Autonomous Data Error Detection and Recovery in Streaming Applications

RICHARD KLOCKOWSKI, SHIGERU IMAI, COLIN RICE, CARLOS VARELA

CVARELA@CS.RPI.EDU
WORLDWIDE COMPUTING LABORATORY
DEPARTMENT OF COMPUTER SCIENCE
RENSSELAER POLYTECHNIC INSTITUTE
WCL.CS.RPI.EDU

DDDAS 2013 WORKSHOP
INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE
BARCELONA, SPAIN. JUNE 7, 2013
Motivation: Air France Flight 447

- June 1\textsuperscript{st} 2009, Flight 447 from Rio de Janeiro to Paris
- Thunderstorm caused air speed sensors (pitot tubes) to ice and fail
- Autopilot system could not comprehend data failure and disengaged
- Pilots unable to react to erroneous data in a timely manner, eventually stalling the plane into the Atlantic Ocean
- 228 people dead

(http://upload.wikimedia.org/wikipedia/commons/4/4a/Air_France_Flight_447_path.png)
Outline

- **Approach**
  - What to do when (sensor) data goes wrong?

- **ProgrammIng Language for spatiO-Temporal data Streaming applications (PILOTS)**
  - Declarative language support for streaming applications including error detection and correction
  - Multi-modal dynamic data-driven error detection and correction
    - System architecture

- **Error signatures and mode likelihood vectors**
  - Mathematical concepts

- PILOTS program for (air, ground, wind) speed stream(s) checking
  - Using real data to validate approach

- Future work
Approach: Smarter Flight Systems

- The AF447 accident was avoidable
- Air speed could have been recomputed from ground speed and wind speed
- Our goal: to investigate concepts and reasoning and programming technology to facilitate developing smarter (flight) data streaming systems
- Take advantage of functional relationships between independent inputs

\[
\text{ground speed} = \text{airspeed} + \text{wind speed}
\]
Approach: Self-Healing Data Streams

- Monitor input streams; analyze for errors
  - Identify the *cause* of failure
- General concept is to act as an assistant to the human pilot
  - Also situations without human interaction: UAV, Mars rover, ...

**Input data streams**
- (Sensor data)
- • air speed
- • wind speed
- • ground speed
- • performance data
- • fuel levels etc...

**Programming Model**

**Application**

**(Corrected) output data streams**

Error data (measurements) and mode (cause)

**PILOTS:** ProgrammIng Language for spatiO-Temporal data Streaming applications
PILOTS Programming Language

- Designed for spatio-temporal data streaming DDDAS applications
  - Data are related to points in space and time
  - *Ex*: temperatures, moving objects, traffic information, gas prices

- Data selection module
  - How to interpret heterogeneous input as a homogeneous data stream
  - Interpolate existing data to answer application’s queries

- Application module
  - How to generate output and error streams

- Error detection and correction module
  - Error signatures
  - Checks functional relation between input streams
  - Identify potential errors, provide solutions

- PILOTS compiler produces Java code
  - First class support for space and time
  - Generated code uses TCP/IP sockets for input/output data
  - Uses the Java Compiler Compiler (JavaCC)
PILOTS: System Architecture

- **Data Selection**: From heterogeneous to homogeneous data
  - Data corresponding to particular points in space, time
  - Selection operations to approximate data as a contiguous space
    - Closest, Euclidean, Interpolate
- **Error Detection and Correction**
  - Measuring error, matching signatures, and correcting erroneous data
program Twice;

inputs
  a(t) using closest(t);
  b(t) using closest(t);

outputs
  o: b - 2*a at every 1 sec;

errors
  e: b - 2*a;

end;
Running Example:  *Twice*
Running Example: *Twice*

- **A Failure**
- **Timing Jitter**
- **B Failure**
Error Functions and Signatures

- **Error Functions**
  - Make redundancy between independently sensed input streams explicit
    - \( e = b - 2a \)

- **Error Signatures are Constrained Function Patterns**
  - Consist of a function of (space-)time along with a set of constraints
  - Describe relationship between input streams during failure
  - Correspond to particular *known* errors
  - Unknown errors can be recorded to learn/generate new signatures

- **Normal Error Signature (Default)**
  - Behavior of error function under normal circumstances
  - Typically when error function evaluates to zero
Error Signatures: Constrained Function Patterns

- Example:
  \[ S(K): e = t + K, \ K \neq 0; \]
  - \( K \) is unknown constant
  - Interpreted as all lines with slope 1 except the line intersecting \((0,0)\).
program Twice;
inputs
  a(t) using closest(t);
  b(t) using closest(t);
outputs
  o: b - 2*a at every 1 sec;
errors
  e: b - 2*a;
signatures
  s0: e = 0           "Normal";
  s1(K): e = 2*t + K  "A failure";
  s2(K): e = -2*t + K "B failure";
  s3(K): e = K, abs(K) > 20 "Out-of-sync";
correct
  s1: a = b / 2;
  s2: b = a * 2;
end;

S3 is unrecoverable
program Twice;

inputs ...
outputs ...
errors ...
signatures ...
correct ...
end;

public class Twice extends PilotsRuntime {
    private Timer timer_; private SlidingWindow win_o_; private Vector<ErrorSignature> errorSigs_; private ErrorAnalyzer errorAnalyzer_; public Twice( String args[] ) { timer_ = new Timer(); errorAnalyzer_ = new ErrorAnalyzer( errorSigs_, getTau() ); }
    public void getCorrectedData( SlidingWindow win, Value a, Value a_corrected, Mode mode, int frequency ) {
        a.setValue( getData( "a", new Method( Method.Closest, "t" ) ) );
        double e = b.getValue()-2*a.getValue();
        win.push( e );
        mode.setMode( errorAnalyzer_.analyze( win, frequency ) );
        a_corrected.setValue( a.getValue() );
        switch (mode.getMode()) {
            case 1:
                a_corrected.setValue( b.getValue()/2 );
                break;
        }
    }
    public static void main( String[] args ) {
        Twice app = new Twice( args );
        app.startServer();
        app.startOutput_o();
    }
}

public void startOutput_o() {
    try {
        openSocket( OutputType.Output, o, "o" );
    } catch ( Exception ex ) {
        ex.printStackTrace();
    }
    final int frequency = 1000;
    timer_.scheduleAtFixedRate( new TimerTask() {
        public void run() {
            Value a = new Value(); Value a_corrected = new Value(); Value b = new Value(); Value b_corrected = new Value(); Mode mode = new Mode();
            getCorrectedData( win_o_, a, a_corrected, b, b_corrected, mode, frequency );
            double o = b_corrected.getValue()-2*a_corrected.getValue();
            String desc = errorAnalyzer_.getDesc( mode.getMode() );
            dbgPrint( desc +", o=" + o +" at " + getTime() );
            try { sendData( OutputType.Output, o, o ); }
            catch ( Exception ex ) {
                ex.printStackTrace();
            }
        }
    }, 0, frequency );
}
Detecting Error Modes

- **Measured error**
  - Data generated by the application’s *error function* over a window of time (ω most recent samples)

- **Mode likelihood vector**
  - Metric of similarity between measured error and each known error signature
Mode Likelihood Vectors

- Calculate the distance between measured error $e$ and a signature $S_i$

$$\delta_i(t) = \int_{t-\omega}^{t} |e(t) - s_i(t)| dt$$

- Calculate the mode likelihood vector

$$L(t) = < l_0(t), l_1(t), \ldots, l_n(t) >$$ where each $l_i(t)$ is defined as:

$$l_i(t) = \begin{cases} 1, & \text{if } \delta_i(t) = 0 \\ \frac{\min\{\delta_0(t), \ldots, \delta_n(t)\}}{\delta_i(t)}, & \text{otherwise.} \end{cases}$$

If 2nd greatest element of $L$ is greater than threshold $\tau$, error is unknown, else greatest element of $L$ determines current error mode.
Mode Estimation: *Twice*
PILOTS: Performance Metrics

- **Accuracy**
  - How accurately does program determine true error mode?
  - Defined as average number of correct estimated mode $m'$ determinations versus true mode $m$ during a time range $T$.
  - Accuracy = 1 if all mode determinations are correct, and 0 if all are incorrect.

- **Response Time**
  - How quickly does program correctly react to mode changes?
  - Defined as average time it takes to estimate correct mode over all true mode changes in a time range $T$. 
Twice: Experimental Results

- Error detection and correction parameters
  - $\tau =$ threshold to determine mode from likelihood vector
  - $\omega =$ measured error sample size

![Graphs showing corrected error and average response time vs. window size](image)
Calculate ground speed from air speed and wind speed

\[ v_g = \sqrt{(v_w + v_a \cos(\alpha_a - \alpha_w))^2 + (v_a \sin(\alpha_a - \alpha_w))^2} \]

\[ = \sqrt{v_w^2 + 2v_a v_w \cos(\alpha_a - \alpha_w) + v_a^2 \cos^2(\alpha_a - \alpha_w) + v_a^2 \sin^2(\alpha_a - \alpha_w)} \]

\[ = \sqrt{v_w^2 + 2v_a v_w \cos(\alpha_a - \alpha_w) + v_a^2} \]

- \( v_w \): wind speed
- \( \alpha_w \): wind angle
- \( v_a \): air speed
- \( \alpha_a \): air angle
- \( v_g \): ground speed
program SpeedCheck;

inputs
wind_speed, wind_angle (x,y,z)
  using euclidean(x,y), interpolate(z,2);
air_speed, air_angle(x,y,t)
  using euclidean(x,y), closest(t);
ground_speed, ground_angle(x,y,t)
  using euclidean(x,y), closest(t);

outputs
  o: ground_speed = sqrt(air_speed*air_speed + wind_speed*wind_speed + 2*air_speed*wind_speed*cos((PI/180)*(wind_angle-air_angle)))
    at every 1 min;

errors
  e: ground_speed = sqrt(air_speed*air_speed + wind_speed*wind_speed + 2*air_speed*wind_speed*cos((PI/180)*(wind_angle-air_angle)));
end;
**SpeedCheck—Real Flight Data**

- Recorded on an actual flight on April 03, 2012
  - **air speed, air angle:** manually collected by pilot
  - **ground speed, ground angle:** automatically collected from online (radar) data
  - **wind speed, wind angle:** from weather forecast

![Map showing flight path from Albany, NY to Tipton, MD](image)

- Departed at 14:04 on April 3rd, 2012
- Arrived at 15:45 on April 3rd, 2012
**SpeedCheck – Simulated Scenarios**

- **Scenario A**: No errors occurred, actual flight data
- **Scenario B**: Simulate air speed sensors (pitot tube) failure at flight minute 40
  - air speed drops to 50 knots in two minutes
- **Scenario C**: Simulate a GPS failure at minute 40
  - ground speed drops to 0 knots suddenly
- **Scenario D**: Simulate both pitot tube and GPS failures at minute 40 (Scenarios B and C together)
**SpeedCheck: Error signature analysis**

- **No errors**
- **Pitot tube fails at 40 min**
- **GPS fails at 40 min**
- **Both fail at 40 min**

Error signatures show characteristic patterns
program SpeedCheck;
    inputs
    wind_speed, wind_angle (x,y,z)        using euclidean(x,y), interpolate(z,2);
    air_speed, air_angle(x,y,t)        using euclidean(x,y), closest(t);
    ground_speed, ground_angle(x,y,t) using euclidean(x,y), closest(t);
    outputs
    o: ground_speed=sqrt(air_speed*air_speed + wind_speed*wind_speed
                        + 2*air_speed*wind_speed*cos((PI/180)*(wind_angle-air_angle)))
      at every 1 min;
    errors
    e: ground_speed=sqrt(air_speed*air_speed + wind_speed*wind_speed
                        + 2*air_speed*wind_speed*cos((PI/180)*(wind_angle-air_angle)));
    signatures
    s0: e = 0                        "Normal”;
    s1(K): e = K, K > 25            "Pitot tube failure”;
    s2(K): e = K, K > -175, K < -125 "GPS failure”;
    s3(K): e = K, K > -75, K < -25  "Pitot tube + GPS failure”;
    correct
    s1: air_speed = sqrt(ground_speed*ground_speed + wind_speed*wind_speed
                         - 2*ground_speed*wind_speed*cos((PI/180)*(ground_angle-wind_angle)));
    s2: ground_speed = sqrt(air_speed*air_speed + wind_speed*wind_speed
                            + 2*air_speed*wind_speed*cos((PI/180)*(wind_angle-air_angle)));
end;
SpeedCheck: No errors

Mode Estimation:
-1: Unknown
0: Normal
1: Pitot tube failure
2: GPS failure
3: Pitot+GPS failure

Uncorrected Output

Estimated Mode

Both fail (inaccurate)
Normal mode
Unknown mode
SpeedCheck: Pitot tube failure
SpeedCheck: GPS failure

Uncorrected Output

Estimated Mode

Corrected Output

GPS failure mode
SpeedCheck: GPS + Pitot tube failure

Uncorrected Output

GPS + Pitot tube failure mode

Unrecoverable error

Estimated Mode
Concluding Remarks

1. **Error functions**
   - Must make redundancy between independently sensed input streams explicit

2. **Error signature set**
   - Well-behaved: Under normal and known error conditions, must produce nearly orthogonal mode likelihood vectors
   - Ideally comprehensive

3. **Choices of threshold \( \tau \) and window size \( \omega \)**
   - Domain specific
     - *(for Twice and SpeedCheck, \( \tau=0.6 \) and \( \omega=10 \) give best results)*
   - Larger \( \omega \) values lead to less responsive programs; however, for too small \( \omega \), the system enters *unknown* mode more frequently.
   - For well-behaved signature sets, \( \tau \) has less effect on accuracy. Otherwise, for smaller \( \tau \) values, *unknown* mode is entered more frequently, while too large \( \tau \) values produce more false positives.
(Short-term) Future Work

- Applying error signatures
  - Proof of concept with actual Air France data
  - Airplane weight vs. performance analysis (using data from Tunisian Airlines accident)
  - Non-linear error signatures
  - Other domains

- External DDDAS software components
  - Simulated input data
  - Learn new signatures
  - Output visualization
  - Pipelining components and feedback loops
(Long-term) Future Work

A quantitative spatial and temporal logic as a formalism:

- To enable reasoning about data streams that associate values to specific points or intervals of space and time.
- To enable geometric reasoning capabilities, in particular, trigonometric formulae to calculate with aircraft speeds, headings, range, and endurance.

Extensions to logic programming to support stochastic reasoning.

- Language extensions to standard Horn clause-based knowledge bases to incorporate probabilities.
- Special language support for spatial and temporal data streams.
- Incremental reasoning algorithms to dynamically re-compute logical queries efficiently as new data gets injected into the application.

Dynamic Data-Driven Flight Plan Adaptation Examples

<table>
<thead>
<tr>
<th>If...</th>
<th>then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>New pilot report: icing en route</td>
<td>New route</td>
</tr>
<tr>
<td>New winds aloft</td>
<td>New altitude</td>
</tr>
<tr>
<td>New surface winds at destination</td>
<td>New airport</td>
</tr>
<tr>
<td>Imminent engine failure</td>
<td>Nearest airport</td>
</tr>
</tbody>
</table>

Ground speed and crosswind as functions of airspeed, wind, and runway heading:

\[ v_g = v_a + \sin(\alpha_w - \alpha_a) \times v_w \]
\[ v_x = \cos(\alpha_w - \alpha_r) \times v_w \]
Open Source Software

- Download PILOTS 0.2.1 at:
  
  http://wcl.cs.rpi.edu/pilots

  Thanks! Questions?

  Partial support from:
  Air Force Office of Scientific Research
  DDDAS Program
  Dr. Frederica Darema
  (AFOSR Grant No. FA9550-11-1-0332)
  &
  Yamada Corporation Fellowship

New textbook:

PROGRAMMING DISTRIBUTED COMPUTING SYSTEMS
A Foundational Approach

CARLOS A. VARELA

MIT Press, June 2013
Extra Slides
**PILOTS: Language Grammar**

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td><code>::=</code> <code>program Var;</code> (\text{inputs }) <code>Inputs ;</code> (\text{outputs }) <code>Outputs ;</code> (\text{errors }) <code>Errors ;</code> (\text{signatures }) <code>Signatures ;</code> (\text{correct }) <code>Corrects ;</code> (\text{end;})`</td>
</tr>
<tr>
<td>Inputs</td>
<td><code>::=</code> <code>[ (Input ;)* Input ]</code></td>
</tr>
<tr>
<td>Input</td>
<td><code>::=</code> <code>Vars Dim using Methods</code></td>
</tr>
<tr>
<td>Outputs</td>
<td><code>::=</code> <code>[ (Output ;)* Output ]</code></td>
</tr>
<tr>
<td>Output</td>
<td><code>::=</code> <code>Vars : Exps at every Time</code></td>
</tr>
<tr>
<td>Errors</td>
<td><code>::=</code> <code>[ (Error ;)* Error]</code></td>
</tr>
<tr>
<td>Error</td>
<td><code>::=</code> <code>Vars : Exps</code></td>
</tr>
<tr>
<td>Signatures</td>
<td><code>::=</code> <code>[ (Signature ;)* Signature ]</code></td>
</tr>
<tr>
<td>Signature</td>
<td><code>::=</code> <code>Var [ Const ] : Var = Exp , Exps [ String ]</code></td>
</tr>
<tr>
<td>Corrects</td>
<td><code>::=</code> <code>[ (Correct ;)* Correct ]</code></td>
</tr>
<tr>
<td>Correct</td>
<td><code>::=</code> <code>Var [Const] : Var = Exp</code></td>
</tr>
<tr>
<td>Const</td>
<td><code>::=</code> <code>( Var )</code></td>
</tr>
<tr>
<td>Dim</td>
<td><code>::=</code> `(t)</td>
</tr>
<tr>
<td>Method</td>
<td><code>::=</code> `(closest</td>
</tr>
<tr>
<td>Time</td>
<td><code>::=</code> `Number (msec</td>
</tr>
</tbody>
</table>
PILOTS: Language Grammar

**Vars** ::= Var | Var, Vars

**Var** ::= \{ a, b, c, ...\}

**String** ::= \{ “a”, “b”, “c”, ...\}

**Exps** ::= Exp | Exp, Exps

**Exp** ::= Func(Exps) | Exp Func Exp | ‘( Exp ‘)’ | Value

**Methods** ::= Method | Method, Methods

**Func** ::= \{ +, -, *, /, sqrt, sin, cos, tan, abs, ... \}

**Value** ::= Number | Var

**Number** ::= Sign Digits | Sign Digits’.’Digits

**Sign** ::= ‘+’ | ‘-’ | ‘‘

**Digits** ::= Digit | Digits Digit

**Digit** ::= \{ 0, 1, 2, ..., 9\}
PILOTS: Libraries

Data Selection Module

DataReceiver
Socket sock;

Add data
Start server

DataStore
static Vector<DataStore> stores;
String[] varNames;
Vector<SpatioTempoData> data;

SpatioTempoData
double[][] locations;
Date[] times;
Vector<Double> values;

Parse data
Get current location and time

CurrentLocationTimeService
(interface class)

Implement

Simulation Service
SimpleTime Service

PILOTS Runtime Library

PilotsRuntime
Socket[] outputSockets;
Socket[] errorSockets;

Get data
Send data
 ...

Parse arguments
Detect mode

ArgParser

ErrorAnalyzer

Output/Error Hosts

Extend

PILOTS Application

Data Input Client

Send data

Closest (one dimensional)
- x, y, z, or t
- Selects the input data that is closest to the given coordinate

Euclidean (2D or 3D)
- Similar to closest
- Operates on multiple dimensions
- Euclidean distance

Interpolate (any dimensions, number $n$)
- Chooses the $n$ closest data points in the given dimensions
  - Based on weighted sum of Euclidean distance
- Interpolates into one representative data point
PILOTS: Data Selection Operations

**closest(x)**

Current Location: \( x_{\text{curr}} \)

Select

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \]

**interpolate(x,y,2)**

Interpolate

Current Location \( l_{\text{curr}} = (x_{\text{curr}}, y_{\text{curr}}) \)

Select

\[ l_0 = (x_0, y_0) \]

\[ l_1 = (x_1, y_1) \]

\[ l_2 = (x_2, y_2) \]

Data gets selected depending on the current location and time

**euclidean(x,y)**

Current Location \( l_{\text{curr}} = (x_{\text{curr}}, y_{\text{curr}}) \)

Select

\[ l_0 = (x_0, y_0) \]

\[ l_1 = (x_1, y_1) \]

Ignore

\[ l_2 = (x_2, y_2) \]
PILOTS 0.2.1 supports two types of error signatures

- **Constant**
  - Example: \( S: e = K \)

- **Linear**
  - Example: \( S(K): e = 2t + K \)
    - Slope 2 and unknown constant \( K \)

- Each signature has an optional set of constraints
PILOTS: Compiler

- Uses JCC (Java Compiler Compiler)

program example;
inputs ... ;
outputs ... ;
errors ... ;
signatures ... ;
correct ... ;
end;

public class Twice extends PilotsRuntime {
private Timer timer_; 
public Twice() {
timer_ = new Timer();
}
public void startOutput_e() {
try {
openSocket( OutputType.Error, 0, "e" );
}
} catch ( Exception ex ) {
ex.printStackTrace();
}
... }
Related Work

- **Spatio-temporal Logic**
  - Stochastic reasoning
  - PROLOG

- **Data Stream Management Systems**
  - Real-time sensor data analysis
  - Moving objects

- **Flight Systems**
  - Airspace geometry
  - Route/outcome analysis
## Data Format

- **Inputs and outputs share the same format**

<table>
<thead>
<tr>
<th>first line</th>
<th>#var0,var1,...\n</th>
</tr>
</thead>
<tbody>
<tr>
<td>after second line</td>
<td>ex1) x0,y0~x1,y1::val0,val1,...\n</td>
</tr>
<tr>
<td></td>
<td>ex2) x,y,z:t0~t1:val0,val1,...\n</td>
</tr>
<tr>
<td></td>
<td>ex3) x0~x1:t:val0,val1,...\n</td>
</tr>
<tr>
<td></td>
<td>ex4) :t:val0,val1,...\n</td>
</tr>
<tr>
<td>last line</td>
<td>\n</td>
</tr>
</tbody>
</table>

- **Example Data**

  - ex1) 40.100,-76.300~39.600,-76.300::166.0,215.0
  - ex2) 42.749,-73.802,3000:2012-04-03 140000-0500 ~2012-04-03 210000-0500:15.0,320.0
  - ex3) 42.6886~43.9258:2012-04-03 140900-0500 :112.0,222.0
  - ex4) :2012-04-03 141400-0500:42.5486,-74.1142,8100.0
Airspeed/Air Angle Data  
(collected on April 3rd, 2012 flight)

<table>
<thead>
<tr>
<th>#air_speed,air_angle</th>
<th>42.748,-73.802~41.476,-75.483::162,246</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>41.476,-75.483~41.000,-76.000::161,239</td>
</tr>
<tr>
<td></td>
<td>41.000,-76.000~40.100,-76.300::161,221</td>
</tr>
<tr>
<td></td>
<td>40.100,-76.300~39.600,-76.300::166,215</td>
</tr>
<tr>
<td></td>
<td>39.600,-76.300~39.500,-76.400::165,213</td>
</tr>
<tr>
<td></td>
<td>39.500,-76.400~39.085,-76.759::135,204</td>
</tr>
</tbody>
</table>
#ground_speed,ground_angle
42.7672,-73.8061:2012-04-03 140400-0500:79,309
42.7775,-73.8233:2012-04-03 140500-0500:78,221
42.7586,-73.8456:2012-04-03 140600-0500:99,221
42.7353,-73.8733:2012-04-03 140700-0500:104,218
42.7119,-73.8981:2012-04-03 140800-0500:106,221
42.6886,-73.9258:2012-04-03 140900-0500:112,222
42.6656,-73.9536:2012-04-03 141000-0500:113,230
42.64, -73.9953:2012-04-03 141100-0500:154,224
42.6103,-74.0339:2012-04-03 141200-0500:163,224
...
39.3, -76.6 :2012-04-03 153700-0500:153,201
39.2667,-76.6167:2012-04-03 153800-0500:146,201
39.2333,-76.6333:2012-04-03 153900-0500:127,201
39.2, -76.65 :2012-04-03 154000-0500:119,201
39.1667,-76.6667:2012-04-03 154100-0500:128,218
39.15, -76.6833:2012-04-03 154200-0500:116,218
39.1167,-76.7167:2012-04-03 154300-0500:122,237
39.1, -76.75 :2012-04-03 154400-0500:117,201
39.0667,-76.7667:2012-04-03 154500-0500:99,201

First 9 minutes when departing

Last 9 minutes when landing
Wind Speed/Wind Angle Data
(predicted for April 3rd, 2012 flight)

<table>
<thead>
<tr>
<th>wind_speed, wind_angle</th>
<th>42.749, -73.802</th>
<th>3000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 15,320</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>42.749, -73.802</td>
<td>6000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 33,350</td>
</tr>
<tr>
<td></td>
<td>42.749, -73.802</td>
<td>9000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 35,340</td>
</tr>
<tr>
<td></td>
<td>42.749, -73.802</td>
<td>12000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 44,340</td>
</tr>
<tr>
<td></td>
<td>41.338, -75.723</td>
<td>3000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 17,320</td>
</tr>
<tr>
<td></td>
<td>41.338, -75.723</td>
<td>12000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 47,330</td>
</tr>
<tr>
<td></td>
<td>40.639, -73.779</td>
<td>3000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 18,330</td>
</tr>
<tr>
<td></td>
<td>40.639, -73.779</td>
<td>6000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 33,350</td>
</tr>
<tr>
<td></td>
<td>40.883, -78.087</td>
<td>6000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 21,330</td>
</tr>
<tr>
<td></td>
<td>40.883, -78.087</td>
<td>9000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 36,330</td>
</tr>
<tr>
<td></td>
<td>40.883, -78.087</td>
<td>12000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 42,310</td>
</tr>
<tr>
<td></td>
<td>39.510, -76.980</td>
<td>3000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 10,310</td>
</tr>
<tr>
<td></td>
<td>39.510, -76.980</td>
<td>6000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 26,330</td>
</tr>
<tr>
<td></td>
<td>39.510, -76.980</td>
<td>12000:2012-04-03 140000-0500 ~ 2012-04-03 210000-0500: 42,330</td>
</tr>
</tbody>
</table>

Albany, NY
Pittston, PA
JFK, NY
Philipsburg, PA
Westminster, MD