Robotics - Assignment 2

Francesco Saverio Zuppichini, Alessia Ruggeri

May 3, 2018

1 How to run

1.1 Gazebo

In order to launch the Gazebo environment, you need to run:

```
$ roslaunch thymio_course_skeleton
        custom_world.launch name:=thymio10 world:=wall
```

1.2 Code

Assuming you are in the correct working repository, uncomment the first exercise code if needed and run:

```
$ python3 main.py
```

2 Exercise 1

Since we believe that composition is better than inheritance, we implemented a callback approach to add functionality to the `BasicThymio` class. We created the following 'abstract' class:

```
class Hook:
    def __init__(self):
        pass

    def on_update_pose(self, thymio):
        pass
```

We changed the constructor in the following way:

```
def __init__(self, thymio_name, hooks=[]):
  # ...
  self.hooks = hooks
  # ...
```

Then, in the update_state function, we iterate trough the hooks and call the on_update_pose passing the thymio instance. In this way, we can dynamically switch the robot's behavior.

```
class EigthWriter(Hook):
    def __init__(self, linear_vel=0.1, angular_vel=0.5, tol=10
                                        **(-3)):
        self.initial_pose = None
        self.has_move = False
        self.linear_vel = linear_vel
        self.angular_vel = angular_vel
        self.tol = tol
        self.has_finished = False
        self.count = 0

    def on_update_pose(self, thymio):
        if thymio == None: return

        if self.has_finished:
            thymio.vel_msg.linear.x = 0
            thymio.vel_msg.angular.z = 0
            return

        if self.initial_pose == None: self.initial_pose =
                                        from_pos_to_np(thymio.
                                        current_pose)

        current_pose = thymio.current_pose
        pose = from_pos_to_np(current_pose)

        thymio.vel_msg.linear.x = self.linear_vel
        thymio.vel_msg.angular.z = self.angular_vel

        err = np.mean(np.abs(self.initial_pose - pose))
        is_far_from_start = err > self.tol

        self.has_move = self.has_move or is_far_from_start

        if self.has_move and not is_far_from_start:
            self.initial_pose = pose # reset pose to calibrate
            self.angular_vel  = -1 * self.angular_vel
            self.count += 1
        self.has_finished =  self.count == 2
```

In order to write the eight, we check the position from which the robot start to move and, after the robot has written the first circle and has reached again the initial position, we change the angular velocity to its opposite to draw the second circle.

You can find the video of the `rviz` at the following link (it was too big for iCorsi):
`https://drive.google.com/drive/folders/1UzEzDunEO9QfnxQmKhe9UGOZlbFhpR2a?usp=sharing`

3 Exercise 2

Before doing anything, the thymio needs to get the data from the sensor. Using some python tricks, we defined a callback function that is able to take the sensor data, its id and its name. The callback function looks like:

```
 def sensors_callback(self, data, sensor_id, name):
        self.sensors_cache[name] = data

        try:
            for hook in self.hooks:
                hook.on_receive_sensor_data(thymio, data,
                                                        sensor_id, name
                                                        )
        except KeyError:
            pass
```

After that, similar to what we did before, we created a new hook, `GoCloseTurnAndGoForward`, that will be responsible to actually move the robot.

```
class GoCloseTurnAndGoForward(Hook):

    def __init__(self):
        self.stable = False
        self.approach_wall = True
        self.should_turn = False
        self.forward = False

    def done_on_align(self):
        self.thymio.vel_msg.angular.z = 0.0
        self.thymio.vel_msg.linear.x = 0.0
        self.approach_wall = False
        self.should_turn = True
        print('aligned')
        self.thymio.start = self.thymio.time_elapsed

    def not_touch_align(self):
        self.thymio.vel_msg.linear.x = 0.1

    def has_touched_align(self):
        self.thymio.vel_msg.linear.x = 0.0
        self.thymio.vel_msg.angular.z = 0.0
        print('touched')

    def turn_until(self, data,name,  name1, name2, not_touch,
                                    has_touch, done):
        if name != name1 and name != name2: return
        has_touched = np.abs(data.max_range - data.range) > 0.
                                            02
        e = np.abs(self.thymio.sensors_cache[name1].range -
                                            self.thymio.
                                            sensors_cache[name2].
                                            range)
        print  e
        self.stable = e < 0.015
        if not has_touched:
            not_touch()
        if self.stable and has_touched:
            done()
        if has_touched and not self.stable:
            has_touch()
            step = self.thymio.pid.step(e, 0.01)
            if name == name1:
                if  data.range < self.thymio.sensors_cache[
                                                    name2].range:
                    self.thymio.vel_msg.angular.z = step
            elif name == name2:
                if  data.range < self.thymio.sensors_cache[
                                                    name1].range:
                    self.thymio.vel_msg.angular.z = -step

    def on_receive_sensor_data(self,thymio,data, sensor_id,
                                    name):
        self.thymio = thymio

        if self.approach_wall:
            self.turn_until(data,name,'left','right',self.
                                            not_touch_align,
                                            self.
                                            has_touched_align,
                                            self.done_on_align)
        elif self.should_turn:
            if thymio.time_elapsed - thymio.start <  32 * 10 *
                                            2 * 2:
                thymio.vel_msg.angular.z = 0.5
            else:
                thymio.vel_msg.angular.z = 0
                self.forward = True
                self.should_turn = False
                thymio.start = thymio.time_elapsed
        elif self.forward:
            # multiply by 9 since the rear sensors are more or
                                            less 10 cm
            if thymio.time_elapsed - thymio.start <  9 * 10 *
                                            10 * 2 * 2:
                thymio.vel_msg.linear.x = 0.1
            else:
                self.forward = False
                thymio.vel_msg.linear.x = 0
```

We used a PID controller to turn the robot to face the wall. However, the parameters could be tuned better. You can also find a old implementation that uses a bang-bang controller.

4 Exercise 3

We extended the `GoCloseTurnAndGoForward` to add this functionality, the code is in the previous section. We faced one main problem: the rear sensors were not working. Basically, the `range` value was always equal to the `max_range`. So we decided to use an open loop approach and just turn the robot for the correct amount of time.

5 Bonus - Exercise 4

For the first bonus exercise using the real thymio, we faced some problems with the position error, which in the real world is obviously bigger than in the simulator. For this reason, we needed to increase the error tolerance to make it work.

6 Bonus - Exercise 5

There is a big difference between the simulator and the real thymio data. In the first case, when there is nothing in front of the thymio, the range is set to `max_range`, but in the real world it is set to `inf`. Moreover, when an object is closer than 1 cm, the data is `-inf` while in the simulator we still have the correct number. For this reasons, we needed to change the code to make it work. You can find a first draft of the new code inside the `core/after_bonus` folder. We faced also problems with the thymio that was not working, as you can read on the forum, and we were also not able to stop the robot by using the rospy.on_shutdown routine, so we had to restart the machine everytime, loosing around 5/10 minutes for each try. We still hope to get some bonus points for the second parts since we tried hard to fix it.

All the code can be found in the GitHub repository:
`https://github.com/FrancescoSaverioZuppichini/Robotics-2018.git`