Class Documentation

0.0.1 Class Drone

Represents the drone. Performs route planning, than follows that plan to collect sensor data.

- **public** Drone(Settings settings,io.InputController input,io.OutputController output)

  - **Parameters**
    - `settings` – the current Settings
    - `input` – the InputController which handles data input
    - `output` – the OutputController which handles data output

0.0.2 Methods

- **start**

  public void start()

  - **Description**
    Start the drone and perform route planning and data collection for the given settings.

0.0.3 Class Move

A class representing a single move to be made by the drone.

0.0.4 Declaration

public class Move
extends java.lang.Object

0.0.5 Constructor summary

Move(Coords, Coords, int, W3W)

0.0.6 Method summary

getAfter()
getBefore()
getDirection()
getSensorW3W()
toString()

0.0.7 Constructors

- **Move**

  ```
  public Move(geometry.Coords before,geometry.Coords after,int
      direction,W3W sensorW3W)
  ```

  - **Parameters**
    - `before` – the position of the drone before the move
    - `after` – the position of the drone after the move
    - `direction` – the direction of the move in degrees, from 0 to 350 anticlockwise starting from east
    - `sensorW3W` – the location of the sensor visited by the drone at the end of this move, or null if no sensor is visited

0.0.8 Methods

- **getAfter**

  ```
  public geometry.Coords getAfter()
  ```

  - **Returns** – the position of the drone after making the move

- **getBefore**

  ```
  public geometry.Coords getBefore()
  ```

  - **Returns** – the position of the drone before making the move

- **getDirection**

  ```
  public int getDirection()
  ```

  - **Returns** – the direction of move in degrees

- **getSensorW3W**

  ```
  public W3W getSensorW3W()
  ```

  - **Returns** – the W3W of the sensor that this move reaches, or null if it does not reach a sensor

- **toString**

  ```
  public java.lang.String toString()
  ```

0.0.9 Class Results

Holds and processes the calculated flightpath and collected sensor data

0.0.10 Declaration

```
public class Results
 extends java.lang.Object
```

0.0.11 Constructor summary

Results(List) Constructor

0.0.12 Method summary

getFlightpathString() Gets a flightpath String of the following format: 1,[startLng],[startLat],[angle],[endLng],[endLat],[sensor w3w or null]“n 2,[startLng],[startLat],[angle],[endLng],[endLat],[sensor w3w or null]“n ...
getMapGeoJSON() Creates a GeoJSON string of a map which displays the flightpath of the drone and markers displaying the readings or status of the sensors.
recordFlightpath(List) Adds a calculated flight to the results
recordSensorReading(Sensor) Add a sensor with its readings to the results

0.0.13 Constructors

- **Results**

  ```
  public Results(java.util.List sensorW3Ws)
  ```

  - **Description**
    Constructor
  - **Parameters**
    - `sensorW3Ws` – a list of sensor locations as W3W that the drone is visiting

0.0.14 Methods

- **getFlightpathString**

  ```
  public java.lang.String getFlightpathString()
  ```

  - **Description**
    Gets a flightpath String of the following format: 1,[startLng],[startLat],[angle],[endLng],[endLat],[sensor w3w or null]“n 2,[startLng],[startLat],[angle],[endLng],[endLat],[sensor w3w or null]“n ...
  - **Returns** – the flightpath String

- **getMapGeoJSON**

  ```
  public java.lang.String getMapGeoJSON()
  ```

  - **Description**
    Creates a GeoJSON string of a map which displays the flightpath of the drone and markers displaying the readings or status of the sensors.
  - **Returns** – a String of the GeoJSON

- **recordFlightpath**

  ```
  public void recordFlightpath(java.util.List flightpath)
  ```

  - **Description**
    Adds a calculated flight to the results
  - **Parameters**
    * `flightpath` – a list of Moves representing the flightpath

- **recordSensorReading**

  ```
  public void recordSensorReading(Sensor sensor)
  ```

  - **Description**
    Add a sensor with its readings to the results
  - **Parameters**
    * `sensor` – the Sensor

0.0.15 Class Sensor

A sensor with a battery level and reading.

0.0.16 Declaration

```
public class Sensor
 extends java.lang.Object
```

0.0.17 Constructor summary

Sensor(W3W, float, String) Constructor

0.0.18 Method summary

getBattery()
getLocation()
getReading()

0.0.19 Constructors

- **Sensor**

```
public Sensor(W3W w3wLocation, float battery, java.lang.String
    reading)
```

  - **Description**
    Constructor
  - **Parameters**
    * `w3wLocation` – the W3W location of the sensor
    * `battery` – the current battery level of the sensor, as a percentage
    * `reading` – the reading of the sensor as a String

0.0.20 Methods

- **getBattery**

```
public float getBattery()
```

  - **Returns** – the battery level of this sensor as a percentage

- **getLocation**

```
public W3W getLocation()
```

  - **Returns** – the W3W location of this sensor

- **getReading**

```
public java.lang.String getReading()
```

  - **Returns** – the reading of the sensor, as a String. If the battery level is 10% or greater this should contain a float value, but if it is less than 10% the reading cannot be trusted and may be incorrect, null or NaN.

0.0.21 Class SensorMarkerFactory

A factory which constructs sensor markers which displays the location, status and reading of a sensor. This factory does not produce hypothetical SensorMarker instances, but Feature instances instead, since Feature cannot be subclassed due to its lack of a public constructor.

0.0.22 Declaration

```
public class SensorMarkerFactory
 extends java.lang.Object
```

0.0.23 Constructor summary

SensorMarkerFactory()

0.0.24 Method summary

getSensorMarker(W3W, Sensor) Creates a marker located at the position of this sensor.

0.0.25 Constructors

- **SensorMarkerFactory**

```
public SensorMarkerFactory()
```

0.0.26 Methods

- **getSensorMarker**

```
public Feature getSensorMarker(W3W w3w, Sensor sensor)
```

  - **Description**
    Creates a marker located at the position of this sensor. If a sensor reading was taken successfully the marker is coloured and assigned a symbol based on the reading, and if it has low battery or was not visited, assigns different symbols.
  - **Parameters**
    - `w3w` – the location of the sensor as a W3W
    - `sensor` – the Sensor containing the sensor data, or null if the sensor was not visited
  - **Returns** – a Feature containing a Point and various attributes describing the marker

0.0.27 Class Settings

Holds the settings derived from the command line arguments.

0.0.28 Declaration

```
public class Settings
 extends java.lang.Object
```

0.0.29 Constructor summary

Settings(String[])

0.0.30 Method summary

getDay()
getMaxRunTime()
getMonth()
getPort()
getRandomSeed()
getStartCoords()
getYear()

0.0.31 Constructors

- **Settings**

  ```
  public Settings(java.lang.String[] args)
  ```

  - **Parameters**
    - `args` – the input command line args

0.0.32 Methods

- **getDay**

  ```
  public int getDay()
  ```

  - **Returns** – the day to generate the map for

- **getMaxRunTime**

  ```
  public double getMaxRunTime()
  ```

  - **Returns** – the maximum run time of the flight planner in seconds

- **getMonth**

  ```
  public int getMonth()
  ```

- **Returns** – the month to generate the map for

- **getPort**

```
public int getPort()
```

  - **Returns** – the port number of the server

- **getRandomSeed**

```
public int getRandomSeed()
```

  - **Returns** – the random seed to use in the algorithms

- **getStartCoords**

```
public geometry.Coords getStartCoords()
```

  - **Returns** – the starting coordinates of the drone

- **getYear**

```
public int getYear()
```

  - **Returns** – the year to generate the map for

0.1 Class W3W

Holds what3words coordinate and word information.

0.1.1 Declaration

```
public class W3W
 extends java.lang.Object
```

0.1.2 Constructor summary

W3W(Coords, String)

0.1.3 Method summary

getCoordinates()
getWords()

0.1.4 Constructors

- **W3W**

  ```
  public W3W(geometry.Coords coordinates,java.lang.String words)
  ```

  - **Parameters**
    - `coordinates` – the coordinates of the centre of the W3W square
    - `words` – the 3 words

0.1.5 Methods

- **getCoordinates**

  ```
  public geometry.Coords getCoordinates()
  ```

  - **Returns** – the coordinates of the centre of the W3W square

- **getWords**

  ```
  public java.lang.String getWords()
  ```

  - **Returns** – words the 3 words

Chapter 1

Package uk.ac.ed.inf.aqmaps.deserializers

Package Contents *Page*

Classes

1.1 Class CoordsDeserializer

Used for deserialization of Coords, this is needed since the field names in Coords inherit from Point2D so @SerializedName can't be used to rename lng and lat to x and y

1.1.1 Declaration

```
public class CoordsDeserializer
 extends java.lang.Object
```

1.1.2 Constructor summary

CoordsDeserializer()

1.1.3 Method summary

getCoords()

1.1.4 Constructors

- **CoordsDeserializer**

```
public CoordsDeserializer()
```

1.1.5 Methods

- **getCoords**

```
public uk.ac.ed.inf.aqmaps.geometry.Coords getCoords()
```

  - **Returns** – a Coords object

1.2 Class SensorDeserializer

Used only for deserializing the sensor information from JSON. The real Sensor class stores the location as W3W instead of a String.

1.2.1 Declaration

```
public class SensorDeserializer
 extends java.lang.Object
```

1.2.2 Constructor summary

SensorDeserializer()

1.2.3 Method summary

getBattery()
getLocation()
getReading()

1.2.4 Constructors

- **SensorDeserializer**

```
public SensorDeserializer()
```

1.2.5 Methods

- **getBattery**

  ```
  public float getBattery()
  ```

  – **Returns** – the battery level of this sensor as a percentage

- **getLocation**

  ```
  public java.lang.String getLocation()
  ```

  – **Returns** – the location of the sensor as a W3W string

- **getReading**

  ```
  public java.lang.String getReading()
  ```

  – **Returns** – the reading of the sensor, as a String

1.3 Class W3WDeserializer

Used for deserialization of W3W, `CoordsDeserializer` (in 1.1, page 10) for why this is necessary.

1.3.1 Declaration

```
public class W3WDeserializer
 extends java.lang.Object
```

1.3.2 Constructor summary

W3WDeserializer()

1.3.3 Method summary

getW3W()

1.3.4 Constructors

- **W3WDeserializer**

  ```
  public W3WDeserializer()
  ```

1.3.5 Methods

- **getW3W**

  public uk.ac.ed.inf.aqmaps.W3W getW3W()

  - **Returns** – a W3W object

Chapter 2

Package uk.ac.ed.inf.aqmaps.flightplanning

Package Contents *Page*

Classes

2.1 Class EnhancedTwoOptTSP

A modified version of TwoOptHeuristicTSP from JGraphT which is used to compute tours which visit all sensors and returns to the starting point. Source code of TwoOptHeuristicTSP can be found here (GitHub).

JGraphT main website, GitHub source, accessed 30/11/2020

The following JavaDoc is unchanged from the the original:

The 2-opt heuristic algorithm for the TSP problem.

The travelling salesman problem (TSP) asks the following question: "Given a list of cities

and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?".

This is an implementation of the 2-opt improvement heuristic algorithm. The algorithm generates *passes* initial tours and then iteratively improves the tours until a local minimum is reached. In each iteration it applies the best possible 2-opt move which means to find the best pair of edges $(i,i+1)$ and $(j,j+1)$ such that replacing them with (i,j) and $(i+1,j+1)$ minimizes the tour length. The default initial tours use RandomTour, however an alternative algorithm can be provided to create the initial tour. Initial tours generated using NearestNeighborHeuristicTSP give good results and performance.

See wikipedia for more details.

This implementation can also be used in order to try to improve an existing tour. See method `EnhancedTwoOptTSP` (in 2.1, page 14)}.

2.1.1 Declaration

```
public class EnhancedTwoOptTSP
 extends <any>
```

2.1.2 Constructor summary

EnhancedTwoOptTSP(int, int, Coords, FlightPlanner) Constructor

2.1.3 Method summary

getTour() Computes a tour by first using JGraphT's TwoOptHeuristicTSP (the superclass of this) to find a short tour using the edge weights in the provided graph, which are straight line (obstacle avoiding) distance measures.

improveTour() (Code unchanged from library code other than type parameters)

2.1.4 Constructors

- **EnhancedTwoOptTSP**

```
public EnhancedTwoOptTSP(int passes,int seed,uk.ac.ed.inf.aqmaps
    .geometry.Coords start,FlightPlanner flightPlanner)
```

  - **Description**

    Constructor

  - **Parameters**

    * `passes` – how many initial random tours to check when running 2-opt
    * `seed` – the random seed
    * `start` – the start position of the drone
    * `flightPlanner` – the FlightPlanner to use for the second 2-opt pass to compute tour weights as the number of moves needed by the drone

2.1.5 Methods

- **getTour**

  ```
  public <any> getTour(<any> graph)
  ```

  - **Description**
    Computes a tour by first using JGraphT's TwoOptHeuristicTSP (the superclass of this) to find a short tour using the edge weights in the provided graph, which are straight line (obstacle avoiding) distance measures. Then, it runs a second pass of 2-opt to further improve upon the tour by instead using a FlightPlanner to generate the actual drone moves along the tour and using the number of moves as the weight of a tour.
  - **Parameters**
    * `graph` – the input sensor graph containing the start location and the sensors, and edge weights of the shortest path between two points which avoids obstacles.
  - **Returns** – the tour as a GraphPath

- **improveTour**

  ```
  public <any> improveTour(<any> graphPath)
  ```

  - **Description**
    (Code unchanged from library code other than type parameters)
    Try to improve a tour by running the 2-opt heuristic using the FlightPlanner to measure the length of tours.
  - **Parameters**
    * `graphPath` – a tour
  - **Returns** – a possibly improved tour

2.2 Class FlightCacheKey

A class which holds data about the input values of the flight planning algorithm from a position to a sensor, potentially with a next sensor. This is for use in the cache, so stores hashed value directly in order to save memory and time calculating extra hashes.

2.2.1 Declaration

```
public class FlightCacheKey
 extends java.lang.Object
```

2.2.2 Constructor summary

FlightCacheKey(Coords, Coords, Coords) Constructor

2.2.3 Method summary

equals(Object) Needed to work with a HashMap, automatically generated by IntelliJ.

hashCode() Since this class stores the hashcode directly, we do not do any computation and just return it.

2.2.4 Constructors

- **FlightCacheKey**

```
public FlightCacheKey(uk.ac.ed.inf.aqmaps.geometry.Coords
    startPosition,uk.ac.ed.inf.aqmaps.geometry.Coords
    currentTarget,uk.ac.ed.inf.aqmaps.geometry.Coords nextTarget)
```

  - **Description**
    Constructor
  - **Parameters**
    - `startPosition` – the start position of the drone.
    - `currentTarget` – the current target
    - `nextTarget` – the next target if there is one, or null otherwise

2.2.5 Methods

- **equals**

```
public boolean equals(java.lang.Object o)
```

  - **Description**
    Needed to work with a HashMap, automatically generated by IntelliJ.

- **hashCode**

```
public int hashCode()
```

  - **Description**
    Since this class stores the hashcode directly, we do not do any computation and just return it.

2.3 Class FlightCacheValue

A class which holds data about the output values of the flight planning algorithm from a position to a sensor, potentially with a next sensor. This does not hold the actual tour, and is only used for the size of the tour, as it would use a lot of memory.

2.3.1 Declaration

```
public class FlightCacheValue
 extends java.lang.Object
```

2.3.2 Constructor summary

FlightCacheValue(int, Coords) Constructor

2.3.3 Method summary

getEndPosition()
getLength()

2.3.4 Constructors

- **FlightCacheValue**

  ```
  public FlightCacheValue(int length,uk.ac.ed.inf.aqmaps.geometry.
      Coords endPosition)
  ```

  - **Description**
    Constructor
  - **Parameters**
    - `length` – the number of moves in this flight path section
    - `endPosition` – the ending position of the drone in this flight path section

2.3.5 Methods

- **getEndPosition**

  ```
  public uk.ac.ed.inf.aqmaps.geometry.Coords getEndPosition()
  ```

  - **Returns** – the ending position of the drone in this flight path section

- **getLength**

  ```
  public int getLength()
  ```

  - **Returns** – the number of moves in this flight path section

2.4 Class FlightPlanner

Handles the creation of a flight plan for the drone. Uses JGraphT's TwoOptHeuristicTSP algorithm as part of process, which was the best performing of JGraphT's Hamiltonian Cycle algorithms, however this could be changed easily.

2.4.1 Declaration

```
public class FlightPlanner
 extends java.lang.Object
```

2.4.2 Constructor summary

FlightPlanner(Obstacles, List, int, double) Construct a flight planner with the given time limit in seconds.

2.4.3 Method summary

computeFlightLength(List) Computes the length of a flight plan which follows the given sensor coordinate tour.

createBestFlightPlan(Coords) Create a flight plan for the drone which visits all sensors and returns to the start.

2.4.4 Constructors

- **FlightPlanner**

  ```
  public FlightPlanner(uk.ac.ed.inf.aqmaps.noflyzone.Obstacles
      obstacles,java.util.List sensorW3Ws,int randomSeed,double
      timeLimit)
  ```

  - **Description**

    Construct a flight planner with the given time limit in seconds. If the time limit is not greater than 0, turns it off and uses a maximum number of iterations instead.

  - **Parameters**

    * `obstacles` – the Obstacles containing the no-fly zones
    * `sensorW3Ws` – the W3W locations of the sensors
    * `randomSeed` – the initial random seed to use
    * `timeLimit` – the time limit for the algorithm in seconds. If it is equal to 0 then disables the time limit and runs for a fixed number of iterations.

2.4.5 Methods

- **computeFlightLength**

```
public int computeFlightLength(java.util.List tour)
```

  - **Description**
    Computes the length of a flight plan which follows the given sensor coordinate tour.
  - **Parameters**
    - `tour` – a list of Coords specifying the order to visit the sensors
  - **Returns** – the number of moves in the flight plan

- **createBestFlightPlan**

```
public java.util.List createBestFlightPlan(uk.ac.ed.inf.aqmaps.
    geometry.Coords startPosition)
```

  - **Description**
    Create a flight plan for the drone which visits all sensors and returns to the start. Runs the algorithm a large number of times with different random seeds, in parallel, and chooses the shortest.
  - **Parameters**
    - `startPosition` – the starting position of the drone
  - **Returns** – a list of Moves representing the flight plan

2.5 Class SensorGraph

A graph of the sensors and their distances to each other, taking into account obstacle evasion.

2.5.1 Declaration

```
public class SensorGraph
 extends <any>
```

2.5.2 Method summary

createWithStartLocation(Coords, Collection, Obstacles) Creates a complete weighted graph with the points of all of the sensors and the starting position.

2.5.3 Methods

- **createWithStartLocation**

```
public static SensorGraph createWithStartLocation(uk.ac.ed.inf.
    aqmaps.geometry.Coords startPosition,java.util.Collection
    sensorCoords,uk.ac.ed.inf.aqmaps.noflyzone.Obstacles
    obstacles)
```

  - **Description**
    Creates a complete weighted graph with the points of all of the sensors and the starting position. The edge weights are the shortest distance between the points, avoiding obstacles if necessary.
  - **Parameters**
    - `startPosition` – the starting position of the drone
    - `sensorCoords` – a Collection of the Coords of the sensors to be visited
    - `obstacles` – the Obstacles that need to be avoided
  - **Returns** – a SensorGraph

2.6 Class WaypointNavigation

A class which handles the navigation of the drone along a series of waypoints to a target. This is the core of the drone control algorithm that plans the movement of the drone itself including all rules about move lengths and directions.

2.6.1 Declaration

```
public class WaypointNavigation
 extends java.lang.Object
```

2.6.2 Field summary

END_POSITION_RANGE The drone must be within this distance of the end position at the end of the flight
MOVE_LENGTH The distance the drone travels in one move.
SENSOR_RANGE The drone must be within this distance of a sensor to be able to read it.

2.6.3 Constructor summary

WaypointNavigation(Obstacles)

2.6.4 Method summary

navigateToLocation(Coords, List, W3W) Find a sequence of moves that navigates the drone from the current location along the waypoints to the target.

2.6.5 Fields

- `public static final double` **MOVE_LENGTH**
  - The distance the drone travels in one move.
- `public static final double` **SENSOR_RANGE**
  - The drone must be within this distance of a sensor to be able to read it.
- `public static final double` **END_POSITION_RANGE**
  - The drone must be within this distance of the end position at the end of the flight

2.6.6 Constructors

- **WaypointNavigation**

```
public WaypointNavigation(uk.ac.ed.inf.aqmaps.noflyzone.
    Obstacles obstacles)
```

  - **Parameters**
    - `obstacles` – the obstacles for collision checking

2.6.7 Methods

- **navigateToLocation**

```
public java.util.List navigateToLocation(uk.ac.ed.inf.aqmaps.
    geometry.Coords startingPosition,java.util.List waypoints,uk.
    ac.ed.inf.aqmaps.W3W targetSensorW3W)
```

  - **Description**
    Find a sequence of moves that navigates the drone from the current location along the waypoints to the target.
  - **Parameters**
    - `startingPosition` – the starting position of the drone
    - `waypoints` – a list of Coords waypoints for the drone to follow on its way to the target.
    - `targetSensorW3W` – the W3W of the target sensor, or null if the target is not a sensor.
  - **Returns** – a list of Moves that navigate the drone from the starting position to in range of the target

Chapter 3

Package uk.ac.ed.inf.aqmaps.geometry

Package Contents *Page*

Classes

3.1 Class Coords

Holds a longitude and latitude pair, using a Point2D. This class uses euclidean geometry and its calculations do **not** match with real life.

3.1.1 Declaration

```
public class Coords
 extends java.awt.geom.Point2D.Double
```

3.1.2 Constructor summary

Coords(double, double)

3.1.3 Method summary

bisectorDirection(Coords, Coords) Let this point be P.
buildFromGeojsonPoint(Point) Convert a mapbox point into a Coords
directionTo(Coords) Let this point be P.
getPointOnBisector(Coords, Coords, double) Let this point be P.

getPositionAfterMoveDegrees(double, double) Creates a new Coords which is the result of moving from the current location at the specified angle for the specified length.

getPositionAfterMoveRadians(double, double) Creates a new Coords which is the result of moving from the current location at the specified angle for the specified length.

roundedDirection10Degrees(Coords, int) Let this point be P.

toString() Used for printing Coords for debugging

3.1.4 Constructors

- **Coords**

```
public Coords(double lng,double lat)
```

  - **Parameters**
    - `lng` – longitude
    - `lat` – latitude

3.1.5 Methods

- **bisectorDirection**

```
public double bisectorDirection(Coords A,Coords B)
```

  - **Description**
    Let this point be P. Calculate the direction of the acute bisector between the lines PA and PB.
  - **Parameters**
    - `A` – point A
    - `B` – point B
  - **Returns** – the direction of the bisector in radians

- **buildFromGeojsonPoint**

```
public static Coords buildFromGeojsonPoint(Point p)
```

  - **Description**
    Convert a mapbox point into a Coords
  - **Parameters**
    - `p` – the mapbox Point
  - **Returns** – an equivalent Coords

- **directionTo**

  ```
  public double directionTo(Coords A)
  ```

  - **Description**
    Let this point be P. Calculates the direction or angle of the line PA with respect to the horizontal, where east is 0, north is pi/2, south is -pi/2, west is pi
  - **Parameters**
    - `A` – point A
  - **Returns** – the direction in radians

- **getPointOnBisector**

  ```
  public Coords getPointOnBisector(Coords A, Coords B, double
      distance)
  ```

  - **Description**
    Let this point be P. Creates a point a specified distance away in the direction of the acute bisector between the lines PA and PB.
  - **Parameters**
    - `A` – point A
    - `B` – point B
    - `distance` – the distance the new point should be away from this point
  - **Returns** – the new point

- **getPositionAfterMoveDegrees**

  ```
  public Coords getPositionAfterMoveDegrees(double degrees, double
      length)
  ```

  - **Description**
    Creates a new Coords which is the result of moving from the current location at the specified angle for the specified length. Angle in degrees version.
  - **Parameters**
    - `degrees` – the direction of the move as an angle in degrees
    - `length` – the length of the move
  - **Returns** – a Coords containing the calculated point

- **getPositionAfterMoveRadians**

  ```
  public Coords getPositionAfterMoveRadians(double radians, double
      length)
  ```

- **Description**
  Creates a new Coords which is the result of moving from the current location at the specified angle for the specified length. Angle in radians version.
- **Parameters**
  - `radians` – the direction of the move as an angle in radians
  - `length` – the length of the move
- **Returns** – a Coords containing the calculated point

- **roundedDirection10Degrees**

```
public int roundedDirection10Degrees(Coords A, int offset)
```

  - **Description**
    Let this point be P. Calculates the direction of the line PA, rounded to the nearest 10 degrees, offset by an amount, and expressed in the range [0,350].
  - **Parameters**
    - `A` – point A
    - `offset` – the offset
  - **Returns** – the direction in degrees

- **toString**

```
public java.lang.String toString()
```

  - **Description**
    Used for printing Coords for debugging

3.1.6 Members inherited from class Point2D.Double

`java.awt.geom.Point2D.Double`

- `public double` **getX**`()`
- `public double` **getY**`()`
- `public void` **setLocation**`(double` **arg0**`, double` **arg1**`)`
- `public String` **toString**`()`
- `public` **x**
- `public` **y**

3.1.7 Members inherited from class Point2D

`java.awt.geom.Point2D`

- `public Object` **clone**`()`
- `public double` **distance**`(double` **arg0**`, double` **arg1**`)`
- `public static double` **distance**`(double` **arg0**`, double` **arg1**`, double` **arg2**`, double` **arg3**`)`
- `public double` **distance**`(Point2D` **arg0**`)`

- `public double` **distanceSq**`(double` **arg0**`, double` **arg1**`)`
- `public static double` **distanceSq**`(double` **arg0**`, double` **arg1**`, double` **arg2**`, double` **arg3**`)`
- `public double` **distanceSq**`(Point2D` **arg0**`)`
- `public boolean` **equals**`(java.lang.Object` **arg0**`)`
- `public abstract double` **getX**`()`
- `public abstract double` **getY**`()`
- `public int` **hashCode**`()`
- `public abstract void` **setLocation**`(double` **arg0**`, double` **arg1**`)`
- `public void` **setLocation**`(Point2D` **arg0**`)`

3.2 Class Polygon

Holds a polygon as a list of the Coords that make up the vertices, in order

3.2.1 Declaration

```
public class Polygon
 extends java.lang.Object
```

3.2.2 Field summary

OUTLINE_MARGIN The margin to use when generating a polygon which outlines another, see `generateOutlinePoints()`

3.2.3 Method summary

buildFromFeature(Feature) Create a Polygon from a GeoJSON Polygon
contains(Coords) Checks whether a given point is containing within this polygon.
generateOutlinePoints() Generates the points of a new polygon which contains the original by a very small margin.
getPoints()
getSegments() This method is currently only used in a test, but it is kept to test whether the segments have been created properly.
lineCollision(Coords, Coords)

3.2.4 Fields

- `public static final double` **OUTLINE_MARGIN**
  - The margin to use when generating a polygon which outlines another, see `generateOutlinePoints()`

3.2.5 Methods

- **buildFromFeature**

  ```
  public static Polygon buildFromFeature(Feature feature)
  ```

  - **Description**
    Create a Polygon from a GeoJSON Polygon
  - **Parameters**
    - `feature` – a GeoJSON Feature containing a Polygon
  - **Returns** – the converted Polygon

- **contains**

  ```
  public boolean contains(Coords p)
  ```

  - **Description**
    Checks whether a given point is containing within this polygon.
  - **Parameters**
    - `p` – the point
  - **Returns** – true if the polygon contains the point, false otherwise

- **generateOutlinePoints**

  ```
  public java.util.List generateOutlinePoints()
  ```

  - **Description**
    Generates the points of a new polygon which contains the original by a very small margin. It generates points a distance of 1.0e-14 from each point in the original Polygon in the direction of the bisecting angle between the two adjacent sides, or the opposite direction if that point is inside the polygon. The resulting polygon will be larger than the original by a margin of 1.0e-14 on all sides.
  - **Returns** – the outlining Polygon

- **getPoints**

  ```
  public java.util.List getPoints()
  ```

  - **Returns** – the points which make up the vertices of this polygon

- **getSegments**

```
public java.util.List getSegments()
```

- **Description**
  This method is currently only used in a test, but it is kept to test whether the segments have been created properly.
- **Returns** – the segments which make up the edges of the polygon

- **lineCollision**

```
public boolean lineCollision(Coords start,Coords end)
```

Chapter 4

Package uk.ac.ed.inf.aqmaps.io

Package Contents *Page*

Interfaces

Classes

4.1 Interface InputController

Handles interaction with input from all remote information sources and devices. Gets information on sensor locations, no-fly zones, W3W locations, and sensor readings. Implementations of the interface can gather the information from any source, such as a simple web server for testing purposes, or from a full system where data is also read from real sensors.

4.1.1 Declaration

```
public interface InputController
```

4.1.2 All known subinterfaces

ServerInputController (in 4.5, page 34)

4.1.3 All classes known to implement interface

ServerInputController (in 4.5, page 34)

4.1.4 Method summary

getNoFlyZones() Gets information about no-fly zones from a remote source
getSensorW3Ws() Gets the list of sensors that need to be visited from a remote source
readSensor(W3W) Reads information from the sensor at the provided W3W location

4.1.5 Methods

- **getNoFlyZones**

```
java.util.List getNoFlyZones()
```

  - **Description**
    Gets information about no-fly zones from a remote source
  - **Returns** – a FeatureCollection containing the locations of the no-fly zones

- **getSensorW3Ws**

```
java.util.List getSensorW3Ws()
```

  - **Description**
    Gets the list of sensors that need to be visited from a remote source
  - **Returns** – a list of W3W locations of the sensors

- **readSensor**

```
uk.ac.ed.inf.aqmaps.Sensor readSensor(uk.ac.ed.inf.aqmaps.W3W
    location)
```

  - **Description**
    Reads information from the sensor at the provided W3W location
  - **Parameters**
    * `location` – the location of the sensor as a W3W class
  - **Returns** – a Sensor object representing the current status of the sensor

4.2 Interface OutputController

Handles interaction with all output locations. Implementations may output to any source, such as to a file or to a server.

4.2.1 Declaration

```
public interface OutputController
```

4.2.2 All known subinterfaces

FileOutputController (in 4.4, page 33)

4.2.3 All classes known to implement interface

FileOutputController (in 4.4, page 33)

4.2.4 Method summary

outputFlightpath(String) Outputs the flightpath planned by the drone
outputMapGeoJSON(String) Outputs the GeoJSON map containing the flightpath and the sensor readings collected by the drone

4.2.5 Methods

- **outputFlightpath**

  ```
  void outputFlightpath(java.lang.String flightpathText)
  ```

  - **Description**
    Outputs the flightpath planned by the drone
  - **Parameters**
    * `flightpathText` – the String containing the flightpath data

- **outputMapGeoJSON**

  ```
  void outputMapGeoJSON(java.lang.String json)
  ```

  - **Description**
    Outputs the GeoJSON map containing the flightpath and the sensor readings collected by the drone
  - **Parameters**
    * `json` – the GeoJSON String

4.3 Interface Server

Handles requesting data from a server.

4.3.1 Declaration

```
public interface Server
```

4.3.2 All known subinterfaces

WebServer (in 4.6, page 36)

4.3.3 All classes known to implement interface

WebServer (in 4.6, page 36)

4.3.4 Method summary

requestData(String) Request the data that is located at the given URL.

4.3.5 Methods

- **requestData**

  ```
  java.lang.String requestData(java.lang.String url)
  ```

  - **Description**
    Request the data that is located at the given URL. Will cause a fatal error if it cannot connect to the server, or if the requested file is not found.
  - **Parameters**
    - `url` – the URL of the file to request
  - **Returns** – the requested data as a String

4.4 Class FileOutputController

Outputs to the current directory of the filesystem

4.4.1 Declaration

```
public class FileOutputController
 extends java.lang.Object implements OutputController
```

4.4.2 Constructor summary

FileOutputController(Settings)

4.4.3 Method summary

outputFlightpath(String)
outputMapGeoJSON(String)

4.4.4 Constructors

- **FileOutputController**

```
public FileOutputController(uk.ac.ed.inf.aqmaps.Settings
    settings)
```

  - **Parameters**
    - `settings` – a Settings holding the current input arguments

4.4.5 Methods

- **outputFlightpath**

```
void outputFlightpath(java.lang.String flightpathText)
```

  - **Description copied from OutputController** (**in 4.2, page 31**)
    Outputs the flightpath planned by the drone
  - **Parameters**
    - `flightpathText` – the String containing the flightpath data

- **outputMapGeoJSON**

```
void outputMapGeoJSON(java.lang.String json)
```

  - **Description copied from OutputController** (**in 4.2, page 31**)
    Outputs the GeoJSON map containing the flightpath and the sensor readings collected by the drone
  - **Parameters**
    - `json` – the GeoJSON String

4.5 Class ServerInputController

Implements the Remote interface using a connection to a simple web server.

4.5.1 Declaration

```
public class ServerInputController
 extends java.lang.Object implements InputController
```

4.5.2 Constructor summary

ServerInputController(Server, int, int, int, int) Create a new ServerInputController instance with the given Server, date, and port number
ServerInputController(Settings) Create a new ServerInputController instance with the given settings

4.5.3 Method summary

getNoFlyZones()
getSensorW3Ws()
readSensor(W3W)

4.5.4 Constructors

- **ServerInputController**

```
public ServerInputController(Server server,int day,int month,int year,int port)
```

  - **Description**
    Create a new ServerInputController instance with the given Server, date, and port number
  - **Parameters**
    - `server` – a Server
    - `day` – the day
    - `month` – the month
    - `year` – the year
    - `port` – the port of the server

- **ServerInputController**

```
public ServerInputController(uk.ac.ed.inf.aqmaps.Settings settings)
```

  - **Description**
    Create a new ServerInputController instance with the given settings
  - **Parameters**
    - `settings` – the Settings object containing the current settings

4.5.5 Methods

- **getNoFlyZones**

```
java.util.List getNoFlyZones()
```

- **Description copied from InputController** (**in 4.1, page 30**)
  Gets information about no-fly zones from a remote source
- **Returns** – a FeatureCollection containing the locations of the no-fly zones

- **getSensorW3Ws**

```
java.util.List getSensorW3Ws()
```

- **Description copied from InputController** (**in 4.1, page 30**)
  Gets the list of sensors that need to be visited from a remote source
- **Returns** – a list of W3W locations of the sensors

- **readSensor**

```
uk.ac.ed.inf.aqmaps.Sensor readSensor(uk.ac.ed.inf.aqmaps.W3W
    location)
```

- **Description copied from InputController** (**in 4.1, page 30**)
  Reads information from the sensor at the provided W3W location
- **Parameters**
  - `location` – the location of the sensor as a W3W class
- **Returns** – a Sensor object representing the current status of the sensor

4.6 Class WebServer

Handles requesting data from an HTTP server.

4.6.1 Declaration

```
public class WebServer
 extends java.lang.Object implements Server
```

4.6.2 Constructor summary

WebServer()

4.6.3 Method summary

requestData(String)

4.6.4 Constructors

- **WebServer**

```
public WebServer()
```

4.6.5 Methods

- **requestData**

```
java.lang.String requestData(java.lang.String url)
```

  - **Description copied from Server (in 4.3, page 32)**
    Request the data that is located at the given URL. Will cause a fatal error if it cannot connect to the server, or if the requested file is not found.
  - **Parameters**
    - `url` – the URL of the file to request
  - **Returns** – the requested data as a String

Chapter 5

Package uk.ac.ed.inf.aqmaps.noflyzone

Package Contents *Page*

Classes

5.1 Class ObstacleGraph

A graph of the vertices of the obstacles and edges between them if they have line of sight.

5.1.1 Declaration

```
public class ObstacleGraph
 extends <any>
```

5.1.2 Method summary

prepareGraph(List, Obstacles) Prepare a weighted graph containing all points which form an outline around the polygons as vertices, and edges connecting them if they have line of sight, which have a weight equal to the distance between them.

5.1.3 Methods

- **prepareGraph**

```
public static ObstacleGraph prepareGraph(java.util.List
    outlinePoints,Obstacles obstacles)
```

  - **Description**
    Prepare a weighted graph containing all points which form an outline around the polygons as vertices, and edges connecting them if they have line of sight, which have a weight equal to the distance between them. The graph uses outline polygons since if it used the original polygons, their points would occupy the same location and any line emerging from the corner of an obstacle would be considered to be colliding with it. See see `generateOutlinePoints()` .
  - **Parameters**
    - `outlinePoints` – the points which form the outline of the obstacle polygons
    - `obstacles` – the obstacles to prepare the graph for
  - **Returns** – a graph representation of the obstacles

5.2 Class ObstaclePathfinder

Handles obstacle evasion. Uses Obstacles and an ObstacleGraph to find paths between points which do not collides with any obstacles.

5.2.1 Declaration

```
public class ObstaclePathfinder
 extends java.lang.Object
```

5.2.2 Constructor summary

ObstaclePathfinder(ObstacleGraph, Obstacles) Construct an Obstacle evader with the given graph and obstacles.

5.2.3 Method summary

getPathBetweenPoints(Coords, Coords) Find the shortest path between the start and end points, navigating around obstacles if necessary.
getShortestPathLength(Coords, Coords) Find the length of the shortest path between the start and end points, navigating around obstacles if necessary.

5.2.4 Constructors

- **ObstaclePathfinder**

```
public ObstaclePathfinder(ObstacleGraph graph, Obstacles
    obstacles)
```

  - **Description**
    Construct an Obstacle evader with the given graph and obstacles.
  - **Parameters**
    - `graph` – a graph of the obstacles
    - `obstacles` – the Obstacles

5.2.5 Methods

- **getPathBetweenPoints**

```
public java.util.List getPathBetweenPoints(uk.ac.ed.inf.aqmaps.
    geometry.Coords start,uk.ac.ed.inf.aqmaps.geometry.Coords end
    )
```

  - **Description**
    Find the shortest path between the start and end points, navigating around obstacles if necessary.
  - **Parameters**
    - `start` – the starting point
    - `end` – the ending point
  - **Returns** – a list of points specifying the route

- **getShortestPathLength**

```
public double getShortestPathLength(uk.ac.ed.inf.aqmaps.geometry
    .Coords start,uk.ac.ed.inf.aqmaps.geometry.Coords end)
```

  - **Description**
    Find the length of the shortest path between the start and end points, navigating around obstacles if necessary. Euclidean distance is used as the length measure.
  - **Parameters**
    - `start` – the starting point
    - `end` – the ending point
  - **Returns** – the length of the path in degrees

5.3 Class Obstacles

Holds information about the obstacles or no-fly zones that the drone must avoid.

5.3.1 Declaration

```
public class Obstacles
 extends java.lang.Object
```

5.3.2 Field summary

BOTTOM_RIGHT A Point representing the southeast corner of the confinement area.
TOP_LEFT A Point representing the northwest corner of the confinement area.

5.3.3 Constructor summary

Obstacles(List) Constructs Obstacles out of a list of Polygons specifying their locations

5.3.4 Method summary

getObstaclePathfinder() Gets an ObstaclePathfinder using these Obstacles.
isInConfinement(Coords) Determines whether or not a point is inside the confinement area
lineCollision(Coords, Coords) Determines whether the line segment between the start and end points collides with a obstacle.
pointCollides(Coords) Determine whether the given point is inside an obstacle, or outside the confinement area.

5.3.5 Fields

- `public static final uk.ac.ed.inf.aqmaps.geometry.Coords` **TOP_LEFT**
  - A Point representing the northwest corner of the confinement area.
- `public static final uk.ac.ed.inf.aqmaps.geometry.Coords` **BOTTOM_RIGHT**
  - A Point representing the southeast corner of the confinement area.

5.3.6 Constructors

- **Obstacles**

```
public Obstacles(java.util.List polygons)
```

- **Description**
  Constructs Obstacles out of a list of Polygons specifying their locations
- **Parameters**
  - `polygons` – the Polygons which make up the obstacles

5.3.7 Methods

- **getObstaclePathfinder**

```
public ObstaclePathfinder getObstaclePathfinder()
```

  - **Description**
    Gets an ObstaclePathfinder using these Obstacles. The ObstaclePathfinder uses a clone of the obstacle graph, allowing it to be used concurrently with other ObstaclePathfinder.
  - **Returns** – an ObstaclePathfinder instance with these obstacles

- **isInConfinement**

```
public boolean isInConfinement(uk.ac.ed.inf.aqmaps.geometry.
    Coords point)
```

  - **Description**
    Determines whether or not a point is inside the confinement area
  - **Parameters**
    - `point` – the point to examine
  - **Returns** – true if the point is inside the confinement area, false otherwise

- **lineCollision**

```
public boolean lineCollision(uk.ac.ed.inf.aqmaps.geometry.Coords
    start,uk.ac.ed.inf.aqmaps.geometry.Coords end)
```

  - **Description**
    Determines whether the line segment between the start and end points collides with a obstacle.
  - **Parameters**
    - `start` – the coordinates of the start point
    - `end` – the coordinates of the end point
  - **Returns** – true if the segment collides with an obstacle, false otherwise

- **pointCollides**

public boolean pointCollides(uk.ac.ed.inf.aqmaps.geometry.Coords coords)

- **Description**
  Determine whether the given point is inside an obstacle, or outside the confinement area. This is currently only used in testing to generate random starting points.
- **Parameters**
  - `coords` – the point
- **Returns** – true if there is a collision, false otherwise