Level 3 - Simple Planning with Obstacles (Waypoints)

The goal of level 3 planning is to generate a path for the end-effector that avoids obstacles in a way that is both quick to generate and effective. The current implementation does not support learning or ethological categories. This is justified by the fact that any movement produced using level 3 will be saved in memory automatically by level 1, and in the future it can be accessed again by memory-lookup in level 1.

The term “waypoint” comes from the fact that adding points to the new path to go around obstacles generates a new path. The “way” around the obstacle is through these generated points.

To determine which points are valid choices for waypoints, we need a way to quickly determine if a path through a point is clear of obstacles. We assume (for speed) that a clear linear line-of-sight between two points constitutes a clear path. This may lead to problems in that when Level 2 generates a movement for a given path, it does not follow a straight line. In other words, a path may be clear by linear line of sight, but the best movement to match that path may still encounter obstacles.

To correct for the above problem, we add a second type of waypoint: one that moves the path away from a specified point, rather than searching for obstacles. In this version, the algorithm again loops over each segment, but it does not check for line of sight. Instead, it keeps track of which feature of the path was closest to the given point. A feature may be either a vertex or segment. If the given point can be projected onto a segment such that it lies between the endpoints of the segment, then the normal distance (point to segment) is used to find minimum distance. If, on the other hand, the given point is closer to one of the vertices of the path than a segment or it is not in the region perpendicular to any segment, then the closest vertex is considered the closest ‘feature’ (see figure 1).

If the closest feature to the point that we are trying to avoid is a vertex, that vertex will be *replaced*. If the closest feature is a segment, the waypoint will be inserted into the middle of that segment. The point to add is determined by reflecting the given point (which we are avoiding) across the closest-feature and scaling the reflection by some weight factor. For example, if reflecting across a vertex, the equation is `np = cv+w*(cv-g)`, where `np` is the new point, `cv` is the closest vertex, `w` is the weight, and `g` is the given point. To reflect across a segment, simply replace `cv` with the point `g` projected onto the segment.

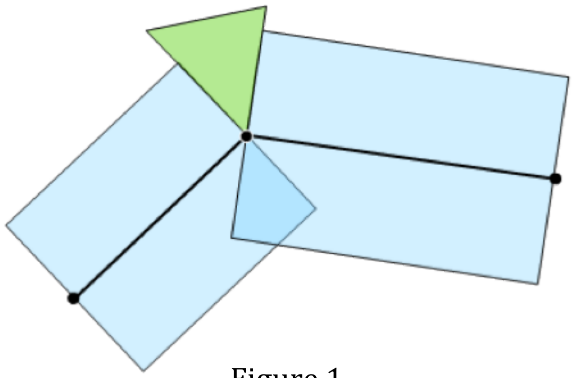


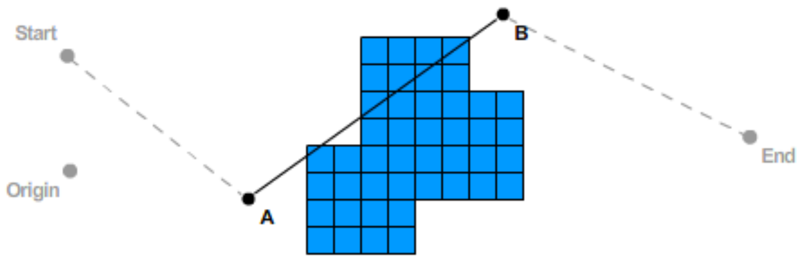
Figure 1.
Dark circles are points on a path. The dark lines between them are the path segments. All points in a blue region are closest to a segment. All points in a green region are closest to a vertex.

When a new path is generated by level 3, it is given to level 2 to generate a movement. If the movement results in a collision, level 3 is called again and given the location of the collision as `g` – the point to avoid. Level 3 then outputs a new path that takes special effort to circumvent the collision location. In this way, the waypoints algorithm responds to failures in movement generation.

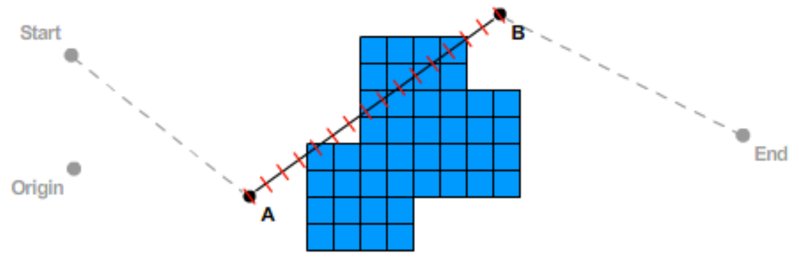
If, after some set number of attempts, levels 3 and 2 aren't able to coordinate and generate a successful movement, we give up and go to level 4.

Visual Description of Waypoints

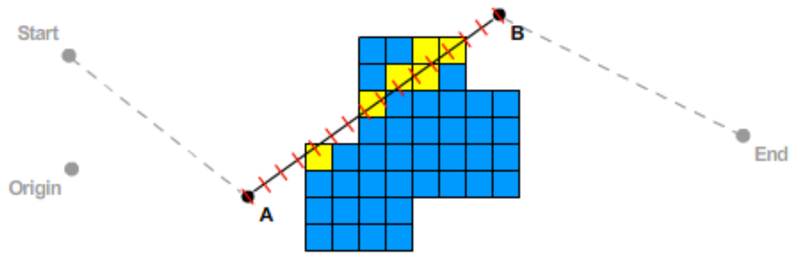
1. Given path and obstacles. Current segment highlighted.



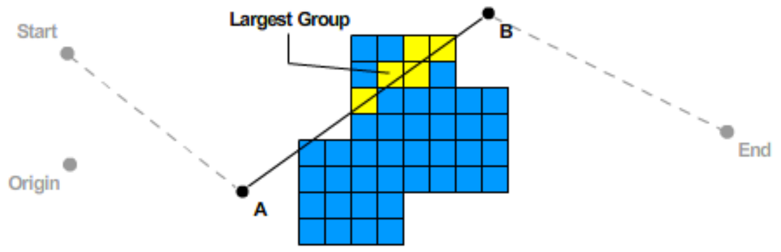
2. Locations at which segment is checked for obstacle



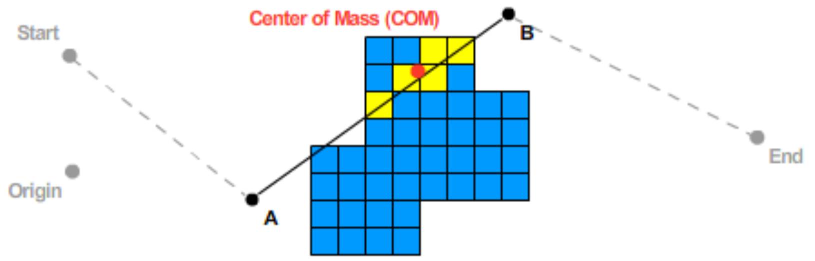
3. Hit locations highlighted



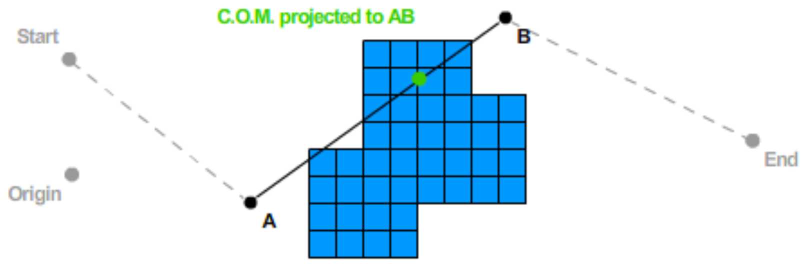
4. Largest group isolated



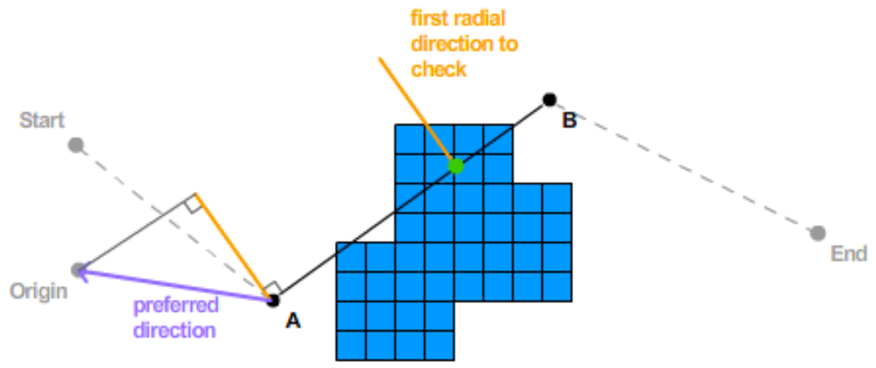
5. Center of Mass (COM) calculated for largest group



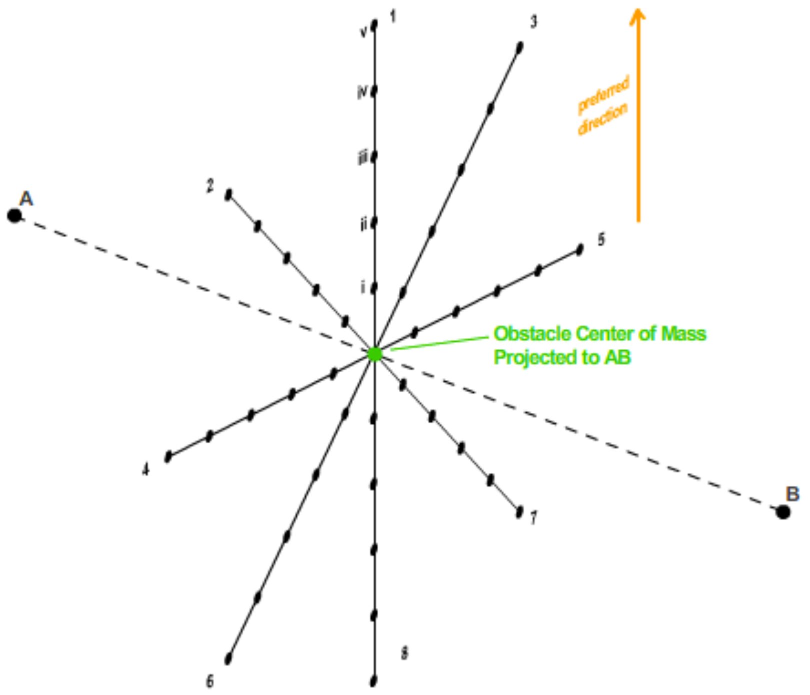
6. COM projected onto segment AB



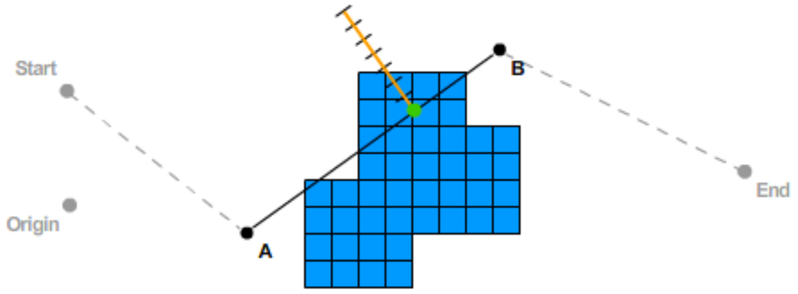
7. *Preferred Direction* is the first direction to start looking for free space around COM. This is set to –A so that preference is given to movements back to center rather than farther out. Search is actually done in the direction of *preferred direction* projected to the plane defined by AB as the normal.



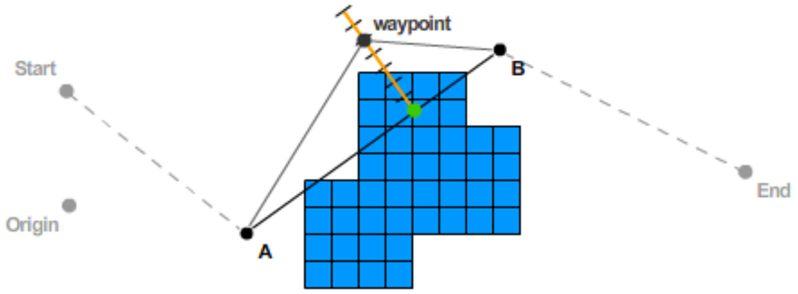
8. The order in which points are checked. Outside loop: large numbers. Inside loop: small roman numerals. The entire spoke along the preferred direction is checked before checking spokes on either side simultaneously (i.e. 2i, 3i, 2ii, 3ii, 2iii, 3iii, etc…). Also note that the points checked are in the plane defined by the normal AB.



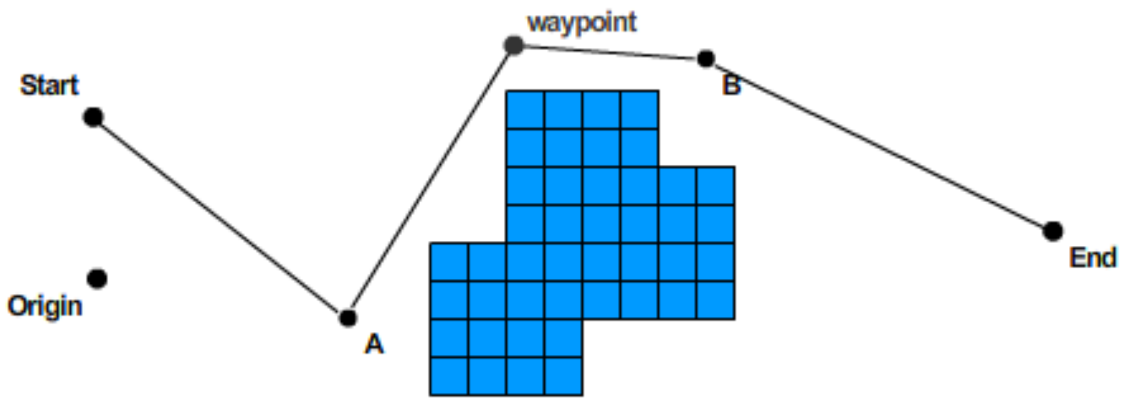
9. Only spoke 1 needs to be checked in our example (checked at tick marks)



10. The first point checked to have a line of sight to *both* A and B is taken as the waypoint. If no such point exists, then the first point found that had one open line of sight is used. If still no point exists, the first point that was not in an obstacle is used. In this example, there is a point on the preferred direction that has a line of sight to both A and B.



11. The resulting path


waypoint
Start
B
Origin
A
End