Modeling the Disappearance of MH370

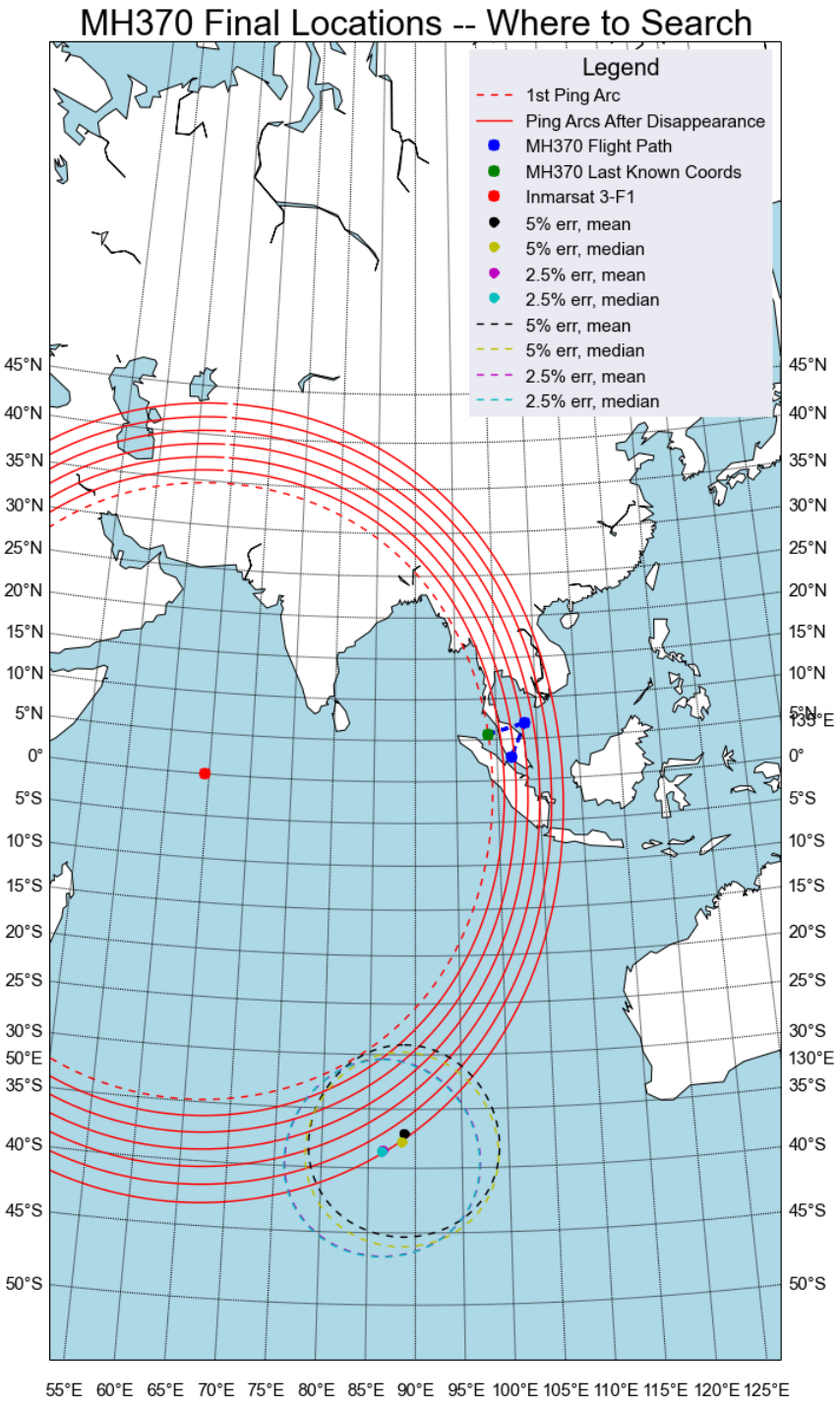

Figure 1- Part 1, Version 2. Monte Carlo Model results: Most likely locations of MH370 are within the dashed circles.

Table of Contents

Control + click on page number to go to section

Introduction

Malaysia Airlines 370 disappeared. A variety of information sources have been obtained, which can clue us into its possible locations. We have scoured the Internet and reached out to a variety of information experts in an effort to get inputs.

Our AM207 final project is divided into two parts.

Part 1

First, we use a Bayesian approach and approximation joint probability distribution setup to make a Markov Chain Monte Carlo (MCMC) model using a computationally efficient scenario specific algorithm, to identify the most likely final destination of the plane. We use, and are constrained by, the available MH370 information in order to do this. We also show the setup of a Metropolis-Hastings (M-H) approach, which we do not carry out computationally, to demonstrate how it could be done, and consequently how it differs from what was performed.

Part 2

In the first two weeks, it was unclear whether MH370 had gone in the ocean or had landed at many of the thousands of runways that a Boeing 777 could successfully land on in its hypothesized maximum range (~6.5 hrs flight time). To use more sophisticated techniques introduced later in AM207 course material, we examine redirection scenarios with a Hidden Markov Model (HMM), while abstracting specifics of MH370. For example, identifying scenarios where the plane was heading toward a runway in range, or one of several cities, from the Igari Waypoint (where it deviated from its normal flight path).

Part 3

Finally, we filter out noise from the heading changes of the scenarios scrutinized in Part 2, by implementing a Kalman Filter, and demonstrate the generality of the function we wrote with a separate example.

Part 1: Monte Carlo Model

Data

The data we used:

- The plane's point of deviation from its normal flight path from Kuala Lumpur to Beijing (Igari Waypoint).
- Its last known location known to Indonesia military radar (Palau Perak).
- The heading between them (~255º, where 0º is due north and 270º is due west).

and

- A series of pings relayed through an Inmarsat geostationary satellite which put constraints on the radial distance the plane was, every ~1 hr for 6+ hrs after disappearance, in relation to the satellite.
- Errors associated from those pings based on the technical analysis of the electronic handshakes established between the plane and the satellite. (2.5% or 5% of the radius, depending on if the calculation was done one-way or round trip. We are conservative and take the distance of 2.5% and 5% of the last ping radius, instead of using progressively less error for earlier, closer pings.)

Methods

Our conceptual setup:

Bayesian Approach

Bayes Theorem can be expressed as:

$$P(A|X) = \frac{P(X|A)\,P(A)}{P(X|A)\,P(A) + P(X|{\sim}A)\,P(\sim A)}$$

Model Parameters

$a_{\text{next}} = \{Long_{\text{next}}, Lat_{\text{next}}\}$ Next position of plane

$a_{\text{prev}} = \{Long_{\text{prev}}, Lat_{\text{prev}}\}$ Previous position of plane

h_{next} Next heading of plane

h_{prev} Previous heading of plane

s The point $\{Long_{\text{sat}}, Lat_{\text{sat}}\}$ directly beneath the geostationary satellite

A Proposition that the plane is next at a_{next}

X Proposition that the ping timing is the given one for this step

$P(A)$ Probability of being at a_{next} given the previous state. This is zero everywhere except the points along an arc a distance $D\left(\frac{plane\ speed}{time} * time\right)$ away from a_{prev}. Along the arc, the probability is given by a probability distribution (Gaussian, Von Mises, Wrapped Cauchy) based on h_{prev}.

$P(X|A)$ The probability that the ping timing occurs for a plane at a_{next}. This is a Gaussian (normal) distribution in the radius from the point underneath the geostationary satellite, with the value of 2.5% or 5% of the radius, as the standard deviation of the distribution; our error.

$P(X|A)\ P(A) + P(X|{\sim}A)\ P({\sim}A)$ The denominator of Bayes Theorem; essentially, the probability that we measure a ping at all. In the case of the plane, we can always detect the pings, so this is essentially just a normalization factor.

Procedure in Pseudocode

As a result, the probability at each point a_{next} is

$$P(A|X) = P(X|A)\ P(A)$$

In practice this is done by creating a list of 360 discrete points, one for each integer degree heading h_{next}. Here is the overall procedure as written in Python:

- Make a list of 360 points a on an arc distance D away from a_{prev}
- Make a list of 360 headings h
- For each point a
    - Calculate $P(X|A)$ – the normal distribution based on the Haversine distance from s to a
    - Calculate $P(A)$ – the heading distribution along the arc (Gaussian, Wrapped Cauchy, Von Mises) based on h_{prev}
    - Multiply to make w, the weighted likelihood of each point
- Normalize the weights to make probabilities $p = \frac{w}{\sum w}$ so that $\sum p = 1$
- This gives a vector of 360 probabilities. Essentially, $p = P(A|X)$
- Select one of the points using numpy.random.choice
- This also determines the new heading h_{next}, and new point a_{next}

Model Setup

In this first model we want to sample a probability distribution p over the set of possible locations for the plane at each ping point in time. The plane starts in a known position (Pulau Perak). After 56 min the first ping occurs. Thereafter at ~60 min intervals there are 6 subsequent pings after the plane is lost to radar. There is no 7th ping after disappearance. For each ping, there is a distance from the point s directly underneath the geostationary satellite. This distance has between 2.5% and 5% error which we assume is normally distributed. We assume that the plane flies at a constant speed (0.84 Mach, Boeing 777 cruising speed).

We do not know which headings (direction) the plane flies in. The heading (course) could change or drift due to many factors:

1. Wind. The autopilot corrects this in its typical modes of operation, but not others.
2. The plane could be flying on headings derived from constant magnetic compass headings, which vary at extreme latitudes (i.e. far north or far south).
3. The plane could change course due to manual input (i.e. someone flying it).
4. The plane could change course due to automated input (i.e. someone programmed a course change in the flight management system).

So we need to calculate $p_1, p_2, p_3, p_4, p_5, p_6$ where each corresponds to the probability of each point on earth where the plane could be at each of these after-disappearance pings. These ps are path dependent. p_1 depends on p_0, and so forth.

Metropolis-Hastings Setup

Here we show how we would setup a Metropolis-Hastings approach.

If the initial position is $Long_0, Lat_0$, then p_1 is a function f of the following:

$$p_1 = f_1(\, Long_0, Lat_0, Long_1, Lat_1)$$

A graph of the result of $p_1$would look like a ring centered at the plane's last known position across Earth, with radius d_0 (d, being the distance the plane can go in an hour). It will be zero essentially anywhere except for that ring.

$$p_2 = \int_{Lat_1} \int_{Long_1} f_2(\, Long_0, Lat_0, Long_1, Lat_1, Long_2, Lat_2)\; dLong_1\; dLat_1$$

$p_2$is the probability distribution over $Long_2, Lat_2$ and so forth, like the preceding example. $dLong_1\; dLat_1$ is instead of the typical $dx\; dy$ differential naming convention that we are familiar with in an introductory multivariable calculus class with x and y variables. Notice that $Long_2, Lat_2$ are in the equation but we're not integrating over them. We have to sum up the probabilities of the first hop, multiplied by the probabilities of the second hop, at each location to

get the complete picture of where the plane most likely is after 2 hours. (In principle, we're summing over the entire Earth, but in practice, we don't because it's still zero most places – see subsequent section.)

$$p_3 = \int_{Lat_2}\int_{Long_2}\int_{Long_1}\int_{Long_1} f_3(\,Long_0, Lat_0, Long_1, Lat_1, Long_2, Lat_2, Long_3, Lat_3)\; dLong_1\; dLat_1 dLong_2\; dLat_2$$

In p_3, we integrate over all possible first hops and all possible second hops. We follow a similar setup for all subsequent ps until we get to:

$$p_6 = \iint\iint\iint\iint\iint f_6(\,Long_0, Lat_0, \ldots, Long_6, Lat_6)\; dLong_1\; dLat_1 \ldots dLong_5\; dLat_5$$

Which is a ten dimensional integral over latitude and longitude for pings 1 through 5.

Each probability depends on all of the previous ones because for each function f:

$$f_j(\ldots Long_i, Lat_i, Long_j, Lat_j \ldots) = \begin{cases} 0 & \text{if } H(Long_i, Lat_i, Long_j, Lat_j) \neq d_j \\ ping(Long_j, Lat_j) & \text{if } H(Long_i, Lat_i, Long_j, Lat_j) = d_j \end{cases}$$

Where m = Mach number, M = speed of sound, the distance d is given by

$$d_0 = \left(\frac{56}{60}\right) m\, M$$

$$d_i = m\, M \;\text{ for } i > 0$$

Where H is the Haversine distance formula between two latitude and longitude points on Earth and the d_i are the distances the plane can fly for the interval in between pings.

The ping probability, $ping$, for each ping arc is:

$$ping(Long_j, Lat_j) = \frac{1}{2\pi\sqrt{\sigma}} e^{-\frac{1}{2}\left(\frac{H(Long_{sat}, Lat_{\text{sat}}, Long_j, Lat_j) - \mu_j}{\sigma}\right)}$$

Where H is the Haversine distance formula between two latitude and longitude points on Earth, σ is the standard error in the ping measurement, $ping$ is the ping probability for each ping arc (modeled as 2D Gaussian), and μ_j is the ping distance for the jth ping.

One way to state the problem is that we want to find p_6 and then extrapolate from there where the final resting place is. We could in principle do the 10 dimensional integral. It's ten dimensions because we integrate over all possible intermediate positions at each of the previous hops.

However, because the distance is zero except on a ring, instead of doing all possible latitude and longitude combinations in our implementation, we took a heading distance which basically selects a point along the circle, and disregards the rest of the Earth. So we equivalently formulate p_6 as a 5 dimensional integral over all possible intermediate headings. By doing this, we're avoiding doing an accept-reject method. The integrand will be zero everywhere except the ring, d_j. By formulating in terms of the headings, which takes into account our hourly distance plane constraint, we replace the two parameters latitude and longitude, with one parameter, heading.

We could still formulate an M-H approximation to the 5 dimensional integral. In that regard we can use the proposal distribution q with headings h_i, h_j :

$$q\left(Long_j, Lat_j\right) = \begin{cases} \rho(h_i, h_j) & for\ H\left(Long_i, Lat_i, Long_j, Lat_j\right) = d_j, \\ 0 & \text{otherwise} \end{cases}$$

This selects points on an arc a Haversine distance d_j from the previous point $Long_i, Lat_i$, and where the new heading h_j is prioritized by a Gaussian, Von Mises or Wrapped Cauchy distribution ρ. If choose the right parameter for these distributions, it's a uniform distribution around the arc, and then we can use priors to accept/reject after the fact.

Joint Probability Implementation – Our Actual Approach

Instead, we pick a parameter such that the parameter prioritizes points which are closer to maintaining the same heading. A random walk would eventually sample all the points, but then you'd have to filter down afterward to get, what you can set, from the get-go – paths with less of a degree of course corrections, i.e. heading changes. But doing this directly also would also be expensive, even though reducing to 5 is better than 10.

A different interpretation is that really, it's the ping that's the fundamental thing we're after – it's our acceptance distribution – while the proposal distribution is our heading. This proposal distribution eliminates most of the points we would otherwise need to consider, which would fail the distance test (i.e. that the point must lie on the arc). The target distribution is then given by the ping probability $ping\left(Long_j, Lat_j\right)$.

Our implementation directly generates the points the plane could be at each next step (360 – around a circle -- for each step of a simulation, so many cumulative points but fewer than the whole Earth every time). To weight the points, we multiplying the probabilities of the heading and the ping -- a joint probability distribution of the heading and ping -- to get the probabilities (suitably weighted) that we randomly select from, for the plane's next position.

The use of the heading distribution around the circle is our proposal distribution, while the ping arc probability is our acceptance distribution. This is analogous and inspired by, but not equivalent, to the proposal and acceptance distributions of the Metropolis-Hastings case.

Because plane only along a ring (a circle arc 1 hr's cruising speed flight time), we choose probability distribution on the circle and then test it with the "acceptance distribution" which is the ping probability.

We've written an optimized simulation that exploits features (hourly pings that determine time step, and maximum distance a 777 can fly in that time period) of the specific problem (finding where MH370 most likely is), since we want an end area of the most probably region MH370 likely is; not just the probability at one specific latitude and longitude.

You can think of it this way: we're currently throwing darts to get the plane's final location at the end. If we do the integral analytically, or numerically via an M-H approach, we're counting the number of times a dart hits one tiny point. An M-H approach would be a better choice if we only cared about one point in the end. If we want the probability adjacent to that one tiny point we'd have to do the computations all over again. But instead, if we throw all of the darts and see where they lie, we can interpolate and get the probability of all of the points at once.

Results

Several versions of the model account for the most likely scenario, known at the time of the model development. Here we only show one example figure for each version for the sake of brevity; more permutations are available in the IPython Notebook.

Version 1

The first version, based on information known until mid-March, uses 5 pings, representing what was thought to be 5 hours of flying time after MH370's disappearance. The plane's position tends to go toward each of these five pings (1 hr, 2 hrs, 3 hrs, 4 hrs, and 5 hrs after disappearance) accounting for varying degrees of ping error measurement, with a pre-filtering condition that assumes the plane's movement is consistent from what we would physically expect from a plane trying to go somewhere: extremely unlikely to execute a sharp turns, or double back on itself at each time step. (Accomplished by drawing from a relative heading distribution which was normal with a mean of 0 and a standard deviation of 30º, where a relative heading of 0º is defined as in the same absolute heading direction – degrees north, south, east, or west, where 0º is due north, 180º is due south, 90º is due east, and 270º is due west – the plane was previously going in.)

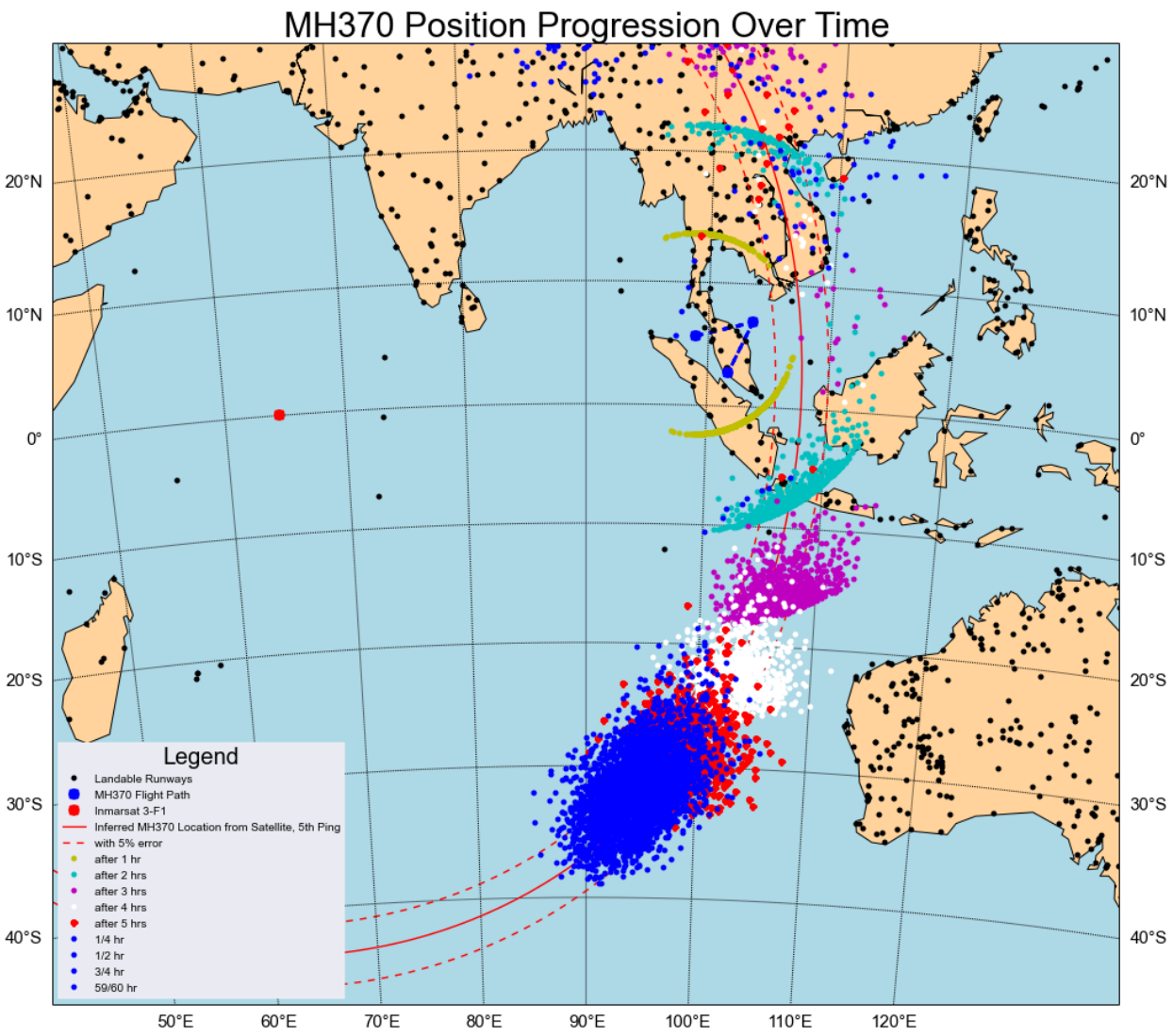

Figure 2 - In the most realistic scenario in Version 1 (based on likely ping error, normal (relative) heading distribution), the plane's final location is off the coast of Australia. Blue dots include up to the last hour of flight time after the simulation ends.

Version 2

The second version, based on information known through late March, uses 6 pings, representing the updated 6+ hours of flying time, at their approximate locations inferred from a variety of media infographics familiar with the undisclosed quantitative details by Malaysian authorities. Since the last ping in both Version 1 and Version 2 remain in the same location, effectively we are modeling $\frac{5}{6}$ cruising speed in Version 1 for each of 5 pings, and Boeing 777 cruising speed in Version 2 for each of 6 pings (the last ping being at the same location in each instance.) The ping error estimates were also reduced in light of new information suggesting them to be substantially lower than what we assumed from mid-March information.

Additionally, the relative heading was modeled with a Wrapped Cauchy and Von Mises distribution, in addition to a normal (Gaussian). The results for the least amount of error (2.5% radius of the last ping arc) are shown below, to best reveal the behavior of how the relative heading distribution affects the results distribution, i.e. where the MH370 ends up.

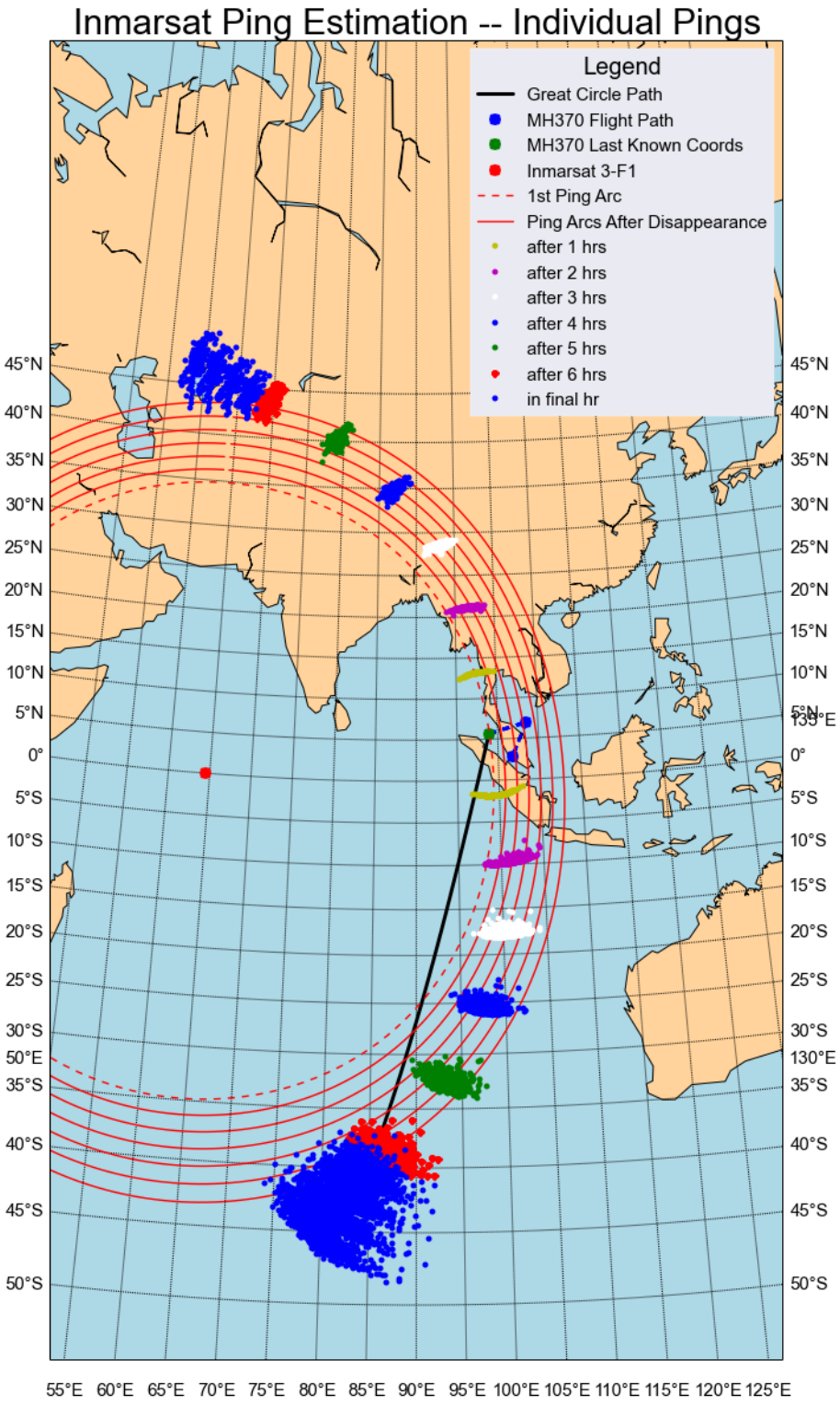

Figure 3 - Von Mises (relative) heading distribution filter, 2.5% ping arc error.

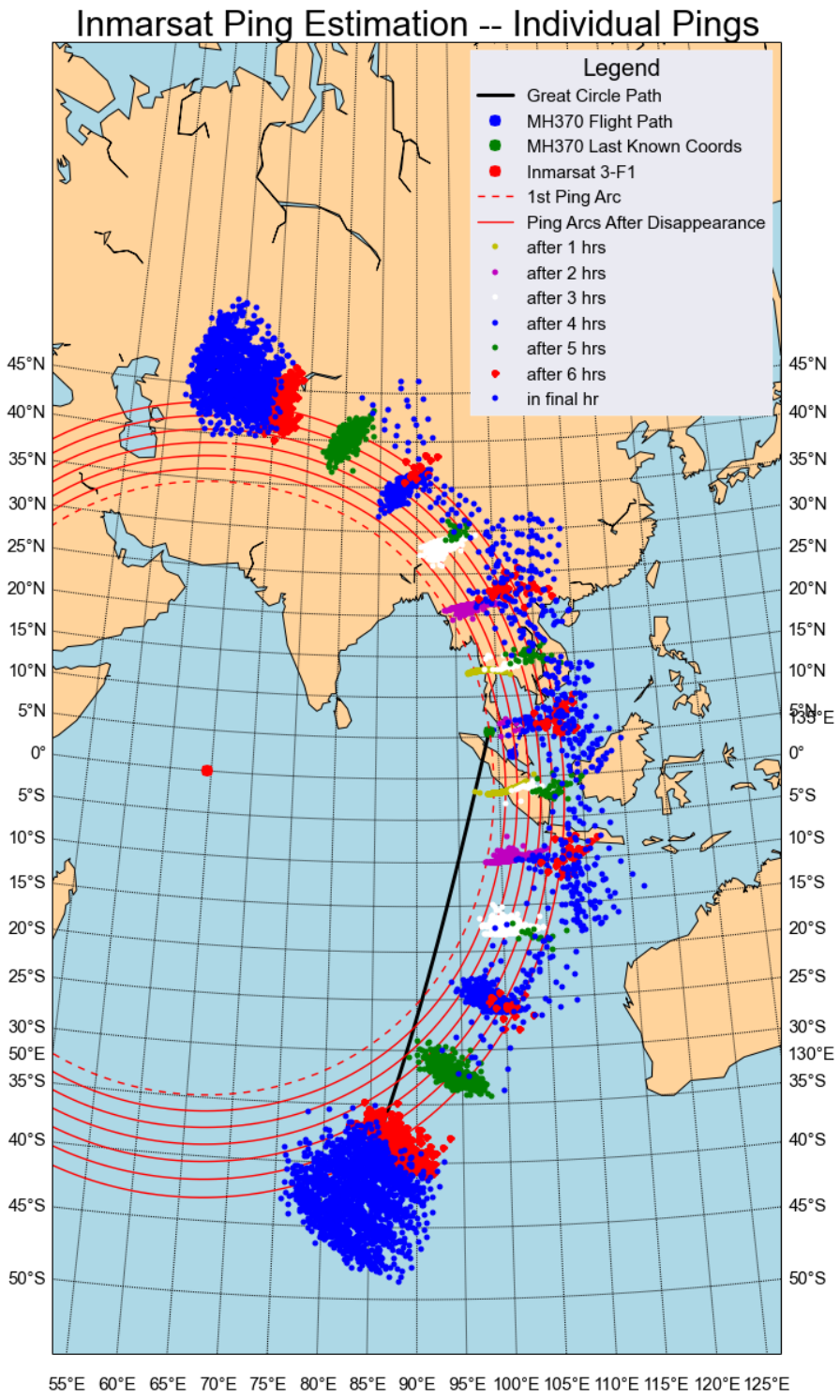

Figure 4 - Wrapped Cauchy (relative) heading distribution filter, 2.5% ping arc error. Note how this distribution causes the plane to go the other direction in some instances, much more so than the Von Mises or Gaussian heading distributions.

Version 3

The final version, based on information known through mid-April, recaps Versions 1 & 2 but with 10,000 simulations – an order of magnitude higher – and also ultimately omits the up-to-last-hour blue dots, which cannot be averaged, and instead plots an hour's radius around the last ping arc means and medians, to infer the area over which MH370 is likely to be located.

The title image *Figure 1*- Part 1, Version 2. Monte Carlo Model results: Most likely locations of MH370 are within the dashed circles. is Version 2. Below are the averages for Version 1:

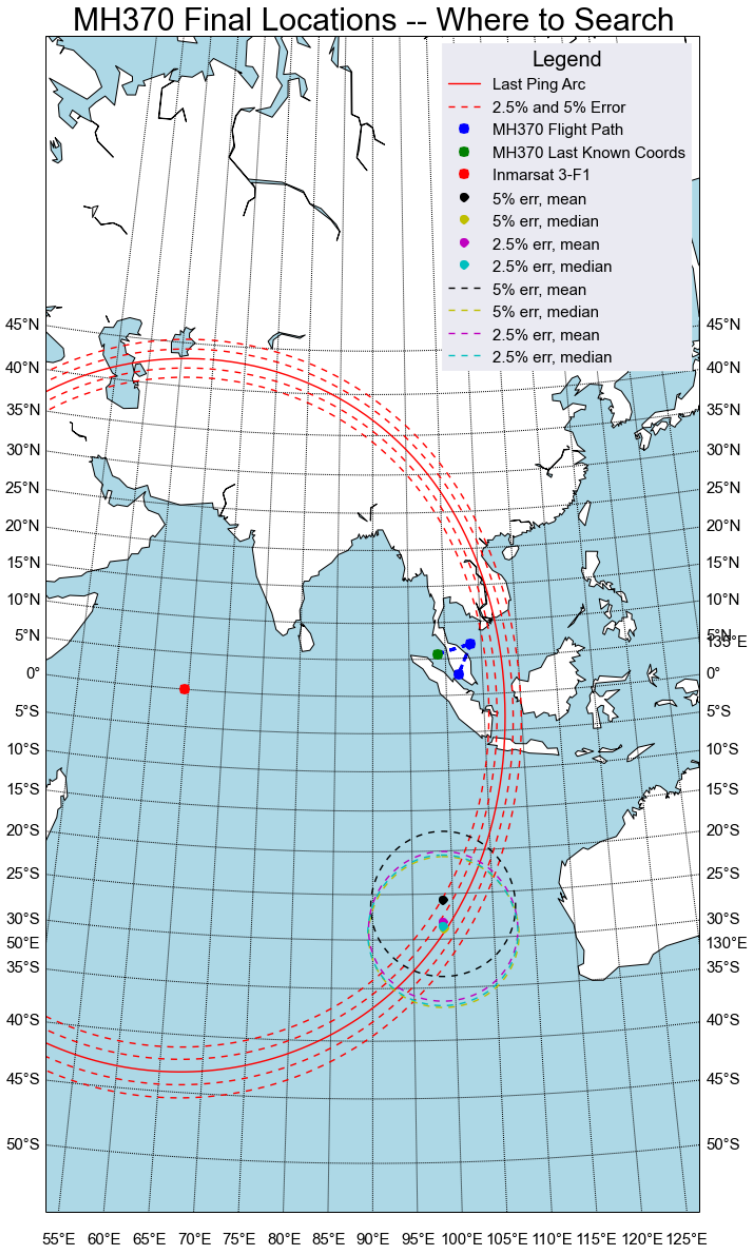

Figure 5 - Part 1, Version 1. Monte Carlo Model results: Most likely locations of MH370 are within the dashed circles.

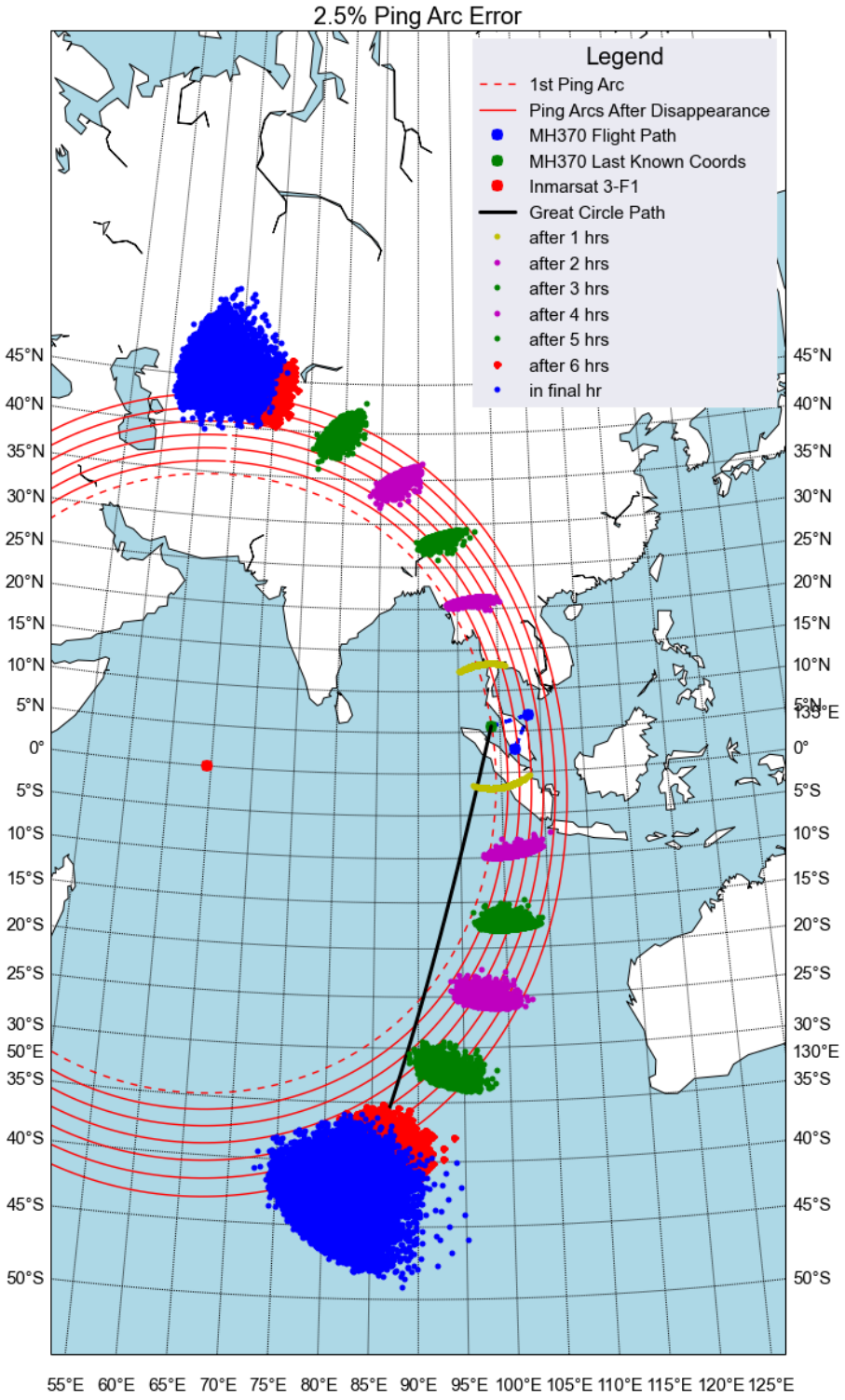

Figure 6 - One of the most realistic scenarios in Version 3, with a normal (relative) heading and 2.5% ping arc error, for the 6 Inmarsat satellite pings. The dashed line circle is the first ping, which occurred before MH370 disappeared from radar, so is not included in our model but is plotted for completeness (it actually serves as a useful conceptual constraint.)

Part 2: Hidden Markov Model

The first model is useful because it lets us explore the space over which MH370 could be in, but it doesn't let us explore some of the methods we learned in AM207 in later parts of the course, such as the Hidden Markov Model (HMM). In the second part of our project, we abstract the plane finding problem and test two general scenarios to where the plane could go if it was intentionally redirected. These scenarios have a set of rules, the latent state of the world that's hidden away from the actual observations of the plane paths we will look at. Basically, this is a methodical exploratory offshoot of exactly what was in the news, for the first few days after disappearance; just after MH370 had disappeared, but when only the plane's last known location before deviating was known.

Scenarios

Each scenario received its own function. The plane will head…

- **Scenario 1** …To a particular random runway whose length is greater than 5000 ft, where a Boeing 777 can land.
- **Scenario 2** …To one of the top 10 most populous Asian cities.
- **Scenario 3** …Minimizing the sum of the combined distance of the top 10 most populous Asian cities to the plane at every time step. (Heading toward them.)
- **Scenario 4** …Maximizing the sum of the combined distance of the top 10 most populous Asian cities to the plane at every time step. (Heading away from them.)
- **Scenario 5** …Maximizing the sum of the combined distance of all 777-landable runways in the amount of distance the 777 can fly over, that covers the time period we are examining. (So if we're running the simulation for 2 hours, we make a 2 hour radius of where the plane can fly, and sum up the distance of the runways within that 2 hour radius and maximize that distance, at every time step. That determines where the plane goes next.)
- **Scenario 6** …Minimizing the sum of the combined distance of all 777-landable runways in the amount of distance the 777 can fly over, that covers the time period we are examining, at every time step.
- **Scenario 7** Scenario 1 permutation. Picks a random runway – with a probability (adjustable) of switching the runway destination at every time step.
- **Scenario 8** Scenario 2 permutation. Picks a quasi-random city (weighted in accordance with fractional population of that city compared to the sum of all populations for top 10 cities), with a switching-city probability.
- **Scenario 9** Wrapper function. Picks either Scenario 1 or Scenario 2 to run (probability adjustable).
- **Scenario 10** Wrapper function. Runs either Scenario 7 or Scenario 8 (probability adjustable).

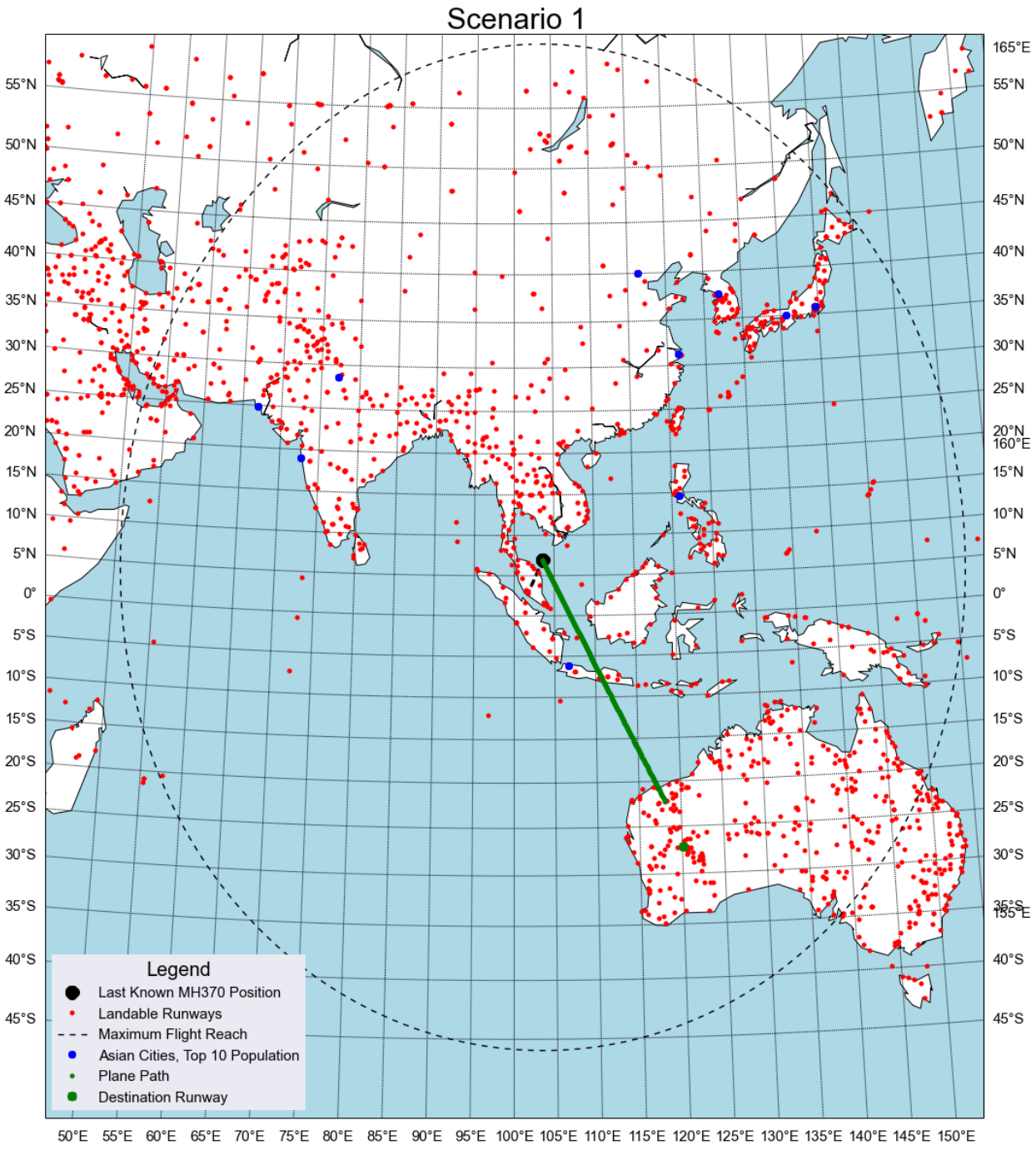

Figure 7 - Heading to a random runway in Australia. We use 4 hrs for these function demonstrations, although we do realize that the plane might overshoot its selection. In our HMM example, we use 2 hrs. Using 2 hours for the functions would not allow for some interesting post-2 hr behavior, particularly in optimizing distance scenarios.

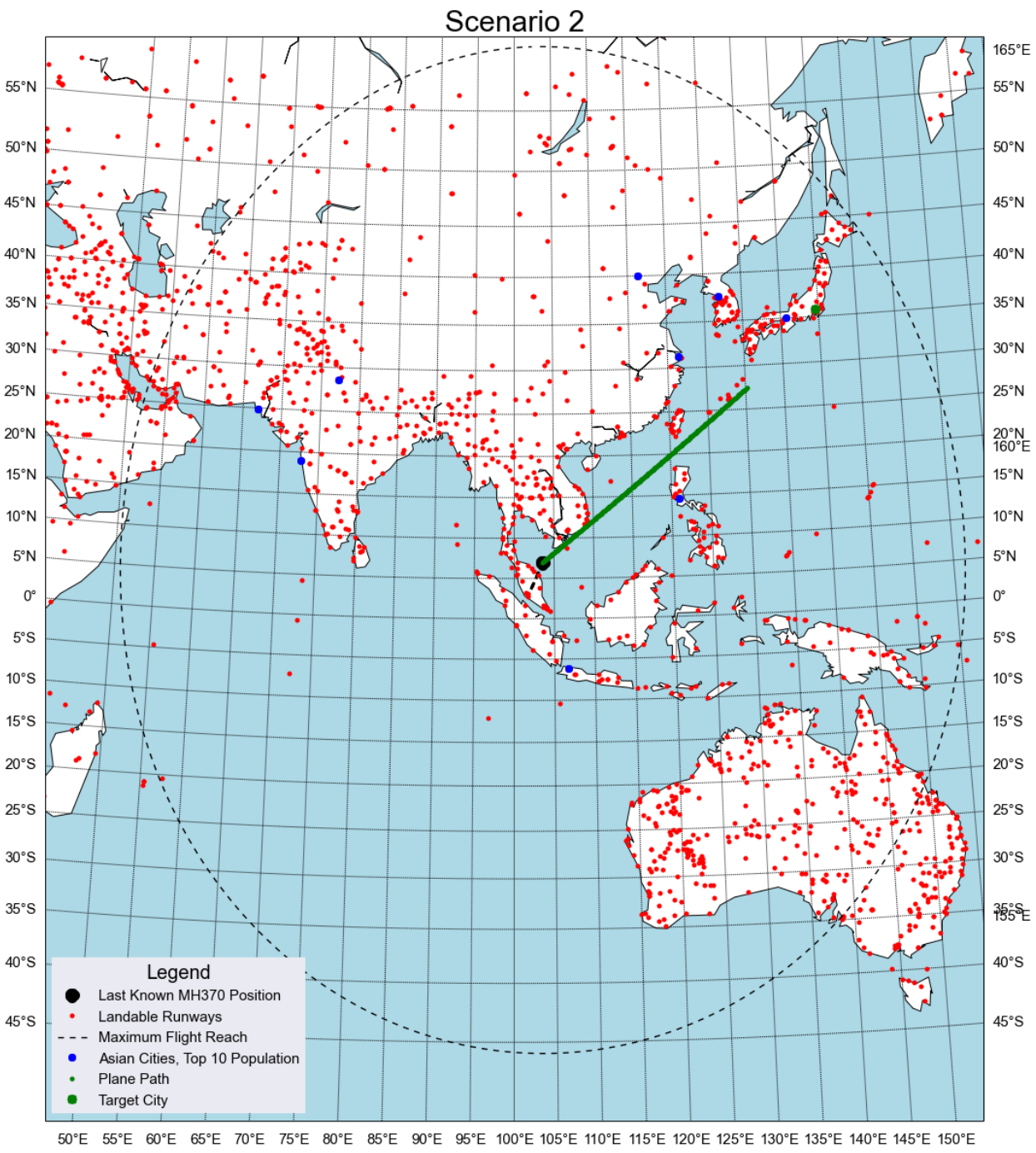

Figure 8 - In this case, the city selected is Tokyo.

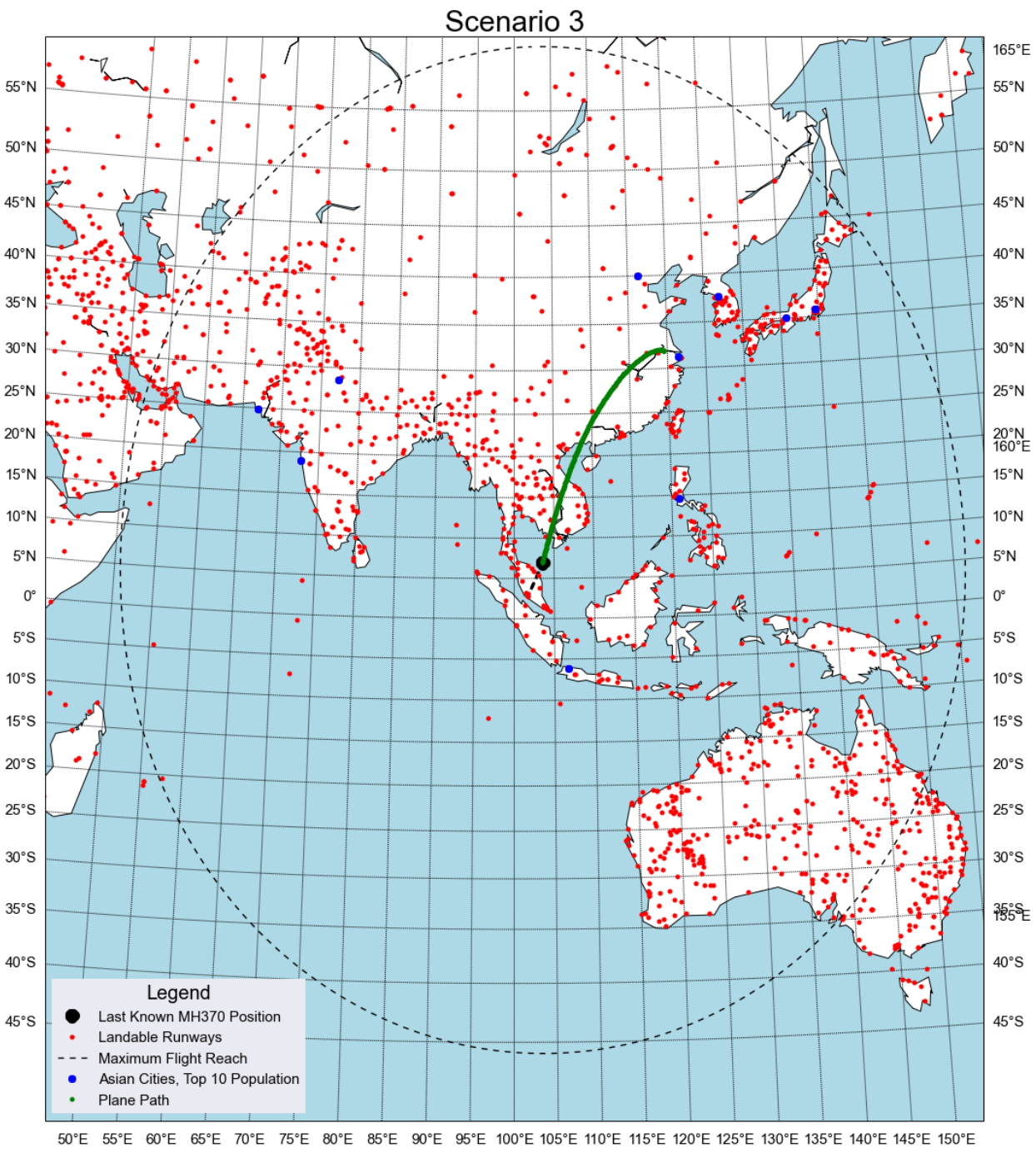

Figure 9 - Notice a hook shape as the way to minimize the distance is to go to the 5 cities between China, Korea, and Japan. This is a fixed property of this scenario; i.e. it happens every time given the same starting location.

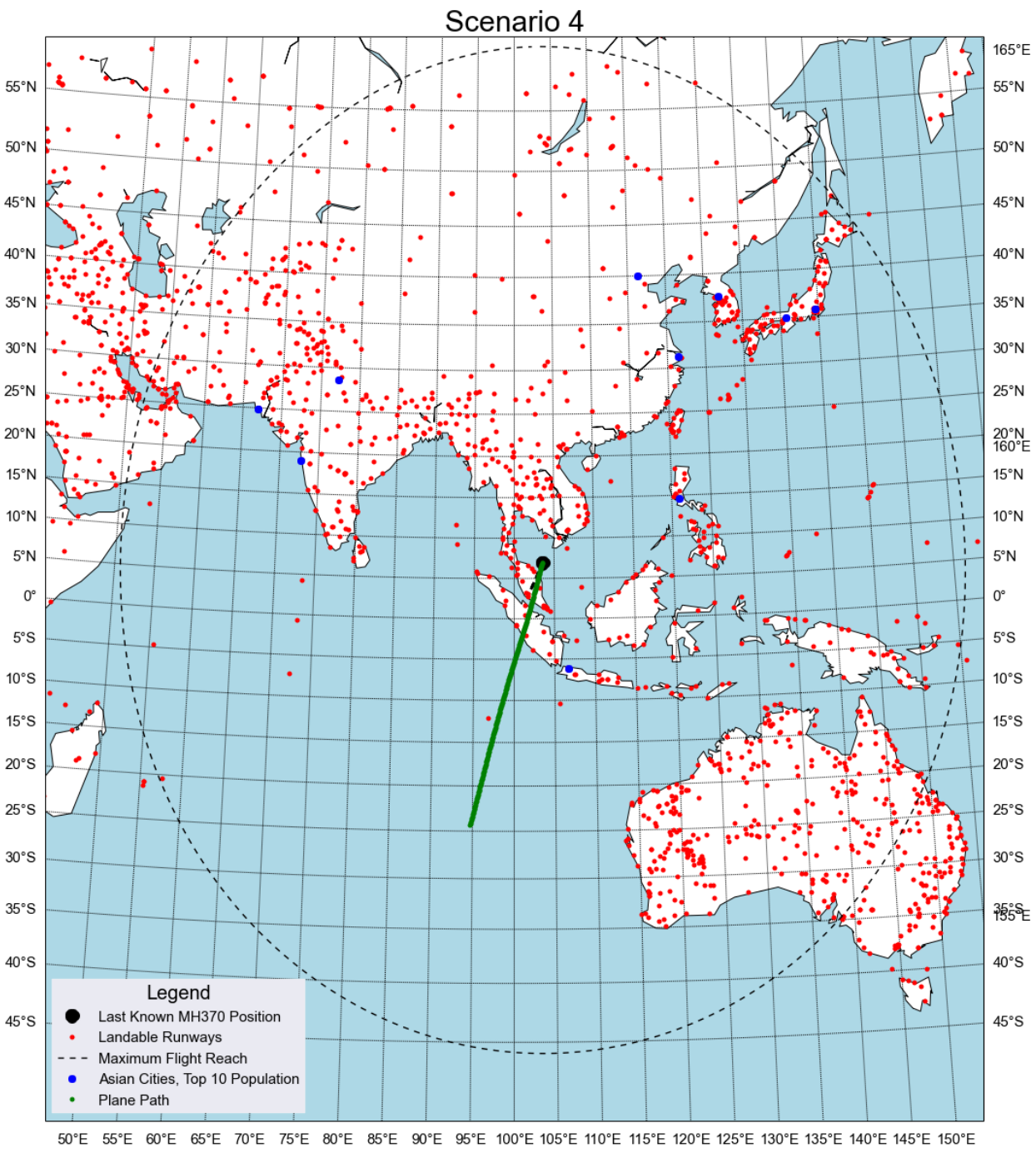

Figure 10 - Unsurprisingly, the plane heads out over the ocean.

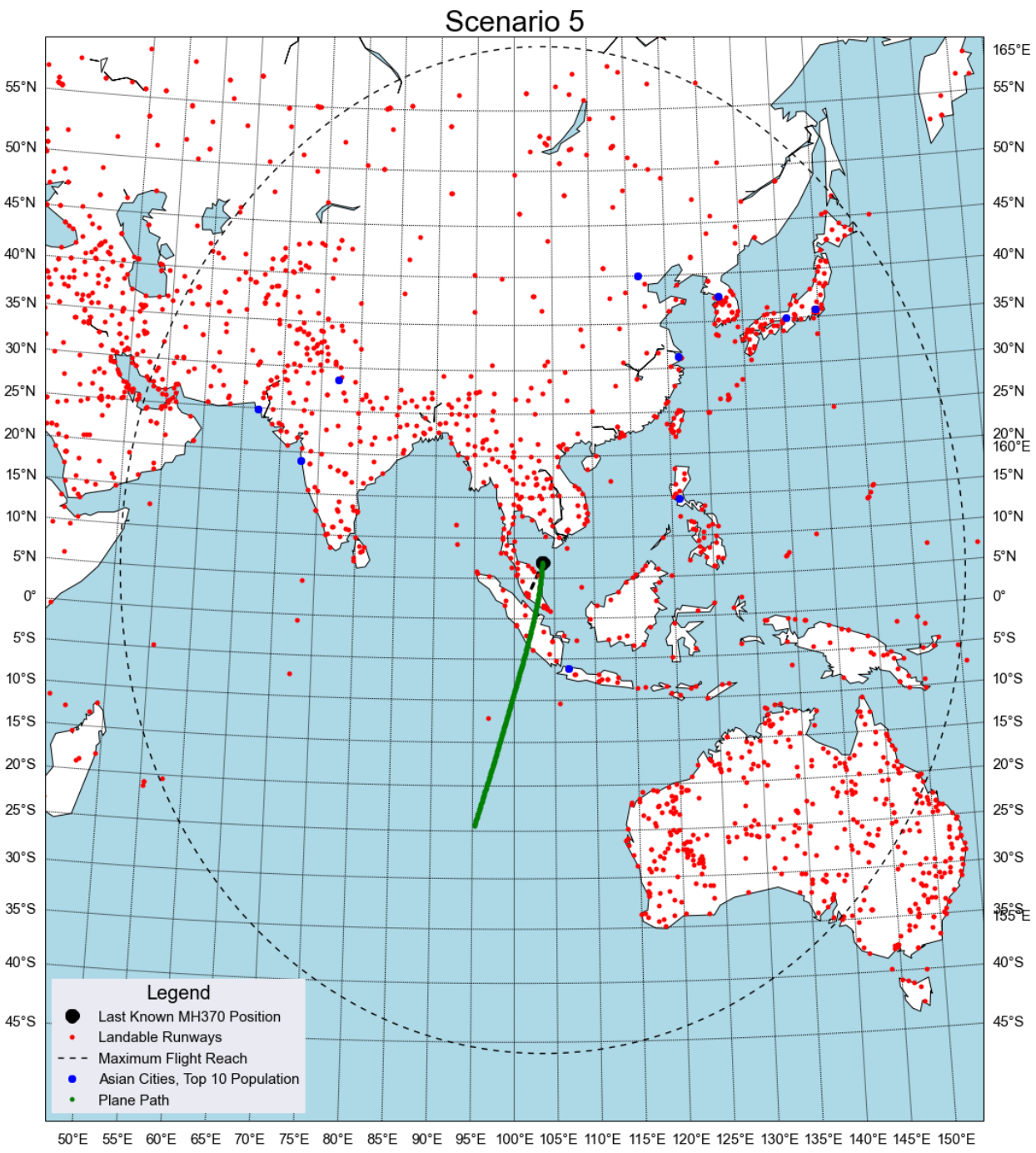

Figure 11 - A slight curve to the path when heading away from runways as opposed to cities.

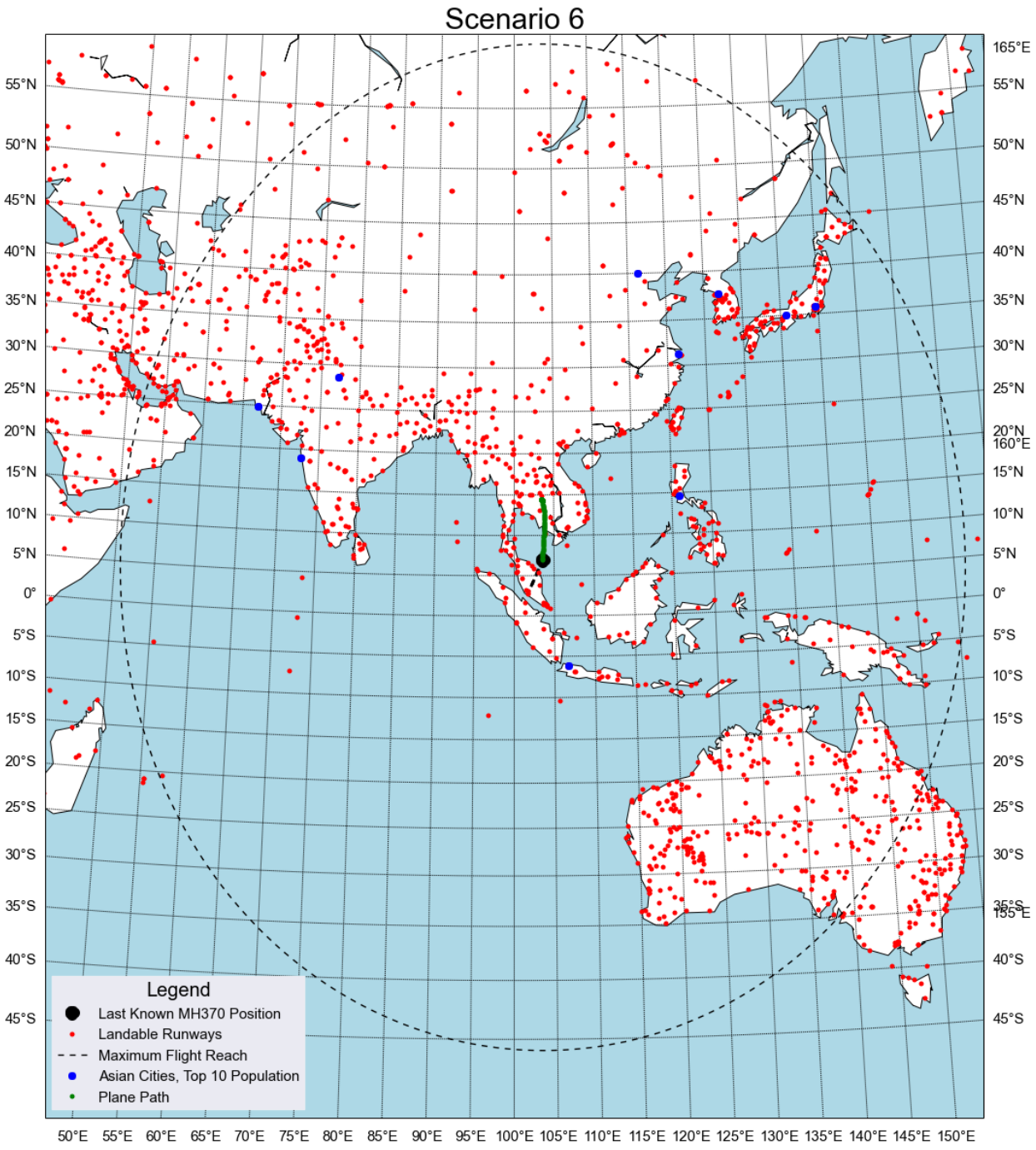

Figure 12 - Note that the plane reaches a local maximum, and then just goes back and forth the rest of the time between two locations.

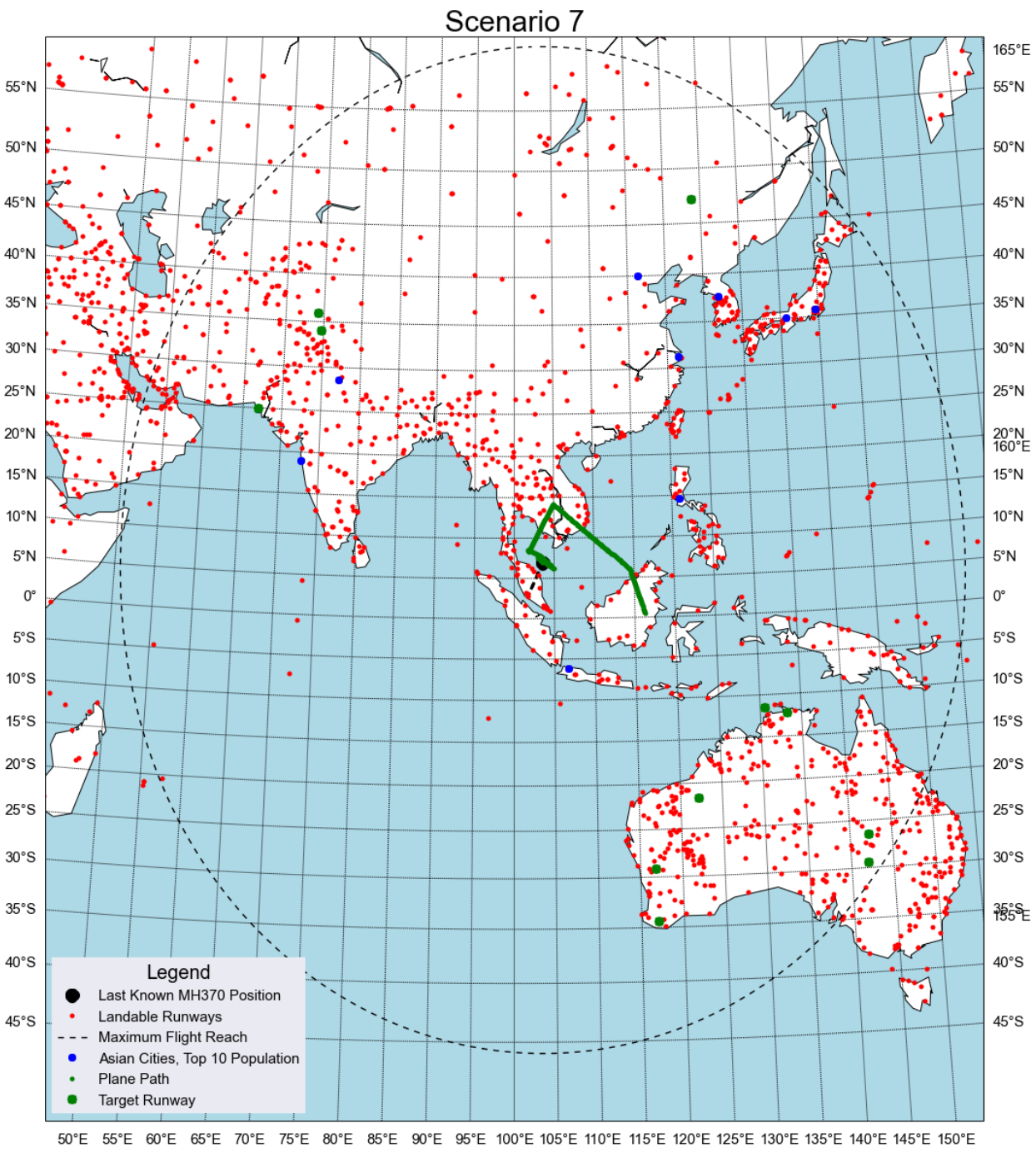

Figure 13 - With a 5% switching probability, the destination changes several times over the course of 4 hrs.

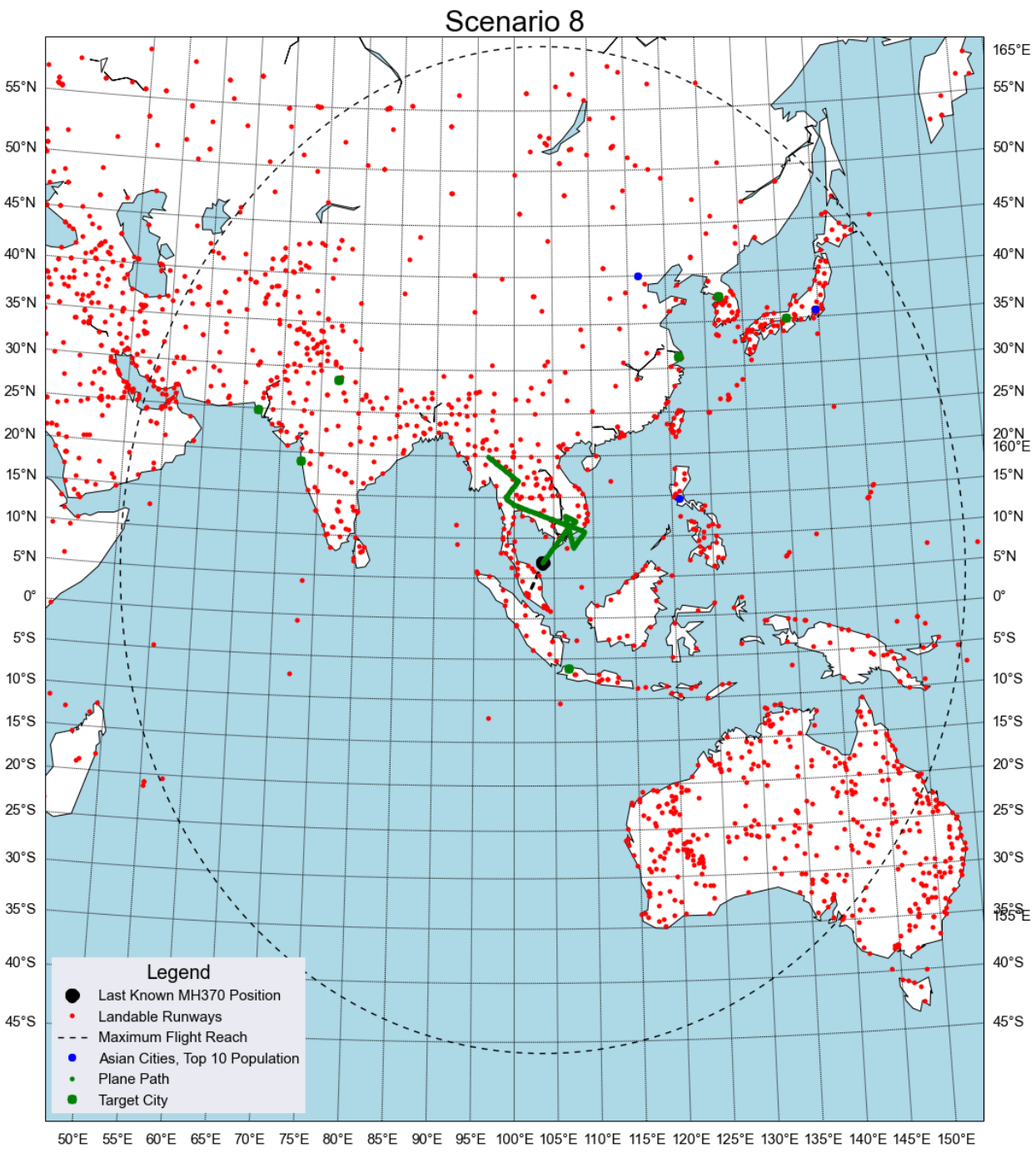

Figure 14 - The same switching effect for cities occurs as it did with runways.

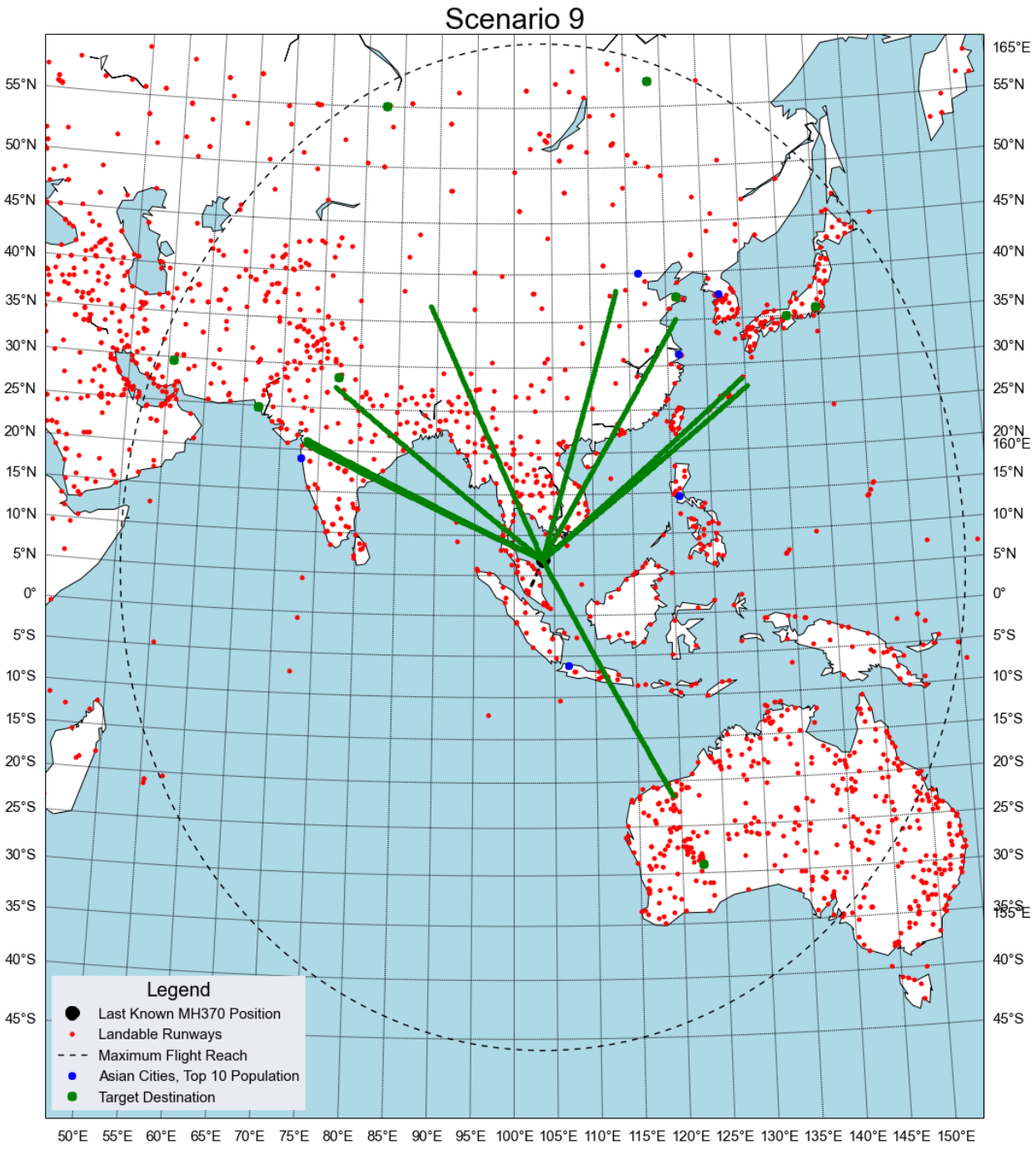

Figure 15 - Running 10 simulations, picking a city or a runway half of the time.

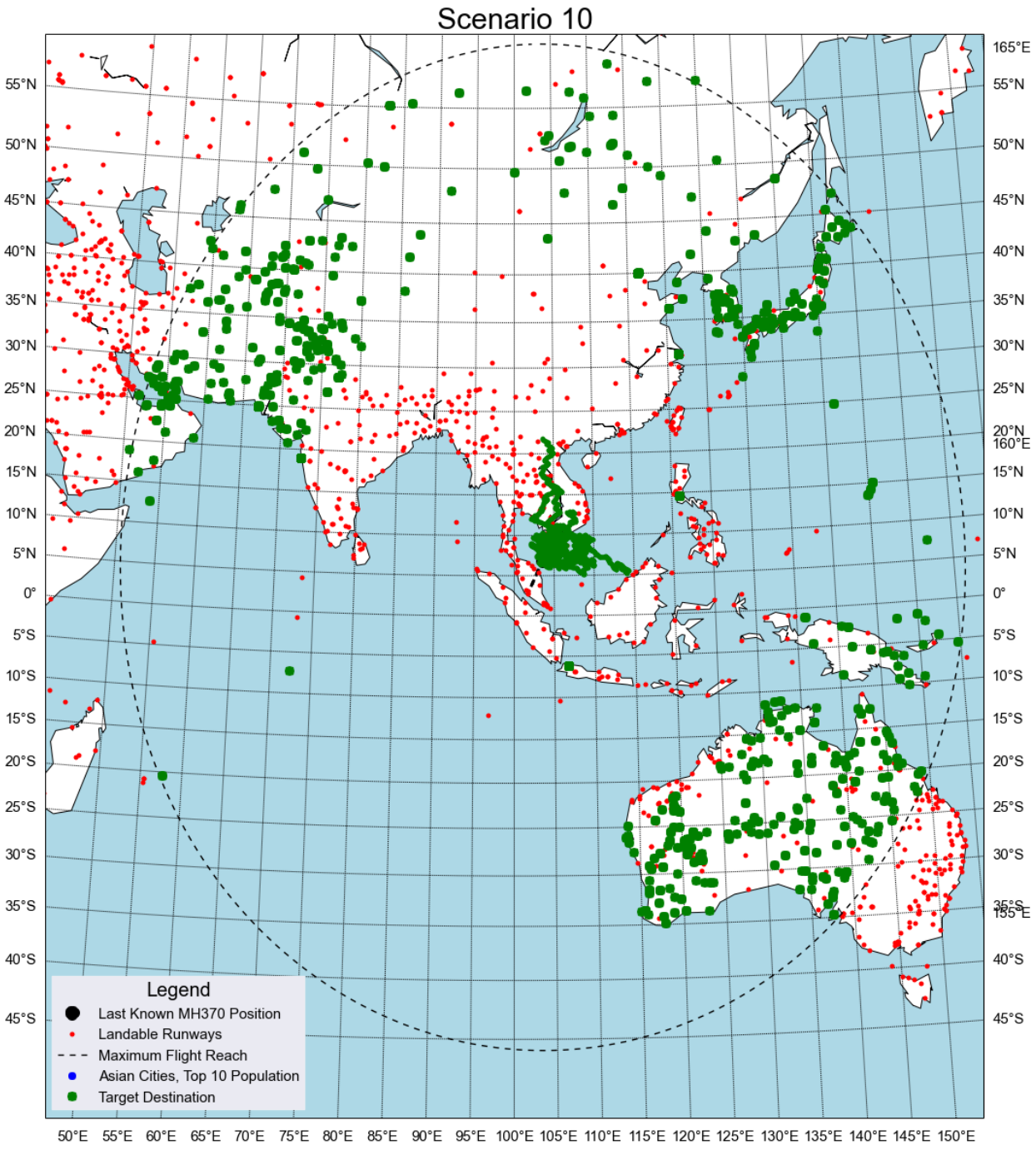

Figure 16 - Running 10 simulations, picking a city or a runway half the time...with a 5% switching probability for each simulation, at each time step.

More generally, we can think of these scenarios as either heading to a place in particular, or trying to optimize by distance, in conjunction with multiple places. So we have two scenario types: particular place, or distance optimization. Associated with each simulation of a scenario, is a random perturbation of the resulting plane path that we see, so that we will not see the underlying states that govern the movement of the plane, actually reflected in our noisy data.

Methods

We run the plane position over time, and sample its observed position (which is based on scenarios, with some perturbation, to reflect that fact that underlying states of the world are not completely reflected in, but heavily influence, the observed data). We use a simple HMM to test which of our proposed latent states are more likely at each stage of the simulation.

Since our goal is to show a proof of concept of how an HMM works, we ran 200 simulations that were either Scenario 1 (90% of time) or Scenario 2 (10% of time). Our goal is to detect the rarer scenario from the more common one. Thus we added more variable error to the rare city scenarios, under the hypothesis that they under such a scenario, passengers (or original pilots) would be perhaps be unwilling to along with the plan, and that such a plan would require on-the-fly maneuvering, so to speak, resulting in a deviated flight path toward the eventual city (compared to, say, a remote runway destination, where the plan would likely be landed safely.) The fact that one type of scenario would have more variance than the other is thus one of our priors, and is a key component for HMM recognition between the two scenario types.

Results

Our preconditions and priors resulted in 20 city scenarios and 180 runway scenarios. (The fact that the percentages exactly worked out was probable to a certain extent, but was not guaranteed.) We sampled over 2 hours, at a sample per minute. For the city scenarios, our HMM properly gauged that it *was* a city scenario, 1958 times and was incorrect 422 times, for a ~82% success rate. For all scenarios, the HMM is right 23203 times and wrong 21842 times, for a ~52% success rate.

If the HMM picked everything to be runways, it'd be right 90% of the time but miss all of the scenarios it wanted it to find; useless. If the HMM picked everything to be cities to ensure we had them all, it would right only 10% of the time, and equally useless. Instead, our simple HMM implementation, using the scikit-learn Python library, picks up *most* cities, but is overzealous in identifying city scenarios compared to a runway. Thus, our HMM is great at identifying the rare city scenarios when they do occur, but it also raises many false alarms. Since we *really* want to detect our city scenarios, our 50+% rate is thus better than the null naïve guess of 10%. We pick up a lot of false instances of runways as cities (when they're in fact runways), to ensure we get and recognize most of the city scenarios when they do in fact occur.

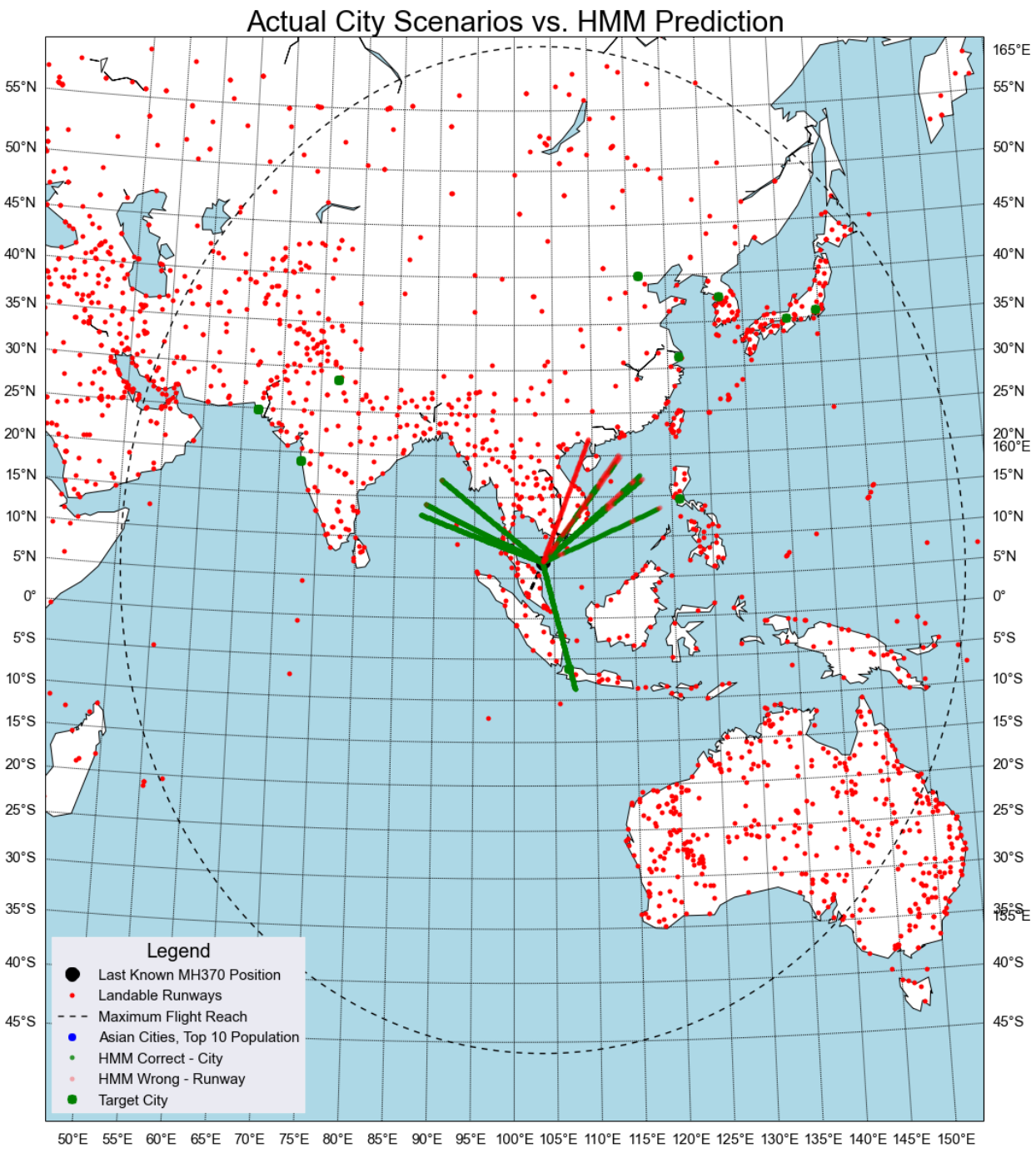

Figure 17 - When the city scenario did occur, the HMM predicted it 80+% of the time at each sampling. Note that the opacity was adjusted in the figure, since red overwhelms green when they're on top of each other.

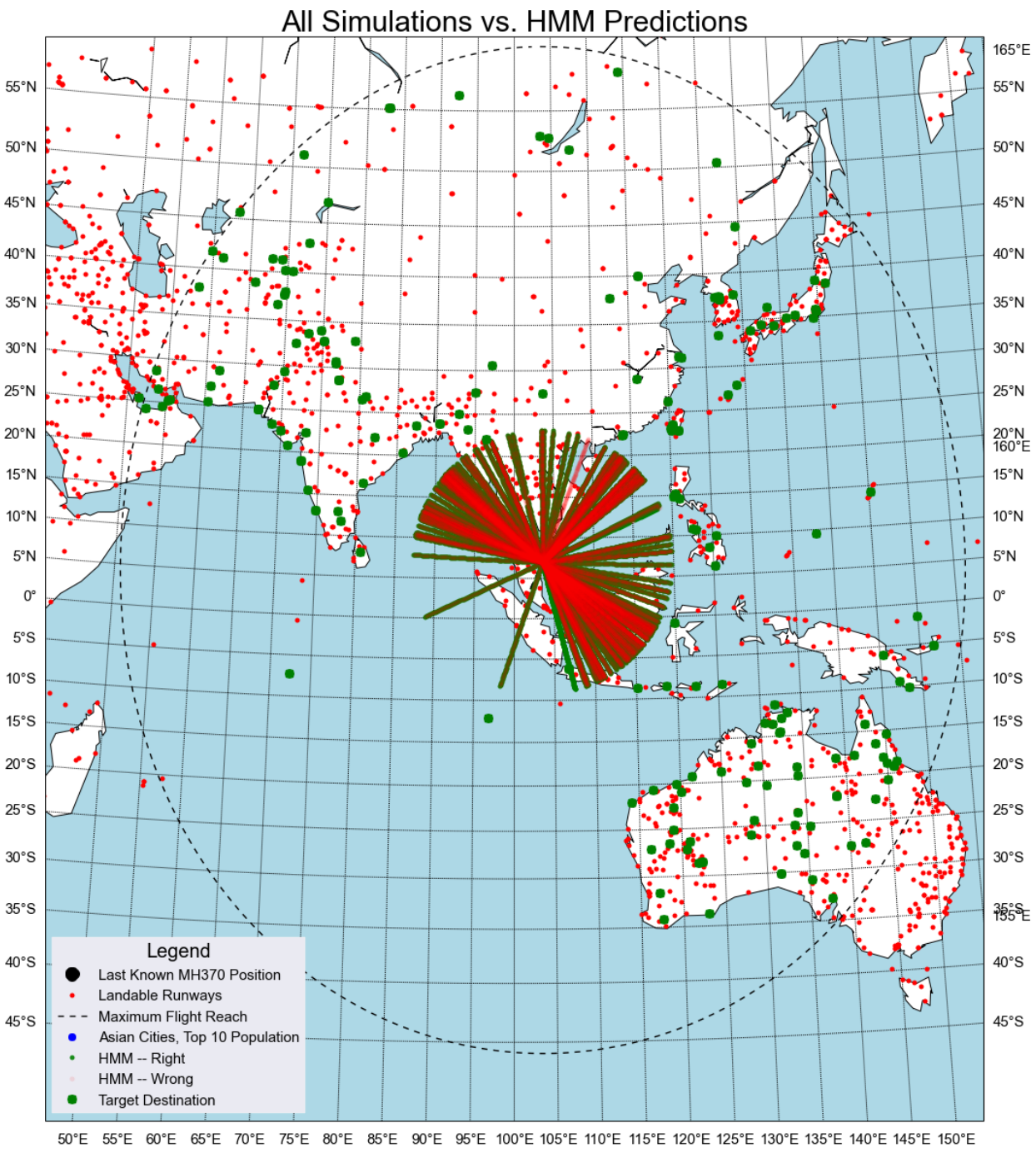

Figure 18 - Overall, predicting the city scenario most of the time came with a cost: a lot of false alarms. Here we see that the scikit-learn HMM makes the correct predictions a little over half the time. (Opacity adjusted to show this visually.)

Part 3: Kalman Filter Model

In our scenario setup examples, our generated straight flight paths represent the plane's path to the target destination. Those original fixed heading paths are idealistic and error-free in terms of how we see them. In reality, like GPS measurements, we can't omnisciently observe the plane moving about without error. In Part 2, our HMM recognized the difference between city and runway after we perturbed these paths, creating varying variance in the scenario samples that each path was composed of.

Now, this noisiness is a complicating factor we ought to be able to deal with and remove. Can we filter out the noise from our lat/lon data added in Part 2, to reveal the true plane path as we track it traveling? Fortunately, there is a fix: we can indeed do so by a Kalman Filter, initially developed from a 1960 paper. Its intended application was actually in the aviation industry, improving the reliability of radar measurements of plane movement. So we are tackling a classic problem with a modern twist from our modeling efforts in Python.

The basic idea of a Kalman Filter is that you create a model for data you are trying to de-noise, and update the model depending on what you actually see over time. Essentially, a Kalman Filter puts a model and measurement together to make an estimate that surpasses using either on their own.

There are many examples of Kalman Filtering and other equivalent methods in today's electronics. For a fresh example, AI autofocus on professional Canon cameras likely uses Kalman Filtering. The autofocus attempts to focus on something moving toward you or away from you (as you see it in the viewfinder.); and incorporates that information in part of the algorithm of a best guess of how that something will move in the future. In the AF case, the sensor measuring is the autofocus distance and you can create a model of the object that has different velocities, or accelerations, or various other things, which then generates a prediction for the next time step and incorporates feedback about that prediction into the next, and so on.

If you have multiple sensors, you can apply what's called sensor fusion. Suppose you're tracking the position of a car. GPS has error, but you also have speedometer which gives an independent measure of velocity, which you can compare with the GPS to constrain and moderate that error.

Methods

We follow examples of a constant voltmeter or tank water level (essentially these are about estimating a constant value that appears to be changing due to measurement error), to reduce the noise that was added to the heading of the plane at each step in Part 2. The change in heading should normally remain constant – specifically, zero – in all cases where target destination is fixed, yet this is no longer be true for our error-added simulations. Then, using the Kalman Filter and initial state guesses about how the simulations *should* look, we reduce the noise in the heading change. Note that this is separate from our HMM model but uses the same error data as a starting point.

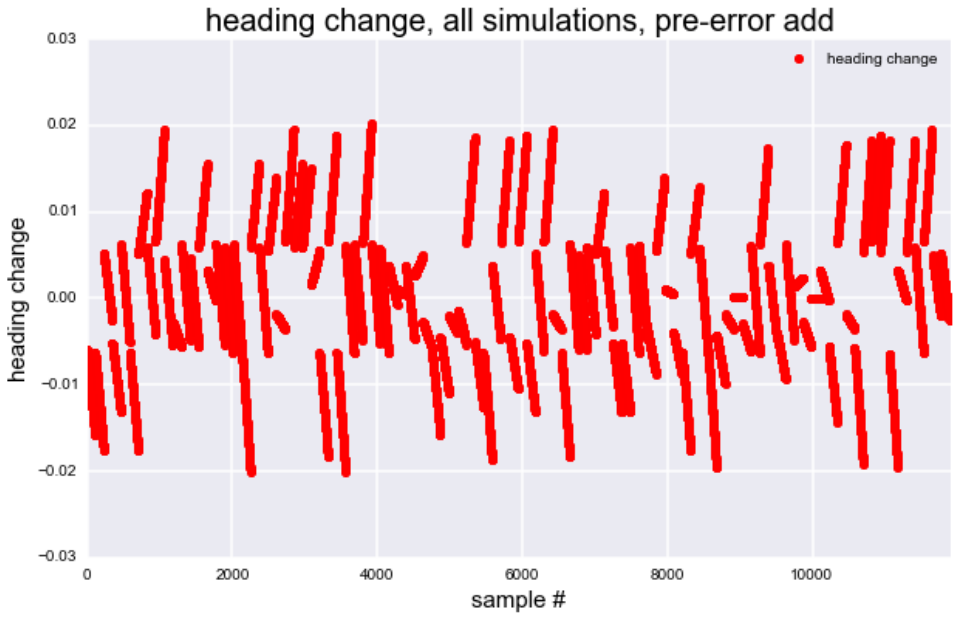

Figure 19 - Heading change error is minimal for straight-path flight simulations. Each simulation consists of 2 hrs' worth of samples taken once per minute (for a total of 120 samples / simulation). The order of occurrence is irrelevant here.

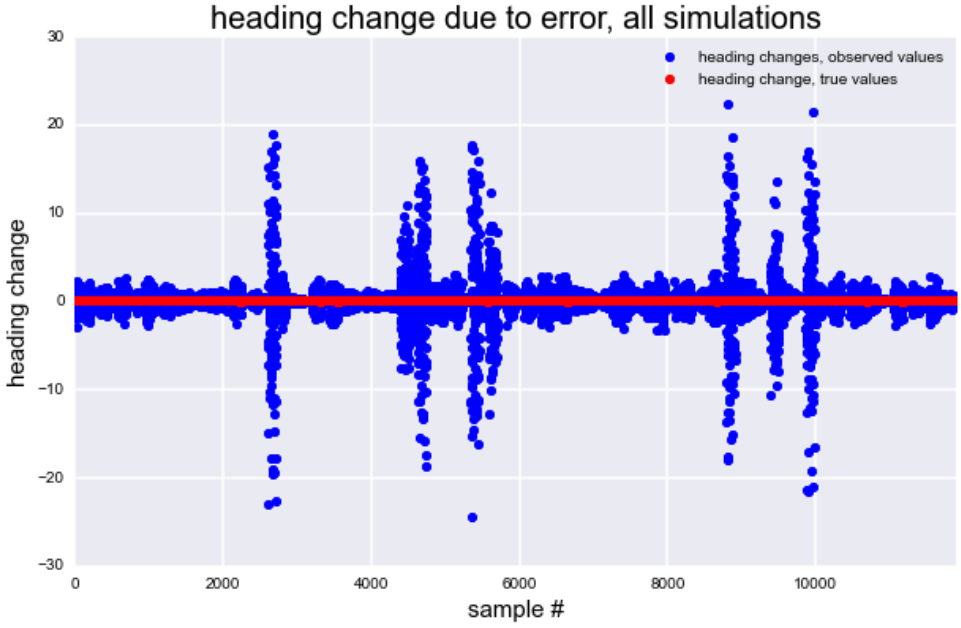

Figure 20 - Yet when we perturb our flight simulations, we discover moderate to large heading changes throughout all of the simulation time steps (samples).

Results

Through an iterative process adjusting the input parameters, our Kalman Filter was able to largely reduce the noise, showing that despite the variance in observations, the true value for the heading change was likely 0, implying the straight flight paths which we created. (Note that other than the input conditions, no one "told" the Kalman Filter that this was the case.) Weaker input parameters lead to less noise reduction, i.e. higher red spikes that are reduced from the blue noise observations, yet with substantial height above zero.

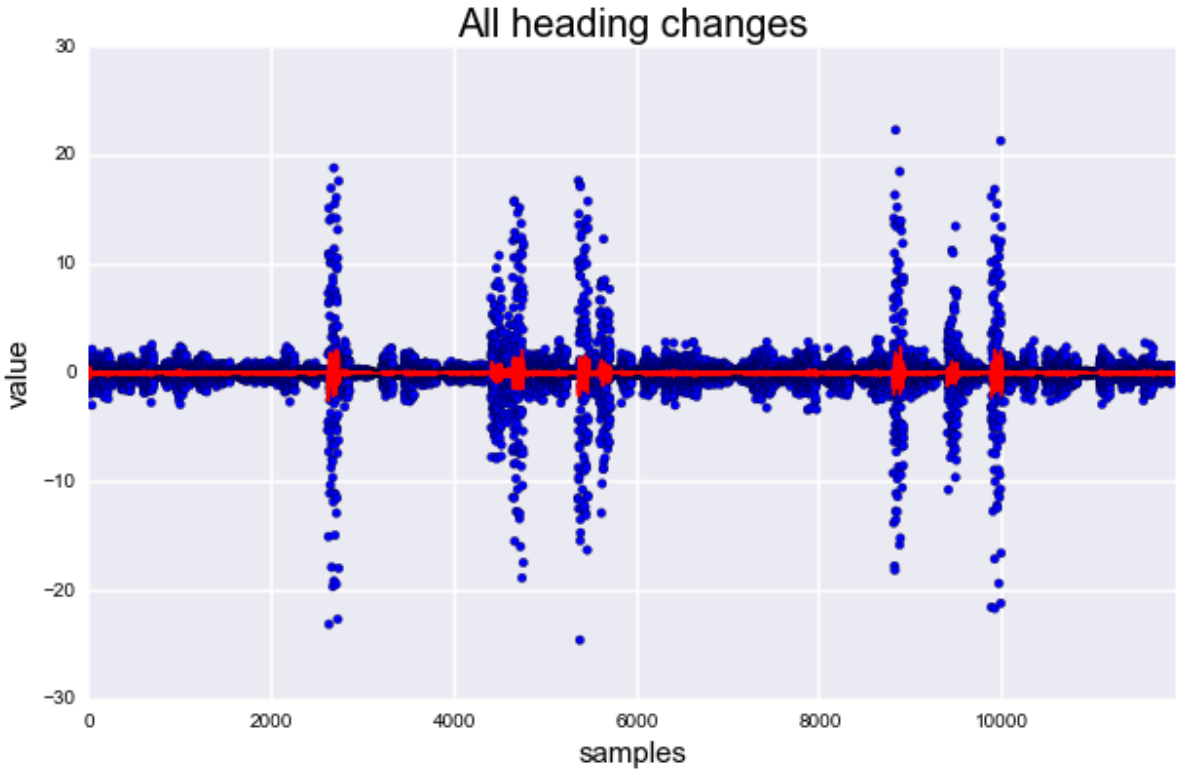

Figure 21 - Our Kalman Filter was successful in reducing noise introduced for our HMM implementation.

What are the remaining spikes? Separating the city scenario samples from the runway samples, we see that city samples are the higher remaining variance, sticking out from the flat lines that stick out from more common runway ones.

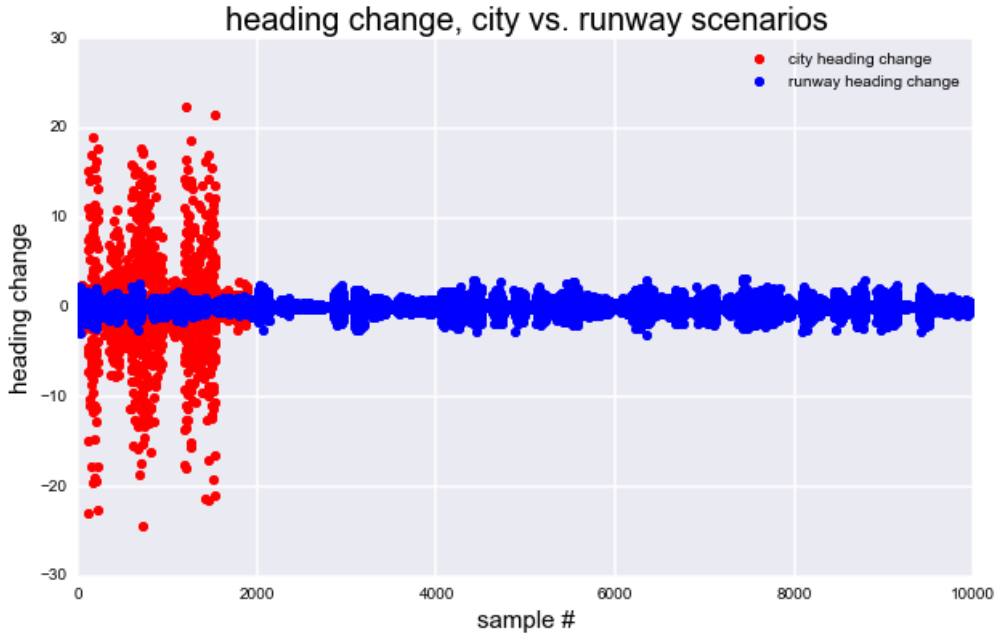

Figure 22 - City scenarios are red; runway scenarios are blue.

Conclusions

Part 1

In Part 1, we were able to establish that MH370 was far more likely to go to the south than to the north, and ended up off the coast of Australia (the latitude range, and to a lesser extent the longitude range, depending on the speed parameter of MH370.) Further information about the Versions 1,2, and 3 are extensively documented in IPython Notebooks and in articles about the general conclusions as published online in *Fast Company*[1], a major media company based in New York City.

Part 2

In Part 2, we show that a HMM implementation can distinguish between rare target destinations (a city) from common ones (a runway), in a way that is much improved from the naïve, null strategies (calling everything a runway or a city). Further improvements are beyond the scope of this project.

Part 3

In Part 3, we implemented a Kalman Filter to eliminate the noise generated for our HMM in Part 2. This implementation is in general useful for eliminating noise to see the underlying path a plane is moving as we observe it in real-time, and could be also be applied more directly to other measurements like latitude or longitude.

[1] Specifically, the Co.Labs section. "**Co.Labs** is a fast-moving look at the creative technologists who are pushing industries into the future--and the methods, philosophies and concepts underlying their code. This site doubles as FastCompany's in-house publishing lab, where we conduct programmatic experiments to try to move the needle on traffic and engagement, and explore novel ways of storytelling. On all its beats, Co.Labs is knowledge in action--the ultimate resource for makers and creators building software to revolutionize the way we do business."

Further Information

More specific sources not relevant to the AM207 material implementations are linked in the IPython Notebooks.

Lectures

AM207 Lecture 3: Basic Monte Carlo. February 3, 2014.

AM207 Lecture 5: Bayesian Formalism 1. February 11, 2014.

AM207 Lecture 7: MCMC. February 18, 2014.

AM207 Lecture 18: Hidden Markov Models. April 10, 2014.

HMM

scikit-learn Hidden Markov Model documentation

Gaussian HMM of Stock Data

Kalman Filter

Introduction to Kalman Filtering: An Engineer's Perspective

Predict-Update Equations (pg 2) in Kalman Filter Applications, Subject MI63: Kalman Filter Tank Filling

An Introduction to the Kalman Filter

Greg Czerniak's Website - Kalman Filters for Undergrads 1

On Reduced-Order Kalman Filters For GPS Position Filtering (pg 2)

Filtering Sensor Data with a Kalman Filter

The Scalar Kalman Filter

Kalman Filter for Dummies