**Algorithm Description**

The goal of this algorithm is to create a path between the current position of the plan and the next waypoint that minimizes the risk of collision with obstacles.

It will do this by locating possible collision areas, deciding between dodge directions, and determining the least risky path for each direction.

This algorithm will be a core component of the dynamic pathfinding module in the avionics software. It will interact with the server intermediary code as well as the pixhawk communication code.

**Inputs:**

* Current Path
* Obstacle objects (Locations and size information)
* Current location of plane
* Fly boundaries

**Output:**

* Modified coordinate stack to be uploaded to the Pixhawk

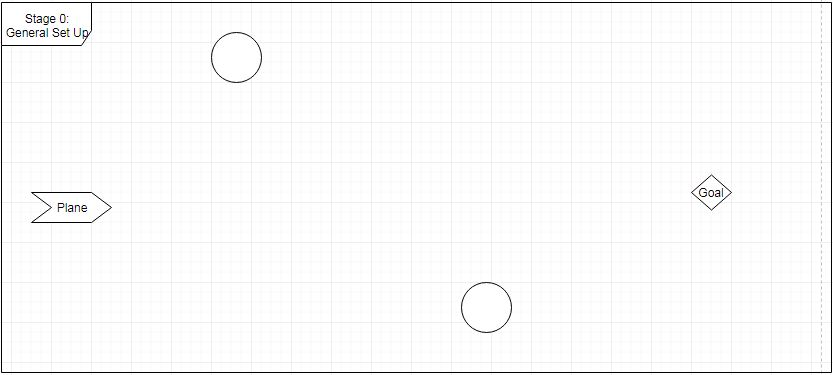
**Algorithm Stages**

1. Check for predicted collisions
2. Create dodge anchor points
3. Generate Prediction Equations for dynamic obstacles
4. Generate alternative paths
5. Evaluate alternative paths
6. Select dodge point
7. Repeat algorithm with new points. Iterate until a sufficient path to the objective is determined.

**Stage 0: Basic Set Up**

Inputs: Object and plane previous locations

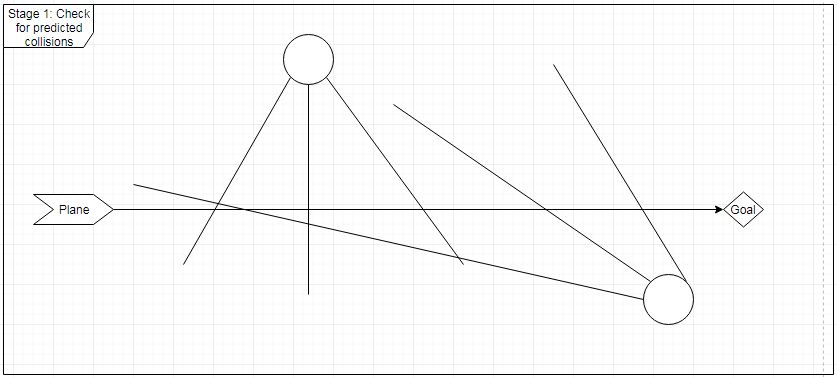
Outputs: Direction vectors for the objects and plane.



**Stage 1: Check for predicted collisions**

Inputs: Obstacle objects, plane location and heading, coordinate of next waypoint.

Output: List of possible collision locations.



Description:

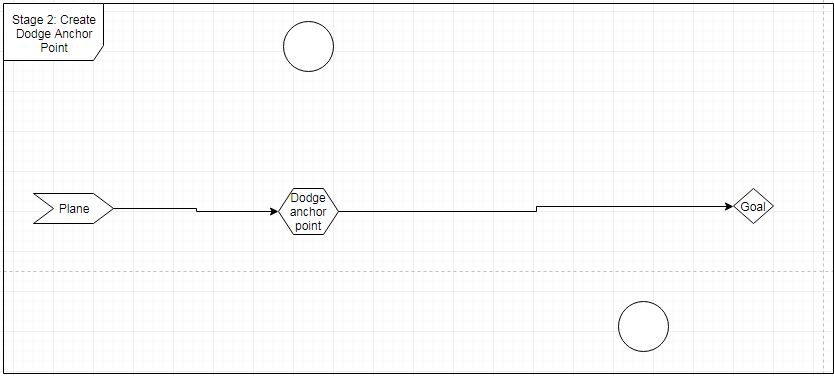
Create three vectors going from each dynamic obstacles location, at -45, 0, and 45 degrees from its current heading. Check each vector for intersection with the path. If there are intersections, average the coordinates of the intersections and remember that value. For example, if two vectors hit, find where each of them intersect and choose the location in the middle as a possible anchor point.

For static obstacles, create four vectors boxing in the object. Then do the same thing. (Or if there is a better way to do this, do that.

**Stage 2: Create dodge anchor point**

**Input:** Possible Collision Locations

**Output:** Anchor point for the dodge nodes

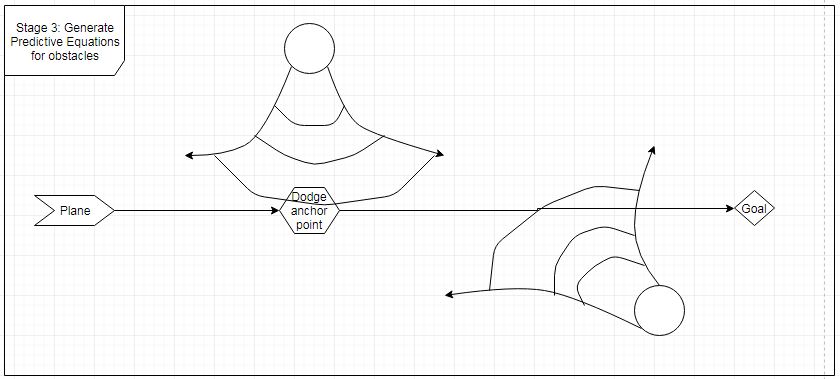


**Overview:** Among the possible collision locations we want to select the one that is closest to the current location of the plane. This should just be a simple sort. After finding the closest collision, remember its location for stage 4.

**Stage 3: Generate Prediction Equations for dynamic obstacles**

**Input:** Obstacle Objects

**Output:** Equations that represent the predicted future locations of the obstacles.



Overview: I put this stage before the next one because I think that it would be efficient to have the equations pre-prepared and ready to be rapidly run for the evaluation stage. The equations needed to determine the risk level of a certain point in space due to an obstacle are the following.

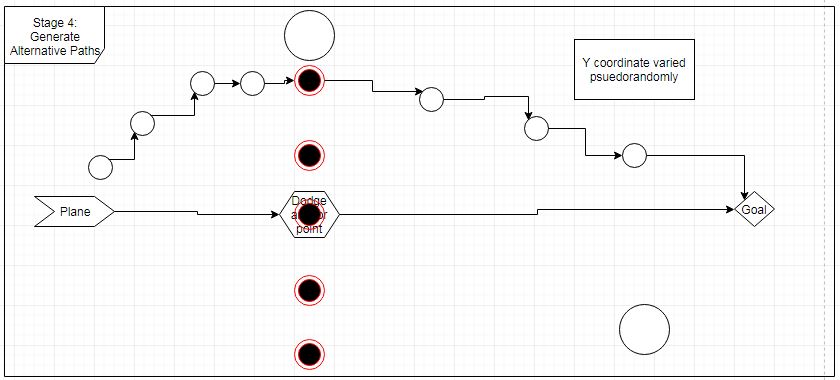
* Inner radius: 0.5-1 Seconds back from the predicted distance that the obstacle will go.
* Outer radius: 0.5 to 1 seconds forward from the predicted distance that the obstacle will go.
* Right and left curves: These can be represented by two more circles. These circles will be centered perpendicularly to the path of the obstacle at a distance of: obstacle turn radius + ½ obstacle diameter.
* Direction Check: With the above equations, locations behind the obstacle could be included. To make sure that the selected area is in front of the obstacle, use a dot product of the obstacle direction and (test point – current obstacle location) to confirm that the test point is in front of the obstacle.

Outcome: These equations should combine to test whether, for a certain time value, a point in space has a risk of collision. This risk of collision should be represented by:

**Stage 4: Generate alternative paths**

**Input:** Anchor point

**Output:** Possible paths that the plane could take



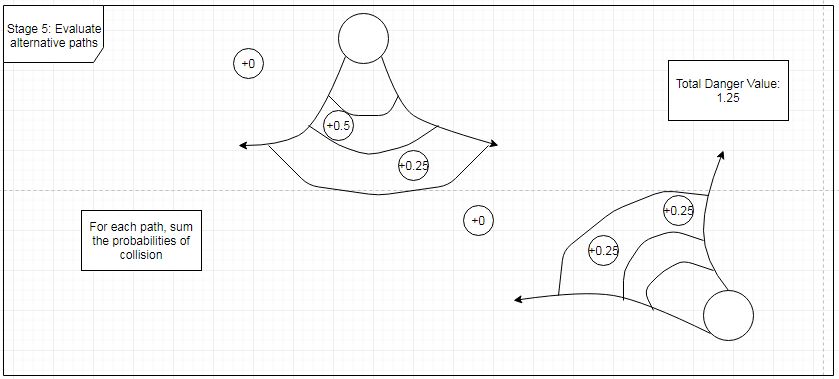
Overview: This stage is closely intertwined with stage 5. It begins by creating five possible dodge points. One point is on the location of the dodge anchor point. Two are perpendicularly above this point at X distance from each other. Two are perpendicularly below this point at X distance from each other.

For each of these dodge points, X possible paths will be created. These paths will be created using Y number of pathing points that we will place initially on the straight lines between the plane, the dodge point, and the goal. After placing these points initially, we will vary their perpendicular distance from the path pseudo-randomly (Kyle can create this). We will do this X number of times for each dodge point, evaluating the created path after each time.

**Stage 5: Evaluate alternative paths**

**Input:** Possible path

**Output:** Path risk values, average risk value for each dodge node.



Overview: With each of the paths that we created, we need to evaluate how risky it is. To do this we will determine the risk value for each point in the path. To do this, determine whether each point is in the region of any of the obstacles. If so, add the area/area value that we determined in stage three to the overall risk value for the path. At the end of this calculation, there should be a overall risk value for the path.

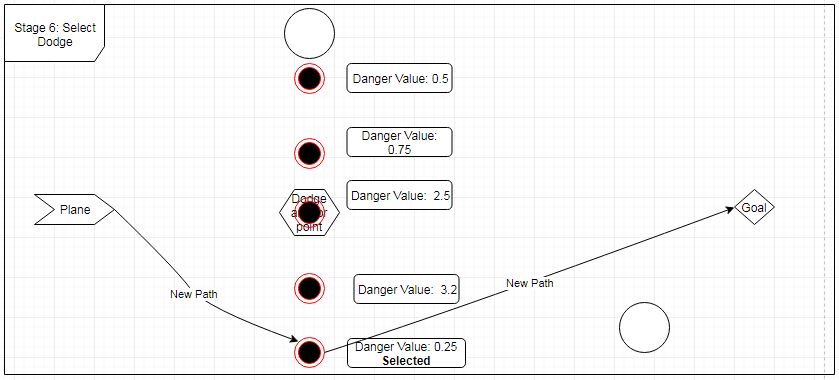
Additionally, as we calculate the risk value for each path, we will maintain an average risk value number for each dodge node.

Finally, we want to choose the best path, so, for each dodge node, keep a record of the safest path and its risk value.

**Stage 6: Select Dodge Points**

**Input:** Average and best case risk values for each dodge node.

**Output:** Altered path

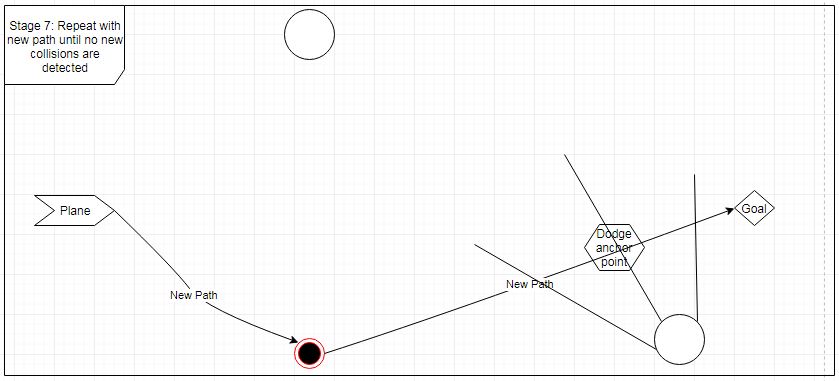


Overview: Determine which dodge node has the lowest average risk value. For this dodge node, select the path that had the lowest risk value. Change the path of the plane to be this new path.

**Stage 7: Repeat algorithm between selected point and end point until a sufficient path has been determined.**

**Input:** Chosen dodge node from last stage.

**Output:** The rest of the path to the objective.



Overview: To determine a full path, we must repeat this algorithm until one has been determined. To do this, run the algorithm again, this time between the selected dodge node and the goal.

**Testing:**

The best case testing for this algorithm would be a visual view of the predicted obstacle movement and the chosen path. This is probably too much to do though, so in lieu of this, we might be able to use Q ground control or a simple graphing utility to visually check whether it works. Otherwise, we should test each stage individually and watch out for any craze path deviations. Worst case scenario, the path should default to the straight line between the plane and the objective.

**Edge Cases:**

There are a few edge cases of note to watch out for:

A created path goes outside of the boundaries.

Restrict variation of pathing nodes to stay within bounds.

The obstacles create a really crazy situation.

This algorithm should still minimize damage. However, it might be worth it to create a method to drop the plane’s altitude or something to dodge.

There is a dangerous area immediately following a waypoint.

This algorithm should reduce the odds of a collision at a waypoint. However, in the future, we might want to create something that checks and deals with this possibility.