;
  - Redo;
  - Cut;
  - Copy;
  - Paste;
  - Clear,
• View
  o Property Inspector;
  o Object Browser;
  o M – file editor;
  o View Callbacks.
• Layout
  o Snap to Grid;
  o Bring to Front;
  o Send to Back;
  o Bring to Front;
  o Send Backward.
• Tools
  o Run;
  o Align object;
  o Grid and rules;
  o Menu editor;
  o Tab order editor;
  o GUI options;
• Help

5.1.2. Interface objects control

First we choose an object from the palette to the left and then we place the mouse at the place we want the object to be. The mouse cursor marks the upper left corner of the object. Then we can change the size of the object and when we press up the button the size is fixed. We can also change the position with the mouse. After we are done the actual position is written in the Position property of the Property Inspector shown on Figure 5.6, where (x, y) are the coordinates of the object and width and height are the sizes of the object. The object can be deleted when we select it and press Delete.

Figure 5.6
For the Grid control we choose Tools\Grid and Rules – shown on Figure 5.7

For aligning objects we choose Align object shown on Figure 5.8
To show how alignment works we take a couple of buttons. To align them the buttons need to be selected first. This is possible if we hold <Ctrl> and then click on the objects we want to select. We can align objects horizontally or vertically by left or right border, or by up or down border. In the example below we use left side alignment and then we press <Apply> shown on Figure 5.9. On Figure 5.10 we use equal space between objects.
5.1.3. **Property Inspector**

The property inspector provides detailed information about the properties of the selected object like for example color, style, string, tag, position etc. It is easy to use because every property is visible and easy to change. These properties can be modified in the command line too but is much more comfortable using property inspector while building the GUI.

![Property Inspector](image)

Figure 5.11

The window on the figure above is divided by two sides – on the left are the object properties and on the right are the characteristics properties. If there is a square with cross inside this means that this is vector. For example Position is a vector property shown on Figure 5.6. During the execution of the program we can get/set all the property characteristics with get/set function:

```matlab
get(h)
get(h,'PropertyName')
<m-by-n value cell array> = get(H,<property cell array>)
a = get(h)
a = get(0,'Factory')
a = get(0,'FactoryObjectTypePropertyName')
a = get(h,'Default')
a = get(h,'DefaultObjectTypePropertyName')

set(H,'PropertyName',PropertyValue,...) 39
set(H,a)
set(H,pn,pv...)
set(H,pn,<m-by-n cell array>)
a= set(h)
a= set(0,'Factory')
```
5.1.4. Object browser

The object browser is a tool that provides information about the objects hierarchy on the developed GUI. On the figure below illustrates the top level object (parent) – the figure and the other three objects (children) of the figure – pushbutton, slider and radio button. Each object on the screen is shown in the object browser too.

![Object browser diagram](image)

Figure 5.12
5.1.5. GUIDE’s .m file anatomy

To illustrate how GUIDE works we take an example with two pushbuttons. When we put them on the editor the editor automatically assign them values to String, Tag and Position properties. We can see this in Property Inspector. We change the value of the pushbutton1’s string to ‘Button1’ and the same to pushbutton2 shown on Figure 5.13.

![Property Inspector](image)

**Figure 5.13**

We start the application by pressing Run. Then a screen shows and asks us for the name of the file. We choose ‘twobutton’. GUIDE creates two files with the same names – ‘twobuttons.m’ and ‘twobuttons.fig’. The .fig file is a binary file where are all descriptions of the objects in the figure and we cannot open and look at it. The second file is regular .m file and we can check what is inside. The content of the .m file is shown below and the rows numbers are not part of the program.

```matlab
1 function varargout = twobutton(varargin)
2  % TWOBUTTON M-file for twobutton.fig
3  % TWOBUTTON, by itself, creates a new TWOBUTTON or raises the existing
4  % singleton*.
5  % H = TWOBUTTON returns the handle to a new TWOBUTTON or the handle to
6  % the existing singleton*.
7  %
8  % TWOBUTTON('CALLBACK',hObject,eventData,handles,...) calls the local
9  % function named CALLBACK in TWOBUTTON.M with
10  % the given input arguments.
```
TWOBUTTON('Property','Value',...) creates a new
TWOBUTTON or raises the existing singleton*.
Starting from the left, property value pairs are applied
to the GUI before twobutton_OpeningFcn gets called.
An unrecognized property name or invalid value makes
property application stop. All inputs are passed to
twobutton_OpeningFcn via varargin.

*See GUI Options on GUIDE's Tools menu.
Choose "GUI allows only one instance to run (singleton)".
See also: GUIDE, GUIDATA, GUIHANDLES
Edit the above text to modify the response to help twobutton

--- Executes just before twobutton is made visible.
function twobutton_OpeningFcn(hObject, eventdata,...
handles, varargin)
This function has no output args, see OutputFcn.
 hObject handle to figure
 eventdata reserved - to be defined in a future version of MATLAB
 handles structure with handles and user data (see GUIDATA)
 varargin command line arguments to twobutton (see VARARGIN)
 Choose default command line output for twobutton
 handles.output = hObject;
 Update handles structure
guidata(hObject, handles);
 UIWAIT makes twobutton wait for user response (see UIRESUME)
 uiwait(handles.figure1);
 Outputs from this function are returned to the command line.
 function varargout = twobutton_OutputFcn(hObject, eventdata,
 handles)
 varargout cell array for returning output args (see VARARGOUT);
 hObject handle to figure
 eventdata reserved - to be defined in a future version of MATLAB
73% handles structure with handles and user data (see GUIDATA)
74
75% Get default command line output from handles structure
76 varargout{1} = handles.output;
77%
78%
79% --- Executes on button press in pushbutton1.
80 function pushbutton1_Callback(hObject, eventdata, handles)
81 hObject handle to pushbutton1 (see GCBO)
82 eventdata reserved – to be defined in a future version of MATLAB
83 handles structure with handles and user data (see GUIDATA)
84
85% --- Executes on button press in pushbutton2.
86 function pushbutton2_Callback(hObject, eventdata, handles)
87 hObject handle to pushbutton2 (see GCBO)
88 eventdata reserved – to be defined in a future version of MATLAB
89 handles structure with handles and user data (see GUIDATA)

The first row is the name of the function. There is one input argument – ‘varargin’ (shown in row 1). There is one output argument – ‘varargout’ (shown in row 1). This gives us the opportunity to call the function from the command line in different ways, for example:

>> twobutton
>> hApp=twobutton
>> twobutton('Position',[10 10 250 200])
>> twobutton(x1,x2,x3)
>> [y1 y2] = twobutton(...) 

The second version returns a pointer after the execution of the program. The third version makes our figure to be shown in concrete position on the screen with concrete sizes. Version four gives values x1, x2 and x3 as values of varargin in position 1, 2 and 3 of the array. In the last version the output values are stored in y1 and y2 variables.

Rows 2 to 29 are comments and they are only used for hints. Rows 30 to 46 are rows for program initialization and we do not have to change them. At row 49 begins the opening function. It is with fixed number of arguments (hObject, eventdata, handles, and varargin) and with a standard name ‘xxx_OpeningFcn. This function works before the screen becomes visible and it has access to all pointers of the interface elements.

In this function we can change figure properties and objects. The input arguments of the function have the following meaning:

- hObject – figure pointer;
- eventdata – empty argument saved for future use;
- handles – structure, consists of the pointers of all interface elements on the figure;
- varargin – empty array or with values brought by the command line;

The handles structure is accessible to all functions in the file. Each function can add a new field in this structure, for example with the name field and to form the desired value.

handles.name_field = value;

We store this field in an exact place if we want it to be accessible to other functions or the main function.
The opening function is the function that accepts and process input arguments from the command line. We can use this function for interface’s objects and objects properties dynamic adjustment before the screen is shown. By default it does only 2 operations – adds the output field in the handles structure (row 59) and then saves the modified structure handles (row 62).

The other function – OutputFcn works before the application is closing.

```
varargout = twobutton_OutputFcn(hObject, eventdata, handles)
```

It transfers the arguments hObject, eventdata and handles to the varargout array. The main purpose of this function is to return values to the command line after the ending of the application. By default it returns through the varargout array the value of the field handles.output.

The callback functions are created by GUIDE and in theirs body is the code that is executed by the executing of the objects. The name of the function is created by joining the tag name and the word callback – pushbutton1_Callback. If we want some reactions when a button is pressed we add the following code to the functions:

```
set(hObject,'String','Pushed');
set(handles.pushbutton2,'String','Button2')
set(hObject,'String','Pushed');
set(handles.pushbutton1,'String','Button1')
```

The result is shown in the following picture:

![Figure 5.14](image)

If we wanted to create this application with the dynamical way it will be possible with a lot of commands. Now our .m file is big but there are a lot of commentaries in it created by GUIDE.
6. Interface realization and simulations

This chapter shows how our interface is built. Building the interface itself is not so complicated. Just put some buttons, some axis some panels and edit text boxes and align the objects and we have GUI now shown on Figure 6.1 The much more complicate task is to make this GUI work.

![Figure 6.1](image_url)

6.1. Sensor status definition

The main problem of the simulation is sensor status defining. How we will be able to check is the sensor working or not? Some of the sensors have PC output and they will provide information about the sensor is it working or not through continuous signal. But some of the sensors are not equipped with PC output and we need to use another algorithm instead of simulating continuous signal. For example these are the gas analyzer sensor and the radiation sensor. They will write data in structured text files. Other sensors like for example the acoustic sensor and stereo head and CCD camera will not write data into files. They will just provide the current information they receive. The problem with status defining of the last three sensors is not so big because we can always create a certain noise for example to check the acoustic sensor. The same situation is with the CCD camera and stereo head because they will provide information non stop and if the information stops this means that the sensor is not working. This is the reason to add two small buttons just near each of them to be able to set up the status manually. By pressing the small green button we color the sensor status indicator in green which corresponds to working sensor. Pressing the small red button causes red coloring to the sensor status indicator – corresponds to not working. This is why these sensor indicators are not colored when the GUI is executed.
The status defining problem for the rest of the sensors, except the two which writes data into text files, is solved by the following algorithm: we check the status in the previous moment and compare it to the current status. If it is been changed – the sensor is not working and then we color it in red.

The algorithm for the other two sensors which are writing data into structured text files is different. The structure of the files is known and we are looking for something that can be used like key element – something unique. The structure of the file is shown in Appendix.

This is enough information for us to see that the measurement is done during defined time intervals and every measurement has number (column #1, first element of the row). This is enough to build program that reads this element and compare it with the same in previous moment.

The other sensor – gas analyzer’s generated file is similar. Its structure is shown in Appendix.

The algorithm here is similar – each measurement starts with:

```
04/04/06-11:36:59                   LOG DATA REPORT                    PAGE 1
```

We can search for the string ‘PAGE 1’, going back to the beginning of the row and read the string there which illustrates the date and the time. If the time is not been changed – the sensor is not working.

To be able to realize these algorithms we need to have continuously writing in these text files. The function that writes into text file as simulating the real radiation sensor is called ‘timer_fun_1’ and the full code does the writing is shown in Appendix. The other function that simulates writing into the other file (that simulates the gas analyzer sensor) is called ‘timer_fun_2’ and its full code is shown in Appendix too. By this continuous writing into files we simulate these two sensors’ work.

The algorithm of the situation for these two sensors – gas analyzer and radiation sensors is described below. Opposite on each sensor name (on the screen GUI) there is a button which is green in the beginning and it says ‘Working’. When the monitoring GUI is launched it automatically launches the screen GUI. Launching screen GUI activates writing into these two files for the gas analyzer and radiation sensors. Writing into these files simulates the real situation when the sensor is working and writing data into file. This is why the simulation buttons on the screen GUI are colored in green and the string is ‘Working’ at the beginning. To complete the simulation for these sensors we need to reed these two files and to check are they changing after current time period. If they are changing – the sensors are working. Stopping writing into the text file should cause the GUI to present the sensor as not working – becoming red. This can be done by pressing the green ‘Working’ buttons on the right to the sensor name. The button becomes red and says ‘Notworking’. Stopping writing into files is equal to stopping the sensors work. If there is no writing into files, when we read it the last read value is the same as the value, read before. This means the sensor is not working and in this case it is colored in red.
The simulation screen is shown on the figure above. It consists of several panels with buttons, sliders, checkboxes and edit text boxes. The first panel is for sensor status testing. Facing the name of the sensor is the edit box where we put value. By default every value there is 0 (zero), except the environment status fields (value 1 – corresponds to save environment). If we change the value in the corresponding edit box this is equal to changing the status of the sensor. The result is the sensor is becoming red.

The code realizes this algorithm is shown in Appendix in ‘timerset.m’ file in mycall_1 function. It is labeled with ‘% sensors status 1-8 (8matrixes)’ where the first part (1-2) is for the first two sensors – gas analyzer and sensor for radiation and the rest (3-8) is the part fore every other sensor.

The coloring itself is done with the code shown later in the same function. It is labeled ‘% sensors, environment and bumpers color status refreshing’.

To be able to make this work we assign variable to each object – edit text box or slider, or checkbox. The variable is actually an element of a temporary array M with 31 elements and each element of the array corresponds to an exact object. For example the explosion detection sensor is assigned to the element #3 from M.
function edit_3_Callback(hObject, eventdata, handles)
global M
M(3) = str2num(get(hObject,'String'));

It is similar with every other edit box fields.

6.2. Battery, velocity and angle status
The simulation of the battery and robocar velocity and angle is proper to be done by sliders because we can easily set the values limits. It is easier with the sliders and checkboxes because then we get the current value directly with get and we don’t need to transform it from string to number like in the example above, because slider and checkbox objects returns number value (for checkboxes only ‘1’ or ‘0’ – true or false):

6.2.1. Battery
For the battery we use standard values – from more than 10 to 6. The code that assigns value to the temporary variable M is shown in Appendix. It is the same for every slider. The only difference is the number of the element in M that the value is written in. The code that does the battery coloring is shown in Appendix in ‘timerset.m’ file in ‘mycall_1’ function. It is labeled with ‘battery status (3 matrixes)’.

Status_new is a temporary variable type array, where are the current values received from sensors. We use it to compare it with fixed number in this case. Comparing with numbers simulates the voltage of the battery. If it is equal to ‘6’ then we have no voltage left in the battery and this is signalized by coloring the cells in white. If the value is between grater than ‘6’ and less or less and equal to ‘8’ we color the most left cell in red and the rest of the battery in white.

In case when the value is grater than ‘8’ and less or less and equal to ‘10’ we color the most left and the middle cell in yellow and the right in white. The situation if the value is more than ‘10’ is equal to full energy and it is visualized by three green battery cells. The full code realizes this is shown in Appendix.

6.2.2. Velocity and angles simulation
The velocity and angles simulation works in the same way – the value is set by the slider position and then it is been read and visualized in the edit box. The edit box is set to be forbidden to change it from the monitoring GUI. First we read the value set by the corresponding slider and then we visualize it on the edit box. The full code for the reading and visualizing is shown in Appendix. The reading of the value is done by:

6.3. Environment status simulation
The environment status simulation works similar. First we declare that we have three values for the environment status:

- ‘1’ corresponds to ‘Save’ equal to green color;
- ‘2’ corresponds to ‘Average’ level of risk, equal to yellow color;
- And ‘3’ corresponds to ‘High’ level of risk and it is colored in red.

In the edit text fields we put one of these 3 values.

The elements in M from 9 to 18 are reserved for the environment status. We check their values and according to the values we change the corresponding statusRGB row with one of these three values [0 1 0] (green), [1 1 0] (yellow) or [1 0 0] (red). The full code that reads and visualizes the status is shown in Appendix.
6.4. Bumpers status
The bumpers status is visualized by red color for not working and green color for working. We simulate the status with checkboxes and if a checkbox is checked then the corresponding bumper is not working – colored in red. In the other case – if checkbox is not checked – the bumper is working – colored in green. The reading of the values and coloring code is shown in Appendix.

6.5. Manipulators and motion status
We changes in manipulators and motion status by pressing the corresponding button on the screen GUI and visualizing it on the monitoring GUI with the corresponding color for working or not.

6.5.1. Motion
When the robocar is moving we signalize it with the indicator ‘Motion’. For the simulation we have the signal ‘1’ for moving and ‘0’ for not moving. When it is ‘1’ we illustrate it with coloring the indicator in green and change the string to ‘YES’ (at the beginning is ‘NO’ and it is red). We simulate ‘1’ and ‘0’ by pressing the motion simulation button on the screen GUI. If we have ‘0’ from the motion button (not pressed on the screen GUI) the velocity slider is inactive and its value is set to ‘0’. When we press the motion button (it automatically sets the string on the screen GUI to ‘Motion : YES’) we color the indicator in green and set the motion string to ‘YES’ (on the monitoring GUI) and at the same time set the velocity slider to enable, to be able to simulate speed through it. The full code does this is shown in ‘motion_callback’ function in Appendix.

The manipulators status visualization is similar. There are two situations:

6.5.2. Position status
The manipulators have two positions – ‘initial’ – when they are near the robot’s body, and ‘Working’ – when they are doing something and are not near the robot’s body. With the second part of the code shown in ‘position_callback’ function (in screen,m file) shown in Appendix we just change the string on the simulation button to be more clear what situation should be visualized on the monitoring screen.

When we press the button on the screen GUI – it returns ‘1’ and then we set the color of the ‘initial’ indicator to white and the ‘working’ indicator to green. If the value is ‘0’ – corresponding to not pressed button on the screen GUI it makes the color of the initial indicator green and the color of the other indicator white. The full code does that is shown in ‘timerset.m’ file in Appendix.

6.5.3. Manipulators control
The algorithm is the same as the one described above. When the button on the screen GUI is pressed it returns ‘1’ and the string on it is ‘Auto’ and when it is not pressed – ‘0’ and ‘Tele’. The coloring works in the same way – if the value is ‘1’ – the ‘Tele’ indicator is white and the other is green, and if the value is ‘0’ – ‘Tele’ is green and ‘Auto’ is white. The full code makes it is shown also in Appendix.

6.6. Images
We use several images on the monitoring GUI at this moment. One of them is the robocar picture in ‘Robot status’ panel. It gives us information for the angles describing the robocar position. It is set by visualizing image on axis on the GUI without but without the axis itself, only the image. The same is with the picture of the human in
‘Human presence’ panel and for the map in ‘Area map’ panels. It is done by the code shown in the beginning of the ‘monitoring.m’ file, shown in Appendix.

6.7. Human presence
Human presence simulation is done by using checkbox. This is an easy way to make it, because it returns two values – ‘1’ when it is checked and ‘0’ when it is not checked. The algorithm is when we have ‘1’ (checked) the picture of the human to be visible on the monitoring GUI and when we have ‘0’ to make this picture not visible. It is done by the code shown in ‘timerset.m’ file, shown in Appendix.

6.8. Rest of the buttons
The rest of the buttons, situated in the ‘Command buttons’ and ‘Data from sensors’ panels are foreseen to call other screens which will produce more detailed information about data from sensors and advising system, mapping and so on.
7. Analyze and conclusion

This GUI gives the users information like status of the main features of the robocar like status of the environment, sensors status, manipulators status, robot status, map of the investigated area, which visualize risk zones and human presence (if there is human presence). It gives us a certain amount of control buttons through which we are able to call other GUIs like for example local and global mapping GUIs. Compared to the other similar solutions it gives more information about the environment’s status and human presence two of the most important advantages.

At the beginning of this report I mentioned that the purpose of the project is to be created fast, accurate and functional interface. Now when it is done I think it covers these requirements – it is accurate because it is simple constructed. We chose to visualize values ‘1’ and ‘0’ for all the statuses and present them with working or not working, except the battery and environment status. The battery is a bit more complicated – comparing with values: greater than 10 equals to full battery, between 10 and 8 equals to 2/3 of the battery, between 8 and 6 – low battery capacity and equal to 6 represents critically low battery. For the environment status is simpler than the battery – we have only three values – ‘1’, ‘2’, and ‘3’. They are formed in another module of ERMMAS and they are transferred to the monitoring module. It is visualized on the monitoring GUI by red, yellow and green for high, middle level of risk and green for safe environment. This GUI is also fast because it reflects all the changes in sensor, robocar or environment status fast enough. I say fast enough because the visualizing of the signal on the interface doesn’t take time. More complicated is the signal to be formed by the other modules of ERMMAS (which is part of another team’s work). It is useful and functional because it provides information for sensor, environment and robocar status at the same time on one screen. This is not presented on the screens we compare with. None of them presents the environment status which is very important for human search and rescue missions. It also provides buttons for additional settings and/or screens to be called through to present all the necessary information.

In conclusion we can say that this is a new hybrid system between existed USAR and bomb disposal robots, combines all the good characteristics of these two different robot systems making it faster, smarter and more universal.

7.1. Future work

As a future work it is planned to be developed 3D presenting of the angles alpha, beta and gamma, illustrating the robocar’s position in the 3D space, not only picture for the angle and the angle value. This will make the interface more functional and easy to use presenting the real position in the space. We can include coloring of the angles when they reached critical values.

At this moment ERMMAS is developing in MATLAB environment as a standard. But MATLAB is heavy environment and as a future work is planned this software to be compiled in C or C++ to be more universal in different situations.
REFERENCES:


[3] – The University of Rousse development team materials


[5] – UMass Lowell report -

Appendix

function varargout = monitoring(varargin)

MONITORING M-file for monitoring.fig

MONITORING, by itself, creates a new MONITORING or raises the existing singleton*. H = MONITORING returns the handle to a new MONITORING or the handle to the existing singleton*. MONITORING('Property','Value',...) creates a new MONITORING using the given property value pairs. Unrecognized properties are passed via varargin to monitoring_OpeningFcn. This calling syntax produces a warning when there is an existing singleton*.

MONITORING('CALLBACK') and MONITORING('CALLBACK',hObject,...) call the local function named CALLBACK in MONITORING.M with the given input arguments.

"See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)"."

See also: GUIDE, GUIDATA, GUIDEHANDLES

% Edit the above text to modify the response to help monitoring

% Last Modified by GUIDE v2.5 21-May-2006 16:04:21

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @monitoring_OpeningFcn, ...
    'gui_OutputFcn', @monitoring_OutputFcn, ...
    'gui_LayoutFcn', [], ...
    'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before monitoring is made visible.
function monitoring_OpeningFcn(hObject, eventdata, handles, varargin)
    handles.output = hObject;

    % Update handles structure
    guidata(hObject, handles);

    [H,c] = imread('data/human.bmp');
    R = imread('data/robocar.png');
    colormap(c);
    image(H,'Parent', handles.human_pr_axes);   set(handles.human_pr_axes, 'Visible', 'off')
    image(R,'Parent', handles.robocar_axes);    set(handles.robocar_axes, 'Visible', 'off')
    load('data/map_data');
    show_map(map_data, handles.map_axes);

    t_1 = timerset(handles);
    set(hObject,'CloseRequestFcn',{@cls_fun,t_1});

    screen;
    % ---------------------------------
    function cls_fun(hObject, eventdata,t_1)
    stop(t_1);
    delete(t_1);
    delete(hObject);

    % --- Outputs from this function are returned to the command line.
    function varargout = monitoring_OutputFcn(hObject, eventdata, handles)
    varargout{1} = handles.output;

    % --- Executes on button press in sh_green.
    function sh_green_Callback(hObject, eventdata, handles)
    set(handles.stereo_head_status, 'BackgroundColor', [0 1 0]);

    % --- Executes on button press in sh_red.
    function sh_red_Callback(hObject, eventdata, handles)
    set(handles.stereo_head_status, 'BackgroundColor', [1 0 0]);

    % --- Executes on button press in ccd_green.
    function ccd_green_Callback(hObject, eventdata, handles)
    set(handles.cameras_status, 'BackgroundColor', [0 1 0]);

    % --- Executes on button press in ccd_red.
    function ccd_red_Callback(hObject, eventdata, handles)
    set(handles.cameras_status, 'BackgroundColor', [1 0 0]);

    % --- Executes on button press in acoustic_green.
    function acoustic_green_Callback(hObject, eventdata, handles)
    set(handles.acoustic_sensor_status, 'BackgroundColor', [0 1 0]);

    % --- Executes on button press in acoustic_red.
    function acoustic_red_Callback(hObject, eventdata, handles)
    set(handles.acoustic_sensor_status, 'BackgroundColor', [1 0 0]);
% timerset.m
function t_1 = timerset(handles)
global statusRGB M status_old br

br = zeros(1,8);

M = zeros(1,31);
M(23) = 12;
status_old = M;
statusRGB = repmat([0 1 0],size(M,2),1);

% sensors
mat_handles(1,1) = handles.gas_analizer_status;
mat_handles(2,1) = handles.sensor_for_radiation_status;
mat_handles(3,1) = handles.explDetection_sensor_status;
mat_handles(4,1) = handles.comminucation_status;
mat_handles(5,1) = handles.compass_sensor;
mat_handles(6,1) = handles.sh_angles_sensor;
mat_handles(7,1) = handles.bumpers_sensor;
mat_handles(8,1) = handles.other_sensors_status;

% environment status
mat_handles(9,1)  = handles.expl_mat_env;
mat_handles(10,1) = handles.dangerous_gases_env;
mat_handles(11,1) = handles.spill_env;
mat_handles(12,1) = handles.dangerous_solid_env;
mat_handles(13,1) = handles.dangerous_substances_env;
mat_handles(14,1) = handles.radiation_sources_env;
mat_handles(15,1) = handles.ied_env;
mat_handles(16,1) = handles.unexploded_ordnances_env;
mat_handles(17,1) = handles.local_fires_env;
mat_handles(18,1) = handles.local_destructions_env;

% bumpers
mat_handles(19,1) = handles.left_front_bumper;
mat_handles(20,1) = handles.right_front_bumper;
mat_handles(21,1) = handles.right_back_bumper;
mat_handles(22,1) = handles.left_back_bumper;

% -----------------
t_1 = timer('TimerFcn', {@mycall_1,handles,mat_handles}, ...
'Period', 1.0, ...
'ExecutionMode', 'fixedrate', ...
'Name','timer_1');
start(t_1);
% -----------------

function mycall_1(ev,df,handles,mat_handles)
global statusRGB M status_old br

% gas analyzer's file reading
fid = fopen('testIQ.txt','r');
ff = fgetl(fid);
while ~isequal(tmp,-1) && isempty(ff)
    tmp = fgetl(fid);
    ff = findstr(tmp,'PAGE');
end
for i = 1:8
    tmp = fgetl(fid);
end

for i = 1:8
    tmp = fgetl(fid);
    time = tmp(10:17);
    hour = str2num(time(1:2));
    min = str2num(time(4:5));
    sec = str2num(time(7:8));
    M(1) = hour*3600 + min*60 + sec;
fclose(fid);

% radiation sensor's file reading
fid = fopen('LB123.txt','r');
fseek(fid,-50,'eof');
tmp = fgetl(fid);
s = textscan(fid, '%s %s %s');
fclose(fid);
M(2) = str2num(s{1,1}{3,1});

% sensors status 1-8 (8 matrixes)
for i = 1:2
    if status_old(i) == status_new(i)
        br(i) = br(i) + 1;
        if br(i) == 12
            statusRGB(i,:) = [1 0 0];
        end
    else
        br(i) = 0;
    end
end
for i = 3:8
    if status_old(i) ~= status_new(i)
        statusRGB(i,:) = [1 0 0];
    end
end

status_old = status_new;

% environment status 9-18 (10)
for i = 9:18
    switch status_new(i)
        case 1, statusRGB(i,:) = [0 1 0];
        case 2, statusRGB(i,:) = [1 1 0];
        case 3, statusRGB(i,:) = [1 0 0];
    end
end

% bumpers status (4 matrixes with colors)
for i = 19:22
    if status_new(i)
        statusRGB(i,:) = [1 0 0];
    else
        statusRGB(i,:) = [0 1 0];
    end
end

% sensors, environment and bumpers color status refreshing
for i = 1:22
    - 46 -
set(mat_handles(i,1), 'BackgroundColor', statusRGB(i,:));
end

% battery status (3 matrixes)
if status_new(23) == 6
    set(handles.battery_3, 'BackgroundColor', [1 1 1]);
    set(handles.battery_2, 'BackgroundColor', [1 1 1]);
    set(handles.battery_1, 'BackgroundColor', [1 1 1]);
elseif status_new(23) > 6 && status_new(23) <= 8
    set(handles.battery_3, 'BackgroundColor', [1 0 0]);
    set(handles.battery_2, 'BackgroundColor', [1 1 1]);
    set(handles.battery_1, 'BackgroundColor', [1 1 1]);
elseif status_new(23) > 8 && status_new(23) <= 10
    set(handles.battery_3, 'BackgroundColor', [1 1 0]);
    set(handles.battery_2, 'BackgroundColor', [1 1 0]);
    set(handles.battery_1, 'BackgroundColor', [1 1 1]);
elseif status_new(23) > 10
    set(handles.battery_3, 'BackgroundColor', [0 1 0]);
    set(handles.battery_2, 'BackgroundColor', [0 1 0]);
    set(handles.battery_1, 'BackgroundColor', [0 1 0]);
end

% robocar status (motion or not)
if status_new(24)
    set(handles.motion, 'BackgroundColor', [0 1 0], 'String', 'YES');
else
    set(handles.motion, 'BackgroundColor', [1 0 0], 'String', 'NO' );
end

% manipulators status
if ~status_new(25)
    set(handles.manip_pos_initial, 'BackgroundColor', [0 1 0]);
    set(handles.manip_pos_working, 'BackgroundColor', [1 1 1]);
else
    set(handles.manip_pos_initial, 'BackgroundColor', [1 1 1]);
    set(handles.manip_pos_working, 'BackgroundColor', [0 1 0]);
end

% control condition
if ~status_new(26)
    set(handles.manip_control_tele, 'BackgroundColor', [0 1 0]);
    set(handles.manip_control_auto, 'BackgroundColor', [1 1 1]);
else
    set(handles.manip_control_tele, 'BackgroundColor', [1 1 1]);
    set(handles.manip_control_auto, 'BackgroundColor', [0 1 0]);
end

% robocar's velocity, alpha, beta and gamma
set(handles.velocity_value, 'String', sprintf('%4.2f',status_new(27)));
set(handles.alpha_angle,    'String', sprintf('%4.2f',status_new(28)));
set(handles.beta_angle,     'String', sprintf('%4.2f',status_new(29)));
set(handles.gamma_angle,    'String', sprintf('%4.2f',status_new(30)));

- 47 -
% human presence
if M(31)
    set(get(handles.human_pr_axes,'Child'),'Visible','on');
else
    set(get(handles.human_pr_axes,'Child'),'Visible','off');
end
function varargout = screen(varargin)

SCREEN, by itself, creates a new SCREEN or raises the existing singleton*.

H = SCREEN returns the handle to a new SCREEN or the handle to the existing singleton*.

SCREEN('Property','Value',...) creates a new SCREEN using the given property value pairs. Unrecognized properties are passed via varargin to screen_OpeningFcn. This calling syntax produces a warning when there is an existing singleton*.

SCREEN('CALLBACK') and SCREEN('CALLBACK',hObject,...) call the local function named CALLBACK in SCREEN.M with the given input arguments.

"See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

See also: GUIDE, GUIDATA, GUIHANDLES

Edit the above text to modify the response to help screen

% Last Modified by GUIDE v2.5 24-May-2006 21:22:23

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
'gui_Singleton', gui_Singleton, ...
'gui_OpeningFcn', @screen_OpeningFcn, ...
'gui_OutputFcn', @screen_OutputFcn, ...
'gui_LayoutFcn', [], ...
'gui_Callback', []);
if nargin && ischar(varargin{1})
gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% End initialization code - DO NOT EDIT

% --- Executes just before screen is made visible.
function screen_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for screen
handles.output = hObject;

global M n
n = 0;
M = zeros(1,31);
M(23) = 12;
handles.t_gas = timer('TimerFcn', @timer_fun_2, 'Period',10.0, ...
'ExecutionMode','fixedrate','Name','timer_gas');
handles.t_rad = timer('TimerFcn', @timer_fun_1, 'Period', 10.0, 'ExecutionMode', 'fixedrate', 'Name', 'timer_rad');

set(handles.gas_analizer, 'String', 'Working', 'BackgroundColor', [0 1 0]);
set(handles.radiation_sensor, 'String', 'Working', 'BackgroundColor', [0 1 0]);

start(handles.t_gas);
start(handles.t_rad);

% Update handles structure
guidata(hObject, handles);

set(hObject, 'CloseRequestFcn', {@cls_fun, handles});

function cls_fun(obj, env, handles)
stop(handles.t_rad);
stop(handles.t_gas);
delete(handles.t_rad);
delete(handles.t_gas);
delete(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = screen_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

% sensors status
function gas_analizer_Callback(hObject, eventdata, handles)
if strcmp(handles.t_gas.Running, 'on')
    if strcmp(handles.t_gas.Running, 'on')
        stop(handles.t_gas);
        set(handles.gas_analizer, 'String', 'Notworking', 'BackgroundColor', [1 0 0]);
    elseif strcmp(handles.t_gas.Running, 'off')
        start(handles.t_gas);
        set(handles.gas_analizer, 'String', 'Working', 'BackgroundColor', [0 1 0]);
    end
end

function radiation_sensor_Callback(hObject, eventdata, handles)
   global n
   if strcmp(handles.t_rad.Running, 'on')
       stop(handles.t_rad);
   end

   set(handles.radiation_sensor, 'String', 'Notworking', 'BackgroundColor', [1 0 0]);
   elseif strcmp(handles.t_rad.Running, 'off')
       n = 0;
       start(handles.t_rad);
   end

   set(handles.radiation_sensor, 'String', 'Working', 'BackgroundColor', [0 1 0]);
function edit_3_Callback(hObject, eventdata, handles)
global M
M(3) = str2num(get(hObject,'String'));

function edit_4_Callback(hObject, eventdata, handles)
global M
M(4) = str2num(get(hObject,'String'));

function edit_5_Callback(hObject, eventdata, handles)
global M
M(5) = str2num(get(hObject,'String'));

function edit_6_Callback(hObject, eventdata, handles)
global M
M(6) = str2num(get(hObject,'String'));

function edit_7_Callback(hObject, eventdata, handles)
global M
M(7) = str2num(get(hObject,'String'));

function edit_8_Callback(hObject, eventdata, handles)
global M
M(8) = str2num(get(hObject,'String'));

% environment status
function edit_9_Callback(hObject, eventdata, handles)
global M
M(9) = str2num(get(hObject,'String'));

function edit_10_Callback(hObject, eventdata, handles)
global M
M(10) = str2num(get(hObject,'String'));

function edit_11_Callback(hObject, eventdata, handles)
global M
M(11) = str2num(get(hObject,'String'));

function edit_12_Callback(hObject, eventdata, handles)
global M
M(12) = str2num(get(hObject,'String'));

function edit_13_Callback(hObject, eventdata, handles)
global M
M(13) = str2num(get(hObject,'String'));

function edit_14_Callback(hObject, eventdata, handles)
global M
M(14) = str2num(get(hObject,'String'));

function edit_15_Callback(hObject, eventdata, handles)
global M
M(15) = str2num(get(hObject,'String'));

function edit_16_Callback(hObject, eventdata, handles)
global M
M(16) = str2num(get(hObject,'String'));

function edit_17_Callback(hObject, eventdata, handles)
global M
M(17) = str2num(get(hObject,'String'));

function edit_18_Callback(hObject, eventdata, handles)
global M
M(18) = str2num(get(hObject,'String'));

% bumpers
function bumper_front_left_Callback(hObject, eventdata, handles)
global M
M(19) = get(hObject,'Value');

function bumper_front_right_Callback(hObject, eventdata, handles)
global M
M(20) = get(hObject,'Value');

function bumper_back_right_Callback(hObject, eventdata, handles)
global M
M(21) = get(hObject,'Value');

function bumper_back_left_Callback(hObject, eventdata, handles)
global M
M(22) = get(hObject,'Value');

% battery
function battery_slider_Callback(hObject, eventdata, handles)
global M
M(23) = get(hObject,'Value');

% robocar condition
function motion_Callback(hObject, eventdata, handles)
global M
M(24) = get(hObject,'Value');
if get(hObject,'Value')
    set(hObject,'String','Motion : YES');
    set(handles.Velocity_slider,'enable','on');
else
    set(hObject,'String','Motion : NO');
    set(handles.Velocity_slider,'Enable','Inactive','Value',0);
    M(27) = get(handles.Velocity_slider,'Value');
end

% manipulators condition
function position_Callback(hObject, eventdata, handles)
global M
M(25) = get(hObject,'Value');
if get(hObject,'Value')
    set(hObject,'String','Position : Working');
else
    set(hObject,'String','Position : Initial');
end

% control condition
function control_Callback(hObject, eventdata, handles)
global M
M(26) = get(hObject,'Value');
if get(hObject,'Value')
    set(hObject,'String','Control : Auto');
else
    set(hObject,'String','Control : Tele');
end

% velocity
function Velocity_slider_Callback(hObject, eventdata, handles)
    global M
    M(27) = get(hObject,'Value');
end

% alpha angle
function Alpha_slider_Callback(hObject, eventdata, handles)
    global M
    M(28) = get(hObject,'Value');
end

% beta angle
function Gamma_slider_Callback(hObject, eventdata, handles)
    global M
    M(29) = get(hObject,'Value');
end

% gamma angle
function Beta_slider_Callback(hObject, eventdata, handles)
    global M
    M(30) = get(hObject,'Value');
end

% human presence
function human_presence_checkbox_Callback(hObject, eventdata, handles)
    global M
    M(31) = get(hObject,'Value');
end

% writing into radiation sensor’s file
function timer_fun_1(obj,evn)
    global n
    n = n+1;
    a = clock;
    value = '0.002';
    fid = fopen(['LB123.txt','a']);
    fprintf(fid, '%u	%02.0f:%02.0f	%s
', n, a([4 5]), value);
    fclose(fid);
end

% writing into gas analyzer's file
function timer_fun_2(obj,evn)
    fid = fopen(['testIQ.txt','a']);
    fprintf(fid,'04/04/06-11:36:30                   LOG DATA REPORT                 PAGE 1');
    fprintf(fid,'04/04/06-11:36:30 TO 04/04/06-11:36:30');
    fprintf(fid,'SESSION START/FINISH DATE-TIMES');
    fprintf(fid,'ALARM SETPOINTS (LOW/MID/HIGH)');
    fprintf(fid,'1: 50.A/ 100.A/ 150.A PPM METHANE');
    fprintf(fid,'DATE-TIME              READING');
fprintf(fid, '%s
', [datestr(clock, 'mm/dd/yy-HH:MM:SS') '
0. PPM METHANE P']);
fprintf(fid, '%s
', 'M2: 1550. PPM *CARBON DIOXIDE-2K P');
fprintf(fid, '%s
', '3: 0.0 PPM CHLORINE P');
fprintf(fid, '%s
', '4: 0.0 PPM NITROGEN DIOXIDE P');
fprintf(fid, '%s
', '03/21/06-14:09:27 1: 0. PPM METHANE P');
fprintf(fid, '%s
', 'M2: 1560. PPM *CARBON DIOXIDE-2K P');
fprintf(fid, '%s
', '3: 0.0 PPM CHLORINE P');
fprintf(fid, '%s
', '4: 0.0 PPM NITROGEN DIOXIDE P');
fclose(fid);
Radiation sensor’s log files structure:

*** LB 123                      Ser. # 6108

*** ----------------------------------------

***           Counter Timer

*** ----------------------------------------

*** 21.03.06 10:50                   V 3.03

***

***

*** LB 1236-H10    CF: 0.214    TH: 1.000

B KG: 0CPS

***

*** Nr. Time   uSv/h

*** LB 123                      Ser. # 6108

*** ----------------------------------------

***           Counter Timer

*** ----------------------------------------

*** 21.03.06 10:52                   V 3.03

***

***

*** LB 1236-H10    CF: 0.214    TH: 1.000

B KG: 0CPS

***

*** Nr. Time   uSv/h

9  11:140.080
10  11:150.091
11  11:150.070
12  11:150.230
13  11:160.155
14  11:160.080
15  11:160.102
16  11:17.166
17  11:17.080
18  11:17.155
19  11:18.155
20  11:18.134
21  11:18.198
22  11:19.123
23  11:19.091
24  11:19.155
25  11:20.241
26  11:20.134
27  11:20.198
28  11:21.112
29  11:21.102
30  11:21.166
31  11:22.134
32  11:22.155
33  11:22.166
34  11:23.123
35  11:23.123
36  11:23.112
37  11:24.102
38  11:24.187
39  11:24.134
40  11:25.177
41  11:25.091
42  11:25.070
43  11:26.177
44  11:26.155
45  11:26.134
...
...
96  11:43.155

*** LB 123                      Ser. # 6108

*** ----------------------------------------
***           Counter Timer
*** ----------------------------------------

*** 21.03.06 14:52                   V 3.03

***

***

*** LB 1236-H10    CF: 0.214    TH: 1.000

B KG:  0CPS
<table>
<thead>
<tr>
<th>Nr.</th>
<th>Time</th>
<th>uSv/h</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14:52</td>
<td>0.123</td>
</tr>
<tr>
<td>2</td>
<td>14:53</td>
<td>0.091</td>
</tr>
<tr>
<td>3</td>
<td>14:53</td>
<td>0.166</td>
</tr>
<tr>
<td>4</td>
<td>14:53</td>
<td>0.155</td>
</tr>
<tr>
<td>5</td>
<td>14:54</td>
<td>0.145</td>
</tr>
<tr>
<td>6</td>
<td>14:54</td>
<td>0.209</td>
</tr>
<tr>
<td>7</td>
<td>14:54</td>
<td>0.145</td>
</tr>
<tr>
<td>8</td>
<td>14:55</td>
<td>0.145</td>
</tr>
<tr>
<td>9</td>
<td>14:55</td>
<td>0.145</td>
</tr>
<tr>
<td>10</td>
<td>14:55</td>
<td>0.145</td>
</tr>
<tr>
<td>11</td>
<td>14:56</td>
<td>0.091</td>
</tr>
<tr>
<td>12</td>
<td>14:56</td>
<td>0.145</td>
</tr>
<tr>
<td>13</td>
<td>14:56</td>
<td>0.123</td>
</tr>
<tr>
<td>14</td>
<td>14:57</td>
<td>0.091</td>
</tr>
</tbody>
</table>
Gas analyzer’s log files structure:

<table>
<thead>
<tr>
<th>Date-Time</th>
<th>Reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>03/21/06-14:09:12</td>
<td>1: 0. PPM METHANE</td>
</tr>
<tr>
<td></td>
<td>M2: 1550. PPM</td>
</tr>
<tr>
<td></td>
<td>*CARBON DIOXIDE-2K</td>
</tr>
<tr>
<td></td>
<td>3: 0.0 PPM</td>
</tr>
<tr>
<td></td>
<td>CHLORINE</td>
</tr>
<tr>
<td></td>
<td>4: 0.0 PPM</td>
</tr>
<tr>
<td></td>
<td>NITROGEN DIOXIDE</td>
</tr>
<tr>
<td>03/21/06-14:09:27</td>
<td>1: 0. PPM METHANE</td>
</tr>
<tr>
<td></td>
<td>M2: 1560. PPM</td>
</tr>
<tr>
<td></td>
<td>*CARBON DIOXIDE-2K</td>
</tr>
<tr>
<td></td>
<td>3: 0.0 PPM</td>
</tr>
<tr>
<td></td>
<td>CHLORINE</td>
</tr>
<tr>
<td></td>
<td>4: 0.0 PPM</td>
</tr>
<tr>
<td></td>
<td>NITROGEN DIOXIDE</td>
</tr>
<tr>
<td>04/04/06-11:36:37</td>
<td>LOG DATA REPORT</td>
</tr>
<tr>
<td>SESSION START/FINISH DATE-TIMES</td>
<td>03/21/06-14:09:12 TO 03/21/06-14:09:27</td>
</tr>
<tr>
<td>ALARM SETPOINTS (LOW/MID/HIGH)</td>
<td>1: 50.0 A/150.0 A PPM METHANE</td>
</tr>
<tr>
<td></td>
<td>2: 500.0 A/2000.0 A PPM *CARBON DIOXIDE-2K</td>
</tr>
<tr>
<td></td>
<td>3: 2.5 A/7.5 A PPM CHLORINE</td>
</tr>
<tr>
<td></td>
<td>4: 5.0 A/10.0 A PPM NITROGEN DIOXIDE</td>
</tr>
<tr>
<td>DATE-TIME</td>
<td>READING</td>
</tr>
<tr>
<td>03/21/06-14:19:29</td>
<td>1: 0. PPM METHANE</td>
</tr>
<tr>
<td></td>
<td>M2: 1360. PPM</td>
</tr>
<tr>
<td></td>
<td>*CARBON DIOXIDE-2K</td>
</tr>
<tr>
<td></td>
<td>3: 0.0 PPM</td>
</tr>
<tr>
<td></td>
<td>CHLORINE</td>
</tr>
<tr>
<td></td>
<td>4: 0.0 PPM</td>
</tr>
<tr>
<td></td>
<td>NITROGEN DIOXIDE</td>
</tr>
<tr>
<td>03/21/06-14:19:39</td>
<td>1: 0. PPM METHANE</td>
</tr>
<tr>
<td></td>
<td>M2: 1040. PPM</td>
</tr>
<tr>
<td></td>
<td>*CARBON DIOXIDE-2K</td>
</tr>
<tr>
<td></td>
<td>3: 0.0 PPM</td>
</tr>
<tr>
<td></td>
<td>CHLORINE</td>
</tr>
<tr>
<td></td>
<td>4: 0.0 PPM</td>
</tr>
<tr>
<td></td>
<td>NITROGEN DIOXIDE</td>
</tr>
<tr>
<td>04/04/06-11:36:44</td>
<td>LOG DATA REPORT</td>
</tr>
<tr>
<td>SESSION START/FINISH DATE-TIMES</td>
<td>03/21/06-14:19:29 TO 03/21/06-14:09:15</td>
</tr>
<tr>
<td>ALARM SETPOINTS (LOW/MID/HIGH)</td>
<td>1: 50.0 A/150.0 A PPM METHANE</td>
</tr>
<tr>
<td></td>
<td>2: 500.0 A/2000.0 A PPM *CARBON DIOXIDE-2K</td>
</tr>
<tr>
<td></td>
<td>3: 2.5 A/7.5 A PPM CHLORINE</td>
</tr>
<tr>
<td></td>
<td>4: 5.0 A/10.0 A PPM NITROGEN DIOXIDE</td>
</tr>
<tr>
<td>DATE-TIME</td>
<td>READING</td>
</tr>
<tr>
<td>03/21/06-14:19:39</td>
<td>1: 0. PPM METHANE</td>
</tr>
<tr>
<td></td>
<td>M2: 1050. PPM</td>
</tr>
<tr>
<td></td>
<td>*CARBON DIOXIDE-2K</td>
</tr>
<tr>
<td></td>
<td>3: 0.0 PPM</td>
</tr>
<tr>
<td></td>
<td>CHLORINE</td>
</tr>
<tr>
<td></td>
<td>4: 0.0 PPM</td>
</tr>
<tr>
<td></td>
<td>NITROGEN DIOXIDE</td>
</tr>
</tbody>
</table>
04/04/06-11:36:52            LOG DATA REPORT       PAGE 1

SESSION START/FINISH DATE-TIMES
03/21/06-14:19:43 TO 03/21/06-14:09:36

ALARM SETPOINTS (LOW/MID/HIGH)
1:  50.0A/ 100.0A/ 150.0A PPM METHANE
2:  500.0A/1000.0A/2000.0A PPM *CARBON DIOXIDE-2K
3:  2.5A/  5.0A/  7.5A PPM CHLORINE
4:  5.0A/ 10.0A/ 15.0A PPM NITROGEN DIOXIDE

DATE-TIME  READING
03/21/06-14:19:43  1:   0.0 PPM METHANE          P
                  M2: 1040.0 PPM *CARBON DIOXIDE-2K  P
                  3:   0.0 PPM CHLORINE          P
                  4:   0.0 PPM NITROGEN DIOXIDE P
03/21/06-14:20:07  1:   0.0 PPM METHANE          P
                  M2: 1010.0 PPM *CARBON DIOXIDE-2K  P
                  3:   0.0 PPM CHLORINE          P
                  4:   0.0 PPM NITROGEN DIOXIDE P
04/04/06-11:36:59  LOG DATA REPORT       PAGE 1