Path-Following AGV for Battlefield Ammunition Supply Using Lyapunov-Based Control and NGAWWI

FINAL PROJECT

Submitted as partial fulfillment for the subject
TF4107 (Robotics and Automation) / Class 01

Team:

Muhammad Fadlan Abhyasa	13623036
Theodore Kimi Vickenson Gunawan	13623032
Billmar Sandiego	13623051

Lecturer:
Augie Widyotriatmo, S.T., M.T., Ph.D.

ENGINEERING PHYSICS STUDY PROGRAM
FACULTY OF INDUSTRIAL TECHNOLOGY
INSTITUT TEKNOLOGI BANDUNG

2025

1. General Description

[1] Mobile robots are robots that can move around to interact with their surroundings, usually doing their role in big area inspection, logistics, patrol, rescue operations, etc. Mobile robots require integrated systems to perceive and adapt to their surroundings. First, of course a mobile robot needs to have their own way of traversing the environment, varying from wheels, tracks, legs, etc. Each way of traversing can differ more specifically, for example robots equipped with wheels can differ into omnidirectional wheels, differential drive wheels, single wheels, etc.

With actuators in place, robots can traverse their environments; however, effective navigation requires awareness of surroundings. But it is limited if they don't know the physical environment of their surroundings. To overcome this problem, some mobile robots are equipped with controllers that could be autonomous or controlled by humans. [2] For autonomous robots, these control strategies can vary from path following, trajectory following, point stabilization, etc.

To achieve a reliable guiding system for these control strategies, a stability analysis and controller support is a key requirement to be implemented in the robot systems. [2] One of the controls that can be used is *Lyapunov Controller*. The purpose of stability control is to maintain and ensure that autonomous robots run on their determined path as accurately and precisely as possible. This is crucial especially for environments where lots of obstacles and boundaries are present and may harm autonomous robots.

2. Project Background

On the battlefield, supplies are one of the key factors that will determine the result of the battle. The supplies may vary from medical needs, demolition tools, to personal ammunition and firearms. In this project we will develop an autonomous ground vehicle simulation on the battlefield as an ammunition supplier. The challenge to successfully develop such a robot comes from creating a good control system that could automatically guide the robot from its starting point to its destination. The battlefield is a chaotic environment, having unclear and random paths within each grid of the map. To overcome this challenge, the robot must be capable of detecting its surrounding terrain to be able to traverse the easiest and safest path to its destination. It is assumed that for this case the

robot already has the knowledge of the battlefield map. In this case, the repetitive task that the robot must do is deliver supplies to multiple locations according to the path it has chosen. To simplify the simulation, some assumptions are applied:

1. 2D map with no contour,
2. Neglect battlefield disturbance (explosions, shocks, etc.),
3. Only use kinematics to model the robot.

3. Differential Drive Kinematics

To achieve the goal, it is necessary to begin with the robot's kinematics model. These models will help determine and explain how the robot moves throughout the battlefield in the simulation. The schematic of the robot will be made simple for the simulation as the following:

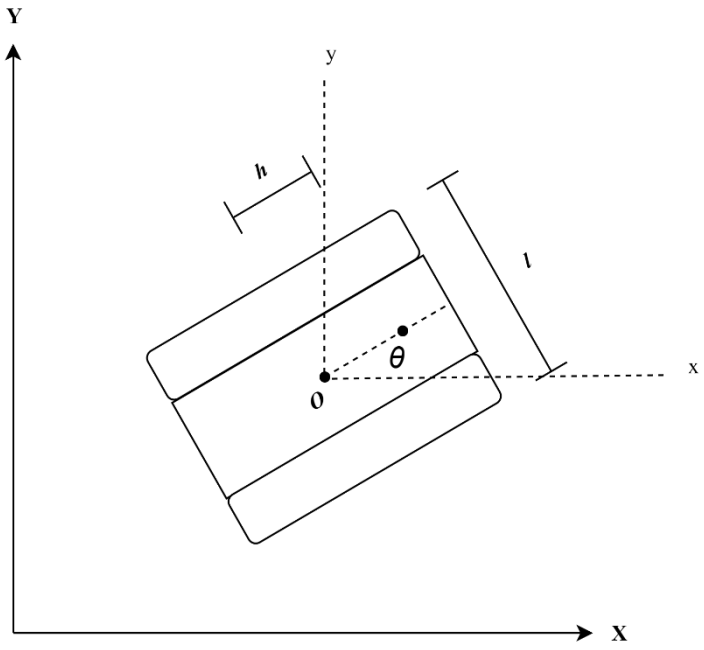


Figure 3.1. Kinematic Model of AGV

The figure above is a robot equipped with 2 tracks with each individual motor to drive and steer. For the sake of simplicity, each track on each side of the robot is assumed to be wheels. The robot can move along the 2D map with its velocity v and change its heading angle with its angular rotation ω. The tracks are placed within a distance l from the

middle point *O* of the robot. The general coordinate of this system is $x = [X, Y, \phi\ ]$and $\dot{x} = [\dot{X}, \dot{Y}, \dot{\phi}]$. For a certain point located with distance *l* from point *O*, the coordinate is:

$$x_h = x_o + hcos\phi$$

$$y_h = y_o + hsin\phi$$

Taking the time derivative of the equation gives us:

$$\dot{x}_h = \dot{x}_o - \dot{\phi} h \sin\phi$$

$$\dot{y}_h = \dot{y}_o + \dot{\phi} h \cos\phi$$

Simplifying the time derivative terms using *v* and *ω* and by formulating the equations into a matrix:

$$\begin{bmatrix} \dot{x}_l \\ \dot{y}_l \end{bmatrix} = \begin{bmatrix} \cos\phi & -l\cos\phi \\ \sin\phi & l\cos\phi \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

The dimensions for the AGV in the Figure 3.1 above is as follows:

Dimension	Length (m)
l	1
r	0.4

Table 3.1. AGV Geometric Properties

The mathematical model of the steering will be as the following:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \frac{r}{2} \begin{bmatrix} 1 & 1 \\ \frac{1}{l} & -\frac{1}{l} \end{bmatrix} \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix}$$

$$= T \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix}$$

$$\Leftrightarrow \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} = T^{-1} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Where *l* is the distance between both wheels and R is the wheel radius.

The robot's state (i.e. its position and orientation in space) is defined as:

$$\bar{x} = \begin{bmatrix} x \\ y \\ \phi \end{bmatrix} \Rightarrow \dot{\bar{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v\cos\phi \\ v\sin\phi \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\phi & 0 \\ \sin\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Additionally, the next state of the system can be approximated using Euler's method:

$$\bar{x}_{i+1} \approx \bar{x}_i + \dot{\bar{x}}\Delta t$$

Where Δt is the time step used in the simulation. The value used in the simulation is 0.1.

4. Error Calculation

To ensure the path following system to be as accurate as possible, an error calculation for the robot is an essential step. The error calculation for the robot to follow its desired path and achieve its next pose is done through a set of comparisons which are lateral error and heading error. Since the robot is traversing in a 2D plane in the simulation, it is important to first have the robot's coordinate to measure the lateral error between the robot to its supposed coordinate according to the pre-determined path. The lateral error is the perpendicular distance from the path that the robot is moving on to the pre-determined path that it is supposed to follow.

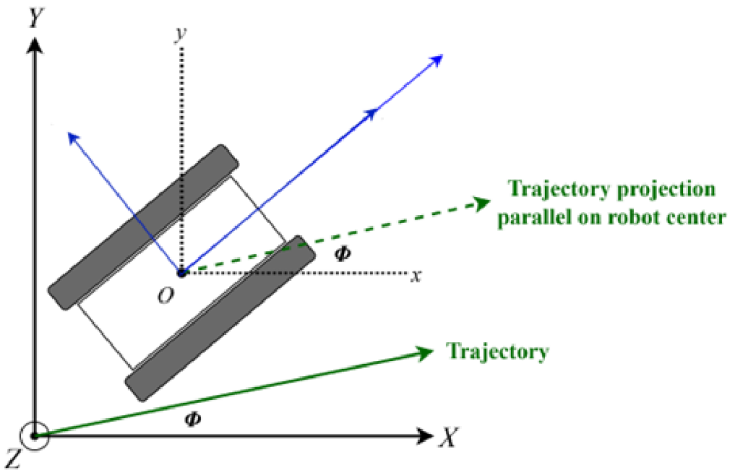


Figure 4.1. Path Following AGV Schematic

Suppose we have the robot coordinate $[x_r\ y_r]^T$, However, the robot is offset from its desired path, thus it must travel a certain distance perpendicularly to return to its originally planned path. In this case we are going to call it the lateral distance. Suppose we

have a path angled ψ with respect to x-axis. To find the robot's lateral distance from the desired path, we can use a rotation matrix as follows:

$$\begin{bmatrix} x_r' \\ y_r' \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x_r \\ y_r \end{bmatrix}$$

Hence, the lateral distance error h_e can be obtained from the second equation above which is:

$$h_e = y_r' = -x_r \sin\phi + y_r \cos\phi$$

Now, we take the time derivative of h_e:

$$\frac{dh_e}{dt} = \dot{h}_e = -\dot{x}_r \sin\phi + \dot{y}_r \cos\phi$$

Since the robot moves forward at speed v in direction θ:

$$\dot{x}_r = v\cos\theta\,, \dot{y}_r = v\sin\theta$$

$$\dot{h}_e = -v\cos\theta\sin\phi + v\sin\theta\cos\phi$$

$$= v[-\cos\theta\sin\phi + \sin\theta\cos\phi]$$

Using the trigonometric identity $\sin(\theta-\phi) = \sin\theta\cos\phi - \cos\theta\sin\phi$, we acquire:

$$\dot{h}_e = v\sin(\theta-\phi)$$

The angular error of the robot with respect to the path is defined as $\theta_e = \theta - \phi$ where θ is the angle of the robot's velocity vector with respect to the global x axis:

$$\dot{h}_e = v\sin\theta_e$$

Since ϕ is mostly constant with respect to time (apart from changing waypoints):

$$\dot{\theta}_e = \dot{\theta} = \omega$$

5. Path Following Control Scheme

In our control scheme, we want the robot to follow the line and move parallel against it, meaning we want h_e and θ_e to approach zero. We can think of the *Lyapunov* function as a "measurement of total system error". The candidate *Lyapunov* function is defined as:

$$V = \frac{1}{2}h_e^2 + \frac{1}{2}\theta_e^2$$

The time derivative of the above Lyapunov function is therefore:

$$\dot{V} = h_e\dot{h}_e + \theta_e\dot{\theta}_e$$

$$\dot{V} = h_e v\sin\theta_e + \theta_e\omega$$

A control *input* for ω is then constructed as follows:

$$\omega = -k_\theta \theta_e - k_h h_e \frac{\sin \theta_e}{\theta_e}$$

Where k_θ and k_h are tuning constants used to adjust the responsiveness of the control algorithm. Since k_θ and k_h are both positive, we acquire that the derivative of the Lyapunov function is:

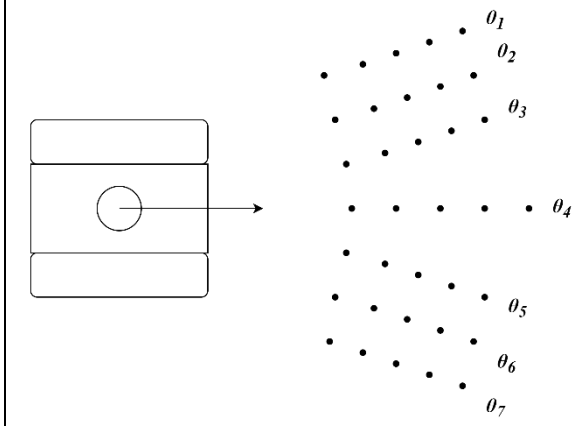
$$\dot{V} = -k_\theta \theta_e^2 \leq 0$$

Since $\dot{V} < 0$, we can therefore infer that the system is always stabilizing (converging). To avoid dividing by zero, the last term of the control *input,* that is, $\frac{\sin \theta_e}{\theta_e}$, is set equal to one whenever θ_e equals zero.

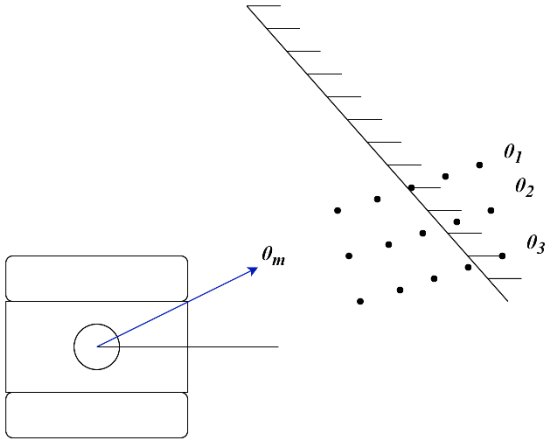
6. Obstacle Avoidance

For a robot that operates on a battlefield, a crucial feature it must include in its system is obstacle avoidance. Sometimes having the real-time mapping of the battlefield is difficult, especially in areas that are not under control. We propose an obstacle avoidance algorithm named *Node-Generative Angle-Weighted Waypoint Injection* (NGAWWI). The main principle of this algorithm is by measuring unique angles to decide how far from the trajectory the robot should deflect from a detected obstacle/wall.

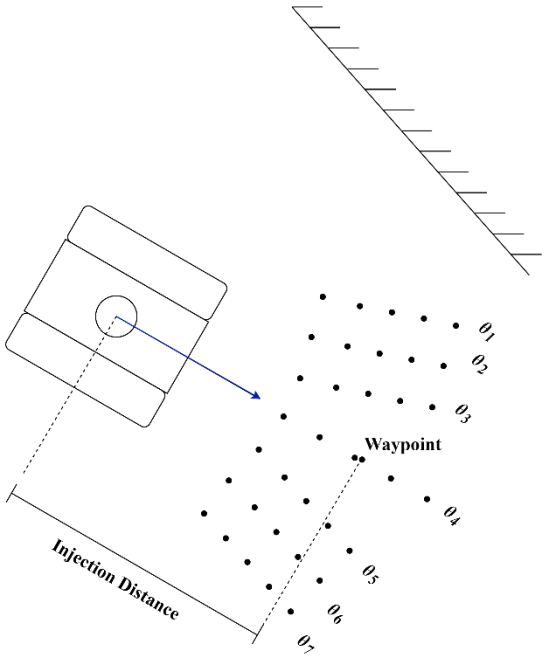
The NGAWWI obstacle avoidance works by creating several nodes in front of the robot in multiple unique angles. These nodes are spread from the inner radius to the outer radius for each unique angle. Upon interfering with walls or obstacles, after a certain number of N nodes have been marked as occupied, the robot will measure the average of the angles that have occupied nodes, this average angle will be called θ_m (In the illustration, θ_m will be equal to the average of θ_1, θ_2 and θ_3). Also, the number of angles that interfere will also be stored in a variable n (In the illustration, n = 3). These values are then used by the robot to calculate how far it must deflect to avoid collision. After calculating how far it must deflect, the robot will create a waypoint that it must reach first to navigate to the destination again. Upon interfering with another obstacle, the process will repeat until the destination has been reached. This algorithm is made possible by defining some functions which are node generating and waypoint injecting.



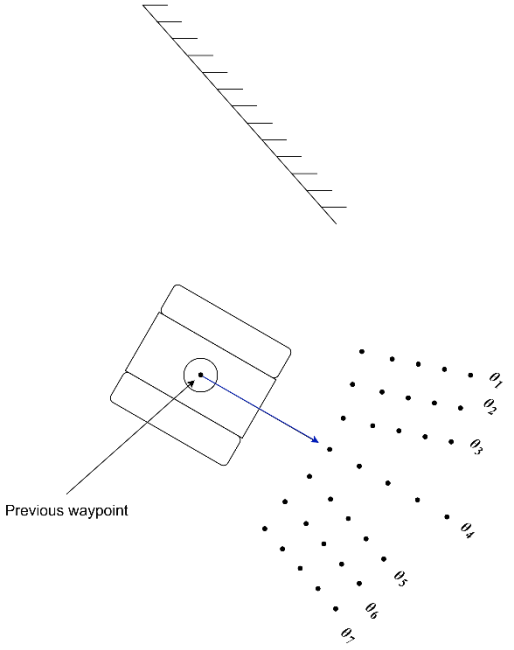
1: Robot use nodes to scan surrounding environment



2: Upon interference, calculate mean angle of angles with occupied nodes



3: Generate waypoint function uses the θ_m to direct robot away from obstacle



4: Robot reaches waypoint and continue heading to the destination

Figure 6.1. Obstacle Avoidance steps

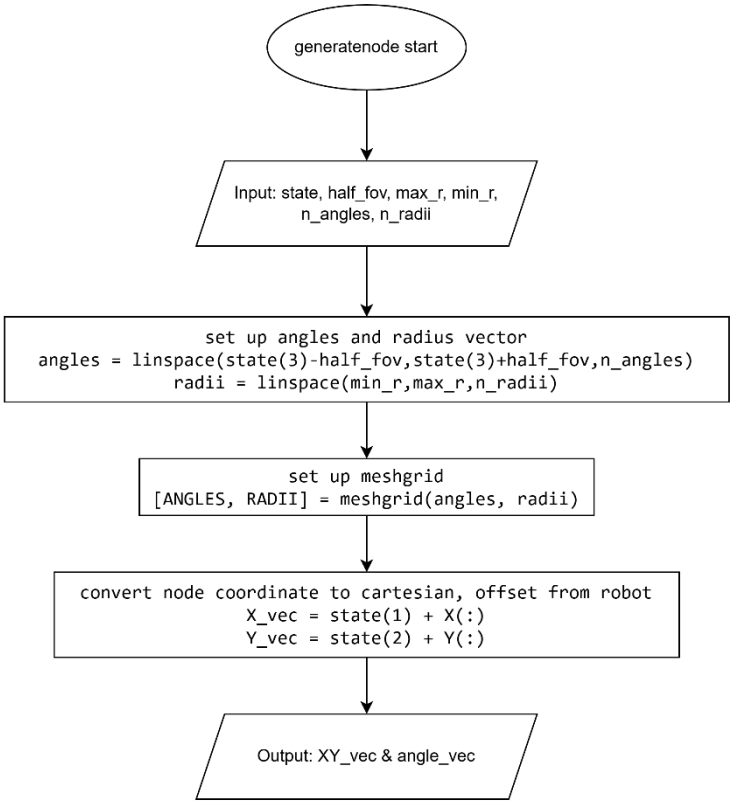


Figure 6.2. Generate Node function flowchart

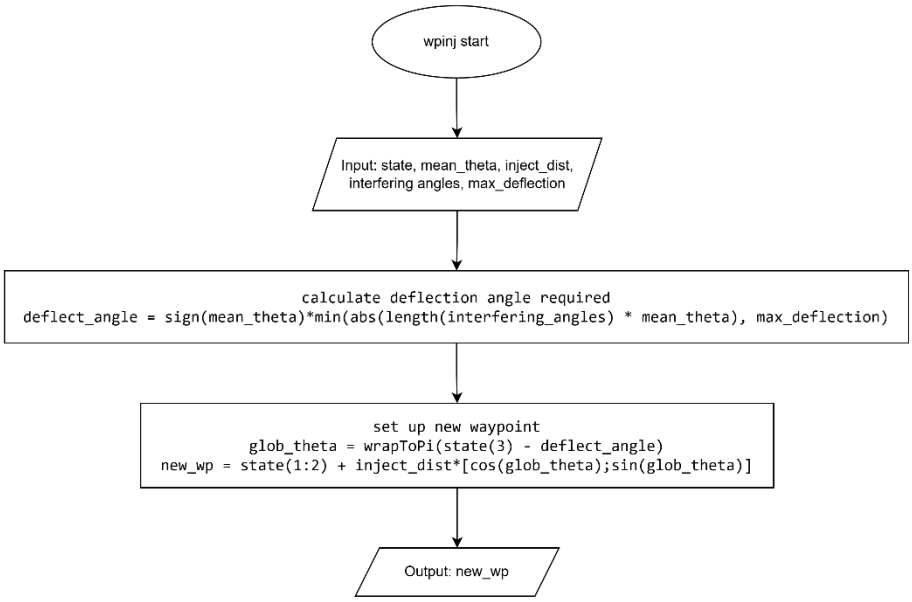


Figure 6.3. Waypoint Injection function flowchart

7. Path Following Simulation & Analysis

7.1. Simulation

For this simulation, the path that our robot will follow is as follows:

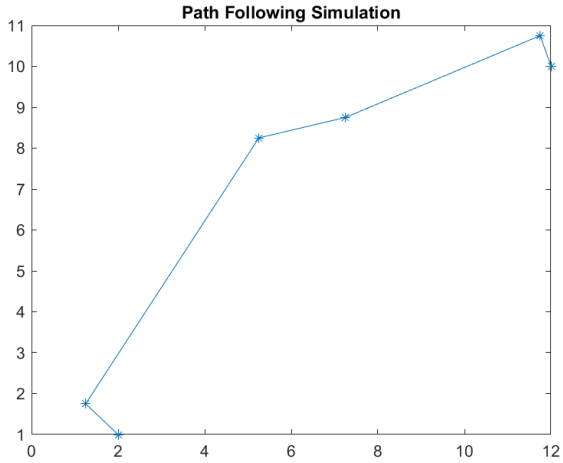
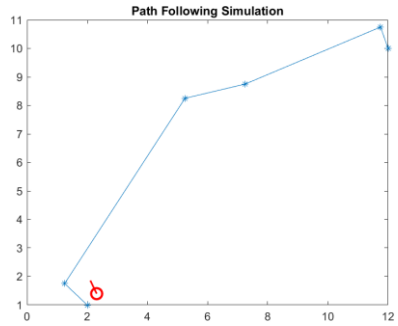
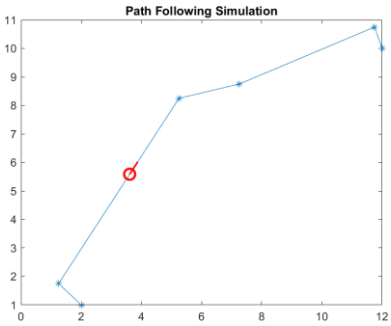


Figure 7.1.1. Desired path to follow

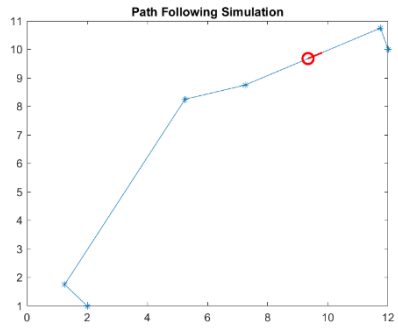
The next few images are some representations of the robot following its desired path taken from the simulation.



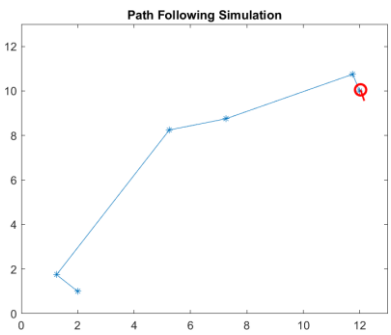
Robot State at t = 1s



Robot State at t = 10s



Robot State at t = 20s



Robot State at t = 28s (final)

Figure 7.1.2. Robot path following simulation

The error plot throughout the simulation is given by the following figures:

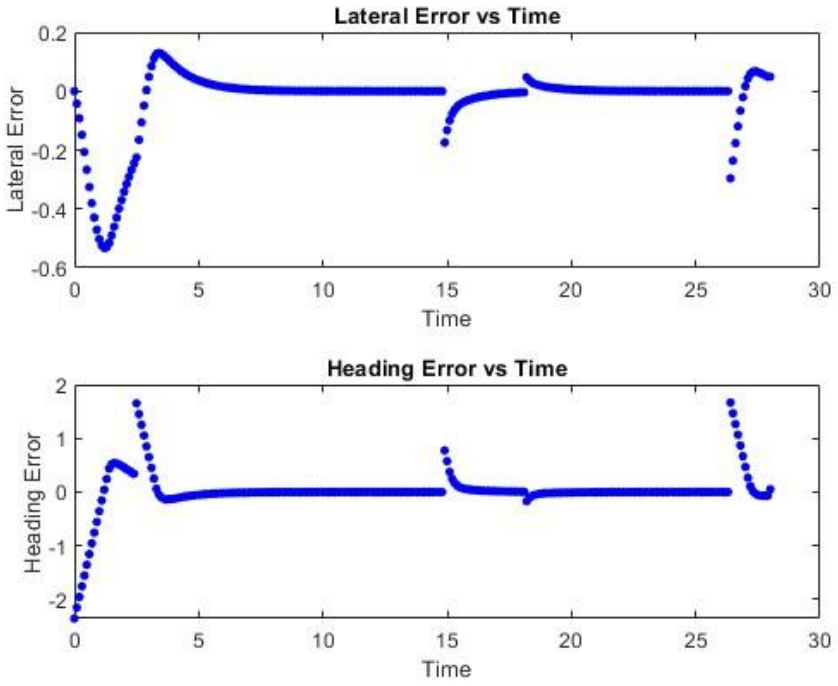


Figure 7.1.3. Error vs time plot

7.2. Analysis

By observing the error plots above we can see that there are some error spikes throughout the simulation. For both the lateral and heading error, from initial t = 0s until about t = 4s, the robot is still adjusting its pose to follow the assigned path. The lateral error is caused due to the robot having translational velocity since the beginning of the simulation while not having the correct heading, causing it to veer off track until it stabilizes its pose after t = 4s. During the first 4 seconds, the error also oscillates. This may be caused due to proportional overshoots in stabilizing the robot to follow its desired path. Moving on to approximately t = 15s, the robot encounters more errors. In this case, the error is caused by the discontinuity of the path.

Due to sudden changes, the robot cannot smoothly follow the path. Lastly, the error spikes again near the end of its path. Similar like the previous error, this is caused by path discontinuity. The error spikes are larger in this case because the path gives a very sharp turn. This also can be shown by a different path comparing the path following result.

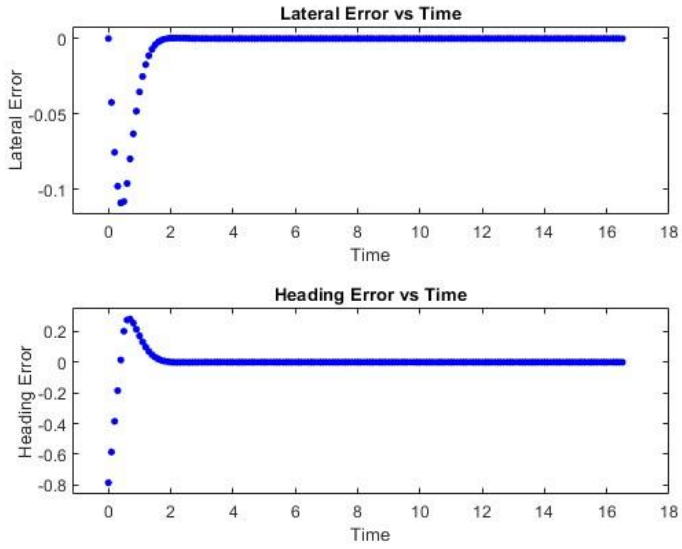


Figure 7.2.1. Error plot for a straight-line path

8. Obstacle Avoidance Simulation

The obstacle avoidance simulation of the path following AGV result using an algorithm that we came up with, *Node-Generative Angle Weighted Waypoint Injection* is as the following figure.

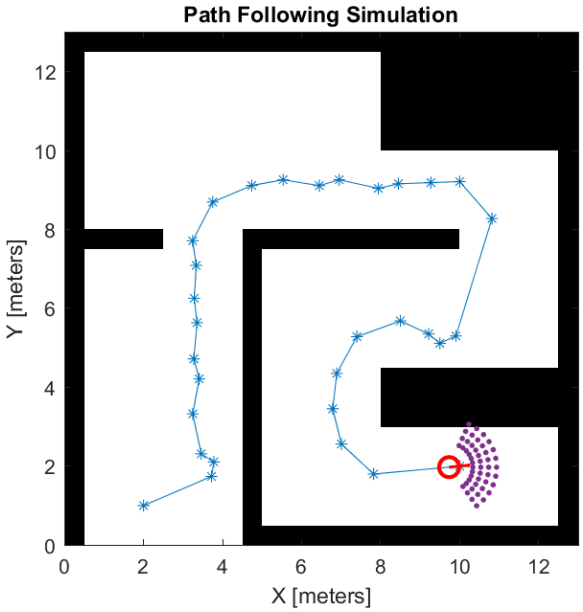


Figure 8.1. Final robot trajectory obtained from simulation

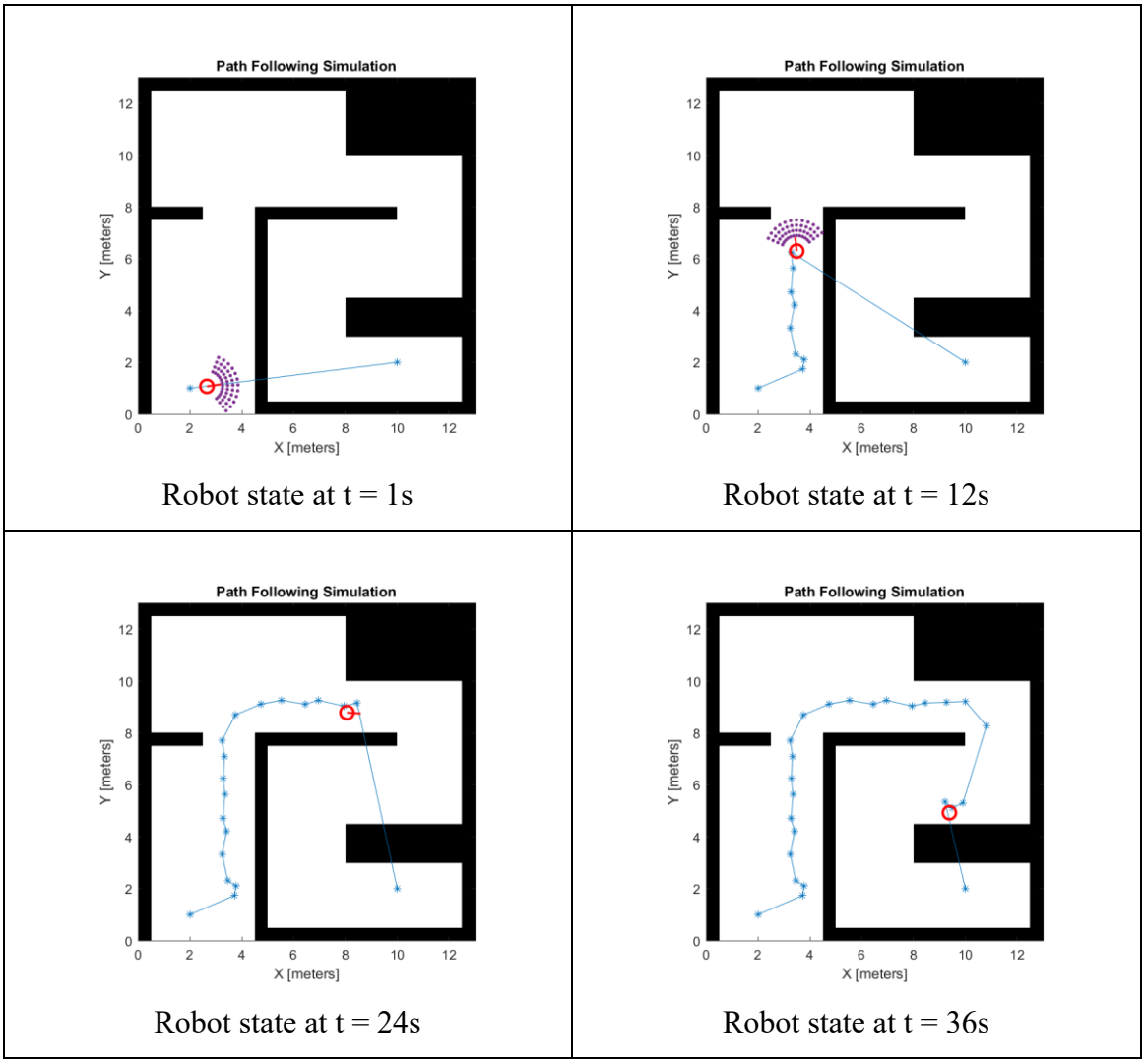


Figure 8.2. Robot trajectory following simulation

9. Limitations and Disadvantages

The Node-Generative Angle-Weighted Waypoint Injection (NGAWWI) algorithm represents a foundational approach that prioritizes simplicity and computational cost. However, it exhibits several inherent limitations that restrict its applicability in complex environments that may require more advanced navigation techniques.

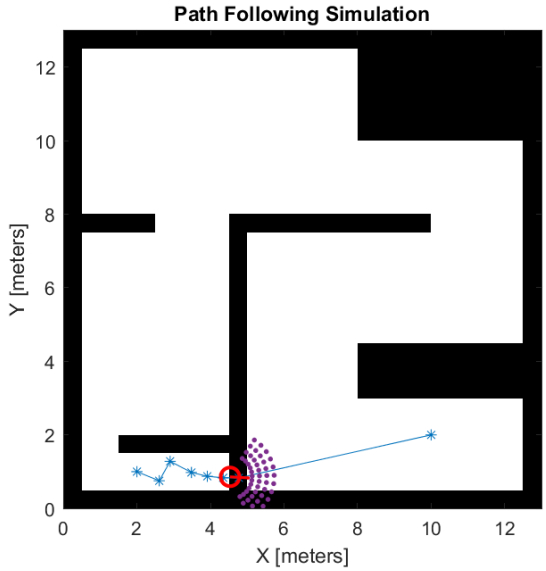


Figure 9.1. Robot stuck on corner because the destination is located behind the dead-end (local minima problem)

The most critical limitation of NGAWWI is its susceptibility to local minima situations, as demonstrated in Figure 9.1. When the robot encounters U-shaped obstacles or corner configurations where the destination lies behind an obstruction, the algorithm fails to provide adequate escape mechanisms. The reactive nature of NGAWWI means it only considers immediate obstacle avoidance without global path re-planning capabilities. In the illustrated scenario, the robot becomes trapped because the algorithm cannot reason about backing away from the obstacle to find an alternative route around the barrier. These limitations suggest that while NGAWWI serves as an effective foundation for basic obstacle avoidance, it requires integration with complementary algorithms or significant enhancements to handle complex, real-world navigation scenarios effectively.

1. Several strategies solutions could be implemented to address the critical local minima problem. Implementing a memory system that tracks known positions and triggers reverse motion when it detects repetitive positioning patterns or incorporating bug algorithm principles by adding a wall-following mode when NGAWWI becomes trapped, might help the AGV circumvent the local minima problem when it meets U-shaped obstacles or dead-ends.
2. The most promising improvement for the foreseeable future involves systematic integration with complementary algorithms. For example, combining NGAWWI with global path planners (such as A*, RRT*) where NGAWWI handles local obstacle avoidance while global planners provide strategic guidance.

10. Conclusion

This paper presents a comprehensive autonomous navigation system that integrates a Lyapunov-based path following controller with a novel obstacle avoidance algorithm termed Node-Generative Angle-Weighted Waypoint Injection (NGAWWI). The Lyapunov controller ensures stable path following with guaranteed convergence properties, while NGAWWI provides computationally efficient, real-time obstacle avoidance through angular analysis and deflective waypoint injection. Simulation results demonstrate successful navigation through some complex obstacle configurations, validating the practical effectiveness of combining established control theory with innovative algorithmic approaches. The modular architecture enables systematic parameter tuning and real-time visualization, making the system suitable for both research applications and rapid prototyping in resource-constrained environments.

While the current implementation successfully balances theoretical rigor with computational simplicity, several limitations have been identified that present opportunities for future enhancement. The reactive nature of NGAWWI and its susceptibility to local minima situations highlight the need for predictive planning capabilities and escape mechanisms. However, the system's modular design philosophy positions it as a solid foundation for evolutionary development, enabling integration of advanced features such as adaptive sensing, path optimization, and dynamic environment handling without fundamental architectural changes. This work demonstrates that sophisticated autonomous navigation behavior can emerge from the careful integration of well-understood components, emphasizing engineering principles of modularity and maintainability over algorithmic complexity for its own sake.

References

1. Rubio, Francisco & Valero, Francisco & Llopis-Albert, Carlos. (2019). A review of mobile robots: Concepts, methods, theoretical framework, and applications. International Journal of Advanced Robotic Systems. 16. 172988141983959. 10.1177/1729881419839596.
2. Widyotriatmo, A. (2018). Robot Beroda. In *DASAR-DASAR MEKATRONIKA DAN ROBOTIKA* (1st ed., pp. 111–129). essay, ITB PRESS.