1 Problem Formulation

There is an increasing interest and need for swarm based mobile robots in many different fields such as ecology and defense. The hindrance in deploying such systems is the lack of formal algorithmic techniques that can efficiently plan the behaviour of the robots. This paper proposes a new system based on general flocking principles that can enable control of a swarm by utilizing desirable properties from potential fields and probabilistic road maps (PRM).

2 Method Summary

The algorithm uses a mixture of potential fields and very biased random walks for the local planner and also uses PRMs for the global planner. The superposition of the potential fields is what governs how the flock behaves and the PRM governs where the flock will go. The PRM is very important because it can take into account different environmental factors. A weight can be can be added to each node of the roadmap as well as to each edge. The weight on the edge represents the distance between two verticies in the graph whereas the node weight can represent how well suited this area is for a swarm to go. This will allow for more intellegent planning and could reduce the probability that a boid will be trapped in a stuck state (i.e. local minima).

At a lower level, each boid has to obey three rules but a fourth was because of the use of potential fields:

1. There is seperation between boids
2. There is long range attraction between boids (or to a common goal).
3. A boid's heading is influenced by the headings of its neighbours.
4. A boid will be repulsed from obstacles

The first rule is satisfied by creating a weak repulsive sigmoidal potential field between the boids that allow them to travel close together but to be repulsed or pushed away when the distance is too small. The second is satisfied by creating a attractive sigmoidal field from a boid to a goal. These goals (as stated before) are set by the global planner and in this case the PRM. For the third rule, a dynamic function is utilized that will assign probabilities to the boids that represent how well of a neighbour each boid in the swarm will be. The function is dynamic because the requirements of a good neighbour are dependent on the state of the boid. The headings of the k neighbours that are chosen are averaged and combined with the potential field superposition to determine the new heading and position of the boid. The last requirement for boid movement is governed by a repuslive inverse squared potential field from a boid to an obstacle. As will be explained below, this field takes priority due to its very volatile potential function.

Another unique aspect of this project is the inherit nature of the dynamism present in the governing function. As will be explained below. There are a set of transitions that can be applied to the potential functions as well as the neighbour choosing function, γ. These transitions create a dynamic potential field that can be more easily maneuvered with a lower probability of getting into a local minima (stuck state).

3 Functions & Algorithms

3.1 Boid Object

Throughout this paper a variable will reoccur with is called a *boid*. It is a member of flock or swarm of mobile robots. In the functions in section 3.2, the variable F is used to represent a flock. There are variables that are associated with each boid and there are functions applied to these variables. With the set of functions and the set of variables, one is able to dictate the behaviour of the flock. Below is a list of the variables attached to the boid object and their meanings.

$$
\begin{aligned}
b &= \text{The boid object} \\
b_r &= \text{The radius of the circular boid} \\
b_g &= \text{The current goal point. The goal has a circular representation and} \\
&\quad \text{also has a radius and position vector} \\
b_p &= \text{A vector representation of the boid's position} \\
b_{buf} &= \text{A buffer of positions of length } n \\
b_{h_x}, b_{h_y} &= x \text{ and } y \text{ directions of the heading}
\end{aligned}
$$

The subscripts that are attached to the boid objects have the same meaning when applied to other variables. For instance, to represent a point the variable q is used. This point q also has a radius and a position and therefore the same notation is used.

3.2 Functions

Obstacle Potential

$$P(b, o) = \frac{\beta \cdot b_r \cdot o_r}{(\|b_p - o_p\| - b_r - o_r)^2} \qquad \beta \in \mathbb{R}$$

The potential that is derived due to an obstacle is modelled using an inverse squared relationship. This means that as the boid approaches an obstacle, the repulsive potential from the object will tend towards infinity. This gives the obstacle potential priority over the other fields. Also, it should be noted that the denomenator of the fraction does ot simply hold the distance from the boid to the obstacle, but also takes into account the radiuses of the boid and the obstacle in order for their edges to not overlap.

Sigmoid Potential

$$Sig(b, q) = \frac{\alpha \cdot b_r \cdot q_r}{1 + \beta \cdot \exp(\delta \cdot \|b_p - q_p\|)} + c \qquad \alpha, \beta, \delta, c \in \mathbb{R}$$

The function above is used to derive the attractive potential towards a goal and a repulsive potential from other boids. The sigmoid curve allows for a constant potential outside some radius of influence without haveing to threshold anything, Also, the equation does not tend towards infinity on the vertical axis, so the numbers will always be dealt with appropriately. The function is utilized in order to seperate boids outlined in the first common rule of flocking

Neighbours

$$N(F, k) = b \in \max_k(\{(b, \rho) : \rho = \gamma(b), b \in F\})$$

What seperates the algorithm presented in this paper and those present in others is the presence of the γ function. This function is used to determine a probability that the given boid would be a suitable neighbour. γ is an undefined function because it is dynamic and may change due to different states of each boid (i.e. if the boid is in a stuck state, closer neighbours may be chosen).

Comment: What the notation is trying to show in N is that corresponding b values for the k largest ρ values are returned in a set.

Direction Vector

$$D(b,q) = \frac{s \cdot (q_p - b_p)}{||q_p - b_p||} \qquad s \in \mathbb{R} \text{ (indicating speed)}$$

Due to the nature of the simulation, a function that gives the value of direction vector of a certain length was needed. A direction vector is derived by multiplying the *speed* and the unit vector in the desired direction. This function returns the direction vector from objects b to q.

Stuck Threshold

$$T_s(b) = |\sum_{i=0}^{l-1} b_{buf}[i]_x - b_{buf}[i+1]_x| + |\sum_{i=0}^{l-1} b_{buf}[i]_y - b_{buf}[i+1]_y|$$
$$l = |b_{buf}|$$

It is also very crucial to determine whether or not a boid is in a *stuck* state. If a boid is in a stuck state, it will have lower probability that it will make a good neighbour for another boid. This function determines the wholistic movement in either direction which can be used to determine if the boid is stuck or not.

Determine Stuck Boid

$$S(b) = \begin{cases} 1 & , \quad T_s(b_{buf}) < T \\ 0 & , \quad \text{otherwise} \end{cases} \qquad T \in \mathbb{R} \text{ (Representing the stuck threshold)}$$

Determining whether a boid is stuck or not is of crucial importance due to its implications in the γ function used to pick neighbors. Neighbours which are considered *stuck* should exhibit different behaviour and should also have different neighbours. They should also be less considered to be a neighbour. In this function one is assuming that there is a threshold T that would constitute a boid being in a stuck state. This T could be learned by checking the relative speed of the neighbours and checking for a statistical or wholistic difference.

Function Transformation

$$\tau(f, t) = f\prime \qquad t \text{ is a translation applied to } f$$

The most interesting feature by having a function choose the best fitting neighbour and determine the attractive and repulsive potentials is the fact that a function is dynamic. Once the boid has entered a stuck state or fast state, the functions can be changed to adapt to the environment.

The defined function transformations are:

$$\begin{aligned} t_{\gamma_{\neg s}} &= \text{Transformation to } \textit{gamma} \text{ when the boid is not stuck} \\ t_{\gamma_s} &= \text{Transformation to } \textit{gamma} \text{ when the boid is stuck} \\ t_{P_{\neg s}} &= \text{Transformation to } P \text{ when boid is not stuck} \\ t_{P_s} &= \text{Transformation to } P \text{ when boid is stuck} \end{aligned}$$

3.3 Algorithms

3.3.1 Goal Generation

The **GoalGenerator** algorithm is a weighted for of a PRM. Random distributed points are distributed in the configuration space. These points are then attached using some function. In our case, one simply uses a threshold to define which two verticies in the graph should be neighbours because the desired out come is a smooth path. Another key component of creating the weighted graph is that each node also has a weight. This weight is referred to as the *Environmental Weight Function* which takes a point as a parameter and returns a weight corresponding to how well a boid will travel in this section of the configuration space. In practice, this function could try to maximize spacing between sample points and obstacles in order to limit the amount of trapping in local minima. Another implicit function in **GoalGenerator** is the *GetDistanceTupleSet(s, S)* function. This returns a set of tuples in which the first element being distance from s to some point p in S, and the second element is p. This helps to establish a relation between two points.

Algorithm 3.1: GoalGenerator(p_s, p_e)

```
$\mathcal{S} \leftarrow RandomPoints(p_s, p_e)$
$\omega \leftarrow EnvironmentWeightFunction()$
$\mathcal{G} \leftarrow EmptyGraph()$
for each $s \in \mathcal{S}$
    do
        $\mathcal{D} \leftarrow GetDistanceTupleSet(s, \mathcal{S})$
        $\mathcal{N} \leftarrow$ Set()
        for each $(d, p) \in \mathcal{D}$
            do
                if $d < d_T \wedge d \neq 0$
                    then $\mathcal{N}$.add$(\omega(p) + d, p)$
        $\mathcal{G}$.add$(s, \mathcal{N})$
$\mathcal{P} \leftarrow Dijkstra(\mathcal{G}, p_s, p_e)$
return $(\mathcal{P})$
```

3.3.2 Boid Updating

The algorithm moves a swarm across a configuration space. The main loop checks if the swarm has reached a goal. If the whole flock has not reached the goal, the position of each boid is updated with regards to the three potential fields in use. The reason for all the summations is that the resultant direction and position of a boid at any given time is a weighted vector sum of all the potential fields (a weighted superpositioning). These summations are used in the to determine the new heading of the boid. Within the algorithm, there is a central planning distinction between a boid being stuck or not. If the boid is not in a stuck state, a special transition is applied to γ in which allows the boid to choose more suitable neighbours. Likewise, another transition is applied to the obstacle potential function which allows for more fluid travel. Similarily, different transitions are applied to these functions when the boid is in a stuck state. However, there is a key difference between boid planning whilst the boid is in a stuck state and when it is in a free state. The boids in a stuck state will display a very biased random walk. This moves the boid out of a local minima to hopefully be attracked towards a goal in such a way that it is in motion. A random walk can be switched with a number of different planning algorithms but most noticably an low range EST would work well.

Algorithm 3.2: UPDATEBOID(F, O)

```
$F_G \leftarrow GoalGenerator(p_{start}, p_{end})$
while $\neg ReachedGoal(F)$
  do
    for each $b \in F$
      do
        $C_b \leftarrow \textbf{Set}()$
        $(r_w, r_{lower}, r_{higher}) \leftarrow GetRandomWalkValues()$
        if $\neg \mathrm{InGoal}(b)$
          then
            $R_b \leftarrow -\sum_{q \in C_b} Sig(b,q) \cdot D(b,q)$
            $R_{b_{sum}} \leftarrow \sum_{q \in C_b} Sig(b,q)$
            $R_O \leftarrow -\sum_{q \in O} P(b,q) \cdot D(b,q)$
            $R_{O_{sum}} \leftarrow \sum_{q \in O} P(b,q)$
            $A_g \leftarrow Sig(b, b_g) \cdot D(b, b_g)$
            $A_{g_{sum}} \leftarrow Sig(b, b_g)$
            if $\neg S(b_{buf})$
              then
                $\gamma \leftarrow \tau(\gamma, t_{\gamma_{\neg s}})$
                $P \leftarrow \tau(P, t_{P_{\neg s}})$
                $\eta \leftarrow N(F, k)$
                $b_{h_x} \leftarrow \frac{[\sum_{n \in \eta} n_{h_x}] + R_{b_x} + R_{O_x} + A_{gx}}{R_{b_{sum}} + R_{O_{sum}} + A_{g_{sum}} + k}$
                $b_{h_y} \leftarrow \frac{[\sum_{n \in \eta} n_{h_y}] + R_{b_y} + R_{O_y} + A_{gy}}{R_{b_{sum}} + R_{O_{sum}} + A_{g_{sum}} + k}$
              else
                $\gamma \leftarrow \tau(\gamma, t_{\gamma_s})$
                $P \leftarrow \tau(P, t_{P_s})$
                $\eta \leftarrow N(F, k)$
                $r_x \leftarrow RandomInt(r_{lower}, r_{higher})$
                $r_y \leftarrow RandomInt(r_{lower}, r_{higher})$
                $b_{h_x} \leftarrow \frac{[\sum_{n \in \eta} n_{h_x}] + R_{b_x} + R_{O_x} + A_{gx} + r_x \cdot r_w}{R_{b_{sum}} + R_{O_{sum}} + A_{g_{sum}} + k + r_w}$
                $b_{h_y} \leftarrow \frac{[\sum_{n \in \eta} n_{h_y}] + R_{b_y} + R_{O_y} + A_{gy} + r_y \cdot r_w}{R_{b_{sum}} + R_{O_{sum}} + A_{g_{sum}} + k + r_w}$
          else $b_g \leftarrow NextGoal()$
        $UpdatePosition(b)$
```