

ELEN90095 — AI For Robotics — Sim-2-Real Part B

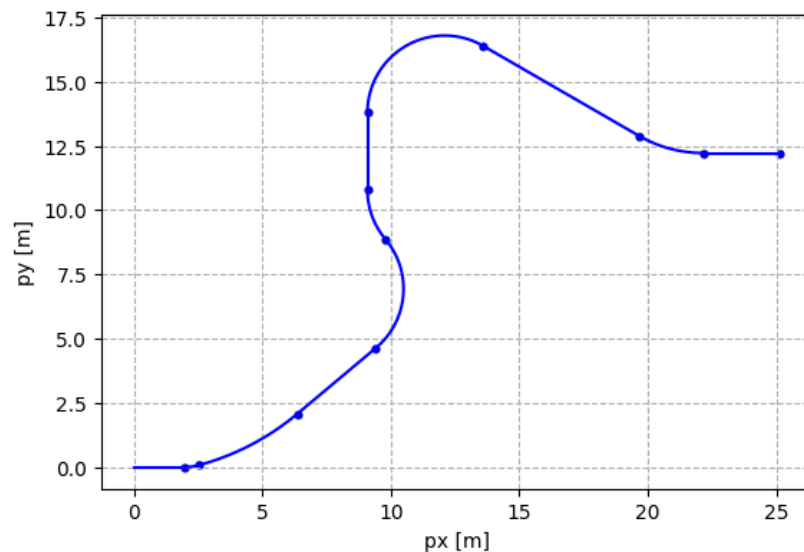
Tianchen Lou – 1078368 – tianchenl@student.unimelb.edu.au

Arya Araban – 1439683 – aaraban@student.unimelb.edu.au

B2:

- **Length between front and rear axles (L):** 0.335 m. obtained directly from the published specifications sheet for the 1/10 scale Traxxas radio-controlled car.
- **Maximum steering angle:** 25 degrees. Arrived at by turning the steering wheel of the 1/10 car to its limits and measuring the maximum angle.
- **Maximum velocity:** 30 mph (13.4 m/s). obtained from the official specifications sheet published by Traxxas. While the exact maximum speed may vary slightly, this top speed represents a reasonable estimate of the maximum velocity this model can achieve.
- **Maximum rate-of-change of steering angle:** 45 degrees. arrived at by turning the steering wheel back and forth as fast as possible and measuring the maximum angular rate.
- **Maximum rate-of-change of velocity:** 5.9 m/s². Estimated based on acceleration testing data published for the 12T 550 motor. The motor test showed an acceleration from 0 to 60 ft/s (18.3 m/s) achieved in 3.09 seconds. This equates to a rate-of-change of velocity of 18.3 m/s / 3.09 s = 5.9 m/s².

The constructed road for the simulations appears as follows:



The road was made to include left and right turns, as well as a sharper corner to test the adaptiveness of future control policies.

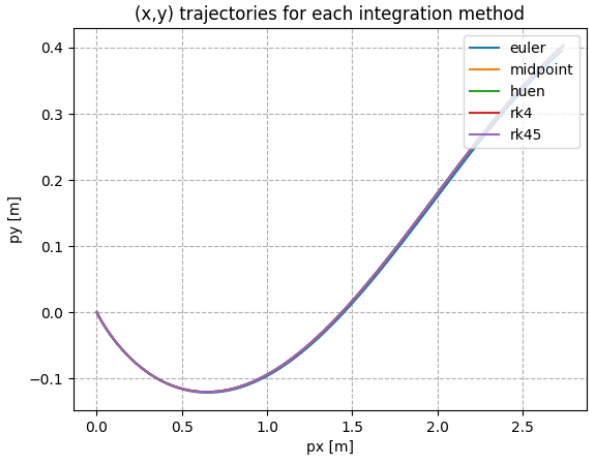
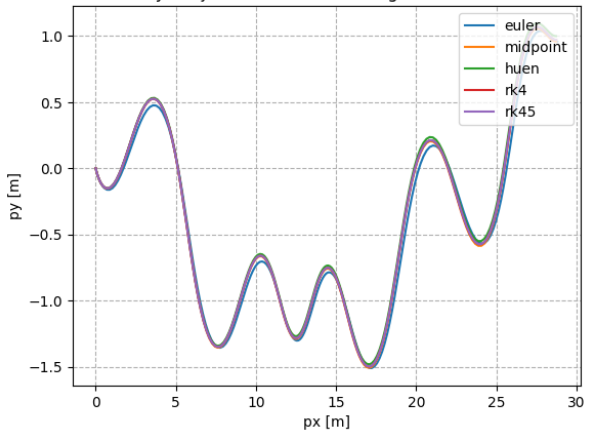
B3:

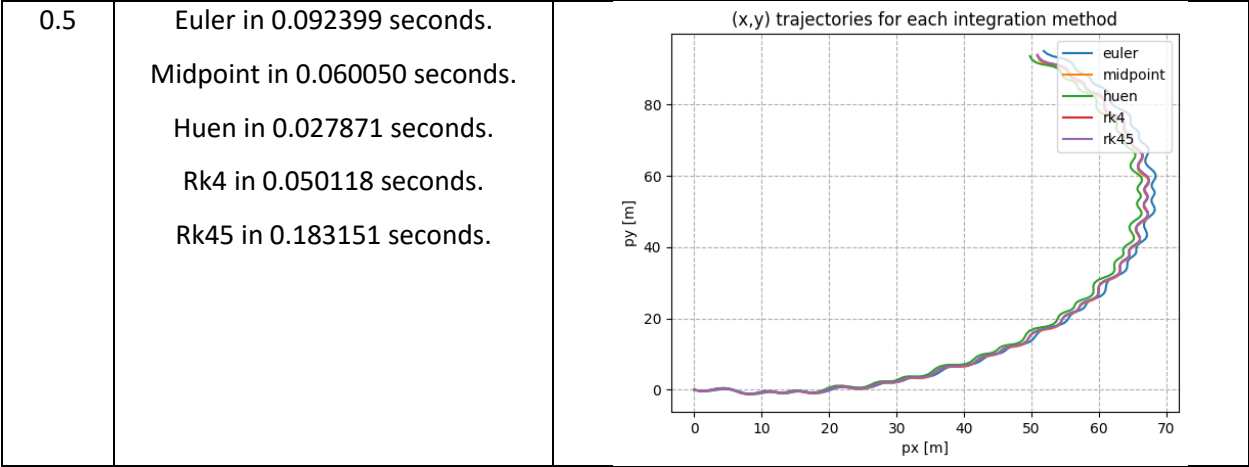
To analyze the accuracy and performance of different integration methods, we set up simulations using a fixed open-loop sequence of actions over time. The key parameters that were varied were:

Integration method: Euler, Midpoint, Heun's, RK4, RK45

Integration timestep (T_{s_sim}): 0.01 sec, 0.1 sec, 0.5 sec

The Integration timestep can be used to determine the “aggressiveness” of the path, allowing better comparisons of integration methods. By keeping the simulation duration fixed over the same number of steps ($N_{sim} = 200$), a larger timestep stresses integration methods more, better characterizing stability and accuracy under different conditions despite higher computation time.

| T_{s_sim} | Computational Time | (x,y) Trajectory Plot |
|--------------|---|--|
| 0.01 | Euler in 0.016002 seconds. Midpoint in 0.014844 seconds. Huen in 0.013999 seconds. Rk4 in 0.016288 seconds. Rk45 in 0.066322 seconds. |  <p>(x,y) trajectories for each integration method</p> <p>The plot shows the trajectories of five integration methods (euler, midpoint, huen, rk4, rk45) for a simulation with a timestep of 0.01 seconds. The x-axis is labeled 'px [m]' and ranges from 0.0 to 2.5. The y-axis is labeled 'py [m]' and ranges from -0.1 to 0.4. The trajectories are very close to each other, forming a smooth curve that starts at (0,0), dips to a minimum around x=0.7, and then rises to approximately (2.2, 0.25). The legend indicates the methods: euler (blue), midpoint (orange), huen (green), rk4 (red), and rk45 (purple).</p> |
| 0.1 | Euler in 0.011096 seconds. Midpoint in 0.012839 seconds. Huen in 0.023084 seconds. Rk4 in 0.024897 seconds. Rk45 in 0.109497 seconds. |  <p>(x,y) trajectories for each integration method</p> <p>The plot shows the trajectories of five integration methods (euler, midpoint, huen, rk4, rk45) for a simulation with a timestep of 0.1 seconds. The x-axis is labeled 'px [m]' and ranges from 0 to 30. The y-axis is labeled 'py [m]' and ranges from -1.5 to 1.0. The trajectories are more spread out than in the 0.01 second plot, showing a complex, oscillatory path. The legend indicates the methods: euler (blue), midpoint (orange), huen (green), rk4 (red), and rk45 (purple).</p> |



With a timestep size of 0.01 seconds, all methods yielded virtually indistinguishable trajectories, indicating their high accuracy at this timestep. However, compared to simpler techniques such as Euler and Midpoint, RK45 exhibited slightly slower performance. As the timestep increased to 0.1 seconds, small discrepancies emerged among the methods. Notably, Euler integration began to display slight deviations from the other solutions, suggesting a loss of accuracy with this longer timestep. When using a larger timestep of 0.5 seconds, the limitations of Euler and Huen became significantly apparent. They exhibited substantial trajectory errors and deviated from the intended path, while both RK4 and RK45 continued to generate smooth and accurate solutions.

Based on the analysis, a timestep of 0.1 seconds emerges as the optimal choice, striking a balance between accuracy and computational efficiency. At this size, the errors remain controlled, and the computational cost of using the RK45 integration method is reasonably fast. Crucially, the 0.1-second timestep harnesses the strengths of RK45, which can maintain stability and accuracy even with larger timesteps. In contrast, other methods like Euler would require significantly smaller timesteps to preserve accuracy.

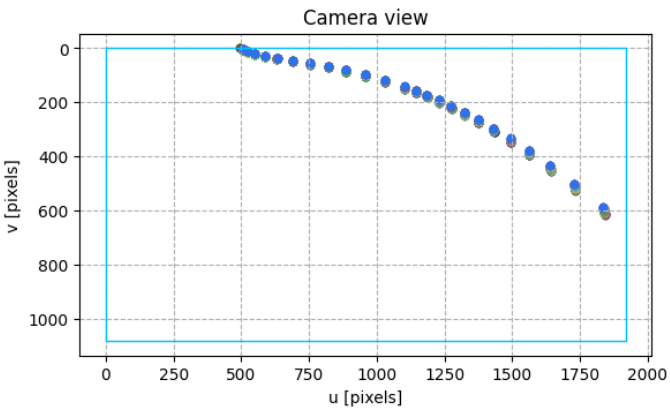
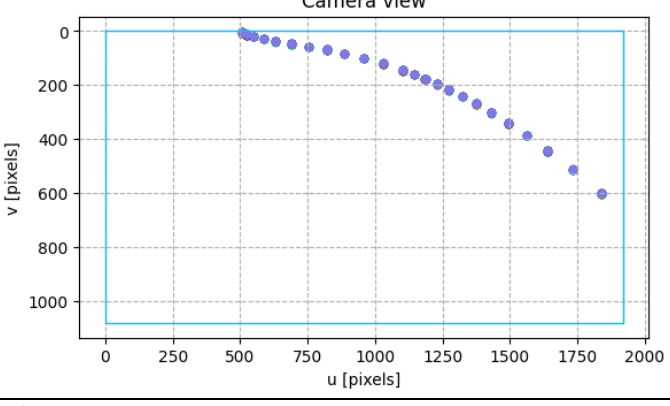
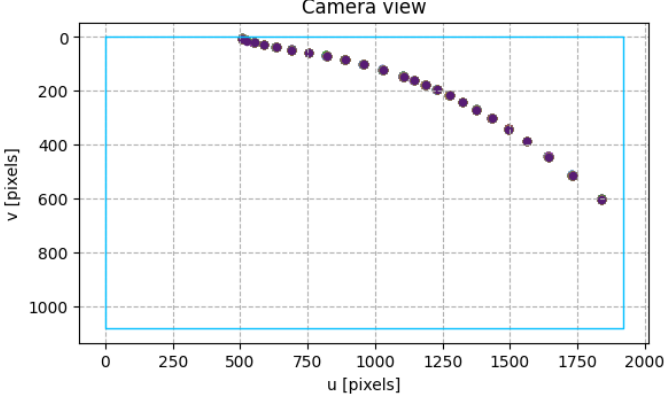
Therefore, due to RK45's superior precision qualities, it was selected as the overall integration method, despite having higher computational overhead than simpler techniques. And the integration timestep was set to 0.1 seconds to best exploit RK45's capabilities while retaining efficiency.

B4:

Both extrinsic and intrinsic parameters are essential for the simulation of measurements. They are tabulated below with discussion on their expected error margins. Note that all error margins are expected to follow a Gaussian distribution.

| Name | Value and range | Discussion |
|-----------------------------|--|---|
| Camera origin to body frame | -32cm down, -11cm back ± 0.5 cm | The value was measured with a ruler. A higher error margin was selected due to the rough measurement, and to account for changes in car height due to its large suspension. |
| Angle of camera | 25 degrees ± 2 degrees | The value was calculated roughly by measuring the mounting of the camera and using trigonometry. The error range was chosen due to the uncertainty of the measurement. |
| Camera resolution | 1920x1080p | The high resolution value was selected in the simulation as the team wanted to test how the car would operate given the most accurate data. |
| Intrinsic camera values | Fx: 1423.2 ± 0.80 Fy: 1419.5 ± 0.86 Cx: 950.5 ± 1.04 Cy: 518.6 ± 1.84 | The values were taken as the median value of the given list of calibrated cameras. The range was selected to be 2 times the given standard deviation. |

A for loop was developed to run in the gymnasium to test 100 different values within the error range of the given values to understand how it would affect the camera view. They are displayed below.

| Name | Value and range | |
|-----------------------------|--|--|
| Camera origin to body frame | -32cm down, -11cm back $\pm 0.5\text{cm}$ |  |
| Angle of camera | 25 degrees ± 2 degrees |  |
| Camera resolution | 1920x1080p | N/A as there is no error margin |
| Intrinsic camera values | Fx: 1423.2 ± 0.80 Fy: 1419.5 ± 0.86 Cx: 950.5 ± 1.04 Cy: 518.6 ± 1.84 |  |

Overall, it is clear that no parameter causes significant changes or deviations in their error ranges. For the 4 parameters of the intrinsic camera values, they were tested separately but had close to zero differences in the output view as their error margins were so small, therefore their results were collated into the one view above.

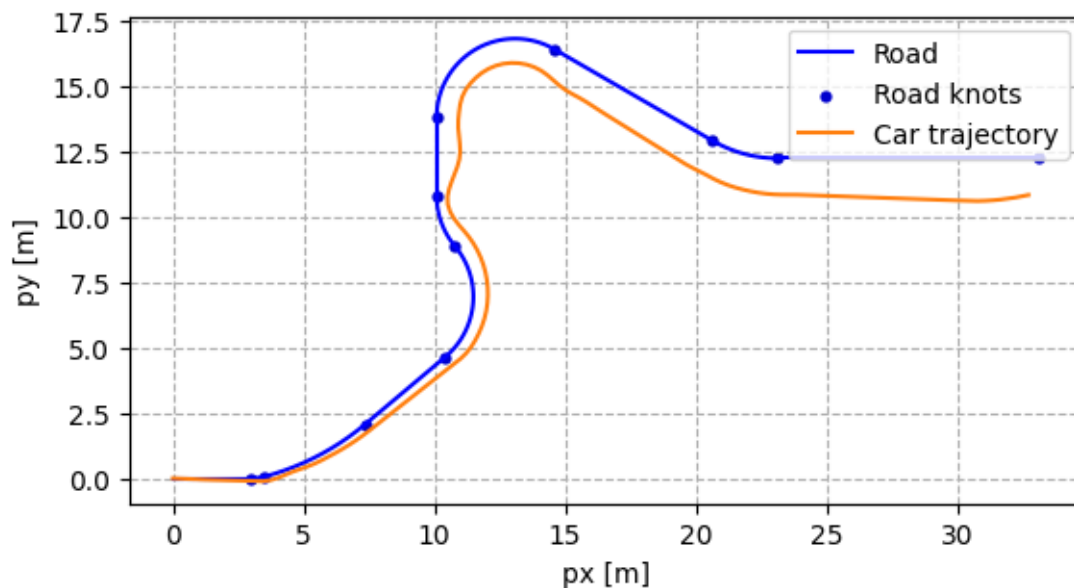
B5:

Rule Based Policy:

We developed a rule-based line following controller that utilizes logical rules and a constant steering angle offset parameter (set to $-\pi/45$ radians) to determine the steering angle of the robot. The controller uses the "road_angle_at_closest_p" measurement obtained from line detection, which represents the tangent angle of the road at the closest point captured by the camera image.

The steering action is determined by subtracting the constant angle from the road angle to obtain the target angle. The difference between the target angle and the current direction of the robot is then computed to decide the steering action. The constant angle is used to make the robot slightly exceed the tangent of the road, ensuring it stays in the center and avoids favoring one side over the other. Through simulation, we fine-tuned the value of this constant angle to maintain vehicle stability on the road. Smaller values caused oscillations, while larger values caused the robot to deviate from the desired path. By finding the right offset, we achieved a balance that allowed the robot to navigate the road smoothly and effectively.

We evaluated the performance of our rule-based controller using a metric that computed the total deviation of the trajectory from the road centerline by taking the distance from each (x,y) point to the nearest road coordinate. In simulation, our rule-based controller achieved a total deviation of 69.33 over the run, indicating that it was able to follow the road with some drifting from the centerline. It also finished the course in 177 time steps. Together, the 2 performance metrics offers insight into how closely the car can follow the road, and how fast it is in completing the course.

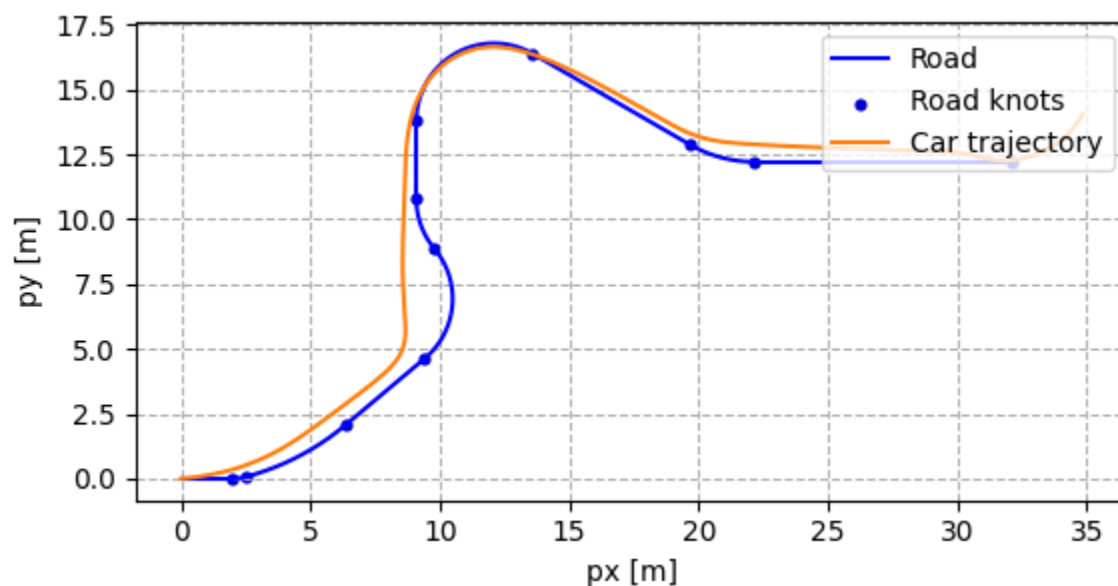


Pure pursuit policy:

A pure pursuit controller was also developed which resembles a proportional controller, but with the added change of looking ahead for targets in the distance, rather than immediate points on the road. The controller uses the “uv_coords” variable which lists points on the road within view of the camera. An additional “look_ahead” variable is added to adjust how far into the distance the robot will attempt to target.

The steering action is programmed by considering the pixel coordinates of the target point on the road in the frame of the camera. Given that the resolution is 1920x1080p from section B4, simple calculations on the pixel distances can give the necessary delta the robot needs to take in order to reach the target. Through iterative simulations, the look ahead distance and the sharpness of the robot’s turning was adjusted for best performance.

The figure below displays the output of the robot following the road. It achieved a performance value of 61.3012 in 154 time steps. Evidently this policy is able to prioritise speed, as looking ahead to a future point instead of immediate points means shortcuts will be taken by the robot. This sacrifices accuracy in line following, but improves the speed at which the robot can finish the course.



B6:

Transferring the policies from the simulation to the actual robot was a difficult experience due to the nature of robotics. The team faced many challenges with the camera, and it was difficult to debug whether the issue was occurring due to a camera disconnect, a policy error, or a ROS node being turned off. A key issue that was found was the difference in variable names. In the simulation, key information was stored in “uv_coords” and “info_dict”, but uv_coords was named differently in the robot code, and info_dict was not able to be found. This meant that the team was only able to test the default policy.

With external help in debugging the connections, the car managed to run although it did not follow the line very well. Pictures attached below show the car in motion on the line before it crashed and had to be picked up. Unfortunately after testing, the camera connection had issues once again and the car was not able to be run again either.

