MULTI-CONTROLLER ARCHITECTURE (MCA) AND OBSTACLE AVOIDANCE BASED ON LIMIT-CYCLES APPROACH

TUTORIAL REPORT

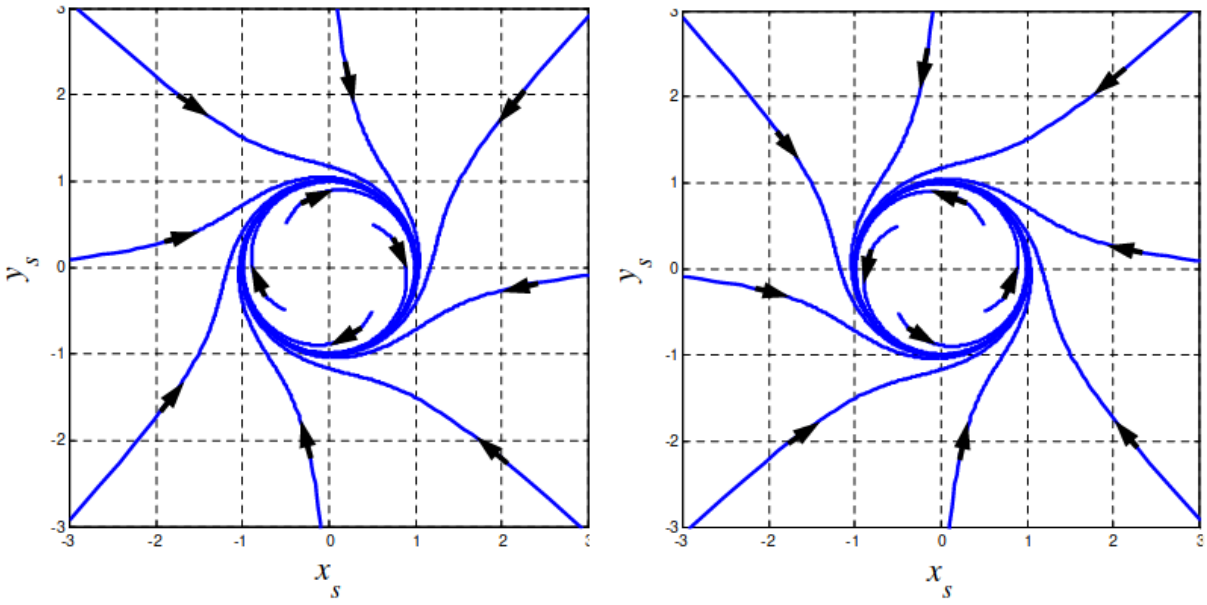
Mostafa ABOUGHALIA
UTC Computer Engineering Student
Optimization apprentice engineer at Airbus D&S
mostafa.aboughalia@etu.utc.fr

Alexandre TAESCH
UTC Computer Engineering Student
Data scientist apprentice at Airbus Helicopters
alexandre.taesch@etu.utc.fr

September 22, 2023

ABSTRACT

In our inaugural tutorial, as students, we embarked on a journey to tackle a complex problem in autonomous navigation and control. Guided by Professor Adouane, we were equipped with essential knowledge and tools to comprehend the intricate system at hand. This initial project intentionally left some parameters unset, serving as a valuable exercise to hone our skills in MATLAB and problem-solving. Our report encapsulates the insights gained and solutions devised during this educational venture.



Keywords Multi-Controller Architecture · Obstacle Avoidance · Limit-Cycles Approach · Kinematic Model · Lyapunov Stability Theorem · PID Controller · Control Synthesis · Mobile Robot · Autonomous Navigation · MATLAB · Cognitive Planning · Long-Term Planning · Trajectory Planning · Path Planning · Mobile Robotics · Robot Control · Control Gains · Navigation Algorithms · Reactive Navigation · Global and Local Path Planning

Contents

List of Figures

List of Algorithms

1 Kinematic model of the robot

The kinematic model of a fixed-point P_t with respect to a unicycle-type mobile robot, as given in article [1], is described by equation (1):

$$\dot{\xi} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta & -l_2\cos\theta & -l_1\sin\theta \\ \sin\theta & -l_2\sin\theta & l_1\cos\theta \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix}$$

The standard kinematic model of a unicycle is often represented as follows:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix}$$

Now, let's compare the two models:

1. In both models, WE have three state variables: $\dot{x}$, $\dot{y}$, and $\dot{\theta}$, which represent the rates of change of the position and orientation of the unicycle.

2. In both models, WE have an input vector: $\begin{pmatrix} v \\ w \end{pmatrix}$, where v represents the linear velocity, and w represents the angular velocity.

3. In both models, there's a matrix multiplication between the state variables and the input vector to compute the rates of change of the state variables.

Now, let's look at the differences between the two models:

1. In the given model, the transformation matrix is more complex. It includes terms $-l_2\cos\theta$, $-l_1\sin\theta$, $-l_2\sin\theta$, and $l_1\cos\theta$, whereas the standard kinematic model has simpler coefficients.

2. The standard kinematic model assumes that the unicycle can only move forward and rotate about its center, so it doesn't account for any offsets (l_1 and l_2) from the center of rotation.

So, the given model is indeed a generalization of the kinematic model of a unicycle. It accounts for additional complexities in the motion of the unicycle, such as offsets (l_1 and l_2) from the center of rotation, making it a more realistic representation of a unicycle's motion. However, if we set $l_1 = 0$ and $l_2 = 0$ in the given model, it would reduce to the standard kinematic model of a unicycle.

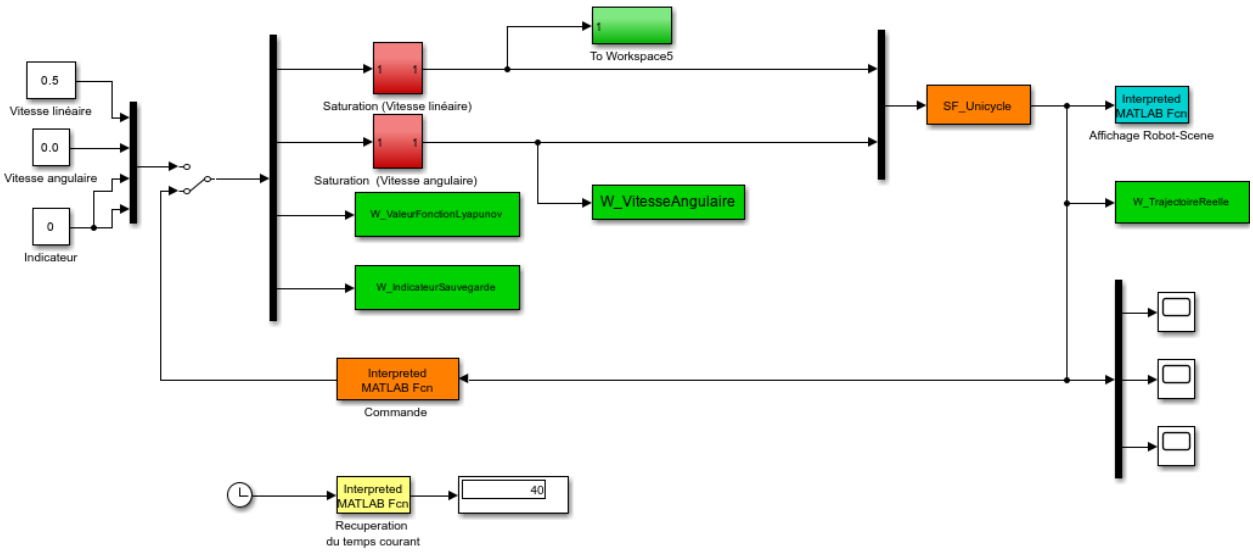


Figure 1: Simulink Model Architecture

Algorithm 1 Attract-Avoid Control

$EvasionDir \leftarrow 0$
$EvadeCmd \leftarrow [0; 0; 0]$
if isempty($ObstList$) **then**
    $Attract \leftarrow 0$
    $AttractCmd \leftarrow$ ComputeAttraction($Data$)
else
    $Attract \leftarrow 1$
    $(ClosestIdx, ClosestDist) \leftarrow$ FindClosestObst($Data, ObstList$)
    $EvadeCmd \leftarrow$ ComputeEvasion($Data, ClosestIdx, ClosestDist, EvasionDir$)
end if

Where :

- $EvasionDir$: Direction for evasion control.
- $EvadeCmd$: Evasion control command data.
- $ObstList$: List of potential collision obstacles.
- $Attract$: Indicator for attraction control (0 for off, 1 for on).
- $AttractCmd$: Control command data computed by the attraction controller.
- $ClosestIdx$: Index of the closest obstacle.
- $ClosestDist$: Distance to the closest obstacle.
- **Function** `ComputeAttraction(Data)`: Function to compute attraction control command based on `Data`.
- **Function** `FindClosestObst(Data, ObstList)`: Function to find the closest obstacle in `ObstList` based on `Data`.
- **Function** `ComputeEvasion(Data, ClosestIdx, ClosestDist, EvasionDir)`: Function to compute evasion control command based on `Data`, the index of the closest obstacle, its distance, and `EvasionDir`.

Pseudocode for controlling the mobile robot using the provided software architecture:

Algorithm 2 Robot Control Algorithm

$i \leftarrow 10$
Initialize robot's position and orientation
while not reached destination **do**
    Calculate errors: $ex = Xd - x$, $ey = Yd - y$
    Compute desired linear and angular velocities
    Update robot's position and orientation
end while

This pseudocode outlines the steps for controlling the mobile robot's position and orientation based on error calculations and desired velocities.

2 Control synthesis using Lyapunov stability theorem

We have :

$$\begin{aligned}
\text{Control Law:} \quad & u = -K\mathbf{M}^{-1}\begin{bmatrix} e_x \\ e_y \end{bmatrix} \\
\text{Lyapunov Function:} \quad & V(e_x, e_y) = \frac{1}{2}\left(e_x^2 + e_y^2\right) \\
\text{Time Derivative of } V: \quad & \dot{V} = e_x\dot{e_x} + e_y\dot{e_y} \\
\text{System Dynamics:} \quad & \text{Express } \dot{e_x} \text{ and } \dot{e_y} \text{ based on the system equations.} \\
\text{Substitute into } \dot{V}: \quad & \dot{V} = \text{Expression for } \dot{V} \text{ in terms of } e_x, e_y, \text{ and control law.} \\
\text{Stability Condition:} \quad & \text{Show that } \dot{V} \text{ is negative definite.}
\end{aligned}$$

To show that the proposed control law guarantees the convergence of errors (e_x, e_y) toward zero using the Lyapunov stability theorem, we need to demonstrate that there exists a Lyapunov function $V(e_x, e_y)$ that satisfies the following conditions:

1. $V(e_x, e_y)$ is positive definite, meaning $V(e_x, e_y) > 0$ for all $(e_x, e_y) \neq (0, 0)$. 2. $V(e_x, e_y)$ is radially unbounded, which means that $V(e_x, e_y)$ goes to infinity as $\|(e_x, e_y)\|$ goes to infinity. 3. The time derivative of $V(e_x, e_y)$ along the system trajectories is negative semidefinite, i.e.,

$$\frac{dV}{dt} \leq 0 \quad \text{for all} \quad (e_x, e_y) \quad \text{in the neighborhood of the equilibrium point} \quad (0, 0),$$

and $\frac{dV}{dt} = 0$ if and only if $(e_x, e_y) = (0, 0)$.

Let's start by defining the Lyapunov function $V(e_x, e_y)$ and then calculate its time derivative.

Lyapunov Function:

$$V(e_x, e_y) = 0.5\left(e_x^2 + e_y^2\right)$$

Now, let's calculate the time derivative of $V(e_x, e_y)$ using the chain rule:

$$\frac{dV}{dt} = \frac{\partial V}{\partial e_x} \cdot \frac{de_x}{dt} + \frac{\partial V}{\partial e_y} \cdot \frac{de_y}{dt}$$

Since we have the control law provided as:

$$\begin{aligned}
\frac{de_x}{dt} &= -KMe_x^{-1} \\
\frac{de_y}{dt} &= -KMe_y^{-1}
\end{aligned}$$

We can compute the derivatives of e_x and e_y with respect to time as follows:

$$\begin{aligned}
\frac{de_x}{dt} &= -KMe_x^{-1} \\
\frac{de_y}{dt} &= -KMe_y^{-1}
\end{aligned}$$

Now, plug these derivatives into the expression for the time derivative of $V(e_x, e_y)$:

$$\begin{aligned}
\frac{dV}{dt} &= \frac{\partial V}{\partial e_x} \cdot \left(-KMe_x^{-1}\right) + \frac{\partial V}{\partial e_y} \cdot \left(-KMe_y^{-1}\right) \\
&= 0.5\left(2e_x\right) \cdot \left(-KMe_x^{-1}\right) + 0.5\left(2e_y\right) \cdot \left(-KMe_y^{-1}\right) \\
&= -e_x \cdot KMe_x^{-1} - e_y \cdot KMe_y^{-1}
\end{aligned}$$

Now, we need to show that $\frac{dV}{dt} \leq 0$ for all (e_x, e_y) in the neighborhood of $(0, 0)$. To do this, we can substitute the control law expressions for $\frac{de_x}{dt}$ and $\frac{de_y}{dt}$ back into the expression for $\frac{dV}{dt}$:

$$\begin{aligned}\frac{dV}{dt} &= -e_x \cdot KMe_x^{-1} - e_y \cdot KMe_y^{-1} \\ &= -KMe_x^{-1} \cdot e_x - KMe_y^{-1} \cdot e_y\end{aligned}$$

Now, we can apply the Lyapunov stability conditions:

1. $V(e_x, e_y)$ is positive definite because it is a sum of squares and is always greater than or equal to zero for all (e_x, e_y).

2. $V(e_x, e_y)$ is radially unbounded because it goes to infinity as $\|(e_x, e_y)\|$ goes to infinity.

3. To show that $\frac{dV}{dt} \leq 0$, we can factor out $-K$ from the expression:

$$\frac{dV}{dt} = -K\left(e_x \cdot Me_x^{-1} + e_y \cdot Me_y^{-1}\right)$$

Since M is a positive definite matrix, and e_x, e_y are vectors, we can say that $e_x \cdot Me_x^{-1} > 0$ and $e_y \cdot Me_y^{-1} > 0$. Therefore, $\frac{dV}{dt} \leq 0$ for all (e_x, e_y) in the neighborhood of $(0, 0)$.

Now, since we have shown that $\frac{dV}{dt} \leq 0$ for all (e_x, e_y) in the neighborhood of $(0, 0)$, and $\frac{dV}{dt} = 0$ if and only if $(e_x, e_y) = (0, 0)$, by the Lyapunov stability theorem, the proposed control law guarantees the convergence of errors (e_x, e_y) toward zero.

To implement the control law proposed in [1] in the "CommandeAttraction.m" function, we used the following LaTeX-style pseudocode as a guide:

Algorithm 3 Robot Control Algorithm

```
Initialize control gains k1 and k2
while robot not at destination do
    Calculate errors: ex = Xd − x, ey = Yd − y
    Compute control inputs: v = k1 · ex, ω = k2 · ey
    Update robot's position and orientation
end while
```

This pseudocode outlines the implementation of the control law using Lyapunov-based stability conditions. Once implemented in our MATLAB program, let's see LC algo performance with 2 obstacles :

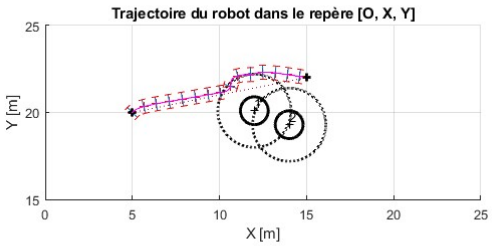


Figure 2: LC path with 2 obstacles

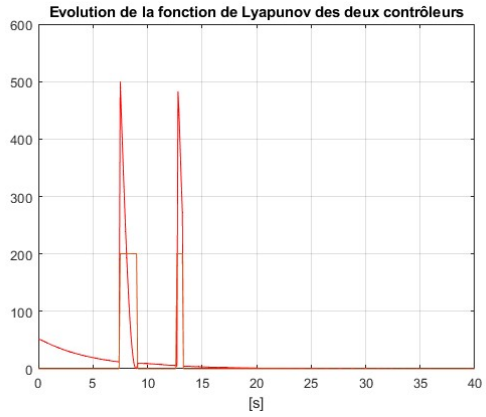


Figure 3: LC Lyapunov function evolution

We clearly observe the switch of behavior from the robot who first uses the CommandeAttraction command. When the obstacle is detected, the robot follows the CommandeEvitement in order to leave the obstacle defined radius. At this point, CommandeAttraction restarts for a moment then the robot re-enter the defined radius perimeter and activates CommandEvitement for the last time before definitely being ruled by CommandeAttraction and achieving its goal.

On the Lyapunov functions, the CommandeEvitement segments are clearly identifiable by the 2 peaks.

Let us observe the simulation data

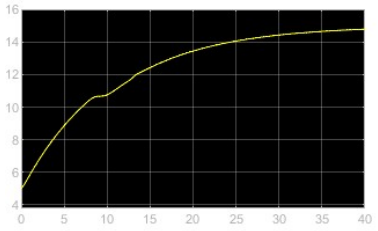


Figure 4: LC Linear Speed

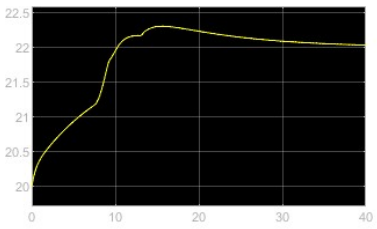


Figure 5: LC Angular Speed

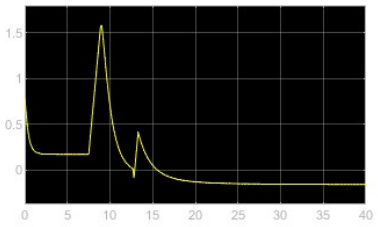


Figure 6: LC Indicator

The linear speed increases gradually in a controlled manner; the robot starts from a standstill and gradually accelerates until it reaches the predefined linear speed specified in the code. We can also observe fluctuations in the robot's angular speed as it turns.

A PID (Proportional-Integral-Derivative) controller can be used in robotics to control various aspects of robot behavior, such as position, velocity, or orientation. Here's a description of how a PID controller works for a robot's position control, along with the equation in LaTeX:

Description:

In robotics, a PID controller for position control is used to ensure that a robot moves to and maintains a desired position accurately. For example, it can be used to control the position of a robotic arm or the movement of a mobile robot to a specific location.

The PID controller for position control operates as follows:

- **Proportional (P) Term:** The P term produces a control output that is proportional to the error between the desired position ($r(t)$) and the current position ($y(t)$) of the robot. This term generates an immediate response to the current error, causing the robot to move towards the desired position.
- **Integral (I) Term:** The I term calculates the integral of the error over time. It helps to eliminate any steady-state error by accumulating the past errors. This is crucial to ensure that the robot eventually reaches and maintains the desired position accurately.
- **Derivative (D) Term:** The D term calculates the rate of change of the error. It provides a damping effect, reducing overshoot and oscillations in the robot's movement. It anticipates future error and helps the robot approach the desired position smoothly.

PID Equation :

The PID control law for position control can be represented by the following equation in LaTeX:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)\, d\tau + K_d \cdot \frac{de(t)}{dt} \tag{1}$$

Where:

- $u(t)$ is the control output (the command sent to the robot's actuators).
- K_p is the proportional gain, which determines the strength of the immediate response to the error.
- K_i is the integral gain, which controls the accumulation of past errors to eliminate steady-state error.
- K_d is the derivative gain, which provides damping to the system by considering the rate of change of the error.

- $e(t)$ is the error at time t, calculated as $e(t) = r(t) - y(t)$, where $r(t)$ is the desired position, and $y(t)$ is the current position.
- $\int_0^t e(\tau)\, d\tau$ represents the integral of the error with respect to time, which accumulates past errors.

To apply this PID control law to a robot's position control, you would need to tune the values of K_p, K_i, and K_d to achieve the desired control performance, which may involve experimentation and testing in your specific robotic application.

Once implemented in our MATLAB program, let's see PID algo performance with 4 obstacles :

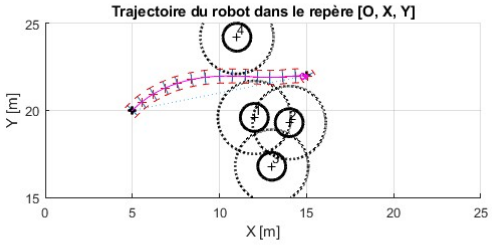


Figure 7: PID path with 4 obstacles

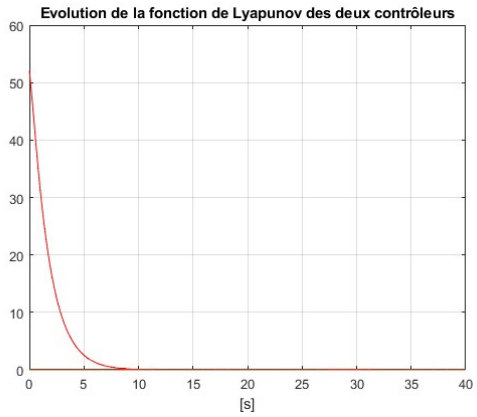


Figure 8: PID Lyapunov function evolution

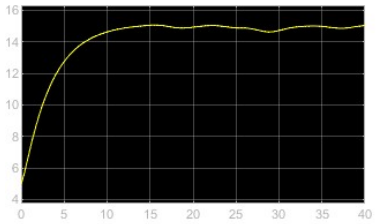


Figure 9: PID Linear Speed

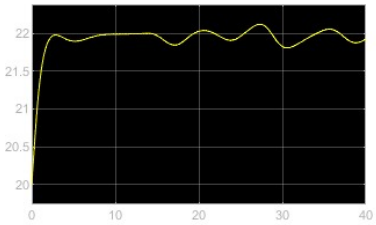


Figure 10: PID Angular Speed

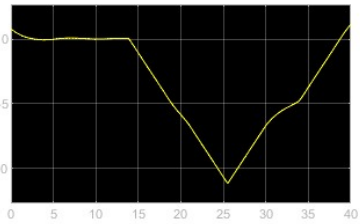


Figure 11: PID Indicator

The linear speed increases gradually in a controlled manner; the robot starts from a standstill and gradually accelerates until it reaches the predefined linear speed specified in the code. We can also observe fluctuations in the robot's angular speed as it turns and adapts its trajectory.

3 Obstacle avoidance based on limit-cycle method, and multi-controller architecture

To demonstrate using the Lyapunov stability theorem that the proposed control law (equation 9) guarantees the convergence of the orientation error θ_e towards zero, we can follow these steps:

Define a Lyapunov function $V(\theta_e)$ as:

$$V(\theta_e) = \frac{1}{2}\theta_e^2$$

Now, calculate the time derivative of V along the trajectory of the system using the chain rule:

$$\dot{V} = \frac{dV}{d\theta_e} \cdot \frac{d\theta_e}{dt}$$

Substitute the given control law (equation 9) for $\dot{\theta_e}$:

$$\dot{V} = \theta_e \cdot (-k_1 \sin(\theta_e))$$

To apply the Lyapunov stability theorem, we need to show that $\dot{V}$ is negative definite or, in other words, $\dot{V} < 0$. To do this, consider that θ_e is the orientation error, and k_1 is a positive control gain.

Since θ_e represents the error between the desired orientation and the current orientation, $\sin(\theta_e)$ will always have a magnitude less than or equal to 1. Therefore, $\dot{V}$ will always be negative or zero, indicating that the error θ_e is guaranteed to converge to zero, ensuring stability.

This demonstrates that the control law, as proposed in equation 9, guarantees the convergence of the orientation error θ_e towards zero using the Lyapunov stability theorem.

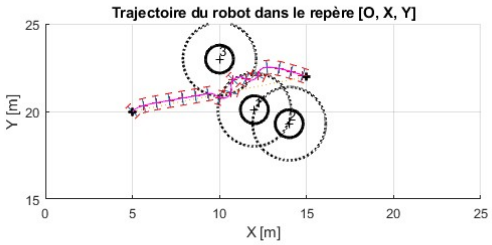


Figure 12: Control law (9) with 1 more obstacle

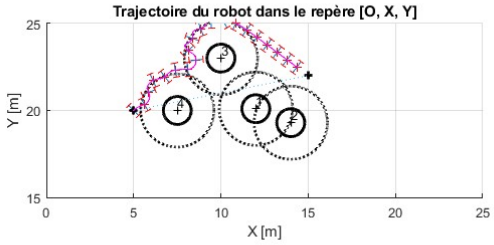


Figure 13: Control law (9) with 2 more obstacle

In the presence of obstacles in the robot's path, it is initially observed that the robot intrudes into the boundary region of the obstacle and attempts to maintain an orbit around it. More specifically, it oscillates due to the ON/OFF control function employed; either the robot is in target attraction mode or in obstacle avoidance mode.

When additional intermediate obstacles are introduced, it quickly becomes evident that the robot's path is neither optimal nor close to it. However, this method works exceptionally well without global vision because it operates instantaneously and is very quick to execute.

When the robot navigates between two obstacles that share a boundary region, it behaves appropriately by passing closest between the two. With its ON/OFF attraction system, it stays very close to the orbit of both obstacles.

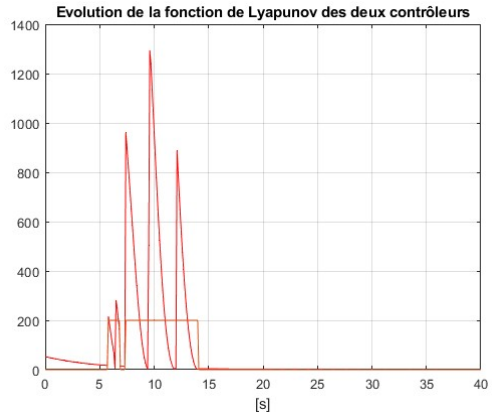


Figure 14: Control law (9) with 1 more obstacle

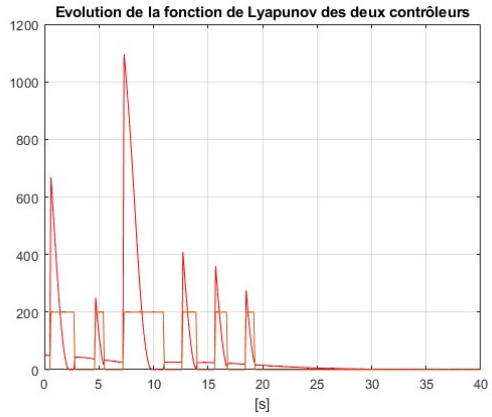


Figure 15: Control law (9) with 2 more obstacle

Analyzing the stability of the multi-controller architecture based on the figures 14 & 15 representing the evolution of Lyapunov functions with respect to time, we observe:

In the first figure, when passing between two obstacles at their boundary, the robot oscillates between them. The Lyapunov function reveals four state changes between Attraction and Avoidance modes. This oscillation in the function has a high amplitude but a low frequency.

In the second figure, with two additional obstacles, the robot chooses to navigate above them. Here, 12 state changes between Attraction and Avoidance are noticeable. The Lyapunov function in this case oscillates with a diminishing amplitude and a frequency higher than in the first scenario.

The stability of the multi-controller architecture, as evidenced by the evolution of the Lyapunov functions over time, seems to be influenced by the number and positioning of obstacles. When the robot navigates between closely placed obstacles, it exhibits oscillatory behavior, leading to multiple state changes between Attraction and Avoidance modes. This oscillatory nature, characterized by varying amplitude and frequency, raises questions about the inherent stability of the system. The more obstacles introduced, the higher the frequency of state changes, suggesting that the architecture may require further refinement to ensure consistent and robust stability, especially in complex environments.

4 Cognitive long-term planning

Executing the original code without any modifications yields the following resultant paths. It's worth noting that altering the number of iterations doesn't necessarily guarantee a better result; surprisingly, a lower number of iterations often aids in achieving a more reliable outcome.

Additionally, establishing the correct interval for the 'mu' parameter is crucial for obtaining a globally optimal curvature path. As demonstrated in this code, the effectiveness of the algorithm is closely tied to this parameter tuning process.

The green path is often considered the "global optimal path, denoted as gLC*. However, we believe that this may not always be the case for several reasons. It's essential to note that the robot maintains a fixed linear speed, which means that the shortest path is generally the quickest. Some segments of the green path, particularly from point A to point B, could potentially be traversed more efficiently by simply calculating the Euclidean distance between these points rather than strictly following a curved path, which may be longer and slower.

In our algorithm, we aim to systematically identify the shortest path, referred to as the global optimal gLC*. To achieve this, our algorithm explores a decision tree, and we need to introduce additional decision possibilities:

1. **Euclidean Path:** If it's feasible to create a direct Euclidean path between two obstacles without intersecting any unsafe zones, then this path should be selected as a viable option.
2. **Curvature Path:** In cases where a curvature path is necessary due to obstacles or constraints, our algorithm aims to find the optimal parameter, denoted as 'μ,' which minimizes the maximum curvature length along the path. To achieve this, we propose a hybrid method that initially identifies a lower bound for 'μ' within a broad interval. Subsequently, it employs a dichotomy approach with a smaller step size to fine-tune and determine the optimal 'μ' after a predetermined number of iterations.

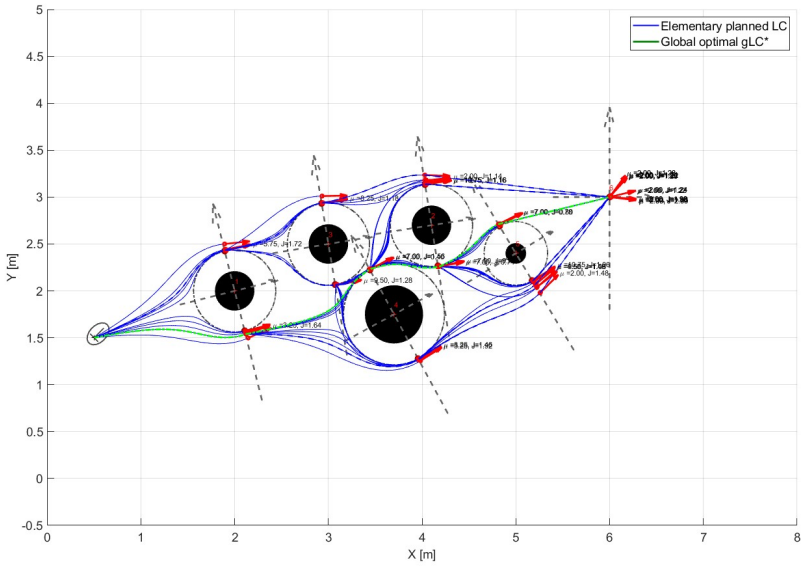


Figure 16: Global optimal gLC* path with 5 obstacles

By incorporating these decision possibilities into our algorithm, we can systematically identify the most efficient and optimal path, considering both Euclidean and curvature paths, while taking into account the robot's fixed linear speed. However, it's important to note that while our algorithm aims to approach the optimal value of 'mu' to minimize curvature length, the precision of this approach depends on the chosen number of iterations

References

[1] L. Adouane. Orbital Obstacle Avoidance Algorithm for Reliable and On-Line Mobile Robot Navigation In *Robotica'09, Castelo-Branco Portugal, appeared in Portuguese Journal "Robótica"*. July 2010.

[2] L. Adouane. Reactive versus cognitive vehicle navigation based on optimal local and global PELC*. In *Robotics and Autonomous Systems (RAS), DOI 10.1016/j.robot.2016.11.006*, volume 88, pp. 51–70, February 2017.

[3] Vikram Balaji, M.Balaji, M.Chandrasekaran, M.K.A.Ahamed khan, Irraivan Elamvazuthi ptimization of PID Control for High Speed Line Tracking Robots In *2015 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS 2015)*