Development of a Swarming Algorithm Based on Reynolds Rules to control a group of multi-rotor UAVs using ROS

Rafael G. Braga[a 1], Roberto C. da Silva [a] and and Alexandre C. B. Ramos[a]

[a]Institute of Mathematics and Computer Science, University of Itajubá, Itajubá, MG, Brazil

Received on September 30, 2016 / accepted on October 20, 2016

Abstract

The objective of this article is to present a swarming algorithm based on Reynolds flocking rules to drive a group of Unmanned Aerial Vehicles (UAV) together as a coherent swarm. We used quadrotors controlled at the low level by a Pixhawk autopilot board and carrying an embedded Linux board (Raspberry Pi) to execute the control algorithms. Each robot is also equipped with distance sensors used to detect other robots next to itself. The swarming algorithm runs on the ROS platform and was implemented in C++. A simulating environment based on the Gazebo Simulator was prepared to test and evaluate the algorithm in a way as close to the reality as possible.

Keywords: Robot swarm, Unmanned Aerial Vehicles, Reynolds rules, Pixhawk, ROS.

1. Introduction

In recent years UAVs have become very popular because they are easy to build and suitable for many different applications, both civil and military, such as aerial photography, crop dusting and surveillance operations. As the technology to build these small vehicles becomes cheaper it becomes possible to apply a group of UAVs to perform a mission instead of only one unit. This aproach has a number of advantages: a group of robots is able to acquire more data from sensors and cover a bigger area than a single one; the group becomes more aware of its surroundings and is able to more effectively defend itself from threats; and even if some of the robots are lost the mission can be carried on by the remaining ones, thus the system becomes more robust.

To design strategies to control groups of agents researchers frequently seek inspiration in nature [1] [2]. Many kinds of animals show collective

[1]E-mail Corresponding Author: fael_gb@yahoo.com.br

behavior such as flocks of birds, schools of fishes, swarms of insects and herds of land animals. In these examples we observe that although each individual animal acts independently from the others, following its own rules, a group behavior emerges, and the swarm moves as there was a central control [1].

In 1987 Craig Reynolds was the first to propose a set of behavioral rules to simulate the movement of a flock of birds [3]. He implemented a computer program in which entities called boids sense their environment and move following these rules, without need of a central control. The resulting movement looked very close to real bird flocks. Since then, many researchers have tried to use Reynolds rules in the area of swarm robotics to drive groups of robots.

Hauert et al. used Reynolds rules to create a flock of 10 fixed-wing UAVs both in simulation and reality [4]. However, their UAVs flew at different altitudes, so they didn't have to avoid collisions with each other. In [5] a flock of 3 UAVs flew together, but again, in different altitudes.

Kushleyev et al. developed an impressive flock of 20 minature quadrotors capable of assuming many different formations, using a centralized control strategy [6]. Bürkle et al. created an outdoor quadcopter swarm also using central processing at a ground station [7]. However central control is something that do not exist in a real flock of birds and for this reason other works distributed the control, programming each UAV to make its own decisions. A remarkable example is [2], where Vsrhelyi et al. flew a flock of 10 quadcopters in an outdoor environment using Reynolds rules. An application of flocking UAVs can be seen in [8], where a group of simulated UAVs is used to monitor an area.

All these works however rely on communication between robots to create the flocking behavior. This is something that we also do not observe in nature, since animals detect each other through vision or other types of sensing. Moeslinger et al. attacked this issue, creating a swarm of robots that detect each other using infrared sensors [1].

Based on these previous work, our goal is to develop a swarming algorithm using Reynolds flocking rules to drive a group of quadrotor UAVs together while avoiding collisions with each other. To better simulate the behavior of real groups of animals, our robots use sensors to detect neighbors instead of communicating their positions.

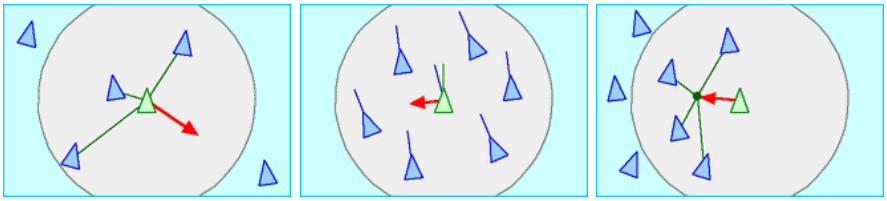
2. Materials and Methods

2.1. Flocking Rules

Here we describe the flocking rules proposed by Craig Reynolds. Figure

1 illustrates how each of them affects robot movement.

- Separation: robots try to move away from other robots to avoid collisions
- Alignment: robots try to match their heading with other robots
- Cohesion: robots try to move closer to other robots to form a swarm



(a) Separation (b) Alignment (c) Cohesion

Figure 1 - Reynolds Flocking Rules [3].

These three rules alone are sufficient to make the robots group and move randomly together. But since we want to control the swarm, we incorporated a fourth rule, called Migration. This rule lets us set a destiny location we call migration point to where the robots try to move.

2.2 Robots

Our flying robots are small quadrotors with 250mm diameter controlled by an open source flight controller board called Pixhawk [9]. The Pixhawk is equipped with 3D accelerometers, Gyroscope and Barometer, and also connected to an external GPS module. Its software is able to obtain the robot's position and pose from these sensors and automatically stabilize it in the air and hold altitude. Figure 2 is a photo of one of our robots.

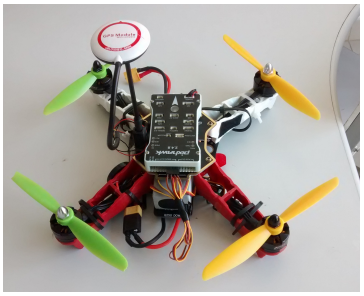


Figure 2 - UAV used in the laboratory.

Originally the Pixhawk is intended to be controlled by a human operator via radio controller or receive commands from a ground station. However, it is also able to communicate with other devices via a protocol called MAVLink. We use this protocol to send commands from an embedded Linux computer (Raspberry Pi) which is also carried by the UAV. This way we can program the UAV to fly autonomously, while still being able to regain manual control at any time.

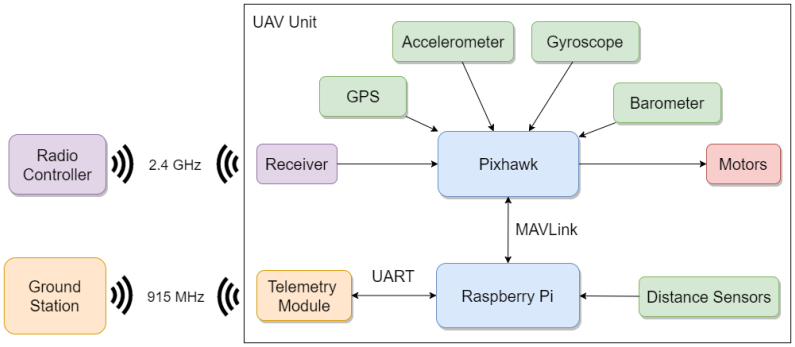


Figure 3 - UAV components.

The robots are also equipped with a telemetry module, to send flight information to the ground station, and four distance sensors, used to detect other UAVs during flight. Figure 3 shows a schematic view of the UAV components.

2.3. Mission Description and modeling

Our system consists of a group of robots that fly over an area and a ground station with which they communicate through a telemetry link. The ground station is responsible for collecting flight data from each robot, but not controlling them. The control algorithm is distributed and runs in each UAV. The objective is to move the swarm to a destination point, using the flocking rules to avoid collision and optimize movement.

Each robot is able to move with 6 degrees of freedom (x, y, z, roll, pitch and yaw axis, illustrated in Figure 4). However, to simplify the control strategy we fix the altitude to a predetermined value and consider angular changes only the heading (Yaw angle), what is valid for most surveillance and search and rescue applications. Each robot is able to localize itself using its GPS and embedded sensors, and other robots using its distance sensors.

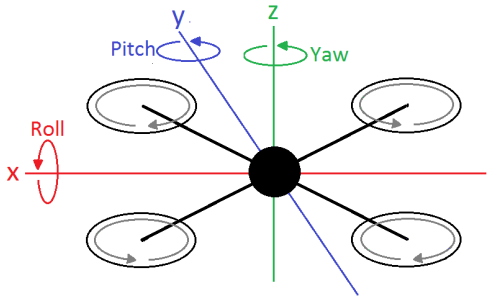


Figure 4 - Degrees of freedom achieved by the quadrotor and rotation direction of the motors.

The destination, or migration point as we call it, is given by the coordinates X_m and Y_m. As we will see, the robots use this information and the positions of the detected neighbors to calculate which direction they should move to obtain the swarming behaviour.

3. Results and Discussion

3.1. System Architecture

Our algorithm was implemented in C++ and runs on the ROS platform. ROS (Robot Operational System) is an open source technology created to aid researchers in developing robotic applications [10]. ROS provides us with many tools and facilities that were very useful in our work.

A ROS application is a network of nodes that communicate with each other. Each node is an independent program in the system and a Master node also exists to manage the network. Nodes that generate data publish this information in topics in the form of messages, while nodes that need that data subscribe to the corresponding topics and receive the published messages.

In our application the master node runs in the ground station. Each robot runs its own instance of the control node, where our swarming algorithm is implemented. The control node gets the data from the distance sensors and uses the Reynolds rules to calculate the direction the robot should move. It sends this information to another node called mavros that create a MAVLink message and send it over a serial connection to the Pixhawk. Figure 5 illustrates the system architecture.

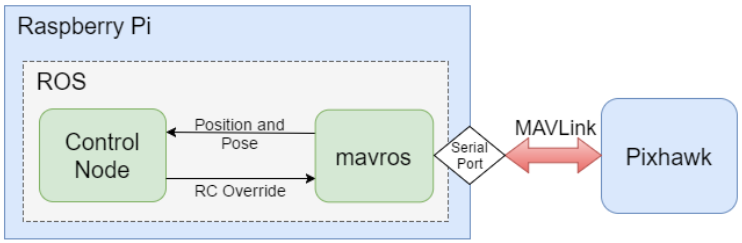


Figure 5 - System architecture overview.

3.2. Algorithm Implementation

The algorithm runs on the embedded Linux computer on each robot. The goal is to detect nearby robots and obstacles and then use the flocking rules to calculate the direction the UAV should move to form a swarm with its neighbors.

Algorithm 1 Main loop of the swarming algorithm

```
1: procedure SWARM
2:     v1 ← separation()
3:     v2 ← alignment()
4:     v3 ← cohesion()
5:     v4 ← migration()
6:     vres ← r1 * v1 + r2 * v2 + r3 * v3 + r4 * v4
7:     move(vres)
8: end procedure
```

First the program creates four vectors which will be used to represent the influence of each rule in the robot's movement. Each of the swarming rules is implemented in a separate function. In line 6 the four vectors are combined through a weighted sum to generate the final resulting direction the robot should move. The weight applied to each rule determines how much that rule influences in the final result. The values of the weights can be changed to obtain different results. Rules can be even completely removed from the calculation by setting its weight to zero. After some testing we obtained good results using the following values:

$$\text{R1} = 1;\ \text{R2} = 1.5;\ \text{R3} = 1;\ \text{R4} = 1;$$

In line 7 the calculated resulting direction is passed to another function to move the UAV. This function uses the mavros package functions to send MAVLink messages to the autopilot, driving it to the correct direction.

Separation Rule: This rule tries to move the robot away from other robots to avoid collisions. This is done creating a vector for each one of the detected neighbors pointing to the exactly opposite direction of that neighbor. These vectors are then combined into a resulting vector and returned to the main program.

Algorithm 2 Separation Rule

```
1:  procedure SEPARATION
2:      Vector v ← 0
3:      for all neighbors n do
4:          vn ← this.position − n.position
5:          vn.normalize()
6:          vn ← vn ∗ distance(this, n)
7:          v ← v − n
8:      end for
9:      return v
10: end procedure
```

We also want the robot to move faster the closer it is from its neighbor. We do this adjusting the vectors magnitude. First we normalize it so it becomes a unit vector. Then we divide it by the distance between the two robots.

Alignment Rule: This rule represents an urge to move the robot in the same direction its neighbors are moving. We loop trough each detected neighbor and calculate the average direction they are moving. We then return this vector to the main function.

Algorithm 3 Alignment Rule

```
1: procedure ALIGNMENT
2:     Vector v ← 0
3:     for all neighbors n do
4:         v ← v + n.velocity
5:     end for
6:     v ← v/neighbors.length
7:     return v
8: end procedure
```

Cohesion Rule: This rule tries to move the robot to the center of mass of the other robots, to form a swarm. The implementation is similar to the Alignment rule, but instead of calculating the average direction we calculate the average position of the neighbors.

Algorithm 4 Cohesion Rule

```
1: procedure COHESION
2:     Vector v ← 0
3:     for all neighbors n do
4:         v ← v + n.position
5:     end for
6:     v ← v/neighbors.length
7:     return v
8: end procedure
```

Migration Rule: This is the rule used to move the swarm. Once a migration point is set by the operator, the robots try to move to that point. It is a simple rule that just returns a vector poiting to the migration point.

Algorithm 5 Migration Rule

```
1: procedure MIGRATION
2:     Vector v ← 0
3:     if migrationPoint is set then
4:         v ← migrationPoint − this.position
5:     end if
6:     return v
7: end procedure
```

3.3. Simulation

We decided to run simulations of a group of UAVs using our algorithm as a proof-of-concept and also as a way to evaluate the algorithm's swarming capability. The simulations were run on the Gazebo Simulator [11], a software that is capable of simulating an entire 3D environment as well as robots moving through this environment and the readings of their sensors. Using software provided by the Pixhawk team and the Gazebo team, we were able to simulate quadrotors running the Pixhawk firmware in Gazebo and integrate it with ROS. This way we can test our ROS applications in the simulator as we were running in real UAVs.

Figure 6 shows one simulated UAV, as visible in the simulator. The blue cone in the front of the aircraft represents the range measured by one of the distance sensors. The simulator presents us a visual representation of the UAVs in movement, as we would see in real life. In the future, we plan on collecting and processing the flight data to compare the performance of our implementation to other similar works.

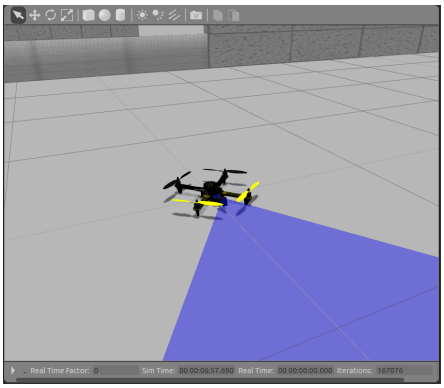


Figure 6 -Simulated UAV with one distance sensor in the front.

4. Conclusion

This paper presented the design of a swarming algorithm that works on quadrotor UAVs with the Pixhawk autopilot board. Using Craig Reynolds flocking rules the algorithm is able to move a small group of UAVs to a common goal while avoiding collisions with each other. The algorithm was implemented in C++ and runs on the ROS platform. A simulation environment based on the Gazebo simulator was prepared to test and evaluate the algorithm in an use case as close as possible to reality.

This implementation however is still very simple and many other improvements could be done in the future. First, our implementation requires the UAVs to fly at a fix altitude, limiting the flock to a two dimensional movement. Further work can be done to eliminate this limitation, obtaining a full three dimensional swarm. Also a new collision avoidance rule can be added to prevent UAVs from colliding with obstacles such as walls.

Acknowledgments. The authors would like to thank the funding institution: CAPES.

References

[1] Moeslinger, C., Schmickl, T., Crailsheim, K. A minimalist flocking algorithm for swarm robots, European Conference on Artificial Life: 375-382, 2009.

[2] Vásárhelyi, G., Virágh, C., Somorjai, G., Tarcai, N., Szörényi, T., Nepusz, T., Vicsek, T. Outdoor flocking and formation flight with autonomous aerial robots, IEEE/RSJ International Conference on Intelligent Robots and Systems: 3866-3873, 2014.

[3] Reynolds, C. Flocks, herds and schools: A distributed behavioral model, Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques: 25-34, 1987.

[4] Hauert, S., Leven, S., Varga, M., Ruini, F., Cangelosi, A., Zufferey, J.C., Floreano, D. Reynolds flocking in reality with fixed-wing robots: communication range vs. maximum turning rate, IEEE/RSJ International Conference on Intelligent Robots and Systems: 5015-5020, 2011.

[5] Quintero, S.A., Collins, G.E., Hespanha, J.P. Flocking with fixed-wing UAVs for distributed sensing: A stochastic optimal control approach, American Control Conference: 2025-2031, 2013.

[6] Kushleyev, A., Mellinger, D., Powers, C., Kumar, V. Towards a swarm of agile micro quadrotors, Autonomous Robots: 287-300, 2013.

[7] Bürkle, A., Segor, F., Kollmann, M. Towards autonomous micro uav swarms, Journal of intelligent & robotic systems: 339-53, 2001.

[8] De Benedetti, M., D'Urso, F., Messina, F., Pappalardo, G., Santoro, C. UAV-based Aerial Monitoring: A Performance Evaluation of a Self-Organising Flocking Algorithm, 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing: 248-255, 2015.

[9] Pixhawk homepage: https://pixhawk.org/. Acessed: September, 2016.

[10] ROS homepage: http://www.ros.org/. Acessed: September, 2016.

[11] Gazebo homepage: http://gazebosim.org/. Acessed: September, 2016.