Augmented Reality Interface for the Corobot System

by

Qiyun Sun

A Project Report Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in
Computer Science

Supervised by

Dr. Zack Butler

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

May 2016
Abstract

Augmented Reality Interface for the Corobot System

Qiyun Sun

Supervising Professor: Dr. Zack Butler

Human-robot interaction has long been a challenge to the mobile robot communities. On one hand, the robot developers want to show more informative robot topics for debugging purpose. On the other hand, interactions between traversing robots and people with no robotic background are also inevitable in the robot developing environment. Thus, an cognitive but also easily interactive user interface is in demand. To coordinate the needs of both sides, by embracing the power of Augmented Reality (AR), we propose a visual-friendly interface for the corobot system. Our AR interface real-timely composes a 3D graphic scene with various kinds of visualized robot topics in a web page which can be opened from any modern web browser. The interface is mostly written in JavaScript and is built on several state-of-the-art WebGL graphic libraries. By using our interface, robot developers can creatively add new interface and visualize complex robot status, while humans in the environment can easily figure out a general idea about the robot and perform simple interactions, such as information querying.
# Contents

Abstract ................................................................. ii

1 Introduction .......................................................... 1

2 Related Work ......................................................... 4

3 Proposed System ...................................................... 6
   3.1 Rosbridge Server and Topic Tools ......................... 8
   3.2 ROS Web Video Server .................................. 8
   3.3 Corobot AR UI .............................................. 9
      3.3.1 Kinect Point Cloud Interface ..................... 10
      3.3.2 ROS Topic Visualization Interfaces .......... 11
      3.3.3 User Interaction Interfaces .................... 12
   3.4 Entrane Proxy Server .................................... 13

4 Experiments ........................................................ 16
   4.1 Human-corobot Interaction ................................ 16
   4.2 Using the CARI for Development ...................... 17

5 Conclusion ......................................................... 19

Bibliography ........................................................ 21

A API Manual .......................................................... 23
   A.1 Prerequisites ................................................ 23
   A.2 Installation ................................................. 24
   A.3 Interfaces .................................................. 25
      A.3.1 Using User Interaction Interfaces ............. 26
      A.3.2 Using ROS Topic Visualization Interfaces .... 26
      A.3.3 Creating Custom Interfaces .................. 27
List of Tables
List of Figures

1.1 The Corobot .................................................. 2
2.1 The original corobot GUI .................................. 5
3.1 The overall architecture of the CARI system ............. 7
3.2 Kinect data processing workflow with an example. The max_range is 10 meters. ........................................ 9
3.3 The CARI graphic layers ................................... 10
3.4 The standalone Kinect Point Cloud interface. The upper left image is the front view. The upper right image is the top view. The bottom image shows the KPC with an orange flag as the waypoint subscribed. The camera in the bottom scene is 0.2-meter higher then the upper left scene, which is horizontal. .................................................. 11
3.5 The pipe line from Kinect images to point cloud by using OpenGL heightmap. ...................................... 12
3.6 The ROS Topic Visualization Interfaces ................. 13
3.7 The User Interaction Interfaces .............................. 14
3.8 The forwarding method changes in the HTTP image streaming pipe line ................................. 14
3.9 The synthetic view of the CARI ............................. 15
4.1 The debugging view of the CARI. The figure shows the current robot location correctly matches the Kinect point cloud borders .................................................. 17
4.2 The debugging view of the CARI. A negative case with a pure red Kinect point cloud. The auxiliary UI panel is hidden from the screen ........................................ 18
Chapter 1

Introduction

There has been extensive work on finding ways of interaction between humans and robots, from command lines to graphic figures, from voice commands to hand/body gestures, etc. However, in order to work with all these methods above, it is necessary for the user to know some basic robotic knowledge of what the robot is built for and how the robot acts, as well as the literal commands or key actions which are specified by the robot developers for controlling the robots. Nowadays, with more and more common appearance of the mobile robots in the daily life of everyday people [1], it has become more and more important to develop an easy-to-master mobile robot graphic interface for the users without any robotic background. In the recent future, mobile robots might just be as popular as today’s smart phones.

In order to show the complex content of the mobile robot in a both informative and easy-understanding way, which matches people’s “common sense”, we adopt the idea of Augmented Reality (AR). AR applications have been widely used on procedural assistance work. A recent paper shows that creating augmented reality content from real-time video and performing accurate object tracking are valuable and scalable [10]. Our experiment platform is the corobot system which runs on the top of the Robot Operating System (ROS) [14]. And the corobot is our robot hardware as shown in Fig. 1.1. The term “corobot” is used to connote this idea of robots working alongside humans instead of in isolation [1].

In this project, our approach is to develop a graphic user interface (GUI) that puts the robot’s vision on a laptop screen and overlays it with useful information gleaned from the Kinect depth image as well as the robot’s internal states while the robot traverses the
building of the CS department of RIT. Different from other people’s work introduced in Chapter 2, our GUI fits both robot developers as well as anyone who would ever have a chance to stay with our corobots in the same place.

Through the laptop screen on the top of the corobot, by using our interface, people can mainly achieve two kinds of things:

1. **Learning the status of the robot:**  *The user can easily figure out what kind of robot it is, what task the robot is carrying, and a general idea of what the robot is capable of doing.*

2. **Interacting with the robot:**  *The user can understand what possible ways the robot is able to interact with, such as how to get helpful message from the robot or how to leave a message to other users through the robot.*

3. **Debugging the robot:**  *The robot developers can dynamically create new graphic interface to monitor or control the robot topics even when the whole system is running. Along side with the possibility of interactive programming, informative 3D graphic objects can easily be created. Thus, a cognitive developing process can be built.*

By the power of modern JavaScript engine [4], our Corobot AR Interface (CARI) gathers all kinds of information from the corobot hardware at runtime, including the inputs of
the keyboard and touch pad of the laptop as the user input, the Kinect camera data and odometer motion data as the environmental reality, and other data generated by a selected set of corobot applications as possible augmented elements. The core graphic server of our system takes the advantage of WebGL technology, converts each kind of data to its visualized form and combines all the converted graphic elements together as a 3D graphic scene on the corobot screen. To show the possibility of robot programming in both interactive and cognitive ways, an example of finding the error between the position of the robot in the real-world and the position where the robot “thinks” it is located at is introduced in Chapter 4.

The remaining chapters of the paper are presented as follows: Chapter 2 introduces mobile robot interfaces developed previously in the robotic field. Chapter 3 gives a detail description on how our system works. The real-time practicing result of our system is stated in Chapter 4. And, in Chapter 5, our comments to the AR interface system are given and the possibility of the future development is discussed.
Chapter 2

Related Work

There are two major GUI projects developed for the ROS [16, 17]. Both of the implementation are based on the idea of bringing the human-robot interaction studies online [15]. The proposed framework achieves 3D visualization of the robot topics by implementing 3D models of the robot as well as the whole environment. Their system works as an online version of Gazebo [11]. But it indicates neither the possible error between the modeling world and the real world, nor any direct connection between the robot in the real environment and the virtual robot figure on the screen. Though this shortage may be overcame by using the robot in a camera monitored environment, the difference between the virtual world and real world is still hard to be measured. In the same team’s later work [19], a depth camera based real-world-object online modeling is implemented. This method bricks the barrier between the virtual 3D world and the reality, which allows the possibility of visualizing robot topics in an augmented way.

Meanwhile, with the development of smart phones and wearable devices, such as Google Cardboard [6], Oculus Rift [9] etc., both virtual reality (VR) and augmented reality (AR) have become trends in the application of user interfaces. Instead of using either simulator or hardware environment, the idea of hybrid simulation is proposed [3]. Furthermore, instead of simply rendering virtual object into a real-world camera scene, a paradigm of cognitive AR for procedural tasks is published [10]. In the ROS community, a tryout of AR based marker pose tracking system is implemented [12]. Though there are many limitations for the marker tracking system, the cognitive idea behind the scene can be applicable to the later ROS development.
The original corobot graphic user interface (GUI) shows the current position of the corobot mainly in two forms: text based description of the coordinates of the robot and a 2D marker on the graphic map, as shown in Fig. 2.1. A Kinect based laser map for wall detection is also shown on the right part of the GUI. A simple robot’s trajectory will also be drawn while the pose of corobot changes over time.

![Figure 2.1: The original corobot GUI.](image)

By subscribing the same pose topic from the corobot system, along with both Kinect RGB image and Kinect depth image, our AR based 3D GUI takes the advantages of the previous work and achieves visual based human-robot interaction.
Chapter 3

Proposed System

The proposed AR GUI system is built on the corobot system [1]. There are four major components:

1. **Entrance Proxy Server:** the main proxy server redirects the content of both Corobot AR UI component and ROS Web Video Server component for modern browser entrance.

2. **Corobot AR UI:** the main GUI server provides interfaces for ROS topic visualization and user interaction.

3. **ROS Web Video Server:** the video server converts both raw Kinect RGB image stream and raw Kinect depth image stream into web-friendly JPEG format streams.

4. **Rosbridge Server and Topic Tools:** the bridge server provides ROS topic JavaScript interfaces for Corobot AR UI component.

Fig. 3.1 illustrates the overall architecture of our system. In general, a socket port is assigned to each component. The components forwards data to each other via socket based communication. The Rosbridge Server subscribes the text based ROS topics, converts those ROS messages into JSON messages, sends the JSON messages to Corobot AR GUI and vice versa. For Kinect video streams, the ROS Web Video Server directly subscribes the raw Kinect image topics, converts them to JPEG streams and sends the stream to the Entrance Proxy Server. When the user opens a web page client from a modern browser,
the Entrance Proxy Server will send both the content from Corobot AR GUI and ROS Web Video Server to the user-side client. The content from Corobot AR GUI includes the desired ROS messages in JSON format, the methods of robot topic visualization and user interaction, and static UI materials, such as button images, etc. The content from ROS Web Video Server is the converted JPEG image streams. The actual GUI composing work is done by the web page client in the user’s browser.

Figure 3.1: The overall architecture of the CARI system.

In the sections below, each component will be introduced from bottom to top.
3.1 Rosbridge Server and Topic Tools

The Rosbridge Server (roslibjs) is the core JavaScript library for interacting with ROS from the browser [18]. The server uses the WebSocket, a low-latency, bidirectional communication layer, to send and get messages with the web page clients.

By using the ROS topic drop tool [2], we are able to republish the data from a subscribed topic to another topic at a lower rate. Specifically, this tool is used to reduce the publishing rates of the “pose” topic. Thus, the message latency can be significantly reduced. Also, a set of interested topics can be pre-defined with this tool in a roslaunch file. e.g., if an ROS obstacle avoidance program publishes a waypoint and the waypoint needs to be displayed in the CARI, an extra topic named /cari/waypoints can be defined.

The Rosbridge Server and ROS topic drop tool together provides the full ROS topic accessing for our system.

3.2 ROS Web Video Server

The ROS Web Video Server is originally a package from ROS. It provides real-time HTTP access of ROS image topics. This module comes with ROS Indigo, but can be compiled and run on Groovy without any modifications. However, there needs an extra image format converting step when it processes the depth image directly from the OpenNI Kinect driver. A small path is applied to fix the problem. From the code snippet, the default maximum depth value is set at 10000, which is 10 meters in the real world.

```cpp
} else if (msg->encoding.find("16UC1") != std::string::npos) {
    cv::Mat depth_msg = cv_bridge::toCvCopy(msg, "32FC1") ->image;

    // Scale floating point images
    cv::Mat_<float> float_image = depth_msg;
```
double max_val = 10000;
float_image *= (255 / max_val);

img = float_image;

Figure 3.2: Kinect data processing workflow with an example. The max_range is 10 meters.

The Kinect data processing workflow is shown in Fig. 3.2. The ROS Web Video Server streams Kinect video images to the Entrance Proxy Server via HTTP socket.

### 3.3 Corobot AR UI

The Corobot AR UI is the main GUI server of the project. This component provides various methods for ROS topic visualization and user interaction. The server is built on the Node.js JavaScript engine [4] and powered by three HTML Canvas based WebGL Graphic Library: Three.js [8], P5.js [5] and Zebkit [20].

As shown in Fig. 3.3, the three graphic libraries are simply integrated in a three-layer framework and used in different scenarios:

- Three.js: the library is used to render complex 3D scene, such as the Kinect Point Cloud and other augmented 3D objects in the Camera View layer.
Figure 3.3: The CARI graphic layers.

- P5.js: the library is a JavaScript version from the visual art library named Processing. This library is good at drawing complex 2D patterns, but rather weak for a 3D scene. This library is suitable for drawing the items in both the Auxiliary UI layer the AR Overlay layer.

- Zebkit: the library provides a rich set of common UI components, such as buttons, sliders, forms, image boxes and text boxes, which let us easily draw the user interaction related components in the Auxiliary UI layer.

### 3.3.1 Kinect Point Cloud Interface

The Kinect Point Cloud (KPC) interface includes the Kinect Point Cloud generated from both Kinect RGB image and Kinect depth image, a virtual map of the environment, a grid plane showing the size and the direction of the virtual map, and a virtual flag indicates the next waypoint (if there exists one). A standalone Kinect Point Cloud interface is shown in Fig. 3.4.

The whole interface is in the Camera View layer and drawn with the Three.js. The
Figure 3.4: The standalone Kinect Point Cloud interface. The upper left image is the front view. The upper right image is the top view. The bottom image shows the KPC with an orange flag as the waypoint subscribed. The camera in the bottom scene is 0.2-meter higher than the upper left scene, which is horizontal.

Kinect Point Cloud is generated by the OpenGL heightmap method as shown in Fig. 3.5. A pre-defined map of the Golisano building is drawn on the “floor” of the scene, along with a scalable and rotatable grid map. The KPC interface also subscribes the “pose” topic which is used to match both the size and the direction of ground map with the point cloud. A waypoint topic can be subscribed and be drawn in the scene as the orange flag in Fig. 3.4.

3.3.2 ROS Topic Visualization Interfaces

The ROS Topic Visualization Interfaces (RTVI) by default, includes a snapshot of the Kinect RGB view at the upper left corner of the scene and a round navigation map at the lower left corner of the scene, as shown in Fig. 3.6. The interfaces is located in the AR Overlay layer and are drawn by using the P5.js library.

The RTVI subscribes Kinect RGB image stream from the Entrance Proxy Server, the pose and the waypoint topics from the corobot system. The 2D navigation map shows the
robot position where the robot “thinks” it is at. If a waypoint is added and is able to be observed on the screen, a green arrow which indicates the robot moving direction will also be drawn, pointing towards the waypoint location from the middle point of the bottom edge of the screen.

3.3.3 User Interaction Interfaces

The user interaction interface is drawn by using the Zebkit library in the Auxiliary UI layer. As shown in Fig. 3.7, there are currently three sample interactions:
When the user clicks the “Save” button on the “Message Query” tab, the text inputed by the user will be recorded on the information panel below. The user can see the messages left by other users and input new messages via the keyboard on the laptop.

When the user inputs an interested ROS topic with its correct type, and clicks the “Subscribe” button on the “ROS Topic Query” tab, the related topic data will show on the information panel below, if there is one being published.

When the developer wants to observe the Kinect point cloud with a single color instead of the RGB image as texture, a click on the “Enter collision detection mode” button on the “Control” tab would let the system render the KPC with pure green points.

3.4 Entrane Proxy Server

This server is the main entrance of the CARI. The main reason of using a proxy server here is the browser regulation of different domain resource accessing with cross-origin request in HTTP access control (CORS). In general, a browser won’t allow one domain change
the web content from another domain and display the altered content. Requests from same IP address but different ports are considered cross-origin. In the case of our system, we design to use the methods from the domain of Corobot AR UI to process the content from the domain of the ROS Web Video Server.

Another reason is that the ROS Web Video Server pushes images to the clients by using multipart XHR response. This is no longer supported by many browsers [13]. As shown in Fig. 3.8, the Entrane Proxy Server redirects the image streams to the main AR UI server with HTTP Server-Sent Events (SSE).
Finally, to the user-end browser, all the data is coming a single socket of the server machine. A synthetic view of all the interfaces is shown in Fig. 3.9

Figure 3.9: The synthetic view of the CARI.
Chapter 4

Experiments

In this chapter, we show the CARI works under a general human-corobot interaction mode, while the corobot is traversing the third floor of the Golisano building. Also, we introduce a paradigm of how to use the CARI in the robot software development.

4.1 Human-corobot Interaction

Several cases of human-corobot interaction were observed during the debugging process of this project. Many people show interests about the corobot, especially when the corobot is moving. But none of them tried touching the keyboard of the corobot. The same situation happens when the corobot is left alone in the hallway during the time between classes on the Monday of the week before the finals. The CARI is remotely opened in the browser of the laptop mounted on the corobot, with the message query tab shown on top of all the other UI components. People often give a glance at the corobot but with no further interaction.

Also, three students were asked to leave a comment about the CARI. All of them think the interface is interesting. And the corobot seems to be helpful if there are useful information can be found by using the message query interface. But to most people in the environment, since there is no meaningful information for them on the robot, the message query interface is not quite attractive at the current stage. One person also mentioned that using the keyboard on the laptop is not a friend way for him to interact with the robot.

Though the interfaces work functionally, let the robot be our friend is not an easy task [7]. Perhaps, a demonstration among robot enthusiasts or adding an interface of sound
interaction would give us a more informative result. Furthermore, a more automatic user interface, with the function of meaningful purposes, such as deliver a message or a note to a specific person in a specific location, would be a better approach.

4.2 Using the CARI for Development

Here is an example of using the CARI during a common robot software development, for detecting the possible error between the position of the robot in the real world and the position of the robot indicated by a localization algorithm.

The first step is to launch the robot with its ROS launch file. Before launching the CARI, it is recommended to remap the “pose” topic and the “waypoints” topic to “/cari/pose” and “/cari/waypoints”. Then, launch the CARI system, go to the “Control” tab and click “Enter collision detection mode”. The Kinect point cloud becomes pure green, which is more easier for observing the difference between border line of the objects on the ground map and the Kinect point cloud. Fig. 4.1 shows a sample view of CARI at this stage.

Figure 4.1: The debugging view of the CARI. The figure shows the current robot location correctly matches the Kinect point cloud borders.

A negative case is shown in Fig. 4.2. The left border of the Kinect point cloud goes
through the virtual wall indicated by the ground map. The color of the Kinect point cloud can be changed dynamically in the JavaScript at the browser side and the auxiliary UI panel can be temporarily removed from the screen for a wider view of the collision scene.

Figure 4.2: The debugging view of the CARI. A negative case with a pure red Kinect point cloud. The auxiliary UI panel is hidden from the screen.

Further more, the distance and the angle between the border of the Kinect point cloud and the border of the ground map can be quantized on a graph, by using OpenGL projection methods, such as ray tracing and shader framebuffer object for collision detection. Then, the error data can be measured by converting the metric on the graph back to the metric in the real world. The whole collision detection process can be automated and dynamically gives the feedback of the error of the robot position to the localization system. Thus, an automatic robot position correction can be performed during the robot’s path execution process. This application of our system can also be used to check if the Kinect is correctly calibrated.
Chapter 5

Conclusion

In this capstone project, we present a novel augmented reality based graphic user interface which is built on the Corobot System. Our system generates a 3D graphic scene from Kinect camera views as well as converts each selected ROS topic into its single or multiple visualized forms in runtime. A brief message query interface is also implemented for interaction with human in the environment via a web page on the laptop screen.

The user interaction experiment shows that people do have more interests to interact with our corobot if there are interfaces designed for this purpose on the robot. But further development of a whole set of user interaction modules may be required to let the people and the corobot approach to each other in a more active way. Moreover, as the development of various input methods, such as gesture recognition, speech recognition, etc., the way of the interaction between human and robot can be vary as well. Though there is still a way for letting the corobot become a helpful friend, who can provide a map direction for a lost new student, show class schedule change information in the hallway, and perhaps show someone’s current location if he/she wants to be found, our project contributes a promising start in the study of this topic.

For using the CARI in the mobile robot development, as the example experiment shows, by default, our system provides several visualized interfaces for common ROS topics, such as the pose of the robot and the waypoint for robot path execution. The developers can easily wrap the interfaces, create new interfaces and change the way of the visualization of the interface according to their needs in runtime. By using our proposed paradigm, the process of the robot software development could become interactive and cognitive. Though
the knowledge of JavaScript and basic computer graphics are required for the advanced usage of the system, such technologies can be easily mastered in just a few tryouts.

The further development of this project can be conducted along with other robot development. The developer can improve or add new interfaces for our system. And the CARI system can simultaneously provide debugging information as feedbacks for the developer’s project. Considering this mutual interactive developing pattern, from one aspect, one could say that the corobot participates in the developing process. The idea of “robots working alongside humans” is exactly the goal of the corobot.
Bibliography


Appendix A

API Manual

This manual includes the steps of installation and running the Corobot Augmented Reality Interface, as well as instructions for using the default interfaces, modifying the default interfaces and creating new interfaces.

All the related source files implemented by us are located at Github pages:

https://github.com/lukeqsun/web_video_server.git
https://github.com/lukeqsun/corobot_ar_gui.git

A.1 Prerequisites

The interface is tested on Ubuntu 12.04 and Ubuntu 14.04. As a corobot based system, both the ROS and Corobot System needs to be installed before the installation. The Node.js JavaScript engine is needed as well.

For the installation of the Ubuntu OS: https://help.ubuntu.com/community/Installation.

For the installation of the ROS: http://wiki.ros.org/indigo/Installation/Ubuntu. The CARI alone works from ROS Fuerte to the ROS Jade. Indigo is the recommended stable version. (http://wiki.ros.org/Distributions)

For the installation of the Corobot System: https://github.com/corobotics/corobots. Only works on ROS Fuerte but can be remotely used with CARI in Indigo.

For the installation of the Node.js: https://github.com/creationix/nvm. The version used in this project is 5.5.
A.2 Installation

A few extra ROS packages are needed for this project:

- The Rosbridge server:

  ```
  sudo apt-get install ros-indigo-rosbridge-server
  ```

- The modified version of web video server:

  ```
  sudo apt-get install ros-indigo-async-web-server-cpp
  git clone https://github.com/lukeqsun/web_video_server.git
  mv web_video_server ~/catkin_ws/src
  cd ..
  catkin_make
  ```

To install the CARI:

```
git clone https://github.com/lukeqsun/corobot_ar_gui.git
mv corobot_ar_gui ~/catkin_ws/src
cd ~/catkin_ws/src/corobot_ar_gui
npm install
cd ~/catkin_ws
catkin_make
```

To starting the CARI with saved test data, open the file at

```
corobot_ar_gui/config/source.json
```

change the “setting” option to “0” and uncomment the area marked “Test Data” in the files listed below:

```
corobot_ar_gui/publish/javascripts/three/kinect_point.js
corobot_ar_gui/publish/javascripts/process/ar.js
corobot_ar_gui/publish/javascripts/process/aui.js
```

Then, launch the system directly with the ROS launch file:

```
roslaunch corobot_ar_gui ar_gui.launch
```
For starting the CARI with corobot or Gazebo simulator, open the file at

```
corobot_ar_gui/config/source.json
```

change the “setting” option to 2. Then, launch the CARI system after launch the Corobot System:

```
roslaunch corobot_bringup minimal.launch
roslaunch corobot_bringup 3dsensor.launch
roslaunch corobot_bringup app.launch
rosrun topic_tools drop pose 9 10 cari/pose
rosrun topic_tools drop waypoints 1 1 cari/waypoints
roslaunch corobot_ar_gui ar_gui.launch
```

In the end, open the CARI web client in a modern browser (Chrome is the recommended one.). By default, the address listed below is used:

```
http://127.0.0.1:3000
```

A.3 Interfaces

According to the usages, there are two kinds of interfaces: User Interaction Interfaces, ROS Topic Visualization Interfaces. Each interface is under the directory named after the graphic library used in the interface. The files under

```
corobot_ar_gui/publish
```

will be sent to each connected web client and be executed there. All the other files are mainly for building the framework of the system, which needs to be maintained only if there are API changes during the updating of the dependent libraries.

```
corobot_ar_gui/views/index.jade

corobot_ar_gui/publish/javascripts/main.js
```
A.3.1 Using User Interaction Interfaces

There are three interfaces for user interaction, all of them are defined in the zebraCanvas module, which is related to files at

corobot_ar_gui/publish/javascripts/ui

- `ui.js` composes all the other modules using Zebra library.
- `basic.js` provides several custom UI components, such as a framed surrounding box, etc.
- `view_panel.js` provides the surrounding box for the thumbnail of the Kinect RGB images.
- `roslib_panel.js` contains all the three tabs of the user interaction interfaces.

The interfaces are also implemented in this file, as “messagePage”, “modePage” and “topicPage”.

A.3.2 Using ROS Topic Visualization Interfaces

The ROS Topic Visualization Interfaces are implemented by using two different libraries: Three.js and P5.js.

The folder

corobot_ar_gui/publish/javascripts/three

contains the main Kinect Point Cloud scene in the kinect_point.js, along with its vertex shader kinect_point_vs.js and its frame shader kinect_point_fs.js

The KPC scene includes the grid lines, the Kinect point mesh, the ground map panel and the waypoint flag panel. A general three.js animation framework is also added there to control the frames.

The folder

corobot_ar_gui/publish/javascripts/process
contains the rest one AR interfaces: the waypoint pointer and two auxiliary interface: Kinect RGB thumbnail and the navigation map.

- **rgbd.js** defines the Kinect RGB thumbnail which is shown at the upper left corner of the screen.

- **aui.js** defines the navigation map. The navigation map can be zoomed with the slider bar and moved by mouse grabbing inside its purple circle area.

- **ar.js** defines the waypoint pointer, which is the green segment arrow, always pointing at the next waypoint if there is one.

### A.3.3 Creating Custom Interfaces

Below is the recommended steps for adding and new visual interface to the CARI system:

1. Decide what kind of graphic library to use for the new interface, put the main UI files inside a related existing folder or create a new folder.

2. Link the new files in the

   `corobot_ar_gui/views/index.jade`

3. Integrate the main interface file to its proper layer in the framework. More layers can be created in the index.jade if necessary.

4. Start implementing the new interface. Link its data model either at the global data model space in the main.js or in its own locations.

5. Test and debug the new interface, and use its feedback to help the robot development.