BLine

Sparse Lightweight Polyline Path Generation and Tracking for Holonomic Robots

Edan Liahovetsky
Team 2638 Rebel Robotics

----------Abstract----------

Holonomic (omnidirectional) mobile robots rely on trajectory generation and tracking to execute autonomous motion. However, common curved-path approaches including Bézier and polynomial trajectories increase computational cost and complexity of implementation. This work introduces BLine, a sparse, lightweight polyline path generation and tracking method designed for holonomic drivetrains. BLine models paths using translation targets, rotation targets, and waypoints with parameterized handoff radii that advance the active target holonomic heading when the robot intersects translation targets within a local neighborhood. During path tracking, the translational velocity is calculated using proportional-integral-derivative (PID) control minimizing remaining path distance. Rotational velocity is controlled independently using a PID controller on linearly interpolated translation targets using distance-ratio progression between targets. Kinematic constraints are enforced through velocity clamping and acceleration limiting. To evaluate performance, 20 randomized obstacle maps were generated in MonteCarlo trials and solved using a Theta* seeding solution and optimized using an Artificial Bee Colony (ABC) optimizer with physics simulation. Across trials, BLine achieved a 97% reduction in path build time, maintained negligible differences in total energy use, and introduced a modest 2.6% increase in execution time against curved-type paths. The primary tradeoff with BLine compared to curved type paths is higher intermediate cross-track error (CTE) with a 15.5% increase in CTE compared to curved-type paths, reflecting the inherent behavior of the BLine algorithm. Thus, BLine is a strong choice when fast planning, lightweight computation, and accurate waypoint interception are critical while curved-type paths remain optimal when minimizing continuous CTE and smoothness are priorities.

----------Introduction----------

Autonomy achieved through trajectory planning and tracking:

Autonomous motion is achieved through the process of trajectory generation and tracking. During this process, start and end points are defined by the mission goal, where the starting position is generally the robot's current position. The strategy used to plan the guidance of a system from its starting to ending position is known as trajectory generation or path planning. In this phase of autonomous motion, the idealistic trajectory or path is computed which is then followed during the final phase known as trajectory tracking. During this final stage, the robot chassis is guided along the trajectory via a tracking control loop, which calculates the necessary chassis speeds needed to command the drivetrain along the path and towards the final destination.

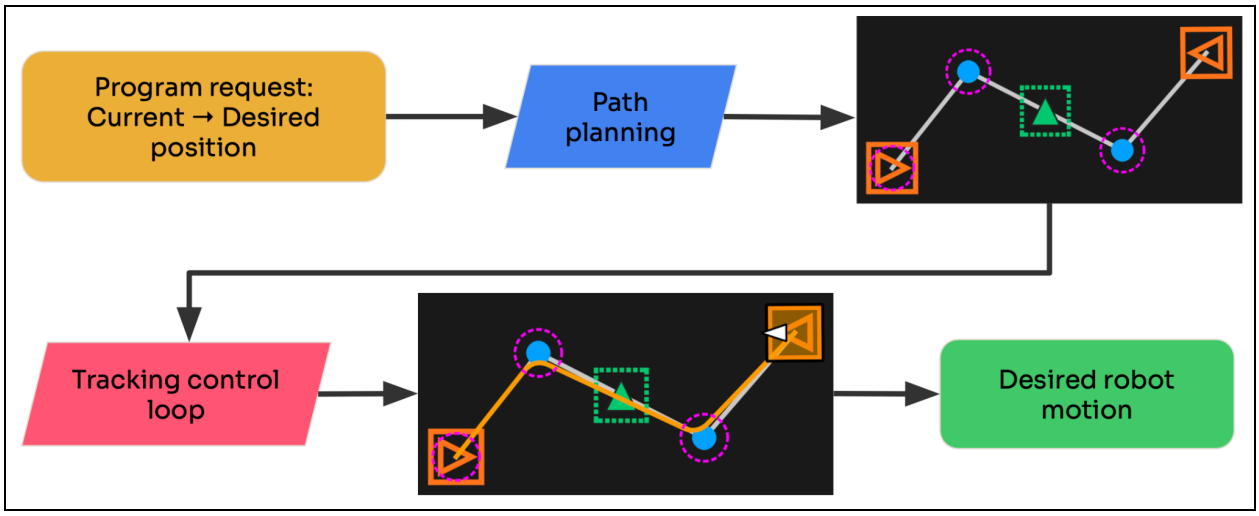

Figure 1. Path planning/tracking flow diagram (figure generated by student author using BLine GUI and Google Slides (2025))

Curved-type paths - Béizer:

Curved type trajectories are characterized by their smooth and continuous nature. One of the most common styles of curved path is the béizer path. Bézer paths are defined through the use of anchors and control points, which provide finite control over the path profile. The tracking of curved type paths generally yields a motion profile relatively even accelerations and low jerk. The computation of these paths requires the discretization of the curve profile and is relatively intense compared to that of line-type paths.

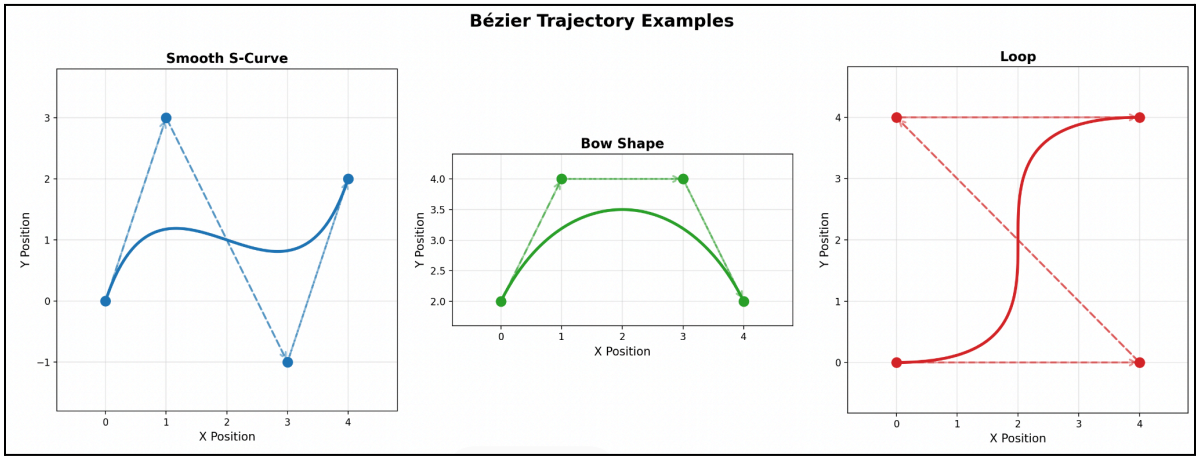

Figure 2. Bezier trajectory examples (figure generated by student author using Matplotlib (2025)).

Line-type paths - polyline paths:

Line-type paths (also known as polyline paths) are defined as a set of linearly interpolated sparse waypoints. These paths are made up of multiple small line segments (also known as polylines). Depending on the use case, polyline paths can be discreetly defined or defined continually. The tracking of line-type paths results in a relatively jagged robot motion due to the high centripetal acceleration imposed on the chassis at each bend of the polyline. Compared to curved-type paths, the computational complexity of line-type paths is far lower due to the simple nature of linear interpolation. Furthermore, the definition of these paths is also relatively simple, without the need for extra control anchors (as used in bézier paths).

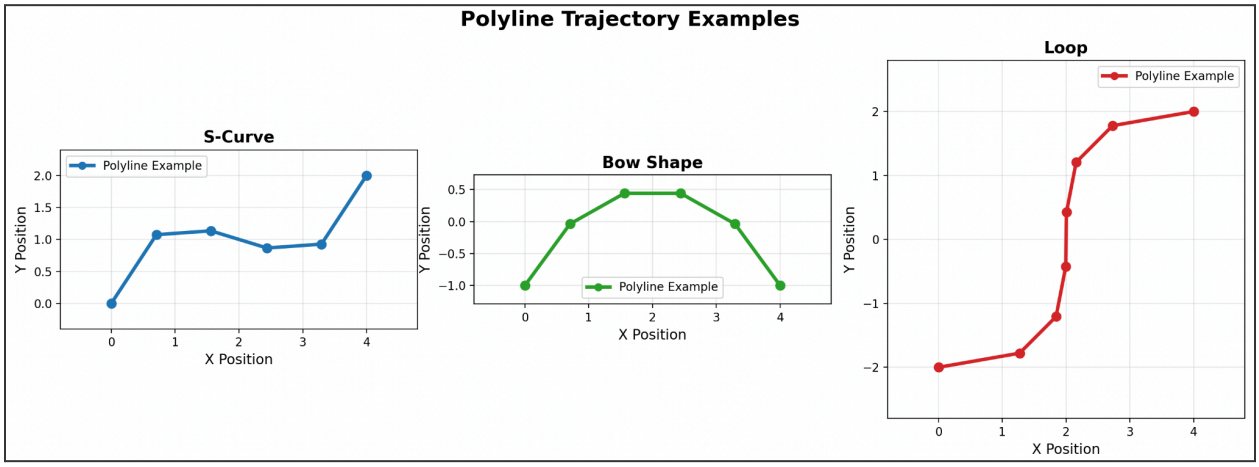

Figure 3. Polyline path examples (figure generated by student author using Matplotlib (2025)).

Trajectory Tracking:

Path tracking is the means by which a generated trajectory (curved or line type) is followed. This is achieved via the trajectory tracking controller, which commands chassis speeds to the robot such that the trajectory is most optimally followed. Similar to the methods of trajectory generation, there exists multiple methods of trajectory tracking. These include controllers which model the system in state space such as MPC, LQR.

----------Methodology----------

Path structure and elements (figures 4 and 5)

The path consists of three types of elements: translation targets, rotation targets, and waypoints. All elements are stored in a list in the order that the robot encounters them as it follows the path.

1. Translation target: describes a translation which the robot encounters along the path. Defined by point in 2d space with a variable called “handoff radius”.
2. Rotation target: describes a desired robot rotation along a point of the path. Defined by a rotation variable the distance ratio between two translation targets.
3. Waypoint: a translation target with a rotation target tacked right on top. Packaged for user simplicity.

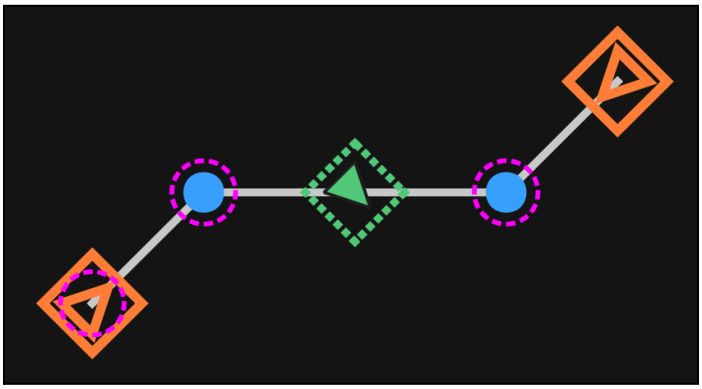

Figure 4. BLine path with waypoints (orange), translation targets (blue), and rotation targets (green) with handoff radii (magenta) and example robot tracking (made by researcher).

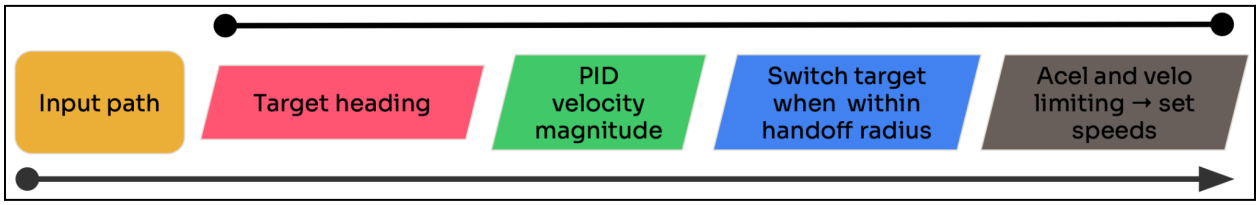

Figure 5. BLine control flow (made by researcher).

Path following algorithm (figures 4 and 5)

A. Calculate translational velocity
   1. If the drivetrain is within the handoff radius of the current translation target / waypoint, switch to the next translation target / waypoint.
   2. The drivetrain heading is the direction from the drivetrain's current position towards the current target translation.

3. The magnitude of the drivetrain's velocity is calculated by the total remaining path distance fed into a PID controller which aims to drive the total remaining path distance to zero.
    a) The path distance is calculated by taking the sum of straight line distance between the robots current position and the next immediate target and adding it to the straight line distance between all remaining consecutive translation targets.
    b) The PID controller is empirically tuned to the drivetrain.
4. The drivetrain's velocity is then componentized into vx and vy speeds

B. Calculate rotational velocity
    1. Calculate the distance ratio of the robot between the current translation target and the previous translation target
        a) Calculate the robot's distance from the previous translation target
        b) Calculate the total segment length between the two translation targets
        c) Divide the robot distance by the segment length to get the ratio
    2. If the robot translation ratio is greater than the current rotation target ratio, move on to the next rotation target.
    3. Calculate the robot rotational velocity using a PID controller with a setpoint of the rotation target and a measurement of the current robot position.
C. Limit the drivetrain speeds to path constraints
    1. Due to the aggressive step changes which result from the PID controller calculations and the great range of controller domain mapping, it is necessary to limit the velocity and acceleration of the calculated speeds.
    2. First, limit the translation and rotation velocities by polling the path for defined constraints and then simply clamping the speeds. The translation is clamped by magnitude and not by its individual components.
    3. Limit the rotational acceleration via a simple slew rate limiter.
    4. Limit the translational acceleration by making sure the magnitude of the vector mapping the previous desired translational velocity to the current translational velocity does not exceed the maximum allowable translational acceleration divided by the delta time.
D. Finally, pass the speeds into the drivetrain subsystem.

Simulation

The simulation is built on top of the WPILib library, using its flywheel, DCMotor, and kinematics models to create a digital twin of a holonomic swerve drivetrain. The simulation models the entire robot software stack and hardware suit, including path generation, path tracking chassis calculator, robot chassis kinematics, motors, and odometry based positional estimation. It runs on a simulated 20ms loop cycle, which models the real world performance of the robot hardware specifications.

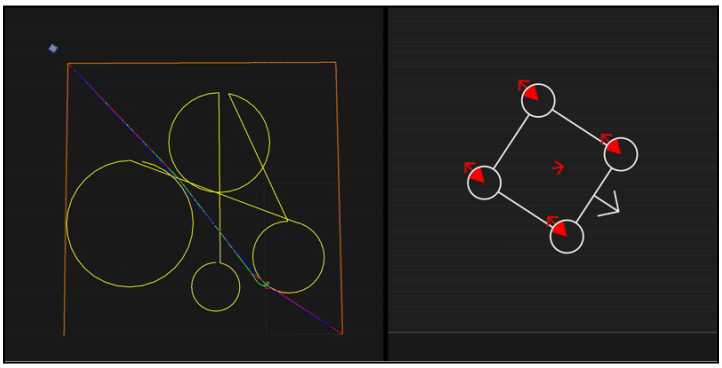

Figure 6. Physics simulation (made by researcher).

Test Map Generation

In order to properly test the performance of path planning approaches, randomized test maps were generated. These maps were designed to test a broad range of pathplanning cases ranging from straight line motion to multi-turn testing. Each map was 30 by 30 meters in size and was populated with circles (acting as obstacles) ranging from 2 to 7 meters in diameter, with the requirement of a minimum of 0.5 meters in spacing between each obstacle. Each map was also tagged with a max allowable velocity and acceleration to compare the performance of the pathplanning approaches over a variety of drivetrain constraints.

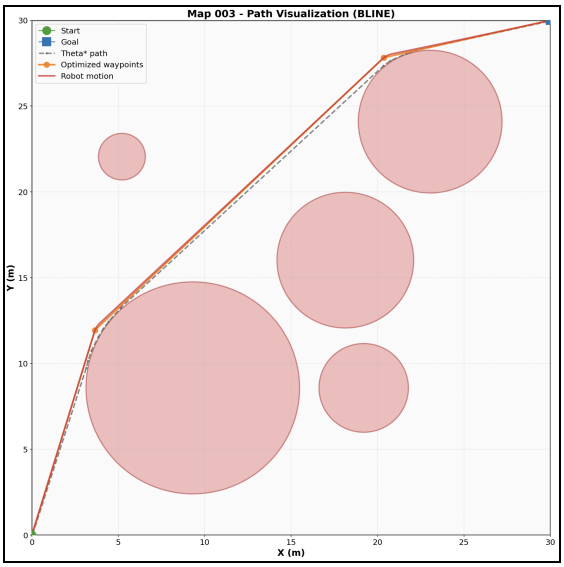

Figure 7. Resultant map and solution from ABC optimizer (made by researcher).

Initial Map Solving

In order to give the Artificial Bee Colony (ABC) optimizer a good starting point for map solving, the Theta* algorithm was used to give an ideal shortest path trajectory. Theta* is a path planning algorithm based on the popular A* algorithm. The benefit of Theta* is that it eliminates redundant graph points which are within the line-of-sight of each other, which prevents over sampling.

Artificial Bee Colony (ABC) Optimizer

ABC is inspired by the way that natural bee colonies scavenge for food. There are employed bees who exploit known sources of optimization, onlookers who select probable dimensions to optimize, and scouts who randomly search for sources to optimize.
The algorithm cycles between the three "bee population phases" to optimize the given criteria. The ABC algorithm was used due to its relative simplicity and efficacy compared to other optimization approaches such as the genetic algorithm (GA). When compared to GA, ABC demonstrates comparable if not better performance while reducing parameter count and tuning complexity.

The cost function acts as the validation metric by which the optimizer aims to appease. The optimizer looks to minimize the function, which is defined as follows.

1. The penalty for time (Ctime) is calculated by multiplelying scaler Kt by the absolute value of the ratio between the simulated path following time (Ts) to the ideal Theta* path time (Ti): $C_{\text{time}} = K_t|\frac{T_s}{T_i} - 1|$
2. The penalty for cross track error (Ccte) is calculated by multiplying scaler Kc by the total cross track error accumulated over the course of the path divided by path length (Lb): $C_{\text{cte}} = K_c\frac{\text{CTE}}{L_b}$

3.
4. The penalty for waypoint precision (Cwp) is calculated by multiplying scaler Kc by the sum of the CTE at each interior (CTEW) waypoint over the total number of interior waypoints (N): $C_{\text{wp}} = K_w \frac{\text{CTEW}}{N}$
5. The penalty for small segment length (Cseg) is calculated by multiplying scaler Ks by the clamped value of total path length (Lb) over the smallest segment length (S) times the number of interior waypoints (N) all minus 1: $C_{\text{seg}} = K_s \max\left(0, \frac{L_b}{SN} - 1\right)$
6. The penalty for total path length (Clen) is calculated by multiplying scaler Kl by the total path length (Lb) over the ideal path length (Li): $C_{\text{len}} = K_l \frac{L_b}{L_i}$
7. The penalty for collisions (Ccoll) is calculated by multiplying scalers Kb, Kp, Kr, and Kd by the polyline path collision flag (Wpl), the Wpl and the maximum normalized polyline path obstacle overlap (Wdl), the robot trajectory collision flag (Wp), by and the Wp combined with the maximum normalized robot trajectory obstacle overlap (Wd):
   $$C_{\text{coll}} = K_b W_{\text{pl}} + K_p W_{\text{pl}} W_{\text{dl}} + K_r W_p + K_d W_p W_d$$
8. The total cost (C) is the summation of all previously defined cost terms:
   $$C = C_{\text{time}} + C_{\text{cte}} + C_{\text{wp}} + C_{\text{seg}} + C_{\text{len}} + C_{\text{coll}}$$

ABC Path Manipulation

The paths were initialized as the set of waypoints solved by Theta* for a given map. The ABC optimizer was able to dictate the presence of a waypoint via a binary scaler ranging from 0-1 with values greater than 0.5 signaling presence. The ABC optimizer was able to dictate the handoff radius of each waypoint (if applicable). The ABC optimizer was able to dictate the offset position of each waypoint's translation by up to 0.5 meters in the x and y axis. The ABC optimizer was also able to insert up to 5 additional waypoints based on the relative position of other waypoints with translation, presence, and handoff radius control.

Validation Design

20 randomized test maps were generated. Each test map was initially solved by the Theta* algorithm. These initial solutions were passed to the ABC optimizer, which optimized within the constraints of the map. The optimizer was run for 80 epochs and 12 workers for each map. The optimizer loops through the physics simulation and calculates cost based on simulated robot performance. It outputs the solution with the lowest cost to serve as the final "best case" path for the respective trajectory tracking and generation solutions.

Data analysis

Each map solution and robot path was compared for general visual semanticse. Path structure - linearity, curvature, vagueness, sparsity. Robot deviations from paths at set points. Cost function history to ensure proper optimizer functionality. Final cost function output to cross

compare points of tradeoff that the optimizer made for each algorithm. Total drivetrain power consumption was compared, and minimum, maximum, and average consumption was analyzed to determine efficiency differences in the algorithms. Drive and steer motors were independently measured. The average time time of each path solution over the 20 generated maps was compared to understand the relative swiftness of each approach. The average path construction time (time it takes for the computer to construct paths in code measured via system clock) was calculated in each run and compared to determine relative computational efficiency. Average energy consumption (Wh) was compared on a per path basis. Average power (W) was compared on a per path basis. CTE power path length was compared on a per path basis. CTEW was compared on a per path basis. Power and CTE (W) metrics reveal energy and precision performance of both solutions respectively.

BLine was also rigorously tested and deployed during the FIRST Robotics Competition season and was validated against other FIRST teams based on autonomous score ranking. This served as a generalized performance metric for the system and its integration with hardware, and software in a production environment. Performance was also compared to prior years of competition.

----------Results----------

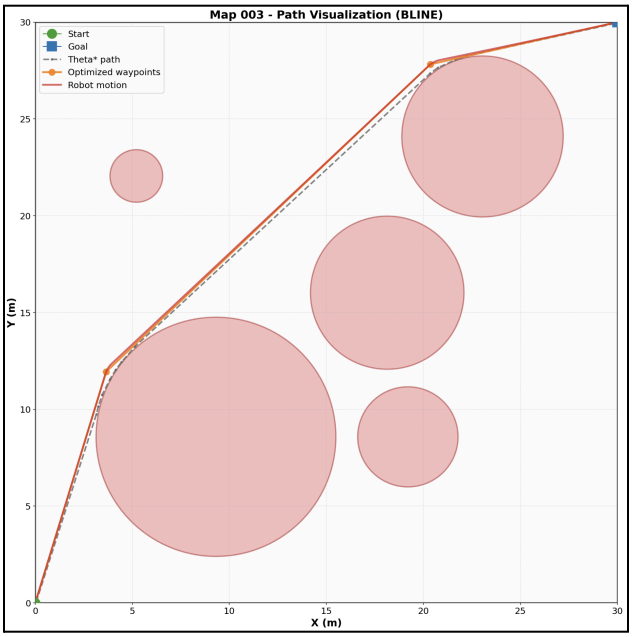

Figure 8. Generated problem maps (figure generated by student author using Matplotlib (2025)).

20 Monte Carlo trials were run. In each trial, two maps were generated, solved, and optimized for BLine type paths and curved type paths. Data was then compiled using the Numpy and MatPlotLib libraries. Figure 8 is an example resultant map from trial 3. Shown is the BLine generated trajectory, the initial theta* path, and the final resultant robot trajectory. The figure shows the optimized BLine path from the sample. The close alignment of the robot to the original path demonstrates the functionality of the ABC-simulation-tracking pipeline. Additionally, the non-collision metric along with CTEW and relatively low time penalization with only 10% overshoot from the absolute best case unrealistic scenario where the robot has no limit on centripetal acceleration show that the pipeline is working reasonably.

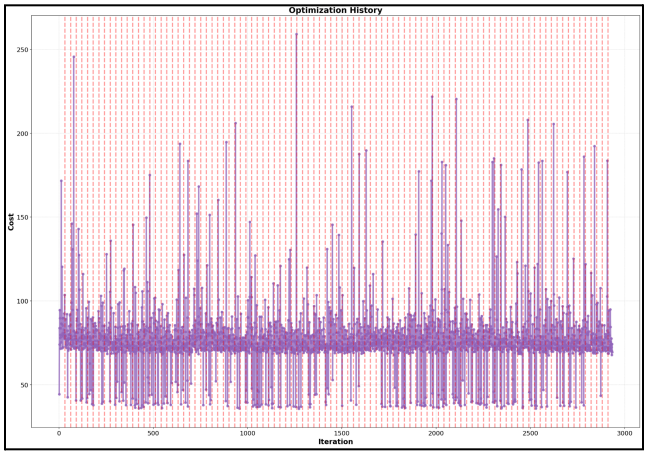

Figure 9a. Resultant optimization history from ABC optimizer (figure generated by student author using Matplotlib (2025)).

Figure 9a. Is the result of the ABC optimization cost function over the ~3000 iterations run over the course of a single BLine path optimization in trial 3. The lowest of these values was chosen. It can be seen occasional low and high spikes of good and bad optimization runs, which demonstrates the general functionality over time classic of a ABC optimizer.

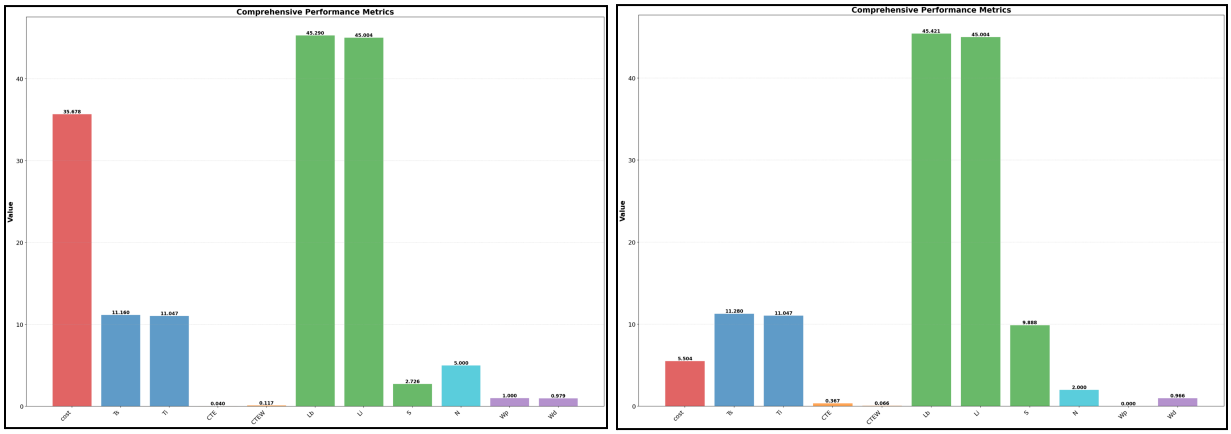

Figure 10a/b. Resultant best cost evaluation from ABC optimizer (figure generated by student author using Matplotlib (2025)).

Figure 10a/b are resultant optimization history metrics obtained from the optimizer for trial 3 for BLine and curved type paths respectively. It can be seen that BLine demonstrates higher CTE but lower CTEW, which signals precision at each waypoint but deviation from the path over longer stretches due to the direct heading nature of the path. This also signals potential undertaking of the CTE reduction PID controller which is designed to mitigate these errors.

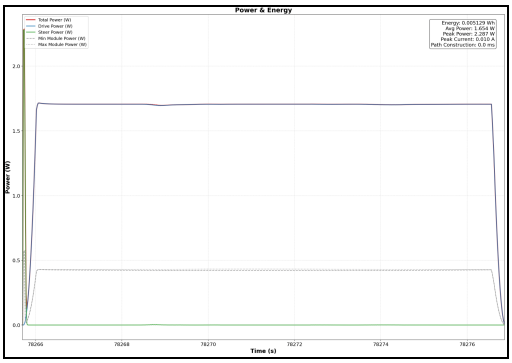

Figure 11a. Resultant power and energy consumption from physics simulation (figure generated by student author using Matplotlib (2025)).

Figure 11a represented the resultant energy consumption over time as measured by the physics simulation on trial 3 best curved-type path. The consumption curve is appropriate, showing a high period of sustained power draw with start and end periods of acceleration/deceleration.

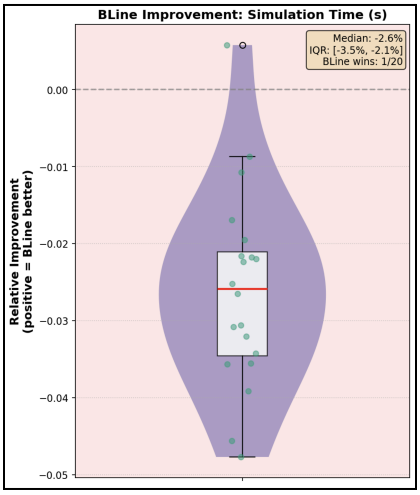

Figure 12. Improvement in total mission duration as measured by simulated system clock (figure generated by student author using Matplotlib (2025)).

Figure 12 is a violin plot of the relative improvement of BLine when compared to the curved type path in respect to total path following execution time (from the start to end motion of the chassis). It shows a large spread of values with a general concentration along the median with a tight IQR of 1.4%. The median improvement of -2.6% shows a slight performance drop when compared to the curve-type configuration.

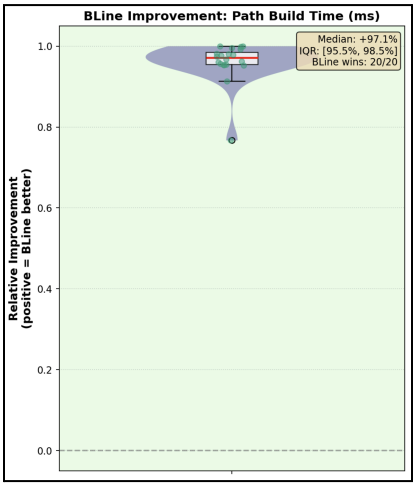

Figure 13. Improvement in built time in real system clock time (figure generated by student author using Matplotlib (2025)).

Figure 13 demonstrates the vast gains in computational efficiency when constructing a BLine path compared to a curved-type path with rightly clustered gains of 97.1%, showing

vast improvement. The path build time was measured using the simulation workstation's internal real time system clock.

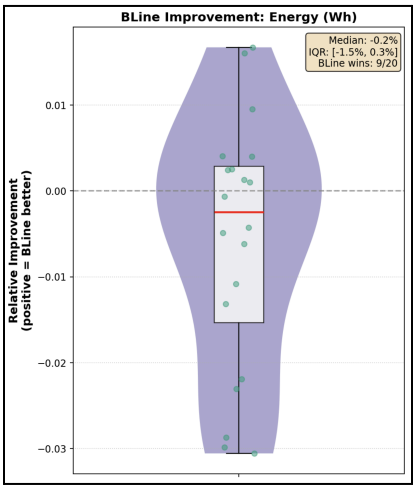

Figure 14. Improvement in simulated power consumption in watt-hours (figure generated by student author using Matplotlib (2025)).

Figure 14 demonstrates the improvement in simulated power consumption with negligible difference between the two groups. It was measured using the motor power equation on the physics simulated motors and integrating wattage over the course of the path run.

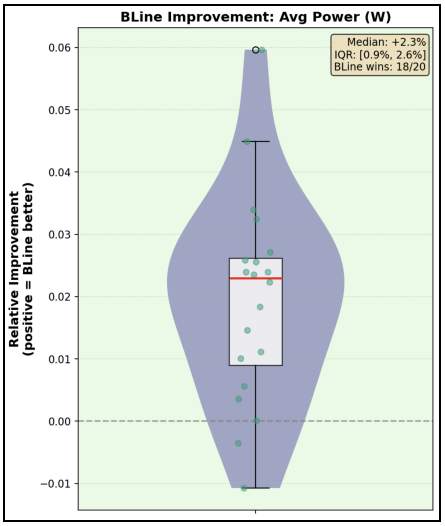

Figure 15. Improvement in simulated average power in watts (figure generated by student author using Matplotlib (2025)).

Figure 15 demonstrates that the average power draw of BLine is less than that of curved type paths. It was measured similarly to the wh consumption.

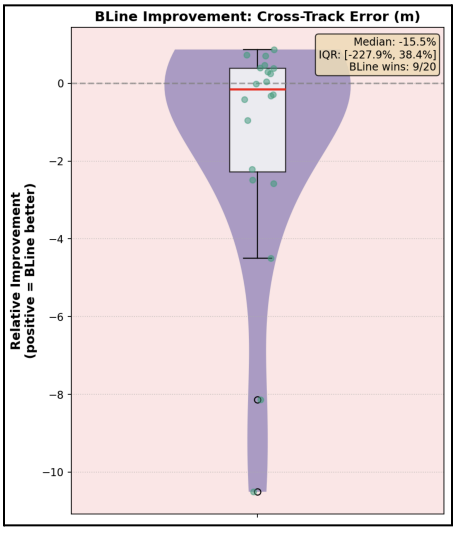

Figure 16. Improvement in cross track error per meter (figure generated by student author using Matplotlib (2025)).

The cross track error was measured by integrating the cross track distance between the robot chassis from the ideal path over the

course of the trial run. There is a decrease in CTE performance with a median of -15.5% improvement.

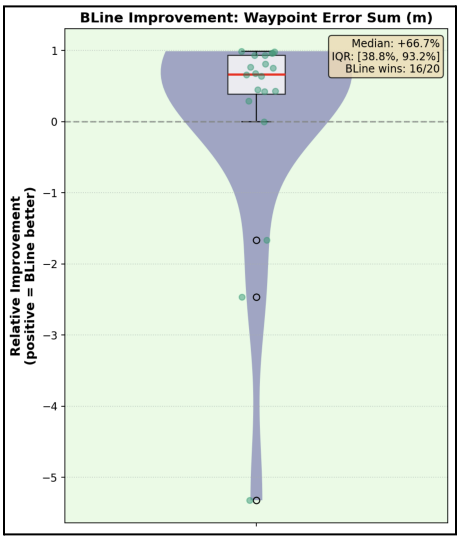

Figure 17. Improvement in total cross track error at waypoint(figure generated by student author using Matplotlib (2025)).

The CTEW was measured by summing the CTE only at each waypoint, not over the entire course of the path. It is a metric of waypoint precision. BLine shows a significant improvement of 66.7% over curved type paths.

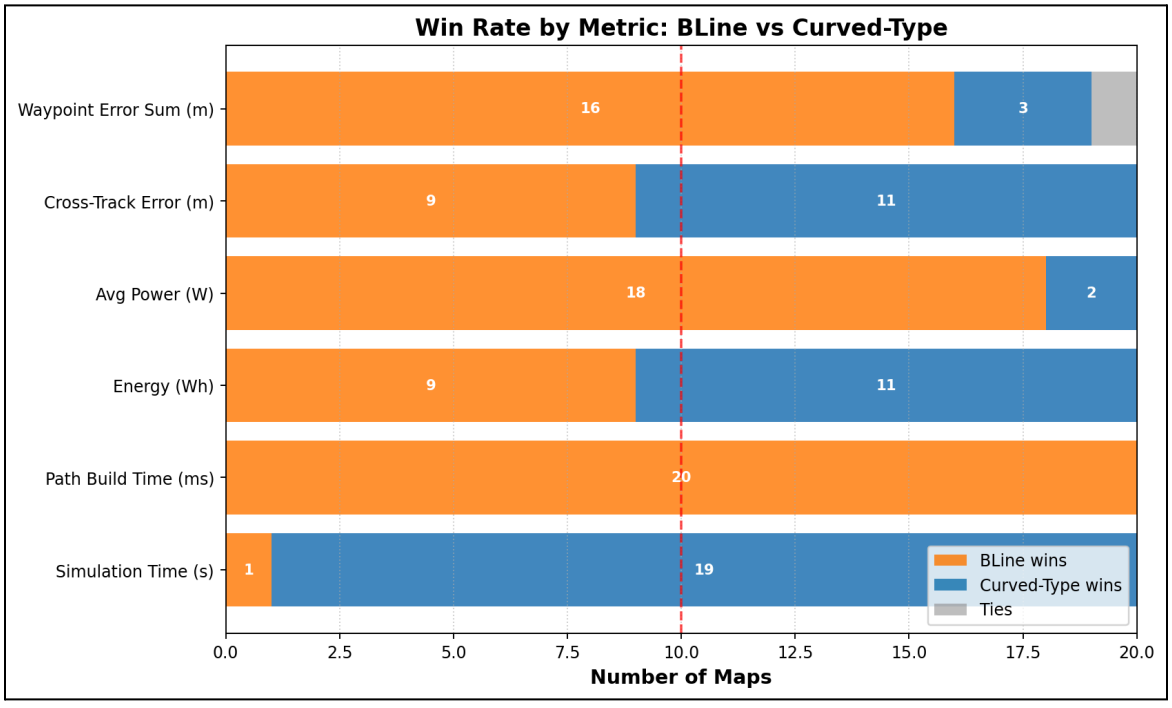

Figure 18. Win rate by metric (figure generated by student author using Matplotlib (2025)).

A win for either solution was calculated on each trial, checking if improvement is positive and negative compiling to a win for BLine and curved type paths respectively.

----------Discussion----------

The goal of this study was to develop and evaluate BLine, a sparse lightweight polyline path planning and tracking solution against existing curved type path approaches. The hypothesis was that BLine would yield comparable tracking and planning performance while offering gains in computational efficiency, simplicity, and user code integration. These tradeoffs were often overlooked by previous literature which focused on maximal performance while compromising ease of use.

Evaluation of key findings:

Computational efficiency

Notably, BLine achieved 97% gains in computational efficiency when compared to curved type approaches. This increase in efficiency was a result of the simplicity of the construction of BLine paths, not requiring the continuous construction of all robot possible positions at path creation as do curved paths. Instead, BLine only needs to store the discrete sparse user defined waypoints and reference them during run time. As a result, the only computation needed at initialization is simply loading classes and the user parameters into memory. These gains were consistent across every single evaluation / case, showing that they are independent of path size and length.

Precision-tracking tradeoff

A 66.7% improvement in (CTEW) was found, the precision and intersection of waypoints during path following, shows that BLine more precisely intersects waypoints when compared to BLine. This suggests that BLine is amply effective at intercepting user waypoints.

A 15.5% increase in total path cross-track error (CTE) was found across cases. However, this is an artifact of the ABC optimization process which prioritized total path time and collision avoidance with obstacles. CTE was not of critical importance to the optimizer and the deviation is negligible for the majority use cases. Users can easily reduce cross-track error through tuning of the cross-track-error feedback loop and handoff radii.

This has implications for user choice of when to use BLine or curved type paths. In applications where intermediary precision is paramount, curved type paths would be deemed more effective.

In applications where waypoint precision and minor intermediary imprecision are acceptable, BLine comes out to be the better choice for its other benefits.

Energy and time

There is negligible difference in total energy consumption (Wh) between approaches with BLine showing slightly lower average power draw (W). BLine's slightly longer execution by 2.6% is offset by less demanding energy requirements. BLine's less extreme energy demands could be explained by less demand for sustained turning and more simple general motion, with steer motors only being active during certain portions of motion while curved type paths require steer motors to act continuously along the path's motion.

Ambiguity

These results are in line with current literature, especially the work done by and others which explicitly compare the performance of straight and curved type paths. It is also in line with the theoretical basis for the hypothesis, as mentioned previously.

Limitations:

This study utilized physics simulation in order to model a holonomic drivetrain, allowing for the use of the ABC optimizer and path generation to test a broad range of use cases. The simulation used assumed best case scenarios for all systems, and did not model frictional forces, ground traction, etc. Motors were modeled as flywheels. In order to improve the generalizability of the results, a more holistic simulation could have been constructed, as is generally the case with this type of work. Conversely, it is possible to take the inverse approach, opting for a massless drivetrain with modeling only on the chassis speed level, and not beyond that. This would allow for the testing of potentially an order of magnitude more cases given the relative computational simplicity of this approach. The aforementioned research direction of modeling the system more thoroughly would also greatly increase computational time, so there is a tradeoff to consider there.

Real life validation and hardware in the loop testing would also prove valuable to compare the efficiency and performance of solutions in the real world on edge computing hardware with physical motors. A series of post validative trials aiming to affirm the findings of Thai study would solidify the understanding of the effects, benefits, and tradeoffs of BLine vs Bézier approaches. It would also more rigorously test the claims of improved user code implementation and so forth as in simulation, pipelining for real hardware is not necessary and there is less time pressure to implement paths etc.

The methodology for generating maps could also have been improved, using multiple geometric shapes in order to push the optimizer to test around corners.

----------Conclusion----------

This study set out to determine whether the BLine, the proposed sparse polyline path planning and tracking approach for holomic drivetrains, could achieve comparable performance to curved type paths while improving computational efficiency, simplicity, and ease of user code integration. This study's results support this goal. Across 20 randomized montecarlo simulations, BLine showed a 97% reduction in path build (construction) time, maintained similar overall energy use, and a small penalty in execution time of 2.6% compared to curved-type paths. BLine also demonstrated substantially better waypoint precision (CTEW) of 66.7%, which suggests that BLine shows high reliability in hitting user-defined waypoints.

The primary tradeoff observed between BLine when compared to curved type paths is higher intermediate cross-track error (CTE) with a 15.5% increase in CTE when compared to curved-type paths, reflecting the inherent behavior of the BLine tracking algorithm. This yields the conclusion that BLine is a strong choice when fast planning, lightweight computation, and accurate waypoint interception are critical while curved-type paths remain optimal when minimizing continuous CTE, and smoothness and inherently strict path adherence is the priority (although CTE could have been reduced further through user tuning of the CTE PID controller).

Computational Efficiency

A direct improvement over PathPlanner and Choreo is computational efficiency and simplicity when creating simple paths.

BLine does not need to precompute or discretize a Bézier trajectory into finite timestamps before the controller can follow the path. The path gets passed to the tracking controller immediately after creation, which provides loop cycle time gains for real-time teleop applications. This means you don't have to do any pre-computing as you might with Bézier trajectories. Of course, for pre-baked paths there's no difference—the majority of computation happens during path creation regardless of the tool.

Ease of Controller Tuning

The BLine tracking controller achieves good (if not better) results with less hassle. Tuning a time-parameterized PID controller to follow curved paths (Bézier or otherwise) can be difficult,

especially for newer teams (I unfortunately speak from experience). Over-tuned gains easily cause erratic behavior and jittering, while under-tuned gains cause the robot to fall behind during acceleration. This is exacerbated when the drivetrain isn't perfectly tuned, making it harder to push a chassis to its true max acceleration and velocity limits. Additionally, it's historically been very common for teams to run a second profiled PID alignment routine after a PathPlanner-style path ends to overcome the challenges of time-parameterized tracking and early finishes.

BLine avoids these issues simply by the nature of its PID controller setpoint: the path's endpoint. By having the translation controller minimize total path distance remaining, the controller output is high at the start and properly tapers off at the end. By imposing user-defined acceleration and velocity limits on the controller output, users can ensure the robot properly utilizes chassis acceleration and hits max velocity irrespective of drivetrain or tracking controller tuning.

The tuning of BLine's translation PID controller is only critical at the very end of path tracking, where the robot must decelerate and stop. This domain is far more manageable compared to the time-parameterized approach. In my testing, I achieved a good translational controller config in around 5 minutes of tuning. There's no large performance penalty for under-tuning—the difference between optimally and sub-optimally tuned controllers is only noticeable at the very end of the robot's motion, making the tracking controller very forgiving.

In contrast, a time-parameterized controller's response is apparent along the entire path, and poorly tuned chassis or gains are noticeable across the robot's entire motion, making sub-optimal tuning less forgiving.

In comparison to AutoPilot, the ease of tuning should be similar, as AutoPilot also circumvents time-parameterized tracking issues via its own approach.

Path Simplicity

BLine paths are simple and relatively quick to create.

Consider a straight-line path with no intermediary elements, simply going between two points where the robot is at rest at both. Both Choreo and PathPlanner would essentially create a trapezoidal motion profile, introducing the aforementioned time-parameterized controller pains for a very simple path. This takes more computational resources, adds hassle (control points, anchors, Choreo optimizer), and results in a nearly identically performing path compared to BLine.

BLine can also function with only one path element (waypoint), useful for aligning to the Reefscape Reef or Crescendo Amp in teleop or auto. This essentially turns BLine into a bare-bones drive-to-point PID command without the hassle of a separate solution.

Intermediary Elements & Constraints

For paths with intermediary elements, BLine retains its ease of controller tuning and computational simplicity. Just as Bézier paths have control points and anchors, BLine paths use path elements, handoff radii, and translational velocity limiting as the primary means of motion control.

Ranged constraints ensure the robot doesn't overshoot intermediary elements due to high velocities. To create more complex BLine paths, users manually define these constraints. After creating one or two paths, users quickly build intuition for appropriate constraints. The GUI simulation also aids in understanding robot behavior.

Through this system of constraints, handoff radii, and path elements, the forgivingness and performance of the translational PID controller and computational simplicity are maintained. Furthermore, users gain fine control over exact robot behavior at individual path elements (velocity and precision). This opens the door for rapid and effective empirical tuning, testing, and validation—a product and benefit (depending on your team's workflow) of the BLine architecture.

BLine provides as much intermediary control as the user wants (within polyline reason, of course) and can create complex autonomous routines. In contrast, AutoPilot would perform well in the straight-line case and single-point on-the-fly case, however it's not designed for paths requiring intermediary element control (3+ translation elements) and shines best in simpler cases.