Evaluating performance of encoders of Robotic System for use in localization applications using odometry

Amey Rawool Shivang Gangadia

December 14, 2022

Abstract- For intelligent systems to navigate a complex environment, remembering the path allows the robot to optimize its movements. This study focuses on evaluating the performance of wheel encoders of Pololu 3Pi+ to follow the path determined by way-points. We propose a direction-determining method and odometry to process the path memorized by the robot earlier and replicate it. We have evaluated its performance by mapping the accuracy of motion along various path configurations. We found that system is able to follow the straight line path more accurately as compared to a curved path with its accuracy increasing with an increase in way-points collection frequency.

1 Introduction

Line following systems is essentially the equivalent of "hello world" programs in computer science for robotics. However, such systems find much use in industry today as their simplicity, efficiency, and cost of operation are unparalleled even by the most advanced of computer vision systems. In cases of automated factory floors and warehouses, line following systems are an indispensable resource as they offer fast and precise navigation for robots throughout the factory, just as trains do across the country. But, what happens when the lines themselves are compromised? The robots need to have some sort of a backup plan that allows them to navigate the designated areas in case of broken or completely absent guiding lines, at least as a temporary measure. One solution could be memorizing the line using way-points. However, to map these way-points such that they capture the route of lines well enough and the intended obstacles are avoided, some localization technique is required. Odometry could fill in this gap. Most robots with wheels have encoders attached to them which could be used to calculate their position relative to a starting point. This begs the question "how reliable are these encoders for reconstructing a path using odometry compared to the original line-following approach ?". Our research attempts to answer this very question by designing and conducting experiments to quantify this reliability. The limitations of such systems are processing power, real-time processing, and measurement accuracy [2]. This limitation could be resolved by limiting the components creating a compact robot and removing the need to develop expensive systems with a wide array of sensors hence reducing the processing time. It allows for keeping the cost of the robot systems low. We use Pololu 3pi+ as our robot platform as it only uses in-built line sensors and wheel encoders to perform all necessary computation. It used line sensors to record the path they have traversed earlier so that they have awareness of the environment to make the necessary decision based on the tasks assigned to them. The system then uses wheel encoders to count the number of turns taken by each wheel and the order of such movements in indexed arrays and implements odometry to predict the position of the robot across the run time [1].

1.1 Experiment Objectives

The research aims to evaluate the performance of encoders and investigate the sources of error in using Pololu 3pi+ robots for localization applications using odometry.

2 Structure

Section 4 provides details on initial experiments done to characterize our robot. This helps point out and remove errors not associated with Odometry, but are a result of the hardware inaccuracies and the environment. These can be assumed to stay independent of the path selected in further experiments and hence can be compensated for in the software itself; Section 5 presents information about the algorithm and equations used to conduct the experiment and provide a means to replicate results; the methods used to conduct the experiment and collect results are explained in Section 6; while the collected results are explained in Section 7 and the insights derived with further research are elaborated in Section 8.

3 Hypothesis

We hypothesize:

1. The robot will be able to track coordinates of a path(way-points) recorded on a previous run if its encoders generate similar values over a successive run, thus establishing consistent and reliable operation.
2. The odometry performance will be comparatively worst on sharp turns, better on curves, and best on straight lines.
3. Higher frequencies of way-point recording will allow the robot to follow the intended path closely, but will have no effect on the endpoint.

4 Characterization Study

A thorough examination of the wheel encoders and applied algorithms is done to determine the accuracy and calibrate the system for the optimized performance. A total of 10 readings for speed + 7 readings for distance + 6 readings for orientation repeated × 3 = 69 readings were taken for this study.

4.1 Errors based on wheel speed

The robot was made to run for a fixed distance with varying speeds(ω_{target}). As seen in figure 1 there is an increase in error with speed more than and less than 2.5 RPS. The error at higher speeds could be because the inertia of the robot makes it slip when suddenly stopping at the end. The error at lower speeds could be presumably because the encoder is unable to pick all pulses from the wheels due to low resolution. Hence, the speed, ω_{target}, is decided to be 2.5 rotations per second.

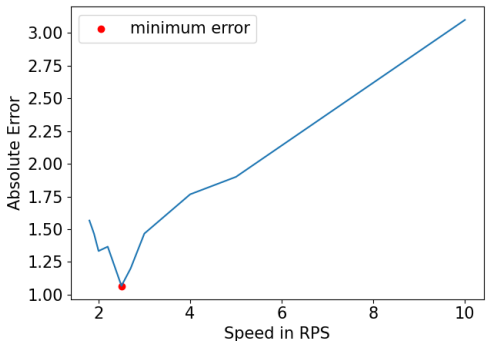


Figure 1: Error with change in speed

4.2 Errors based on distance

The robot was made to run different distances at a fixed speed. Figure 2 shows a proportional increase in absolute error with distance. Thus the slope and intercept of the line were calculated using linear regression to predict error and adjust the target distance accordingly.

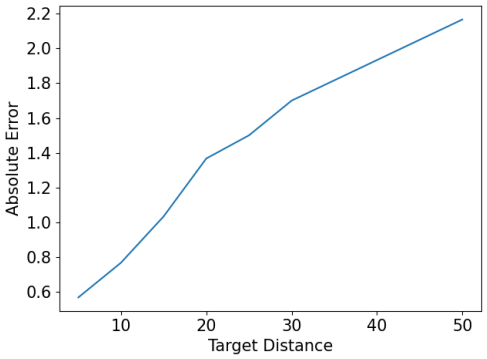


Figure 2: Error with change in distance

4.3 Errors in rotation

The robot was given various target orientation values without translational speed and it corrected its orientation using only odometry. As seen in figure 3, The error seemed consistent and low for a variety of angles tested, with the maximum error being around 1.33°. This error was used to tune the correction in Orientation by varying PID constants.

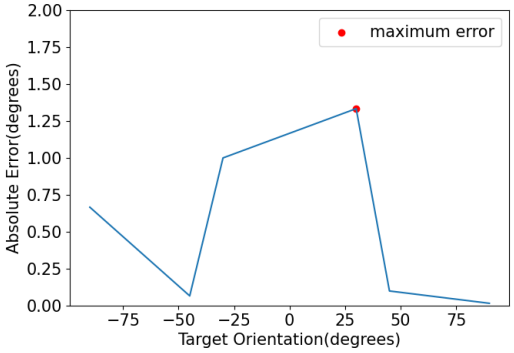


Figure 3: Error at different orientations

5 Implementation

5.1 Translation and Rotation

The robot is equipped with 2 wheels, each with it's own rotary encoder. However, the encoders are not directly connected to the wheel shafts and require a gear ratio resolution before the counts per rotation can be obtained. The reasoning behind this design is detailed in the manufacturer's manual[4].

Let n be the number of counts recorded in each iteration of the program's main loop. The number of rotations for the current iteration of the main program's loop, R_n, is found as:

$$R_n = \frac{n}{358.3} \tag{1}$$

where the Counts per rotation = 358.3 is given in the manufacturer's manual [4]. The corresponding number of rotations for left and right wheel, thus calculated will be R_{n-L} and R_{n-R} respectively.
Let t_{loop} be the time taken per iteration of the main program's loop. The speed of the wheels was thus measured as $\omega_{measured-L}$ and $\omega_{measured-R}$ for left and right wheels respectively:

$$\omega_{measured-L} = \frac{R_{n-L}}{t_{loop}} \tag{2}$$

$$\omega_{measured-R} = \frac{R_{n-R}}{t_{loop}} \tag{3}$$

The difference in speeds of the left and right wheels, $\Delta\omega_{target}$, is used to make the robot rotate in place. This requires specification of the target speed of the robot: ω_{target}, which remains constant or 0 depending on whether the robot needs to move or stop.

$$\omega_{target-L} = \omega_{target} + \Delta\omega_{target} \tag{4}$$

$$\omega_{target-R} = \omega_{target} - \Delta\omega_{target} \tag{5}$$

Thus the translation and rotation of the robot can be controlled by varying ω_{target} and $\Delta\omega_{target}$.

5.2 Odometry

To calculate the distance moved, we will use the number of rotations from equation 1. The wheel's circumference was measured as $C_{wheel} = 32\pi$. Hence distance moved by each wheel, d_L and d_R, can be calculated as:

$$d_L = R_{n-L} \times C_{wheel} \tag{6}$$

$$d_R = R_{n-R} \times C_{wheel} \tag{7}$$

The instantaneous distance moved by the robot (δD_{robot}) as a whole, calculated per iteration of the main program's loop, would be an average of the distances moved by individual wheels

$$\delta D_{robot} = \frac{d_L + d_R}{2} \tag{8}$$

The difference in distances moved by the wheels, Δd, gives the distance the robot has rotated, along its circumference. Since our robot is circular, Δd can be used to calculate the angle by which the robot rotated. If C_{robot} is the circumference of the circular robot, instantaneous degrees moved $\theta_{measured}$ is calculated as:

$$\Delta d = d_L - d_R \tag{9}$$

$$\delta\theta_{measured} = \Delta d \times \frac{360}{C_{robot}} \tag{10}$$

The instantaneous orientation ($\delta\theta_{measured}$) and instantaneous distance traveled (δD_{robot}) are used to update the robot's current orientation and total distance traveled every iteration.

$$D_{robot} = D_{robot} + \delta D_{robot} \tag{11}$$

$$\theta_{current} = \theta_{current} + \delta\theta_{measured} \tag{12}$$

5.3 Line Following

The robot (Pololu 3pi+) uses Infra-Red sensors which work on the principle of differential reflectance of Infra-Red light for different colors. The details of its working can be found in the manufacturer's manual [4]. The amount of time for which IR light is reflected is recorded. This time is greater for darker colors since they absorb more IR light and hence reflect for longer. Let's call this time $\vec{\tau}$. Since there are 5 such IR sensors, $\vec{\tau} = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$. These timings are calculated at every iteration of the program's main loop and are calibrated and normalized to minimize variance among the sensors. The difference in these timings is used to estimate the robot's orientation with respect to the line. The metric that describes how much the robot is leaning towards the left or right is calculated as:

$$\lambda_L = \tau_1 + \tau_2 + 0.5 \times \tau_3 \tag{13}$$

$$\lambda_R = \tau_5 + \tau_4 + 0.5 \times \tau_3 \tag{14}$$

The 3rd sensor is the center sensor and hence adds to the weights of both sides. The final difference between λ_L and λ_R is directly used to influence $\Delta\omega_{target}$

$$\Delta\lambda = \lambda_L - \lambda_R \tag{15}$$

$$\Delta\omega_{target} = P \times \Delta\lambda \tag{16}$$

where P is the proportionality constant used to tune line-following aggressiveness.

5.4 Storing and following way-points

The calculated polar coordinates from equations 11 and 12 are stored periodically in pairs($\theta_{current}$, D_{robot}) according to defined frequency. For following the stored way-points, the IR sensors are turned off and target orientation (θ_{target}) and target distance is retrieved from stored way-points. This is then used to calculate $\Delta\omega_{target}$.

$$\Delta\omega_{target} = (\theta_{target} - \theta_{current}) \times P_{orientation} \tag{17}$$

where $P_{orientation}$ is the proportional error constant. This $\Delta\omega_{target}$ is then applied to the motors like equation 4 and 5. Figure 4 shows variables involved in way-point following. The reason for using polar

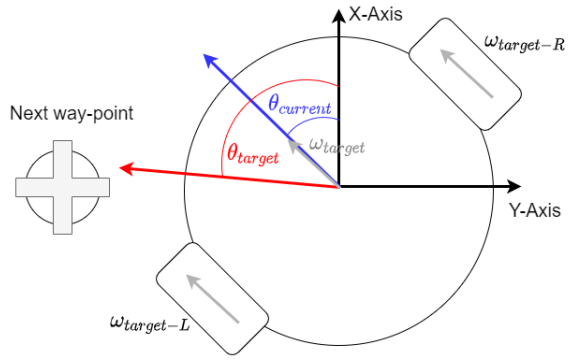


Figure 4: Variables involved in way-point following

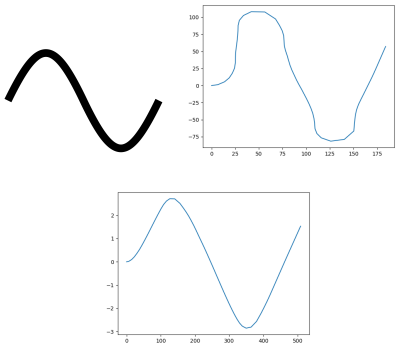


Figure 5: Clockwise from top left: The original path, the path the robot recorded in cartesian coordinates, and the path recorded in polar coordinates

Algorithm 1 Way-point following

$\omega_{target} \leftarrow SPEED$ {Start robot}
$i \leftarrow 0$
$WAYPOINTS \leftarrow getRecordedWaypoints()$
while $i < size(WAYPOINTS)$ **do**
  $\theta_{target} \leftarrow way - pointS[i][\theta]$
  if $D_{robot} \geq way - pointS[i][r]$ **then**
    $i \leftarrow i + 1$
  end if
  correctOrientation(θ_{target})
  updateLocalization() {Update D_{robot} and $\theta_{current}$}
  applySpeedToMotors()
end while
$\omega_{target} \leftarrow 0$ {Stop robot}
applySpeedToMotors()

coordinates as opposed to Cartesian is that for the robot's normal line following routine, the readings lead to polar coordinates naturally. Converting them to Cartesian and back again to polar causes much loss of data. For example, figure 5 shows the recorded way-points in both polar and Cartesian coordinates compared to the original path. This caused much worse performance when the robot is made to follow the path again.

6 Methodology

The robot is put at a starting point, $(x_{start-0}, y_{start-0})$, and made to follow a line using line sensors as detailed in section 5.3. At a specified frequency, the robot stores the odometric variables, $\theta_{current}$ and D_{robot}, as polar coordinates in an array.

$$WAYPOINTSS[i] \leftarrow [\theta_{current}, D_{robot}] \tag{18}$$

When the line path ends at (x_{end-0}, y_{end-0}), the robot stops and resets $\theta_{current}$ and D_{robot}. Now the data is transferred to a computer and the robot is placed at the starting point $(x_{start-0}, y_{start-0})$. This time, it follows the previously stored way-points as detailed in section 5.4 and stops at point (x_{end-0}, y_{end-0}) when it "thinks" it is at the correct endpoint. The error in the endpoints is measured by a ruler as shown in Figure 6. However, the total length of the path travelled is measured by the robot $(X_{measured})$ which makes it error-prone as established in section 4.2. Hence the actual distance travelled (X_{actual}) is calculated using as follows:

$$X_{actual} = (X_{measured} * m) + c \tag{19}$$

where m and c are the slope and intercept calculated by linear regression in section 4.2. This difference in the robot's calculated perception and the measured ground truth gives a metric of the odometry's performance consistency and hence reliability. The final error (X_{error}) is calculated using absolute measured error (ΔX) as a percent of the actual distance travelled (X_{actual}):

$$X_{error} = \frac{\Delta X}{X_{actual}} \times 100 \tag{20}$$

We use different types of paths, with way-points recorded at different frequencies, to determine sources of errors based on curvature and sharpness of curves and length of the path traveled. The diameter of the robot was found to be 9cm, hence the sine and square wave paths could not have wavelength less than $9 \times 2 = 18cms$.

7 Results

The paths used are a straight line to test the errors in the pure forward motion of the robot, a square wave to test the errors in taking sharp turns, and lastly, a sine wave to test a mix of both forward as well as rotational motion. The frequency of way-point recording is also varied to check if accuracy is proportional to the number of way-points recorded by the robot. This gives us $3paths \times 4frequencies \times 5repetitions =$

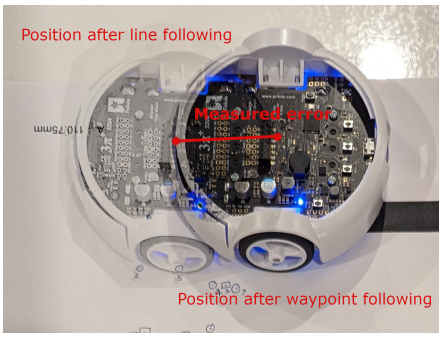


Figure 6: Measuring error

60*readings*. The results of the experiment are shown in Figure 7.

The maximum average error across paths was found

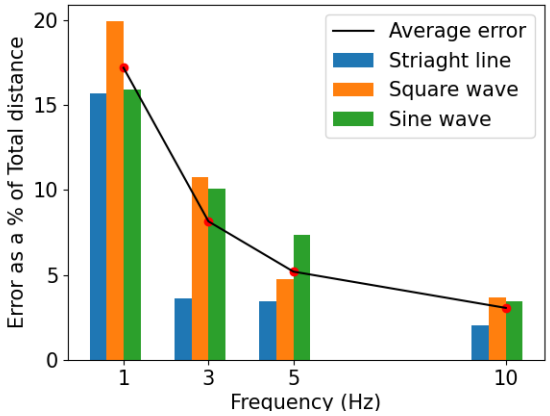


Figure 7: Change in error for different paths and way-point recording frequencies

at the lowest frequency of collecting way-points and the lowest at the highest frequency. This goes against our initial hypothesis that frequency of collection of way-points should have minimal effect on the robot reaching the endpoint. On further investigation, it was found that changing the frequency of way-point collection has a major impact on the curves the robot has to negotiate. As is evident by Figure 8, lower frequencies require sharper turns which causes the robot to execute turns faster, which in turn causes slipping due to inertia which leads to false measurements in Odometry.

The theory of sharp turns causing more error is reinforced by the fact that in most cases (except $f = 5$), the square wave path has the most error and the straight line path has the least. The reconstruction of the square wave path compared to the sine wave path also shows that the latter is reconstructed better by the recorded way-points than the former, as shown in figure 9. This could be because when following a path, for the robot the shape of its corners is irrelevant and the line is comfortably followed as

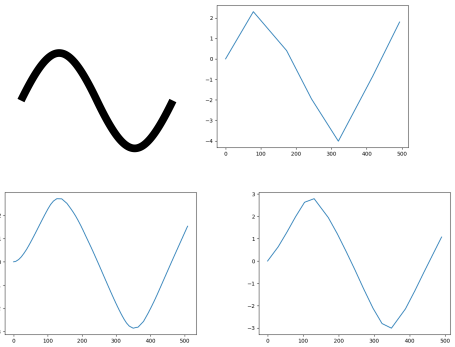


Figure 8: Robot's perception of paths with change in frequency of way-point collection. Clockwise from top-left, original path, path reconstruction for frequency 1, 3, and 10

long as the line sensors detect something. It could be said that the followed and thus reconstructed path is the shortest route the line-following robot could take. However, this does not affect the robot's ability to follow the sharper turns, as is evident by visual inspection of the way-point following in the supplementary video [3], but it does introduce more errors than smoother curves or straight lines.

Figure 7 also shows a general trend of decrease in

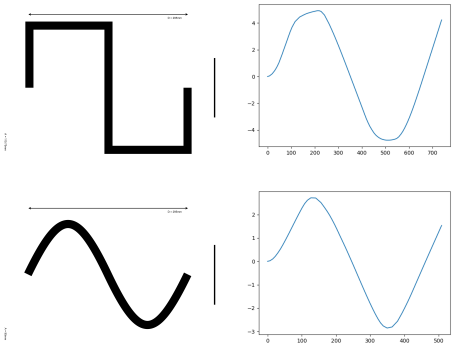


Figure 9: Robot's perception of paths. Clockwise from top-left, original square-wave path, reconstructed square-wave path, original sine-wave path, reconstructed sine-wave path

error with the increase in frequency of way-point collection. The performance of the robot increases considerably when it collects 3 way-points per second. The average percentage error across paths, of 17.16% for way-point collection frequency of 1 is reduced by 8% when the frequency is set to 3, which is a considerable change. However, the same difference is lower when increasing frequency from 5 to 10 (5.2% to 3%). This may indicate that there are diminishing returns as the frequency of way-point collection is increased. We could not test frequencies greater than 10 because the way-points are stored in the SRAM of the robot which has limited memory and could offer

only enough space for a frequency of 10 at the chosen movement speed and path length.

8 Conclusion and Future Work

By conducting experiments based on our hypothesis, we can conclude that the encoders are consistent enough for applications in localization and way-point following. Odometry was found to differ in performance based on the type of path it is recording as the way-points could miss the features required for a smooth path travel. The order of performance in path types was found to be generally best on straight lines, slightly worse on sine wave paths and worst on square wave paths. Also, the route memorizing algorithm performs better as the robot records more way-points per second of travel as reducing the distance between two way-points generates smaller vectors and gradual change in target orientation which results in a smoother transition from one way-point to the next. The errors due to bad path reconstruction could be solved by using a higher frequency of way point recording or slower speed of movement provided the system has enough memory to store more way-points.
Some errors also resulted from inertial specially at sudden stops and slipping at sharp turns. Such errors can be corrected by a minimum acceleration trajectory calculated using Euler-Lagrange Equations as speed can be commanded.
For optimizing the system, future work would consist of feeding the error back into the system so that a machine learning algorithm can be implemented to predict and reduce the error by learning from previous iterations. The robot memorizing can be further used to optimize navigation and localization in robots such as Roomba and rescue operation robots

9 References

References

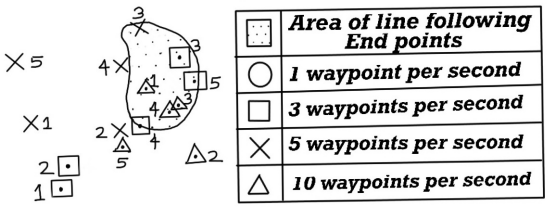
[1] Mohamed Jallouli, Lobna Amouri, and Nabil Derbel. "An effective localization method for robot navigation through combined encoders positioning and retiming visual control". In: *Journal of Automation, Mobile Robotics and Intelligent Systems* 3.2 (Jan. 2013), pp. 15–23. URL: https://www.jamris.org/index.php/JAMRIS/article/view/639.

[2] Rahul Kumar et al. "Maze Solving Robot with Automated Obstacle Avoidance". In: *Procedia Computer Science* 105 (2017). 2016 IEEE International Symposium on Robotics and Intelligent Sensors, IRIS 2016, 17-20 December 2016, Tokyo, Japan, pp. 57–61. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.01.192. URL: https://www.sciencedirect.com/science/article/pii/S1877050917302107.

[3] Amey Rawool and Shivang Gangadia. *Pololu 3pi+ route-memorizing robot*. 2022. URL: https://www.youtube.com/watch?v=GUuIf_tNyNM.

[4] Pololu Robotics and Electronics. *Pololu 3pi+ 32U4 User's Guide*. 2021. URL: https://www.pololu.com/docs/0J83/5.

10 Appendix A



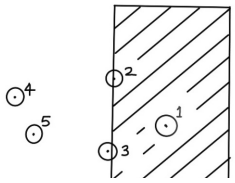


Figure 10: Straight line path: Line-following V/S route-memorizing endpoints

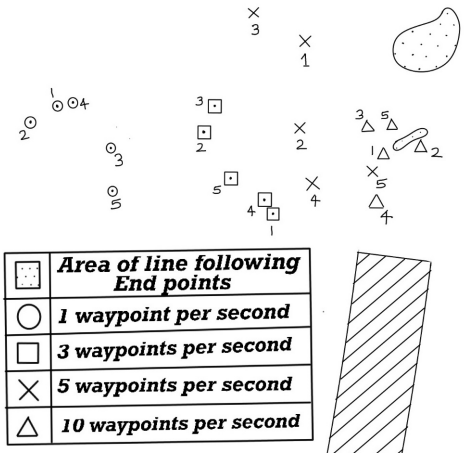


Figure 11: Sine wave path Line-following V/S route-memorizing endpoints

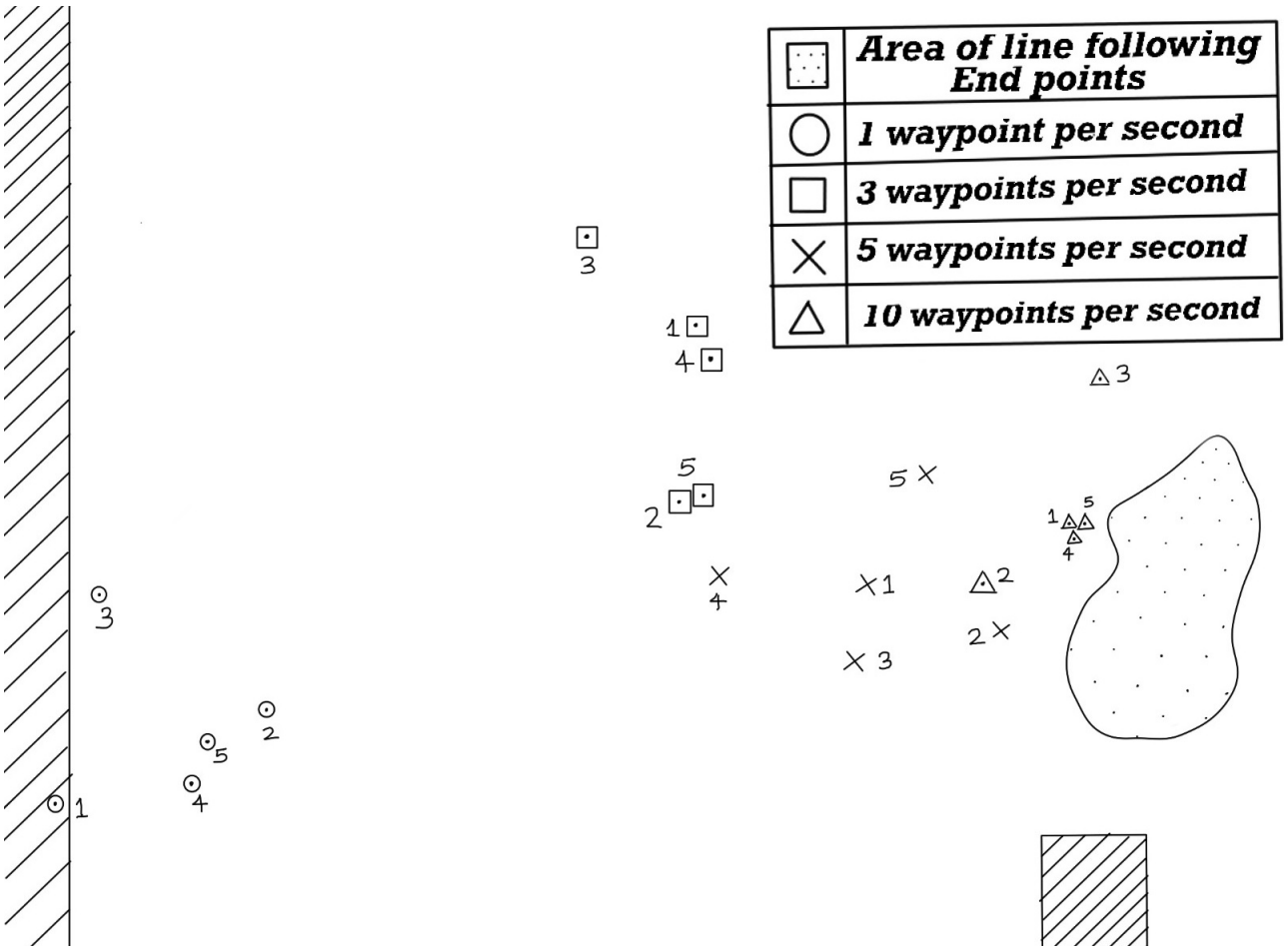


Figure 12: Square wave path: Line-following V/S route-memorizing endpoints