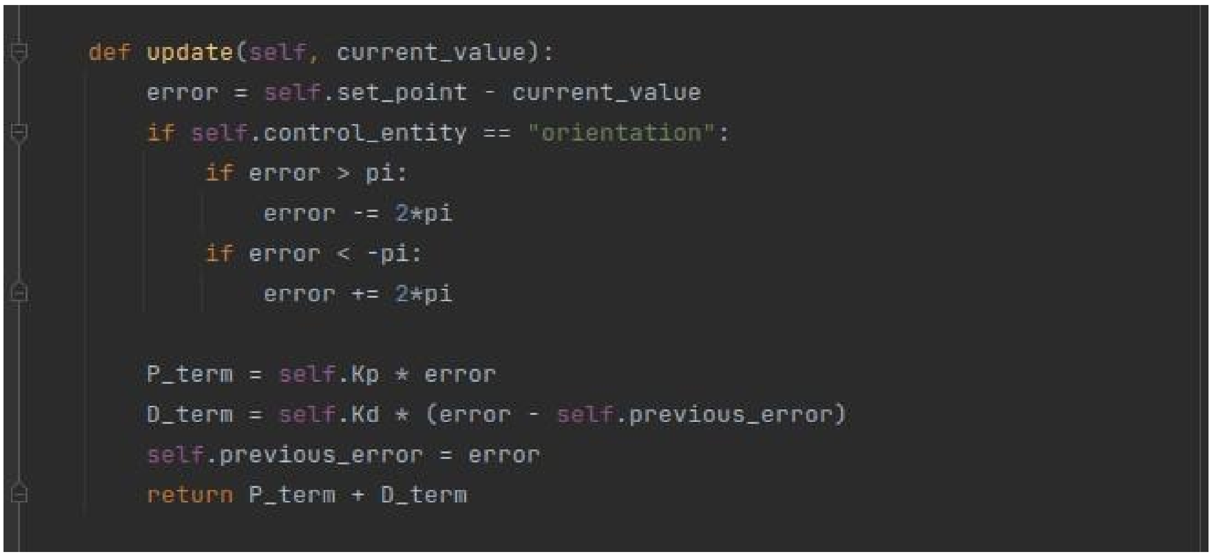
ROS Assignment 4 Report

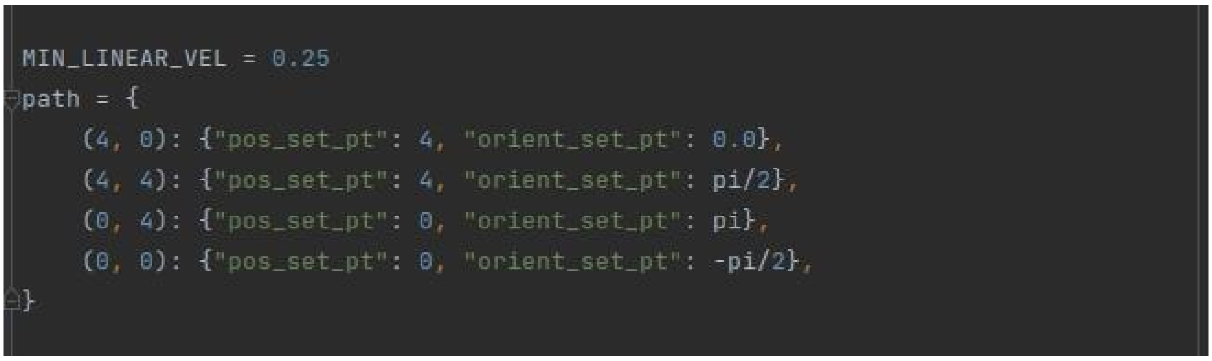
Script Breakdown

1. The ‘update’ function in the controller class was updated to return the control input. The change of orientation for angles greater than pi and less than -pi was taken into account if the controller was instantiated for orientation control.



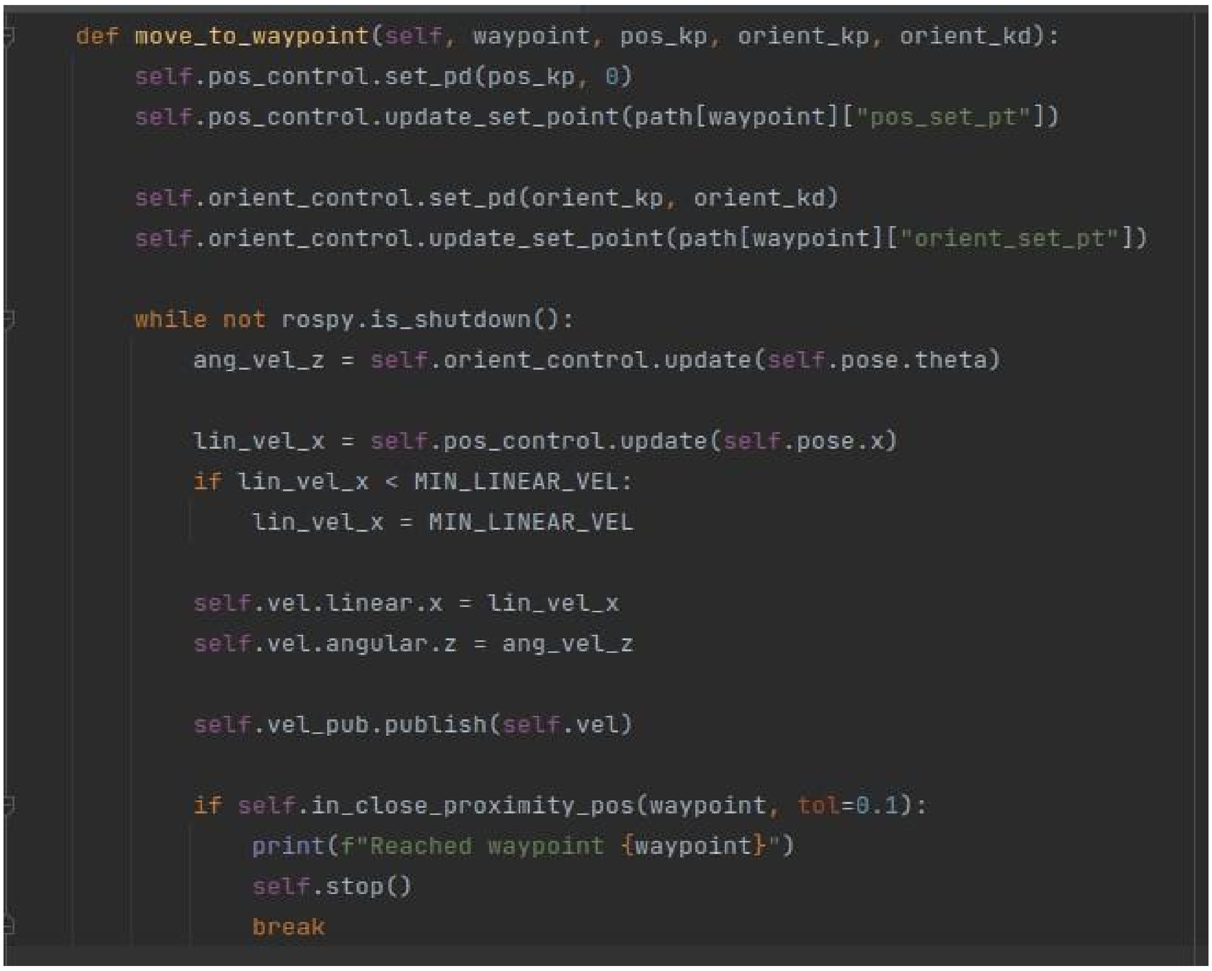
```python
def update(self, current_value):
    error = self.set_point - current_value
    if self.control_entity == "orientation":
        if error > pi:
            error -= 2*pi
        if error < -pi:
            error += 2*pi

    P_term = self.Kp * error
    D_term = self.Kd * (error - self.previous_error)
    self.previous_error = error
    return P_term + D_term
```

2. Some initializations were made.
   - Since position control is handled by just the P controller, a minimum velocity is defined to deal with the steady-state error.
   - The trajectory to follow and the setpoints to be considered for position & orientation at each waypoint is defined into a dict structure



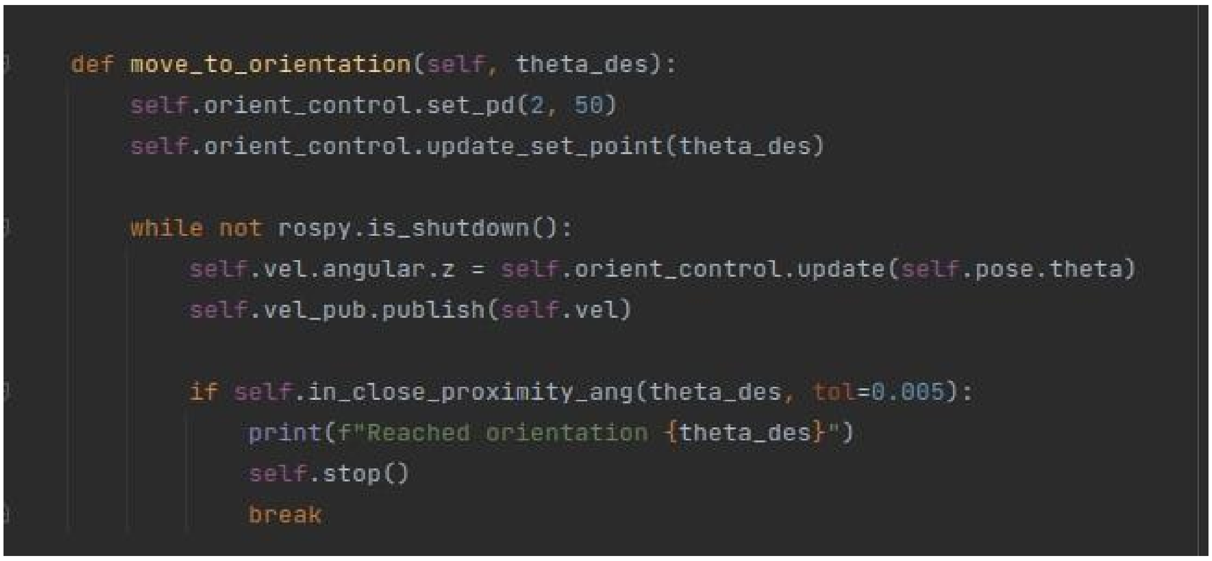
```python
MIN_LINEAR_VEL = 0.25
path = {
    (4, 0): {"pos_set_pt": 4, "orient_set_pt": 0.0},
    (4, 4): {"pos_set_pt": 4, "orient_set_pt": pi/2},
    (0, 4): {"pos_set_pt": 0, "orient_set_pt": pi},
    (0, 0): {"pos_set_pt": 0, "orient_set_pt": -pi/2},
}
```

3. A generic function for navigating to a waypoint that takes in the waypoint alongwith the proportional gain for position control and the proportional and derivative gains for orientation control is defined. It respects the minimum linear velocity constraint and causes the turtlebot to stop when in proximity to the waypoint with a specific tolerance(0.1)



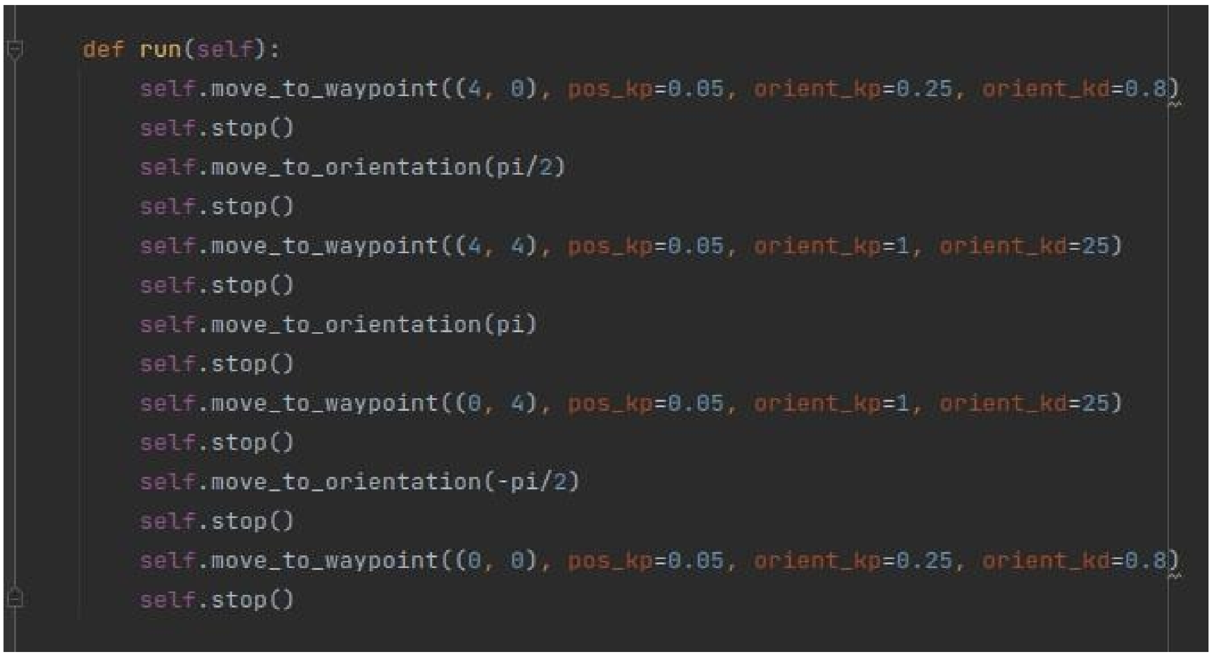
```python
def move_to_waypoint(self, waypoint, pos_kp, orient_kp, orient_kd):
    self.pos_control.set_pd(pos_kp, 0)
    self.pos_control.update_set_point(path[waypoint]["pos_set_pt"])

    self.orient_control.set_pd(orient_kp, orient_kd)
    self.orient_control.update_set_point(path[waypoint]["orient_set_pt"])

    while not rospy.is_shutdown():
        ang_vel_z = self.orient_control.update(self.pose.theta)

        lin_vel_x = self.pos_control.update(self.pose.x)
        if lin_vel_x < MIN_LINEAR_VEL:
            lin_vel_x = MIN_LINEAR_VEL

        self.vel.linear.x = lin_vel_x
        self.vel.angular.z = ang_vel_z

        self.vel_pub.publish(self.vel)

        if self.in_close_proximity_pos(waypoint, tol=0.1):
            print(f"Reached waypoint {waypoint}")
            self.stop()
            break
```

4. Likewise, a similar function to rotate in place to the desired orientation is defined. This just needs orientation control and stops the turtlebot when in proximity to the desired orientation with a tolerance of 0.005.



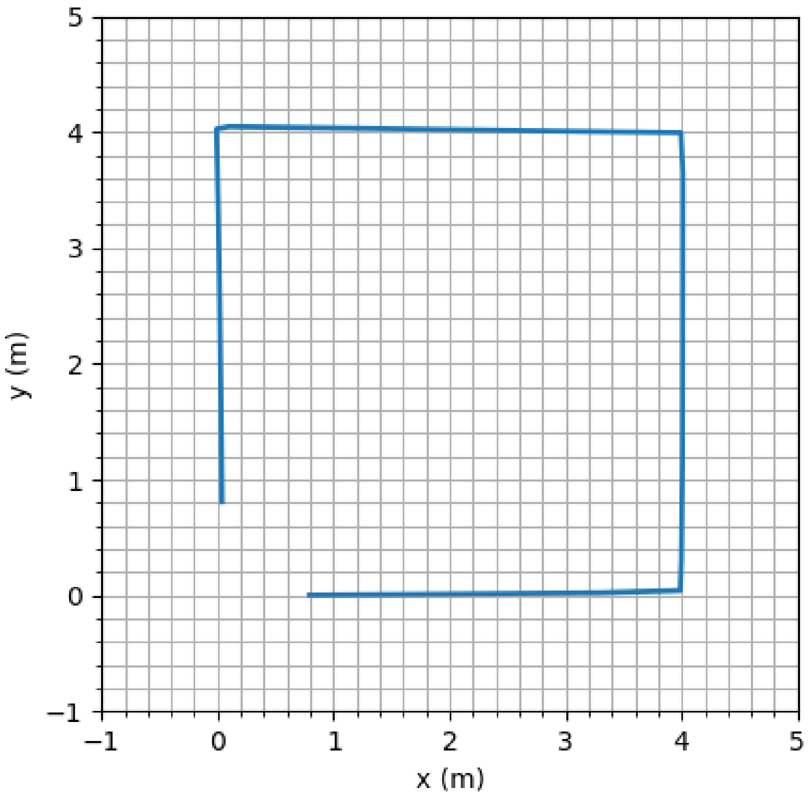
```python
def move_to_orientation(self, theta_des):
    self.orient_control.set_pd(2, 50)
    self.orient_control.update_set_point(theta_des)

    while not rospy.is_shutdown():
        self.vel.angular.z = self.orient_control.update(self.pose.theta)
        self.vel_pub.publish(self.vel)

        if self.in_close_proximity_ang(theta_des, tol=0.005):
            print(f"Reached orientation {theta_des}")
            self.stop()
            break
```

5. Finally, everything comes together in the ‘run()’ function, where the above two functions are called repeatedly with appropriate position, orientation and controller gains as input, to have the turtlebot navigate the expected trajectory.



```python
def run(self):
    self.move_to_waypoint((4, 0), pos_kp=0.05, orient_kp=0.25, orient_kd=0.8)
    self.stop()
    self.move_to_orientation(pi/2)
    self.stop()
    self.move_to_waypoint((4, 4), pos_kp=0.05, orient_kp=1, orient_kd=25)
    self.stop()
    self.move_to_orientation(pi)
    self.stop()
    self.move_to_waypoint((0, 4), pos_kp=0.05, orient_kp=1, orient_kd=25)
    self.stop()
    self.move_to_orientation(-pi/2)
    self.stop()
    self.move_to_waypoint((0, 0), pos_kp=0.05, orient_kp=0.25, orient_kd=0.8)
    self.stop()
```

Trajectory



Effect of controller gains

The thought process followed to set the Kp, Kd values was quite simplistic. It captures the effect of varying the controller gains:

Navigating to waypoint

- Set Kp value for position control (started with a small value of 0.1)
- Observe the effect on the speed of the turtlebot.
- If the speed is reasonable, retain the Kp value or adjust accordingly.
- Set minimum velocity to deal with the steady state error in position.
- At this point, for orientation control, set some Kp value to deal with the drift in the orientation of the turtlebot.
- Observe the effect on the orientation of the turtlebot. If the Kp value is too small, it rotates very slowly and leads to a considerable drift from the expected trajectory. Kp value corresponding to an appropriate speed is set so that the turtlebot follows the trajectory closely.

Rotating to orientation

- Some Kp value is set for orientation control of the turtlebot.
- If the Kp value is too small, it rotates very slowly. Kp value corresponding to an acceptable speed is set.
- Towards the end, the control input becomes very small and so it takes a lot of time to reach the desired orientation, so the Kd value is now adjusted so that the desired orientation is reached quickly.