Christine Talbot
ITCS 8151
Project Writeup
Due 4/25/13

Problem Description

This project investigates one path planning method (EP/N) and implements it. The focus is on the Evolutionary Planner / Navigator method. It utilizes a planar rigid body (mobile) robot, with circles for physical obstacles to avoid in the workspace of the robot.

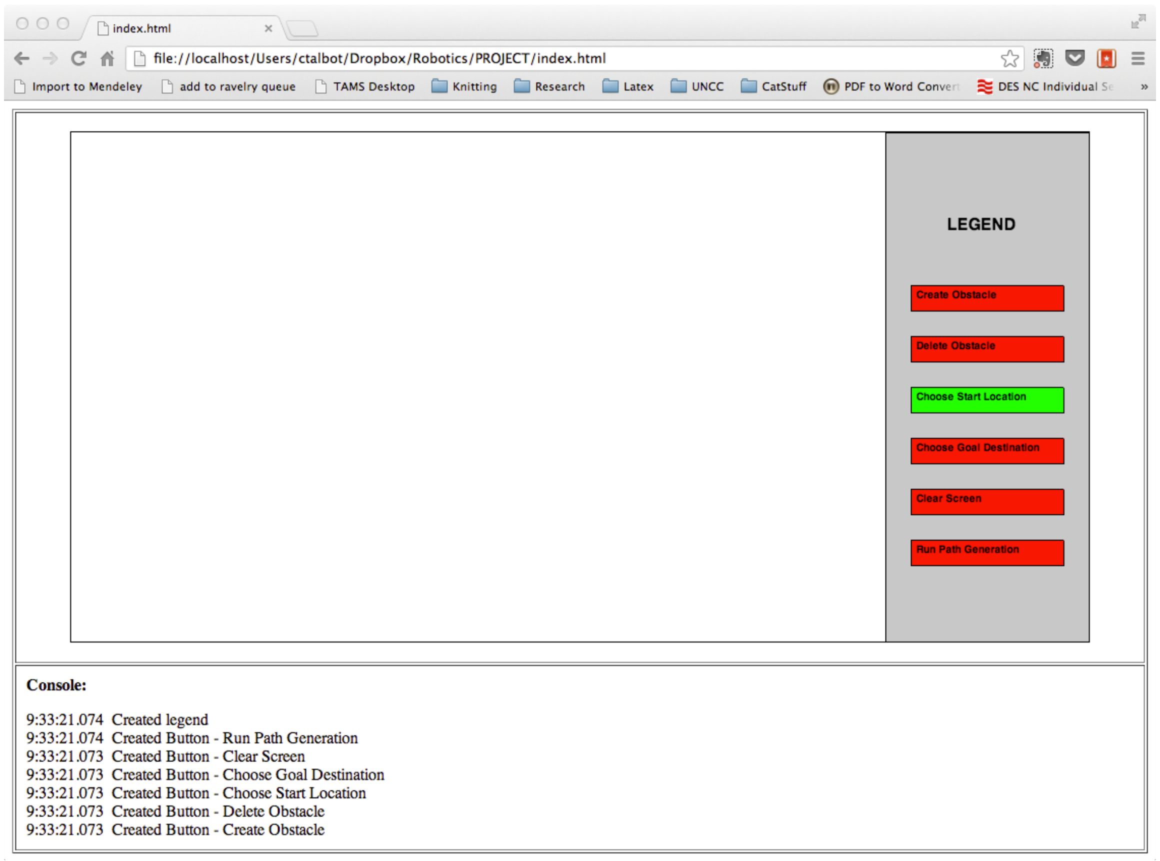
The application allows a user to create different environments, as well as different initial and goal configurations for the robot. Once a path is found, the program displays the path and emulates the robot motion following the path from the initial configuration to the goal configuration in a graphic display.

As for the optimization criteria of the path-planner, the shortest distance, smoothness, penetration distance, and clearance criterion is utilized. It uses off-line planning, and provides actions and status information for the user within the application.

Program Details

Starting the Application

The application is run by opening a web browser (such as Chrome) and navigating to the index.html file in the directory. Once opened, the user will see the following screen:



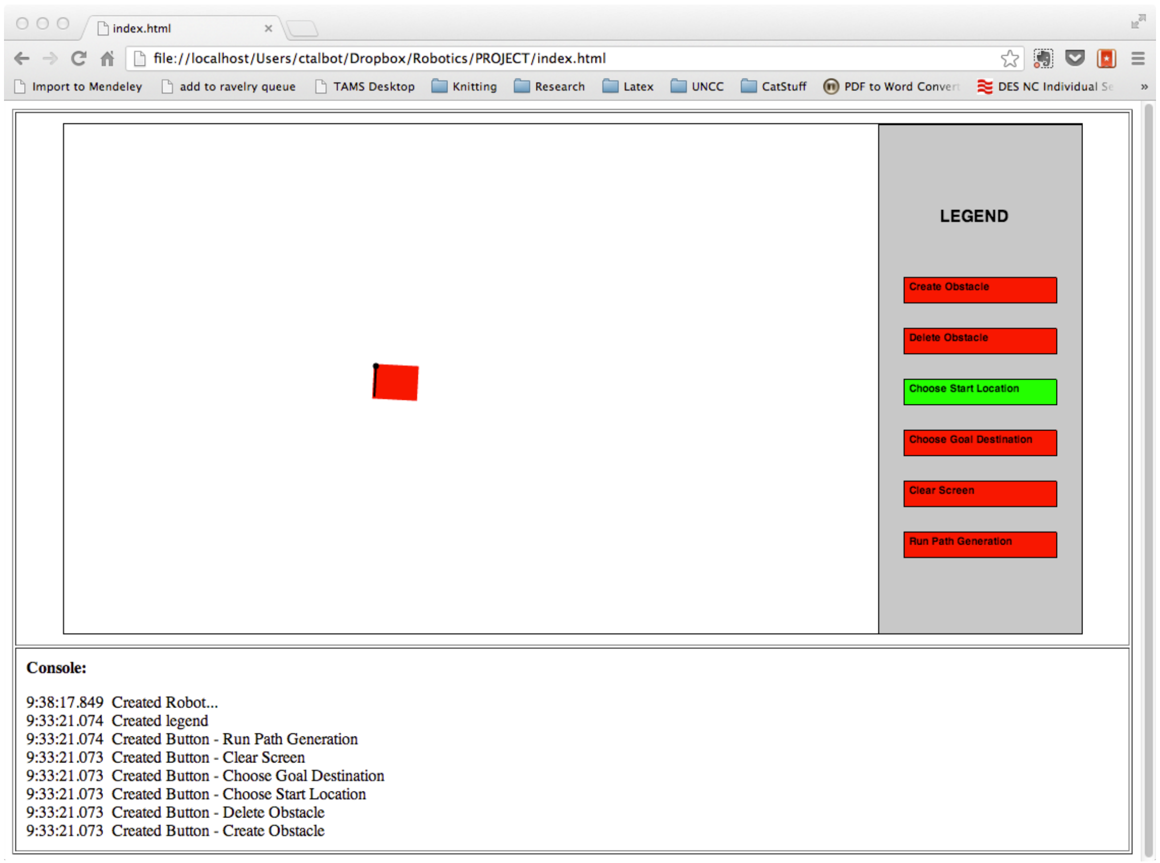
Console:

```
9:33:21.074  Created legend
9:33:21.074  Created Button - Run Path Generation
9:33:21.073  Created Button - Clear Screen
9:33:21.073  Created Button - Choose Goal Destination
9:33:21.073  Created Button - Choose Start Location
9:33:21.073  Created Button - Delete Obstacle
9:33:21.073  Created Button - Create Obstacle
```

The user will notice that additional information about what is happening will appear in the Console section of the screen. It will display up to the last 50 events performed by the application.

Choosing Robot Start Position

The user will notice that the “Choose Start Location” is selected by default, which indicates clicking on the screen will create the starting location of the robot (and draw the robot in its initial position). The clicked location is used as the starting point, and the system randomly chooses an orientation for the robot at that location. See below for an example of what the choice of the robot position looks like.



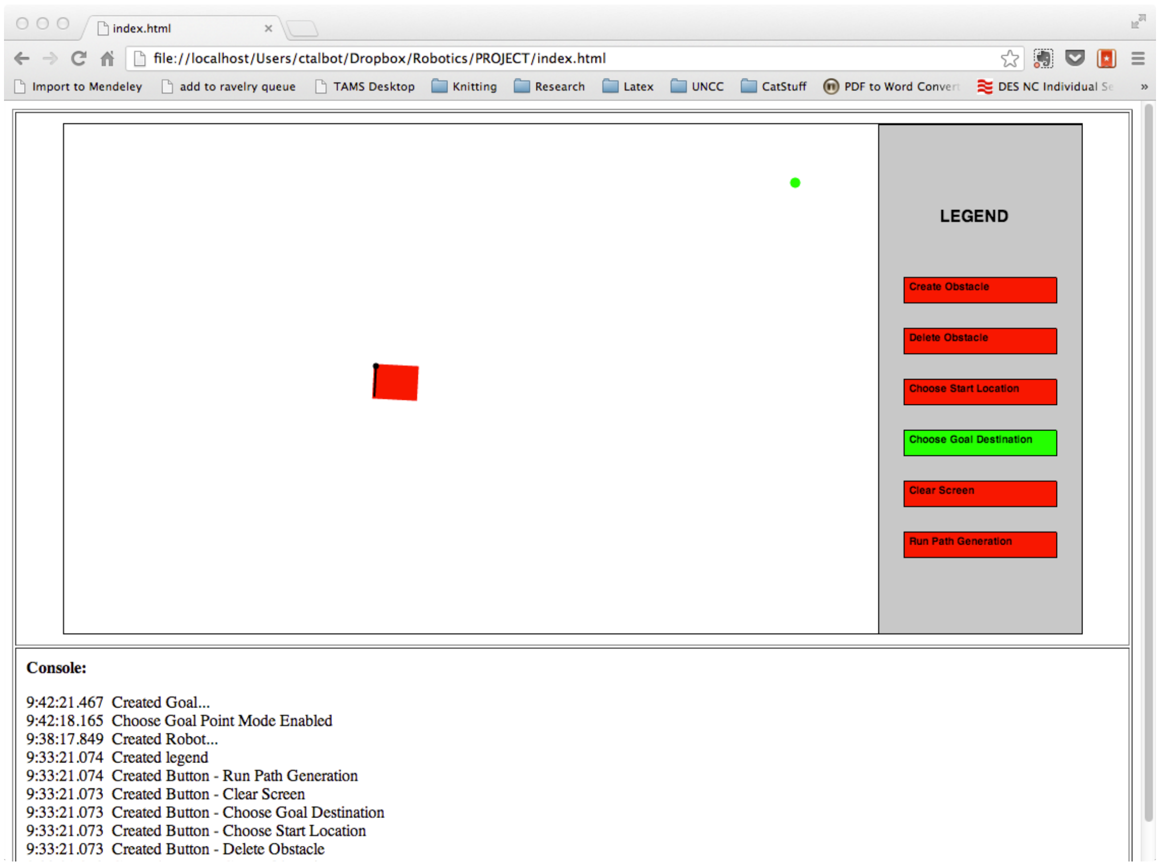
Console:

```
9:38:17.849  Created Robot...
9:33:21.074  Created legend
9:33:21.074  Created Button - Run Path Generation
9:33:21.073  Created Button - Clear Screen
9:33:21.073  Created Button - Choose Goal Destination
9:33:21.073  Created Button - Choose Start Location
9:33:21.073  Created Button - Delete Obstacle
9:33:21.073  Created Button - Create Obstacle
```

The front of the robot has a black line, with the robot's reference frame origin displayed as a black circle. Clicking in additional spots will move the robot to a new location, with a new random orientation. Each click will result in an update to the Console's event log.

Choosing Goal Position

To choose a goal position, the user can click on the "Choose Goal Destination" button on the right. This will change that button to be green (highlighting the active state of the application). Just like choosing a starting position, clicking on the screen will allow the user to choose a destination for the robot to reach. The goal position will appear as a green circle at the clicked location, as seen below:



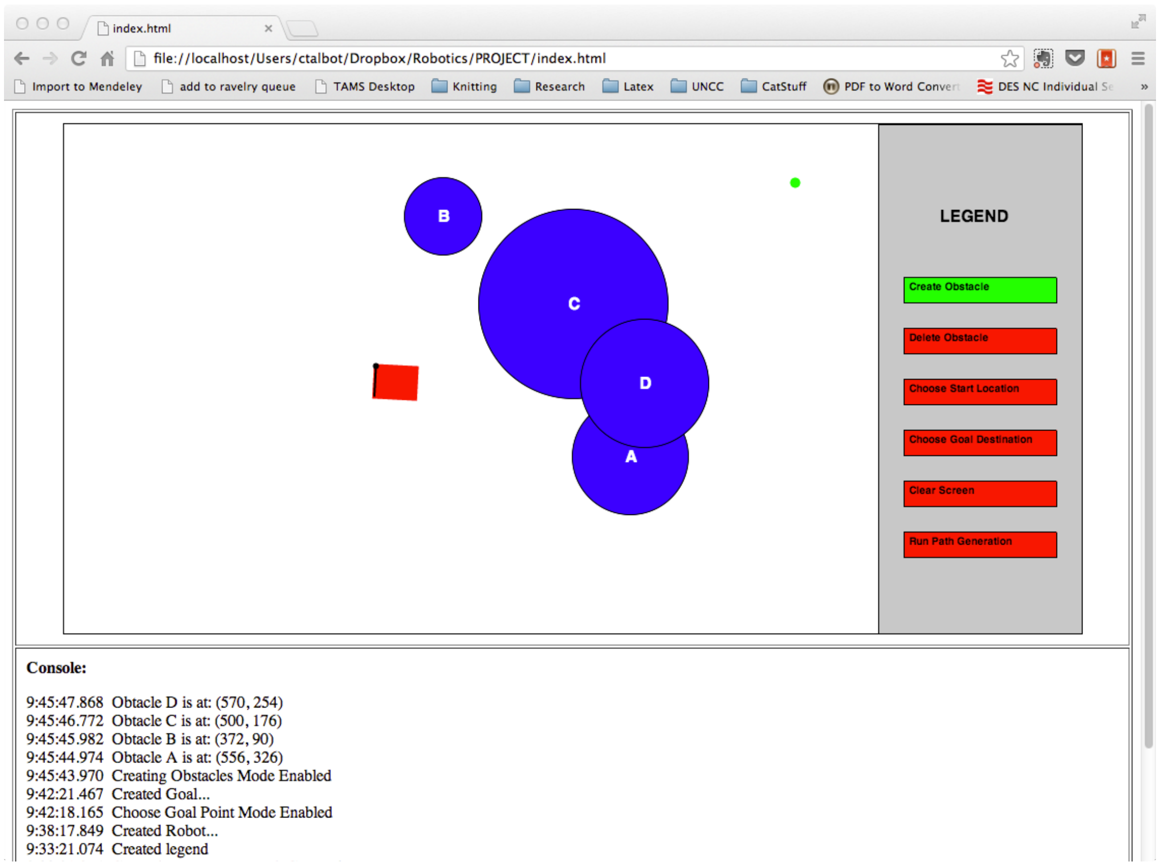
Console:

```
9:42:21.467  Created Goal...
9:42:18.165  Choose Goal Point Mode Enabled
9:38:17.849  Created Robot...
9:33:21.074  Created legend
9:33:21.074  Created Button - Run Path Generation
9:33:21.073  Created Button - Clear Screen
9:33:21.073  Created Button - Choose Goal Destination
9:33:21.073  Created Button - Choose Start Location
9:33:21.073  Created Button - Delete Obstacle
```

The goal's position can be moved as often as desired by the user, and the result of making these choices is reflected in the Console event log.

Creating Obstacles

To create obstacles in the application, the user should click on the button labeled "Create Obstacle". This will highlight the "Create Obstacle" button in green, indicating that clicks on the screen will generate new obstacles. Clicking within the environment will allow the user to choose the center of one of the circular obstacles. The system will randomly choose the size of the obstacle at each location and provide a label for them (A, B, C, …). Each click will result in a new obstacle, with obstacles allowed to overlap each other, as seen below:



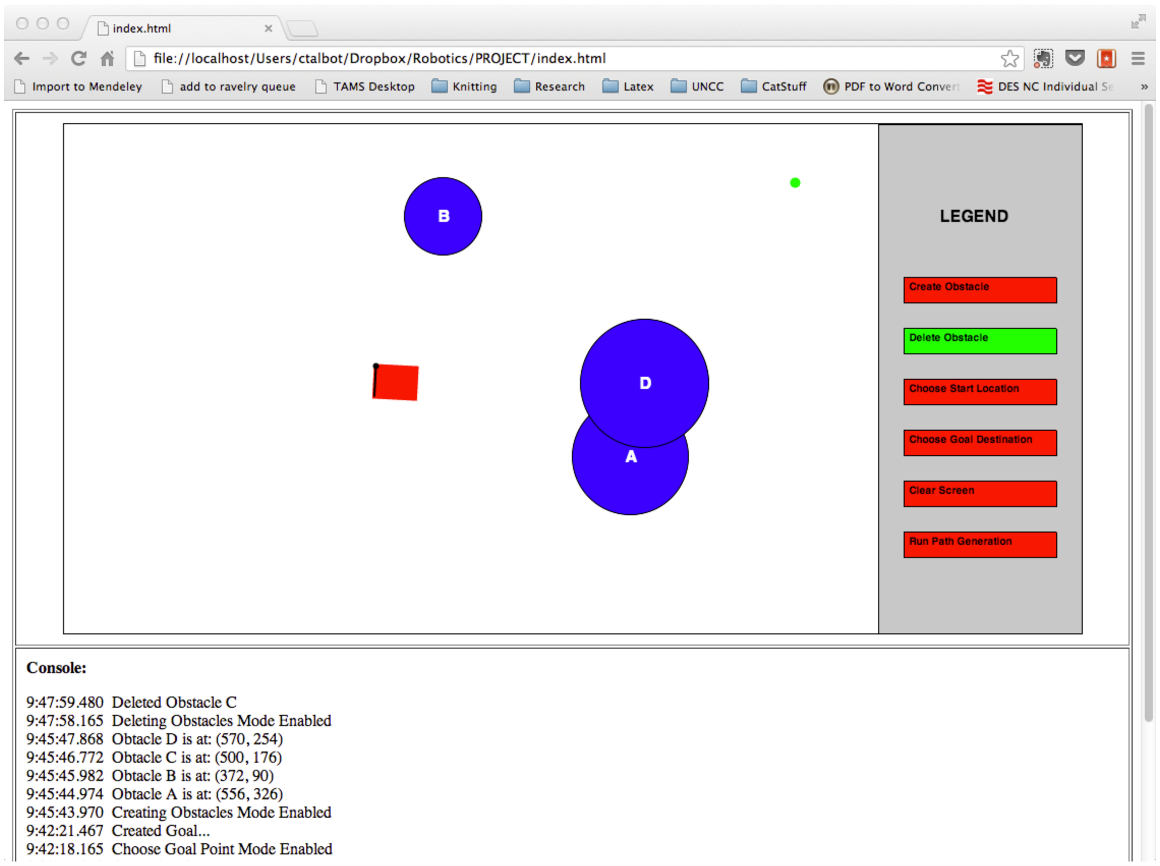
Console:

```
9:45:47.868  Obtacle D is at: (570, 254)
9:45:46.772  Obtacle C is at: (500, 176)
9:45:45.982  Obtacle B is at: (372, 90)
9:45:44.974  Obtacle A is at: (556, 326)
9:45:43.970  Creating Obstacles Mode Enabled
9:42:21.467  Created Goal...
9:42:18.165  Choose Goal Point Mode Enabled
9:38:17.849  Created Robot...
9:33:21.074  Created legend
```

The console will show the location of each obstacle's center and radius as the user creates them.

Deleting Obstacles

If the user does not like an obstacle, it can be deleted. By choosing the "Delete Obstacle" button, it is highlighted in green indicating that clicking on a blue obstacle will remove it. As seen below, clicking on the C obstacle seen in the previous picture will remove it from the screen:

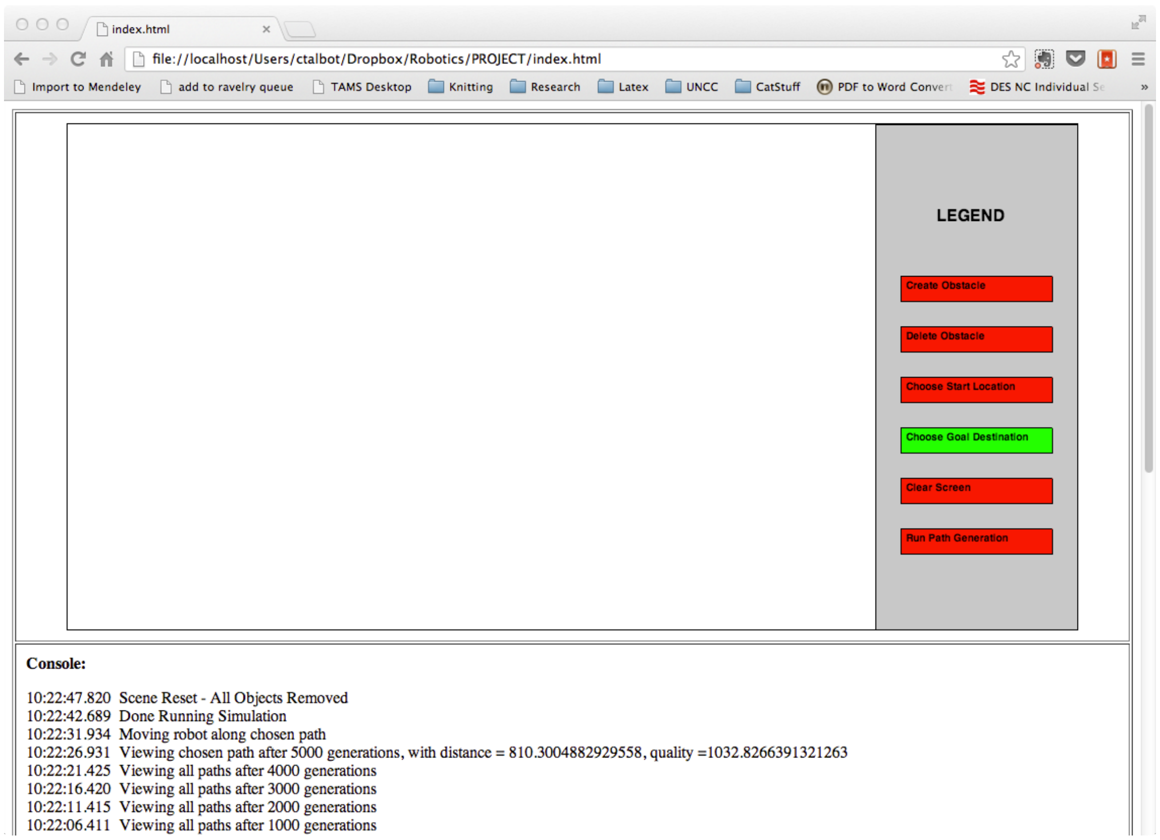


Console:

```
9:47:59.480  Deleted Obstacle C
9:47:58.165  Deleting Obstacles Mode Enabled
9:45:47.868  Obtacle D is at: (570, 254)
9:45:46.772  Obtacle C is at: (500, 176)
9:45:45.982  Obtacle B is at: (372, 90)
9:45:44.974  Obtacle A is at: (556, 326)
9:45:43.970  Creating Obstacles Mode Enabled
9:42:21.467  Created Goal...
9:42:18.165  Choose Goal Point Mode Enabled
```

Clearing Screen

To reset the application / start over, the user can click the "Clear Screen" button. Doing so will remove the robot, goal location, and all obstacles (if they exist) within the application. This allows the user to start fresh with a new environment, as seen below:



Console:

```
10:22:47.820  Scene Reset - All Objects Removed
10:22:42.689  Done Running Simulation
10:22:31.934  Moving robot along chosen path
10:22:26.931  Viewing chosen path after 5000 generations, with distance = 810.3004882929558, quality =1032.8266391321263
10:22:21.425  Viewing all paths after 4000 generations
10:22:16.420  Viewing all paths after 3000 generations
10:22:11.415  Viewing all paths after 2000 generations
10:22:06.411  Viewing all paths after 1000 generations
```

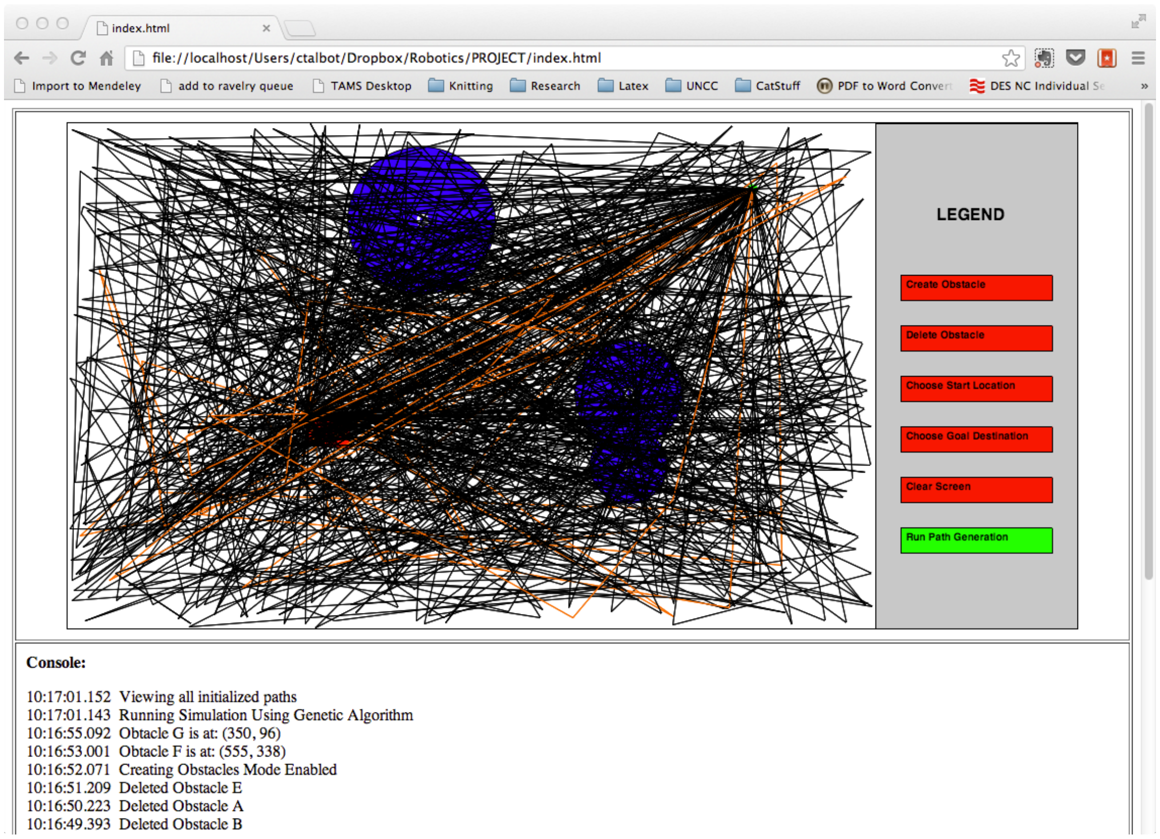
This will also leave the user in whichever mode they were last in.

Running Path Generation

To run the path generation EP/N algorithm, the user must have selected (at a minimum) a starting and goal location on the screen. Any number of obstacles can be created, including no obstacles at all. By clicking on the "Run Path Generation" button, if these conditions are met, the EP/N algorithm for generating a path will begin.

Initialization

First, the system will initialize 100 random paths of random lengths of 3 to 17 nodes (including the start and goal points). These will be displayed on the screen as shown below:



Console:

```
10:17:01.152  Viewing all initialized paths
10:17:01.143  Running Simulation Using Genetic Algorithm
10:16:55.092  Obtacle G is at: (350, 96)
10:16:53.001  Obtacle F is at: (555, 338)
10:16:52.071  Creating Obstacles Mode Enabled
10:16:51.209  Deleted Obstacle E
10:16:50.223  Deleted Obstacle A
10:16:49.393  Deleted Obstacle B
```

Any viable paths (not crossing any obstacles) will be shown in orange, while non-viable paths will be shown in black.

Each Set of Generations

Next, the application will perform the first set of generation modifications to these paths using the 4 modifications: add a node, delete a node, move a node, crossover of paths, and swap paths.

Modifications

The system will randomly choose up to 50 paths (or pairs of paths) to modify. After generating these new paths, the worst (up to 50) paths are removed from further use. The system will perform 4 rounds of 50 generations of paths.

Add a Node

This modification will take an existing path which has < 17 nodes (including start/finish) already, and add a new random node between the start and goal nodes.

Delete a Node

This modification will take an existing path which has > 3 nodes (including start/finish) already, and randomly remove one node somewhere between the start and goal nodes.

Move a Node

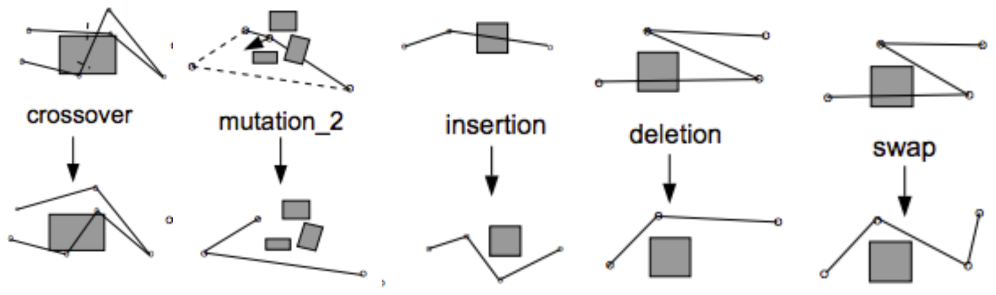
This modification will take an existing path, and randomly change one of the nodes between the start and goal nodes to a new location.

Swap Paths

This modification will take two existing paths, and will choose a random split location within the length of the shortest/fewest nodes path. The resultant path will have the first part of its route defined by path #1 up to this random split point, then will have the second part of its route defined by path #2's route from the split point to the goal node. The resultant path will be the same length as path 2.

Crossover

This modification will take two existing paths, and will choose a random split location within the length of the shortest/fewest nodes path. The resultant path will be the same as path #1 except at this split point, where the path will take on path #2's position instead. The resultant path will be the same length as path 1.

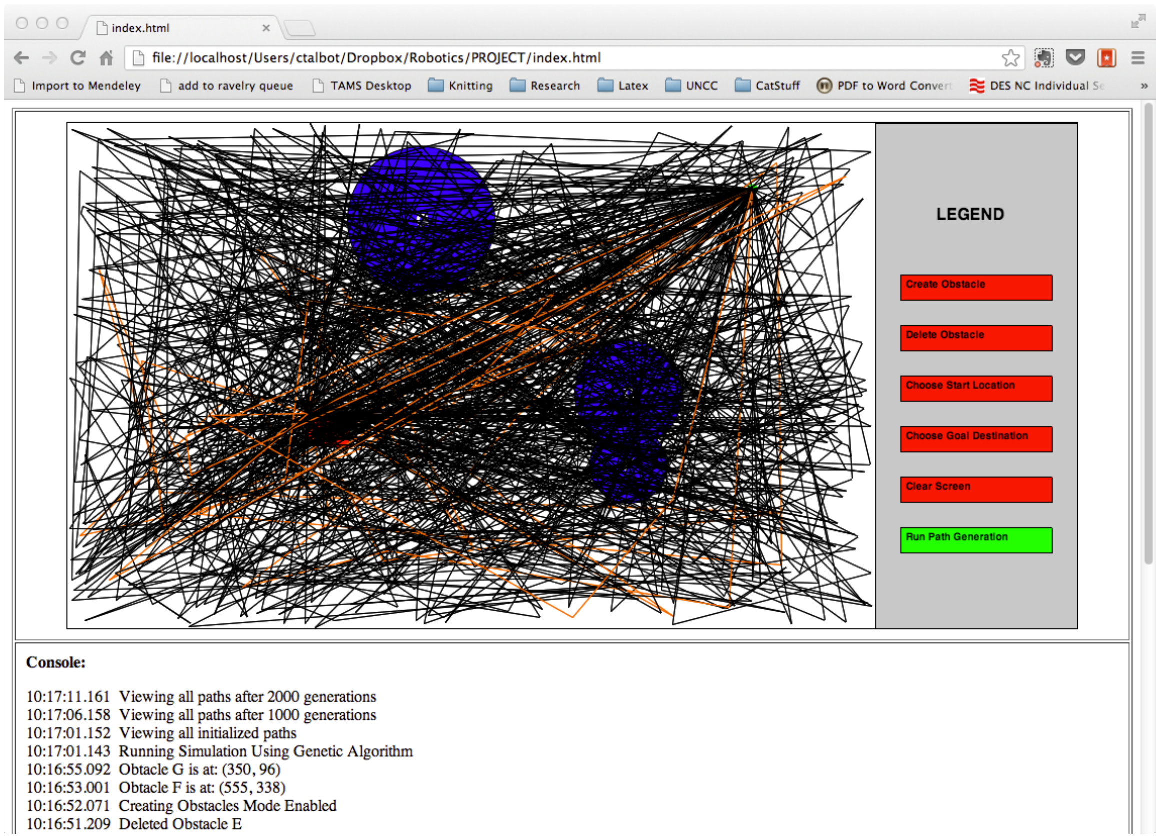


Comparisons

Each viable path is guaranteed to be considered "better" then any non-viable path. Within the viable paths, the path with the combination of shortest length, greatest smoothness, and greatest clearance are considered the best. Within the non-viable paths, the paths with the combination of shortest length and minimal penetration depth are considered the best.

Display

For each 50 generations, the system will display the current state of all the best paths so far. Each viable path will appear in orange, with each non-viable path appearing in black, as seen below:

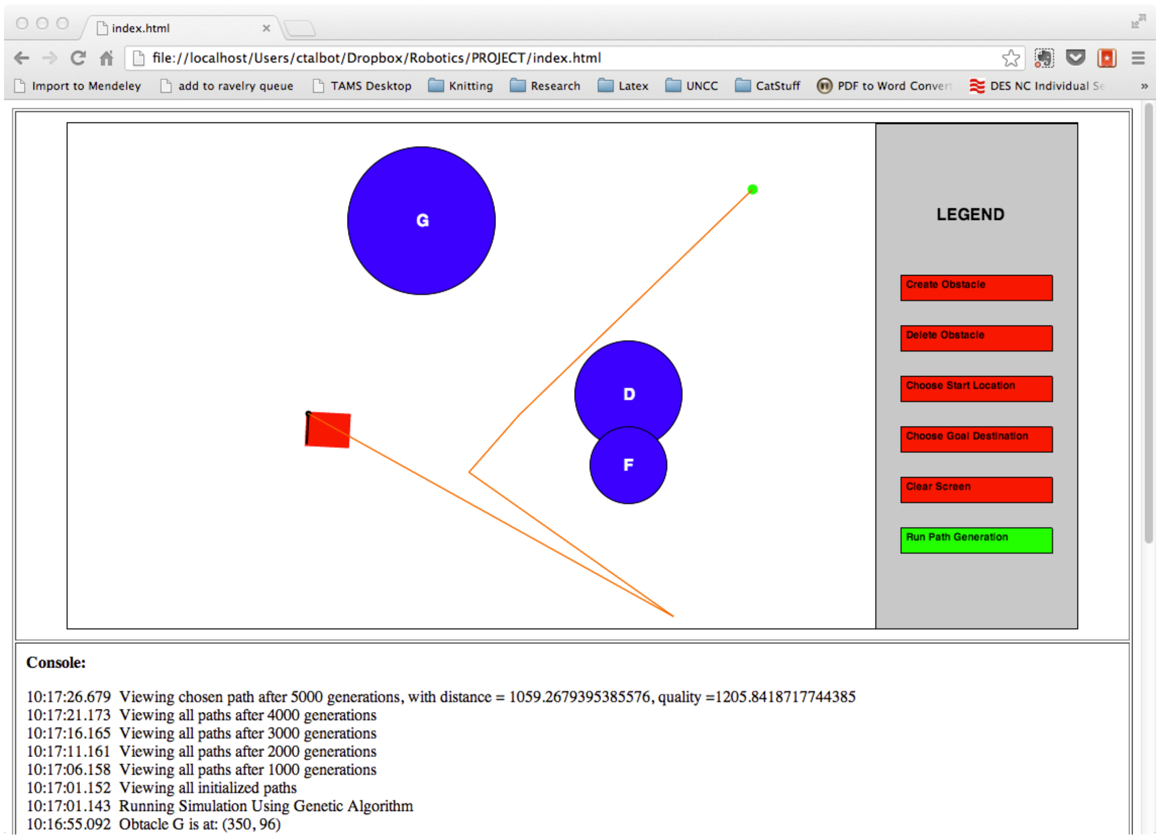


Console:

```
10:17:11.161  Viewing all paths after 2000 generations
10:17:06.158  Viewing all paths after 1000 generations
10:17:01.152  Viewing all initialized paths
10:17:01.143  Running Simulation Using Genetic Algorithm
10:16:55.092  Obtacle G is at: (350, 96)
10:16:53.001  Obtacle F is at: (555, 338)
10:16:52.071  Creating Obstacles Mode Enabled
10:16:51.209  Deleted Obstacle E
```

Last Set of Generations

The application will perform one more set of 50 generation modifications using the same methods as before. It will then choose the best path from all the generations so far. This single path will be shown in the application, as seen below:

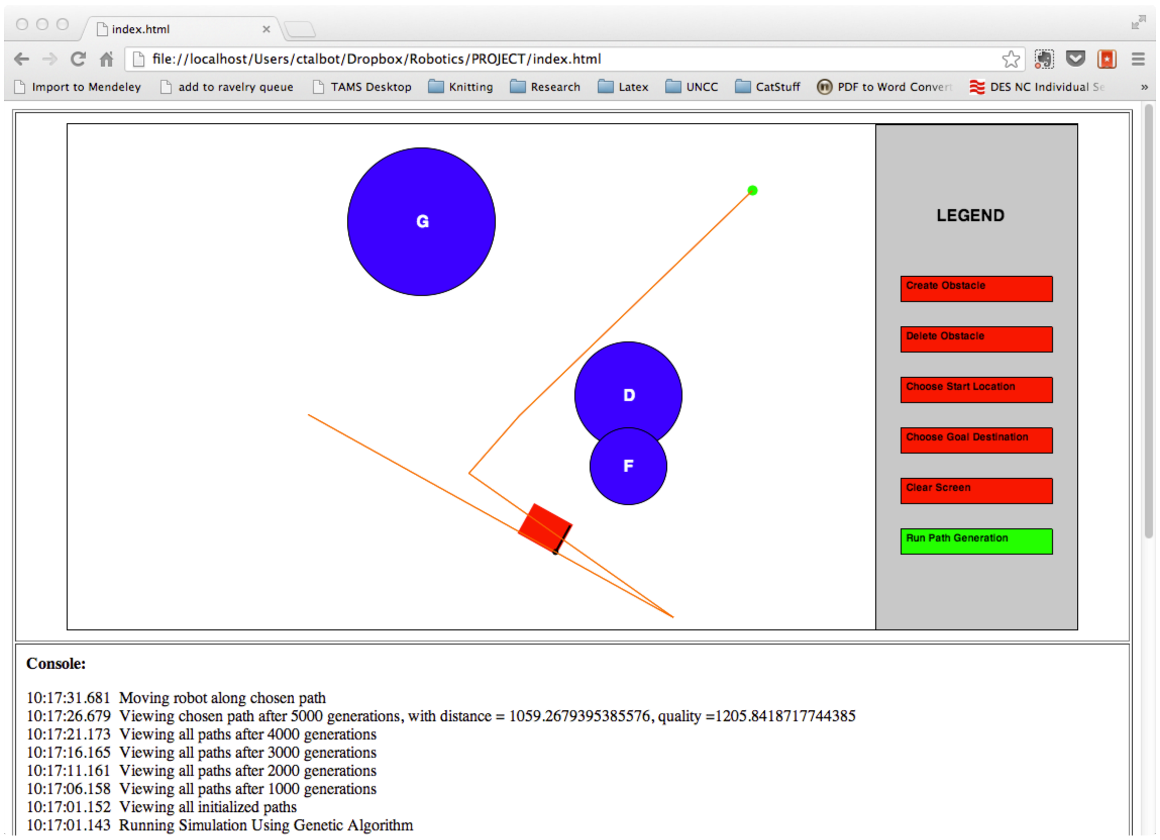


Console:

```
10:17:26.679  Viewing chosen path after 5000 generations, with distance = 1059.2679395385576, quality =1205.8418717744385
10:17:21.173  Viewing all paths after 4000 generations
10:17:16.165  Viewing all paths after 3000 generations
10:17:11.161  Viewing all paths after 2000 generations
10:17:06.158  Viewing all paths after 1000 generations
10:17:01.152  Viewing all initialized paths
10:17:01.143  Running Simulation Using Genetic Algorithm
10:16:55.092  Obtacle G is at: (350, 96)
```

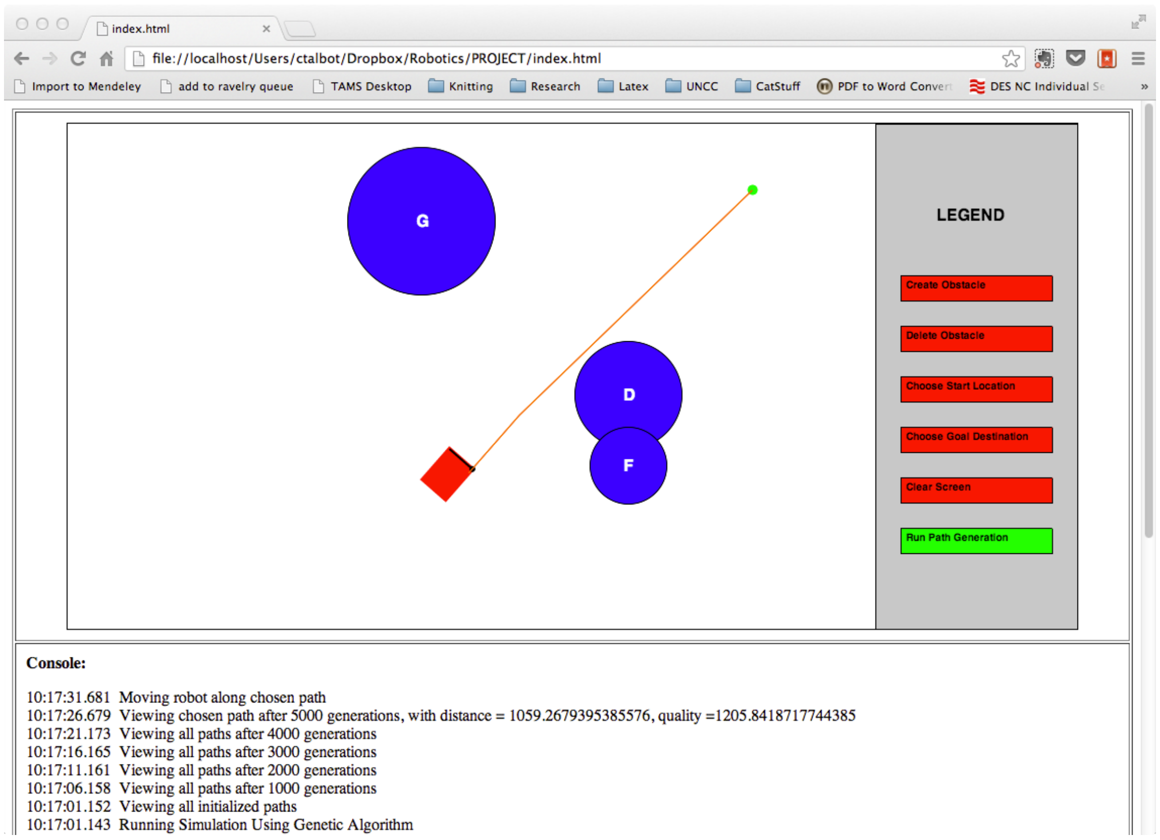
If this path is viable, it will be orange and the robot will begin moving along the path. If this path is NOT viable, it will be black and the console will display a message stating that it could not find a path in the environment, and the application will reset itself.

As the robot finishes each leg of the chosen path, that segment/leg will be removed from the screen. When the robot reaches the goal position, no paths will appear. Samples of this can be seen below:



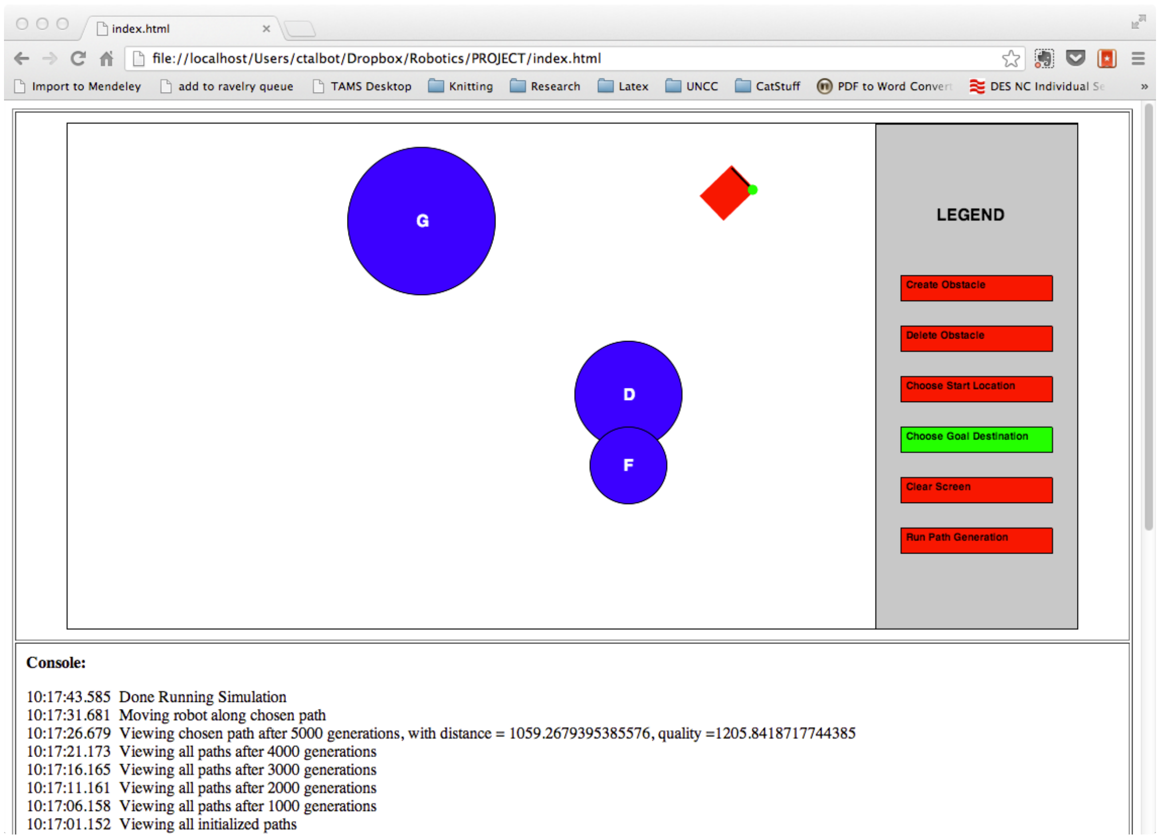
Console:

```
10:17:31.681  Moving robot along chosen path
10:17:26.679  Viewing chosen path after 5000 generations, with distance = 1059.2679395385576, quality =1205.8418717744385
10:17:21.173  Viewing all paths after 4000 generations
10:17:16.165  Viewing all paths after 3000 generations
10:17:11.161  Viewing all paths after 2000 generations
10:17:06.158  Viewing all paths after 1000 generations
10:17:01.152  Viewing all initialized paths
10:17:01.143  Running Simulation Using Genetic Algorithm
```



Console:

```
10:17:31.681  Moving robot along chosen path
10:17:26.679  Viewing chosen path after 5000 generations, with distance = 1059.2679395385576, quality =1205.8418717744385
10:17:21.173  Viewing all paths after 4000 generations
10:17:16.165  Viewing all paths after 3000 generations
10:17:11.161  Viewing all paths after 2000 generations
10:17:06.158  Viewing all paths after 1000 generations
10:17:01.152  Viewing all initialized paths
10:17:01.143  Running Simulation Using Genetic Algorithm
```



Console:

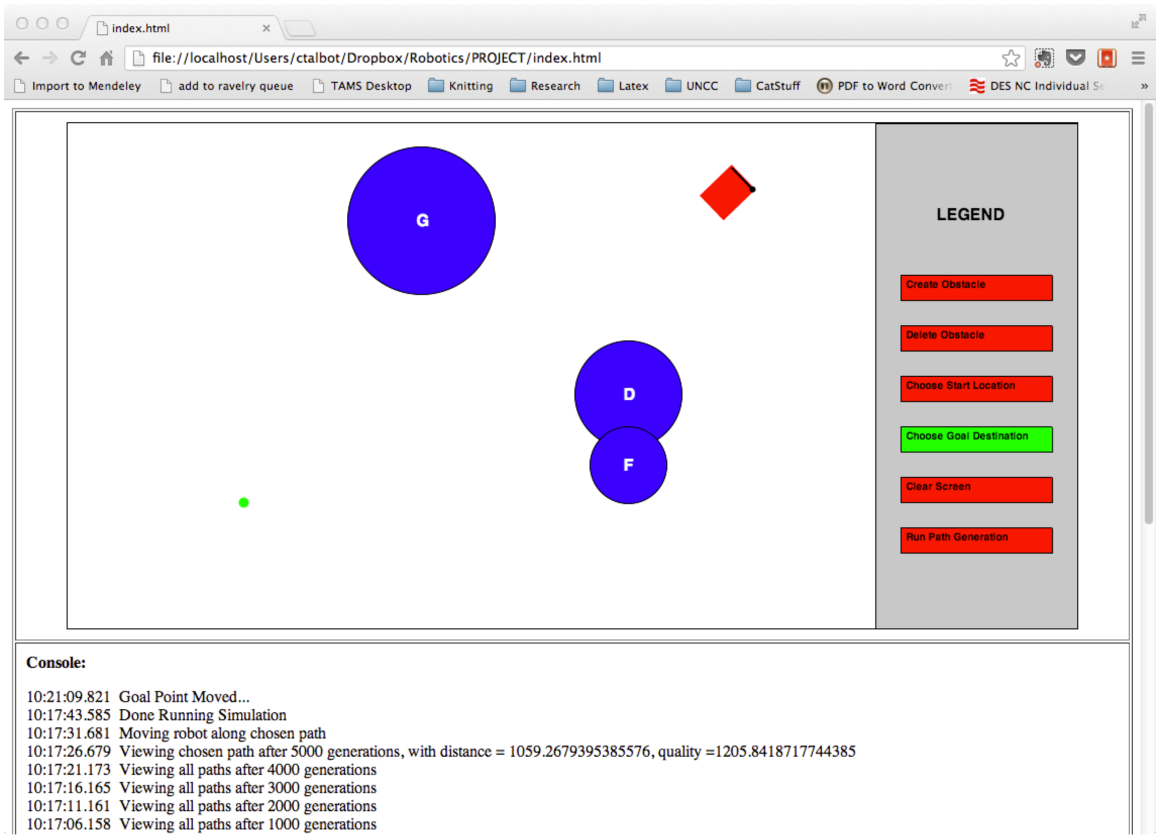
```
10:17:43.585  Done Running Simulation
10:17:31.681  Moving robot along chosen path
10:17:26.679  Viewing chosen path after 5000 generations, with distance = 1059.2679395385576, quality =1205.8418717744385
10:17:21.173  Viewing all paths after 4000 generations
10:17:16.165  Viewing all paths after 3000 generations
10:17:11.161  Viewing all paths after 2000 generations
10:17:06.158  Viewing all paths after 1000 generations
10:17:01.152  Viewing all initialized paths
```

Finishing Run

When the robot is finished moving (or when no path could be found), the system returns to the “Choose Goal Destination” to allow the user to choose a new goal within the same environment and run again. Any other options are still available to the user for clearing, changing, or re-running the application.

Re-Running Program

Because of the default setup, which puts the user in the “Choose Goal Destination” mode, the user can just click on a new location for the goal and re-click the “Run Path Generation” button again. Any other combinations of steps discussed above can also be performed prior to re-running. An example is shown below:



Console:

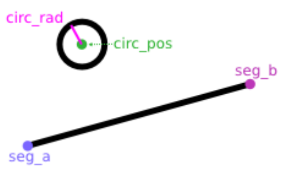
```
10:21:09.821  Goal Point Moved...
10:17:43.585  Done Running Simulation
10:17:31.681  Moving robot along chosen path
10:17:26.679  Viewing chosen path after 5000 generations, with distance = 1059.2679395385576, quality =1205.8418717744385
10:17:21.173  Viewing all paths after 4000 generations
10:17:16.165  Viewing all paths after 3000 generations
10:17:11.161  Viewing all paths after 2000 generations
10:17:06.158  Viewing all paths after 1000 generations
```

Analysis & Discussion

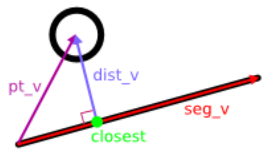
Some key attributes about the genetic algorithm EP/N are interesting to note. For instance, the calculations of the collision detections, penetration, clearance, and smoothness are key in the algorithm's success.

Collision Detection

To detect a collision between the path segment and the circular obstacles, the key was to find the closest point on the line relative to the center of the circle. To start with, we need to define a few points: circ_pos as the center (x, y) coordinate of the obstacle; circ_rad as the radius of the obstacle's circle; seg_a as the starting point (x, y) of the path segment being checked; and seg_b as the ending point (x, y) of the path segment being checked.



Next, we want to calculate the two vectors seg_v (the line segment being checked) and pt_v, the vector from the line segment's starting point to the obstacle's center point.

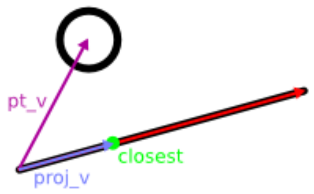


This is done by:

seg_v = seg_b – seg_a;

pt_v = circ_pos – seg_a;

Now, we want to project the vector pt_v onto our line segment to find the closest point on the line segment to our obstacle. We'll call it "closest".



This is done by taking the dot product of the vector pt_v and the unit vector of the projection target (proj_v), or:

|proj_v| = pt_v • (seg_v / |seg_v|)

This only gives us the length of proj_v, not the vector itself. Also, this point could lie outside the line segment's endpoints, so we need to check |proj_v|'s size. If it is smaller than 0, then it is before seg_a on the line, so seg_a will be the closest point to the obstacle's center. If it is larger than the length of seg_v, then it is after seg_b on the line, so seg_b will be the closest point to the obstacle's center. If neither of these is true, then proj_v's point is the closest point on the line to our obstacle's center.

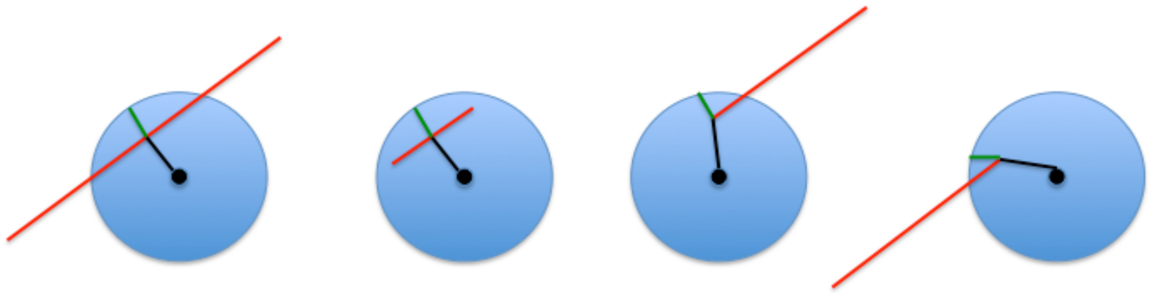
Now we just need to get the actual coordinates of the proj_v (if it is the closest point), which can be done by:

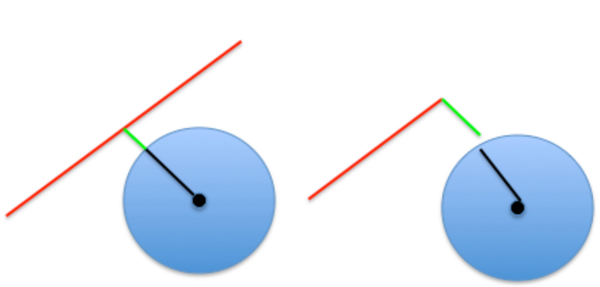
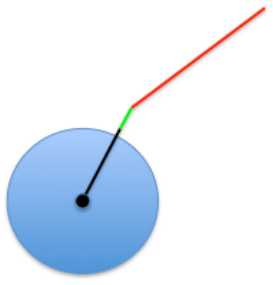
proj_v = |proj_v| * (seg_v / |seg_v|);

```
closest = seg_a + proj_v;
```

Once we have this, it is pretty simple to determine if our line intersects the obstacle by comparing the distance from our circle center and our newly calculated closest point. If this distance is less than the radius of the obstacle, then we have an intersection. If not, there is no intersection.

Once we know this information, we can then calculate our penetration depth and clearance for this line. If the line is in collision with the obstacle, then the radius of our obstacle minus the distance from the closest point to the obstacle's center is our penetration depth. If the line is NOT in collision, then our clearance is the distance from the closest point to the obstacle's center minus the obstacle's radius.


Examples of Collision detection: Green line is the penetration used for each scenario



Examples of Clearance Detection: Green line is the clearance used for each scenario

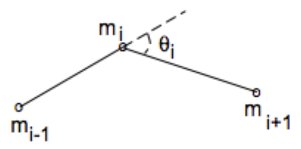
Along these same lines, each turn in the path (including the initial turn) is analyzed for potential collisions during the turn. The location of the diagonal position from the robot's origin is calculated for either the robot's current position (for the first segment of the path), or the prior segment's orientation, as well as for the required orientation to travel the next segment.

The line segment between these two points are used as another segment to compare to the obstacles, as described above. If there is an intersection with that line, the path is marked infeasible. However, if there isn't an intersection, but the closest point on that segment to the obstacle is not one of the end points, then we need to check if the distance from the obstacle's center and the path's turning point is less than or equal to the sum of the obstacle's radius and the robot's diagonal length.

This second comparison ensures the arc that the robot's diagonal (longest radius) doesn't intersect with the obstacle.

Smoothness Calculations

To calculate the smoothness of the line, we need to determine the angle between two consecutive segments of the path (i.e.: mi-1 to mi and mi to mi+1), θi. This is accomplished by utilizing the dot product of the two vectors to obtain the angle. Minimizing the size of this angle equates to a smoother path.



Future Improvements

Future improvements could include any of the following:

1. Validation of robot and goal position feasibility with respect to obstacles prior to setting their positions
2. Expansion to a 2R or 3R robot
3. Expansion to a non-holonomic robot
4. Obstacles of different shapes
5. Ability to modify the size of the obstacles manually
6. Ability to set the robot orientation manually
7. Expansion to online path planning