# Project description

In this project, multiple robots will be navigating through a busy warehouse autonomously by communicating with each other in a decentralised manner (no central system for communication, i.e. peer-to-peer). This will happen in a safe manner as the paths of the robots will be projected for humans to see. In this document we will be discussing the problem in this project and how we're going to tackle this problem. We will also discuss the project requirements, phases and risks.

**Priority-Based Navigation (PBN)**

Because ROS2 is a decentralised system, there is no server that knows the positions of all robots and mediates the communication between them. If any two robots need to communicate, they have to find each other first. This means that when robots plan their path, they cannot take into account other robots’ paths and priorities.

The problem occurs when the paths of two robots cross in a way that could lead to a collision. Obstacle avoidance can make sure that the robots do not collide, but it does not take into account their priorities. If one robot has a higher priority than the other, then obstacle avoidance is not sufficient because it may unnecessarily slow down the high priority robot.

**Human Understandable Behaviour (HUB)**

The second part addresses a problem that occurs when humans and robots navigate in close proximity. For example in figure 3.1, the warehouse employee might assume the robot is following route number 3 and correct their path accordingly while the robot actually follows path number 1. In the worst case it will lead to an impact, but even if a collision is avoided, it can startle the employee, cause the robot to stop abruptly, replan and waste time.

Shape, rectangle

Description automatically generated

If the employee is unsure of what path the robot will take, they might stop and wait until it becomes apparent which is also inefficient and tiresome.

It is also possible that the employee does not see the robot until it is too late or that they have reduced mobility because they are driving a forklift or are in a wheelchair for instance.

# System description

This system consists of 4 main packages: **Dede, hupbrb, path\_projection** and **priority\_based\_robot\_costmap\_plugin**. There is also a **hupbrb\_msgs** package which contains the custom messages this system uses to communicate between nodes. Each of these packages has its own responsibility.

## Dede

The dede package contains information about the robot this project is using. It also contains the simulated worlds used for testing. In the dede package, there are multiple important launch files:

* **Simulation.launch.py**

This launch file will start gazebo with a specified world and spawn a dede robot.

* **Nav2\_slam.launch.py**

This launch file is for starting the navigation of a singular robot. This file needs a map.data file as a parameter. You can find these files in dede/maps or create your own using the dede/launch/**slam\_toolbox.launch.py**.

* **Rviz2.launch.py**

This file launches Rviz2 with a specified namespace.

* **Multi\_simulation.launch.py**

This launch file starts gazebo with a specified world and spawns multiple dede robots. (2 at the time of writing this document)

* **Multi\_nav2\_slam.launch.py**

This launch file starts navigation for multiple robots. This file also takes a map.data file as a parameter. (see **Nav2\_slam.launch.py**)

## Hupbrb

The hupbrb package has the functionality of the robot. It contains the id scanner (QR scanner) node, the priority controller node, and the collision detection node. **The id scanner** node reads the camera data and tries to find a QR code with a number. By finding a QR code it knows there is a second robot and they establish a connection. **The priority controller** takes the priority number of the robots into account and solves collision issues in an optimal way using costmaps (no go zones for the robot). **The collision detection** node takes the paths of the robots into account and detects the point of a collision before it happens.

## Path projection

The path\_projection package is what is running on the raspberry pi connected to the projector. This package contains 2 nodes. **Path\_projection.py** and **multi\_path\_projection.py**. These nodes display the path of a robot (or robots) on the floor of the environment via a projector. The nodes receive the path over the /{robot\_name}/plan topic and update this plan every second to see exactly where the robot is going on the floor.

## Priority\_based\_robot\_costmap\_plugin

This package contains the plugin code for automatically putting a costmap in the right area around the detected collision point. This package differs from the other packages because it is written in C++. This is because plugins can only be written in C++. This means the package contains a **.hpp** file in the include folder which declares the functions used in the .cpp file. The **.cpp** file can be found in the src folder and contains the code for the **collision\_layer\_plugin**.

# How to start

In this paragraph we will be describing how to start multiple robot navigation with the dede robots in a custom world and separately how to set up the projector for displaying the paths on the floor.

## Navigation

The first thing we need to do is start gazebo with a world. Open a terminal and run the following:

**ros2 launch dede simulation.launch.py world:{*.world file*}**

This launches gazebo in your custom world file or without the world keyword the specified world file in simulation.launch.py.

When gazebo is launched we can start the navigation stack. If you have not installed navigation/slam yet, you can find a tutorial on how to install and use slam and navigation inside the dede package (dede/doc/Mapping\_Navigation.md).

To start navigation open a new terminal and type the following command:

**ros2 launch dede multi\_nav2\_slam.launch.py map:={path/to/slam\_map}**

Path to slam\_map refers to the .data and .nox files you receive after you slam (again follow the mapping\_navigation tutorial for these files). For our project they were in hupbrb\_ws/src/dede/maps. You know navigation is running well and does not crash when it displays the following as the last message: **Managed nodes are active.**

After navigation is running, we can launch RVIZ2 which will help visualise the environment and the robot sensor data. To do this open a new terminal and simply run the following command:

**ros2 launch dede rviz2.launch.py namespace:={robot\_name}**

The namespace is the robot name because RVIZ has to be launched for each robot. For our project this would mean dede1 or dede2.

The next step is launching the robots priority controller. This also includes the id\_scanner which uses the robots camera to detect QR codes and connect with other robots. Open a new terminal and run the following command:

**ros2 launch hupbrb prio\_nav.launch.py namespace:={robot\_name}**

You will have to launch this for each robot as well. If your robot does not have a camera/qr code you can manually connect the 2 robots by entering the following command:

**ros2 topic pub /dede1/scanned\_ids hupbrb\_msgs/msg/Identifier "{name : '/dede2', id : 2}" -1**

This command publishes dede2 to the dede1 robot and establishes a connection. -1 is for publishing to the topic only once. If you use another robot the robot name should be different so pay attention to that.

After this you can navigate by publishing goal poses via the terminal or by clicking the 2d Goal Pose button in RVIZ and selecting a place on the map.

## Path projection

Before we start the path projection you need to make sure the raspberry connected to the projector is on the same network as where you are running the simulation. For us we had to use a projector on 10.42.0.69. You will need the ip address of the projector for this.

To start the path projection there are two things you have to do. First of you have to start navigation, otherwise there will be no path and no data published over the /plan topic.

The second thing you need to do is ssh into the raspberry pi connected to the projector. To do this you simply type the following into a terminal:

**ssh ubuntu@10.42.0.69**

This was the IP we used for our project but this will of course be the IP you use for connecting the raspberry over the network.

Then it will ask for the ubuntu password of the raspberry pi which we set to: **AdaptiveRobotics**

This will give you control over the raspberry via the network. Go to the hupbrb\_ws folder by doing the following:

**cd hupbrb\_ws**

**. install/setup.bash**

From here we will launch the path projection node which is simply the following:

**ros2 run path\_projection multi\_path\_projection\_node**

This will start up the path projection node and show paths as soon as it detects a robot is connected and a path is published.

Note: Make sure the ROS\_DOMAIN\_ID specified in your .bashrc file is the same on the pc running the simulation and the raspberry on the projector. This is crucial if you want the two to communicate.