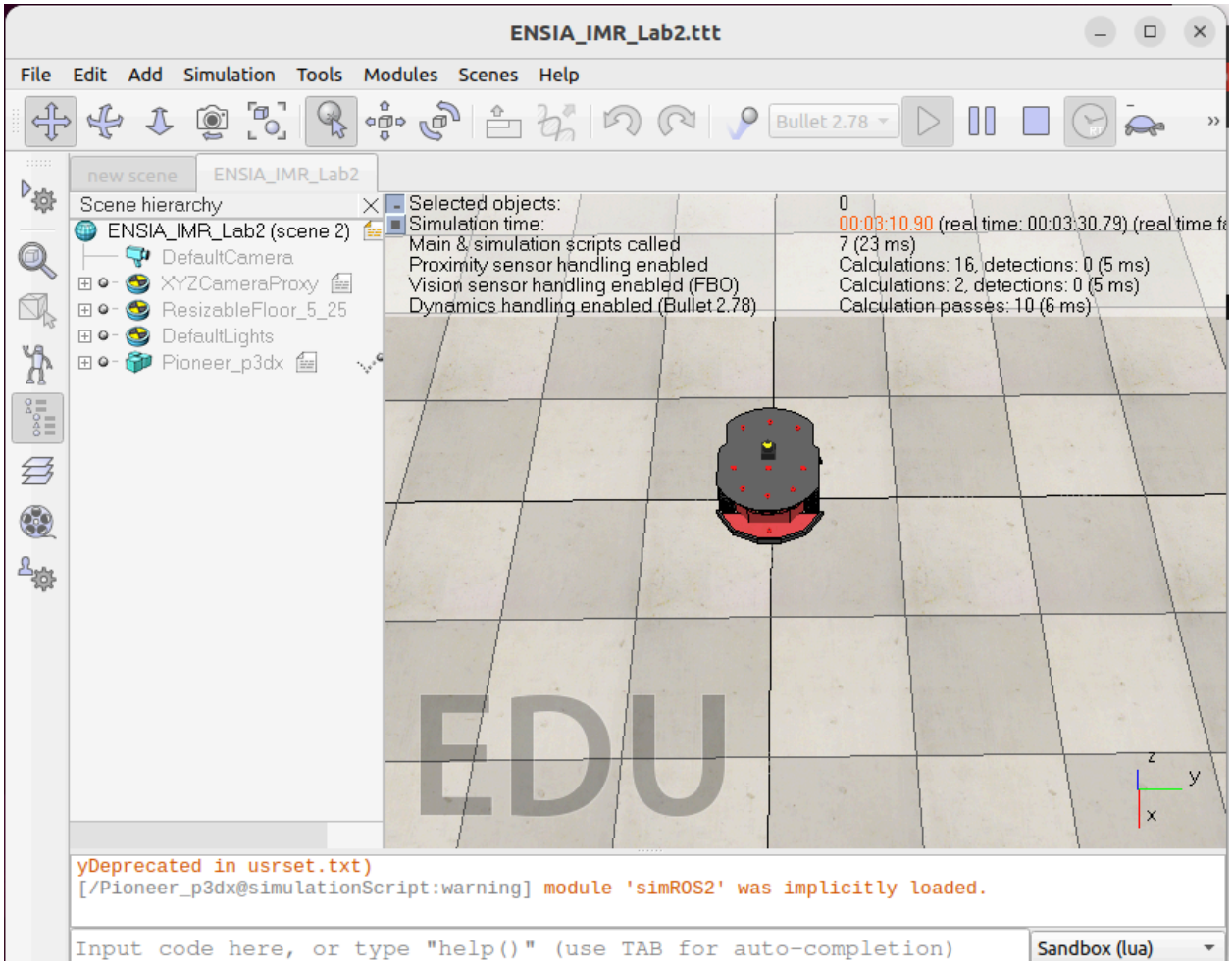
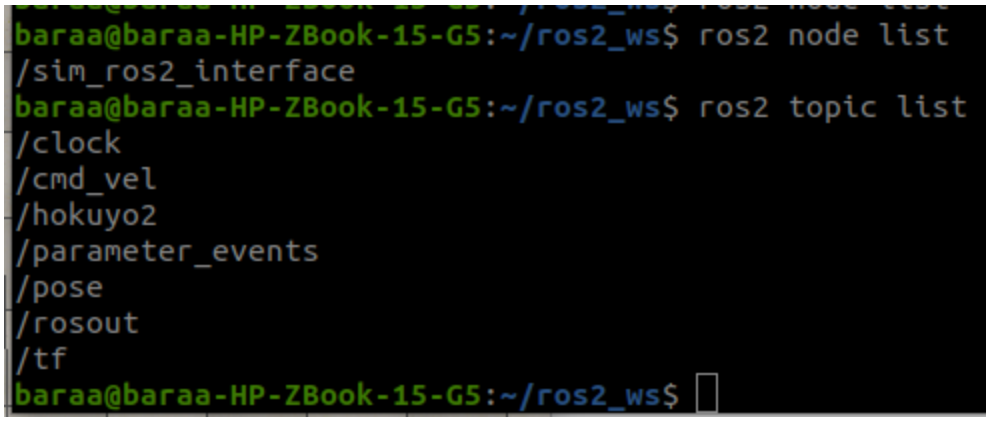
Mobile Robotics
LAB 02

CoppeliaSimulator and ROS2

- Uploading the scene into coppeliaSim



- Checking the active nodes and topics:

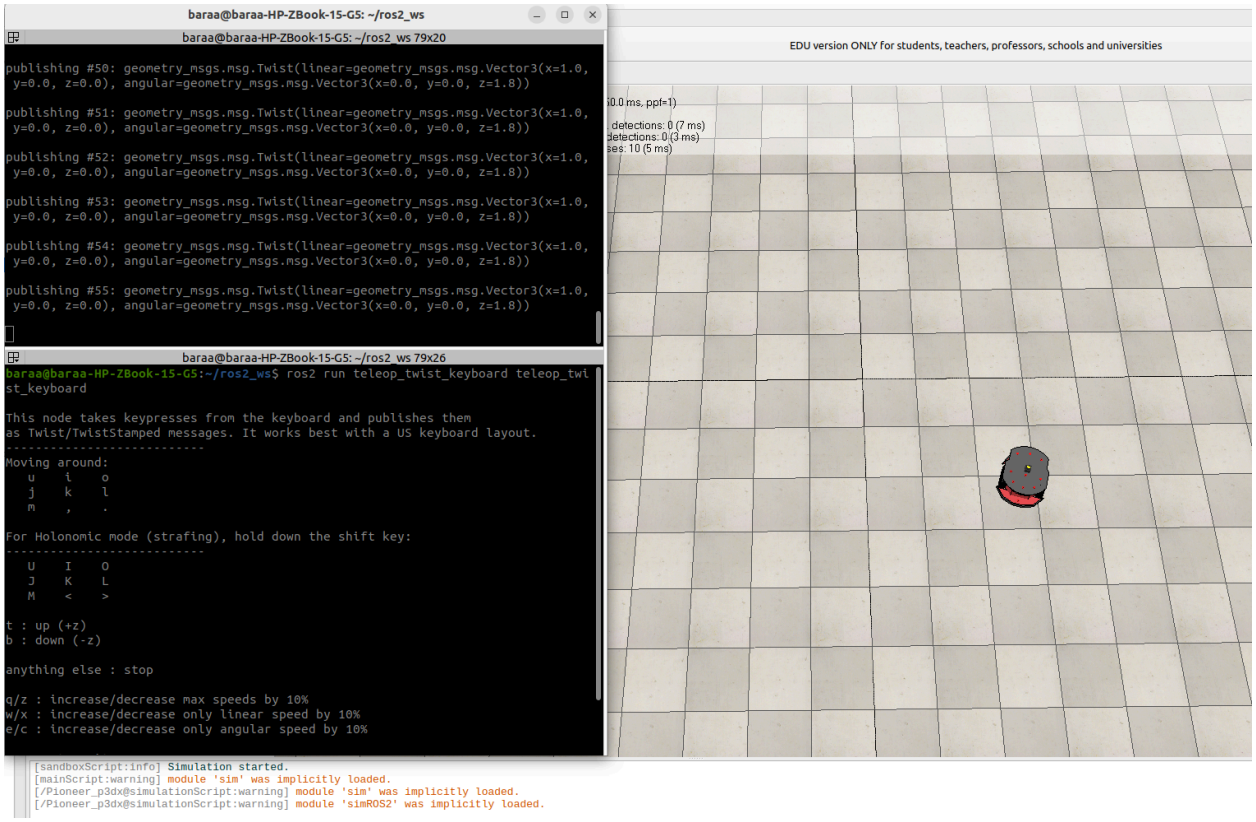


Open Loop Control

- Deriving phi_r(.) and phi_l(.)
  - phi_dot_r = (2* linear_cmd_vel + ROBOT_WIDTH*angular_cmd_vel) / 2*WHEEL_WIDTH
  - phi_dot_l=(2* linear_cmd_vel - ROBOT_WIDTH*angular_cmd_vel) / 2*WHEEL_WIDTH

```
function setVelocity_cb(msg)
    linear_cmd_vel = msg.linear.x
    angular_cmd_vel = msg. angular.z

    --  Calculate the wheel spinning speeds phi_dot_R and phi_dot_L
    --  using the provided linear and angular velocities of the robot.

    phi_dot_L = (2 * linear_cmd_vel + ROBOT_WIDTH * angular_cmd_vel) /  (2 * WHEEL_RADIUS)
    phi_dot_R = (2 * linear_cmd_vel - ROBOT_WIDTH * angular_cmd_vel) /  (2 * WHEEL_RADIUS)

    sim.setJointTargetVelocity(left_wheel, phi_dot_L )
    sim.setJointTargetVelocity(right_wheel, phi_dot_R)

end
```

- Starting teleop_keyboard twist and moving the pioneer using the keyboard:



Closed Loop Control

- Making the pioneer controller package

The subscriber/ publisher code:

```
#include <rclcpp/rclcpp.hpp>
#include <geometry_msgs/msg/twist.hpp>
#include <geometry_msgs/msg/pose2_d.hpp>  // For Pose2D
#include <geometry_msgs/msg/pose.hpp>     // For Pose
#include <cmath>

class PoseController : public rclcpp::Node {
public:
  PoseController() : Node("pose_controller") {
    // Declare and get parameters for the goal pose
    this->declare_parameter("goal_x", 5.0);
    this->declare_parameter("goal_y", 5.0);
    this->declare_parameter("goal_theta", 0.0);
    this->declare_parameter("k_rho", 1.0);
    this->declare_parameter("k_alpha", 2.0);
    this->declare_parameter("k_beta", -1.0);

    goal_x_ = this->get_parameter("goal_x").as_double();
    goal_y_ = this->get_parameter("goal_y").as_double();
    goal_theta_ = this->get_parameter("goal_theta").as_double();
    k_rho_ = this->get_parameter("k_rho").as_double();
    k_alpha_ = this->get_parameter("k_alpha").as_double();
    k_beta_ = this->get_parameter("k_beta").as_double();

    // Subscriber for Pose2D (x, y, theta)
    pose2d_subscriber_ = this->create_subscription<geometry_msgs::msg::Pose2D>(
```

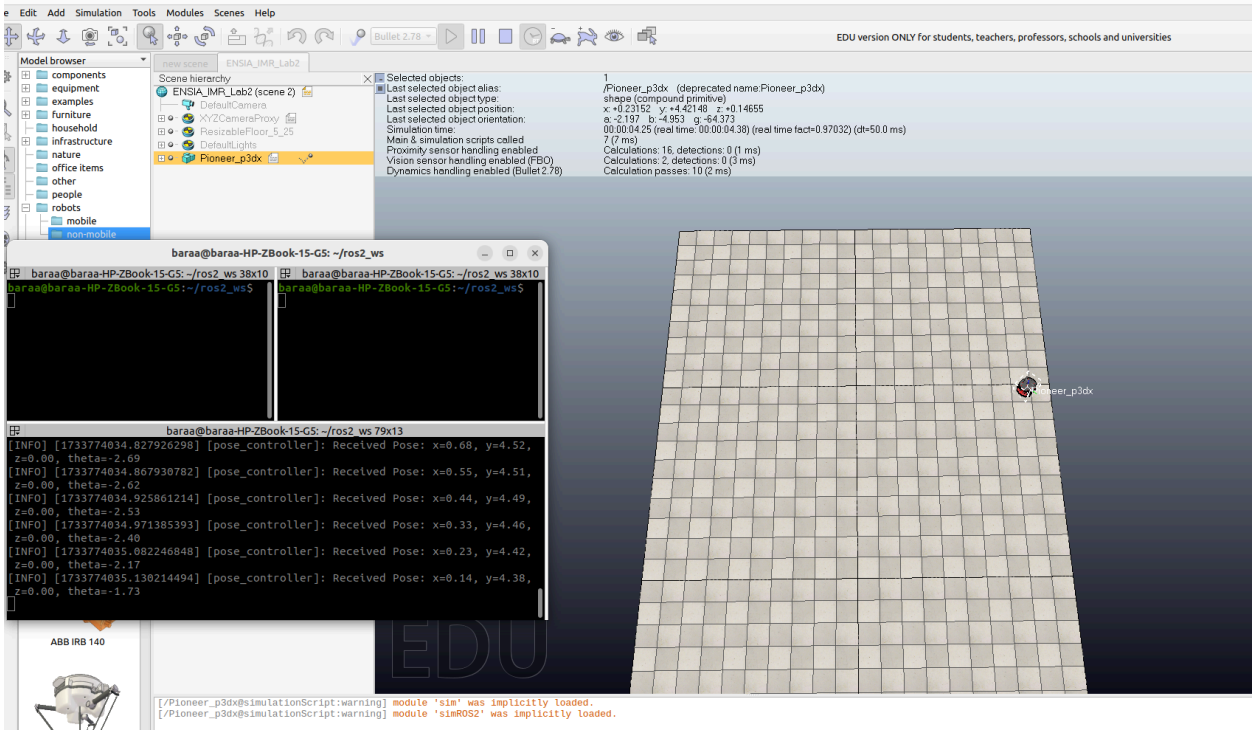
```
      "/pose2d", 10, std::bind(&PoseController::pose2dCallback, this,
std::placeholders::_1));

    // Subscriber for Pose (x, y, z, quaternion)
    pose_subscriber_ = this->create_subscription<geometry_msgs::msg::Pose>(
      "/pose", 10, std::bind(&PoseController::poseCallback, this, std::placeholders::_1));

    // Publisher for velocity commands
    velocity_publisher_ = this->create_publisher<geometry_msgs::msg::Twist>("/cmd_vel",
10);

    RCLCPP_INFO(this->get_logger(), "Pose Controller Node initialized");
  }

private:
  // Callback for Pose2D messages (x, y, theta)
  void pose2dCallback(const geometry_msgs::msg::Pose2D::SharedPtr pose) {
    RCLCPP_INFO(this->get_logger(), "Received Pose2D: x=%.2f, y=%.2f, theta=%.2f",
pose->x, pose->y, pose->theta);

    // Calculate errors
    double dx = goal_x_ - pose->x;
    double dy = goal_y_ - pose->y;
    double rho = std::sqrt(dx * dx + dy * dy);  // Distance to goal
    double alpha = std::atan2(dy, dx) - pose->theta;  // Angle to goal
    double beta = goal_theta_ - pose->theta - alpha;  // Orientation error

    // Normalize angles
    alpha = normalizeAngle(alpha);
    beta = normalizeAngle(beta);

    // Stop if goal is reached
    if (rho < 0.1) {
      publishVelocity(0.0, 0.0);
```

```
        RCLCPP_INFO(this->get_logger(), "Goal Reached!");
        return;
    }

    // Control law
    double linear_velocity = k_rho_ * rho;
    double angular_velocity = k_alpha_ * alpha + k_beta_ * beta;

    // Publish velocity commands
    publishVelocity(linear_velocity, angular_velocity);
  }

  // Callback for Pose messages (x, y, z, quaternion)
  void poseCallback(const geometry_msgs::msg::Pose::SharedPtr pose) {
    double x = pose->position.x;
    double y = pose->position.y;
    double z = pose->position.z;
    double theta = yawFromQuaternion(pose->orientation);

    RCLCPP_INFO(this->get_logger(), "Received Pose: x=%.2f, y=%.2f, z=%.2f, theta=%.2f",
x, y, z, theta);

    // Calculate errors
    double dx = goal_x_ - x;
    double dy = goal_y_ - y;
    double rho = std::sqrt(dx * dx + dy * dy);  // Distance to goal
    double alpha = std::atan2(dy, dx) - theta;  // Angle to goal
    double beta = goal_theta_ - theta - alpha;  // Orientation error

    // Normalize angles
    alpha = normalizeAngle(alpha);
    beta = normalizeAngle(beta);

    // Stop if goal is reached
```

```
  if (rho < 0.1) {
    publishVelocity(0.0, 0.0);
    RCLCPP_INFO(this->get_logger(), "Goal Reached!");
    return;
  }

  // Control law
  double linear_velocity = k_rho_ * rho;
  double angular_velocity = k_alpha_ * alpha + k_beta_ * beta;

  // Publish velocity commands
  publishVelocity(linear_velocity, angular_velocity);
}

// Helper function to publish velocity commands
void publishVelocity(double linear, double angular) {
  auto cmd_vel_msg = geometry_msgs::msg::Twist();
  cmd_vel_msg.linear.x = linear;
  cmd_vel_msg.angular.z = angular;
  velocity_publisher_->publish(cmd_vel_msg);
}

// Helper function to normalize angles to [-pi, pi]
double normalizeAngle(double angle) {
  while (angle > M_PI) angle -= 2.0 * M_PI;
  while (angle < -M_PI) angle += 2.0 * M_PI;
  return angle;
}

// Helper function to extract yaw from quaternion (for Pose message)
double yawFromQuaternion(const geometry_msgs::msg::Quaternion &q) {
  return std::atan2(2.0 * (q.w * q.z + q.x * q.y),
                    1.0 - 2.0 * (q.y * q.y + q.z * q.z));
}
```

```
  // ROS 2 subscribers and publisher
  rclcpp::Subscription<geometry_msgs::msg::Pose2D>::SharedPtr pose2d_subscriber_;
  rclcpp::Subscription<geometry_msgs::msg::Pose>::SharedPtr pose_subscriber_;
  rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr velocity_publisher_;

  // Parameters
  double goal_x_, goal_y_, goal_theta_;
  double k_rho_, k_alpha_, k_beta_;
};

// Main function
int main(int argc, char *argv[]) {
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<PoseController>());
  rclcpp::shutdown();
  return 0;
}
```

- Simulation: (the pioneer went crazy and fell)



Closed-loop Control Enhanced

- The enhanced closed loop code:

```
#include <rclcpp/rclcpp.hpp>
#include <geometry_msgs/msg/twist.hpp>
#include <geometry_msgs/msg/pose2_d.hpp>
#include <geometry_msgs/msg/pose.hpp>
#include <cmath>

class EnhancedPoseController : public rclcpp::Node {
public:
  EnhancedPoseController() : Node("enhanced_pose_controller") {
    // Declare and get parameters for the goal pose
    this->declare_parameter("goal_x", 5.0);
    this->declare_parameter("goal_y", 5.0);
    this->declare_parameter("goal_theta", 0.0);
    this->declare_parameter("constant_speed", 0.5);
    this->declare_parameter("k_alpha", 2.0);
```

```
    this->declare_parameter("k_beta", -1.0);

    goal_x_ = this->get_parameter("goal_x").as_double();
    goal_y_ = this->get_parameter("goal_y").as_double();
    goal_theta_ = this->get_parameter("goal_theta").as_double();
    constant_speed_ = this->get_parameter("constant_speed").as_double();
    k_alpha_ = this->get_parameter("k_alpha").as_double();
    k_beta_ = this->get_parameter("k_beta").as_double();

    // Subscriber for Pose2D (x, y, theta)
    pose2d_subscriber_ = this->create_subscription<geometry_msgs::msg::Pose2D>(
      "/pose2d", 10, std::bind(&EnhancedPoseController::pose2dCallback, this,
std::placeholders::_1));

    // Subscriber for Pose (x, y, z, quaternion)
    pose_subscriber_ = this->create_subscription<geometry_msgs::msg::Pose>(
      "/pose", 10, std::bind(&EnhancedPoseController::poseCallback, this,
std::placeholders::_1));

    // Publisher for velocity commands
    velocity_publisher_ = this->create_publisher<geometry_msgs::msg::Twist>("/cmd_vel",
10);

    RCLCPP_INFO(this->get_logger(), "Enhanced Pose Controller Node initialized");
  }

private:
  // Callback for Pose2D messages (x, y, theta)
  void pose2dCallback(const geometry_msgs::msg::Pose2D::SharedPtr pose) {
    RCLCPP_INFO(this->get_logger(), "Received Pose2D: x=%.2f, y=%.2f, theta=%.2f",
pose->x, pose->y, pose->theta);

    // Calculate errors
    double dx = goal_x_ - pose->x;
```

```
  double dy = goal_y_ - pose->y;
  double rho = std::sqrt(dx * dx + dy * dy);  // Distance to goal
  double alpha = std::atan2(dy, dx) - pose->theta;  // Angle to goal
  double beta = goal_theta_ - pose->theta - alpha;  // Orientation error

  // Normalize angles
  alpha = normalizeAngle(alpha);
  beta = normalizeAngle(beta);

  // Control law for constant forward speed
  double linear_velocity = constant_speed_;
  double angular_velocity = k_alpha_ * alpha + k_beta_ * beta;

  // If the robot is close to the goal, stop it
  if (rho < 0.1) {
    publishVelocity(0.0, 0.0);
    RCLCPP_INFO(this->get_logger(), "Goal Reached!");
    return;
  }

  // Adjust the direction (forward or backward) based on position relative to goal
  if (rho < 0.0) {
    linear_velocity = -constant_speed_;  // Reverse direction
  }

  // Publish velocity commands
  publishVelocity(linear_velocity, angular_velocity);
}

// Callback for Pose messages (x, y, z, quaternion)
void poseCallback(const geometry_msgs::msg::Pose::SharedPtr pose) {
  double x = pose->position.x;
  double y = pose->position.y;
  double z = pose->position.z;
```

```
    double theta = yawFromQuaternion(pose->orientation);

    RCLCPP_INFO(this->get_logger(), "Received Pose: x=%.2f, y=%.2f, z=%.2f, theta=%.2f",
x, y, z, theta);

    // Calculate errors
    double dx = goal_x_ - x;
    double dy = goal_y_ - y;
    double rho = std::sqrt(dx * dx + dy * dy);  // Distance to goal
    double alpha = std::atan2(dy, dx) - theta;  // Angle to goal
    double beta = goal_theta_ - theta - alpha;  // Orientation error

    // Normalize angles
    alpha = normalizeAngle(alpha);
    beta = normalizeAngle(beta);

    // Control law for constant forward speed
    double linear_velocity = constant_speed_;
    double angular_velocity = k_alpha_ * alpha + k_beta_ * beta;

    // If the robot is close to the goal, stop it
    if (rho < 0.1) {
      publishVelocity(0.0, 0.0);
      RCLCPP_INFO(this->get_logger(), "Goal Reached!");
      return;
    }

    // Adjust the direction (forward or backward) based on position relative to goal
    if (rho < 0.0) {
      linear_velocity = -constant_speed_;  // Reverse direction
    }

    // Publish velocity commands
    publishVelocity(linear_velocity, angular_velocity);
```

```
  }

  // Helper function to publish velocity commands
  void publishVelocity(double linear, double angular) {
    auto cmd_vel_msg = geometry_msgs::msg::Twist();
    cmd_vel_msg.linear.x = linear;
    cmd_vel_msg.angular.z = angular;
    velocity_publisher_->publish(cmd_vel_msg);
  }

  // Helper function to normalize angles to [-pi, pi]
  double normalizeAngle(double angle) {
    while (angle > M_PI) angle -= 2.0 * M_PI;
    while (angle < -M_PI) angle += 2.0 * M_PI;
    return angle;
  }

  // Helper function to extract yaw from quaternion (for Pose message)
  double yawFromQuaternion(const geometry_msgs::msg::Quaternion &q) {
    return std::atan2(2.0 * (q.w * q.z + q.x * q.y),
                      1.0 - 2.0 * (q.y * q.y + q.z * q.z));
  }

  // ROS 2 subscribers and publisher
  rclcpp::Subscription<geometry_msgs::msg::Pose2D>::SharedPtr pose2d_subscriber_;
  rclcpp::Subscription<geometry_msgs::msg::Pose>::SharedPtr pose_subscriber_;
  rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr velocity_publisher_;

  // Parameters
  double goal_x_, goal_y_, goal_theta_;
  double constant_speed_;
  double k_alpha_, k_beta_;
};
```

```
// Main function
int main(int argc, char *argv[]) {
   rclcpp::init(argc, argv);
   rclcpp::spin(std::make_shared<EnhancedPoseController>());
   rclcpp::shutdown();
   return 0;
}
```

Simulation (the pioneer did not fall)