AARHUS UNIVERSITET

Department of Electrical and Computer Engineering

Assignment 1

Implementing robot circling with RL and measuring individual vs. group performance

Group 4

Alba Arcos I Ribas
Student no. 202503224

Birnir Þór Árnason
Student no. 202503220

Martin Skelde Krøjmand
Student no. 201710685

Wouter Jonathan Oudijk
Student no. 202109059

Zsombor Krisztián Szöke
Student no. 202411164

Teacher: *Mirgita Frasheri and Lukas Esterle*

Number of characters: *3702 (2 standard pages)*

Aarhus, October 6, 2025

Contents

List of Figures

List of Tables

1 Implementation

1.1 Hard-coded solution

To get familiar with the IR-Sim library, we first hard-coded a robot to move in a circle. To do this we first found the source code for the *Dash* behaviour when using *Diff* kinematics[1]. This basic behaviour simply makes the agent move directly towards the goal, slowing down to turn if necessary. Since the agent wants to move directly towards the goal, the desired angle of the agent should be equal to the angle between the goal and the agent. So the DiffDash behaviour starts by getting the angle of the agent and the desired angle (the angle between the agent and the goal). The difference between these angles is calculated and then the agent simply turns to reduce this difference to zero, making it's angle equal to the desired angle.

To make the agent instead move along a circle, we simply add 90 degrees to the desired angle. This makes the agent move along the circle tangent and assuming infinetly small steps, this would result in a perfect circle. However since we don't have infinetly small steps, the agent will move further and further away from the circle with each step. To account for this, we simply take the distance from the circle center and add this to the equation, increasing the angle when the distance is less than the desired radius and decreasing the angle when the distance is more than the desired radius. This results in the code seen in listing 1.1.

```python
# Get the distance and the angle to the goal (Circle center):
distance, radian = relative_position(state, goal)

# Calculate the distance correction amount:
distanceCorrection = (circle_radius - distance) * correction_multiplier

# Add 90 degress + the distance correction to the desired angle
radian += np.pi/2 + distanceCorrection

# Calculate the difference between the agents angle and the desired angle
diff_radian = WrapToPi(radian - state[2, 0])
```

Listing 1.1: Implementation of getting the desired angle for the circle following agent

1.2 Reinforcement Learning

Having gotten a deeper understanding of the IR-Sim library, we started implementing the circle following behaviour using reinforcement learning. In our group we implemented RL in 3 slightly different ways, which allows us to later compare the different solutions, including the hard-coded solution, and determine which is the best for creating circle following agents.

1.2.1 Alba

1.2.2 Krisztián

1.2.3 Wouter Jonathan

1.3 Subsumption Architecture and Obstacle Avoidance

2 Testing methodology

To measure the quality of our solutions, we measure based on the following metrics:

1. **Radius Error**: Every simulated step, we add the current distance between the agent and the desired circle radius, adding together the error of all the agents to get one value per solution. A lower value would equal the agent having spent more time closer to the circle it is meant to follow.
2. **Angle Error**: Every simulated step, we add the amount the desired angle for the agent has changed, adding together the error of all the agents to get one value per solution. While following a circle, the angle should stay almost constant, so a lower value would equal the agent having followed a smooth path.
3. **Speed**: The amount of time the agents on average spend doing one lap around the circle. A higher speed is better.

And to measure the quality of the solutions using the subsumption architecture, we also add the following metrics:

4. **Collisions**: The amount of agents that collide.

3 Results

First we will go through every metric, showing a graph of how the different solutions compare on the specific metric throughout the simulation time. In section 4 *Discussion* we will collect the results in table 4.1 and discuss our findings.

3.1 Radius Error

3.2 Angle Error

3.3 Speed

3.4 Collisions

4 Discussion

Solutions	Radius Error	Angle Error	Speed	Collisions
Hard-coded				
Alba				
Krisztián				
Wouter Jonathan				
Subsumption				

Table 4.1: Overview of all the test results

5 Conclusion

Bibliography

Websites

[1] Hanruihua. *IR-SIM DiffDash documentation.* Last visited: 02-10-2024. URL: `https://ir-sim.readthedocs.io/en/stable/_modules/irsim/lib/behavior/behavior_methods.html#DiffDash`.