Experimental-Lab-Assignments-Robotics

Generated by Doxygen 1.8.13

Contents

Chapter 1

Behavioral Architecture

The scenario describes a behavior simulation of a pet robot that interacts with humans and moves in a discrete 2D environment. The human can interact with the robot through speech comands and pointing gestures while the robot can have three beaviors: sleep, play, normal. At the beginning, the robot is in a sleep state. Everytime the robot is in the sleep state, it reaches the home position and after some time switches to normal state. Everytime the robot is in the normal state, it moves randomly and reaches three different positions, then it listens to user's comands. If user indicates a position to reach, it takes some time to reach the position and change the state into 'sleep' or 'play' with a random choice. If user tells him to go to play or to sleep, it respectively changes state into 'play' or 'sleep'. When the state is 'play' the robot reaches person's position, after that it waits for a pointing gesture, if it receives the target position, it reaches this point. After some time, the robot switches to the 'normal' state.

ROS Architecture of the System

The system is made by three ros nodes: "sim_perception.py", "pet_state_machine.py" and "display_position.py". The rqt_graph is shown below.

Rqt_graph

<img src=https://github.com/au1698/Experimental-Lab-Assignments-Robotics/blob/main/pet↩
_robot/Images/rqt_graph_pet_robot.png raw=true">

sim_perception

This node until is active simulates user's random choice between a pointing gesture (2D coordinates generation) and the vocal comands: 'go_to_home' and 'play'. It prints on the screen user's choice. Pointed gesture is of type 'Int64MultiArray' while vocal comands are simply of the type 'string'. It checks if "user_comand" is a string or an array. In the first case "user_comand" data are published on the topic /pointed_comand in the second data are published onthe topic /vocal_comand.

pet_state_machine

This node is a finite state machine composed of three states: PLAY, SLEEP, NORMAL.

<img src=https://github.com/au1698/Experimental-Lab-Assignments-Robotics/blob/main/pet_robot/Images/pet_state_machine.jpeg raw=true">

SLEEP: the state publishes on the topic '/target_point' the home position that the robot should reaches. If the robot is in the 'SLEEP' state and receives the comand 'go_to_home', it notifiess that is already at home.

NORMAL: the state publishes on the topic '/target_point' three random target positions, after that it subscribes to the topic '/vocal_comand' and '/pointed_comand'. If it receives a pointed comand, it publishes it on the topic '/target_position' and then it changes the state choosing randomly from 'go_to_sleep' or 'go_to_play'. Otherwise it checks if the vocal comand is 'go_to_home' switches into the 'SLEEP' state if it is 'play' switches into the 'PLAY' state.

PLAY: the state publishes on the topic '/target_point' a random position which represents person's position. The state takes time to reach person's position and wait for a pointing gesture. It subscribes to the topic '/vocal_comand', if it receives user's vocal comands it return and error because in this wait loop expects a pointing gesture. If it doesn't receive a vocal comand it subscribes to the topic '/pointed_comand' , it publishes on the topic '/target_point' and exit from the loop.

display_position

This node subscribes to the topic '/target_point' and display when the robot arrives to the target.

How to run the code

The first thing to do, after having cloned the repository in the Ros workspace, is to build the package in your workspace with "‘ catkin_make "‘ and give running permissions to it with "‘ $ chmod +x "‘

To run the system:

```
```
roslaunch pet_robot pet_robot.launch
```
```
```

To visualize the Smach Viewer:

```
```
```

rosrun smach_viewer smach_viewer.py

"‘

Working hypoteses

The gesture commands present the same "priority" since they occur in random order. The home position is fixed (0,0). Person's position is generated randomly in 'PLAY' state. There is not a simulator. When the robot is in the 'SLEEP' state, it only reaches the home position and after some time goes to the 'NORMAL' state, even if receives as user's comand: 'play' or a pointed gesture.

Possible improvements

Use the ROS parameter service to define a parameter to scale the simulation velocity. Using a service-client as kind of comunication between the simulation node and the state machine in order to improve the synchronization.

Author:

- Aurora Bertino: bertino.aurora16@gmail.com

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

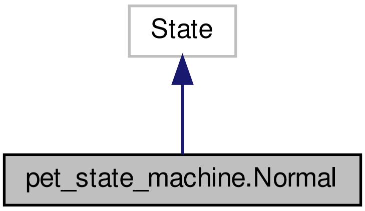
Chapter 5

Class Documentation

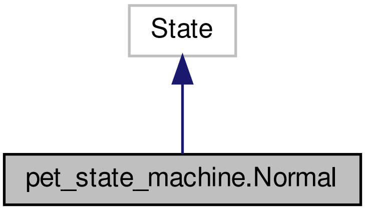
5.1 pet_state_machine.Normal Class Reference

Define state Normal.

Inheritance diagram for pet_state_machine.Normal:



Collaboration diagram for pet_state_machine.Normal:



Public Member Functions

- def __init__ (self)

  Constructor of the class Normal.
- def **execute** (self, userdata)

Static Public Attributes

- outcomes

  Initialization function.
- **input_keys**
- **output_keys**

5.1.1 Detailed Description

Define state Normal.

Definition at line 88 of file pet_state_machine.py.

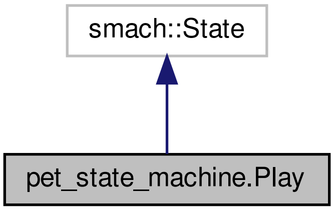
The documentation for this class was generated from the following file:

- src/pet_state_machine.py

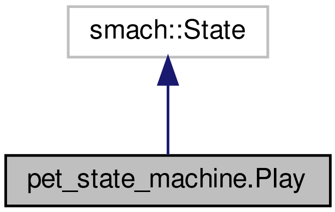
5.2 pet_state_machine.Play Class Reference

Define state Play.

Inheritance diagram for pet_state_machine.Play:



Collaboration diagram for pet_state_machine.Play:



Public Member Functions

- def __init__ (self)

    Constructor of the class Play.
- def **execute** (self, userdata)

Static Public Attributes

- outcomes

    Initialization function.
- **input_keys**
- **output_keys**

5.2.1 Detailed Description

Define state Play.

Definition at line 127 of file pet_state_machine.py.

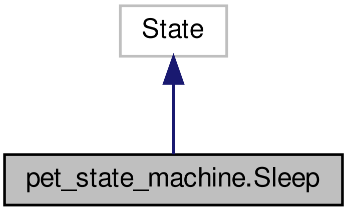
The documentation for this class was generated from the following file:

- src/pet_state_machine.py

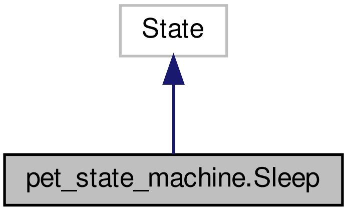
5.3 pet_state_machine.Sleep Class Reference

Define state Sleep.

Inheritance diagram for pet_state_machine.Sleep:



Collaboration diagram for pet_state_machine.Sleep:



Public Member Functions

- def __init__ (self)

  Constructor of the class Sleep.
- def **execute** (self, userdata)

Static Public Attributes

- outcomes

  Initialization function.
- **input_keys**
- **output_keys**

5.3.1 Detailed Description

Define state Sleep.

Definition at line 52 of file pet_state_machine.py.

The documentation for this class was generated from the following file:

- src/pet_state_machine.py

Chapter 6

File Documentation

6.1 src/display_position.py File Reference

This node displays when the target position is reached.

Functions

- def display_position.callback (data)

  Define callback function: callback()
- def **display_position.position_subscribe** ()

6.1.1 Detailed Description

This node displays when the target position is reached.

Details: It subscribe to the topic '/target_position' and publishes when the robot arrives to the target.

6.2 src/pet_state_machine.py File Reference

This node implements a state machine.

Classes

- class pet_state_machine.Sleep

  Define state Sleep.
- class pet_state_machine.Normal

  Define state Normal.
- class pet_state_machine.Play

  Define state Play.

Functions

- def pet_state_machine.callback_vocal_comand (data)

  Define callback function: vocal_comand()
- def pet_state_machine.callback_pointed_comand (data)

  Define callback function: pointed_comand()
- def **pet_state_machine.main** ()

Variables

- **pet_state_machine.array_point** = np.array([0,0])
- **pet_state_machine.target_pub** = rospy.Publisher('/target_point', Int64MultiArray, queue_size=10)
- string **pet_state_machine.vocal_data** = ""

6.2.1 Detailed Description

This node implements a state machine.

Details: It receives commands 'sim_perception' node, implements the state machine and sends the target positions to 'display_position' node.

6.3 src/sim_perception.py File Reference

This node simulates user's comands.

Functions

- def **sim_perception.Simulator** ()

6.3.1 Detailed Description

This node simulates user's comands.

This node simulates user's comands generation.

Details: It generates a vector (pointed gesture)in which there are coordinates x,y and randomly choose between vocal comands 'play', 'go_to_home' and the vector.

Index