On Efficient Methods of Patrolling Streets to Prevent Jaywalking

Nicholas Alessi, Diwakar Raisingh, Shilei Zheng

November 2010

Abstract

In the year 2084, jaywalking seriously threatens the safety of the citizens of Gotham City. We plan to use MAVs on the streets of Gotham to combat this threat. We model Gotham as a graph of streets, where each edge represents a street patrolled by our MAVs. Furthermore, in order to quell any resistance efforts implemented by dissident jaywalkers, we show that a method based on Fleurys Algorithm is on the order of twice as unpredictable as Random Walks. Additionally, we propose a method to break Gotham into Domains and a protocol that allows for the efficient replacement of broken MAVs. Using models of the MAVs we computationally show that maximizing the speed of the MAVs produces the most cost efficient solution to this rampant problem. We assume that the MAVs and repairs were the main components of cost and sought to minimize them to achieve our objective. By having the drones patrol at their top speeds of 30mph, we are able to achieve at least 25% reduction in costs compared to when the MAVs patrol at 20mph. Finally, we apply our model to a real-life geographic section of Manhattan partitioned into 3 sectors with varying pedestrian traffic, ultimately developing a simple scheme for re-partitioning traffic sectors that adapts to dynamic pedestrian movement.

Contents

1 Introduction

Jaywalking is a significant problem in cities as it curtails the flow of traffic. Since few tickets are given to offenders, many people jaywalk. To significantly reduce the number of jaywalking offenses, surveillance is required. This paper will model techniques that can be used to efficiently patrol the streets for jaywalkers from the air.

1.1 Terms

Throughout the paper we will be using several terms that we define below.

- Micro Unmanned Aerial Vehicle = MAV = Drone.
- Stale Time = The amount of time that has passed since a street has been patrolled.
- Traffic Sectors = Areas in which pedestrian traffic has different value. This will be explained in detail later.
- There is a central control station that tracks all the drones and can command all of them.

1.2 Assumptions

- We will use the Draganflyer X6 as our approximation for the MAV. The X6 costs $19,995. The specifications are included in the Appendix.
- The battery for the MAV is very cheap (conservatively estimated to be under $160) compared to the Draganflyer. We assume this because the Draganflyer's battery holds less power than two MacBook Air batteries, which together are lighter than the Draganflyer's.
- A MAV can only see one block in front of it.
- The MAV's only breaks down at street corners and when it breaks, it is immediately appears at the repair station.
- All the MAV's are flying at the same height and this height is optimal for identifying jaywalkers.
- Electricity is cheap therefore charging the batteries does not cost much.
- The MAV's have built in collision avoidance systems

1.3 The Goal of the Model

The goal of our model is to minimize the cost of the anti-jaywalking operation. Because our approximation of the MAV is extremely expensive, it comprises most of the cost of the operation. Thus, to keep costs low we need to minimize the number of MAV's in the Gotham City fleet. The repair times of the drones needs to be minimized because labor costs are likely to be high. We also seek to generate non-deterministic paths that the MAV's will follow so that jaywalkers cannot evade them.

2 Single MAV Domains

If we constrain each MAV to have its own domain then it becomes important to decide the path the MAV follows. The main concerns when choosing a path are whether any edges are covered too infrequently and the predictability of the drone movements. For ease we will only consider the case that the graph of the roads in a domain is a n by n grid. We will also assume that a MAV can only see a road if it goes along that road. Finally, we will construct the domains such that every road in a domain has the same stale value.

2.1 Predetermined Paths

The easiest way to ensure that every edge is covered is to find a path that covers each edge and have the MAV follow that path over and over. This means that if we know the speed that the MAV travels and stale value of the roads on the domain we can figure out the maximum size of a domain. The main difficulty is that a multigraph has an Eulerian cycle if and only if every vertex has even degree [10]. However the n by n grid graph does not have this property, all nodes on the perimeter but not the corners have 3 adjacent edges. To rectify this we will choose to have all edges on the interior of the grid graph to be doubled, so wherever there was one edge there will be 2. This choice means that the drone will patrol on the edge relatively less often, but by making the edges shared between domains all edges will be covered equally.

Saying that the speed is s in edges per hour, the stale time is t in minutes, and that all edges are the same length (which is not true in reality). Then the number of edges is $2 \cdot n \cdot (n-1) - 2 \cdot (n-1) + 2 \cdot n \cdot (n-1) - 2 \cdot (n-1) = 4 \cdot (n-1)^2$. Then to find the maximum size of the domain we want the maximum n such that $s \geq \frac{60}{t} \cdot 4 \cdot (n-1)^2$. This is because we need to cover all $4 \cdot (n-1)^2$ edges $\frac{60}{t}$ times per hour at a speed of s edges per hour.

While we can make nice predictions about this model, so can the jaywalkers! All that has to happen is someone records the path of each MAV, the speed, and the location at some point in time and then they know exactly where the MAV will be at any time. This is especially bad if someone writes a Jaywalking app that takes the time and the collected information to say whether it is safe to jaywalk at any point in time. So the jaywalking problem will still be as bad as ever.

2.2 Random Walks

Instead of looking for homogeneous coverage we can try to make the MAV movements hard to predict. The random walk approach says whenever a MAV reaches a node (intersection) it chooses a direction at random and goes that way. This has as an advantage the fact that no path can be predicted for the MAV, therefore no general counter strategy could be created.

Unpredictability is also a problem because we want to say that every road will be covered as frequently as required for its busyness. The Random Walk model does not ensure that will happen, but there is the hope that in practice the model will cover all roads as much as required and that each road will be covered with the same frequency. To test how well this covers the graph we ran repeated simulations of random walks starting at the same point and calculated the number of times each edge in the graph was crossed. Then we took the average number of times each edge was visit and then took the average over all edges and the standard deviation over all edges to see

how well this worked. The graph was a 10 by 10 grid, and the simulation was run 20 times. The simulation code is many_random_simulate.m (See appendix).

Number of Steps	125	250	375	500	750	1000
Mean	0.6944	1.3889	2.0833	2.7778	4.1667	5.5556
Standard Deviation	0.3963	0.5575	0.6939	0.8473	1.2626	1.4360
Mean / Std	1.752	2.4913	3.0023	3.2784	3.3000	3.8688

These computations suggest that as paths grow longer the random walk approach does tend to cover well and be unpredictable. However, in the short term it does nowhere near as well. So one would imagine the Jaywalking app gives users points for MAV sightings so it has recent information about MAV positions and then it could give users better knowledge about the chance they will be caught jaywalking. So despite being unpredictable in the long run in the short term MAV movements are statistically biased.

2.3 Fleury's Algorithm

Like in the predetermined path section we can turn the MAV's domain into a multigraph that has an Eulerian cycle. In that case the MAV's can use Fleury's Algorithm to compute an Eulerian cycle over the domain[4]. We will modify Fleury's algorithm to do the following:

1. pick any vertex to start at
2. choose an edge adjacent to the current vertex at random under the constraint that removing that edge does not disconnect the graph or there is no other edge choice
3. travel along the chosen edge
4. remove the traveled edge
5. repeat steps 2 to 4 until all edges have been traversed

This is exactly the same as Fleury's Algorithm except the choice in step 2 must be random. Therefore this algorithm will always produce an Eulerian cycle because one does exist. We know that this will cover all the edges in the domain evenly so all that we need to show is that predicting where the MAV will be is hard.

One point suggesting that in practice it will be difficult for people to predict is that fact that counting the number of Eulerian cycles in an undirected multigraph is #P-complete, i.e., there is not and very likely is not an efficient algorithm to compute that value[1].

We also approach the unpredictability of this algorithm by numerical simulation. If the time at which an edge is crossed is uncorrelated to the edge then it is difficult to predict given the time it takes to cycle and the start point of the cycle when the MAV will cross any given edge (and because most edges are doubled it will be even harder). We computationally modeled this using many_Eulerian_simulate.m (see appendix) running 50 runs.

Grid Size	3 by 3	4 by 4	5 by 5	6 by 6	7 by 7	8 by 8	9 by 9	10 by 10
Mean	8.5	18.5	32.5	50.5	72.5	98.5	128.5	162.5
Standard Deviation	2.0294	3.6914	5.9404	8.3988	14.0462	15.9582	21.5337	24.9831
Mean / Std	4.1884	5.0116	5.471	6.0128	5.1615	6.1724	5.9674	6.5044

This computation suggests that using this approach will be unpredictable in practice. Comparatively these ratios are higher than those for the Random Walk model, so we can conclude that using Fleury's Algorithm is less predictable than Random Walks. Additionally, since this guarantees every edge will be covered we have more equal coverage over the entire domain. The final observation is that these values are high for even small domains so we can increase coverage by decreasing domain size and still get high levels of unpredictability.

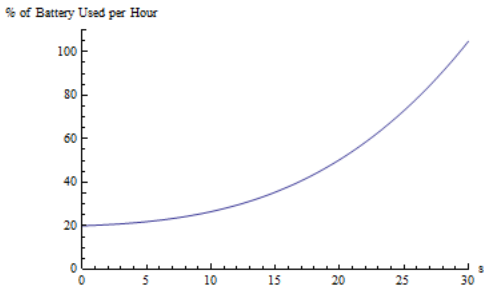
3 Battery Life

We know that the battery life of the MAV depends on the speed at which it is moving and the number of turns that it makes. We based our model of the amount of power used by our MAV on the Tesla Roadster. As the Tesla Roadster is completely battery powered, like our MAV, it is a reasonably good fit[8]. There are certain drawbacks to basing the model off the Roadster such as the car has greater air drag, and it is far heavier. However, because the batteries used are quite similar, both are lithium polymer, we can construct a reasonable model of the rate of power usage that depends on speed.

The formula of the usage of battery power we have the following function:

$$f(s) = 20 + 0.219651s + 0.0213974s^2 + 0.00218341s^3 \tag{1}$$

Where $f(s)$ is the percentage of battery life used in an hour and s is the speed at which the MAV is flying in miles per hour.



This function predicts that the MAV will lose power much faster at higher speeds and will last about 57.25 minutes at the top speed of 30mph.

When the MAV turns, it uses a significant amount of power but is unlikely to exceed the power usage of traveling at top speed for one long block in Manhattan. In order to have a value with which to test our model, we will estimate that a turn requires 0.2% of the battery.

4 Repairs

4.1 Probability That a MAV Breaks

The probability that the MAV breaks in flight is dependent on the speed at which it is flying. The probability that the ith drone fails at any given time is assumed to be binomial with probability p_i.

In the worst case, when all the drones are moving at the fastest speed, each p_i is at its maximum. When this is the case, we need to ensure that at most 30% of the drones fail (Using an assumption from the problem statement). We will limit the probability that at least 30% of the k MAV's fail to be below 1%. We can approximate the binomial distribution with a normal distribution since k will be reasonably large and all the p_i's are the same because all the random variables are equivalent. We denote the maximum value of p_i to be p_{max} and this occurs when the MAV's are moving at the fastest speed possible. When this is the case, we need to ensure that at most 30% of the drones fail. We will limit the probability that at least 30% of the k MAV's fail to be below 1%. Using the normal approximation:

$$P(\text{more than 30\% of MAV's fail}) = 1 - P\left(Z \geq \frac{.3n - kp_{max}}{\sqrt{np_{max}(1-p_{max})}}\right) = .01$$

$$P\left(Z \leq \frac{.3n - kp_{max}}{\sqrt{kp(1-p_{max})}}\right) = .99$$

$$\frac{.3n - kp_{max}}{\sqrt{kp_{max}(1-p_{max})}} = 2.326347877$$

$$p_{max} = \frac{1.79192 \times 10^{17} + 1.98665 \times 10^{16} n - 7.05966 \times 10^{16}\sqrt{6.44273 + n}}{3.58384 \times 10^{17} + 6.62216 \times 10^{16} n}$$

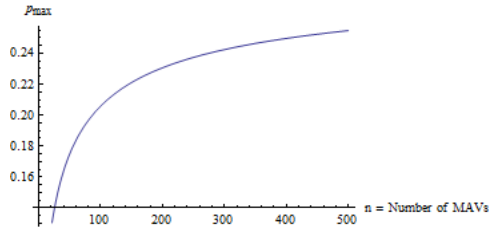


Figure 1: Graphing the possible values of p_{max}, which is dependent on k

The probability that a MAV will fail is based almost entirely on the speed at which it is going. Time is only a significant issue if the aircraft are not properly maintained or if the aircraft is operating longer than its designed lifetime [5][6][9]. Thus, we assume that the MAV's are properly maintained and so that time is not a factor in the failure rate.

When the MAV's speed is zero, we can conclude the probability that it breaks is zero since no forces due to motion are acting on it (also the vibration due to the spinning blades at this speed are well within design parameters). At the maximum speed the probability of the MAV failing is p_{max}. We assume that the probability that MAV fails at 10mph is .02. From these three values, we obtain a quadratic function that gives us the probability that the MAV will fail given the speed at which it is traveling. (Note: we discount the exponential or power functions because at 0mph we have 0 probability)

$$p(s) = \frac{50p_{max} - 3}{30,000}s^2 - \frac{50p_{max} - 9}{3,000}s \tag{2}$$

$p(s)$ is the probability that a MAV will fail and s is its speed.

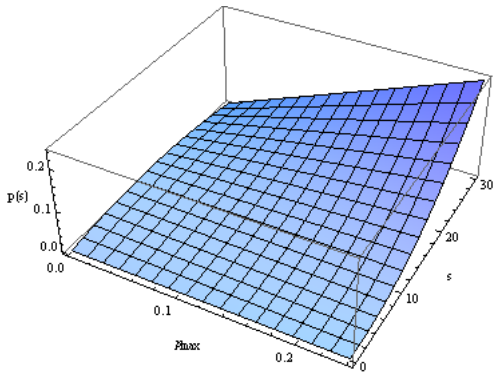


Figure 2: Graphing the possible values of $p(s)$ over varying values of p_{max} and s. Note that p_{max} is constant in the model because the number of MAV's is fixed

4.2 Repair Times of Broken MAV's

After a MAV fails, we assume that it is instantaneously sent to the central repair station. Since the service times between path cycles are assumed to be instantaneous, we can conclude that the repair times are quite short. Based on the repair times of large aircraft, we figure that 99% of the repair times will be completed within a half hour (we scaled for size). We assume that 1% of the repairs will take longer than this time because of catastrophic failures such as a jaywalker shooting down the MAV.

To obtain the desired distribution of the repair times, we decided that a logistic growth equation would be best because it is asymptotic to a given y-value. We also want the function to be a cumulative probability density function on the domain from $[0, \infty)$. We set this function to be

$$f(t) = \frac{2}{1 + e^{-2t \ln(199)}} - 1$$

because f is a cumulative probability density function and is less than .99 when $t < .5$.

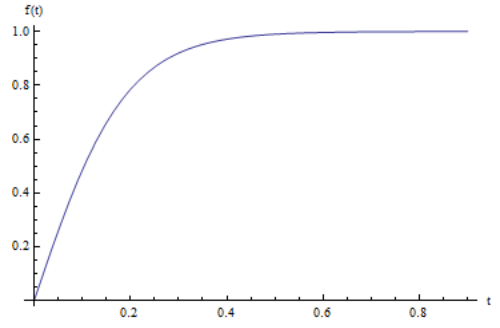


Figure 3: Graph of the cumulative probability density function

f, however, is a function of time and will not give us a random time with the distribution that we want. To get this distribution, we solve for t:

$$t = -\frac{\ln\left(\frac{2}{1+f(t)} - 1\right)}{2 \ln(199)} \tag{3}$$

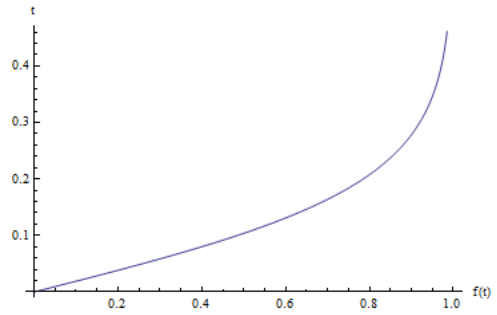


Figure 4: The graph of t with respect to $f(t)$

Since $f(t)$ goes from 0 to 1, we will randomly select a point between 0 and 1 for $f(t)$ to get a random time that it will take to repair the MAV. So the repair time of a MAV is:

$$t = -\frac{\ln\left(\frac{2}{1+rand(1)} - 1\right)}{2\ \ln(199)} \tag{4}$$

where $rand(1)$ is a uniformly random number between 0 and 1.

4.3 Strategy for Replacing Broken MAV's

Since the MAV's are old they will fail frequently. To counter this problem we need to have some extra drones that can be sent out automatically to replace broken drones.

4.3.1 Simple Approach

The most obvious strategy would be to have a central location (or a few locations) that stores drones and launches a replacement drone that flies out to the domain of the broken drone. The problem with this is that that domain will be un-patrolled for the time it takes to get from the central base to the domain. So we can imagine a jaywalker shooting down the drone in a domain so that they can jaywalk as much as they'd like.

4.3.2 Drone Shifting

Instead we propose a strategy that when a drone breaks down we find a chain of domains connected along edges that ends in the domain where the drone broke. It is in fact very easy to minimize this using the greedy algorithm where you always decrease the distance to the domain of the broken drone. Then we push drones along this path and get a battery from the base in the domain one further down. The reason we choose to replace the battery is to ensure that the replacement drone will be able to finish the cycle of the drone it is replacing. Finally, we have a new drone fly to the position where the previous drone ended. By changing the state of a new drone to the state of the previous drone in the domain we only have a short interruption instead of a long wait.

If we assume the battery base is in the center of the domain and that the domains are n by n, we know that wherever a drone is in its domain it will take $\leq \frac{3n}{2}$ moves to get aligned with the base

and $\leq \frac{n}{2}$ to get to the base for a battery. Then from the center it will take $\leq \frac{n}{2}$ to get where the other drone was either horizontally or vertically. Taking the sum we find that it will take at most $3n$ steps before the drones resume their paths in the domains of the replacement chain. So it will only take that small amount of time to replace broken MAV's.

5 Computational Breakdown Modeling

5.1 Computational Model

We modeled the scheme based on drone shifting to replace broken drones. We took a region of 10 square miles in a square and covered it with square domains that are of maximum size given the speed of the drone. For convenience we took blocks to be the same size to be $\frac{3}{2} \cdot 0.05$ miles; this is chosen such that the area of a block in the model is close to that of a real block in Manhattan. We also assumed that the stale time of all the domains was 15 minutes, and that in each domain the time taken for an Eulerian cycle is 15 minutes. We then selected multiple speeds (namely 10, 15, 20, 25, and 30 mph) and for each speed ran 200 simulations for $250 \cdot n^2$ steps. The hours spent repairing drones was computed using the probability distribution created earlier.

The mean number of drones needed was computed by first filling each domain with one drone. Then in the model we kept track of how many drones had been repaired and were not currently flying. Then whenever a drone broke if a repaired drone was sitting around it would be reused, otherwise if no drones were laying around then another MAV would have to be added to the collection. So this computes the number of drones needed given our model of breakdown rates and repair times.

5.2 Results

Speed	10	15	20	25	30
Mean Number of Drones	196.305	162.450	136.295	109.200	108.415
Mean Hours Spent Repairing Drones	18.0045	14.6512	16.2005	12.1194	11.2014
Mean Number of Breakdowns	138.085	111.055	122.755	93.240	85.255

This immediately suggests that we want to have the MAV's going as fast as possible. Since the average time spent repairing drones also decreases in any sort of cost model running the drones faster is better because the number of MAV's and repair time is minimized. Since Manhattan is roughly 20 miles in area, the computational model gives as a quick estimate of 217 drones.

The results closely agree with the analytical predictions. The number of breakdowns decreases as the speed increases because probabilities of failure are fairly close together at the high velocities. With the total number of breakdowns falling, it follows that total repair times decrease.

6 Applying the Model to Manhattan

6.1 Goals

How does our model work in the real world? As we will show, moving from the idealized $n \times m$ grid to the streets of Manhattan requires only minor modifications in our model. Let us begin by

looking at the section of Manhattan between 14^{th} and 116^{th} streets. Our goal is to

Provide tailored coverage across Traffic Sectors. Traffic volume through a neighborhood drastically varies depending on location and time of day. In order to simultaneously patrol the notoriously hectic streets in rush hour Midtown and the broad, placid avenues of the Upper East Side, we divide Manhattan into *Traffic Sectors*.

Maximize MAV efficiency. To avoid redundancy, we assign each MAV its own domain.

Minimize MAV malfunction. Our MAV's run at optimal speeds to minimize energy usage and MAV breakdown.

Minimize Predictability of UAV Behavior We implement a scheme based on Fleury's Algorithm, making it virtually impossible for even the most determined jaywalker to gain non-negligible advantage over the average citizen.

6.2 The Geography

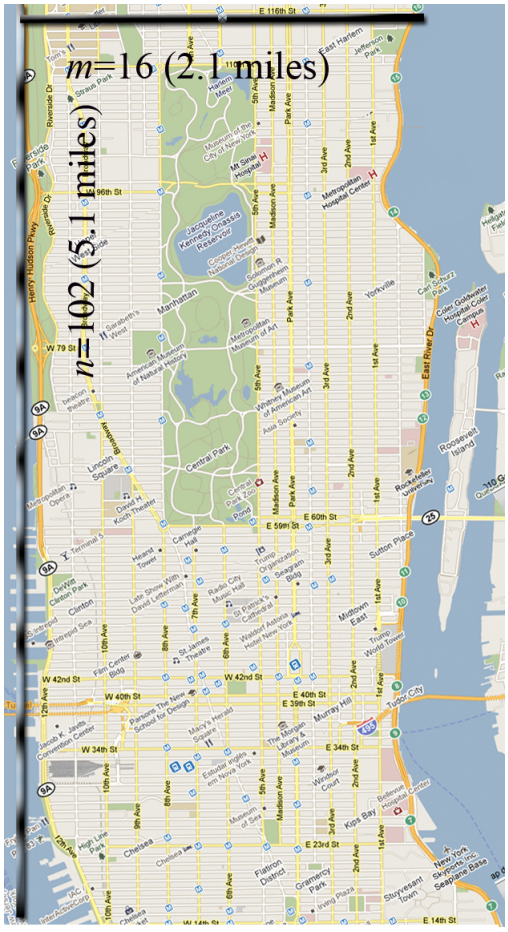


Figure 5: Manhattan from 14^{th} to 116^{th} Street. We model this region using a 103×16 multi-graph grid.

Our section of interest encompasses approximately 10.71 mi^2, has on average $m = 16$ vertical avenues and $n = 103$ horizontal streets. The width and height of a city block are approximately

$l_m = 0.1$ and $l_n = 0.05$ mi, so $l_m = 2l_n$. Hence, we choose to represent this geographical section with a103×16 grid. We will apply the model in two ways: first assuming fixed traffic volumes in each traffic sector, then relaxing this assumption to look at how we can repartition the MAV domains if presented with change in traffic volume.

6.3 Domains Within Static Traffic Sectors

We begin with the simplifying assumption that our geographic section exhibits static pedestrian traffic, that is, the traffic in each traffic sector remains constant over time. In our model, we had assumed square blocks. It might appear that the rectangular blocks would present us with a problem, however, that is not the case! It turns out that, since $l_n = .05$mi and $l_m = .1 = 2l_n$mi, we can idealize our geographical section to grids of $m \times n$ squares with side lengths $a = 1.5l_n$. Then, the perimeter of a city block, $2(l_n + l_m) = 2(l_n + 2l_n) = 6l_n$, is equal to that of a square, $4a = 6l_n$. Since we are interested in the linear distance traveled by the MAV, this approximation does not present a problem!

Definition. *A traffic sector* is a geographic section with a certain traffic volume. We refer to specific traffic sectors by giving their upper-left and lower-right coordinates and their *stale time*, t.

We segment our geographical section of Manhattan into three different **traffic sectors** based on pedestrian volume, orienting the 103×16 grid such that the the upper-left vertex is $(0, 0)$ and incrementing n in the southward direction[7]:

$$\begin{aligned}
S_1 &= \{(116^{th} \text{ Street, Riverside Drive}),\ (59^{th} \text{, East River Drive}), (t = 15)\} \\
&= \{(0, 0), (16, 58), (t = 15)\} \\
S_2 &= \{(59^{th} \text{ Street, Riverside Drive}),\ (23^{th} \text{, East River Drive}), (t = 5)\} \\
&= \{(0, 58), (16, 93), (t = 5)\} \\
S_3 &= \{(23^{rd} \text{ Street, Riverside Drive}),\ (14^{th} \text{, East River Drive}), (t = 10)\} \\
&= \{(0, 93), (16, 103), (t = 10)\}
\end{aligned}$$

See Figure 6 Below As we will see, the lower the t, the more surveillance a traffic sector will have. Furthermore, we partition each traffic sector into MAV domains with the requirement that each domain be efficiently patrol-able by a single MAV. We showed that optimally, the MAV's should be running at speed $s = 30$mph, so we program all MAV's to go at 30mph.

In order to guarantee that MAV's are always able to run Eulerian circuits, we want each of the non-boundary nodes to have degree 8, each corner node to have degree 2, and the remaining nodes to have degree 4. This means that one cycle completed by the MAV will trace each of the non-boundary edges twice and the boundary edges once. (This works out nicely because domains share boundaries, and the outermost boundaries, Riverside Drive and East River Drive are highways and need virtually no surveillance.) Hence, the total number of linear street distance $D(m, n, l_m, l_n)$ in

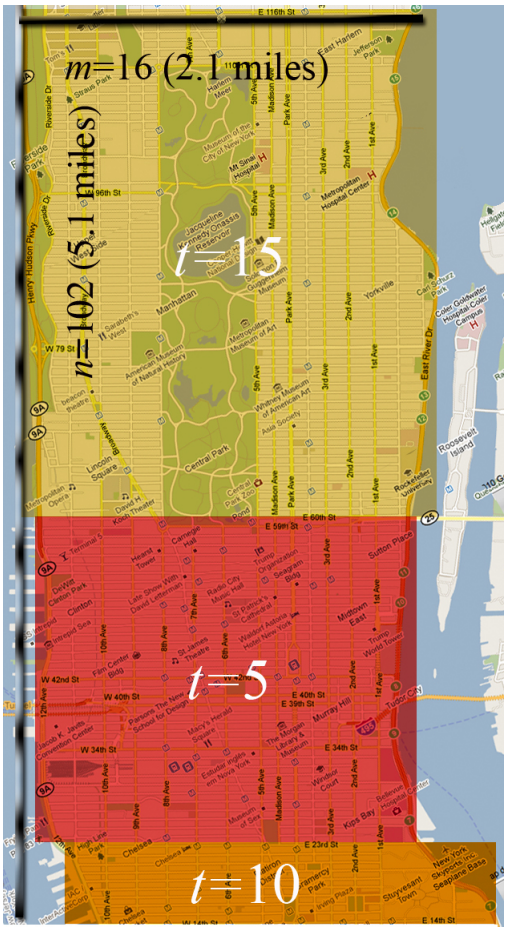


Figure 6: Manhattan divided into three different pedestrian traffic sectors

one cycle of a $n \times m$ grid is given by:

$$\begin{aligned} D &= 2l_m \cdot n(m-1) - 2l_m(m-1) + 2l_n \cdot m(n-1) - 2l_n(n-1) \\ &= 2l_m(m-1)(n-1) + 2l_n(m-1)(n-1) \\ &= 2(m-1)(n-1)(l_n + l_m) \end{aligned}$$

Calculating D for each traffic sector, we get:

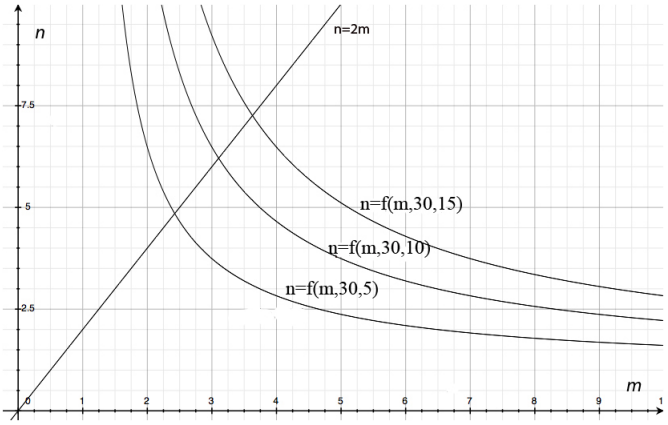
Traffic Sector	Stale Time	Speed	m	n	$l_m + l_m$	D_i
S_1	0.25 hr	30 mph	58	16	0.15 mi	256.5 mi
S_2	0.083 hr	30 mph	36	16	0.15 mi	157.5 mi
S_3	0.17 hr	30 mph	11	16	0.15 mi	45 mi

Note: We included Central Park in our estimates for S_1. We can obtain a cleaner estimate by defining smaller traffic sections around Central Park, but for now, we assume that in the year 2084, Central Park has been overlaid with roads.

For speed 30mph and stale time t, the distance that one MAV is able to observe in one hour is

given by

$$30 \geq 2 \cdot \frac{60}{t}(m-1)(n-1)(l_n + l_m)$$



In table form, the possible dimensions (x, y) along with the distance covered of the MAV domains are:

S_1	D	S_2	D	S_3	D
~~$(2,6)$~~	1.5	~~$(2,5)$~~	1.2	~~$(2,11)$~~	3
$(3,6)$	3	$(3,3)$	1.2	$(3,6)$	3
$(4,4)$	2.7	~~$(4,2)$~~	0.9	$(4,4)$	2.7
~~$(17,2)$~~	4.8	–	–	~~$(11,2)$~~	3

Notice that the long and skinny domains optimize the distance patrolled per Eulerian circuit; however, if m or $n = 2$, the domain would consist of only one row or column of blocks. Then, our MAV would have to make many turns, and its path will be noticeably more predictable since Fleury's algorithm would only produce two Euclidean circuits.

In order to maximize MAV efficiency, we assign domains as follows: within a traffic sector, choose the pair (x_0, y_0) with the largest D value. Moving (first one, then the other) in the m (lateral) and n (vertical) direction, assign the domain of dimension (x_0, y_0) over and over until all the space is covered. If there is leftover grid space, check if it can be covered by the domain of size (x_1, y_1) (if an alternative is available). If not, overlap the domains so that all the grid space is covered by at least one domain.

Let's do a real example with S_2, which has $n_2 = 36$, $m_2 = 16$. Since $(x_0, y_0) = (3, 3)$, we divide n_2 by 3, telling us that we need 12 domains in the vertical direction. Nice. Dividing m_2 by 3, it turns out we need 6 domains in the lateral direction. We can arrange for the overlap to provide double coverage in the busiest blocks. Hence, in total, we need $12 \cdot 6 = 72$ MAV's for S_2.

Repeating the calculations for S_1 and S_3, we need a total of 144 MAV's operating at any given time. This number, however, is only 70% of the total number of MAV's owned since we need to be able to absorb a 30% failure rate. Therefore, this model predicts that approximately 206 drones are required to patrol Manhattan.

6.4 Re-assigning Domains Under Dynamic Traffic

If for instance, a parade shuts down 5^{th} Ave., redirecting traffic into surrounding streets, then our assumption of static traffic sectors is rendered invalid. So how can we re-assign domains quickly to tailor MAV surveillance to dynamic traffic patterns?

As before, we would lay down traffic sectors, then place domains with highest D value contiguously until all of the traffic sector is filled. We can always do this, since our MAV's are tracked and controlled by the central control station.

7 Limitations of the Model

In order to make our model feasible we made several assumptions that simplified the problem. One such assumption was that the MAV can only look straight ahead and can only see one block ahead. This is a highly conservative assumption, as it is likely that MAV's could be configured to look in all directions and detect jaywalkers a considerable distance away. Such a model would likely require fewer MAV's and thus, minimize costs even more than we did in our current model. We also assumed that the drones all fly at a constant altitude, something that could likely be altered to gain even more efficiencies. Our model of repairing the MAV's assumes that it only fails at street corners and that when it fails it is instantly transported to the repair station, which rather unrealistic. The model was also not tested on a complete street map of Manhattan. We constructed a model for the rate at which the battery is discharged but did not incorporate it into the final model. We are aware of all these limitations and would address them if we had more time.

8 Conclusion

Using Fleury's Algorithm to program the MAV's flight-plan returns the best results in terms of unpredictability and even coverage. Unlike the predetermined paths and random walks, our simulations suggested that Fleury's Algorithm would prevent observant jaywalkers with *a posteriori* knowledge from predicting the locations of the MAV's. In the case of MAV malfunction, our drone-shifting scheme will effectively replaces the broken MAV with only a short interruption in normal surveillance. Hence, we recommend Fleury's Algorithm and the drone-shifting scheme as a strong and effective model.

Having the MAV's patrol the streets at top speeds will minimize the cost of the operation. With the MAV's flying a their maximum speed, we use fewer drones and minimize the total repair time. As the cost of the drones and the labor to repair broken ones are the major components of the operating budget, minimizing them effectively minimizes cost.

9 Appendix

9.1 Source Code - Programmed for MATLAB

9.1.1 breakdown_probability.m

```
function prob = breakdown_probability (s, p_max)
  % computes breakdown probability as a function of speed and p_max

  prob = (50 * p_max - 3) / (30000) * s^2 - (50 * p_max - 9) / (3000) * s;
```

9.1.2 city_breakdown_simulation.m

```
function [undercoverage num_drones diwakars_number num_breakdowns] =
city_breakdown_simulation (n, m, len, s, steps)
  % simulate a section of city using the probability of breakdown and repair time
  % return the amount of undercoverage caused by breakdowns and the number of drones
  % needed to account for breakdowns and repair times
  % n is the side length of each domain, the area covered is m by m domains, len is the
  % length of an edge in miles, s is the speed, mph, run the simulation for steps steps

  num_drones = m * m;

  undercoverage = zeros (m, m);

  drones_available = zeros (steps + 1, 1);

  p_max = compute_p_max (num_drones);

  p = breakdown_probability (s, p_max);

  step_t = s / len; % in hours

  diwakars_number = 0;

  num_breakdowns = 0;

  for k = 1 : steps
    broken = (rand (m, m) < p);
    broken_index = find (broken);
    for b = broken_index
      if drones_available (k)
        drones_available (k) = drones_available (k) - 1;
      else
        num_drones = num_drones + 1;
      end
```

```
      i = ceil (b / m); % row
      j = b - (i - 1) * m; % column

      path = [(ones (i - 1, 1)); (zeros (j - 1, 1))];
      path = path (randperm (i + j - 2));

      ip = 1;
      jp = 1;
      undercoverage (ip, jp) = undercoverage (ip, jp) + 2 * n;
      for p = path
        if p
          ip = ip + 1;
        else
          jp = jp + 1;
        end
        undercoverage (ip, jp) = undercoverage (ip, jp) + 3 * n;
      end

      repair_t = repair_time;
      diwakars_number = diwakars_number + repair_t;
      repair_t = ceil (repair_t / step_t);
      if k + repair_t < steps
        drones_available (k + repair_t) = drones_available (k + repair_t) + 1;
      end
      num_breakdowns = num_breakdowns + 1;
    end
    drones_available (k + 1) = drones_available (k) + drones_available (k + 1);
  end

  undercoverage = undercoverage * step_t;
```

9.1.3 city_simulation.m

```
% city_simulation.m
% run city simulation with many different speeds and print the speed followed by useful
% information from the run

number = 5;

speeds = [10; 15; 20; 25; 30]; % speeds choosen for ease
n_vals = [3.89; 4.54; 5.08; 5.57; 5.999]; % values of n corresponding to each speed

len = 3 / 2 * 0.05;

c = 10; % arbitrarily look at 10 square miles
```

```
m_vals = ceil (sqrt (c ./ (len * n_vals) .^ 2));

steps = 250 * (n_vals .^ 2);

for i = 1 : number
  speeds (i)
  for j = 1 : 200
    [undercoverage, num_drones (j), diwakars_number (j) breakdowns (j)] = ...
      city_breakdown_simulation (n_vals (i), m_vals (i), len, speeds (i), steps (i));
  end
  mean (num_drones)
  mean (diwakars_number)
  mean (breakdowns)
end
```

9.1.4 compute_p_max.m

```
function p_max = compute_p_max (n)
  % uses p_max computation outlined in the paper

  p_max = (1.79192 * 10^17 + 1.98665 * 10^16 * n - 7.05966 * 10^16 * sqrt(6.44273 + n));
  p_max = p_max / (3.58384 * 10^17 + 6.62216 * 10^16 * n);
```

9.1.5 eulerian_walk_simulation.m

```
function v = eulerian_walk_simulation (n)
  % simulate an arbitrary eulerian walk on an n by n grid graph starting at floor
  % ((n ^ 2 + n)/2)
  % v tracks when an edge is used, if j and k are connected by two edges then
  % j < k -> v (j, k) < v (k, j)
  % and k < j -> v (k, j) < v (j, k), this increases the correlation of the resultant data

  v = zeros (n ^ 2, n ^ 2);

  t = graph_transformation (n);

  t = t * 2;

  for i = 1 : n ^ 2
    t_i = t (:, i);
    if size (find (t_i)) (1) < 4
      for j = find (t_i)
        if j < n || j > (n ^ 2 - n) || mod (j, n) <= 1
          t (j, i) = t_i (j) / 2;
        end
      end
```

```
  end
end

s = zeros (n ^ 2, 1);
start = floor ((n ^ 2 + n) / 2);
s (start) = 1;

x = 1;
while size (find (t)) (1)
  j = find (s);
  % j is the index of the vertex you are at

  s1 = t * s;
  % s1 is the set of vertices you can go to

  f = find (s1);
  % f is the set of indices of vertices you can go to

  if size (f) (1) == 0
    error ('disconnected');
  end

  i = f (ceil (rand (1) * (size (f) (1))));
  % choose a vertex to go to at random

  t1 = t;
  t1 (i, j) = t1 (i, j) - 1;
  t1 (j, i) = t1 (j, i) - 1;
  % t1 is the graph after you went from j to i

  tn = s;
  for k = 1 : n ^ 2
    tn = t1 * tn;
  end
  % tn is the number of ways each vertex is reachable from s in n ^ 2
  % moves in the modified graph t1

  ttn = t1 * tn;
  % ttn is the number of ways each vertex is reachable from s in n ^ 2 + 1 moves

  % if going over the edge you chose doesn't disconnect the graph or you have no choice
  % then cross that edge
  if (tn (start) ~= 0) || (ttn (start) ~= 0) || (size (f) (1) == 1)
    s (j) = 0;
    s (i) = 1;
    % go to node index i
    t = t1;
    % edit the graph
```

```
      if j < i
        % choose order to increase correlation in v values
        temp = i;
        i = j;
        j = temp;
      end
      if v (i, j)
        v (j, i) = x;
      else
        v (i, j) = x;
      end
      x = x + 1;
      % track stuff with v
    end
  end
```

9.1.6 graph_transformation.m

```
function a = graph_transformation (n)
  % return a transformation from the vector space of the vertices to itself
  % the set of vertices is a n by n graph in the city type description
  % this transformation covers all possibilities for following edges in the graph

  a = zeros (n^2, n^2);
  a1 = zeros (n^2, n^2);
  a2 = zeros (n^2, n^2);

  r = zeros (n, n);
  r1 = zeros (n, n);
  r2 = zeros (n, n);

  r1 (2 : n, 1 : n - 1) = eye (n - 1);
  r2 (1 : n - 1, 2 : n) = eye (n - 1);
  r = r1 + r2;

  a1 (n + 1 : n ^ 2, 1 : n ^ 2 - n) = eye (n ^ 2 - n);
  a2 (1 : n ^ 2 - n, n + 1 : n ^ 2) = eye (n ^ 2 - n);
  a = a1 + a2;

  for i = 0 : n - 1
    a (n * i + 1 : n * i + n, n * i + 1 : n * i + n) = r;
  end
```

9.1.7 many_eulerian_simulate.m

```
function many_eulerian_simulate (nlist, k)
```

```
  % run k eulerian_walk_simulations on each n in nlist
  % for each n in nlist print n then print the mean of the average time
  % visited for each edge, then the standard deviation of the average time
  % visited for each edge

  if k < 1
    k = 1;
  end

  for n = nlist
    n
    vt = eulerian_walk_simulation (n);
    v = vt (find (vt));
    for i = 2 : k
      vt = eulerian_walk_simulation (n);
      v = v + vt (find (vt));
    end
    v = v / k;
    mean (v)
    std (v)
  end
```

9.1.8 many_random_simulate.m

```
function many_random_simulate (list)
  % run 20 random_walk_simulation (10, i) for each i in list
  % for each i in list print i, the mean number of visits for each edge,
  % and the standard deviation for the list of the number of visits on each edge

  for i = list
    i
    vt = random_walk_simulation (10, i);
    v = vt (find (vt)) - ones (360, 1);
    for j = 2 : 20
      vt = random_walk_simulation (10, i);
      v = v + vt (find (vt)) - ones (360, 1);
    end
    v = v (find (v));
    v = v / 20;
    t = size (v);
    if t (1) < 180
      v (180) = 0;
    end
    mean (v)
    std (v)
  end
```

9.1.9 random_walk_simulation.m

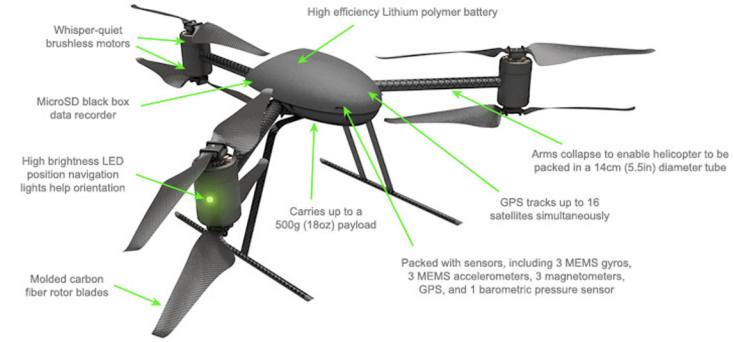
```
function v = random_walk_simulation (n, steps)
  % simulate a random walk on a n by n grid of roads for steps steps
  % v is a matrix such that if i < j then v (i, j) is the number of
  % times the walk went from j to i or i to j plus one, or zero if i is not adjacent to j
  % ignore if i >=j

  t = graph_transformation (n);

  z = zeros (n ^ 2, 1);
  s = z;
  s (floor ((n ^ 2 + n) / 2)) = 1;

  v = t;

  for i = 1 : steps
    s_prime = t * s;
    k = find (s);
    f = find (s_prime);
    if size (f) (1) == 3
      f (4) = f ((mod (f, n) ~= 1) && (mod (f, n) ~= 0) && (f > n) && (f < (n ^ 2 - n)));
    end
    m = f (ceil (rand (1) * (size (f) (1))));
    s = z;
    s (m) = 1;
    if m > k
      temp = m;
      m = k;
      k = temp;
    end
    v (m, k) = v (m, k) + 1;
  end
```

9.1.10 repair_time.m

```
function x = repair_time
  % calculate a repair time in hours
  x = -1 * log (2 / (1 + rand (1)) - 1) / (2 * log (199));
```

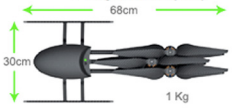
9.2 Dragonflyer X6 Helicopter Technical Specifications

Draganflyer X6 Helicopter Tech Specs



Helicopter Size

- **Dimensions**
  - Width: 91cm (36in)
  - Length: 85cm (33in)
  - Top Diameter: 99cm (39in)
  - Height: 25.4cm (10in)
- **Dimensions**
  (Without Rotors)
  - Width: 55cm (22in)
  - Length: 49cm (19in)
  - Top Diameter: 63cm (25in)
  - Height: 25.4cm (10in)
- **Folded Dimensions**
  - Width: 30cm (12in)
  - Length: 68cm (27in)
  - Front Diameter: 35cm (14in)
  - Height: 25.4cm (10in)



- **Folded Dimensions**
  (Without Landing Gear)
  - Width: 13cm (5in)
  - Length: 66cm (26in)
  - Front Diameter: 14cm (5.5in)
  - Height: 13cm (5in)
- **Folded Dimensions**
  (Without Rotors or Landing Gear)
  - Width: 13cm (5in)
  - Length: 51cm (20in)
  - Front Diameter: 14cm (5.5in)
  - Height: 13cm (5in)

Weight & Payload

- Helicopter Weight: 1,000g (35oz)
- Payload Capability: 500g (18oz)
- Maximum Gross Take-Off Weight: 1,500g (53oz)

Flight Characteristics:

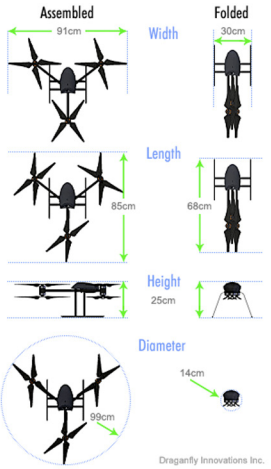
- Unassisted visual reference required
- Max Climb Rate: 7m/s (23ft/s)
- Max Descent Rate: 4m/s (13ft/s)
- Max Turn Rate: 90°/second
- Approx Maximum Speed: 50km/h (30mph)
- Minimum Speed: 0km/h (0mph)
- Launch Type: VTOL (Vertical Take Off and Landing)
- Maximum Altitude ASL: 2,438m (8,000ft)
- Maximum Flight Time: Approx. 20 min (without payload)
- Approx Sound at 1m (3.28ft): 65dB
- Approx Sound at 3m (9.84ft): 60dB

Rotor Blades

- Three Counter-Rotating Pairs (Six Rotors Total)
- Rotor Blade Material: Molded Carbon Fiber
- Upper Rotor Diameter: 40cm (16in)
- Lower Rotor Diameter: 38cm (15in)

Electric Motors

- Brushless Motors: 6
- Configuration: Direct Drive (One Motor per Rotor)
- Safety Features: Stall Protection
- Ball Bearing
- Voltage: 14.8V



RF Communications

- **2.4GHz Data Link**
  - Link Type: Helicopter to Ground & Ground to Helicopter (Two-Way)
  - Helicopter Antenna: Wired Whip Antenna
  - Controller Antenna: Omni-Directional
  - RF Data Rate: 250kbps
  - Receiver sensitivity: -100dBm
  - Transmission Technique: DSSS (Direct Sequence Spread Spectrum)
  - Frequency band: 2.4000 - 2.4835 GHz
  - Certifications: CE, FCC, IC, ETSI
  - Data Link Channel Selection: Automatic (13 Channels)
- **5.8GHz Video Link**
  - Link Type: Helicopter to Ground (One-Way)
  - Transmitter Antenna: Omni-Directional
  - Receiver Antennas: Omni-Directional & Flat Patch
  - Transmission Power: 12dBm
  - Transmitter Power Consumption: 500mW
  - NTSC and PAL Compatible
  - 7 Selectable Channels: 5740MHz, 5760MHz, 5780MHz, 5800MHz, 5820MHz, 5840MHz, 5860MHz
  - Range varies with enviroment

11 Onboard Sensors

- 3 Solid State MEMS (Micro-Electro-Mechanical Systems) Gyros
- 3 Solid State MEMS (Micro-Electro-Mechanical Systems) Accelerometers
- 3 Magnetometers (Magnetoresistive Sensors)
- 1 Barometric Pressure Sensor
- 1 GPS Receiver
  - GPS Battery Backup: 75mAh Lithium Polymer

GPS

- GPS Used For: Position Hold, Location & Velocity Data
- Maximum Satellites Tracked Simultaneously: 16
- Position Update Rate: 4 Hz
- GPS Antenna: Ceramic Patch
- Battery Backup: Lithium Polymer

Camera Attachments

- 10MP (Mega-Pixel) Digital Still Camera with Remote Controlled Tilt, Zoom & Shutter
- 1080p HD (High Definition) Video Camera with Remote Controlled Tilt
- Thermal FLIR (Forward Looking Infra-Red) Camera with Remote Controlled Tilt & onboard digital video recorder
- Low Light (0.0001 lux) Dusk/Dawn Black & White Video Camera with Remote Controlled Tilt & onboard digital video recorder
- Micro Analog Color Video Camera Video Camera with Remote Controlled Tilt & onboard digital video recorder

Position Navigation Lights

- Type: 1 Watt LED Variable Brightness Emitters
- Luminous Flux at Full Brightness: 40lm
- Purpose: Helicopter Orientation Confirmation
- Visible Condition Range: Full Darkness to Direct Sunlight
- Standard Aircraft Colors
  - Red: Left
  - Green: Right
  - White: Tail/Rear

Rechargeable Helicopter Battery

- Cell Chemistry: Lithium Polymer
- Voltage: 14.8V
- Capacity: 2700mAh
- Length: 7.0cm (2.8in)
- Width: 6.7cm (2.6in)
- Height: 2.7cm (1.0in)
- Connectors: Integrated Balance and Power
- Recharge Time: Approx. 30 minutes (after typical flight)

Landing Gear

- Installed Height: 18cm (7in)
- Stance Width: 30cm (12in)
- Skid Length: 30cm (12in)
- Landing Gear Material: Molded Carbon Fiber

Materials

- Carbon Fiber
- Glass Filled Injected Nylon
- Aluminum & Stainless Steel Fasteners
- RoHS Compliant

Operating Requirements

- Recommended Pre-Use Temperature: 10° to 35°C (50° to 95°F)
- Maximum Environmental Operating Temperature: -25° to 38°C (-13° to 100°F)
- Relative Humidity: 0% to 90% Noncondensing
- Maximum Windspeed: 30km/h (18mph)
- Windspeed Recommended for Novice Pilots less than 16km/h (10mph)
- Windspeed for Optimal Video Recording using GPS Position Hold less than 10km/h (6 mph)

Flight Data Recorder

- Flight Data Recording: On-Board
- Stored To: Removable 2Gb MicroSD Memory Card
- Data Recorded: Onboard Sensor Flight Data (Link quality, Orientation, Altitude, Speed, Direction)

References

[1] Brightwell, Graham R., and Peter Winkler."Counting Eulerian Circuits is #P-Complete." *Proc. 7th ALENEX & 2nd ANALCO*, 2005: 259-262.

[2] "Detailed Findings for Boroughs and Community Districts." *Fund for the City of New York Website*. 1998. http://www.fcny.org/ cmgp/streets/pages/1998PDF/Report/DFMN.pdf (accessed November 13, 2010).

[3] Draganfly Innovation Inc. *Draganflyer X6 UAV Helicopter Aerial Video Platform*. 2010. http://www.draganfly.com/uav-helicopter/draganflyer-x6/ (accessed November 13, 2010).

[4] *Fleury's Algorithm*. 2005. http://www.austincc.edu/powens/+Topics/HTML/05-6/05-6.html (accessed November 13, 2010).

[5] Free Online Private Pilot Ground School. *Load Factors*. 2006. http://www.free-online-private-pilot-ground-school.com/Load_factors.html (accessed November 13, 2010).

[6] Kroo, Ilan. *Aircraft Structural Design*. 2010. http://adg.stanford.edu/aa241/ structures/structuraldesign.html (accessed November 13, 2010).

[7] Metropolitan Transportation Authority of the State of New York. "Pedestrian Traffic." *MTA Website*. n.d. http://www.mta.info/capconstr/sas/documents/deis/chapter_9h.pdf (accessed November 15, 2010).

[8] Tesla Motors Inc. *Roadster Efficiency and Range*. December 2008. http://www.teslamotors.com/blog/roadster-efficiency-and-range (accessed November 13, 2010).

[9] United States Army. *Principles of Helicopter Flight*. 1991. http://www. cavalrypilot.com/fm1-514/Ch1.htm (accessed November 13, 2010).

[10] van Lint, J. H., and R. M. Wilson. *A Course in Combinatorics*. Cambridge: Cambridge University Press, 2001.