UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MULTI-ROBOT SYSTEMS

Controlling a Swarm of Sphero Robots in a Simulator Stage using Reynolds Rules – Experimental Phase

Drmić Jakov, Jurković Marin, Kozulić Josip Ante,
Pušnik Filip, Škoro Filip, Vuksan Ivan

Zagreb, February 2023.

Contents

1. Introduction

The goal of this project was to test algorithms based on Reynolds rules to control Sphero robots in laboratory using real Sphero robots and also to implement another algorithm; consensus-based formation control strategy. The results of the experiment will be shown and explained later in the report.

1.1 Equipment

Figure 1: A Sphero SPRK+ model and components

Sure, the Sphero SPRK+ robot is a small, spherical robot that is made of transparent plastic. It has a diameter of 7.5 cm and contains all of the mechanical and electrical components necessary for its operation. The robot moves using two independently controlled wheels located at the bottom of the robot, which come in contact with the inside of the housing.

To ensure the stability of the robot, the center of gravity is set low using the battery, the motor, and a small weight. The robot is equipped with a gyroscope and an accelerometer, which allow it to always know its orientation and use this data to maintain stability and smooth steering.

The robot can communicate with a mobile application or a computer via Bluetooth 4.0 Low Energy protocol, which allows it to always be ready to connect and receive commands while consuming little energy in the passive state.

The robot is also equipped with two RGB LEDs that can be programmed to provide information about the status of the robot, such as battery status, and a blue LED that indicates the rear end of the robot, which facilitates the change of orientation.

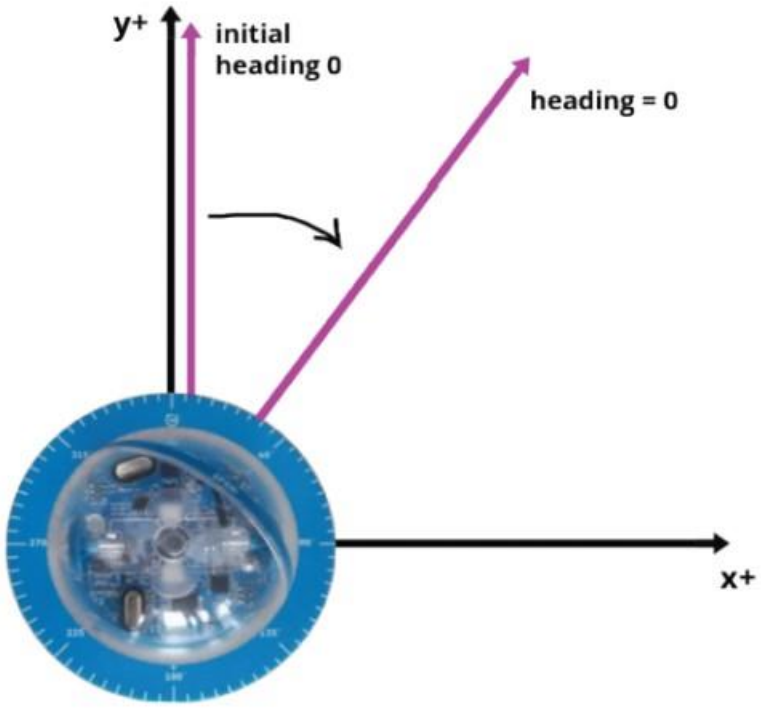

Figure 2: A Sphero SPRK+ in coordinate system displaying drift over time

The robot uses its own coordinate system as shown in Figure 2, which it tries to maintain while moving. However, due to the imperfection of the sensor, the coordinate system may drift with respect to the initial state after some time, especially due to collisions with obstacles, so recalibration is required occasionally.

The robot can also report its position, speed, and acceleration to the user, but these are odometric values which are subject to errors.

1.2. Web cameras

Within this project, we used Logitech c270 web cameras shown in Figure 3 to localize the robots in a fixed world frame. Two cameras are used to cover an area of 210 x 295 centimeters.

Figure 3: Logitech C270 web camera

2. Assignment

Assignment has two main parts. In the first part of assignment our task was to test five Reynolds rules implemented in the first part of the project: separation, alignment, cohesion, navigation and obstacle avoidance in FER laboratory. The goal of the second part was to implement another algorithm; consensus-based formation control strategy and also to test it in laboratory using real Sphero robots.

2.1. Consensus-based formation control strategy

The goal was to use 3 Sphero robots to form triangle, line and column formations as shown in Figure 4 to move from one point in the space to another. The idea was to move the reference Sphero robot, and keep the other Sphero robots at a constant distance from it. Changing the direction and speed of movement of the reference robot changes the direction and speed of movement of the other robots with the purpose of preserving the given formation.

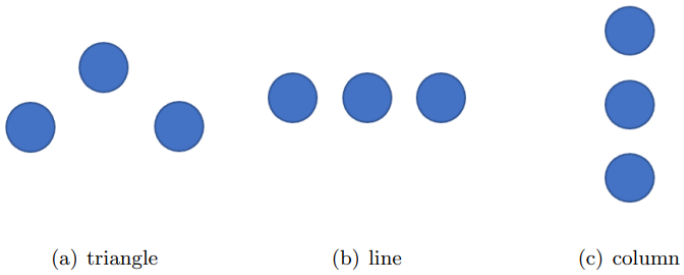

Figure 4: Different Sphero robot formations

2.2. Implementation of basic and additional Reynolds rules

This subchapter describes implemented algorithms.

If current boid has neighbors than it is possible to proceed with algorithms. Main part of separation algorithm is implemented in create_separation_force function which gets current boid position and neighbor boid position as inputs. Function calculates separation force by subtracting boid's positions which is then weighted based on squared distance. Function is shown in figure (Figure 6).

```
68  def create_separation_force(current_robot_position, neighbor_robot_position):
69      """
70      Method that creates force in opposite direction. Force is weighted based on distance. A closer robot has more impact on separation force.
71      :param current_robot_position: TIP, position of current robot
72      :param neighbor_robot_position: TIP, position of other robot in the world
73      return: list, weighted force vector
74      """
75
76      [x1, y1] = current_robot_position
77      [x2, y2] = neighbor_robot_position
78
79      # Compute distance current and other robot
80      distance = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
81
82      # Create weighted force by substracting position vectors
83      force = current_robot_position - neighbor_robot_position
84      force = force/np.linalg.norm(force) / (distance ** 2)
85      # force = -np.array([0 if not (x2-x1) else np.sign(x2-x1)/(x2-x1)**2, 0 if not (y2-y1) else np.sign(y2-y1)/(y2-y1)**2])
86
87      return force
```

Figure 6 Separation force function

Cohesion algorithm is implemented in function *create_cohesion_force*. It gets current boid position and position of local center as inputs. Local center is center of mass calculated by averaging sum of boid's neighbors positions. Cohesion force is calculated by subtracting current robot position from local center. Function is shown in figure (Figure 7).

```
55  def create_cohesion_force(current_robot_position, local_center_position):
56      """
57      Method that creates cohesion force to attract robots to form a group
58      :param current_robot_position: TIP, position of current robot
59      :param neighbor_robot_position: TIP, local center position
60      return: list, cohesion force vector
61      """
62
63      # Create cohesion force by substracting position vectors
64      force = local_center_position - current_robot_position
65
66      return force
```

Figure 7 Cohesion force function

Alignment algorithm is implemented in function *create_alignment_force*. This function gets current boid velocity and local center velocity as inputs where local center velocity is averaging sum of boid's neighbors velocities. Alignment force is calculated by subtracting current robot velocity from local center velocity. Function is shown in figure (Figure 8).

```
89    def create_alignment_force(current_robot_velocity, local_center_velocity):
90        """
91        Method that creates alignment force to attract robots to form a group
92        :param current_robot_velocity: TIP, position of current robot
93        :param neighbor_robot_velocity: TIP, local center position
94        return: list, alignment force vector
95        """
96
97        # Create alignment force by substracting position vectors
98        force = local_center_velocity - current_robot_velocity
99
100       return force
```

Figure 8 Alignment force function

Navigation to a certain point is achieved by create_navigation_force function. This function gets input arguments current boid position and goal position where goal position is desired boid's position. It calculates force by subtracting current boid position from goal position. Function is shown in figure (Figure 9).

```
102 def create_navigation_force(current_robot_position, goal_position):
103     """
104     Method that creates force in opposite direction. Force is weighted based on distance. A closer robot has more impact on separation force.
105     :param current_robot_position: TIP, position of current robot
106     :param neighbor_robot_position: TIP, position of other robot in the world
107     return: list, weighted force vector
108     """
109     # rospy.logwarn(goal_position)
110     # rospy.logerr(current_robot_position)
111     [x1, y1] = current_robot_position
112     [x2, y2] = goal_position.x, goal_position.y
113
114     # Compute distance current and other robot
115     # distance = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
116
117     # Create weighted force by substracting position vectors
118     force = [x2, y2] - current_robot_position
119     # force = force / (distance ** 2)
120     # force = -np.array([0 if not (x2-x1) else np.sign(x2-x1)/(x2-x1)**2, 0 if not (y2-y1) else np.sign(y2-y1)/(y2-y1)**2])
121
122     return force
```

Figure 9 Navigation force function

3. Simulation

This subchapter shows our results of simulation. As mentioned earlier, simulations of algorithms were carried out in FER laboratory. This subchapter shows basic Reynolds rules implemented using three Sphero robots.

After implementing the algorithms, it was necessary to localize all the Sphero robots in the given space. This was performed using already mentioned cameras and an existing script.

The Video shows basic Reynolds rules implemented using three Sphero robots. The implemented rules are cohesion, separation, alignment and navigation. The desired behavior was achieved.

4. Conclusion

As was stated before, the task was to perform an experiment on the implemented algorithms that gave good results during the simulation in the 2D Stage simulator. The experiment was conducted using three Sphero robots in the FER laboratory. The other task was to implement and test new algorithm; consensus-based formation control strategy.

It can be seen that the implemented algorithms work well on real Sphero robots. The obstacle avoidance algorithm was not tested due to technical difficulties with obstacle visualization.

The idea of consensus-based algorithm was explained earlier in the text. Due to some technical problems desired result wasn't achievable.

After the performed simulations and presented results, it can be said that implemented algorithms work as expected.