CSSR for

Culturally Sensitive Social Robotics for Africa

D5.2 Animate Behaviour Subsystem

Due date: **31/03/2024**
Submission Date: **21/12/2024**
Revision Date: **21/12/2024**

Start date of project: **01/07/2023** Duration: **36 months**

Lead organisation for this deliverable: **Carnegie Mellon University Africa**

Responsible Person: **Eyerusalem Mamuye Birhan** 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

Deliverable D5.2 of the Animate behaviour Subsystem, part of the CSSR4Africa project, aims to enhance the robot's lifelike appearance through random movements. This deliverable seeks to create an engaging and interactive experience by implementing a robust ROS-based module named `animateBehaviour`. The module controls the actuators of the Pepper robot to perform random, subtle body movements, flex its hands, and slightly rotate its base along the Z-axis. All joints, except for *headYaw* and *headPitch*, are actuated to generate these motions.

The `animateBehaviour` node generates movements in a randomized pattern and advertises the `/animateBehaviour/set_activation` service to enable or disable its operation, allowing dynamic activation and deactivation. This deliverable includes a comprehensive report on the development process, covering the refinement of initial requirements, definition of functional characteristics, and systematic development procedures. Additionally, it provides a detailed user guide with instructions on the code structure, configuration file parameters, and selective invocation of animate behaviours—body movement, hand flexing, and base rotation.

Overall, this report provides a comprehensive overview of the development and implementation of the animate behaviour subsystem. It details the functional specifications and technical design of the `animateBehaviour` ROS node.

Contents

1 Introduction **4**

2 Requirements Definition **5**

3 Module Specifications **6**

4 Implementation **9**

5 Module Design **14**

6 Executing the Animate behaviour **15**
6.1 Physical Robot Execution 15
6.2 Simulator Execution 17

7 Unit Test **19**
7.1 File Organization and Its Purposes 19
7.2 Test Cases and Types 20
7.3 Executing the Animate behaviour Test 22

References **30**

Principal Contributors **30**

Document History **31**

1 Introduction

Deliverable D5.2 focuses on the development and implementation of the `animateBehaviour` subsystem, a component designed to enhance the lifelike appearance of Pepper robot. By periodically actuating its joints in random patterns, this subsystem aims to give the robot more natural and dynamic movements. This deliverable is part of Task 5.2, which aims to provide both functional software and comprehensive documentation detailing the various stages of the software development process.

The animateBehaviour module is a ROS-based system that controls pepper's actuators. It enables the robot to perform subtle body movements, flex its hands, and slightly rotate its base along the Z-axis. This movement is achieved by actuating all joints except for headYaw and headPitch, which are managed by the Attention subsystem. By generating movements in a randomized pattern, the module creates a natural and lifelike appearance for the robot.

A key feature of the animateBehaviour module is its dynamic activation and deactivation capability via the service `/animateBehaviour/set_activation`. The module's operation is governed by a configuration file, `animateBehaviourConfiguration.ini`, which specifies crucial parameters such as the platform (robot or simulator), type of animate behaviour, range of movement, and topic name of the actuators. This configuration ensures compatibility with both the physical Pepper robot and a simulator. Additionally, the module includes a verbose mode that provides detailed diagnostic information, which is useful for debugging and monitoring the system's performance.

This report outlines the detailed specifications of the animateBehaviour module, covering interface design, module design, and execution of the animate behaviour. The subsequent sections will delve into each aspect comprehensively, ensuring that all components and functionalities are clearly explained and documented.

2 Requirements Definition

The animateBehaviour module has been designed and implemented to enhance the lifelike appearance of the Pepper robot by actuating its joints periodically in random patterns. This section details the functional requirements that have been met to achieve this objective.

Randomized Joint Movements

The primary function of the animateBehaviour module is to actuate the robot joints periodically in a random pattern. This functionality aims to simulate natural movements by keeping the joint angles close to their default home positions. The randomization ensures that the movements are not repetitive, contributing to a more realistic appearance of the robot.

Selective behaviour Invocation

The module supports the selective invocation of any of the three types of animate behaviour. If no specific behaviour is selected, all three behaviours will be invoked using `All`. The supported behaviours are:

- Body Movement: Subtle movements of the robot's body.
- Hand Flex: Flexing movements of the robot's hands.
- Base Rotation: Slight rotation of the robot's base along the Z-axis.
- All: All three behaviours will be invoked

This feature allows for targeted animations depending on the context, enhancing the flexibility and applicability of the module.

Dynamic Activation/Deactivation

To manage the operational state of the module, a service has been implemented to provide dynamic activation and deactivation.

Configuration Flexibility

The animateBehaviour module has been implemented to ensure compatibility with both the physical Pepper robot and a simulator, allowing for development, testing, and deployment across various platforms. Its operation is governed by a configuration file, `animateBehaviourConfiguration.ini`, which specifies key parameters such as platform, type of animate behaviour, range of movement, and topic names for actuators, enabling easy adaptation to different environments without altering the core codebase. A verbose mode is included to provide detailed diagnostic information, which is useful for debugging and monitoring the ROS node performance by outputting detailed logs of operations, movement commands, status updates, and errors.

3 Module Specifications

The animate behaviour module implements three core movement types to create lifelike animations: flexible hand movements, subtle body movements, and base rotation. All movements are periodic and follow a random pattern, with the random positions close to the default home position.

enerating Random Movements

To calculate the random position centered on the home position, two percentage values are used: the `selected range` and the `maximum range`. The `selected range` is the same for all actuators, while the `maximum range` varies between them.

- **Arm Actuators**: The `armMaximumRange` is a list of five values defined in the configuration. Each value corresponds to a specific joint as follows:
  - `RShoulderPitch`
  - `ShoulderRoll`
  - `ElbowRoll`
  - `ElbowYaw`
  - `WristYaw`

  For example, the default maximum range values for the arm are `0.2, 0.2, 0.2, 0.35, 0.2`, corresponding to the above joints respectively.
- **Hand Actuators**: The `handMaximumRange` is a single value defined in the configuration, and it applies to both the left and right hands. This value corresponds to the hand joint. For example, in the configuration file, the preferred value is set to `0.7`.
  - `Hand`
- **Leg Actuators**: The `legMaximumRange` is a list of three values defined in the configuration. Each value corresponds to a specific joint as follows:
  - `HipPitch`
  - `HipRoll`
  - `KneePitch`

  For example, the default maximum range values for the leg are `0.1, 0.1, 0.08`, corresponding to the above joints respectively.
- **Rotation Actuators**: The `rotMaximumRange` is a single value defined in the configuration. This value corresponds to the rotation joint. For example, in the configuration file, the default value is set to `0.3`.
  - `Wheel`

Using the two main percentage values set above, the steps to calculate the random position are described in Table 1 step by step.

Table 1: Step-by-step calculation of random position.

Step [Variable]	**Mathematical Formula**
1. Full Range [FR]	$FR = MaxPosition - MinPosition$
2. Maximum Range Offset [MRO]	$MRO = FR \times MaxRangePercentage$
3. Selected Range Offset [SRO]	$SRO = (MRO \times SelectedRangePercentage)/2$
4a. Upper Position Bound [UB]	$UB = \min(HomePosition + SRO, MaxPosition)$
4b. Lower Position Bound [LB]	$LB = \max(HomePosition - SRO, MinPosition)$
5. Random Position [RP]	$RP \in [LB, UB]$

NOTE

The algorithm generates random positions for actuators, and each position is constrained such that both the maximum range percentage and the selected range percentage must be between 0 and 1. Furthermore, the home position must be between the minimum and maximum positions ($MinPosition \leq HomePosition \leq MaxPosition$). The final random position is generated within these bounds ($LB \leq RP \leq UB$).

To achieve smooth movement patterns, instead of generating and moving to individual target positions, the system uses a list of pre-generated random positions. For all joints except the leg, a list of random positions (configurable via `numPoints`) is generated. In the configuration file, the default value is set to 100, and the actuators move through this list continuously. However, leg joints (HipPitch, HipRoll, KneePitch) require a different approach due to their movement characteristics. For leg joints, continuous movement through 100 positions would result in excessive motion without sufficient pauses. The solution implements a chunked pattern approach, where the total positions (`numPoints` = 100) are divided into chunks. The chunk size is determined by two configuration parameters: `numPoints` and `legRepeatFactor` (set to 8). The `legRepeatFactor` determines how many random positions (`numPointsLeg` set to 2) will be executed while other joints complete their 100-position sequence.

Each chunk follows a specific pattern: the first two positions are random movements, followed by home positions for the remainder of the chunk (about 10 home positions, as `chunkSize` = `numPoints` / `legRepeatFactor` = 100 / 8 = 12). This creates a movement pattern where the leg moves to two random positions, stays at the home position for a time interval, and then moves to the next set of random positions. This cycle continues through all 8 chunks, creating a more controlled and rhythmic movement pattern suitable for leg joints.

Pattern Structure:

```
Total Positions (numPoints) = 100
Leg Repeat Factor = 8
Random Positions per chunk (numPointsLeg) = 2
Chunk Size = numPoints/legRepeatFactor = 100/8 = 12

+--------------------+  +--------------------+  +--------------------+
| R R H H H H H ...|  | R R H H H H H ...|  | R R H H H H H ...| ...
+--------------------+  +--------------------+  +--------------------+
 ^ ^        ^
 | |        |
 2 Random  10 Home positions
```

NOTE

If the chunk size is not a whole number, the total leg positions might not sum up to 100. In such cases, the remaining positions are filled with the home position. For example, if there are 96 positions, the last 4 will be set to the home position. To avoid this, use leg repeater factors that divide evenly into 100.

4 Implementation

File Organization and Its Purposes

The `animateBehaviour` directory is organized to include configuration files, data, header files, source code, services, `README.md`, and `CMakeLists.txt`, as illustrated in Figure 1.

The `config` folder contains configuration settings, such as `animateBehaviourConfiguration.ini`, and defines parameters for behaviour customization. The `data` folder holds critical resources, including `pepperTopics.dat` and `simulatorTopics.dat`, which provide topic mappings for communication with the Pepper robot or its simulator. The `include/animate_behaviour` folder contains header files, such as `animateBehaviourInterface.h`, that define interfaces and declarations used throughout the module.

The `src` folder houses implementation files, such as `animateBehaviourApplication.cpp` for application logic and `animateBehaviourImplementation.cpp` for detailed functional execution. Additionally, the `README.md` file provides documentation for understanding and using the module, while the `CMakeLists.txt` file manages the build system configuration. The directory also includes services to enable and disable the node, allowing dynamic control of its activation.

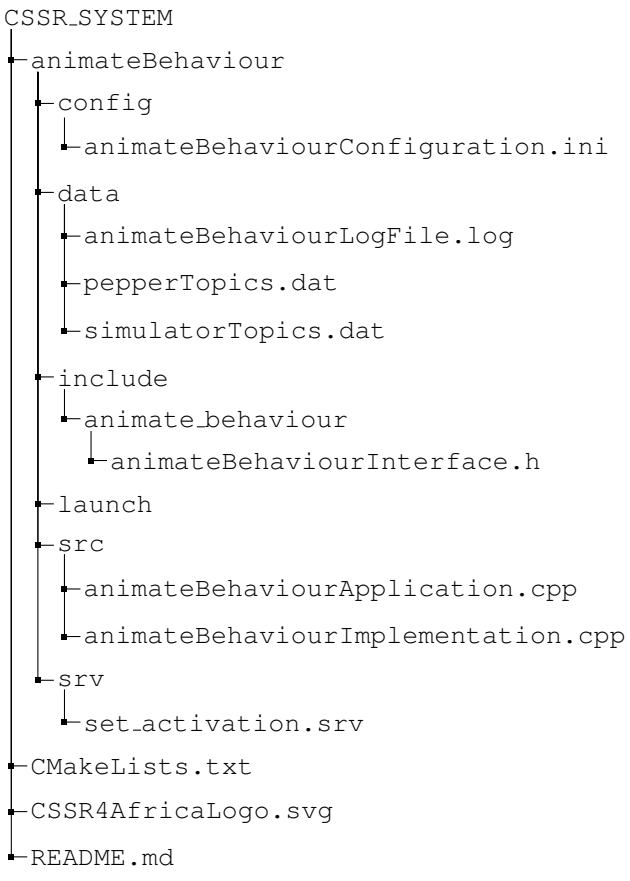

Figure 1: Directory structure for the animate behaviour ROS package

Configuration File

The operation of the `animateBehaviour` node is determined by the contents of a configuration file, `animateBehaviourConfiguration.ini`, which contains a list of key-value pairs as shown in the table below.

Table 2: Configuration parameters for the animate behaviour module.

Key	Description
`platform`	Specifies the target platform to be used, which can be set to either `simulator` or `robot`.
`robotTopics`	Specifies the name of the robot topics file. The robot topics file contains the list of topics for the robot.
`simulatorTopics`	Specifies the name of the simulator topics file. The simulator topics file contains a list of simulator topics.
`verboseMode`	If it is set to `false`, the terminal will print the configuration values, copyright notice, startup messages, configuration values, and heartbeat messages. If there is an error, the error message will also be displayed. When the verbose mode is set to `true`, additional `ROSINFO` messages used for debugging will be printed.
`rotMaximumRange`	Used in the `calculateAngularVelocityZ()` function to determine the maximum rotation range for base movements. The value `0.3` means 30% of the maximum possible angular velocity will be used, and it is multiplied by the full range to limit how fast the robot can rotate. You can increase this value up to `0.5` for more noticeable rotations, but values above this might make the robot unstable.
`selectedRange`	The `selectedRange` parameter acts as a global scaling factor for all movements, except rotation. Set at `0.5` (50%), it provides a good balance between visible movement and stability. You can adjust this value between `0.3` and `0.7`; lower values create more subtle movements, while higher values make movements more noticeable.
`armMaximumRange`	The `armMaximumRange` uses five values for different arm joints: `0.2`, `0.2`, `0.2`, `0.35`, and `0.2` for `ShoulderPitch`, `ShoulderRoll`, `LElbowRoll`, `LElbowYaw`, and `LWristYaw`, respectively. The slightly higher value (`0.35`) for the elbow yaw allows more natural elbow movement. Keep the shoulder values (the first two numbers) lower for stability.
`handMaximumRange`	The `handMaximumRange` set at `0.7` allows significant joint `hand` opening and closing. You can adjust this value between `0.5` and `0.8`, depending on how expressive you want the hand gestures to be.
`legMaximumRange`	The `legMaximumRange` values `(0.1, 0.1, 0.08)` for `HipPitch`, `HipRoll`, and `KneePitch`, respectively, are intentionally conservative to maintain balance. The hip joints (the first two values) are set at 10% range, while the knee is slightly more restricted at 8%. It is recommended not to exceed `0.15` for any leg joint to prevent stability issues.

Table 2: Configuration parameters for the animate behaviour module.

Key	Description
`gestureDuration`	The `gestureDuration` parameter (currently set to `1.0` second) is a timing control that determines how long the robot takes to move between each randomly generated position. When the system calculates random positions for arms, hands, or legs, these positions become waypoints in a movement trajectory, and the `gestureDuration` specifies the time allocated to travel between each waypoint. Setting a lower duration value (like `0.5`) creates faster but potentially more abrupt movements, while higher values (like `2.0`) result in slower motions. The preferred value is `1.0`, which provides balanced timing to create subtle body movement.
`numPoints`	The `numPoints` parameter (preferred value set to `100`) determines the total number of random positions generated for arm and hand movements. The purpose is to create a set of random positions, which are passed as a single trajectory. This approach helps reduce jerks, as it smooths the transitions between each set of `100` joint positions.
`numPointsLeg`	The `numPointsLeg` parameter (preferred value set to `2`) determines the number of random positions generated for leg movements within each repetition cycle. The purpose is to create a small set of random positions, which are then repeated based on the `legRepeatFactor` value. This approach helps maintain stability by using minimal leg movements while still creating subtle motions through controlled repetition.
`legRepeatFactor`	The `legRepeatFactor` parameter (preferred value set to `8`) determines how many times the leg movements generated by `numPointsLeg` are repeated. When `legRepeatFactor` is set to `8`, the system takes the `2` generated positions and repeats them `8` times, creating a total of `16` movement points that run parallel to the `100` points generated for arm and hand movements. This synchronized approach ensures coordinated full-body animation while maintaining stable leg motions.

Input File

This node does not read from an input data file.

Output Data File

This node writes a log file to `animateBehaviourLogFile.log` as output, which will be used as input information for unit test cases.

Topics Subscribed

This node does not subscribe to any topics.

Topics Published

This node publishes actuator topics listed in Table 3, which are specified in the configuration file using the key-value pairs provided in `pepperTopics.dat` and `simulatorTopics.dat`. These `.dat` files are stored in the `data` section of the `animateBehaviour` node.

When `pepperTopics.dat` is set in the configuration file, the node uses the topics defined in `pepperTopics.dat`, which are published in the physical actuators of the robot. Similarly, when `simulatorTopics.dat` is set in the configuration file, the node publishes to the topics specified in `simulatorTopics.dat`, enabling communication with the simulator.

Table 3: Topics, Actuators, and Platforms

Topic	Actuator	Platform
`/pepper_dcm/RightHand_controller/follow_joint_trajectory`	RHand	Physical robot
`/pepper_dcm/LeftHand_controller/follow_joint_trajectory`	LHand	Physical robot
`/pepper_dcm/RightArm_controller/follow_joint_trajectory`	RShoulderPitch, RShoulderRoll, RElbowYaw, RElbowRoll, RWristYaw	Physical robot
`/pepper_dcm/LeftArm_controller/follow_joint_trajectory`	LShoulderPitch, LShoulderRoll, LElbowYaw, LElbowRoll, LWristYaw	Physical robot
`/pepper_dcm/Pelvis_controller/follow_joint_trajectory`	HipRoll, HipPitch, KneePitch	Physical robot
`/pepper_dcm/cmd_moveto`	Wheels	Physical robot
`/pepper/RightArm_controller/follow_joint_trajectory`	RElbowYaw, RElbowRoll	Simulator
`/pepper/LeftArm_controller/follow_joint_trajectory`	LElbowYaw, LElbowRoll	Simulator
`/pepper/Pelvis_controller/follow_joint_trajectory`	HipRoll, HipPitch, KneePitch	Simulator
`/pepper/cmd_vel`	Wheels	Simulator

Services Supported

This node provides and advertises a server for a service `/animateBehaviour/set_activation` to enable or disable the operation of the node, i.e., to activate or suspend the publishing of data on the actuator topics to give the appearance of an animated agent. The service defines a request field `string state` which can have a value of either “enabled” or “disabled”. Depending on the state value, the node will be enabled or disabled. The service returns a response value of “1” for success or “0” for failure. The service is called by the `behaviourController` node to enable or disable animated behavior as needed. The following summarizes the services supported.

Table 4: Summary of Supported Services

Service	Message Value	Effect
`/animateBehaviour/set_activation`	enabled, disabled	Enable or disable animate behaviour

Services Called

This node does not call any services.

5 Module Design

The animate behaviour implementation utilizes several ROS message types to enable coordinated robot motion control. These messages facilitate joint trajectory execution, velocity commands, sensor feedback, and behaviour activation through action-based communication and service calls. Each message type serves a specific purpose in the motion control architecture, from low-level joint commands to high-level behaviour coordination.

Joint Control Messages
`trajectory_msgs::JointTrajectoryPoint`: It is used to specify precise positions and timing for each point in the robot's movement trajectory. It is particularly used in the `moveToListOfPositions` function, where the robot needs to move through a sequence of positions in a smooth, natural-looking way.

`trajectory_msgs::JointTrajectory`: The message is used in two key functions within the provided code to define the desired movement path for the robot's joints. In the `moveToPosition` function, it is used to send a single, fixed target position for the robot's joints to an action server. In the `moveToPositionBiological` function, it is used to create a more complex movement trajectory by generating a sequence of waypoints with their own joint positions and durations. This structured representation of the joint movements is essential for achieving the natural-looking and fluid animations that the robot is designed to perform.

`control_msgs::FollowJointTrajectoryGoal`: The message is a ROS message that is used to define the parameters of joint trajectory actions. It is used in the functions `moveToPosition()` and `moveToPositionBiological()` to control the movements of the robot's arm, hand, and legs by sending trajectory goals to the action servers.

Base Motion Control Messages
`geometry_msgs::Twist`: The message type is a ROS message used to represent velocity commands with both linear and angular components, and it is used in the function `rotationBaseShift()` to control the robot's base rotation. Specifically, the angular velocity values along the z-axis are assigned to `twist.angular.z` and published using `velPub.publish(twist)`, allowing the robot to execute controlled rotational movements.

Action and Service Control Messages
`actionlib::SimpleClientGoalState`: It is a state tracking message that monitors the execution status of action goals through states such as PENDING, ACTIVE, and SUCCEEDED. Provides essential feedback for coordinating robot movements and implementing error handling.

`cssr_system::set_activation`: It is a custom service message type used in the function `setActivation()` to enable or disable the animate behaviour system in the CSSR system. Implements a simple request-response pattern, where requests specify the desired state ("enabled" or "disabled"), and responses indicate success or failure. This message type ensures safe and controlled activation or deactivation of robot behaviours, providing effective control over the animation state.

6 Executing the Animate behaviour

To activate the node, it is essential to first understand the key aspects of the animate behaviour. There are three behaviours in this node: hands and rotation. These values are configured in the settings, as detailed in Table 2. To further understand the random positions calculated for each joint, refer to the mathematical details explained in Section 3. This node provides and advertises a server for the service `/animateBehaviour/set_activation` to enable or disable the operation of the node. The service accepts a value of `"enabled"` or `"disabled"`. Depending on the string value provided, `animateBehaviour` will be activated or deactivated. For an explanation of additional configuration values and their functionality, refer to Section 4. This section outlines the detailed steps required to configure, launch and test the `Animate behaviour` node for both physical robots and simulated environments.

6.1 Physical Robot Execution

To execute the `Animate behaviour` node on a physical robot, follow these steps:

1. Environment Setup

Install all necessary dependencies as per the CSSR4Africa Software Installation Manual. Clone the repository into the robot's workspace: Move to the source directory of the workspace:

```
cd $HOME/workspace/pepper_rob_ws/src
```

Clone the CSSR4Africa software from the GitHub repository:

```
git clone https://github.com/cssr4africa/cssr4africa.git
```

Build the source files:

```
cd .. && catkin_make
```

Source the environment:

```
source devel/setup.bash
```

2. Configure the Node

The configuration file `/animateBehaviourConfiguration.ini` and preferred values are set as shown in the figure below. If you want the desired behaviour (`body`, `hands`, `rotation`, or `All`), please set the behaviour you want to run. For an explanation of each configuration value, refer to Section 4. Otherwise, the preferred values are the ones already set in the configuration file, as shown in Table 5.

Table 5: Configuration Parameters for Animate behaviour Node

Parameter	Value	Type
platform	robot	string
behaviour	all	string
simulatorTopics	simulatorTopics.dat	file
robotTopics	pepperTopics.dat	file
verboseMode	false	boolean
rotMaximumRange	0.3	float
selectedRange	0.5	float
armMaximumRange	0.2,0.2,0.2,0.35,0.2	float array
handMaximumRange	0.7	float
legMaximumRange	0.1,0.1,0.08	float array
gestureDuration	1.0	float
numPoints	100	integer
numPointsLeg	2	integer
legRepeatFactor	8	integer

3. Launch the Node

Move to the workspace directory:

```
cd $HOME/workspace/pepper_rob_ws
```

Launch the robot:

```
cd .. && roslaunch cssr_system LaunchRobot.launch \
robot_ip:=172.29.111.240 network_interface:=wlp0s20f3
```

NOTE

Ensure that the IP address `172.29.111.240` and the network interface `wlp0s20f3` are correctly set based on your robot's configuration and your computer's network interface.

Open a new terminal and run the `Animate behaviour` node:

```
cd .. && rosrun cssr_system animateBehaviour
```

To enable animate behaviour, open a new terminal and run the code below.

```
cd .. && rosservice call /animateBehaviour/set_activation "state: '
    enabled'"
```

Disable the animate behaviour service:

```
rosservice call /animateBehaviour/set_activation "state: 'disabled'"
```

6.2 Simulator Execution

To execute the `Animate behaviour` node on a Simulator follow these steps:

1. Environment Setup

Install all necessary dependencies as per the CSSR4Africa Software Installation Manual. Clone the repository into the robot's workspace: Move to the source directory of the workspace:

```
cd $HOME/workspace/pepper_sim_ws/src
```

Clone the CSSR4Africa software from the GitHub repository:

```
git clone https://github.com/cssr4africa/cssr4africa.git
```

Build the source files:

```
cd .. && catkin_make
```

Source the environment:

```
source devel/setup.bash
```

2. Configure the Node

Update the configuration file located at `/workspace/pepper_sim_ws/src/cssr4africa /animateBehaviour/config /animateBehaviourConfiguration.ini` and set the platform to `simulator` and select the desired behaviour (`body`, `hands`, `rotation`, or `All`). If you want to modify other configuration values, refer to Section 4. Otherwise, the preferred values are the ones already set in the configuration file.

3. Launch the Node

Move to the workspace directory:

```
cd $HOME/workspace/pepper_sim_ws
```

Launch the simulator:

```
cd .. && roslaunch cssr_system LaunchSimulator.launch
```

Open a new terminal and run the `Animate behaviour` node:

```
cd .. && rosrun cssr_system animateBehaviour
```

To enable animate behaviour, open a new terminal and run the code below.

```
cd .. && rosservice call /animateBehaviour/set_activation "state: '
    enabled'"
```

Disable the animate behaviour service:

```
rosservice call /animateBehaviour/set_activation "state: 'disabled'"
```

7 Unit Test

The unit test framework for Animate behaviour is designed to test whether the node operates as expected. It examines four cases: hand, body, rotational, and all movements combined. The framework validates the correct setup, execution, and logging of these actions in physical and simulated robot environments. Detailed logs and performance reports are generated using ROS and Google Test to measure the reliability of the Animate behaviour.

7.1 File Organization and Its Purposes

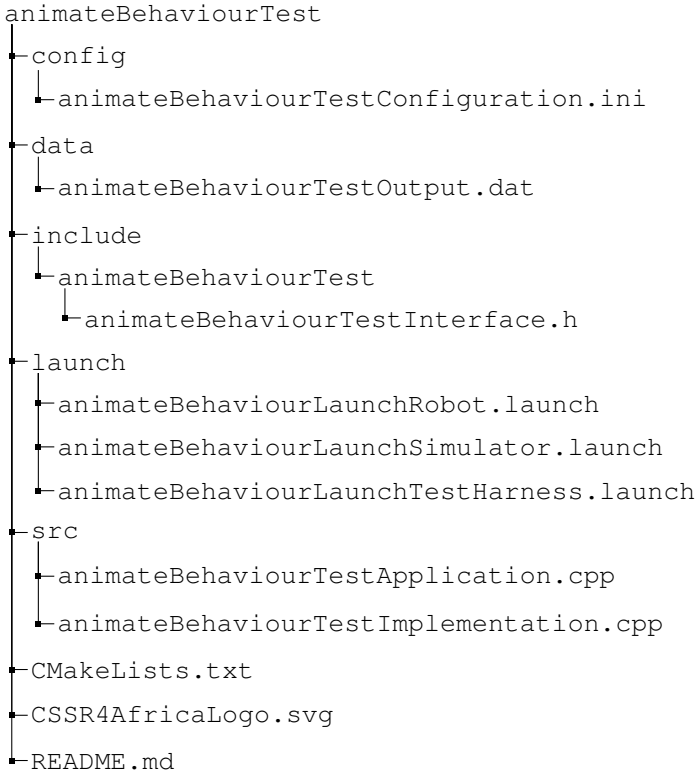

Figure 2: Directory structure of the Animate behaviour Test.

The directory structure of the animated behaviour test framework is organized as follows. The `src` folder contains `animateBehaviourTestApplication.cpp`, which manages the main application logic, and `animateBehaviourTestImplementation.cpp`, which provides detailed test implementations.

The `config` folder includes `animateBehaviourTestConfiguration.ini`, a file used to enable or disable specific tests, such as hand, body, rotation or all movements. The test results are stored in the `data` folder within the `animateBehaviourTestOutput.dat` file. The `include` directory contains the header file `animateBehaviourTestInterface.h`, which defines interfaces and utility functions.

The `launch` folder includes three ROS launch files: `animateBehaviourLaunchRobot.launch`,

which launches the robot; `animateBehaviourLaunchSimulator.launch`, which launches the simulator; and `animateBehaviourLaunchTestHarness.launch`, which launches both the `animateBehaviour` and `animateBehaviourTest` nodes. Furthermore, the `README.md` file provides documentation for understanding and using the test framework, while `CMakeLists.txt` configures the build system.

Configuration File

The operation of the `animateBehaviourTest` node is determined by the contents of a configuration file, `animateBehaviourTestConfiguration.ini`, that contains a list of key-value pairs as shown in Table 5.

7.2 Test Cases and Types

The unit tests in the animate behaviour system are implemented using Google Test (gtest), a comprehensive C++ testing framework developed by Google that provides essential testing capabilities including test fixtures, assertions, test case organization, and result reporting. The testing suite implements four primary test cases:

- Hand movement testing (Test01FlexiHandanimateBehaviour)
- Body movement testing (Test02SubtleBodyanimateBehaviour)
- Rotation testing (Test03RotationanimateBehaviour)
- Combined movement testing (Test04AllanimateBehaviour)

Each test case inherits from a base test fixture class, animateBehaviourRobotTest, which manages test setup and teardown operations, ensuring consistent test environments and proper resource management throughout the testing process.

The testing implementation utilizes Google Test's assertion system and test execution control to validate movement execution, position accuracy, and system stability. The test framework for the Animate behaviour system incorporates several key features to ensure comprehensive and reliable testing. Test configuration is managed through dedicated configuration files, allowing users to selectively activate or deactivate specific tests based on requirements. Automated reporting is an integral component of the framework, generating detailed test reports that document results and capture movement data for thorough analysis. The framework also supports continuous testing, enabling repeated test execution with user-controlled iterations to ensure consistency and robustness across multiple runs.

Class Implementation

animateBehaviourRobotTest: It is a test class that inherits from `testing::Test` for unit testing of robot animate behaviour. The class is structured with two static members: `testReport` (an `ofstream` pointer for test output management) and `alert` (a boolean flag). It includes two protected methods: `SetUp()` for disabling animate behaviour before each test and `TearDown()` for cleanup and report flushing after each test. The public interface provides two static methods: `setTestReportStream()` and `getTestReportStream()` for managing test reporting. The test cases are implemented using the `TEST_F` macro from the Google Test framework.

Configuration Functions: These are standalone helper functions that manage all configuration-related operations and it include `readbehaviourConfig()` which reads test settings from configuration files and returns them as a map, `writeConfigurationFile()` which handles writing robot configuration parameters, `writeInitialConfigurationFile()` which sets up initial behaviour configuration values, and `setConfigurationAndWriteFile()` which updates specific behaviour settings in the configuration file. These functions work together to ensure proper test configuration management and state setup before test execution.

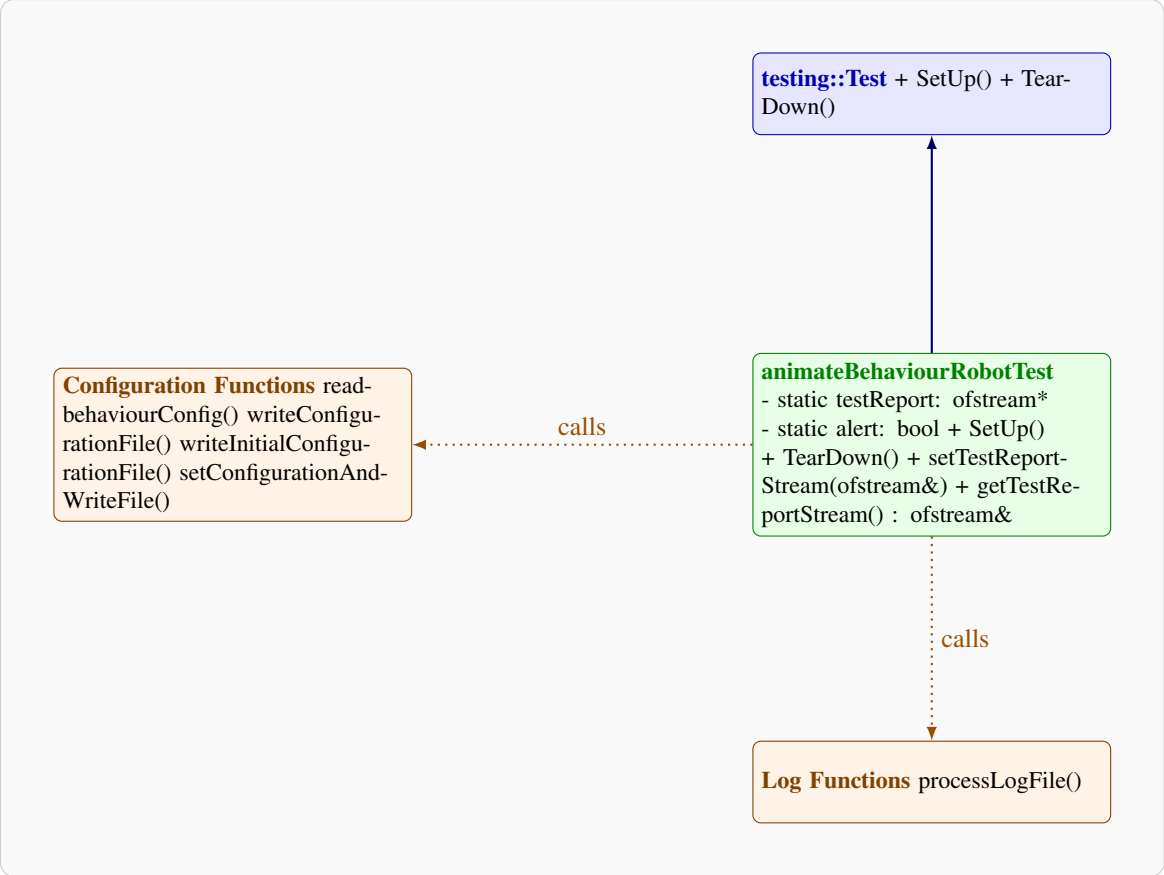

Figure 3: Class Diagram of animateBehaviourRobotTest System

Log Functions: The `processLogFile()` function serves as a standalone utility for analyzing and validating test execution results. It reads log files generated during test execution, captures behaviour state changes, movement events, configuration settings, and various position data. The function processes this information by parsing different message types, tracking movement states, and organizing data into appropriate structures for test validation. It maintains maps for configuration content, joint names, home positions, and random positions while ensuring proper file handling and cleanup operations.

Testing::Test: This is the base class provided by the Google Test framework that defines the fundamental structure for test fixtures. It provides virtual `SetUp()` and `TearDown()` methods that are overridden by test classes to implement proper test initialization and cleanup procedures. The class

serves as the foundation for creating organized and consistent test cases, ensuring proper test lifecycle management and resource handling throughout the testing process.

7.3 Executing the Animate behaviour Test

Test Environment Setup

Before executing the animate behaviour tests, the testing environment must be properly configured. Install all necessary dependencies as per the CSSR4Africa Software Installation Manual. Clone the repository into the robot's workspace.

Move to the source directory of the workspace:

```
cd $HOME/workspace/pepper_rob_ws/src
```

Clone the CSSR4Africa software from the GitHub repository:

```
git clone https://github.com/cssr4africa/cssr4africa.git
```

Build the source files:

```
cd .. && catkin_make
```

Source the environment:

```
source devel/setup.bash
```

This build process compiles all test components, dependencies, and ensures proper integration with the ROS framework.

Configure the Node

The test execution begins with the configuration file setup in `animateBehaviourTestConfiguration.ini`. This file controls test execution, enabling or disabling specific behaviours. As shown in Table 2, the `platform` parameter is set to `robot` for physical robot testing. To test all the behaviours, change the behaviour values to one of (`hands`, `body`, `rotation`, or `all`). Update the configuration file with your preferred settings as needed. The content is the same as the `animateBehaviour` configuration, and the default value is set as specified in Table 2.

Launch the Node

Move to the workspace directory:

```
cd $HOME/workspace/pepper_rob_ws
```

Launch the robot:

```
cd .. && roslaunch unit_tests animateBehaviourLaunchRobot.launch\
robot_ip:=172.29.111.240 network_interface:=wlp0s20f3
```

NOTE

Ensure that the IP address `172.29.111.240` and the network interface `wlp0s20f3` are correctly set based on your robot's configuration and your computer's network interface.

Open a new terminal and run the `animateBehaviour` and `animateBehaviourTest` nodes by launching the `animateBehaviourLaunchTestHarness`.
Move to the workspace directory

```
cd $HOME/workspace/pepper_rob_ws
```

Launch the nodes:

```
cd .. && roslaunch unit_tests animateBehaviourLaunchTestHarness.launch
```

Upon launch, the test will be executed based on the configuration values set for each behaviour. After the completion of each test cycle, the system provides an interactive prompt, allowing the tester to either continue with another iteration or conclude the testing session. If you want continuous testing, please press `y` as shown in the Figure4.

```
[ INFO] [1734104535.211006577]: Tests completed. Run tests again? (y/n):
```

Figure 4: Console output showing the completion of tests and prompting for rerun.

The test reports are attached in Appendix I for detailed analysis and to verify that the generated random positions are centered around the home position. Additionally, all the joints used for each behaviour are listed.

Appendix I

```
======================================================================
=== New Test Run Started at 2024-12-21 16:52:22 ===
======================================================================

======================================================================
Animate behaviour enabled: PASSED
......................................................................
Test 1: Test Flexi Hand Animate behaviour

        Configuration Settings:
        armMaximumRange     : 0.2,0.2,0.2,0.35,0.2
        behaviour           : hands
        gestureDuration     : 1.0
        handMaximumRange    : 0.7
        legMaximumRange     : 0.1,0.1,0.08
        legRepeatFactor     : 8
        numPoints           : 100
        numPointsLeg        : 2
        platform           : robot
        robotTopics         : pepperTopics.dat
        rotMaximumRange     : 0.3
        selectedRange       : 0.5
        simulatorTopics     : simulatorTopics.dat
        verboseMode         : false

        Flexi movement started: PASSED

        Joint names:
            right hand: ["RHand"]

        Ensure the joint moves to the Home position before starting
        random movements.The values of the home positions are:

            right hand: [0.66608]

        After the joint is in the home position, start moving to random
        positions continuously.
        The random positions captured are:
            right hand:
            [0.738231]
            [0.572753]
            [0.684124]
            [0.764743]
            [0.546761]
            [0.838117]
```

```
            [0.838036]
            [0.838036]
            [0.724178]
            [0.830661]
            [0.740903]
            [0.59067]
            [0.703302]
            [0.755561]
            [0.768665]
            [0.827754]
            [0.66264]
            [0.542227]
            [0.74655]
            [0.652943]

        Flexi movement ended: PASSED
.....................................................................

Animate behaviour disabled: PASSED
=====================================================================


=====================================================================
=== New Test Run Started at 2024-12-21 16:47:29 ===
=====================================================================

=====================================================================
Animate behaviour enabled: PASSED
.....................................................................
Test 2: Test Subtle Body Animate behaviour
        Configuration Settings:
        armMaximumRange     : 0.2,0.2,0.2,0.35,0.2
        behaviour           : body
        gestureDuration     : 1.0
        handMaximumRange    : 0.7
        legMaximumRange     : 0.1,0.1,0.08
        legRepeatFactor     : 8
        numPoints           : 100
        numPointsLeg        : 2
        platform           : robot
        robotTopics         : pepperTopics.dat
        rotMaximumRange     : 0.3
        selectedRange       : 0.5
        simulatorTopics     : simulatorTopics.dat
        verboseMode         : false

        Subtle body movement started: PASSED
```

```
The joints used for the subtle body movements are:
    : ["Wheels"]
    left arm: ["LShoulderPitch", "LShoulderRoll", "LElbowRoll",
    "LElbowYaw", "LWristYaw"]
    leg: ["HipPitch", "HipRoll", "KneePitch"]
    right hand: ["RHand"]

Ensure the joint moves to the Home position before starting
random movements.
The values of the home positions are:
    : 2.000000
    left arm: [1.7625, 0.0997, -0.1334, -1.715, 0.06592]
    leg: [-0.0107, -0.00766, 0.03221]
    right hand: [0.66608] [0.66608]

After the joint is in the home position, start moving to random
positions continuously.
The random positions captured are:
    :
    ----------[STARTLEFTHANDANIMATEMOVEMENT]-----------
    JointNames:["LHand"]
    HomePosition:[0.6695]
    [0.726706]
    [0.738164]
    [0.721417]
    [0.801147]
    [0.504824]
    [0.717717]
    [0.577764]
    [0.667403]
    [0.759416]
    [0.556816]
    [0.818487]
    [0.54067]
    [0.542645]
    [0.768598]
    [0.70438]
    [0.535285]
    [0.56874]

    left arm:
    [1.72501,0.162002,-0.312676,-1.66499,0.124467]
    [1.71977,0.173004,-0.273621,-1.92408,0.21852]
    [1.67495,0.167279,-0.340266,-2.03727,0.180508]
    [1.41793,0.0537359,-0.389244,-1.58616,0.102452]
    [1.6395,0.140949,-0.369843,-1.37216,0.110285]
```

```
[1.52581,0.136457,-0.347886,-1.92289,0.128669]
[1.55881,0.142197,-0.386204,-1.49225,-0.0435256]
[1.45031,0.105648,-0.29062,-1.42718,0.183697]
[1.36519,0.163928,-0.36889,-1.79356,0.138906]
[1.60181,0.099757,-0.393341,-1.71675,-0.0675878]
[1.5167,0.126349,-0.330237,-1.4823,0.180838]
[1.38312,0.140332,-0.331121,-1.81992,0.185301]
[1.47407,0.0871244,-0.397883,-1.3614,-0.0322298]
[1.52184,0.0283132,-0.397756,-1.35755,0.188941]
[1.41735,0.02554,-0.350157,-1.525,-0.020932]
[1.68609,0.11631,-0.402841,-2.0048,-0.104303]
[1.65571,0.122178,-0.420076,-1.40839,-0.0969567]
[1.50185,0.175753,-0.289335,-1.69648,0.0703881]
[1.54095,0.0830216,-0.328928,-1.50464,-0.0966312]
[1.44999,0.14011,-0.296466,-1.51586,-0.107222]

leg:
[-0.0630381,-0.01341,0.0409215]
[-0.0669748,-0.00153459,0.0330461]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.0749397,0.0122032,0.0208622]
[-0.0788864,-0.0225882,0.0205542]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]
[-0.1107,-0.00766,0.03221]

right hand:
[0.79493]
[0.497658]
[0.499359]
[0.726291]
[0.661207]
[0.769722]
[0.777139]
[0.611445]
```

```
            [0.550868]
            [0.596061]
            [0.595115]
            [0.827976]
            [0.801416]
            [0.799962]
            [0.558582]
            [0.774794]
            [0.590951]
            [0.60069]
            [0.728943]
            [0.619131]
            [0.555609]
            [0.623146]
            [0.565086]
            [0.589206]
            [0.620317]
            [0.571396]
            [0.526705]
            [0.658601]
            [0.794552]
            [0.550443]
            [0.692201]
            [0.778939]
            [0.638745]
            [0.781342]
            [0.728562]
            [0.667712]
            [0.517182]
            [0.693109]
            [0.775779]
            [0.508294]

        Subtle body movement ended: PASSED
.....................................................................
Animate behaviour disabled: PASSED
=====================================================================

=====================================================================
Test Run Completed with Result: PASSED
=====================================================================



=====================================================================
=== New Test Run Started at 2024-12-21 16:44:31 ===
=====================================================================
```

```
================================================================================
Test 4: Test All Animate behaviour
--------------------------------------------------------------------------------
Initialization:
  Animate behaviour enabled: PASSED

Individual Movement Status:
  Flexi Hand Movement:
    Started: PASSED
    Ended:   PASSED

  Subtle Body Movement:
    Started: PASSED
    Ended:   PASSED

  Rotation Movement:
    Started: PASSED
    Ended:   PASSED

Overall Status:
  All movements started: PASSED
  All movements ended:   PASSED
  behaviour disabled:    PASSED
--------------------------------------------------------------------------------
Final Result: PASSED
================================================================================


================================================================================
Test Run Completed with Result: PASSED
================================================================================
```

Principal Contributors

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

Eyerusalem Birhan, CMU-Africa
Yohannes Haile, CMU-Africa
David Vernon, CMU-Africa

Document History

Version 1.0

First draft.
Eyerusalem Birhan.
21 December 2024.