Robotics - Assignment 2

Francesco Saverio Zuppichini

May 3, 2018

1 How to run

1.1 Gazebo

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

1.2 Code

Assuming you are in the correct working repo. If needed, uncomment the first exercise code 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 behaviour.

```
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
```

Basically, we check when we start to move and, after we reach again the initial position, we change the angular velocity to draw the second zero. You can find the video of the `rviz` at the following link (it was too big for iCorsi):

https://drive.google.com/open?id=180JgrrSszP0WiqxDNdxkJFy6d1ov2-rN

3 Exercise 2

Before doing anything, we need to get the data from the sensor. Using some python tricks, we define 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 create a new hook, `GoCloseTurnAndGoForward`, that will be response 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 decide to use an open loop approach and just turn the robot for the correct amount of time.

5 Bonus - Exercise 1