exp_assignment2

Generated by Doxygen 1.8.11

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 state_manager.find_and_follow_ball Class Reference

This class is used to detect and follow the green ball in the arena.

Public Member Functions

- def __init__ (self)

  It initializes a publisher to the image, one to the robot velocity and one to the camera motor.
- def callback (self, ros_data)

  This function is the callback function of the subscription to the camera image.

Public Attributes

- **image_pub**
- **vel_pub**
- **camera_pub**
- **subscriber**

4.1.1 Detailed Description

This class is used to detect and follow the green ball in the arena.

Definition at line 123 of file state_manager.py.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 def state_manager.find_and_follow_ball.__init__ (*self*)

It initializes a publisher to the image, one to the robot velocity and one to the camera motor.

It subscribes to the camera image

```
Initialize ros publisher, ros subscriber
```

Definition at line 127 of file state_manager.py.

```
127     def __init__(self):
128         '''Initialize ros publisher, ros subscriber'''
129 
130         # Init publishers
131         self.image_pub = rospy.Publisher("/output/image_raw/compressed",
132                                          CompressedImage, queue_size=1)
133         self.vel_pub = rospy.Publisher("/robot/cmd_vel",
134                                        Twist, queue_size=1)
135         self.camera_pub = rospy.Publisher(
136             "/robot/joint_position_controller/command", Float64, queue_size=1)
137 
138         # Subscribed Topic
139         self.subscriber = rospy.Subscriber("/robot/camera1/image_raw/compressed",
140                                            CompressedImage, self.callback,  queue_size=1)
141 
```

4.1.3 Member Function Documentation

4.1.3.1 def state_manager.find_and_follow_ball.callback (*self, ros_data*)

This function is the callback function of the subscription to the camera image.

It looks for a green contour in the image, plots a circle around it and makes the robot approach it. When the robot has the object at a specified distance, it stops, turns its head twice and again looks for the object. If the robot doesn't see the ball for 10 iterations in a row, it sets the ros parameter counter to 10 and then waits for it to be zero again.

Definition at line 147 of file state_manager.py.

```
147     def callback(self, ros_data):
148         global counter
149         global vel_camera
150         global MAX_COUNTER
151 
152         # Read counter ros parameter: proceed only if it's not max
153         counter = rospy.get_param('counter')
154         while counter == MAX_COUNTER:
155             time.sleep(1)
156 
157         if VERBOSE:
158             print('received image of type: "%s"' % ros_data.format)
159 
160         # Convert to cv2
161         np_arr = np.fromstring(ros_data.data, np.uint8)
162         image_np = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)  # OpenCV >= 3.0:
163 
164         # Color limits
165         greenLower = (50, 50, 20)
166         greenUpper = (70, 255, 255)
167 
168         # Create masks
169         blurred = cv2.GaussianBlur(image_np, (11, 11), 0)
170         hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
171         mask = cv2.inRange(hsv, greenLower, greenUpper)
172         mask = cv2.erode(mask, None, iterations=2)
173         mask = cv2.dilate(mask, None, iterations=2)
```

```
174
175         # Find contour
176         cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
177                                 cv2.CHAIN_APPROX_SIMPLE)
178         cnts = imutils.grab_contours(cnts)
179         center = None
180
181         # Only proceed if at least one contour was found
182         if len(cnts) > 0:
183
184             # Find the largest contour in the mask, then use it to compute the minimum enclosing circle and
    centroid
185             c = max(cnts, key=cv2.contourArea)
186             ((x, y), radius) = cv2.minEnclosingCircle(c)
187             M = cv2.moments(c)
188             center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
189
190             # Only proceed if the radius meets a minimum size
191             if radius > 10:
192
193                 # Draw the circle and centroid on the frame, then update the list of tracked points
194                 cv2.circle(image_np, (int(x), int(y)), int(radius),
195                            (0, 255, 255), 2)
196                 cv2.circle(image_np, center, 5, (0, 0, 255), -1)
197                 vel_Play = Twist()
198
199                 # Publish robot vel
200                 vel_Play.angular.z = -0.002*(center[0]-400)
201                 vel_Play.linear.x = -0.01*(radius-100)
202                 self.vel_pub.publish(vel_Play)
203
204                 # When robot has arrived: turn head
205                 if vel_Play.linear.x <= 0.05 and vel_Play.angular.z <= 0.05:
206
207                     # Stop robot completely
208                     vel_Play.angular.z = 0
209                     vel_Play.linear.x = 0
210                     self.vel_pub.publish(vel_Play)
211
212                     # Rotate camera
213                     rospy.set_param('rotate_camera', 1)
214
215             # Go near ball
216             else:
217                 vel_Play = Twist()
218                 vel_Play.linear.x = 0.5
219                 self.vel_pub.publish(vel_Play)
220
221         # Look for ball by turning on the spot
222         else:
223             vel_Play = Twist()
224             vel_Play.angular.z = 0.5
225             self.vel_pub.publish(vel_Play)
226
227             # Increase counter of iterations without seeing the ball
228             counter = counter+1
229             rospy.set_param('counter', counter)
230             time.sleep(1)
231             rospy.loginfo('counter incremented')
232
233             # If counter is max: stop
234             if counter == MAX_COUNTER:
235                 vel_Play.angular.z = 0
236                 self.vel_pub.publish(vel_Play)
237                 self.subscriber.unregister()  # JUST ADDED
238
239         # Show camera image
240         cv2.imshow('window', image_np)
241         cv2.waitKey(2)
242
```

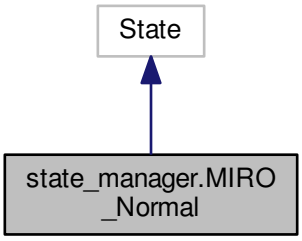
The documentation for this class was generated from the following file:

- scripts/state_manager.py

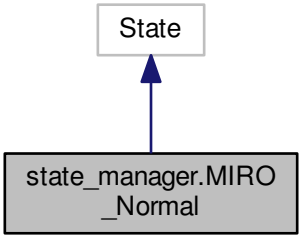
4.2 state_manager.MIRO_Normal Class Reference

Normal state of the smach machine.

Inheritance diagram for state_manager.MIRO_Normal:



Collaboration diagram for state_manager.MIRO_Normal:



Public Member Functions

- def __init__ (self)

  Init function for smach machine normal state.
- def execute (self, userdata)

  Smach machine state normal actions: In a loop: Calls move_dog() function to go to a random position, then subscribes to the camera image.

4.2.1 Detailed Description

Normal state of the smach machine.

Definition at line 302 of file state_manager.py.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 def state_manager.MIRO_Normal.__init__ (*self*)

Init function for smach machine normal state.

Definition at line 305 of file state_manager.py.

```
305     def __init__(self):
306
307         smach.State.__init__(self,
308                              outcomes=['sleep_command', 'play_command'])
309
```

4.2.3 Member Function Documentation

4.2.3.1 def state_manager.MIRO_Normal.execute (*self, userdata*)

Smach machine state normal actions: In a loop: Calls move_dog() function to go to a random position, then subscribes to the camera image.

It reads the ball ros parameter: if it's set to 2 it waits. If it's set to 0, it moves the robot to another random position. If it's set to 1 it switches to play state. At the end of the loop: it switches to sleep state.

Returns

c: command to switch between states.

Definition at line 317 of file state_manager.py.

```
317     def execute(self, userdata):
318
319         global subscriberNORM, LOOPS
320
321         for i in range(0, LOOPS):
322
323             # Go rand (then set rand)
324             time.sleep(3)
325             rospy.loginfo('normal: going rand')
326             # ([random.randrange(0, 9),random.randrange(0, 9), 0])
327             move_dog([-5, 5, 0])
328
329             # Look for ball
330             time.sleep(3)
331             rospy.loginfo('normal: looking for ball')
332             subscriberNORM = rospy.Subscriber("camera1/image_raw/compressed",
333                                               CompressedImage, find_ball,  queue_size=1)
334
335             while rospy.get_param('ball') == 2:
336                 time.sleep(1)
337
338             ball = rospy.get_param('ball')
339             rospy.set_param('ball', 2)
340
341             # Case 0: no ball --> continue in normal state
342             if ball == 0:
343                 rospy.loginfo(
344                     'normal: i see no ball, i will continue doing my things')
345
346             # Case 1: ball --> go in play state
347             elif ball == 1:
348                 rospy.loginfo(
349                     'normal: i do see ball, i will chase it: set play')
350
351                 c = 'play_command'
352                 return c
353
354             # Case err: ball wasn't set yet
355             else:
356                 rospy.loginfo('normal: ball was not set yet')
357                 time.sleep(3)
358
359         c = 'sleep_command'
360         return c
361
362
```

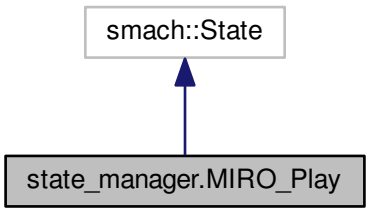
The documentation for this class was generated from the following file:

- scripts/state_manager.py

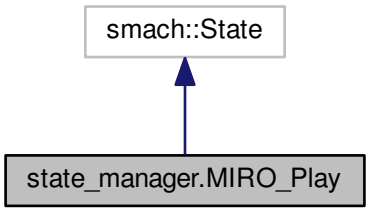
4.3 state_manager.MIRO_Play Class Reference

Play state of the smach machine.

Inheritance diagram for state_manager.MIRO_Play:



Collaboration diagram for state_manager.MIRO_Play:



Public Member Functions

- def __init__ (self)

  Init function for smach machine play state.
- def execute (self, userdata)

  Smach machine state play actions: find and follow the ball.

Public Attributes

- **camera_pub**

4.3.1 Detailed Description

Play state of the smach machine.

Definition at line 366 of file state_manager.py.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 def state_manager.MIRO_Play.__init__ (*self*)

Init function for smach machine play state.

Definition at line 369 of file state_manager.py.

```
369     def __init__(self):
370
371         smach.State.__init__(self,
372                              outcomes=['normal_command'])
373
374         self.camera_pub = rospy.Publisher("/robot/joint_position_controller/command",
375                                           Float64, queue_size=1)
376
```

4.3.3 Member Function Documentation

4.3.3.1 def state_manager.MIRO_Play.execute (*self, userdata*)

Smach machine state play actions: find and follow the ball.

If robot reaches the ball and stops: rotate camera. If robot can not see ball for a while (counter=MAX_COUNTER): switch to normal state.

Returns

c: command to switch between states.

Definition at line 381 of file state_manager.py.

```
381     def execute(self, userdata):
382
383         global subscriberPLAY
384         global MAX_COUNTER
385
386         time.sleep(3)
387         rospy.loginfo('play: chase ball')
388
389         # Find and follow green ball
390         ic = find_and_follow_ball()
391         rotated = 0
392
393         while rospy.get_param('counter') < MAX_COUNTER:
394             '''
395             time.sleep(3)
396             if rotated == 1:
397                 time.sleep(10)
398
399             rotated = 0
400             '''
401
402             if rospy.get_param('rotate_camera') == 1:
403
404                 rospy.loginfo('rotating camera')
```

```
405                 vel_camera.data = 0
406
407                 # Turn head left
408                 while vel_camera.data < 0.5:
409                     vel_camera.data = vel_camera.data + 0.1
410                     self.camera_pub.publish(vel_camera)
411                     time.sleep(1)
412
413                 # Turn head to center
414                 while vel_camera.data > 0.01:
415                     vel_camera.data = vel_camera.data - 0.1
416                     self.camera_pub.publish(vel_camera)
417                     time.sleep(1)
418
419                 # Turn head right
420                 while vel_camera.data > 0.5:
421                     vel_camera.data = vel_camera.data - 0.1
422                     self.camera_pub.publish(vel_camera)
423                     time.sleep(1)
424
425                 # Turn head to center
426                 while vel_camera.data < 0.01:
427                     vel_camera.data = vel_camera.data + 0.1
428                     self.camera_pub.publish(vel_camera)
429                     time.sleep(1)
430
431                 # Rotation finished
432                 rospy.set_param('rotate_camera', 0)
433
434                 # Wait until dog can't see the ball no more, and switch to normal state
435                 rospy.set_param('counter', 0)  # JUST MODIFIED
436
437         rospy.loginfo('back to normal, havent seen ball for a while')
438         c = 'normal_command'
439         return c
440
441
```

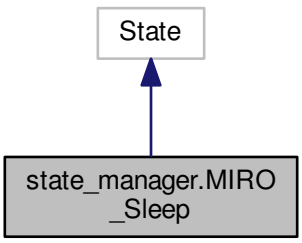
The documentation for this class was generated from the following file:

- scripts/state_manager.py

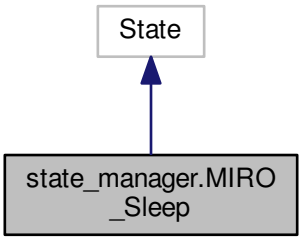
4.4 state_manager.MIRO_Sleep Class Reference

Sleep state of the smach machine.

Inheritance diagram for state_manager.MIRO_Sleep:



Collaboration diagram for state_manager.MIRO_Sleep:



Public Member Functions

- def __init__ (self)

  Init function for smach machine sleep state.
- def execute (self, userdata)

  Smach machine state sleep actions: Calls dog() function to move towards the kennel, waits some seconds and exits state.

4.4.1 Detailed Description

Sleep state of the smach machine.

Definition at line 276 of file state_manager.py.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 def state_manager.MIRO_Sleep.__init__ (*self*)

Init function for smach machine sleep state.

Definition at line 279 of file state_manager.py.

```
279     def __init__(self):
280
281         smach.State.__init__(self,
282                              outcomes=['normal_command'])
283
```

4.4.3 Member Function Documentation

4.4.3.1 def state_manager.MIRO_Sleep.execute (*self, userdata*)

Smach machine state sleep actions: Calls dog() function to move towards the kennel, waits some seconds and exits state.

Returns

c: command to switch between states.

Definition at line 287 of file state_manager.py.

```
287     def execute(self, userdata):
288
289         # Go home and sleep
290         time.sleep(3)
291         rospy.loginfo('sleep: go home')
292         move_dog([0, 0, 0])
293         time.sleep(5)
294
295         # Change state
296         c = 'normal_command'
297         return c
298
```

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

- scripts/state_manager.py

Chapter 5

File Documentation

5.1 scripts/human_commands.py File Reference

This node implements an actionlib client to move the ball-.

Functions

- def human_commands.human_client ()

  This function implements a client for the ball motion server.
- def human_commands.main ()

  Ros node that calls the client for the ball motion server.

5.1.1 Detailed Description

This node implements an actionlib client to move the ball-.

5.1.2 Function Documentation

5.1.2.1 def human_commands.human_client ( )

This function implements a client for the ball motion server.

Definition at line 24 of file human_commands.py.

```
24 def human_client():
25
26     client = actionlib.SimpleActionClient(
27         '/reaching_goal', exp_assignment2.msg.PlanningAction)
28
29     # Waits until the action server has started up and started
30     # listening for goals.
31     client.wait_for_server()
32
33     # Creates a goal to send to the action server.
34     goal = exp_assignment2.msg.PlanningGoal()
35     '''
36     rospy.loginfo('print x')
37     in1 = int(input())
```

```
38      rospy.loginfo('print y')
39      in2 = int(input())
40      rospy.loginfo('print z')
41      in3 = int(input())
42      '''
43      goal.target_pose.pose.position.x = random.randrange(0, 9)
44      goal.target_pose.pose.position.y = random.randrange(0, 9)
45      goal.target_pose.pose.position.z = random.choice([0, 1, 10])
46
47      # Sends the goal to the action server.
48      client.send_goal(goal)
49
50      # Waits for the server to finish performing the action.
51      client.wait_for_result()
52
53      return client.get_result()
54
```

5.2 scripts/state_manager.py File Reference

This node implements a smach state machine to simulate a dog that can sleep, play and wander around.

Classes

- class state_manager.find_and_follow_ball

  This class is used to detect and follow the green ball in the arena.
- class state_manager.MIRO_Sleep

  Sleep state of the smach machine.
- class state_manager.MIRO_Normal

  Normal state of the smach machine.
- class state_manager.MIRO_Play

  Play state of the smach machine.

Functions

- def state_manager.find_ball (ros_data)

  This function is the callback for the normal state.
- def state_manager.move_dog (target)

  This function is a client for the robot motion server.
- def state_manager.main ()

  Ros node that implements a state machine with three states: sleep, play, normal.

Variables

- bool **state_manager.VERBOSE** = False
- int **state_manager.LOOPS** = 10
- **state_manager.vel_camera** = Float64()
- **state_manager.vel_Norm** = Twist()
- int **state_manager.MAX_COUNTER** = 25
- int **state_manager.SEARCH_FOR_BALL** = 15
- **state_manager.vel_pub** = rospy.Publisher("/roboy/cmd_vel", Twist, queue_size=1)

5.2.1 Detailed Description

This node implements a smach state machine to simulate a dog that can sleep, play and wander around.

5.2.2 Function Documentation

5.2.2.1 def state_manager.find_ball (*ros_data*)

This function is the callback for the normal state.

It checks if the ball is visible from the camera. If it is: it sets the ros parameter ball=1. If it isn't: it sets the ros parameter ball=0.

Definition at line 52 of file state_manager.py.

```
52 def find_ball(ros_data):
53
54     global subscriberNORM, vel_Norm, SEARCH_FOR_BALL
55
56     time.sleep(1)
57
58     # Init velocity
59     vel_Norm.linear.x = 0
60     vel_Norm.linear.y = 0
61     vel_Norm.linear.z = 0
62     vel_Norm.angular.x = 0
63     vel_Norm.angular.y = 0
64     vel_Norm.angular.z = 0
65
66     # rospy.loginfo('entered img NORM fnct')
67
68     # Convert to cv2
69     np_arr = np.fromstring(ros_data.data, np.uint8)
70     image_np = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
71
72     # Color limits
73     greenLower = (50, 50, 20)
74     greenUpper = (70, 255, 255)
75
76     # Create masks
77     blurred = cv2.GaussianBlur(image_np, (11, 11), 0)
78     hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
79     mask = cv2.inRange(hsv, greenLower, greenUpper)
80     mask = cv2.erode(mask, None, iterations=2)
81     mask = cv2.dilate(mask, None, iterations=2)
82
83     # Find contour
84     cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
85                             cv2.CHAIN_APPROX_SIMPLE)
86     cnts = imutils.grab_contours(cnts)
87     center = None
88
89     # Only proceed if at least one contour was found
90     if len(cnts) > 0:
91         rospy.loginfo('img NORM fnct: ball')
92         # Ball was found
93         rospy.set_param('ball', 1)
94         time.sleep(1)
95         subscriberNORM.unregister()
96
97     else:
98         # Search again in loop
99         for i in range(0, SEARCH_FOR_BALL):
100             vel_Norm.angular.z = 0.2
101             vel_pub.publish(vel_Norm)
102             time.sleep(1)
103             cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
104                                     cv2.CHAIN_APPROX_SIMPLE)
105             cnts = imutils.grab_contours(cnts)
106             center = None
107
108             if len(cnts) > 0:
109                 rospy.loginfo('img NORM fnct: ball after turning')
110                 # Ball was found
111                 rospy.set_param('ball', 1)
```

```
112                 time.sleep(1)
113                 subscriberNORM.unregister()
114
115         rospy.loginfo('img NORM fnct: no ball')
116         # Ball was not found
117         rospy.set_param('ball', 0)
118         time.sleep(1)
119         subscriberNORM.unregister()
120
121
```

5.2.2.2 def state_manager.main ( )

Ros node that implements a state machine with three states: sleep, play, normal.

It also initializes the ball and counter parameters.

Definition at line 446 of file state_manager.py.

```
446 def main():
447
448     rospy.init_node('state_manager')
449
450     # Init ball ros param to 2 and counter ros param to 0.
451     rospy.set_param('ball', 2)
452     rospy.set_param('counter', 0)
453     rospy.set_param('rotate_camera', 0)
454
455     time.sleep(3)
456
457     # Create a SMACH state machine
458     sm = smach.StateMachine(outcomes=['container_interface'])
459
460     with sm:
461         # Add states to the container
462         smach.StateMachine.add('SLEEP', MIRO_Sleep(),
463                                transitions={'normal_command': 'NORMAL'})
464         smach.StateMachine.add('NORMAL', MIRO_Normal(),
465                                transitions={'sleep_command': 'SLEEP',
466                                             'play_command': 'PLAY'})
467         smach.StateMachine.add('PLAY', MIRO_Play(),
468                                transitions={'normal_command': 'NORMAL'})
469
470     # Create and start the introspection server for visualization
471     sis = smach_ros.IntrospectionServer('server_name', sm, '/SM_ROOT')
472     sis.start()
473
474     # Execute the state machine
475     outcome = sm.execute()
476
477     # Wait for ctrl-c to stop the application
478     rospy.spin()
479     cv2.destroyAllWindows()
480     sis.stop()
481
482
```

5.2.2.3 def state_manager.move_dog (*target*)

This function is a client for the robot motion server.

It sends the desired position.

Definition at line 246 of file state_manager.py.

```
246 def move_dog(target):
247
248     time.sleep(3)
249
250     client = actionlib.SimpleActionClient(
251         '/robot_reaching_goal', exp_assignment2.msg.PlanningAction)
252
253     client.wait_for_server()
254
255     rospy.loginfo('Going to %d %d %d', target[0], target[1], target[2])
256
257     # Creates a goal to send to the action server.
258     goal = exp_assignment2.msg.PlanningGoal()
259     goal.target_pose.pose.position.x = target[0]
260     goal.target_pose.pose.position.y = target[1]
261     goal.target_pose.pose.position.z = target[2]
262
263     # Sends the goal to the action server.
264     client.send_goal(goal)
265
266     # Waits for the server to finish performing the action.
267     client.wait_for_result()
268
269     rospy.loginfo('arrived, exiting dog fnct')
270
271     return client.get_result()
272
```

Index