Differential Drive Robot Pathing

By Robert Sylvia

I. Abstract

This project will explore variations in robot path planning pertaining to differential drive robots. The nonholonomic constraints of differential drive robots create complications in navigating compared to the standard systems explored in class. Special focus will be placed on comparing different internode pathing methods to address the irregular motion constraints. Additionally, there will be consideration placed towards modifications to the occupancy grid and path traversal algorithms to attempt to improve performance.

II. Methods

This simulation testing for these experiments will be done in pybullet. Two randomly generated obstacle arrangements were used for testing. The obstacles grids for these can be seen in Figure 1. Upon creation, these obstacle grids and the obstacle information were saved to csv files. The obstacle information file was then used by pybullet to set up the simulation environment. The robot used for testing has a circular base, two velocity-controlled wheels, and a zero-friction ball at the front acting as a castor wheel. Both the robot model and tone of the simulation environments are shown in Figure 2.

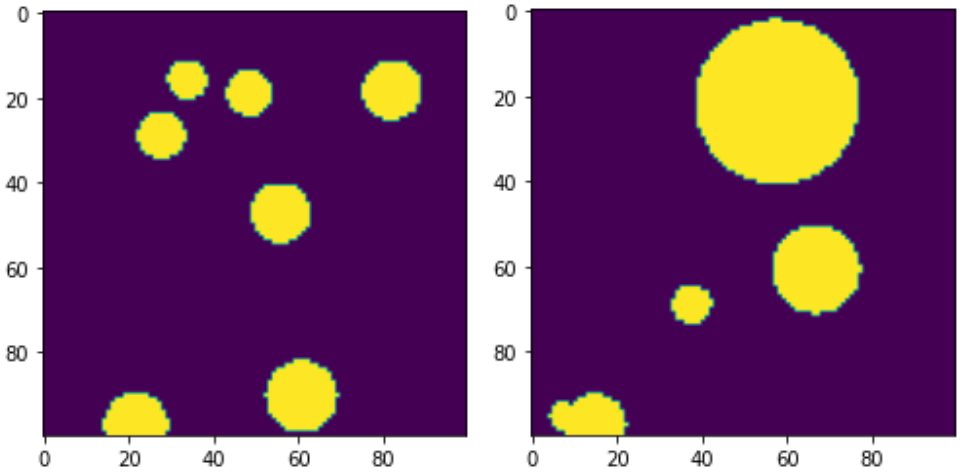

Figure 1: Occupancy grids for the two obstacles arrangements used for testing.

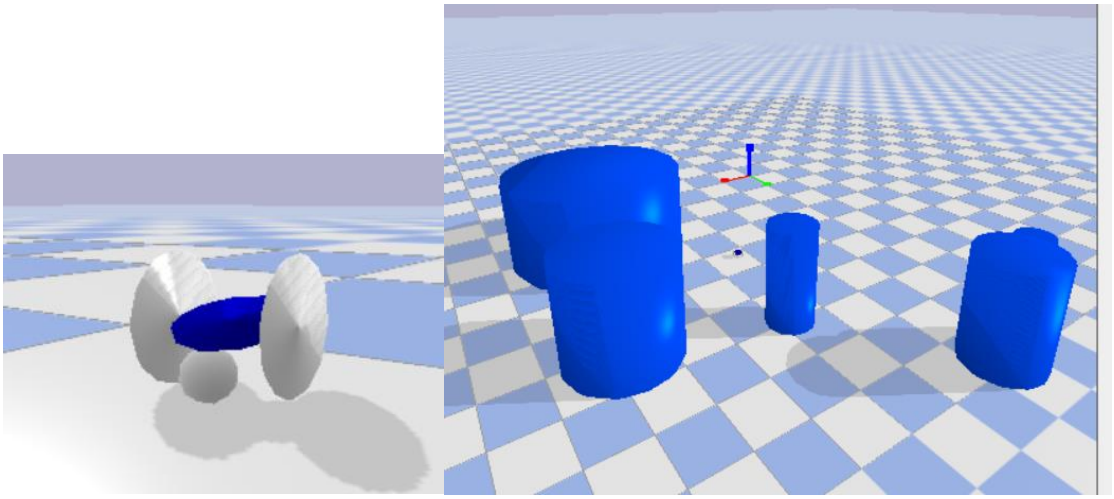

Figure 2: The rover used for testing (left) and a pybullet simulation environment (right).

The basis for the pathing algorithms used was an RRG and A* search. RRG was chosen because this study wanted to consider how to address pathing errors and assumed the system would want to repeatedly attempt navigation over the same space making it superior to tree based pathing methods which would have to be remade for every navigation. The A* search algorithm seemed a natural choice as the graph object has a list of neighbors and their costs built into it. For A* search, the L2 distance between the positional components of the node and goal poses was used for the heuristic to estimate cost. Each node of this graph represents a randomly sampled SE(2) pose sampled from the unoccupied space and each edge has a "cost" or "weight" as well as a series of wheel speeds and times used to traverse between the connected nodes. These edges are directed because the wheel commands and potentially the cost will be different depending on the direction of traversal. Edges are not included in the graph if the path outcome either is too far from the target pose or collides with an obstacle. The cost of traversal is determined by the estimated time required to traverse the edge is shown below in Equation 1. T is the total time taken, the error is an evaluation of the error between the target node pose and predicted end pose, dir_mult is a term that penalizes reverse movement, the ω terms are the right and left wheel velocities, and occ_grid represents the value of occupancy grid at position (x,y), which would be index (y,x) due to the difference in conventions between spatial grids and matrix indexing.

Equation 1: Edge Cost calculation

$$cost = dir\_mult * (error + T + 10 * \int_0^T occ\_grid(x,y) \ * 0.5(abs(\omega_l) + abs(\omega_r))dt)$$

Four variations of the internode pathing algorithms that demonstrated some success were considered using the names "spin and move", "basic curve", "iterated curve", and "estimated curve". Among these, "spin and move" is considered the baseline as it is the most basic and its behavior the most reflective of traditional straight line pathing methods. It uses equal in magnitude, but opposite direction wheel velocities to turn a robot in place towards the position of its target pose, moves straight forward, then does the same rotation as before to spin into the target pose's orientation. However, while simple and effective, the need to rotate and move independently implies some measure of inherent waste. To address this, the other methods attempt to make use of Lie Algebra to evaluate wheel speeds to traverse curves. The Lie Algebra refers to the tangent space of a differentiable manifold, called a Lie

group, and traversal of the Lie group with a particular Lie Algebra can be easily found using Equation 2. This can be a little confusing since the term “Lie Algebra” refers to the set of all entries that make up the tangent space, but often is also used to refer to a specific entry in that set corresponding to a particular motion. The actual Lie Algebra of the robot is determined by the robot parameters and wheel speeds according to the calculations shown in Table 1. Using some algebra, it is possible to obtain Equation 3 which shows the calculation to obtain the lie algebra to go from the robot pose X and the target pose Y. The t in the equation is simply the time of the motion, but the $\dot{\Omega}$ is more complicated and must take the form of a 3x3 matrix with the entries shown in Table 1. The green entries would be used to evaluate the necessary wheel velocities. However, the concern is the red entry as the first and second rows of the third column represent the derivatives of the x and y position of the robot respectively, and the red entry being zero is a result of the robot’s holonomic movement constraint that prevents motion perpendicular to its wheel direction. This constraint is not guaranteed by the calculation of Equation 3. The plots in Figure 3 demonstrate how violating the constraint on the red entry in the Lie algebra causes divergence between desired path and the actual path using extracted wheel speeds. The left image depicts how trying to use a Lie Algebra that significantly violates constraints to calculate wheel speeds produces a significantly different motion than expected compared to the right image which only had a minor violation and thus the divergence is within a tolerable margin of error. In brief, the constraints of the robot limits the set of valid Lie Algebra entries such that only some pose transformations are possible assuming induvial wheel speeds are constant throughout the motion. What differentiates the other three internode pathing algorithms from each other is how each one addresses this issue.

Equation 2: Calculation to determine pose transformation, or traversal of SE(2) space, from Lie Algebra.

$$Y = X * \exp(\dot{\Omega} * t)$$

Equation 3: Lie Algebra Calculation

$$\dot{\Omega} * t = \log(X^{-1} * Y)$$

0	$r/w*(\omega_l - \omega_r)$	$r/2*(\omega_l + \omega_r)$
$r/w*(\omega_l - \omega_r)$	0	0
0	0	0

Table 1 : Form of the 3x3 matrix $\dot{\Omega}$ in Equation 3 for a differential drive robot with r being wheel radius, w being robot diameter and the ω values being wheel velocities. A non-zero value at any of the entries with a 0 in the above matrix implies a Lie Algebra that violates the constraints of the robot. Of particular concern to the algorithm, a non-zero entry at the position of the red zero implies that the Lie Algebra corresponds to a motion that violates the holonomic constraints of the robot.

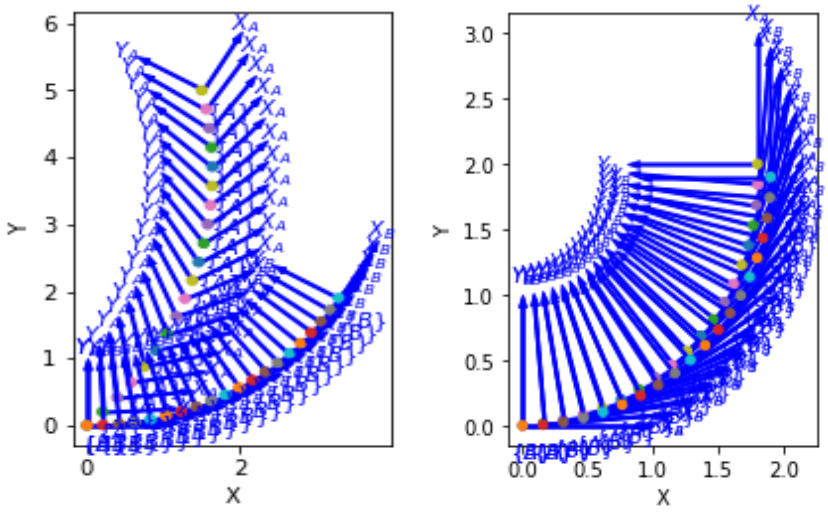

Figure 3:Divergence between paths of calculated Lie algebra (Frame A) and path traversed by robot with wheel speeds obtained from Lie Algebra (Frame B). Left image is a Lie algebra with non insignificant constraint violation. Right image has very little constraint violation in Lie Algebra.

The “basic curve” method took the simple brute force solution to the issue. It simply discarded the connections between nodes with Lie Algebras that violated the constraint. However, this severely limits which nodes connect to each other and the RRG will need many more nodes to produce a fully interconnected graph.

The “iterated curve” method modifies the path algorithm sightly to reduce the restriction on which nodes connect. Like the “basic curve” it will accept any connections with little Lie Algebra constraint violations as an edge, but in the case there is a constraint violation, it will modify its wheel speeds extracted from the lie algebra so as to rotate the robot more towards an orientation more aligned with the total velocity obtained from the Lie Algebra’s third column. It then steps through a fraction of the time obtained from the original wheel speed calculation, before repeating the process from its new position. It will then either converge to an acceptable result or fail and be discarded. A successful expel of this approach can be seen in Figure 4.This produces more connections but costs much more computationally.

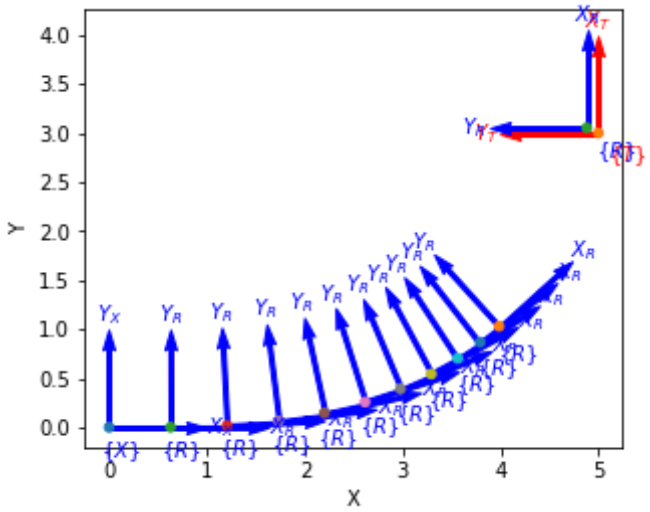

Figure 4:A successful example of the iterated curve algorithm. The blue frames represent the pose of the robot and the red frame represents the target pose.

The final "estimated curve" is extrapolated from the base assumption that a path can be found between any two poses by traversing along a series of circular paths that intersect tangentially. This was further simplified to be limited to connecting poses with a single circular path and a straight line, which is technically a circular path with infinite radius, as shown in the left image of Figure 5. An intermediary point is found geometrically and then each curve is solved independently using the Lie Algebra calculation. For more complicated pose transformations, the signs of the calculation require more thought and the path is not very optimal regardless. Incidentally, these complicated transformations are easy to identify because it is when the change in horizontal coordinates has a sign opposite that of the orientation change. Therefore, it was better to find an intermediary point calculated by finding the segment connecting a point directly in front of the initial pose and one directly behind the target pose. The intermediary pose would be located at the midpoint of this segment and be directed along it. This would allow the path to use a series of the two more simplified curve to produce a more complicated one as shown in the right image of Figure 5.

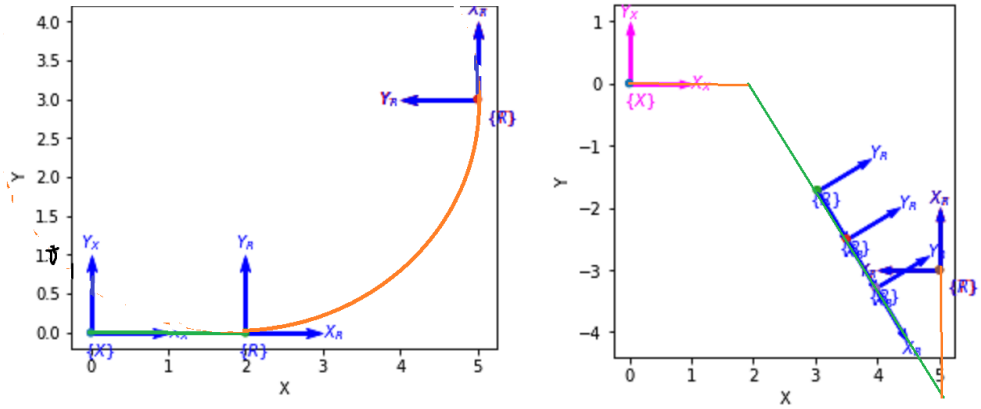

Figure 5: Estimated curve paths. Left image shows the simple curve estimation. The right image demonstrates how a intermediary point is found to create a more complicated curve using a series of two simple ones.

Additionally, two methods of addressing pathing errors were considered. The first being two alterations to the occupancy grid, expanding the buffer zone around obstacles to be larger than the robot radius and applying a gaussian blurring filter to be used by the path cost calculation. This was more of a passive change that would make the robot prefer "safer" paths around obstacles. The second was an attempt to course correct whenever the robot found its poses had deviated too far from the node pose it was supposed be located at. To further expand this, a particle filter was created to see how the correction algorithm worked with uncertain measurements when using the particle filter compared to just assuming each measurement was accurate, but there were some complications with this.

III. Experimentation

For the sake of testing these algorithms a series of trials were conducted. 5 Trials of each type were done on each occupancy grid then the mean was calculated for the final values. Each set had to test each one of the four pathing algorithms and each of these algorithms had to have tests using the course correction algorithm. However, all tests use the occupancy grid mask as many of the algorithms have collision issues as is. Graphs made with 100 and 200 nodes and an adjacent node search radius of 20 will each be evaluated on the time it takes to construct and its edge number, but the effective node number for the different methods is so distinct that testing some methods at lower node numbers would be pointless so not all of these will have performance evaluations. For testing the actual pathing itself, random start and goal poses will be sampled from the spaces, discarding trivial short distance paths between nodes. These will then be added to the graph and a path the A* algorithm will attempt to find a path between them. Each run will be evaluated based on position error, orientation error, simulation steps required, ability to find a path, and in the case of the course correction tests, how many times it had to correct itself. Ideally, each experiment type will have 10 runs, but some of the combinations perform so poorly that it will be difficult to obtain meaningful measurements from them. The time required by the A* algorithm will not be addressed as it is extremely fast compared to the time required by other part regardless of the graph that is used.

The data regarding the construction of the RRG's themselves can be found in Table 2. This includes information about a 400 node graph for the basic curve alone because the 100 and 200 node graphs are completely insufficient to even be compared to the other algorithms' graphs as 100 and 200 nodes.

	100 Node Time(s)	100 Node #Edges	200 Node Time(s)	200 Node #Edges	400 Node Time(s)	400 Node #Edges
Spin and Move	8.32	1140	32.89	4540	N/A	N/A
Estimate Curve	14.25	478	59.35	2328	N/A	N/A
Basic Curve	4.3	53	18.79	310	71	1008
Iterate Curve	11.11	240	42.67	846	N/A	N/A

Table 2:Average edge count and Creation times for different Pathing Methods

The data for the trials that did not use the course correction part of the code can be found in Table 3. This applies for the course correction trials as well, for the algorithms that have a high failure rate, not much weight can be placed in the other results for that algorithm as these types of algorithms can only succeed if the traversal is relatively simple and thus their outputs can sometimes be a poor representation of the algorithm's performance. Additionally, the 100 and 200 node graphs for the "basic curve" algorithm were not included in the tests due to not forming a connected graph at all.

No Course Correction	Mean Pos. Error(m)	Mean Orientation Error(radians)	Mean step count	Failure rate(%) (When no path is ever found)
Spin and move:100 node	24.7	1.06	22338.5	0
Spin and move:200 node	41.84	1.06	16435.3	0
Est. Curve: 100 node	38.02	1.33	13332.8	0
Est. Curve: 200 node	13.35	1.66	6895.8	10
Iter. Curve: 100 node	4.03*	0.006*	2761*	90
Iter. Curve: 200 node	19.17	1.05	100337.4	0
Basic Curve: 400 node	47.43	2.36	8662	80

Table 3:Table of results for trials without the course correction algorithm.

The information in Table 4 shows the trials that did use the course correcting algorithm. Something of note, much of the information concerning the step count is missing due to an unresolved quirk of the algorithm. The algorithm is supposed to run recursively until the error terms go below a certain threshold, but during instances when the algorithm keeps running without stopping, while the error terms show indicate it should be as far as 90% of the length of the map off target, the positions reported as the same time indicate the robot is in the vicinity of the goal. The path plot for one of the occasions when this happened is shown in Figure 6. When the program is forcefully stopped, then position of the robot is checked, it is typically right on top of the goal and well within the error boundaries. The code used to check the error after the pathing algorithm is forcefully stopped is of the same construction as the one within the algorithm that reports the error so it is unclear what is causing this. It may be an indexing issue or the simulation could be acting up while it is running for some reason. The latter case would explain why the particle filter is so temperamental as irregularities in the measurements beyond the normal distribution error would cause that to break down. Regardless, since the simulation has the robot arrive and stay at the target location, it will be considered a successful pathing trial and will use the error information found after stopping the algorithm, but since it is difficult to tell when it reached the target and since the variable keeping track of the step count is never returned if the program is forcefully canceled it is difficult to evaluate the step count accurately.

With Course Correction	Mean Pos. Error(m)	Mean Orientation Error(radians)	Mean step count*	Failure rate(%) (When no path is ever found)
Spin and move:100 node	1.53	0.464	N/A	20
Spin and move:200 node	3.76	0.888	N/A	20
Est. Curve: 100 node	3.405	0.185	N/A	0
Est. Curve: 200 node	4.157	0.054	N/A	20
Iter. Curve: 100 node	5.54	2.123	9011	70
Iter. Curve: 200 node	2.88	0.434	6638	0
Basic Curve: 400 node	21.96	2.41	N/A	50

Table 4:Data from trials that used course correction algorithm. An error in the algorithm caused the course correction to sometimes not end after reaching target pose so accurate simulation step counts were often not available.

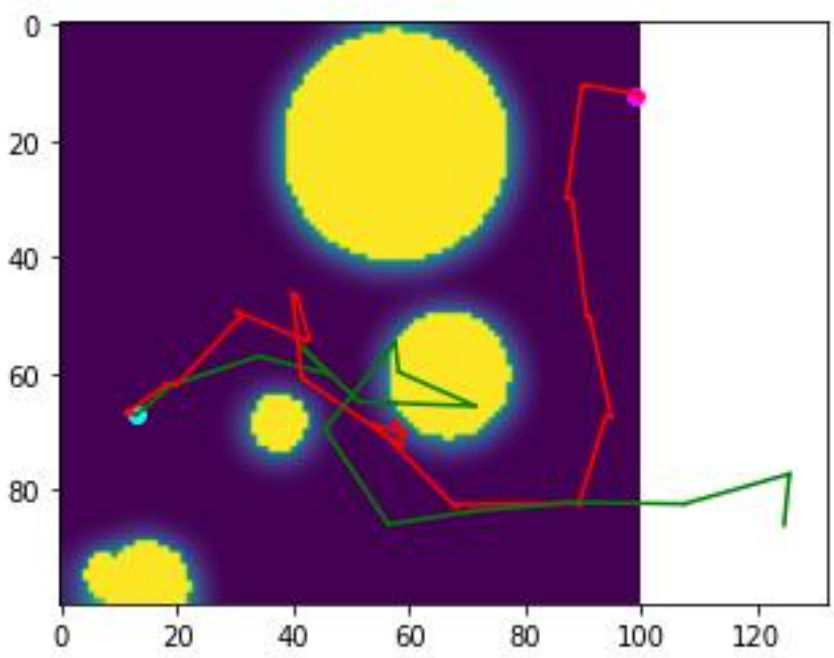

Figure 6: Shows plot of robot path occupancy grid when the course correction never stopped. The red line is the true path of the robot reported by the simulator, while the green is the calculated path based purely on the wheel commands. The start is the cyan dot and the target is the magenta one.

IV. Results

Looking at Table 3, the pathing algorithms suffer terribly from accumulated error over the course of the path, colliding with obstacles that its desired path would pass. Even the relatively best one, the 200

node estimated curve still shows an error that would significantly decrease the effectiveness of the pathing algorithm. The otherwise reliable “spin and move” algorithm in particular suffers due to the propagation of error. Although not shown in the recorded data, earlier in the design, the pathing results without the course correction algorithm were not this bad. As Figure 7 shows, the old pathing shown on the right demonstrates much more noticeable divergence from the planner route than the new one. The new one has a mask applied to its occupancy grid, but that wouldn’t explain the growth in error. One change that might affect it is that the new simulation is scaled up. Originally, there was a disparity between the size of the obstacles and rover and the size used for calculations. This was corrected for using a scale factor term, but when that became cumbersome, the sizes of the obstacles and the rover were increased to match everything else. So, while everything is still relatively the same size so it should in theory work the same, evidently the simulator introduces more error sources than the smaller one did which does make some degree of sense. In any case, this new error source is suitable to demonstrate the effectiveness of the course correction algorithm.

The effectiveness of the course correction algorithm was much more effective than initially expected. As mentioned earlier, the propagation of error was not originally as much of an issue in the simulation and getting the course correction itself to function correctly was a challenge. Although the algorithm not stopping and the mid operation error reports being irregular are still issues, ultimately the algorithm greatly increases the robot’s ability to path close to the desired target. According to the data in Table 4, all of the algorithms greatly improved in performance with the inclusion of the course correction. However, the 400 node “basic curve” graph is still not very effective. The only noticeable downside is the increased failure rate and potentially longer traversal times. The latter is a worthwhile trade off to ensure the robot actually reaches its destination, but the former is an issue. However, the cause of this is likely fixable. These errors were caused when the robot tried to correct its path when in a pose that did not connect to any node in the graph and this typically occurred for one of two reasons given that it is unlikely that it truly was unable to reach any of the nearby nodes. Either the pose it found itself at was outside the occupation grid, or in a pose “occupied” by an obstacle in the occupation grid. Not allowing paths going into the region outside the occupation grid was an arbitrary restriction place on the graph construction algorithm so that is simple to solve. Additionally, the other instance is caused by the robot finding itself within the extended buffer zone placed on the occupation grid by the mask. Getting rid of the extended buffer zone during graph construction is not even necessary to address this, it just needs to be removed from occupancy grid that is given to the path course correction algorithm. This will allow the base graph to still give some clearance between its predicted paths and the surrounding obstacles at least for all its original edges.

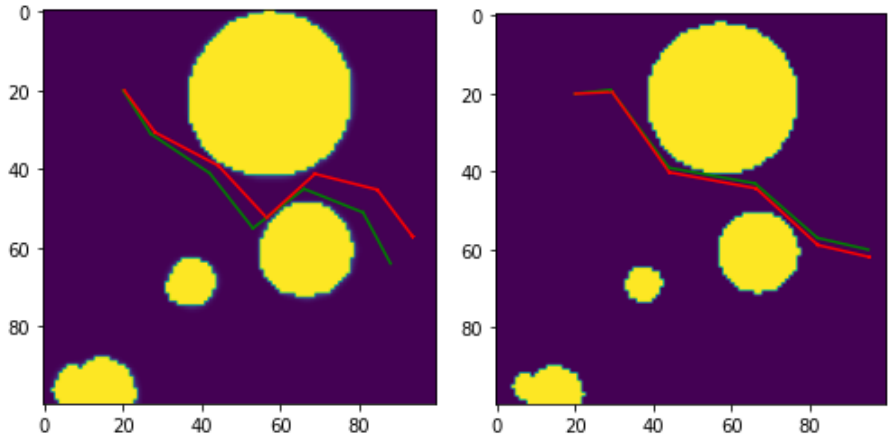

Figure 7:Difference between path divergence trends for smaller scaled simulation (right) and larger scaled one (left).

Now to address the issue of the particle filter. Originally, the plan for this project was to have the robot deal with error in the commands sent to its wheels and while the particle filter does work, it is a bit temperamental and still needs work. The original problem is that the simulation introduces error on its own, so introducing more of it artificially will cause problems if the filter only account for the artificial error. However, if the artificial error is removed, then getting the filter to work is just a matter of finding the correct way to model the existing noise such as experimentally adjusting the variance values used in the particle filter. There is still some possibly some issue in the measurements the simulator returns as shown by the behavior of the course correction algorithm, but the particle filter itself works. It just does not have a use unless something like the course correction algorithm does something with its predictions of the robots current state. Additionally, the filter does need to be improved to account for collisions in its predictions.

V. Conclusion

Looking over the result of this experiment, there is a clear disparity in performance between some of the pathing algorithms. The “basic curve” algorithm simply needs too many nodes to be effective. It is faster than the others to construct a graph with a certain number of nodes, but it still is not effective at double the number of nodes every other method function well at and the graph becomes exponentially more costly to make as the node count increases. Originally, the “spin and move” method seemed the most effective, but it seems to suffer even more than the others from propagation of error likely due to how errors in its initial spin behavior will drastically effect the overall outcome of a movement. The “iterated curve” was another surprise. The algorithm originally was very strict at forcing the convergence with the target node making each connection hit the absolute step limit before accepting the output. This not only made each calculation very costly, but the time components of the later commands in a motion quickly became smaller than the simulation step size which produced a large amount of error. Relaxing the restrictions solved this and even if the result is not as close as the completely converged one, typically the orientation is a good match and the error is primarily from the offset position. This is advantageous because orientation errors typically cause more issues with error propagation that pose ones in this pathing problem. Finally, the “estimated curve” method performed extremely well. It is the slowest among the four in computation time, but only the “spin and move” can function correctly at a relatively low node count. Although it was originally just a small part of the project proposal, the unique

challenges of the inter node pathing required much more thought and consideration than initially expected.

While the pathing methods had some accuracy issues on their own, the course correction algorithm addressed many of their accuracy issues. However, these results also raise the question of whether connecting nodes with a strict series of wheel commands is truly advantageous. Although the correction algorithm makes the pathing more accurate, it tended to be a bit clumsy, producing an irregular and inefficient moving pattern. The set wheel command between nodes exacerbated the issue of error propagation as they act on the assumption there is no initial error. An inter-node pathing that is a bit more dynamic that could adjust based on what it knows about its specific current state may be more advantageous. It could start with some baseline associated with a given edge between nodes, either something like the estimated curve command series or a desired path to traverse, and then use a cost function of some kind to adjust itself to a desired route, taking into account how far away from the desired path it is at any given time. This type of system would need to have an accurate grasp of its current state which would make something like the particle filter even more important. Despite these considerations, the fact that these methods require relatively little in computation time after the graph is already constructed is noteworthy.

VI. Appendix

GitHub Code Repository Link: https://github.com/RobertS9438/Differtial-Drive-Robot-Pathing