Second Assignment for Experimental Robotics Laboratory

Generated by Doxygen 1.8.17

Chapter 1

Experimental Robotics Laboratory Second Assignment

The second assignment builds on top of the first one: we students are now asked to implement a similar robot behaviour on a gazebo-simulated pet robot. The robot is akin to a dog, it's a wheeled robot with a neck and a camera-equipped head that can rotate: it can move randomly, track a ball, move its head and go to sleep.

1.1 System Architecture

1.1.1 Component Diagram

The software architecture is based on **four main components**:

- **Person module**

  This module mimics the behaviour of a person that can either move the ball to a random valid location or make it disappear under the plane.

  It implements an action client which randomly sends one of the two types of goals to the ∗"go_to_point_↩ ball.py"∗ action server: the module then waits for the ball to reach its destination and sleeps for some time before sending another goal.

- **Finite state machine**

  This component implements a finite state machine using "Smach". It features an action client, two publishers and one subscriber.

  The three states, together with the transitions between them, will be further explained in the following paragraph.

- **Robot action server**

  This component implements an action server to control the robot. Since it was provided by the professor and I only modified some gains and some print functions, I'm not going to further explain how it works in detail.

- **Ball action server**

  This component implements an action server to control the ball. As for the robot action server, I'm not going over it in detail.

1.1.2 State Diagram

This is the state diagram that shows how the finite state machine works:
When the robot is in **Normal** state it moves randomly in the plane by sending a goal to the robot action server: if at anytime the robot sees the ball, then it switches to the *Play* state. If instead the robot has performed enough actions (an action is either reaching a location or performing the sequence of camera movements as described in the *Play* state) then it switches to the *Sleep* state.

When the robot is in **Sleep** state, it first reaches the predefined "Home" location (-5, 7), then stays there for some time and finally wakes up, transitioning back to the *Normal* state.
When the robot is in **Play** state, it tracks the ball until it gets close enough: when this happens, the robot stays still and rotates its head 45 degrees on the left, then does the same for the right and finally goes back to the center. This behaviour goes on until the ball can't be found for a certain period of time, after which the robot switches back to the *Normal* state.

1.1.3 rqt_graph

1.2 ROS Messages and Parameters

Custom ROS **messages** are:

- **Planning.action**
  ```

## 1.3 geometry_msgs/PoseStamped target_pose

string stat geometry_msgs/Pose position ```
Describes the action that will be used both for the robot action server and the ball one: `target_pose` represents the "goal", `stat` and `position` are the "feedback": there's no "result" field.
I haven't defined any custom **ROS parameter**.

## 1.4 Packages and File List

Going in alphabetical order:

- **action**
  - `Planning.action`
    Action file described above.
- **config**
  - `motors_config.yaml`
    The file containing the description of the controllers and their parameters.
- **diagrams**
  - `Component_Diagram.png`, `State_Diagram.png` and `rosgraph.png`
    The three diagrams shown in this README file.
- **documentation**
  - **html** and **latex**
    Contain the output of *Doxygen*.
  - `assignment.conf`
    Configuration file used to run *Doxygen*.
- **launch**
  - `gazebo_world.launch`
    The launchfile which runs the gazebo server and client, the action servers, the finite state machine and spawns the ball, robot and human models.
- **scripts**
  - `go_to_point_ball.py`, `go_to_point_robot.py`, `person.py` and `state_machine.py`
    The modules of the architecture, as described in the *System Architecture* section.
- **urdf**
  ```

  - `ball.gazebo`, `ball.xacro`, `human.urdf`, `robot.gazebo` and `robot.xacro`
    The files which describe the models used for the ball, robot and human.

- **worlds**

  - `world_assignment.world`

  The world file in which the simulation environment is defined.

- `CMakeLists.txt` and `package.xml`

  Necessary files to compile and run the system nodes.

1.5 Installation and Running Procedure

First of all, clone this repository inside your ROS workspace's ∗"/src"∗ folder .
Then, navigate to the ∗"/scripts"∗ folder and make the Python scripts executable with:

```
$ chmod +x go_to_point_ball.py
$ chmod +x go_to_point_robot.py
$ chmod +x person.py
$ chmod +x state_machine.py
```

Go back to the root folder of your ROS workspace and execute:

```
$ catkin_make
$ catkin_make install
```

In a separate terminal run:

```
$ roscore
```

Finally, run the launchfile with this command:

```
$ roslaunch erl_second_assignment gazebo_world.launch
```

After a brief setup period in which gazebo is launched and the models are spawned, on the console you will see the transitions between states, when the robot and the ball reach their destinations, and other feedbacks from the system.

1.6 System's Features

This architecture is an evolution of the first assignment, with some improvements. The person can send commands to the ball ignoring completely the state of the robot, which will work in a reactive way. The requirements have all been satisfied, the person module controls the ball without problems and the finite state machine switches correctly between states. The robot uses the camera information to track the ball and when it stops it performs the requested head movements. Since the robot is a system on its own, if the person module or the ball action server fail or stop working no problems arise since it can still move randomly in the plane. This division makes the overall system modular and robust. An important point is that now the control of the robot motion is achieved via an action server, which is a more flexible, non-blocking option with respect to a service/client pattern and features the possibility both to cancel the current goal and to receive a constant feedback. I observed no strange behaviours and the feedback on the console is consistent with what is happening in gazebo.

1.7 System's Limitations

First of all, since we are in fact simulating the robot and its behaviour, we can't guarantee that it will work on a physical robot. I used only 1/4 of the whole arena for debugging purpouses and because the simulation is really slow (most probably due to my computer): a smaller arena makes the head movement sub-state more frequent, which doesn't almost happen when moving in a larger area. In this situation, other solutions should be considered. We aren't simulating collisions between ball and robot, so it may happen that the ball passes through the robot model: this is both unfeasible and in principle unwanted behaviour. Moreover, the robot can collide with the arena boundaries (it happened that when tracking the ball the robot went backwards and hit the barriers) which can make it fall to the ground. The tracking of the ball isn't perfect, so it may happen that it passes in front of the robot, which triggers the state to switch to "Play", and then it immediately goes out of sight, changing the state again to "Normal". When executing the roslaunch, the setup messages mess up with the initial feedback (e.g. the finite state machine, the person): a cleaner way to show the messages to the user should be considered. The robot doesn't show any kind of feedback to the human, such as a colored light. The ball can be recognized and tracked only if it's green.

1.8 Possible Technical Improvements

Right now the robot acts as a reactive system which can't take orders from a human. A nice improvement could be to implement sensing modules that pick up the user's commands and sends them to a processing component: in this way the robot could perform other types of actions. The robot's sensing capabilities could be improved to allow it to track any kind of ball, colored or not. In order to improve the robot's navigation and knowledge about the environment, a SLAM and/or autonomous navigation approach can be implemented.

1.9 Authors and Contacts

Davide Piccinini matricola S4404040 Emails:

- piccio98dp@gmail.com
- 4404040@studenti.unige.it

1.10 Doxygen Documentation

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 go_to_point_ball Namespace Reference

Functions

- def clbk_odom (msg)
- def change_state (state)
- def go_straight_ahead (des_pos)
- def done ()
- def planning (goal)
- def main ()

Variables

- position_ = Point()
- pose_ = Pose()
- int yaw_ = 0
- int state_ = 0
- desired_position_ = Point()
- int yaw_precision_ = math.pi / 9
- int yaw_precision_2_ = math.pi / 90
- float dist_precision_ = 0.1
- float kp_a = 3.0
- float kp_d = 0.5
- float ub_a = 0.6
- float lb_a = -0.5
- float ub_d = 2.0
- float z_back = 0.25
- pub = None
- pubz = None
- act_s = None

6.1.1 Detailed Description

Implements an action server that moves the ball.

6.1.2 Function Documentation

6.1.2.1 change_state()

```
def go_to_point_ball.change_state (
              state )
```

6.1.2.2 clbk_odom()

```
def go_to_point_ball.clbk_odom (
              msg )
```

6.1.2.3 done()

```
def go_to_point_ball.done ( )
```

6.1.2.4 go_straight_ahead()

```
def go_to_point_ball.go_straight_ahead (
              des_pos )
```

6.1.2.5 main()

```
def go_to_point_ball.main ( )
```

6.1.2.6 planning()

```
def go_to_point_ball.planning (
              goal )
```

6.1.3 Variable Documentation

6.1.3.1 act_s

```
go_to_point_ball.act_s = None
```

6.1.3.2 desired_position_

```
go_to_point_ball.desired_position_ = Point()
```

6.1.3.3 dist_precision_

```
float go_to_point_ball.dist_precision_ = 0.1
```

6.1.3.4 kp_a

```
float go_to_point_ball.kp_a = 3.0
```

6.1.3.5 kp_d

```
float go_to_point_ball.kp_d = 0.5
```

6.1.3.6 lb_a

```
float go_to_point_ball.lb_a = -0.5
```

6.1.3.7 pose_

```
go_to_point_ball.pose_ = Pose()
```

6.1.3.8 position_

```
go_to_point_ball.position_ = Point()
```

6.1.3.9 pub

```
go_to_point_ball.pub = None
```

6.1.3.10 pubz

```
go_to_point_ball.pubz = None
```

6.1.3.11 state_

```
int go_to_point_ball.state_ = 0
```

6.1.3.12 ub_a

```
float go_to_point_ball.ub_a = 0.6
```

6.1.3.13 ub_d

```
float go_to_point_ball.ub_d = 2.0
```

6.1.3.14 yaw_

```
int go_to_point_ball.yaw_ = 0
```

6.1.3.15 yaw_precision_

```
int go_to_point_ball.yaw_precision_ = math.pi / 9
```

6.1.3.16 yaw_precision_2_

```
int go_to_point_ball.yaw_precision_2_ = math.pi / 90
```

6.1.3.17 z_back

```
float go_to_point_ball.z_back = 0.25
```

6.2 go_to_point_robot Namespace Reference

Functions

- def clbk_odom (msg)
- def change_state (state)
- def normalize_angle (angle)
- def fix_yaw (des_pos)
- def go_straight_ahead (des_pos)
- def done ()
- def planning (goal)
- def main ()

Variables

- position_ = Point()
- pose_ = Pose()
- int yaw_ = 0
- int state_ = 0
- desired_position_ = Point()
- z
- int yaw_precision_ = math.pi / 9
- int yaw_precision_2_ = math.pi / 90
- float dist_precision_ = 0.1
- float kp_a = -3.0
- float kp_d = 0.5
- float ub_a = 0.6
- float lb_a = -0.5
- float ub_d = 0.6
- pub = None
- act_s = None

6.2.1 Detailed Description

Implements an action server that moves the robot.

6.2.2 Function Documentation

6.2.2.1 change_state()

```
def go_to_point_robot.change_state (
              state )
```

6.2.2.2 clbk_odom()

```
def go_to_point_robot.clbk_odom (
              msg )
```

6.2.2.3 done()

```
def go_to_point_robot.done ( )
```

6.2.2.4 fix_yaw()

```
def go_to_point_robot.fix_yaw (
              des_pos )
```

6.2.2.5 go_straight_ahead()

```
def go_to_point_robot.go_straight_ahead (
              des_pos )
```

6.2.2.6 main()

```
def go_to_point_robot.main ( )
```

6.2.2.7 normalize_angle()

```
def go_to_point_robot.normalize_angle (
              angle )
```

6.2.2.8 planning()

```
def go_to_point_robot.planning (
              goal )
```

6.2.3 Variable Documentation

6.2.3.1 act_s

```
go_to_point_robot.act_s = None
```

6.2.3.2 desired_position_

```
go_to_point_robot.desired_position_ = Point()
```

6.2.3.3 dist_precision_

```
float go_to_point_robot.dist_precision_ = 0.1
```

6.2.3.4 kp_a

```
float go_to_point_robot.kp_a = -3.0
```

6.2.3.5 kp_d

```
float go_to_point_robot.kp_d = 0.5
```

6.2.3.6 lb_a

```
float go_to_point_robot.lb_a = -0.5
```

6.2.3.7 pose_

```
go_to_point_robot.pose_ = Pose()
```

6.2.3.8 position_

```
go_to_point_robot.position_ = Point()
```

6.2.3.9 pub

```
go_to_point_robot.pub = None
```

6.2.3.10 state_

```
int go_to_point_robot.state_ = 0
```

6.2.3.11 ub_a

```
float go_to_point_robot.ub_a = 0.6
```

6.2.3.12 ub_d

```
float go_to_point_robot.ub_d = 0.6
```

6.2.3.13 yaw_

```
int go_to_point_robot.yaw_ = 0
```

6.2.3.14 yaw_precision_

```
int go_to_point_robot.yaw_precision_ = math.pi / 9
```

6.2.3.15 yaw_precision_2_

```
int go_to_point_robot.yaw_precision_2_ = math.pi / 90
```

6.2.3.16 z

```
go_to_point_robot.z
```

6.3 person Namespace Reference

Functions

- def moveBall ()

  Sends a goal to the ball's action server to move it to a random position.
- def disappearBall ()

  Makes the ball disappear by making it go under the plane.
- def person ()

  Randomly perfoms one of the two available commands.

Variables

- actC = None
- pos = PoseStamped()

6.3.1 Detailed Description

Mimics the behaviour of a person controlling the robot. The person can move the ball to a certain goal position or can make it disappear.

6.3.2 Function Documentation

6.3.2.1 disappearBall()

```
def person.disappearBall ( )
```

Makes the ball disappear by making it go under the plane.

6.3.2.2 moveBall()

```
def person.moveBall ( )
```

Sends a goal to the ball's action server to move it to a random position.
After the ball has reached the destination the person wait some time before issuing another command.

6.3.2.3 person()

```
def person.person ( )
```

Randomly perfoms one of the two available commands.

6.3.3 Variable Documentation

6.3.3.1 actC

```
person.actC = None
```

6.3.3.2 pos

```
person.pos = PoseStamped()
```

6.4 state_machine Namespace Reference

Classes

- class Normal

  Define Normal state.
- class Play

  Define Play state.
- class Sleep

  Define Sleep state.

Functions

- def checkForBall (ros_data)

  Use OpenCV to check if the robot camera has detected a ball.
- def trackBall (ros_data)

  Use OpenCV to check if the robot camera has detected a ball: if so, move the robot until the ball is at the center of the image and has a certain radius.
- def main ()

  State machine initialization.

Variables

- actC = None

  Action client.
- velPub = None

  Publishers.
- headJointPub = None
- imageSub = None

  Subscriber.
- pos = PoseStamped()

  Goal pose.
- int sleepCounter = 0

  Counter.
- bool ballFound = False

  Flag to notify that the robot has seen the ball.
- bool robotStopped = False

  Flag to notify that the robot has stopped.

6.4.1 Detailed Description

Defines the different robot behaviours and the transitions between them. Available states are NORMAL, SLEEP and PLAY.

6.4.2 Function Documentation

6.4.2.1 checkForBall()

```
def state_machine.checkForBall (
            ros_data )
```

Use OpenCV to check if the robot camera has detected a ball.

Parameters

ros_data	The compressed image picked up by the camera

6.4.2.2 main()

```
def state_machine.main ( )
```

State machine initialization.

6.4.2.3 trackBall()

```
def state_machine.trackBall (
              ros_data )
```

Use OpenCV to check if the robot camera has detected a ball: if so, move the robot until the ball is at the center of the image and has a certain radius.
When the robot stops, raise a flag.

Parameters

ros_data	The compressed image picked up by the camera

6.4.3 Variable Documentation

6.4.3.1 actC

```
state_machine.actC = None
```

Action client.

6.4.3.2 ballFound

```
bool state_machine.ballFound = False
```

Flag to notify that the robot has seen the ball.

6.4.3.3 headJointPub

```
state_machine.headJointPub = None
```

6.4.3.4 imageSub

```
state_machine.imageSub = None
```

Subscriber.

6.4.3.5 pos

```
state_machine.pos = PoseStamped()
```

Goal pose.

6.4.3.6 robotStopped

```
bool state_machine.robotStopped = False
```

Flag to notify that the robot has stopped.

6.4.3.7 sleepCounter

```
int state_machine.sleepCounter = 0
```

Counter.

6.4.3.8 velPub

```
state_machine.velPub = None
```

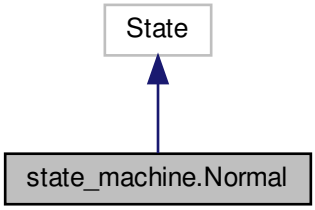
Publishers.

Chapter 7

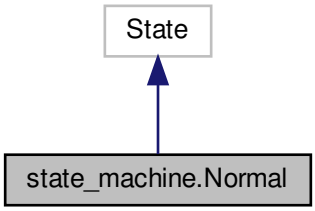
Class Documentation

7.1 state_machine.Normal Class Reference

Define Normal state.
Inheritance diagram for state_machine.Normal:



Collaboration diagram for state_machine.Normal:



Public Member Functions

- def __init__ (self)
- def execute (self, userdata)

Public Attributes

- sleepThreshold

7.1.1 Detailed Description

Define Normal state.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 __init__()

```
def state_machine.Normal.__init__ (
              self )
```

7.1.3 Member Function Documentation

7.1.3.1 execute()

```
def state_machine.Normal.execute (
              self,
              userdata )
```

7.1.4 Member Data Documentation

7.1.4.1 sleepThreshold

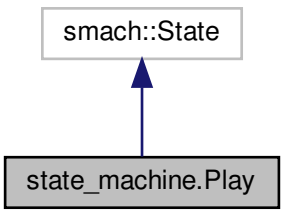
```
state_machine.Normal.sleepThreshold
```

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

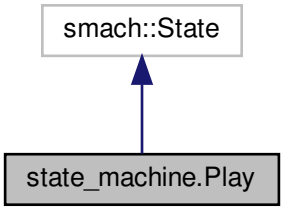
- scripts/state_machine.py

7.2 state_machine.Play Class Reference

Define Play state.
Inheritance diagram for state_machine.Play:



Collaboration diagram for state_machine.Play:



Public Member Functions

- def __init__ (self)
- def execute (self, userdata)

7.2.1 Detailed Description

Define Play state.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 __init__()

```
def state_machine.Play.__init__ (
              self )
```

7.2.3 Member Function Documentation

7.2.3.1 execute()

```
def state_machine.Play.execute (
              self,
              userdata )
```

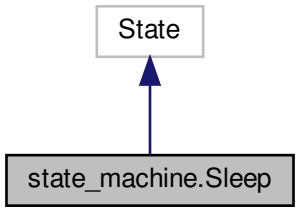
The documentation for this class was generated from the following file:

- scripts/state_machine.py

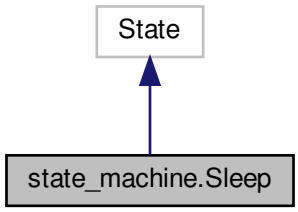
7.3 state_machine.Sleep Class Reference

Define Sleep state.

Inheritance diagram for state_machine.Sleep:



Collaboration diagram for state_machine.Sleep:



Public Member Functions

- def __init__ (self)
- def execute (self, userdata)

7.3.1 Detailed Description

Define Sleep state.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__()

```
def state_machine.Sleep.__init__ (
                self )
```

7.3.3 Member Function Documentation

7.3.3.1 execute()

```
def state_machine.Sleep.execute (
              self,
              userdata )
```

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

- scripts/state_machine.py

Chapter 8

File Documentation

8.1 CMakeLists.txt File Reference

8.2 README.md File Reference

8.3 scripts/go_to_point_ball.py File Reference

Namespaces

- go_to_point_ball

Functions

- def go_to_point_ball.clbk_odom (msg)
- def go_to_point_ball.change_state (state)
- def go_to_point_ball.go_straight_ahead (des_pos)
- def go_to_point_ball.done ()
- def go_to_point_ball.planning (goal)
- def go_to_point_ball.main ()

Variables

- go_to_point_ball.position_ = Point()
- go_to_point_ball.pose_ = Pose()
- int go_to_point_ball.yaw_ = 0
- int go_to_point_ball.state_ = 0
- go_to_point_ball.desired_position_ = Point()
- int go_to_point_ball.yaw_precision_ = math.pi / 9
- int go_to_point_ball.yaw_precision_2_ = math.pi / 90
- float go_to_point_ball.dist_precision_ = 0.1
- float go_to_point_ball.kp_a = 3.0
- float go_to_point_ball.kp_d = 0.5
- float go_to_point_ball.ub_a = 0.6
- float go_to_point_ball.lb_a = -0.5
- float go_to_point_ball.ub_d = 2.0
- float go_to_point_ball.z_back = 0.25
- go_to_point_ball.pub = None
- go_to_point_ball.pubz = None
- go_to_point_ball.act_s = None

8.4 scripts/go_to_point_robot.py File Reference

Namespaces

- go_to_point_robot

Functions

- def go_to_point_robot.clbk_odom (msg)
- def go_to_point_robot.change_state (state)
- def go_to_point_robot.normalize_angle (angle)
- def go_to_point_robot.fix_yaw (des_pos)
- def go_to_point_robot.go_straight_ahead (des_pos)
- def go_to_point_robot.done ()
- def go_to_point_robot.planning (goal)
- def go_to_point_robot.main ()

Variables

- go_to_point_robot.position_ = Point()
- go_to_point_robot.pose_ = Pose()
- int go_to_point_robot.yaw_ = 0
- int go_to_point_robot.state_ = 0
- go_to_point_robot.desired_position_ = Point()
- go_to_point_robot.z
- int go_to_point_robot.yaw_precision_ = math.pi / 9
- int go_to_point_robot.yaw_precision_2_ = math.pi / 90
- float go_to_point_robot.dist_precision_ = 0.1
- float go_to_point_robot.kp_a = -3.0
- float go_to_point_robot.kp_d = 0.5
- float go_to_point_robot.ub_a = 0.6
- float go_to_point_robot.lb_a = -0.5
- float go_to_point_robot.ub_d = 0.6
- go_to_point_robot.pub = None
- go_to_point_robot.act_s = None

8.5 scripts/person.py File Reference

Namespaces

- person

Functions

- def person.moveBall ()

  Sends a goal to the ball's action server to move it to a random position.
- def person.disappearBall ()

  Makes the ball disappear by making it go under the plane.
- def person.person ()

  Randomly perfoms one of the two available commands.

Variables

- person.actC = None
- person.pos = PoseStamped()

8.6 scripts/state_machine.py File Reference

Classes

- class state_machine.Normal

  Define Normal state.
- class state_machine.Sleep

  Define Sleep state.
- class state_machine.Play

  Define Play state.

Namespaces

- state_machine

Functions

- def state_machine.checkForBall (ros_data)

  Use OpenCV to check if the robot camera has detected a ball.
- def state_machine.trackBall (ros_data)

  Use OpenCV to check if the robot camera has detected a ball: if so, move the robot until the ball is at the center of the image and has a certain radius.
- def state_machine.main ()

  State machine initialization.

Variables

- state_machine.actC = None

  Action client.
- state_machine.velPub = None

  Publishers.
- state_machine.headJointPub = None
- state_machine.imageSub = None

  Subscriber.
- state_machine.pos = PoseStamped()

  Goal pose.
- int state_machine.sleepCounter = 0

  Counter.
- bool state_machine.ballFound = False

  Flag to notify that the robot has seen the ball.
- bool state_machine.robotStopped = False

  Flag to notify that the robot has stopped.