Culturally Sensitive Social Robotics
for Africa

D5.4.2 Robot Mission Language

Due date: **30/06/2024**
Submission Date: **05/03/2025**
Revision Date:

Start date of project: **01/07/2023**

Duration: **36 months**

Lead organisation for this deliverable: **Carnegie Mellon University Africa (for Wits)**

Responsible Person:

Revision: **1.0**

Project funded by the African Engineering and Technology Network (Afretec) Inclusive Digital Transformation Research Grant Programme		
Dissemination Level		
PU	Public	**PU**
PP	Restricted to other programme participants (including Afretec Administration)	
RE	Restricted to a group specified by the consortium (including Afretec Administration)	
CO	Confidential, only for members of the consortium (including Afretec Administration)	

Executive Summary

This deliverable represents the outcome of Task 5.4.2. It comprises four elements: (i) a mode of abstract modelling — behavior trees — that can be used to formally specify the interactions in the use case scenarios and enact them in a culturally sensitive manner using the culture knowledge base and an environment knowledge base, (ii) two files containing a specification of the two use case scenarios using behavior trees, (iii) an environment knowledge base file with the information required to complete the robot mission, and (iv) the documented software required to compile a C++ helper class `EnvironmentKnowledgeBase` to read the environment knowledge base file, store the knowledge, and make the knowledge accessible through a suite of access methods. As such, this deliverable provides the input for the development in Task 5.4.3 of an interpreter that can translate this abstract specification into robot actions, thereby enacting the use case scenarios defined in Tasks 2.1, 2.2, and 2.3 and documented in Deliverables D2.1, D2.2, and D2.3.

In the work plan, this deliverable and deliverable D5.4.3 were assigned to the University of the Witswatersrand. However, the material in this report was developed and written by Carnegie Mellon University Africa. This was necessary because of unavoidable delays in the completion of the associated task by Wits, and because the robot mission language and the robot mission interpreter, are essential for integrating and demonstrating the use case scenarios.

Contents

1 Introduction

This deliverable represents the outcome of Task 5.4.2. It has four sections.

Section 2 addresses the specification of robot missions using behavior trees, a popular alternative to state machines as a model of abstract modelling to formally specify the interactions in the use case scenarios.

Section 3 provides the behavior tree specification for the two CSSR4Africa use case scenarios, the robot laboratory tour robot mission in Section 3.1 and the receptionist robot mission in Section 3.2.

Since we we are particularlarly focussed on enacting these missions in a culturally sensitive manner, we require both a cultural knowledge ontology & culture knowledge base, and an environment knowledge ontology & environment knowledge base. The former is described in Deliverable D.5.4.1, while the latter is described in Section 4 of this deliverable.

The deliverable concludes with Section 5 which addresses the implementation of the environment knowledge base and, specifically, with the description of a C++ helper class to read the environment knowledge base file, store the knowledge, and make the knowledge accessible through a suite of access methods. As such, it provides the input for the development in Task 5.4.3 of an interpreter that can translate the abstract behavior tree specifications in Sections 3.1 and 3.2 into robot actions, thereby enacting the use case scenarios defined in Tasks 2.1, 2.2, and 2.3 and documented in Deliverables D2.1, D2.2, and D2.3.

In the work plan, this deliverable and deliverable D5.4.3 were assigned to the University of the Witswatersrand. However, the material in this report was developed and written by Carnegie Mellon University Africa. This was necessary because of unavoidable delays in the completion of the associated task by Wits, and because the robot mission language and the robot mission interpreter, are essential for integrating and demonstrating the use case scenarios.

2 Specification of Robot Missions using Behavior Trees

Pending completion.

3 Use Case Scenario Robot Missions

3.1 Lab Tour Robot Mission Behavior Tree

Pending completion.

3.2 Receptionist Robot Mission Behavior Tree

Pending completion.

4 Environment Knowledge Ontology and Knowledge Base

Figure 1 presents a simple ontology of environment knowledge. In this ontology, internal nodes in the ontology tree form the key in the environment knowledge base, e.g., `robotLocation`. Leaf nodes represent the data entities and their types. This allows multiple elements in a value for each key, e.g., `robotLocation 3 15.2 9.0 45.0`. The identification number value element associated with each key is the means by which the different elements an environment location — robot location, location description, gesture target, pre-gesture message, post-gesture message — are related. The tour specification identifies the number and sequence of locations to be visited in the tour.

```
Environment Knowledge
 |
 |__Robot Location
 |   |__Identification Number (integer)
 |   |__Pose
 |       |__x (float)
 |       |__y (float)
 |       |__theta (float)
 |__Robot Location Description
 |   |__Identification Number (integer)
 |   |__Description (string)
 |__Gesture Target
 |   |__Identification Number (integer)
 |   |__Position
 |       |__x (float)
 |       |__y (float)
 |       |__z (float)
 |__Pre-gesture Message
 |   |__Identification Number (integer)
 |   |__Description(string)
 |__Post-gesture Message
 |   |__Identification Number (integer)
 |   |__Description(string)
 |__Tour Specification
     |__Number of Locations (integer)
     |__Location Sequence
         |__Identification Number (integer),
         |__Identification Number (integer),
         |__...
         |__Identification Number (integer)
```

Figure 1: Environment knowledge ontology.

Table 1 lists the key-value pairs, i.e., each key and the associated multiple numeric or alphanumeric elements of the value that encapsulate the environment knowledge. These numeric or alphanumeric values can then be used directly in the robot mission interpreter, i.e., the `behaviorController` ROS node, and passed as arguments in the service requests it issues to the nodes in the system architecture to conduct a tour or provide directions as a response to an enquiry at reception.

The key-value pairs are stored in a file `environmentKnowledgeBaseInput.dat`. This file is read and the value-pairs are accessed using a helper class `EnvironmentKnowledgeBase` described in Section 5.

Environment Knowledge		
Key	Values	Units
`RobotLocation`	`<IDNumber> <x> <y> <theta>`	Metres, degrees
`RobotLocationDescription`	`<IDNumber> <text>`	String
`GestureTarget`	`<IDNumber> <x> <y> <z>`	Metres
`PreGestureMessage`	`<IDNumber> <text>`	String
`PostGestureMessage`	`<IDNumber> <text>`	String
`TourSpecification`	`<n> <ID1>, <ID2>, ... . <IDn>`	

Table 1: Key-value pairs for specifying environment knowledge actions using the ontology depicted in Figure 1. As noted above, the identification number element of the value associated with each key is the means by which the robot location, the location description, the gesture target, the pre-gesture message, the post-gesture message are related. The tour specification identifies the number of locations and the sequence of locations to be visited in the tour.

5 Environment Knowledge Base Implementation

The key-value pairs listed in Tables 1, comprising an alphanumeric key and associated numeric or symbolic values that encapsulate the environment knowledge, are stored in a file named `environmentKnowledgeBaseInput.dat`. This file is accessed using a C++ helper class `EnvironmentKnowledgeBase` described in this section. Specifically, a C++ object instantiation of the helper class reads the environment knowledge base file, store the knowledge, and make the knowledge accessible through three public access methods. The remainder of this section details the implementation of this C++ helper class.

5.1 File Organization

The files for the environment knowledge base helper class are located in the `utilities` subdirectory, as shown in Figure 2. The constituent files are organized is several subdirectores as shown in Figure 3. There are three C++ source code files: `environmentKnowledgeBaseApplication.cpp`, `environmenteKnowledgeBaseImplementation.cpp`, and `environmentKnowledge.h`. The implementation file contains the helper class definition. The interface file contains the helper class declaration. The application file is essentially a unit test to illustrate how the helper class is used and to verify that it works correctly. It instantiates a C++ helper class object which reads the environment knowledge base file, and uses the access method to retrieve values in the environment knowledge base, implemented using a binary search tree dictionary data structure, write them to the terminal.

```
workspace
└── pepper_rob_ws
    ├── build
    ├── devel
    └── src
        ├── cssr4africa
        │   ├── cssr_system
        │   ├── pepper_interface_tests
        │   ├── system_tests
        │   ├── unit_tests
        │   └── utilities
        ├── naoqi_dcm_driver
        ├── naoqi_driver
        ├── pepper_dcm_robot
        ├── pepper_moveit_config
        ├── pepper_robot
        └── pepper_virtual
```

Figure 2: Directory structure for the CSSR4Africa software repository.

```
utilities
├── config
│   └── environmentKnowledgeBaseConfiguration.ini
├── data
│   └── environmentKnowledgeBaseInput.dat
├── include
│   └── utilities
│       └── environmentKnowledgeBaseInterface.h
├── src
│   ├── environmentKnowledgeBaseApplication.cpp
│   └── environmentKnowledgeBaseImplementation.cpp
├── launch
│   └── environmentKnowledgeBaseExample.launch
├── README.md
└── CMakeLists.txt
```

Figure 3: Directory Structure for the `EnvironmentKnowledgeBase` C++ helper class.

5.2 Configuration File

The population of the knowledge base is determined by the contents of a configuration file `environmentKnowledgeBase.ini` that contain a list of key-value pairs, as shown below in Table 2.

The configuration file is named `environmentKnowledgeBaseConfiguration.ini`.

Table 2: Configuration file for the `EnvironmentKnowledgeBase` helper class.

Key	Value	Description
`knowledgeBase`	`environmentKnowledgeBaseInput.dat`	Specifies the filename of the file in which the cultural knowledge key-value pairs are stored.
`verboseMode`	`true` or `false`	Specifies whether diagnostic data is to be printed to the terminal.

5.3 Environment Knowlege Base

The environment knowledge base file comprises a list of key-value pairs as shown in Table 3.

Table 3: Key-value pairs listed in the knowledge base file `environmentKnowledgeBaseInput.dat`.

```
robotLocationDescription 1 Pepper's starting location
robotLocation            1 2.6 8.1 -90
gestureTarget            1 0.0 0.0 0.0
preGestureMessage        1 Welcome the the robotics lab at Carnegie Mellon University Africa
postGestureMessage       1 I hope you enjoy the tour

robotLocationDescription 2 The (other) Pepper robot
robotLocation            2 2.6 8.1 -45
gestureTarget            2 3.2 8.4 0.82
preGestureMessage        2 This is the Pepper humanoid robot
postGestureMessage       2 We use it for research in social robotics and human-robot interaction

robotLocationDescription 3 Lynxmotion
robotLocation            3 2.0 6.3 -45
gestureTarget            3 0.6 4.8 0.82
preGestureMessage        3 This is the Lynxmotion robot
postGestureMessage       3 We use it for teaching robot manipulation

robotLocationDescription 4 Roomba
robotLocation            4 5.0 3.9 110
gestureTarget            4 6.84.8 0.82
preGestureMessage        4 This is the Roomba
postGestureMessage       4 We use it for teaching mobile robotics

robotLocationDescription 5 Pepper's starting location
robotLocation            5 2.6 8.1 -90
gestureTarget            5 0.0 0.0 0.0
preGestureMessage        5 I hope you enjoyed the tour
postGestureMessage       5 See you again soon

tourSpecification 5 1 4 3 2 5
```

5.4 Output Data File

There is no output data file for the environment knowledge base helper class.

5.5 Class Definition

Instantiating the `EnvironmentKnowledgeBase` class as a C++ object causes the contents of the environment knowledge base file to be read and stored in private dictionary data structure. Diagnostic messages are printed on the screen, depending on the value of `verboseMode` key in the configuration file. The contents of the dictionary are accessed using the identification number. Appendix A provides the full definition of the `EnvironmentKnowledgeBase` class.

5.5.1 Constructor

The `EnvironmentKnowledgeBase()` constructor reads the configuration file to determine the mode of operation, the name of the knowledge base value types file, and the name of the knowledge base file. It sets a private data member flag with the mode of operation, initializes the private dictionary data structure with the key-value pairs read from the knowledge base file. If operating in verbose mode, it echoes the keys and values to the terminal.

5.5.2 Destructor

The `˜EnvironmentKnowledgeBase()` destructor deletes the dictionary data structure and write a diagnostic message if in verbose mode.

5.5.3 Private Data

The dictionary is implemented using a binary search tree with an element of type `struct KeyValueType`, with six fields.

```
typedef struct {
   float x;
   float y;
   float theta;
} RobotLocationType;

typedef struct {
   float x;
   float y;
   float z;
} GestureTargetType;

typedef  struct {
    int                   key; // location identification number
    RobotLocationType     robotLocation;
    char                  robotLocationDescription[STRING_LENGTH];
    GestureTargetType     gestureTarget;
    char                  preGestureMessage[STRING_LENGTH] ;
    char                  postGestureMessage[STRING_LENGTH] ;
} KeyValueType;
```

The first field `key` is the identification number for this location. The data type is integer. This is the key that is used to access data in the binary search tree dictionary data structure.

The second field `robotLocation` is a structure with three fields containing the x, y, and θ floating point values that specify the pose of the robot at this location.

The third field `robotLocationDescription` is a description of this robot location. The data type is a C-string, i.e., a null-terminated array of characters.

The fourth field `gestureTarget` is a structure with three fields containing the x, y, and z floating point values that specify the position of the target to which the robot is to gesture.

The fifth field `preGestureMessage`is a message to be spoken by the robot prior to executing the gesture. The data type is a C-string, i.e., a null-terminated array of characters.

The sixth field `postGestureMessage` is a message to be spoken by the robot after executing the gesture. The data type is a C-string, i.e., a null-terminated array of characters.

In addition to the binary search tree dictionary data structure, there is also a data structure `tourSpecification` to specify the tour. This is a structure with two fields: an integer specifying the number of robot locations in a tour and an array of integer identification numbers specifying sequence of robot locations that the robot should visit during the tour, in the order in which they are stored in the array.

```
typedef struct {
   int numberOfLocations;
   int locationIdNumber[MAX_NUMBER_OF_TOUR_LOCATIONS];
} TourSpecificationType;
```

There are also a small number of other private utility data fields to store the configuration filename, the configuration data, a keyValue, and the verbose mode flag.

5.5.4 Public Access Methods

There are three public methods, one to print the knowledge base to the screen, one to retrieve a key-value pair, given the identification number of the location, and one to retrieve the tour specification. These are `printToScreen()`, `getValue()`, and `getTour()`, respectively.

The `printToScreen()` method does not have any parameters.

The `getValue()` method has two parameters: a key and a value, as follows.

```
bool getValue(int idNumber, KeyValueType *keyValue);
```

The method returns `true` if the key value was successfully retrieved from the knowledge base, `false` otherwise.

The `getTour()` method has one parameter: the tour data, as follows.

```
bool getTour(struct TourSpecificationType *tourSpecification);
```

The method returns `true` if the tour was successfully retrieved from the knowledge base, `false` otherwise.

6 Example Application

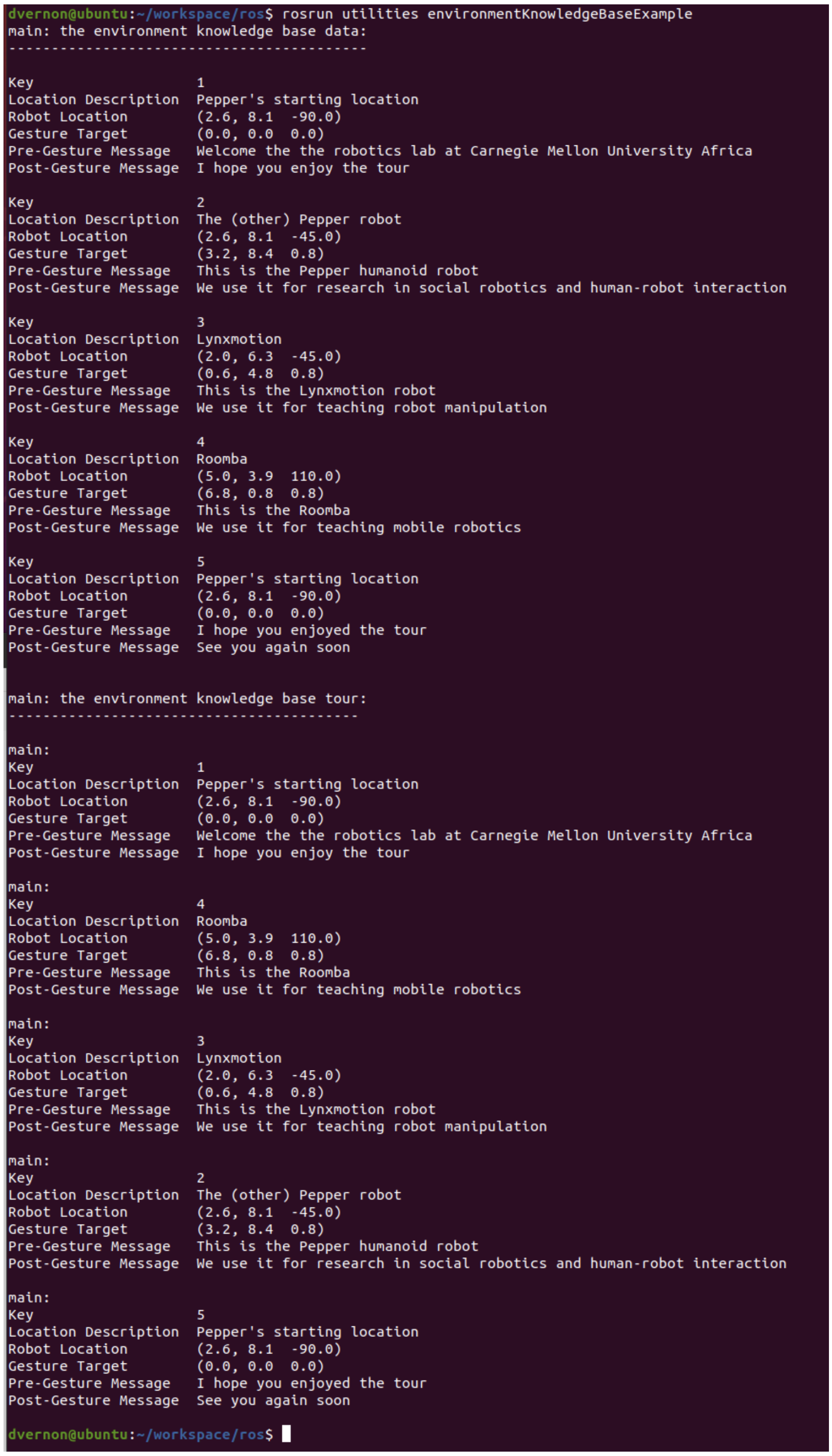
The example application in `environmentKnowledgeBaseApplication.cpp` illustrates the use of the class to read the environment knowledge base file and print each key-value pair, when each value has multiple elements. It also provides an example of how to retrieve the values associated with a robot location given by its identification number, and example of how to retrieve the sequence of robot locatiions in a tour.

```
#include <utilities/environmentKnowledgeBaseInterface.h>

int main() {

   KeyValueType          keyValue; // structure with key and values
   TourSpecificationType tour;     // list of tour locations
   int                   idNumber; // location id
   int                   i;        // counter

   /* instantiate the environment knowledge base object                      */
   /* this reads the knowledge value types file and the knowledge base file  */
   /* as specified in the environmentKnowledgeBaseConfiguration.ini file     */

   EnvironmentKnowledgeBase knowledgebase;

   /* verify that the knowledge base was read correctly */

   printf("main: the environment knowledge base data:\n");
   printf("-----------------------------------------\n\n");

   knowledgebase.printToScreen();


   printf("main: the environment knowledge base tour:\n");
   printf("-----------------------------------------\n\n");

   knowledgebase.getTour(&tour);

   /* query the contents of the knowledge base:               */
   /* retrieve all the locations on a tour                    */
   /* and print them in the order in which they are specified */

   for (i = 0; i <= tour.numberOfLocations; i++) {
      idNumber = tour.locationIdNumber[i];
      if (knowledgebase.getValue(idNumber, &keyValue) == true) {
         printf("main:\n"
         "Key                   %-4d \n"
         "Location Description  %s \n"
         "Robot Location        (%.1f, %.1f  %.1f)\n"
         "Gesture Target        (%.1f, %.1f  %.1f) \n"
         "Pre-Gesture Message   %s \n"
         "Post-Gesture Message  %s \n\n",
         keyValue.key,
         keyValue.robotLocationDescription,
         keyValue.robotLocation.x, keyValue.robotLocation.y, keyValue.robotLocation.theta,
         keyValue.gestureTarget.x, keyValue.gestureTarget.y, keyValue.gestureTarget.z,
         keyValue.preGestureMessage,
         keyValue.postGestureMessage);
      }
   }
}
```

Run the application by entering the following command:

```
rosrun utilities environmentKnowledgeBaseExample
```

This assumes the existence of a `utilities` package, as shown in Figures 2 and 3, and that the package has been installed, as described in Deliverable D3.3 Software Installation Manual.
A screenshot of the output of running this application is shown in Figure 4.



```
dvernon@ubuntu:~/workspace/ros$ rosrun utilities environmentKnowledgeBaseExample
main: the environment knowledge base data:
-----------------------------------------

Key                   1
Location Description  Pepper's starting location
Robot Location        (2.6, 8.1  -90.0)
Gesture Target        (0.0, 0.0  0.0)
Pre-Gesture Message   Welcome the the robotics lab at Carnegie Mellon University Africa
Post-Gesture Message  I hope you enjoy the tour

Key                   2
Location Description  The (other) Pepper robot
Robot Location        (2.6, 8.1  -45.0)
Gesture Target        (3.2, 8.4  0.8)
Pre-Gesture Message   This is the Pepper humanoid robot
Post-Gesture Message  We use it for research in social robotics and human-robot interaction

Key                   3
Location Description  Lynxmotion
Robot Location        (2.0, 6.3  -45.0)
Gesture Target        (0.6, 4.8  0.8)
Pre-Gesture Message   This is the Lynxmotion robot
Post-Gesture Message  We use it for teaching robot manipulation

Key                   4
Location Description  Roomba
Robot Location        (5.0, 3.9  110.0)
Gesture Target        (6.8, 0.8  0.8)
Pre-Gesture Message   This is the Roomba
Post-Gesture Message  We use it for teaching mobile robotics

Key                   5
Location Description  Pepper's starting location
Robot Location        (2.6, 8.1  -90.0)
Gesture Target        (0.0, 0.0  0.0)
Pre-Gesture Message   I hope you enjoyed the tour
Post-Gesture Message  See you again soon


main: the environment knowledge base tour:
-----------------------------------------

main:
Key                   1
Location Description  Pepper's starting location
Robot Location        (2.6, 8.1  -90.0)
Gesture Target        (0.0, 0.0  0.0)
Pre-Gesture Message   Welcome the the robotics lab at Carnegie Mellon University Africa
Post-Gesture Message  I hope you enjoy the tour

main:
Key                   4
Location Description  Roomba
Robot Location        (5.0, 3.9  110.0)
Gesture Target        (6.8, 0.8  0.8)
Pre-Gesture Message   This is the Roomba
Post-Gesture Message  We use it for teaching mobile robotics

main:
Key                   3
Location Description  Lynxmotion
Robot Location        (2.0, 6.3  -45.0)
Gesture Target        (0.6, 4.8  0.8)
Pre-Gesture Message   This is the Lynxmotion robot
Post-Gesture Message  We use it for teaching robot manipulation

main:
Key                   2
Location Description  The (other) Pepper robot
Robot Location        (2.6, 8.1  -45.0)
Gesture Target        (3.2, 8.4  0.8)
Pre-Gesture Message   This is the Pepper humanoid robot
Post-Gesture Message  We use it for research in social robotics and human-robot interaction

main:
Key                   5
Location Description  Pepper's starting location
Robot Location        (2.6, 8.1  -90.0)
Gesture Target        (0.0, 0.0  0.0)
Pre-Gesture Message   I hope you enjoyed the tour
Post-Gesture Message  See you again soon

dvernon@ubuntu:~/workspace/ros$
```

Figure 4: Screenshot of the output of running the example application.

Appendix A The EnvironmentKnowledgeBase Class

Note: documentation comments for the private methods have been removed due to space constraints but are retained in the source file.

```
#define NUMBER_OF_CONFIGURATION_KEYS  3
#define NUMBER_OF_VALUE_KEYS          7
#define MAX_NUMBER_OF_TOUR_LOCATIONS 20

typedef char Keyword[KEY_LENGTH];

typedef struct {
   char knowledgeBase[MAX_FILENAME_LENGTH];
   bool verboseMode;
} ConfigurationDataType;

typedef struct {
   float x;
   float y;
   float theta;
} RobotLocationType;

typedef struct {
   float x;
   float y;
   float z;
} GestureTargetType;

typedef struct {
   int numberOfLocations;
   int locationIdNumber[MAX_NUMBER_OF_TOUR_LOCATIONS];
} TourSpecificationType;

typedef  struct {
    int                   key; // i.e., idNumber
    RobotLocationType     robotLocation;
    char                  robotLocationDescription[STRING_LENGTH];
    GestureTargetType     gestureTarget;
    char                  preGestureMessage[STRING_LENGTH] ;
    char                  postGestureMessage[STRING_LENGTH] ;
} KeyValueType;

typedef struct node *NodeType;

typedef struct node {
            KeyValueType keyValue;
            NodeType left, right;
         } Node;

typedef NodeType BinaryTreeType;

typedef BinaryTreeType WindowType;

class EnvironmentKnowledgeBase {

public:
   EnvironmentKnowledgeBase();
   ~EnvironmentKnowledgeBase();

   bool                  getValue(int key, KeyValueType *keyValue);
   bool                  getTour(TourSpecificationType  *tour);
   void                  printToScreen();

private:
   BinaryTreeType        tree = NULL;
   TourSpecificationType tourSpecification;
   KeyValueType          keyValue;
   ConfigurationDataType configurationData;
   char                  configuration_filename[MAX_STRING_LENGTH] = "environmentKnowledgeBaseConfiguration.ini";

   BinaryTreeType        *delete_element(KeyValueType keyValue, BinaryTreeType *tree);
   KeyValueType          delete_min(BinaryTreeType *tree);
   bool                  getValue(int key, KeyValueType *keyValue, BinaryTreeType *tree);
   void                  initialize(BinaryTreeType *tree);
   int                   inorder_print_to_file(BinaryTreeType tree, int n, FILE *fp_out);
   int                   inorder_print_to_screen(BinaryTreeType tree, int n);
   BinaryTreeType        *insert(KeyValueType keyValue, BinaryTreeType *tree, bool update);
   int                   postorder_delete_nodes(BinaryTreeType tree);
   int                   print_to_file(FILE *fp_out);
   int                   print_to_file(BinaryTreeType tree, FILE *fp_out);
   int                   print_to_screen(BinaryTreeType tree);
   void                  readConfigurationData();
   void                  readKnowledgeBase();
};
```

Principal Contributors

The main authors of this deliverable are as follows (in alphabetical order).

Tsegazeab Tefferi, Carnegie Mellon University Africa.
David Vernon, Carnegie Mellon University Africa.

Document History

Version 1.0

Partial version to address the specification of the environment knowledge base and `EnvironmentKnowledgeBase` helper class.
David Vernon.
5 March 2025.