Weekly project – Python 3 for Robotics

Emmanouil Markodimitrakis - October 2021

I have spitted my code in two classes, one for the Controller (controller_module.py) and one for the Robot (robot_module.py), both have initialization function in order to initialize the properties. The Robot class has also setters and getters for every property. Below I describe the classes in more details. The main (main.py) module is the main script of the project that I initialize all the classes and implement the core functionality.

Robot module

Contains the Robot class, setters and getters and a function to update the robot's position.

Properties, setters & getters

- **name (str)** – The name of the robot
- **max_speed (float)** – The robot's maximum speed, set by the initial function
- **initial_pos ((float), (float))** – The initial position of the robot, set by the user and changes only on the initialization
- **current_pos ((float), (float))** – Current position is firstly inisialized equal to the initial_pos and it get updated constantly by the update_position function
- **rotational_speed (float)** – The rotational speed of the robot, set to 0 by default
- **forward_speed (float)** – The forward speed of the robot, set to 0 by default

Functions

- **update_position(velocity)** – Is called by the main module, get as input the velocity and update the robot's position by adding the velocity to the current position.

```
def update_position(self, vdt):
        """
        Update the robot's current position

        Parameters:
            velocity (float): Velocity
        """

        self.__current_pos = self.__current_pos + velocity
```

Controller Module

Contains the Controller class, and some functions in order to implement the proportional control.

Properties

- **forward_speed_gain (float)** – Initialized on the class construction with a constant that is used to calculate the proportional forward speed.

- **rotational_speed_gain (float)** - Initialized on the class construction with a constant that is used to calculate the proportional rotational speed.

Functions

- **distance_to_target(current_pos, target_pos)** – Get as input the current of the robot and target position, calculate the Euclidean distance and return it.

```
def distance_to_target(self, current_pos, target_pos):
        """
        Calculate the Euclidean distance between the points (X, Y) and (X', Y')

        Parameters:
            current_pos ((float), (float)): (X, Y) - Current position
            targer_pos ((float), (float)): (X', Y') - Target position

        Returns:
            euclidean_distance (float): L2 Euclidean distance between the two points
        """
        current_pos = np.array(current_pos)
        target_pos = np.array(target_pos)

        euclidean_distance = np.sqrt(np.power(current_pos - target_pos, 2).sum())
        return euclidean_distance
```

- **calculate_theta(current_pos, target_pos)** – Get as input the current of the robot and target position, calculate and return the following formula atan2((Y' – Y) , (X' – X)). This is the theta angle that we need to calculate the velocity.

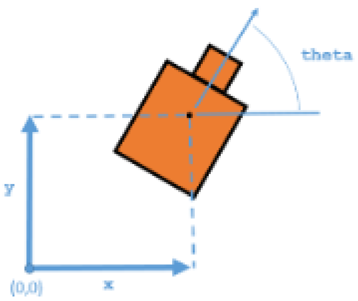

Figure 1: Robot localization sketch

```
def calculate_theta(self, current_pos, target_pos):
        theta = math.atan2( (target_pos[1] - current_pos[1]), (target_pos[0] - current_pos[0]) )
        return theta
```

- **calculate_velocity(K, theta, dt)** – Where input K is the proportional constant that the user specifies, theta is the angle we calculate in the above function and dt is the iteration time. Using these variables it calculates the velocity with the following formula:
  $V\,\Delta t = (V_x, V_y)\Delta t = (V_f.\cos(\theta), V_f.\sin(\theta))\Delta t = (V_f.\cos(\theta_{k-1}+\omega\Delta t), V_f.\sin(\theta_{k-1}+\omega\Delta t))\Delta t$

```
def calculate_velocity(self, K, theta, dt):
        """
        Calculate forward speed and angular velocity

        Parameters:
            K (int): Proportional constant
            theta (float): Angle
            dt (float): Iteration time

        Returns:
            velocity (float): Velocity
        """
        velocity = np.array([self.forward_speed * math.cos(theta + self
.rotational_speed * dt), self.forward_speed * math.sin(theta + self.rot
ational_speed * dt)])
        return velocity
```

- **control(theta_error, K, dt, current_pos, target_pos) –** This is the main function of this module, it calculates two errors, distance_error is used to update the forward speed and theta_error to update the rotational speed. It follows the logic of a proportional controller, by correcting the robot's orientation first with forward speed equals to 0, proportional to the theta error. Once the orientation is corrected, the robot starts moving with forward speed, proportional to the distance error.

```
def control(self, distance_to_target, current_theta, dt, current_pos, target_p
os):
        target_theta = self.calculate_theta(current_pos, target_pos)

        theta_error = self.rotational_speed_gain * (target_theta - current_the
ta)
        distance_error = self.forward_speed_gain * (distance_to_target)

        velocity = self.forward_speed_gain * self.calculate_velocity(target_th
eta, dt, theta_error, distance_error)

        # 5 degrees = 0.08 rads
        if current_theta == 0 or theta_error >= 5:
            # Stop the robot and correct the angle again
            velocity = 0
            forward_speed = 0
            print(f"theta_error:{theta_error} \t for_speed:{forward_speed} \t
velocity{velocity}")
            return target_theta, velocity, theta_error, forward_speed
```

```
        elif theta_error < 5:
            # Forwad speed can be non-
zero, while theta orentation still being corrected
            print(f"theta_error:{theta_error} \t for_speed:{distance_error} \t
 velocity{velocity}")
            return target_theta, velocity, theta_error, distance_error
```

Main Module

First of all, we initialize the attributes and the we initialize two object, one for the robot and another for the controller. Then we run the visualization loop and inside that loop we call the controller's function move_robot in order to calculate the velocity and change the robot's position.

Attributes

- **K = 1 –** The constant we adjust to change robot behaviour
- **iteration_time_sec = 0.001** – Iteration time 0.001 milisecond
- **target_pos = np.array([1, 2])** – Initialize target position to X=1 and Y=2
- **current_time = time.time()** – Initialize current time
- **theta_error = 0** - Initialize theta error to zero

Objects

- **robot = Robot("STUPIDO", 0.03, np.array([1, 1]))**
  - robot_name set to 'STUPIDO'
  - max_speed set to 0.03
  - initial_pos set to point X=1, Y=1
- **controller = Controller(robot.forward_speed, robot.rotational_speed)**
  - forward_speed set equal to robot's forward_speed
  - rotational_speed set equal to robot's rotational_speed

Functions

- **move_robot()** – Is executed inside the visualization loop in order to compute the distance to target, call the controllers control function, and get as output the new theta, velocity, rotational speed and forward speed values.
  Furthermore, this function updates the robot's position by calling the update_position function with velocity as input and set the new speeds to the robot object. The function returns False if the distance to target is lower that 0.01 and terminates the outer visualization loop.

```
def move_robot():
    distance_to_target = controller.distance_to_target(robot.current_pos, targ
et_pos)
    if distance_to_target < 0.01:
        print('Target reached!')
        return False
    else:
```

```
        # Compute forward and rotation speed with controller
        # set speed to robot
        global current_time
        t_k = current_time
        time.sleep(iteration_time_sec)
        current_time = time.time()

        dt = current_time - t_k

        # Update robot pos with t_k and current_time
        global theta
        theta, velocity, rotational_speed, forward_speed = controller.control(
distance_to_target, theta, dt, robot.current_pos, target_pos)

        robot.rotational_speed = rotational_speed
        robot.forward_speed = forward_speed
        robot.update_position(velocity)

        # print(f"distance_to_target: {distance_to_target} \t dt:{dt} \t  Curr
ent Pos:{robot.current_pos} \t Theta error:{theta_error} ")
        return True
```