Virtual Force-driven Coachbot Swarm Navigation around a Nucleus Point

Oscar Poudel
op72@njit.edu

INTRODUCTION

Rotation around a nucleus point characterizes several biological and ecological phenomena in natural systems. Flocks of birds circle around a predator with the intent of deterring its attack, fish schools form in circular formations as a way of defense, and celestial bodies orbit around gravitational centers (Satz, H. 2020). These natural examples of coordinated, nucleus-centered motion provide motivation for robotic swarms to similarly emulate such behaviors, especially when organized and predictable motion patterns are called for in applications such as monitoring, patrolling, or resource allocation. Swarm robotics will be explored in this project concerning decentralizing coordination to enable a swarm of robots to move as a cohesive unit around a fixed nucleus point. Circular, cohesion, and separation forces are combined to develop a system that can enable collision-free, synchronized motion. It would play a critical role in tasks like perimeter surveillance where such motion is required in similar kinds of trajectories over ground to air. This project utilize Coachbot; a swarm robotics testing platform developed at Northwestern University(Vaishnavi et al. 2025). The proposed approach incorporates circular motion swarm behavior inspired by natural rotational dynamics into decentralized communication protocols. Research has also shown that circular motion increases stability in swarm robotics (Hewahi and Almobayed 2017). The work of Wang and Rubenstein (2020) on decentralized swarm algorithms for shape formation has revealed how local task swapping and grid-based motion planning enable reliable formation without requiring centralized supervision. Extending such principles to circular motion introduces new possibilities for dynamic and adaptive swarm behavior, with a wide range of applicability in real-world scenarios. These integrated concepts provide, in this project, a basis for the realization of efficient and scalable robot swarm systems that will robustly execute tasks around a nucleus point. Potential future extensions could be to further optimizations for high-density swarm and testing on different sets of robots (UAV's).

METHODOLOGY

The subsequent section discusses the methodology adopted for this project. The framework for the code is adapted from example given for the Coachbot swarm system GitHub repo.

(a) Algorithm for Swarm System:

The algorithm implements virtual force-based setup with cohesion and separation forces. The backend system (Coachbot swarm platform) handles the message and communication part.

Intialization: 'central nucleus' point i.e origin is initalized. The nucleus is the primary focal reference for which these robots conduct their circular motion on. For the robot group to implement this, three parameters- circular force, cohesion, and separation are also initialized. The circular force guides robots

tangentially around the nucleus so that they will follow a circular trajectory. The cohesion force encourages the robots to move toward the swarm's center of mass which maintains swarm structural integrity and avoids dispersion. Finally, the separation force allows the robots to avoid collisions by generating a repulsive interaction when they get too close to their neighbors. These are combinedly working to provide that the robots move together with no obstacles. Every robot starts their working or motion with some initial velocity for instantaneous motion. Neighbors dictionary is initialized for real-time position data sent by the neighbors for decentralized decision-making.

```
10        # Define the central point for rotation
11        central_point = [0.0, 0.0]  # Fixed central point at the origin
12
13        CIRCULAR_FORCE = 1.5  # Force driving the robots to circle the central point
14        COHESION_FORCE = 0.5  # Cohesion toward the swarm's center of mass
15        SEPARATION_FORCE = .50  # Separation to avoid collisions with neighbors
16
17        # Start robot movement on startup
18        robot.set_vel(30, 30)
19
20        Neighbors = {}
```

Figure 1: Initialization of the parameters and central location

Pose Acquisition and Communication: Each robot keeps querying its pose continuously. The pose includes the position(x,y) and orientation (heading angle). If it is not available it will skip the current iteration in order to avoid system crash.The robots share their positional data with neighbors through a backend communication system (Coachbot Swarm's API). Positional data received from neighbors is processed and stored in a dictionary, with each entry corresponding to a neighbor's unique ID and current pose.

```
25            # Determine robot's current pose
26            pose = robot.get_pose()
27            if not pose:
28                continue
29
30            # Send robot's current position to neighbors
31            robot.send_msg(struct.pack('fffi', pose[0], pose[1], pose[2], robot.id))
32
33            # Receive messages from neighbors
34            msgs = robot.recv_msg()
35            for msg in msgs:
36                data = struct.unpack('fffi', msg[:16])
37                Neighbors[data[3]] = data[:3]  # Update neighbor data
38
```

Figure 2: Communication and Pose Acquisition for the swarm

Force Calculation and Integration: Three main driving force: circular force, cohesion force, and separating forces are calculated for the determination of robot motion. The circle force(vector) is the tangential force applied toward moving the robot in its circular trajectory. This resulting vector is normalized. The cohesion force is obtained by calculating the center of mass of the swarm based on the positions of the neighboring robots that orients the robot toward the CoM to keep the swarm united. The separation force is calculated based on the threshold distance between the robot and its neighbors. The resultant motion vector combines all the three forces with appropriate weights assigned to each for giving importance to one force over the other. Circular force will have the highest weight because that is the main desired motion. Moderate weight will be given to the cohesion force to stay cohesive as a swarm.

```
        # Compute the circular force
        to_center = [central_point[0] - pose[0], central_point[1] - pose[1]]
        mag_to_center = math.sqrt(to_center[0]**2 + to_center[1]**2)
        if mag_to_center > 0:
            circular_force = [-to_center[1] / mag_to_center, to_center[0] / mag_to_cent
        else:
            circular_force = [0, 0]

        # Compute the cohesion force
        com = [0, 0]  # Center of mass
        for nID in Neighbors:
            neighbor_pose = Neighbors[nID]
            com[0] += neighbor_pose[0]
            com[1] += neighbor_pose[1]
        if Neighbors:
            com[0] /= len(Neighbors)
            com[1] /= len(Neighbors)
            cohesion = [com[0] - pose[0], com[1] - pose[1]]
            mag_cohesion = math.sqrt(cohesion[0]**2 + cohesion[1]**2)
            if mag_cohesion > 0:
                cohesion = [cohesion[0] / mag_cohesion, cohesion[1] / mag_cohesion]
        else:
            cohesion = [0, 0]

        # Compute the separation force
        repulsion = [0, 0]
        for nID in Neighbors:
            neighbor_pose = Neighbors[nID]
            dist = math.sqrt((pose[0] - neighbor_pose[0])**2 + (pose[1] - neighbor_pose
            if dist > 0 and dist < 0.25:  # Avoid neighbors within 25 cm
                repulsion[0] += (pose[0] - neighbor_pose[0]) / dist**2
                repulsion[1] += (pose[1] - neighbor_pose[1]) / dist**2

        # Combine forces
        x = (CIRCULAR_FORCE * circular_force[0] +
             COHESION_FORCE * cohesion[0] +
             SEPARATION_FORCE * repulsion[0])

        y = (CIRCULAR_FORCE * circular_force[1] +
             COHESION_FORCE * cohesion[1] +
             SEPARATION_FORCE * repulsion[1])
```

Figure 3: Force calculation and integration

Heading and Motion Calculations: First, the robot calculates its desired heading angle from the resultant motion vector and computes the angular difference between its current and desired headings. After obtaining this difference, the algorithm dynamically adjusts the wheel velocities so that the robot gets oriented along the desired direction with smooth movements. The sharper the turn, the higher the speed of one side, while the speed of the other wheel is reduced proportionally. The changes are then applied as the robot refreshes its wheel velocities, and performs the movement.

```python
        # Calculate the desired heading
        heading = math.atan2(y, x)

        # Update robot's motion
        try:
            current_heading = pose[2]
            distR = math.fmod(abs(current_heading - heading + 2 * math.pi), 2 * math.pi)
            distL = math.fmod(abs(2 * math.pi - (current_heading - heading)), 2 * math.pi)

            fastWheel = 30
            slowWheel = max(15, -30 / math.pi * min(distL, distR) + 30)

            if distR <= distL:
                velR = slowWheel
                velL = fastWheel
            else:
                velR = fastWheel
                velL = slowWheel

            robot.set_vel(velL, velR)
```

Figure 4: Next step Heading and motion calculation

(b) Running simulator and submission

The simulator was setup as per the instructions in the GitHub page. The simulator was developed in python 3 but the instructions said to install python 2.7 which caused some issue with the inital setup.

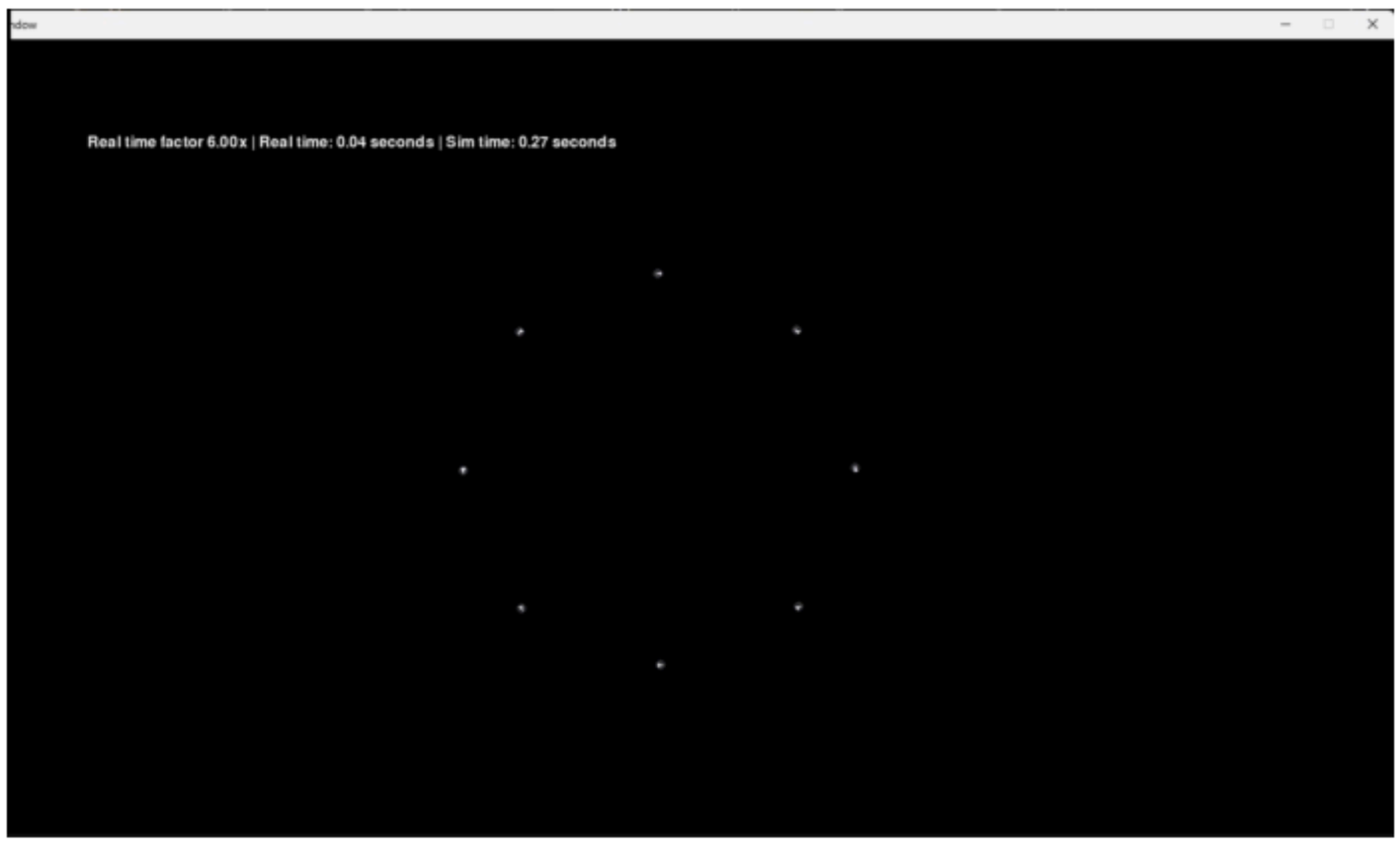


Figure 5: Simulator and initial position of robots for Coachbot swarm system

The initial positon of the robot was set to be at a circumference of 0.75 m from the origin (Nucleus) of the swarm. After the code ran in the simulator the validity was checked and updated to the github submission page of the simulator.

```
(base) PS G:\My Drive\classes\robotics_extra\coachbot-swarm-system-submission-dornadulavaishnavi> python .\input_validity_test_main.py
nitial file loaded
sr_code.py successfully compiled
ending Emails to: ['op72@njit.edu']
'0': [0.0, 0.5, 0.0, 1.57], '1': [1.0, 0.353, 0.353, 2.356], '2': [2.0, 0.0, 0.5, 3.141], '3': [3.0, -0.353, 0.353, -2.356], '4': [4.0, -0.5,
 [6.0, 0.0, -0.5, 0.0], '7': [7.0, 0.353, -0.353, 0.785]}


0.0, 0.5, 0.0, 1.57]
1.0, 0.353, 0.353, 2.356]
2.0, 0.0, 0.5, 3.141]
3.0, -0.353, 0.353, 3.9271853071795864]
4.0, -0.5, 0.0, 4.713185307179586]
5.0, -0.353, -0.353, 5.498185307179586]
6.0, 0.0, -0.5, 0.0]
7.0, 0.353, -0.353, 0.785]

iles Loaded Correctly: True Valid initial positions given True
rue
assed Autograder Test
```

```
(base) PS G:\My Drive\classes\robotics_extra\oscarpoudel> git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 1.72 KiB | 176.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Coachbot-Swarm/oscarpoudel.git
   3cde3da..6dedf2e  main -> main
```

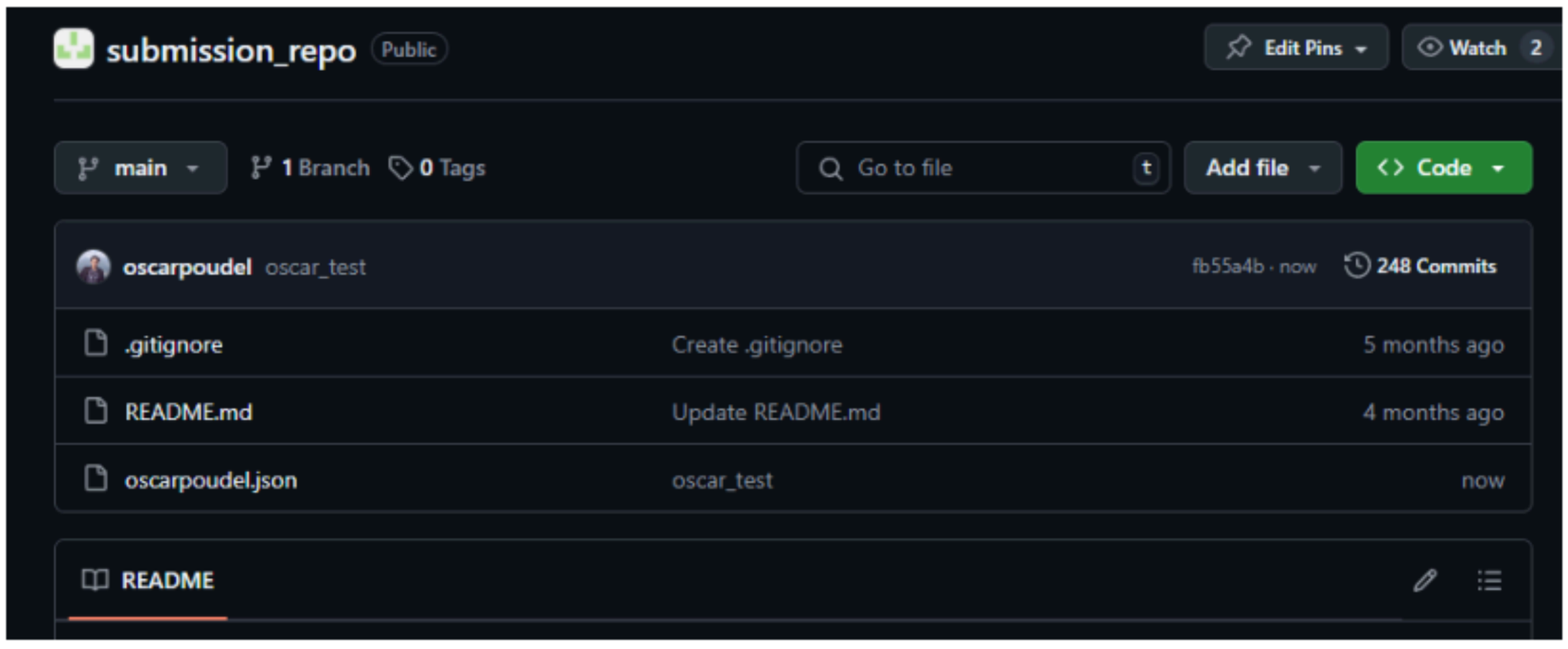
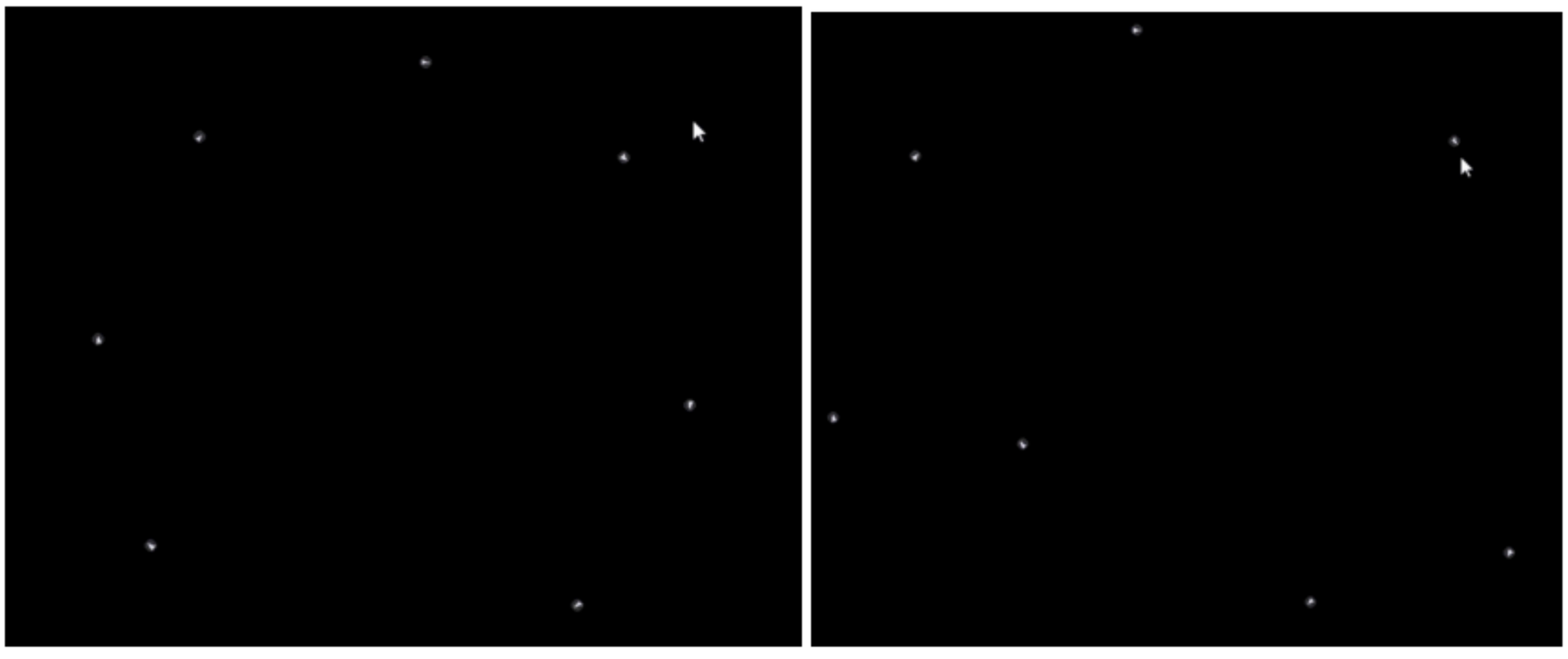


Figure 6: checking the validity of the algorithm and updating it to the submission repo for simulation.

The final simulation was updated in the same repository where the simulation code was update.

RESULT AND DISCUISSION

(a) **(b)**



(c) (d)

Figure 6: Running the algorithm in the simulator

As shown in Figure 6, (a), the initial position of the robot swarm (8 robots used for the experiment) is aligned in the circumference of 0.75m from the nucleus, in (b) the cohesion force acts and the robots come together and also keep on rotating (constant circular force). After they are close enough (as per the threshold) separation force (Figure 6(c)) acts and they are separated again for collision-free motion and rotated around the nucleus at a certain distance. Figure 6(d) shows the final result after the separation force is accounted for.

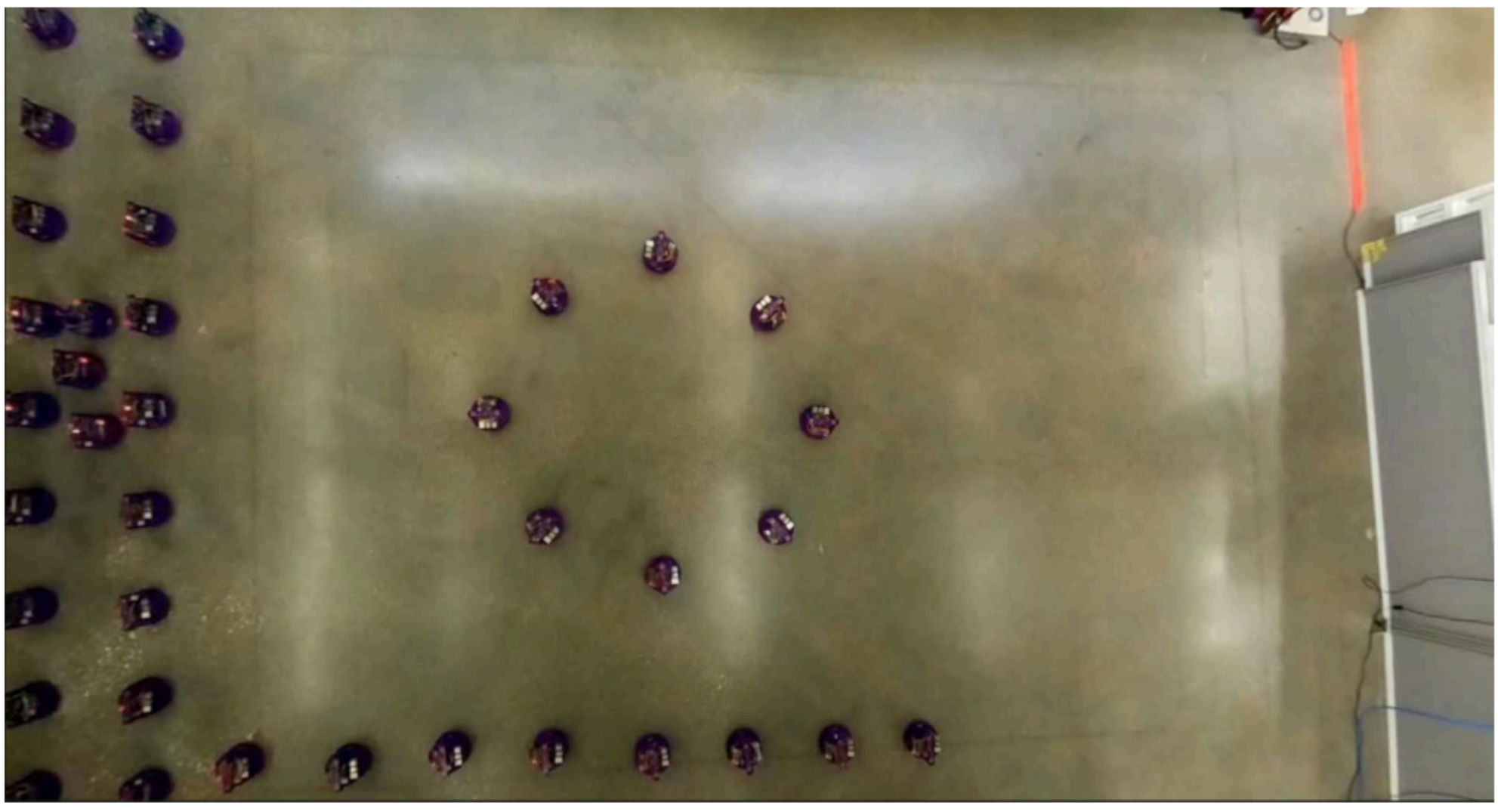


Figure 7: Sim2real Initial position of the robot system

Figure 8: Cohesion being applied in the beginning due to which they come together

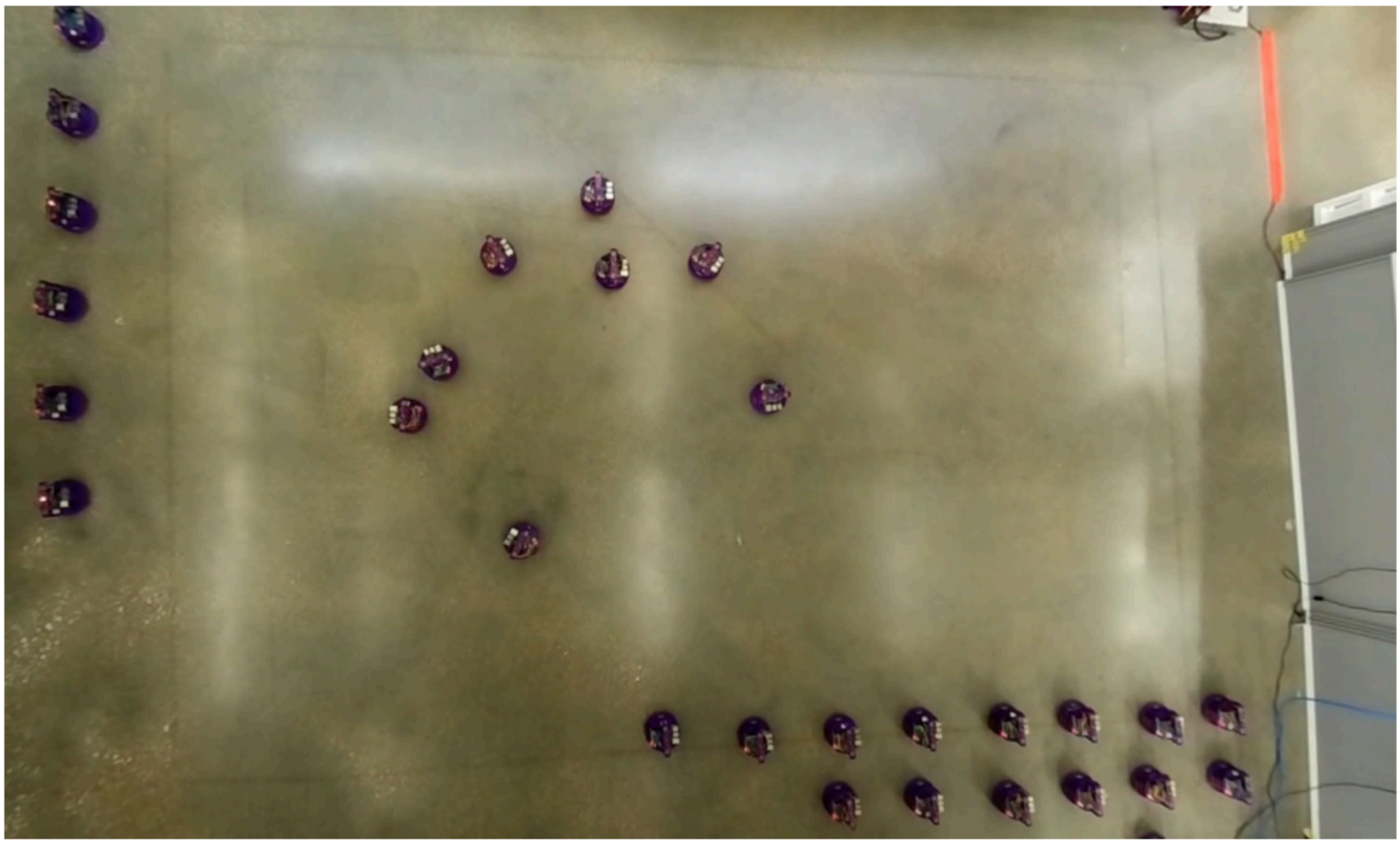


Figure 9: Separation making the robots move away from each other

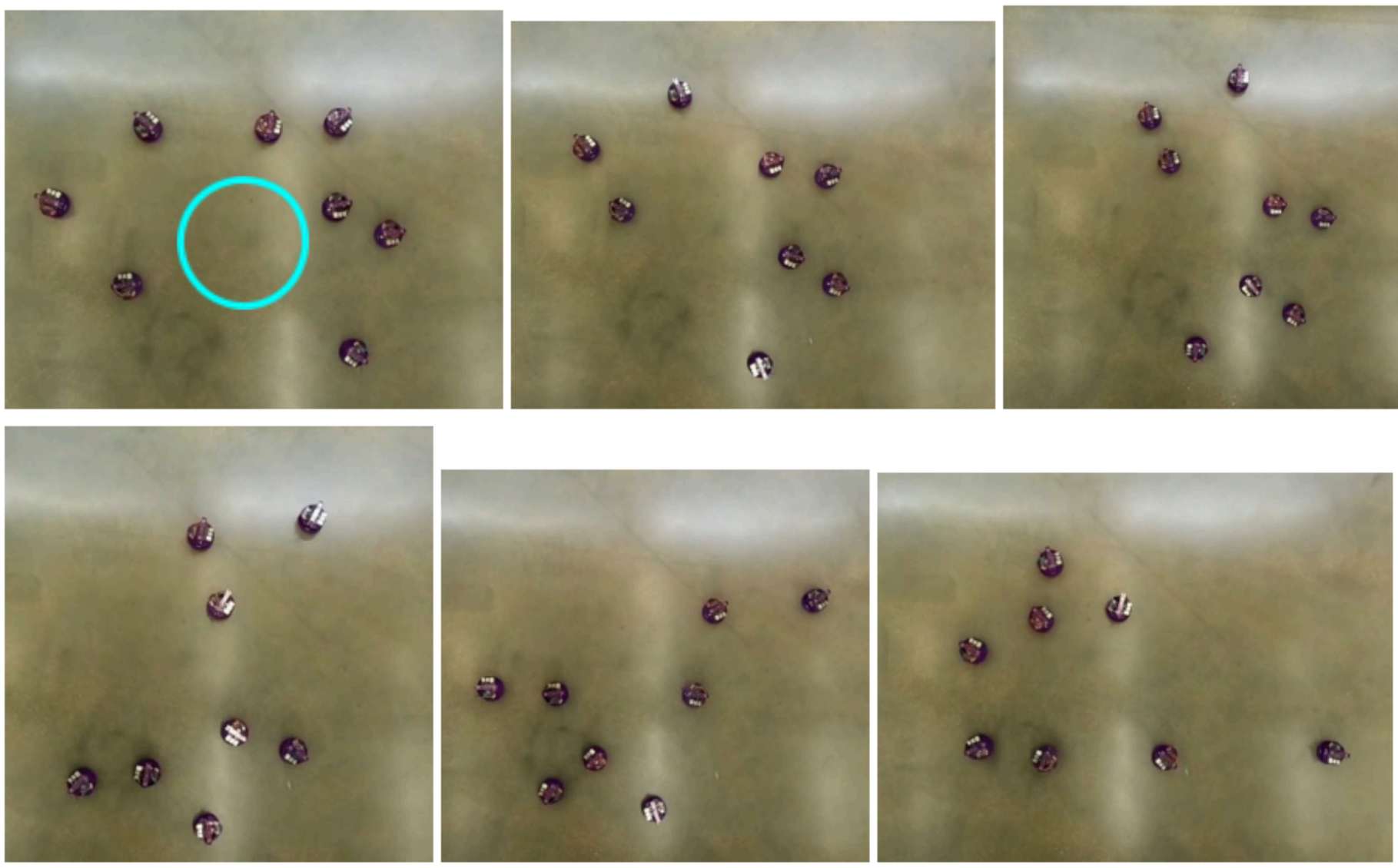


Figure 9: Shows circular motion around a center point (Nucleus) with cohesion and separation force

After it ran successfully in the simulation the algorithm was tested with the real robots as well. Figure 7 shows the initial position of the robots which validates the starting position of the simulation. Figure 8 and 9 verify the cohesion and separation phenomena. Figure 9 shows the snippets of the robots flocking in a circular trajectory around a center point maintaining certain distance with the other robots. The simulation stopped after one of the robots went out of bounds from the predefined limits which shows that the algorithm still needs some optimization.

CONCLUSION

The project proves that a decentralized, force-based approach to swarm coordination is feasible and robust for swarm applications involving organized motion around a nucleus point. Successful results show the algorithm maintaining cohesive, collision-free, and dynamic motion in a scalable and adaptive way. Result in simulation and real world testing indicate possibilities of practical use in applications like surveillance, environmental monitoring, and distributed sensing. Hence, this project establishes a basis for future studies on decentralized swarm coordination and evaluates the effectiveness of these systems in dynamic, real-world settings.

References:

- **Vaishnavi Dornadula, C. Lin Liu, Marko Vejnovic, Michelle Zhang and Michael Rubenstein. (in press). Coachbot Swarm Testbed: A 100-Robot Automated and Remotely-Accessible Platform. Springer SPAR March 2025**
- **Satz, H. (2020). The Rules of the Flock: Self-Organization and Swarm Structure in Animal Societies. Oxford University Press.**
- **Wang, H., & Rubenstein, M. (2020). Shape formation in homogeneous swarms using local task swapping. IEEE Transactions on Robotics, 36(3), 597-612.**
- **Hewahi, N. M., & Almobayed, A. A. A. (2017). Swarm Robotics with Circular Formation Motion Including Obstacles Avoidance. BRAIN. Broad Research in Artificial Intelligence and Neuroscience, 8(2), 125-143.**