جامعة سيدي محمد بن عبد الله

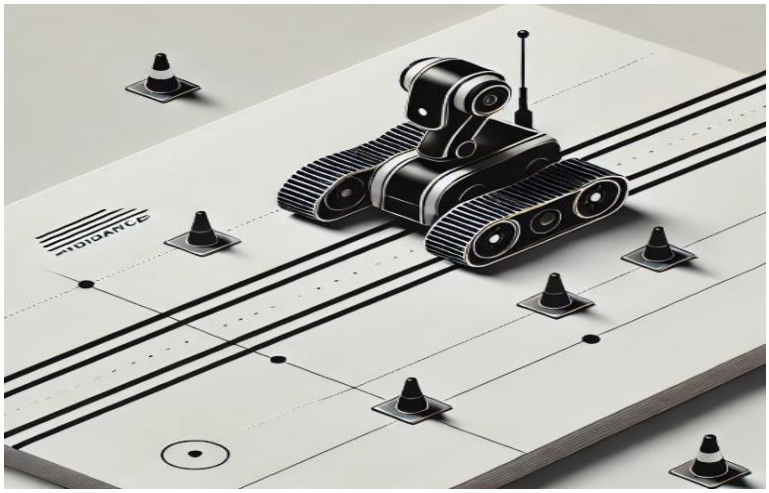
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⵎⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ

Université Sidi Mohamed Ben Abdellah

ENSA Fès

# Rapport de Projet ROS2 :

# Robot Suiveur de Ligne avec Évitement d'Obstacles



**préparé par :** EL Kouche Hafsa N :19 , Meslaha Imane N : 28

**encadrée par :** M.Taoufik Belkebir

**Année Universitaire : 2024-2025**

# 1. Introduction :

Ce rapport présente le développement d'un robot suiveur de ligne avec évitement d'obstacles utilisant ROS2 et Python. Le projet s'articule autour de la création de quatre packages distincts qui interagissent pour assurer la détection des obstacles, le suivi de la ligne, et le contrôle de mouvement du robot.

# 2. Architecture du Projet :

Le projet est structuré en quatre **packages** ROS2 principaux :

**1. sensor_handler :** Gestion des capteurs.

**2. motion_controller :** Contrôle du mouvement du robot.

**3. main_controller :** Coordination principale des différents nœuds.

**4. line_follower :** Suivi de ligne à l'aide de traitement d'images.

Chaque package est conçu avec des nœuds ROS2 individuels, un fichier launch, ainsi qu'une configuration de `package.xml` et `setup.py`.

# 3. Description des Packages

## 3.1 Package : sensor_handler

**Objectif :** Capturer les données des capteurs et les republier pour qu'elles soient utilisées par d'autres nœuds.

**Nœud principal : sensor_node.py**

src > sensor_handlor > sensor_handlor > sensor_node.py

```python
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan

class SensorHandler(Node):
    def __init__(self):
        super().__init__('sensor_node')
        self.scan_pub = self.create_publisher(LaserScan, 'scan_data', 10)
        self.scan_sub = self.create_subscription(
            LaserScan,
            'scan',
            self.scan_callback,
            10)

    def scan_callback(self, msg):
        self.scan_pub.publish(msg)


def main(args=None):
    rclpy.init(args=args)
    node = SensorHandler()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```



**Configuration :** Fichiers `package.xml`, `setup.py` et `launch` créés et configurés pour le bon fonctionnement du package.

## 3.2 Package : motion_controller

**Objectif :** Assurer le contrôle de mouvement du robot en utilisant les données des capteurs.

## Nœud principal : `motion_node.py

src > motion_controller > motion_controller > motion_node.py

```python
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan

class MotionController(Node):
    def __init__(self):
        super().__init__('motion_node')
        self.cmd_pub = self.create_publisher(Twist, 'cmd_vel', 10)
        self.scan_sub = self.create_subscription(
            LaserScan,
            'scan_data',
            self.sensor_callback,
            10)

        self.line_error_sub = self.create_subscription(
            Float32,
            '/line_error',
            self.line_error_callback,
            10)

        self.cmd = Twist()
        self.min_distance = 0.35
        self.side_threshold = 0.15
        self.left_side = 0.0
        self.front = 0.0
        self.right_side = 0.0

    def sensor_callback(self, msg):
        self.left_side = msg.ranges[230]
        self.front = msg.ranges[360]
        self.right_side = msg.ranges[490]
        self.command_publisher()

    def line_error_callback(self, msg):
        # Update the line error
        self.line_error = msg.data

    def command_publisher(self):
        linear_vel = 0.075
        if self.left_side < self.side_threshold and self.front > self.min_distance:
            self.cmd.linear.x = linear_vel * 0.5
            self.cmd.angular.z = -0.15
        elif self.right_side < self.side_threshold and self.front > self.min_distance:
            self.cmd.linear.x = linear_vel * 0.5
            self.cmd.angular.z = 0.15
        elif self.front > self.min_distance:
            self.cmd.linear.x = linear_vel
            self.cmd.angular.z = 0.0
        elif self.front < self.min_distance:
            if self.compare_sides(self.left_side,self.right_side):
                self.cmd.linear.x = linear_vel * 0.25
                self.cmd.angular.z = 0.35
```

```
                else:
                    self.cmd.linear.x = linear_vel * 0.25
                    self.cmd.angular.z = -0.35
            else:
                # Line following logic
                self.cmd.linear.x = 0.2
                self.cmd.angular.z = -1.5 / 100 * self.line_error


            self.cmd_pub.publish(self.cmd)


    def main(args=None):
        rclpy.init(args=args)
        node = MotionController()
        rclpy.spin(node)
        node.destroy_node()
        rclpy.shutdown()

    if __name__ == '__main__':
        main()
```

## 3.3 Package : main_controller

**Objectif :** Coordonner les opérations des différents nœuds et lancer les services associés.

**Nœud principal : `patrol_node.py`**

src > main_controller > main_controller > patrol_node.py

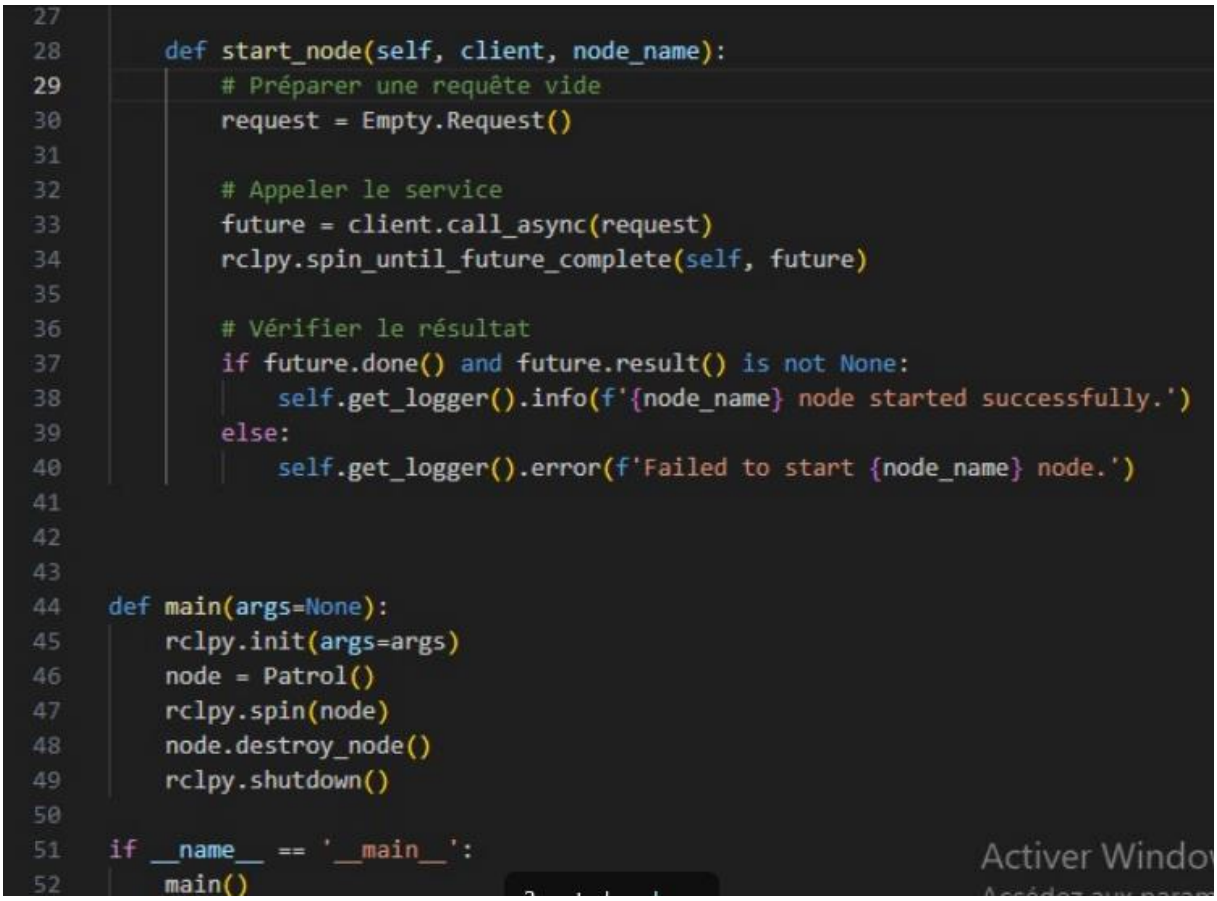
```python
import rclpy
from rclpy.node import Node
from std_srvs.srv import Empty

class Patrol(Node):
    def __init__(self):
        super().__init__('patrol_node')
        self.start_service = self.create_service(Empty, 'start_patrol', self.start_callback)
        # Clients pour démarrer les autres nœuds
        self.image_processor_client = self.create_client(Empty, 'start_image_processor')
        self.motion_controller_client = self.create_client(Empty, 'start_motion_controller')

        # Assurez-vous que les services sont disponibles
        while not self.image_processor_client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Waiting for "start_image_processor" service...')
        while not self.motion_controller_client.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Waiting for "start_motion_controller" service...')


    def start_callback(self, request, response):
        self.get_logger().info('Patrol started')
        # Appel des services pour démarrer les nœuds
        self.start_node(self.image_processor_client, 'Image Processor')
        self.start_node(self.motion_controller_client, 'Motion Controller')

        return response
```



```
    def start_node(self, client, node_name):
        # Préparer une requête vide
        request = Empty.Request()

        # Appeler le service
        future = client.call_async(request)
        rclpy.spin_until_future_complete(self, future)

        # Vérifier le résultat
        if future.done() and future.result() is not None:
            self.get_logger().info(f'{node_name} node started successfully.')
        else:
            self.get_logger().error(f'Failed to start {node_name} node.')



def main(args=None):
    rclpy.init(args=args)
    node = Patrol()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```



**Fonctions principales :**

- Lancer et arrêter les services.

- Assurer la communication entre les différents nœuds.

## 3.4 Package : line_follower

**Objectif :** Suivi de ligne à l'aide d'images et correction d'orientation en fonction des erreurs détectées.

**Nœuds :**

**- `image_processor.py` :**

Traitement d'image pour détecter la ligne.

src > line_follower > line_follower > image_processor.py

```python
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from std_msgs.msg import Float32
from std_srvs.srv import Empty
from cv_bridge import CvBridge
import cv2
import numpy as np

class ImageProcessor(Node):
    def __init__(self):
        super().__init__('image_processor')
        self.error_publisher = self.create_publisher(Float32, '/line_error', 10)
        self.subscription = None  # Subscription inactive par défaut
        self.bridge = CvBridge()

        # Service pour activer le nœud
        self.start_service = self.create_service(Empty, 'start_image_processor', self.start_
```



```python
    def start_callback(self, request, response):
        if not self.subscription:
            self.subscription = self.create_subscription(
                Image,
                '/camera/image_raw',
                self.listener_callback,
                10)
            self.get_logger().info('Image Processor started.')
        else:
            self.get_logger().info('Image Processor is already running.')
        return response

    def listener_callback(self, data):
        current_frame = self.bridge.imgmsg_to_cv2(data, desired_encoding='bgr8')
        hsv_image = cv2.cvtColor(current_frame, cv2.COLOR_BGR2HSV)

        lower_blue = np.array([100, 50, 50])
        upper_blue = np.array([130, 255, 255])
        blue_mask = cv2.inRange(hsv_image, lower_blue, upper_blue)

        contours, _ = cv2.findContours(blue_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

                if M['m00'] > 0:
                    centroid_x = int(M["m10"] / M["m00"])
                    error = centroid_x - (current_frame.shape[1] // 2)

            error_msg = Float32()
            error_msg.data = float(error)
            self.error_publisher.publish(error_msg)

def main(args=None):
    rclpy.init(args=args)
    node = ImageProcessor()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

**- `motion_controller_node.py` :**

Ajuster le mouvement en fonction des erreurs de suivi de ligne.

src > line_follower > line_follower > motion_controller_node.py

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32
from geometry_msgs.msg import Twist
from std_srvs.srv import Empty

LINEAR_SPEED = 0.2
KP = 1.5 / 100

class MotionController(Node):
    def __init__(self):
        super().__init__('motion_controller')
        self.cmd_publisher = self.create_publisher(Twist, '/cmd_vel', 10)
        self.error_subscription = None  # Subscription inactive par défaut

        # Service pour activer le nœud
        self.start_service = self.create_service(Empty, 'start_motion_controller', self.start

    def start_callback(self, request, response):
        if not self.error_subscription:
            self.error_subscription = self.create_subscription(
                Float32,
                '/line_error',
                self.listener_callback,
                10)
            self.get_logger().info('Motion Controller started.')
        else:
            self.get_logger().info('Motion Controller is already running.')
        return response

    def listener_callback(self, error_msg):
        error = error_msg.data
        cmd = Twist()
        cmd.linear.x = LINEAR_SPEED
        cmd.angular.z = -KP * error
        self.cmd_publisher.publish(cmd)

def main(args=None):
    rclpy.init(args=args)
    node = MotionController()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

## 4. Conclusion :

Ce projet ROS2 démontre l'intégration de plusieurs nœuds et packages pour construire un robot capable de suivre une ligne tout en évitant les obstacles. Les packages sont conçus de manière modulaire, facilitant la maintenance et l'évolution future du projet.