Rover

Alex Wallar

June 16, 2014

1 Introduction

Unmanned aerial vehicles, such as quadrotors, are becoming a cheap and feasible way to provide persistent monitoring of an area even when monitoring that area comes with a risk of the quadrotor being detected by possibly hostile parties. These quadrotors are left unattended and need to be able to plan their movements autonomously such that they maximize the quality of their sensors and maintain as much sensor coverage as possible, whilst minimizing the risk of being apprehended. The autonomous planning also needs to take into account that some quadrotors will be dispatched from the swarm to follow certain targets of interest. The planning algorithm should be able to organically readjust the swarm such that maximum sensor quality and minimum risk are preserved.

The proposed solution, Rover, provides a method of organically readjusting the swarm to fill the search space whilst maximizing sensor quality and sensor coverage while minimizing the risk of detection by hostile targets by combining dynamic potential fields and non-linear optimization techniques.

2 Rover

Rover is an planning algorithm that seeks to maximize the sensor quality and sensor coverage for a group of quadrotors, whilst minimizing the risk of a quadrotor being detected by a possibly hostile enemy. This is done by splitting the planning into two different parts, planning for the 2-D plane to promote sensor coverage and planning for the altitude by minimizing risk and maximizing sensor quality.

2.1 2-D Planning

For 2-dimensional planning, Rover needs to make sure that the group of quadrotors fills the search space and guarantee that no one part of the space goes too long without being surveyed. This is done by discretizing the space into a grid (in this paper called a time grid and referred to as $\mathcal{T}$. Whenever a box in the grid has been covered by a sensor on any quadrotor in the group, the current time is stored in that box. For the quadrotors to move in the xy plane, each quad will sample a given number of segments along the sensing ellipse and move towards the segment whose average time is the least. This

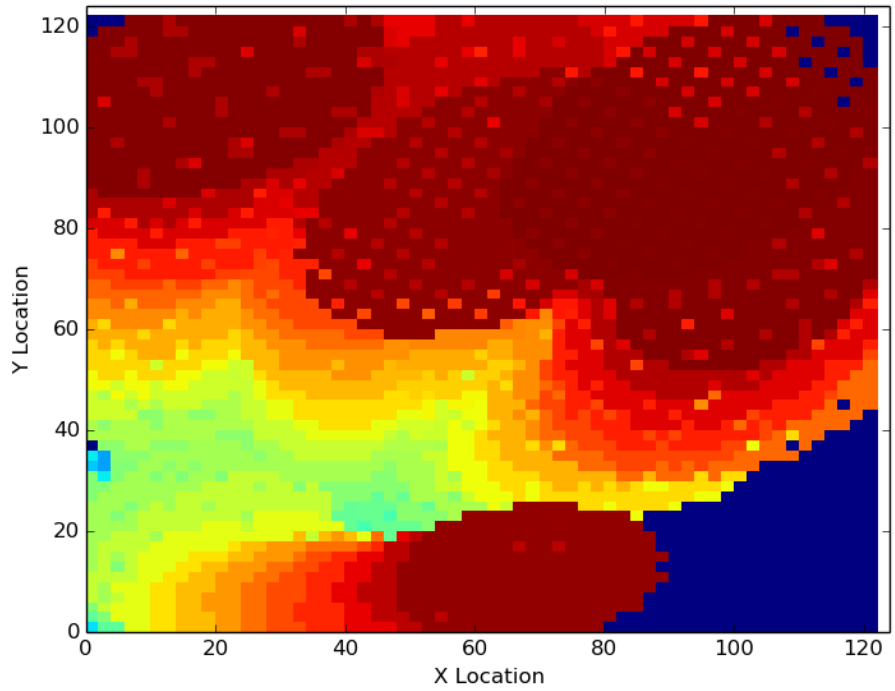

Figure 1: An instance of the time grid where blue represents locations that have not been visited in a large amount of time and where red represents locations that have just been visited

simple rule has extreme emergent properties for the swarm. Since the swarm shares the same grid, the quads will act cooperatively to fill the space without having to provide much guidance. An instance of the time grid for a given instance in time is shown in **Figure 1**. Also, this planning is not dependent on how many quads there are in the swarm. If a quad leaves the swarm, it would simply no longer update the time grid. The other quads would have no knowledge that it left and would still be able to plan accordingly.

The quadrotors used in simulation have a variable camera angle. This means that instead of simply having a down facing camera, the quadrotor can have the camera at an angle, ϕ. This means that the camera's projection onto the xy plane is not a circle but an ellipse. The major and minor axis of

Algorithm 1 GetDirection$(q, \beta, \phi, \mathcal{T})$

$t_s \leftarrow 0$
$\mathcal{P} \leftarrow$ Set()
$\zeta \leftarrow$ GetInnerSearchRange()
$S_\theta \leftarrow$ GetUniformSamples$(0, 2\pi)$
for $\theta \in S_\theta$ **do**
  $S_\omega \leftarrow$ GetUniformSamples$(\theta - \zeta, \theta + \zeta)$
  for $\omega \in S_\omega$ **do**
    $A_M \leftarrow q.z \cdot (\tan(\phi + q.\alpha) - \tan\phi) + \epsilon$
    $A_m \leftarrow q.z \cdot \frac{\tan q.\alpha}{\cos\phi} + \epsilon$
    $x \leftarrow q.z \cdot \tan(\phi - q.\alpha) + A_M \cdot (1 + \cos\omega)$
    $y \leftarrow A_m \cdot \sin\omega$
    $\begin{bmatrix}\hat{x}\\ \hat{y}\end{bmatrix} \leftarrow \begin{bmatrix}\cos\beta & -\sin\beta\\ \sin\beta & \cos\beta\end{bmatrix} \cdot \begin{bmatrix}x\\ y\end{bmatrix} + \begin{bmatrix}q.x\\ q.y\end{bmatrix}$
    $t_s \leftarrow t_s + \mathcal{T}[\hat{x}, \hat{y}]$
    $\mathcal{P}.add(\hat{x}, \hat{y})$
$\bar{x} \leftarrow 0$
$\bar{y} \leftarrow 0$
for $(x, y) \in \mathcal{P}$ **do**
  $w \leftarrow \frac{\mathcal{T}[x,y]}{t_s}$
  $\bar{x} \leftarrow \bar{x} + w \cdot x$
  $\bar{y} \leftarrow \bar{y} + w \cdot y$
return $(\begin{bmatrix}\bar{x} - q.x & \bar{y} - q.y\end{bmatrix}^T, \frac{t_s}{|\mathcal{P}|})$

the ellipse are, $A_M(z, \phi) = z \cdot \tan(\phi + \alpha) - z \cdot \tan\phi$ and $A_m(z, \phi) = \frac{z \cdot \tan\alpha}{\cos\phi}$ where z is the height of the quadrotor, ϕ is the camera angle and α is the viewing angle.

Since an ellipse is not symmetrical for every angle, orientation cannot be ignored. This means that the 2-D planner must plan for not only x and y but also an orientation, β. Also, for completeness, we assume that the camera angle, ϕ, is also dynamic and can be set by the planner. In order to plan the movements, we sample viable new orientations and camera angles, each time determining the new x and y direction to go in. Then once the sampling has completed, we choose a (x', y', β', ϕ') that would lead the quadrotor to a configuration that not has been sensed in the largest amount of time. More

formally, we would like to minimize,

$$E(\mathbf{s}) = \int_{\mathbf{s}_\theta - \zeta}^{\mathbf{s}_\theta + \zeta} \mathcal{T}(e_x(\mathbf{s}, \omega), e_y(\mathbf{s}, \omega)) \cdot \sqrt{\left(\frac{\partial e_x}{\partial \omega}\right)^2 + \left(\frac{\partial e_y}{\partial \omega}\right)^2}\, d\omega$$

where E is the line integral on the curvature of the sensing ellipse within a range centered at the currently sampled heading, θ in which the surface being integrated upon is the time grid, $\mathcal{T}$. The variable ζ represents the interval of the integral and governs the size of the line segment sampled when determining the new heading. The vector, $\mathbf{s} = \langle z, \theta, \beta, \phi \rangle$ in E holds the current state of the altitude, heading, orientation, and camera angle respectively. The equations,

$$\begin{bmatrix} e_x(\mathbf{s}, \omega) \\ e_y(\mathbf{s}, \omega) \end{bmatrix} = \begin{bmatrix} \cos \mathbf{s}_\beta & -\sin \mathbf{s}_\beta \\ \sin \mathbf{s}_\beta & \cos \mathbf{s}_\beta \end{bmatrix} \cdot \begin{bmatrix} \mathbf{s}_z \cdot \tan(\mathbf{s}_\phi - \alpha) + A_M(\mathbf{s}_z, \mathbf{s}_\phi) \cdot (1 + \cos \omega) + \epsilon \\ A_m(z, \mathbf{s}_\phi) \cdot \sin \omega + \epsilon \end{bmatrix}$$

are parametric equations representing points on the sensing ellipse given a quadrotor state, $\mathbf{s}$, along with an angle parameter, $\omega \in [0, 2\pi]$.

Therefore to determine the direction, θ, the orientation, β, and the camera angle, ϕ, that the quadrotor should travel in for a given height, z, we have to minimize E with respect to θ, β, and ϕ. This function is minimized using an iterative sampling approach in which a uniform or random sample depending on the variable is taken such that every element in that sample leads to a feasible configuration. Every combination of these sets is taken iteratively in order to find the combination that minimizes the objective function, E. Once a suitable $\mathbf{s} = \langle \theta, \beta, \phi \rangle$ has been determined using the iterative approach, the quadrotor's orientation and camera angle are set and the current position of the quadrotor is updated to,

$$\begin{bmatrix} q.x \\ q.y \end{bmatrix} = \begin{bmatrix} q.x \\ q.y \end{bmatrix} + k \cdot \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix}$$

where k is the step size of the quadrotors and q is the quadrotor whose position is being updated. This is shown in both **Algorithm** 1 and **Algorithm** 2. **Figure 2** shows the planning algorithm filling the space whilst optimizing the heading, orientation, and c4mera angle.

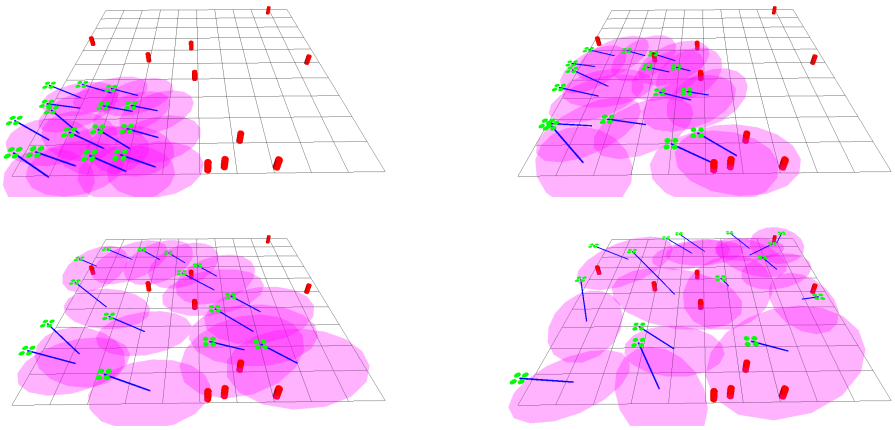

Figure 2: The quadrotors filling up the search space

2.2 Determining the altitude

To determine the altitude, one must solve a non-linear optimization problem that tries to optimize the height of the quad by maximizing the sensor quality and minimizing the risk. This is done by modelling the sensor quality and the risk as functions parametrized by the altitude, and combining them together into an objective function that is then maximized.

Sensor Quality To model the sensor quality, a few assumptions were made. Firstly, that there is an optimal distance away from a tracked object such that the sensor would have the highest quality. For example if one was tracking a person and a tank, the optimal distance to track a person is much less than that of a tank because the tank is larger and moves faster. It is assumed that this optimal height is passed into the algorithm as an argument and changed for each situation. The second assumption that is made is that the sensor quality decreases exponentially as the distance increases. This depends on the type of sensor that is being used and that is why this fall off can be determined the user as well. Below is the function used to model the sensor quality. It is a normal distribution centred at the optimal height set by the user and has a standard deviation representative of the range of the sensor,

$$SQ(z) = \exp\left(-\frac{(\frac{z}{cos\phi} - \mu_{sq})^2}{2\sigma_{sq}^2}\right)$$

where z is the height of the quadrotor, ϕ is the angle of the camera, μ_{sq} is the optimal height for the sensor, and σ_{sq} is a variable indicative of the range of the sensor.

Risk The risk is modelled in two parts. Firstly there is an initial risk, R_0, which when given an x and y position, returns the risk as a value from 0 to 1 inclusively. This initial risk is basically the risk of the quadrotor being detected at the ground level. This initial risk is used to scale the risk function as altitude increases. For instance, if a quadrotor is over a very hostile environment, there will be a very high initial risk. This would cause the risk as height increases to decay more slowly because the quadrotor would need to obtain a higher altitude to have less risk. However, if there is a low initial risk, the risk as altitude increases will decay rapidly allowing the quadrotor to fly closer to the ground. The initial risks are determined by a set of risk points on a 2-D grid. These risk points are known *a priori*. These points are coupled with a constant between 0 to 1 which indicates how hostile this risk point is and a constant which indicates the range of this threat. These points are then represented as the centres of 3-D normal distributions which are then combined to determine the initial risk in a 2-D plane. The equation,

$$R_0(x, y) = \max_{r_p \in RiskPoints} r_p.hostility \cdot \exp\left(-\frac{\sqrt{(r_p.x - x)^2 + (r_p.y - y)^2}}{2 \cdot r_p.range^2}\right)$$

where x, y is the location of the quadrotor, *RiskPoints* is the list of risk points in the grid, $r_p.range$ is the range of the sensor, and $r_p.hostility$ is a measure of the hostility of the risk point from 0 to 1 is the representation of the initial risk, or the risk at the ground level.

Given a function for the initial risk, we are able to determine the risk at an altitude by scaling this initial risk with exponential decay as height increases. To achieve this functional behaviour, a normal distribution has been used with a mean of 0 and a standard deviation based on a scaling of the initial risk. The equation,

$$R(x, y, z) = R_0(x, y) \cdot \exp\left(-\frac{z^2}{K \cdot R_0(x, y)^2}\right)$$

is used to model the risk in three dimensions where x, y, z is the position of the quadrotor, K is a scaling constant, and $R_0(x, y)$ is the initial risk at x, y.

Optimization In order to determine the height of the quadrotor, me must optimize an objective function parametrized by the altitude that maximizes the sensor quality and minimizes the risk. This can be modelled as,

$$J(x, y, z) = SQ(z) - R(x, y, z)$$

and therefore to determine the optimal altitude, we must find the z value that maximizes J for a given x, y. This is shown in the equation,

$$\textsc{DetermineHeight}(x, y) = \underset{z \in [z_{min}, z_{max}]}{\arg\max} \; J(x, y, z)$$

where DetermineHeight takes the current x, y position as parameters and returns the z value that minimizes the risk whilst maximizing the sensor quality. This can be solved with many open source non-linear optimizers. The package being used in this implementation is SciPy. Once the altitude needed for the quad has been determined, the new height is set and the time grid is updated. Since the time grid is updated and it is shared within the swarm, the change in altitude (and therefore a change in the sensor radii) will have reactive side effects by the swarm. If a quad, q, increases it's height, the swarm will move away from it, and if q decreases it's height, there will be more unfilled space around it and the swarm will move to fill the new space. These properties are what allow us to break the planning into two parts, one algorithm determining the new x, y position and orientation and another algorithm determining the new height of the quadrotor.

The complete algorithm is shown in **Algorithm** 2. In the algorithm, ϕ is the camera angle, α is the viewing angle of the camera, k is the step size, and A_M, A_m are the respective semi-major and semi-minor axes.

Algorithm 2 Pseudocode for Rover

$\mathcal{Q} \leftarrow$ GetQuadrotorList()
$\mathcal{T} \leftarrow$ InitTimeGrid()
while TrackingFinished() = `false` **do**
  for $q \in \mathcal{Q}$ **do**
    $t_{min}, v', \beta', \phi' \leftarrow$ InitDynamicVars()
    $S_\phi \leftarrow$ GetRandomSamples(ϕ_{min}, ϕ_{max})
    $S_\beta \leftarrow$ GetRandomSamples$(\beta_{min}, \beta_{max})$
    for $\phi \in S_\phi$ **do**
      for $\beta \in S_\beta$ **do**
        $(v, \bar{t}) \leftarrow$ GetDirection$(q, \beta, \phi, \mathcal{T})$
        if $\bar{t} < t_{min}$ **then**
          $t_{min} \leftarrow \bar{t}$
          $\beta' \leftarrow \beta$
          $\phi' \leftarrow \phi$
          $v' \leftarrow v$
    $\begin{bmatrix} x' \\ y' \end{bmatrix} \leftarrow \begin{bmatrix} q.x \\ q.y \end{bmatrix} + k \cdot \frac{v'}{||v'||}$
    $z' \leftarrow$ DetermineHeight(x', y')
    $q.SetPosition(x', y', z', \beta', \phi')$
    $\mathcal{T}.Update(q)$

3 Results

3.1 Experimental Setup

The experiments seeked to show that the proposed algorithm minimizes the risk, maximizes the sensor quality, and provides sensor coverage to a designated area. The statistics recorded were comprised of the average percent of the total area coverage, the average sensor quality, and the average risk. The sensor quality and risk metrics were determined using the same functions as in the optimization process. The percent of the total area coverage was determined using a Monte Carlo process where a large number of random points (1000) with x and y values within the search area were checked to be within any of the sensing ellipse at each iteration. The ratio of the number of points sampled which are within any of the sensing ellipse to the total number of points sampled is what determines the percent total area coverage

metric. For the experiments, two different independent variables were used; the number of risk points and the number of quadrotors. The risk points were randomly placed into the search space however, the random generator was seeded so the experiments would be deterministic and repeatable. Also, all of the risk points used were assigned an associated hostility and sensor range of 1 and the width of the search space divided by two respectively. The quadrotors all started in a square formation at the top left of the search space. Experiments were carried out by varying the number of risk points and the number of quadrotors such that each number or quadrotors run on every configuration of risk points. The number of risk points varied from 1 to 21 with a step of 2 and the number of quads varied from 1 to 26 with a step of 5. Each configuration was run for 1000 iterations and each configuration only needed to be tested once because the process is deterministic.