Artificial Potential Field Swarm Motion Algorithm

Christian J. Howard

August 16, 2015

Abstract

The following document lays out the mathematical foundation and software architecture of algorithms created to help guide a swarm of drones to a desired location in 2D/3D. The potential functions are designed to avoid obstacles and also ensure members of the swarm don't collide with each other. The potential functions are also based on radial basis functions, meaning they automatically scale to any dimension of movement, so 2D and 3D in this case. The algorithms is designed such that a lead drone navigates towards the desired final location and the rest of the swarm attempts to move along with the lead drone.

Contents

1 Notation Definitions

$$\begin{aligned} c &= \text{Scalar} \\ \mathbf{B} &= \text{Vector} \\ ||\cdot|| &= L_2 \text{ Norm} \end{aligned}$$

2 Problem Statement

The problem at hand is to develop algorithms that allow for 2D and 3D motion of a rotorary wing drone swarm, such as a quadrotor swarm, such that they can maneuver around obstacles to a target location without colliding with each other.

The approach specified is to develop Artificial Potential Functions for the drones and find optimal paths based on a probabilistic Particle Swarm Optimization algorithm or Gradient Descent Optimization.

3 Mathematical Formulation

The approach is to make a cost function that each drone uses that ensures they move towards the final location, they avoid obstacles, and they avoid other drones.

3.1 Artificial Potential Functions

The potential function for the drones is focused on avoiding obstacles, avoiding other drones, and moving towards the final location. The mathematical expression for this potential function of the k^{th} follower drone is:

$$J_k = \sum_{i=1}^{N_o} \phi_1(\mathbf{P}_k, \mathbf{P}_o^i, r_o^i) + \sum_{\substack{i=1 \\ i \neq k}}^{N_d} \phi_2(\mathbf{P}_k, \mathbf{P}_i, r_i) + \gamma ||\mathbf{P}_k - \mathbf{P}_f||^2$$

In this case, ϕ_1 generates potential based on a drone being close to an obstacle, ϕ_2 generates potential based on the proximity of the k^{th} and i^{th} drones, and the last term helps move the k^{th} drone to the final location, $\mathbf{P}_f$. γ helps weight the importance of making it to the final location compared to the other parts of the cost function.

The form of $\phi_1(\mathbf{P}_k, \mathbf{P}_j, r_j)$ is the following:

$$\delta = ||\mathbf{P}_k - \mathbf{P}_j||$$
$$\phi_1 = \begin{cases} \alpha e^{-\left(\frac{3\delta}{r_j}\right)^2} & \delta \leq r_j \\ 0 & \delta > r_j \end{cases}$$

The form of $\phi_2(\mathbf{P}_k, \mathbf{P}_j, r_j)$ is the following:

$$\delta = ||\mathbf{P}_k - \mathbf{P}_j||$$
$$\phi_2 = \begin{cases} \alpha e^{-\left(\frac{3\delta}{r_j}\right)^2} & \delta \leq r_j \\ 0 & \delta > r_j \end{cases}$$

A note to make is that α may be a different value for each of the functions above, so there is no assumption that they share the same value. The goal is to find values for γ, α for each function, β, and τ such that the desired performance is met. For example, a large γ will speed up the movement of the swarm towards the desired end location. Smaller values for α will also make it so collisions are more likely, while a large value will help ensure collisions are avoided. Too large a value for α, however, risks getting drones halted around a neighboring drone or obstacle more easily.

3.2 Particle Swarm Optimization

This optimization approach utilizes stochastic sampling within the search space with movement similar to various naturally observed phenomena, such as food search processes in bees.

The approach utilized in this document seeks to minimize an unconstrained objective cost function, J, by an M iteration swarm procedure using N swarm particles. The pseudocode for this algorithm can be expressed as:

- Given:
  - Upper and Lower Bounds of each dimension in the Search Space
  - Handle to the Cost function
  - Number of Swarm Particles
  - Number of Iterations
- Algorithm Steps
  1. Obtain number of Dimensions based on the bounds of the search space
  2. Initialize the Swarm
  3. Begin the Iteration process
     (a) Find the Cost for Each Swarm Particle
     (b) Find the Swarm Particle with the lowest Cost
        - If the lowest cost is lower than the smallest cost out of all the iterations thus far, save this particle location
     (c) Update location of the Swarm based on the particle of the lowest Cost
  4. Return the best particle location out of all the iterations and the cost value at this point
  5. Return the resulting swarm structure, encasing the final swarm locations and costs at each point

3.3 Gradient Descent Optimization

This optimization approach seeks to optimize a roughly continuous and differentiable unconstrained cost function, J, using the gradient, $\nabla \mathbf{J}$, at a given point. This approach aims to use the gradient to move the down the steepest direction. The equation for this approach can be given as:

$$\mathbf{x}^{k+1} = \mathbf{x}^{k} - \alpha \nabla \mathbf{J}^T|_{\mathbf{x}^k}$$

In the implementation found in this software, each component of $\nabla \mathbf{J}$ is approximated using Finite Difference, using the following formula to find the i^{th} component of $\nabla \mathbf{J}$:

$$\frac{\partial J}{\partial x_i} \approx \frac{J(\mathbf{x}^k + \epsilon \hat{\mathbf{e}}_i) - J(\mathbf{x}^k)}{\epsilon}$$

where $\hat{\mathbf{e}}_i$ is a unit vector in the i^{th} component direction and ϵ is a small value, such as 10^{-4}. The next step of the algorithm was to find α such that the following was true:

$$\alpha = \text{argmin } J(\mathbf{x}^k - \alpha^* \nabla \mathbf{J}^T|_{\mathbf{x}^k})$$

A simple implementation of this was utilized, creating an array of step sizes between 10^{-4} and 10^{-1} and testing each of them in the expression $J(\mathbf{x}^k - \alpha \nabla \mathbf{J}^T|_{\mathbf{x}^k})$ and selecting the step size that results in the smallest value.

3.4 Optimization Method Comments

While these optimization methods will trend towards local minima, it isn't certain it will migrate towards the global optimum. While the Artificial Potential Functions are create such that it should generally tend towards the global optimum, which is the final desired location, there is still the potential for a barrier of obstacles to cut off a swarm member and make them get trapped in a local optimum.

For these cases, a new algorithm would need to be utilized for use in globally optimal searches. One approach could take place if one first discretizes the flight domain into rectangles(2D)/rectangular prisms(3D) and creates a graph where the nodes within the graph are the centroids of the rectangles/rectangular prisms and the paths linking the nodes to one another have a cost based on the average value of the cost function at the interface of the neighboring rectangles/rectangular prisms. Then one could solve this problem using a minimum cost trajectory using a graph-related algorithm, such as Dijkstra's algorithm.

Another possible solution could be an optimal path finding algorithm based on calculus of variations. One example, which becomes similar to the approach noted in the paragraph before, is a dynamic programming solution. Another approach could potentially be applying the Adjoint method, traditionally used for obtaining optimal control and state trajectories, using a Hamiltonian based on the Artificial Potential Function desribed earlier.

4 Software Architecture

4.1 Obstacle Structure

Any obstacle in this set up is described as a circular(2D)/spherical(3D) object of constant radius, r, and a centroid located at $\mathbf{P}_o$. The obstacle is defined as a class in Matlab with the following architecture:

- Variables
  - **pos** — This is a variable holding the position vector of the obstacle
  - **radius** — This is a variable holding the radius of the obstacle
  - **dim** — This is a variable holding the dimension of the object. A value of 2 is a 2D obstacle and a value of 3 is a 3D obstacle.
- Internal Methods
  - Obstacle
    - Purpose
      - This is a constructor method, which means this is the function you call when you want to initialize the obstacle object you create.
    - Inputs

        - **pos** — This input is a vector input that represents the position of the obstacle
        - **radius** — This is a variable that represents the radius of the obstacle
    - Outputs
        - **obj** — This is the output obstacle object
- GetPotential
    - Purpose
        - This function is meant to use the position of a given drone and provide this drone the potential it feels based on this obstacle
    - Inputs
        - **obj** — This is an input that really just passes in the class object that you use to call this. You don't actually pass in an input yourself for this.
        - **pos_drone** — This is the position vector of the drone that wants to find how much potential this obstacle generates for it.
    - Outputs
        - **vals** — This is just the value for the potential this obstacle generates for the drone as position pos_drone.
- GetObstaclePotential
    - Purpose
        - This function is meant to use the distance between the obstacle and a drone to generate the potential the drone feels from the obstacle. This function is the same as ϕ_1 from earlier on, except its input is only δ from that equation.
    - Inputs
        - **obj** — This is an input that really just passes in the class object that you use to call this. You don't actually pass in an input yourself for this.
        - **r** — This is the value for δ from the ϕ_1 equation from earlier.

      - Outputs
        - **val** — This is just the value for the potential this obstacle generates given the distance between this obstacle and a drone is r.

4.2 Drone Structure

Any drone in this setup is designated as a circle/sphere with a radius r, a position vector $\mathbf{P}$, the dimensions it is, and a type parameter that decides whether it is a lead drone (only one of these) or a follower drone.

- Variables
  - **radius** — This is the radius of the drone
  - **pos** — This is the vector position of the centroid of the drone
  - **type** — This is a variable that should hold 1
  - **dim** — This variable holds the dimension number of the drone
- Internal Methods
  - Drone
    - Purpose
      - This is the constructor method called to create a drone object.
    - Inputs
      - **radius** — This is the desired radius for this drone
      - **pos** — This is the desired initial position for this drone
      - **type** — This is the desired type for the drone: 1 for follower drone, 2 for lead drone
    - Outputs
      - **obj** — The output drone object
  - GetPotential
    - Purpose
      - This function gives the potential this drone generates for another drone within the swarm

      * Inputs
        - **obj** — This is the object handle calling the function. You don't pass this parameter in, it gets automatically put in when you call the function.
        - **pos_drone** — This is the position of the drone that wants to find the potential this drone generates for it.
      * Outputs
        - **val** — The output potential
    - GetOutsidePotential
      * Purpose
        - This function gives the potential a drone generates for another drone within the swarm. This is equivalent to the ϕ_2 function from earlier, only as a function of δ in this case.
      * Inputs
        - **obj** — This is the object handle that is passed in when you call the function automatically. Don't worry about this parameter.
        - **r** — This is equivalent to δ from the equation this function describes
      * Outputs
        - **val** — The output potential

4.3 Other Important Functions

- FindLocationPotential
  - Purpose
    * The purpose of this function is to output the $\gamma||\mathbf{P}_c - \mathbf{P}_f||^2$ term of the lead drone's potential function
  - Inputs
    * **drone_pos** — This is the position of the lead drone
    * **final_pos** — This is the desired final location for the lead drone to move towards
  - Outputs

    * **potential** — The potential felt by the lead drone to make it to the final location

- getTotalPotential
  - Purpose
    * The purpose of this function is to output the total potential for a given swarm member
  - Inputs
    * **pos** — This is the position of the current drone that you want to find the potential for
    * **drone_index** — This is the index for the drone you want to find the potential for within the drone_array
    * **drone_array** — This is the array holding the data for each of the drones in the swarm
    * **obst_array** — This is the array holding the data for each of the obstacles
    * **end_loc** — This is the vector location for the desired end location
  - Outputs
    * **J** — The total potential