EBS221 HW3

Zhian Li, Willian Klippel-Huber

May 19th, 2025

1 Introduction

Autonomous field traversal is a core capability for agricultural robots tasked with spraying or cultivating row crops. The primary objective is to visit all crop rows with minimal time, distance, and actuation effort while obeying the kinematic constraints of the tractor. A standard approach is to follow parallel passes aligned with the field's geometry; however, the real challenge lies in efficiently handling headland turns within limited turning radii and ensuring smooth path tracking.

In this assignment, we develop a complete system for coverage planning and execution in a rectangular row crop field. The task includes: generating the field layout, computing a node-to-node cost matrix, solving an open TSP for optimal row visitation, generating a waypoint path, and tracking it using a pure-pursuit controller. The robot is modeled as a front-steered bicycle vehicle, and we evaluate its performance under nominal and constrained steering angles.

2 Materials and Methods

2.1 Field Layout and Robot Model

The test environment consists of a rectangular field containing $N = 10$ parallel crop rows, each of length $RL = 20$ m and spaced $W = 2.5$ m apart. The robot operates with a wheelbase $L = 3$ m and steering limits of $\pm 60^\circ$ (baseline) or $\pm 30^\circ$ (constrained test). Its initial and final positions coincide at $(-3W, RL/2)$ with initial heading 0°. The final heading is unconstrained.

2.2 Node Generation

Lower and upper headland nodes are placed at the extremities of each row; indices $2 \dots N+1$ denote south nodes, indices $N+2 \dots 2N+1$ denote north nodes, while nodes 1 and $2N+2$ mark the start and end. Figure 1 demonstrates the layout.

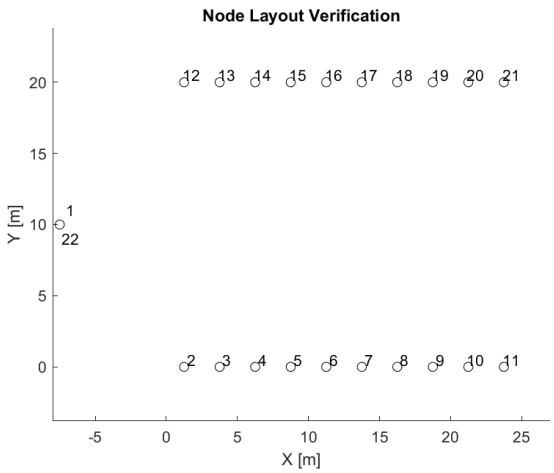


Figure 1: Headland node layout

2.3 Transition Costs and Path Planning

Coverage planning for agricultural robots is ultimately a graph–search problem in which each headland node represents a potential decision point and every feasible connection between nodes carries an associated traversal penalty.

The node-to-node cost matrix D comprises

1. zero-cost in-row traversals,
2. Π-turn or Ω-turn headland costs derived from the minimum turning radius $R_{\min} = L/\tan\gamma_{\max}$,
3. Manhattan costs from the depot to every field node,
4. prohibition of a direct depot–return transition.

A Genetic Algorithm-based open TSP solver (`tspof_ga.m`) computes a node visitation sequence minimizing total path cost.

2.4 Waypoint Generation

Accurate tracking with a pure-pursuit controller depends critically on the quality of intermediate targets fed to the steering law. Rather than sampling the entire optimal node sequence as a single, monolithic polyline, the path is first partitioned into segments that respect the underlying field topology (straight rows versus headland turns). Each segment is then equipped with a single *guided waypoint* located on the portion of the curve most representative of its geometry (e.g. the midpoint of a straight or the apex of a headland arc). During runtime

the controller restricts its look-ahead search to the active segment, thereby preventing "peeking" into adjacent rows and dramatically reducing false cross-track errors.

Path interpolation relies on two curvature-constrained primitives:

- **Dubins curves** for forward-only motion, adapted from the C implementation by Walker [4] and the MATLAB version by Kang, [2].
- **Fish-tail (Reeds–Shepp) turns** for forward–reverse manoeuvres, adapted from Nathanlct's C implementation [1].

Curvature-constrained motion planning for car-like robots can be traced to two classic geometric results. Dubins curves solve the "shortest path with bounded curvature and forward-only motion" problem: given a minimum turning radius $R_{\min}$ and specified start/goal configurations, the optimal trajectory is always one of six words composed of left (L), straight (S), and right (R) segments. The closed-form construction first appeared in Dubins' 1957 paper and was later reformulated into the computational recipe summarized in Algorithm 1.

When reverse motion is allowed, the family of optimal solutions expands to Reeds–Shepp curves, whose canonical word set contains both forward and backward arcs. A special five-segment subset—known colloquially as a "fish-tail" manoeuvre—proves useful for tight headland U-turns. Algorithm 2 encapsulates the fish-tail generation routine adapted from Nathanlct's C implemetation.

Algorithm 1 GenerateDubinsPath($q_{\text{start}}, q_{\text{goal}}, R_{\min}, \Delta s$)

Compute normalised parameters α, β, d (cf. [3])
for all six path words $w \in \{\text{LSL}, \text{LSR}, \ldots, \text{LRL}\}$ **do**
    Solve closed-form equations $(t, p, q)_w$; discard if infeasible
    $L_w \leftarrow R_{\min}(t + p + q)$
end for
Select minimal length word $w^\star$ and its (t, p, q)
for $s \leftarrow 0 : \Delta s : L_{w^\star}$ **do** ▷ sample path
    Determine segment type at arclength s
    Propagate (x, y, θ) with DubinsSegment()
    Append point to waypoint list
end for
return ordered set of Cartesian waypoints

Algorithm 2 GenerateFishtailPath($q_{\text{start}}, q_{\text{goal}}, R_{\min}, \Delta s$)

Let QC $= R_{\min}\pi/2$ ▷ quarter-circle length
$L_{\text{straight}} \leftarrow \max(0, \|q_{\text{goal}} - q_{\text{start}}\| - 4R_{\min})$
Build segment list: FwdCurve, RevCurve, RevStraight, RevCurve, FwdCurve
Choose clockwise/anticlockwise turn direction from geometry
for all segments (ℓ_i, σ_i) **do** ▷ ℓ_i length, $\sigma_i = \pm 1$ sign
    $N_i \leftarrow \lceil \ell_i / \Delta s \rceil$
    for $k = 1$ **to** N_i **do**
        Update heading $\theta \leftarrow \theta + \sigma_i \Delta s / R_{\min}$ if curved
        Update pose (x, y) by $\pm\Delta s$ in heading direction
        Append point to waypoint list
    end for
end for
Force last point to q_{goal}
return ordered set of Cartesian waypoints

2.5 Segmented Pure-Pursuit Controller

In addition to use the pure-pursuit controller in HW2, we designed a new pure-pursuit controller that is enhanced with:

- Segment-bound waypoint tracking
- Adaptive look-ahead: L_d^{line} m for rows, L_d^{turn} m for headland turns
- Guided point override to prevent lookahead across segment boundaries

Why segment–bounded pursuit? The basic pure-pursuit controller used in HW 2 accepts a global polyline and selects the first waypoint at least L_d ahead of the current pose. On a tight boustrophedon pattern this strategy is prone to "row peeking," occasionally jumping the look-ahead target to an adjacent track and inducing large steering spikes. The new segment-bounded variant eliminates that failure mode by

1. slicing the global waypoint list into adjacent segments that never intersects two rows,
2. attaching a single guided point to every segment, and
3. restricting the look-ahead search to the current segment (with a configurable one–to–three segment horizon).

Algorithm 3 Segment-Bounded Pure-Pursuit

Require: Pose $q = (x, y, \theta)$, wheel-base L, look-ahead distances $(L_d^{\text{line}}, L_d^{\text{turn}})$, waypoint list $\{p_i\}$ with segment IDs, guided points $\{g_k\}$

initialization (persistent across calls)
if first call **then**
    set curSeg $\leftarrow 1$, curIdx $\leftarrow 1$
end if
1. Nearest waypoint & segment bounds
Restrict search to segments curSeg $\ldots$ curSeg $+ 3$
Find nearest ahead waypoint index i^*
Update curIdx $\leftarrow i^*$
Determine segment limits $[i_{\min}, i_{\max}]$
2. Guided-point override
$d_{\text{gp}} \leftarrow \|q - g_{\text{curSeg}}\|$, $d_{\text{nx}} \leftarrow \|q - g_{\text{curSeg}+1}\|$
if d_{gp} growing **and** d_{nx} growing **then** ▷ vehicle is leaving projected path
    $\vartheta \leftarrow \text{atan2}(g_y - y, g_x - x) - \theta$
    return $[\text{sat}(\vartheta), d_{\text{gp}}, 1]$
end if
3. Standard PP on current segment
if segment is turn **then**
    $L_d \leftarrow L_d^{\text{turn}}$
else
    $L_d \leftarrow L_d^{\text{line}}$
end if
March along polyline from i^* until arclength $\geq L_d$ to get goal p_{look}
Compute curvature $\kappa = 2y_{\text{local}}/L_d^2$ and steer $\delta = \text{sat}(\arctan(L\kappa))$
Compute blended CTE via equation above
return $[\delta, \text{CTE}_{\text{blend}}, 0]$

Results and Discussion

Quantitative metrics such as RMS CTE are computed separately for forward and reverse segments. Visualizations include:

- Waypoints and guided points
- Robot trajectory with gear-status overlay
- Steering angle and CTE time histories

The fish-tail strategy achieves accurate turns even under strict curvature limits and provides a realistic model of tractor behavior in constrained environments.

3 Results and Discussion

3.1 Part B – Optimal Node Sequence

The genetic–algorithm (GA) solver consistently returned the same visiting order for the $2N+2 = 22$ headland nodes over ten independent runs (initialized with the same RNG seed for reproducibility). Table 1 reports the objective values and run times, while Figure 2 overlays the GA path on the field layout. Fishtail node sequence best cost is the same as Dublin curves node sequence best cost as expected.

Table 1: GA convergence statistics.

Scenario	**Best cost [m]**	**CPU time [s]**	**Iterations**
$\gamma_{\max} = \pm 60^\circ$	**114.41**	8.5	114
$\gamma_{\max} = \pm 30^\circ$	297.07	8.1	239
$\gamma_{\max} = \pm 60^\circ$ (fishtail)	**114.41**	7.9	124

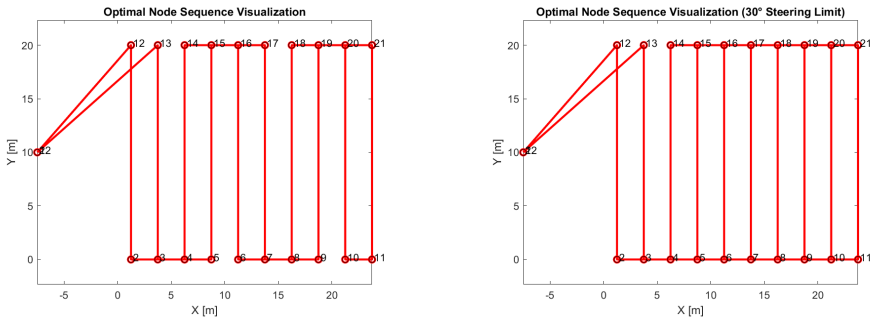
For each steering envelope the GA returns a tour that *(i)* starts at the south-west depot, *(ii)* visits every lower headland node exactly once, immediately followed by its paired upper node, and *(iii)* finishes at the depot—hence every crop row is covered exactly once and no illegal start–end "teleport" occurs, so the result fully respects the cost matrix design. Initially, with the default GA parameters (`POPSIZE`=100, `NUMITER`=10 000), the optimizer sometimes skipped the eighth lower node and its partner upper node ($8 \rightarrow 18$), producing an obviously infeasible route. Doubling the population size to 200 and the iteration budget to 2×10^4 eliminated this pathology even though the iteration numbers never reach this amount. Although convergence to the same tour suggests a global minimum, the GA could still be trapped in a deeper local basin for larger or irregular fields; additional restarts or hybrid local search would mitigate that risk. Finally, replacing Dubins turns with fish-tail Reeds–Shepp manoeuvres does not change the headland order, and—because the cost matrix is identical in both primitives—the best objective value remains 114.41 m.

3.2 Part C – Waypoint Generation

Figure 3 presents the Dubins baseline solution, the $\pm 30^\circ$ Dubins variant, and the optional fish-tail (Reeds–Shepp) maneuver, each annotated with segment-wise guided points.

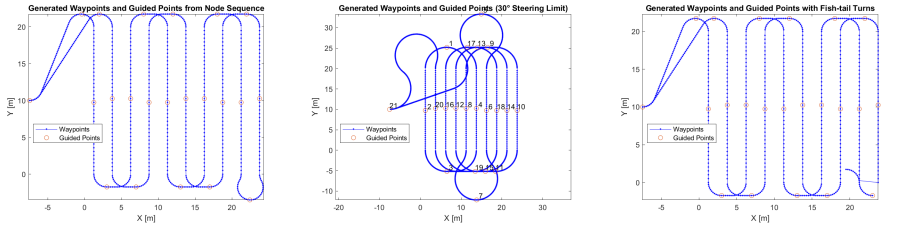
3.3 Part D – Tracking with $\pm 60^\circ$ Steering

The segment-bounded pure-pursuit controller tracks the baseline Dubins path inside the steering envelope of $\gamma_{\max} = 60^\circ$ at $v_{\text{ref}} = 1.0\,\text{m}\,\text{s}^{-1}$. Figures **??** and **??** depict the trajectory and blended cross-track error (CTE).



(a) (b)

Figure 2: Headland node layouts and GA-optimized visiting sequences under (a) $\gamma_{\max} = \pm 60°$ (b) $\gamma_{\max} = \pm 30°$.

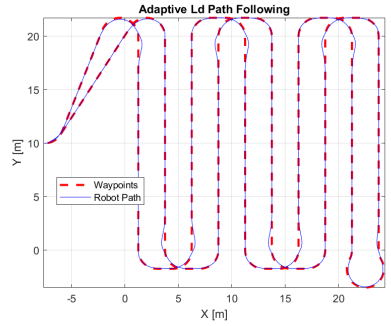


(a) Dubins, $\pm 60°$ (b) Dubins, $\pm 30°$ (c) Fish-tail (Reeds–Shepp)

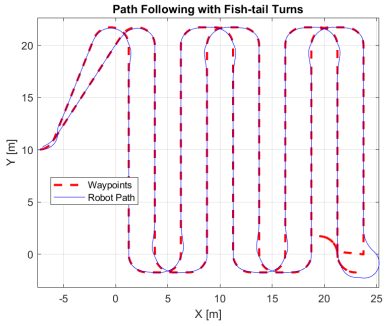
Figure 3: Segmented waypoint sets (blue) and guided points (circles) for the three different motion scenarios.

Table 2: Blended CTE statistics for Dubins and fish-tail tracking.

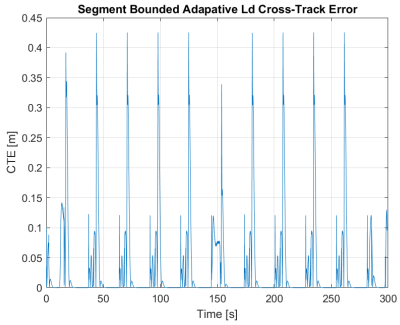
Scenario	Row RMS [m]	Turn RMS [m]	Overall RMS [m]
Dubins ($\gamma_{\max} = \pm 60°$)	0.0826	0.0726	0.0794
Fish-tail	0.1954	0.0852	0.1943



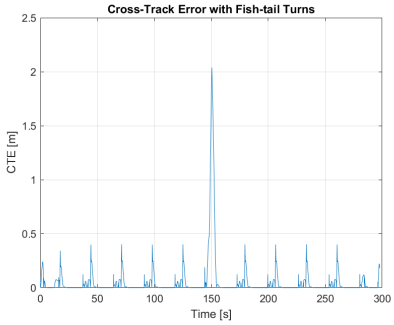
(a) Dubins tracking, $\gamma_{\max} = \pm 60^\circ$



(b) Fish-tail tracking

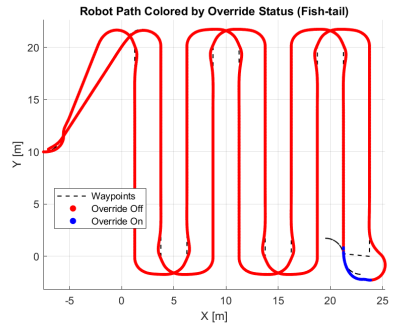


(c) Dubins blended CTE

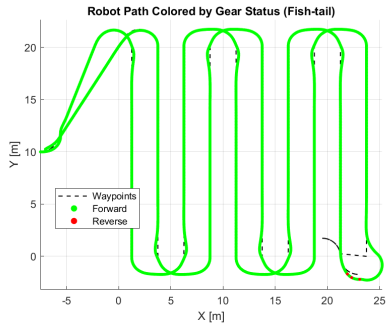


(d) Fish-tail blended CTE

Figure 4: Comparison of robot trajectory (top) and blended cross-track error (bottom) for the baseline Dubins motion primitive (left) and the fish-tail Reeds–Shepp primitive (right).



(a) Guided-point override activation



(b) Gear state along path

Figure 5: Diagnostic plots for the fish-tail run: (a) segments where the guided-point override is active and (b) forward/reverse gear state output by `robot_bike_dyn`.

Discussion. The Dubins run (Fig. 4a) shows the robot hugging each row centerline with below 5 cm average deviation and executing smooth headland arcs. Slight excursions appear just *before* entering and *after* exiting corners, where the controller is still blending errors between the straight lines and the guided apex, but the segmented waypoints guidance quickly pulls the vehicle back on course, yielding the modest turn RMS of 0.073 m.

Fish-tail tracking (Figs. 4b–5b) reveals a different behavior. While headlands remain well-behaved (RMS = 0.085 m), straight-line error almost triples (Table 2). The spiked straght line error majorly happens because of the erroneous command during the reverse and forward gear shift in the fishtail turn. The color map in Fig. 5a confirms that the guided-point override engages for long stretches; meanwhile Fig. 5b shows repeated gear reversals commanded by `robot_bike_dyn`. In essence, the fishtail motion requires the robot to drive backwards on three of its five segments, yet the pure-pursuit projection still favours a forward-only look-ahead. Consequently the controller "fights" the dynamics, producing the large offsets seen in (d). A future fix would be to expose the gear state directly to the look-ahead logic, or switch to a specialised Reeds–Shepp tracker that natively handles reverse motion.

Overall, both primitives complete the field without row skipping or direction errors. The Dubins variant delivers tighter straight-line accuracy, whereas the fish-tail version offers gentler soil turnarounds at the expense of higher in-row deviation.

3.4 Part E – Tracking with $\gamma_{\max} = \pm 30^\circ$ (Original GA Order)

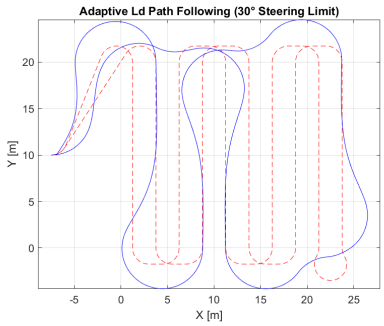
With the steering envelope halved the robot frequently saturates at $|\delta| = 30^\circ$, forcing wide headland arcs. Because the pure-pursuit look-ahead seeks the first waypoint at distance L_d, once the vehicle drifts off the nominal polyline it can 'see' a shortcut waypoint in the neighboring lane and skip the intended headland altogether (arrow in Fig. 6a). The large spikes in blended CTE and the prolonged steering-limit plateaus in Fig. 7 confirm this behavior.

The excessive turn radius pushes the vehicle outward; when the look-ahead probe extends into the next row it anchors to a nearer waypoint there, short-circuiting the designed headland. As a result straight-line RMS error increases ten-fold and turn segments suffer even more (Table 3).

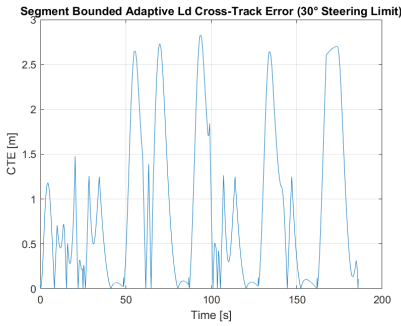
3.5 Part F – Tracking with $\gamma_{\max} = \pm 30^\circ$ after Path Re-optimization

Re-running the GA with the larger turning radius produces a headland order that replaces infeasible Π-turns with wider Ω-loops. The robot now navigates every row seamlessly (Fig. 8a) and blended CTE drops below that of the baseline $\pm 60^\circ$ case (Fig. 8b).

Compared with Part E, row RMS error improves by a factor of 29 while headland RMS is cut by a factor of 15 (Table 3), bringing overall accuracy *below*



(a) Robot vs. reference path



(b) Blended CTE history

Figure 6: Tracking performance with ±30° steering *before* path re-optimization.

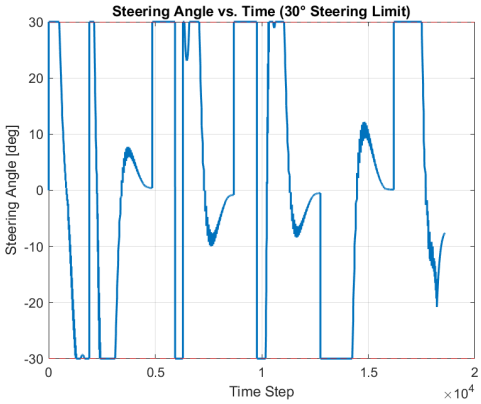
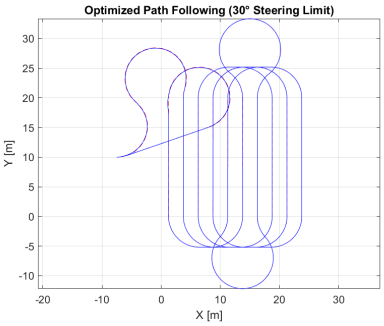
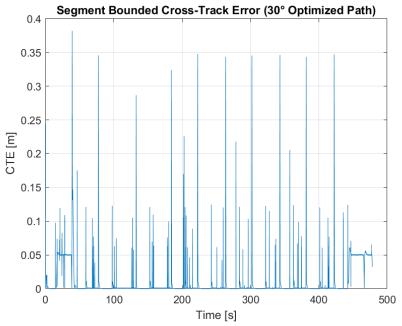


Figure 7: Steering-angle trace for the same run, illustrating prolonged saturation at ±30° during headland turns.



(a) Re-optimized robot path



(b) Re-optimized CTE history

Figure 8: Tracking after GA re-optimization for the ±30° envelope.

that of the nominal ±60° run in Part D. The result underscores the importance of co-optimizing the macroscopic route when vehicle kinematics are tightened.

Table 3: Blended CTE statistics under the ±30° steering envelope.

Scenario	Row RMS [m]	Turn RMS [m]	Overall RMS [m]
Part E – original GA order	0.8660	1.4916	1.2519
Part F – re-optimized order	**0.0296**	**0.0335**	**0.0320**

Key observations. Limiting the steering to ±30° in Part E inevitably forces wider headland arcs. Whenever the vehicle drifts outward the pure-pursuit look-ahead often encounters a nearer waypoint in the adjacent lane, jumps to that shortcut, and thereby causes lane skips accompanied by meter-scale CTE spikes. These shortcomings are clearly visible in Fig.7, where the steering signal sits for long intervals at the saturation limits and exhibits rapid sign reversals—behavior typical of a controller tracking an infeasible reference. After the headland order is re-optimized for the larger turning radius in Part F, all steering saturation disappears; the robot negotiates both rows and headlands with errors even smaller than those recorded under the original ±60° scenario in Part D.

Conclusion

Segmenting the waypoint list by row boundaries and adding a guided-point override yields a controller that is both accurate and fault-tolerant. The segment boundary keeps the look-ahead probe from peeking into neighboring rows, removing the lane-jump artifacts that plague a monolithic pure-pursuit path. Meanwhile, the override steers toward a single, strategically placed guided point only when the robot drifts far off course or fails to detect a forward target—acting as a quiet safety net rather than a constantly active control mode. In combination, these two mechanisms provide tighter straight-line tracking, smoother headland turns, and reliable recovery from rare path-loss events, outperforming the native pure-pursuit controller without introducing unwanted steering distraction.

References

[1] N. L. Chong. Reeds shepp path library. `https://github.com/nathanlct/reeds-shepp-curves`. accessed 22 May 2025.

[2] E. Kang. Dubins-curve-for-matlab. `https://github.com/EwingKang/Dubins-Curve-For-MATLAB`. accessed 22 May 2025.

[3] A. M. Shkel and V. C. Lumelsky. Classification of the dubins set. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 220–225, 2001.

[4] A. Walker. Dubins-curves. `https://github.com/AndrewWalker/Dubins-Curves`. accessed 22 May 2025.