AN INSIGHT INTO:

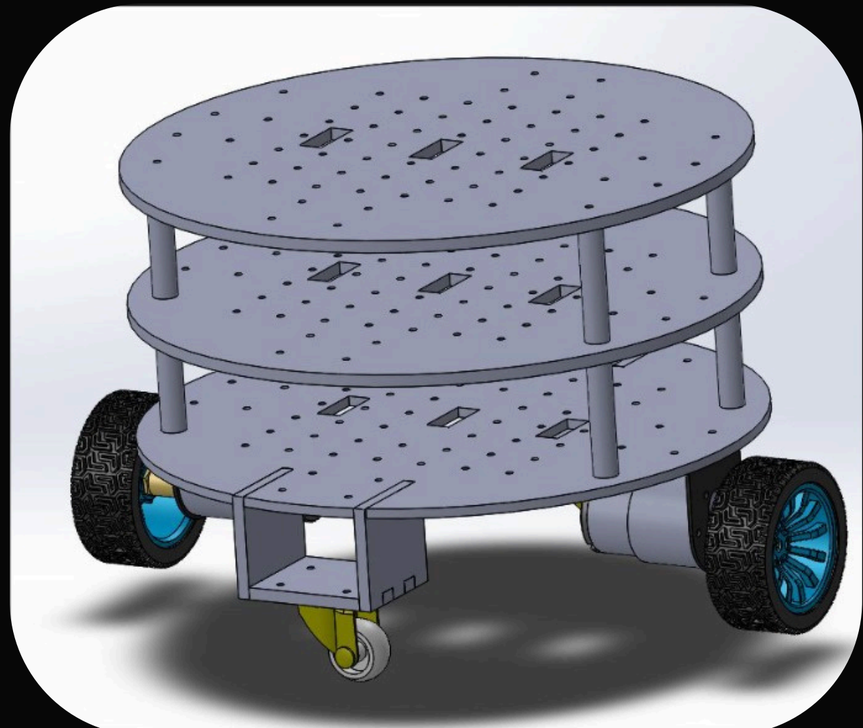
MOBILE PLATFORM

DEFINITIONS:

- Mobile: Ability to move freely.
- Platform: A physical surface on which components can be placed.
- Mobile Platform: A physical surface with the ability to move and on which components can be placed.

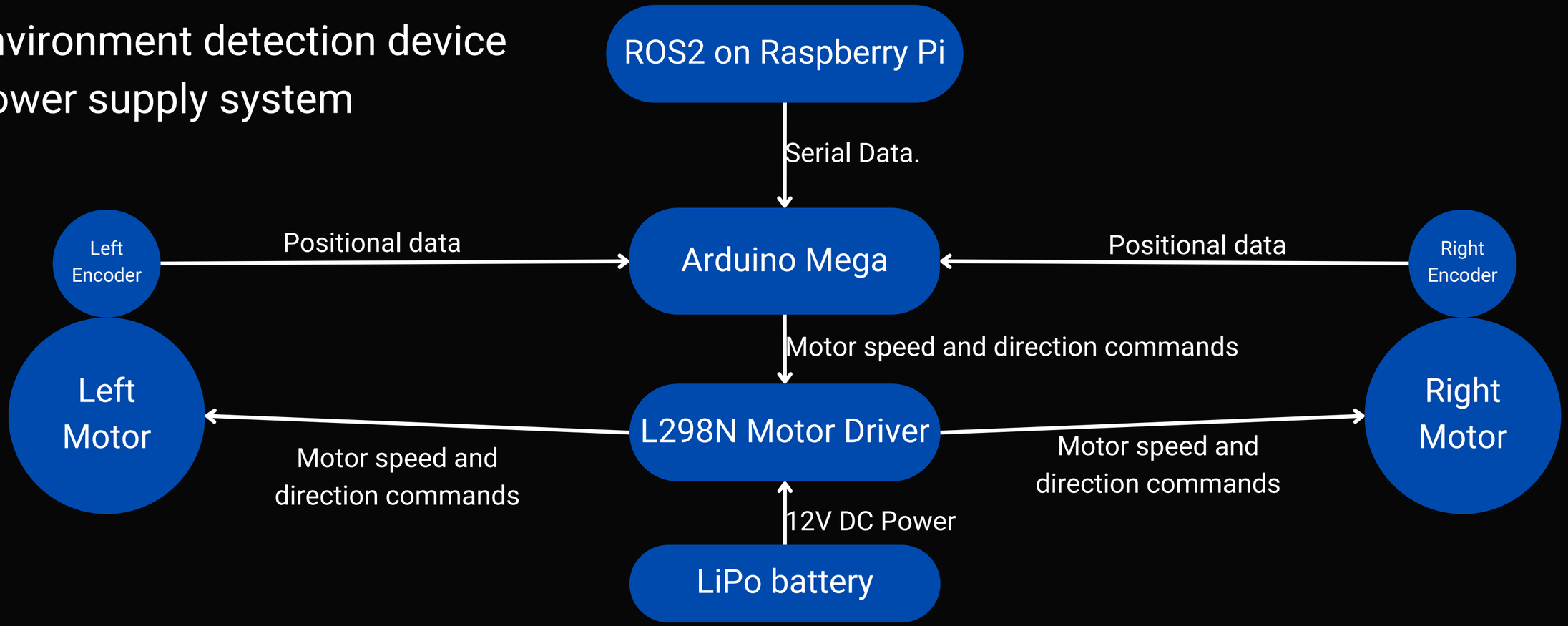
Components making up the mobile platform:

- Electrical components : 3-Cell Lithium Polymer Battery, L298N Motor driver, USB hub and Arduino Mega Microcontroller.
- Electro-mechanical components : Two 12V 200rmp DC motors with encoders and rplidar.
- Mechanical components : Two castor wheels and Acrylic frame.



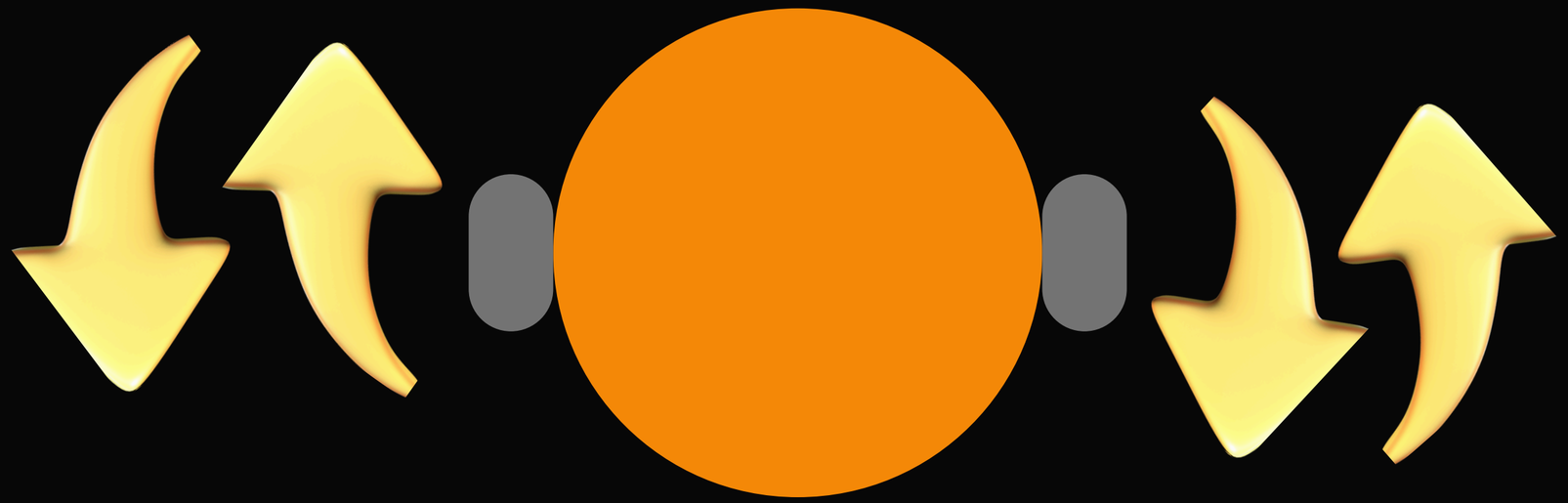
SYSTEMS MAKING UP THE MOBILE PLATFORM

- Drive-train and propulsion
- Robot frame
- Wheel mounts and Wheels
- Control scheme
- Environment detection device
- Power supply system



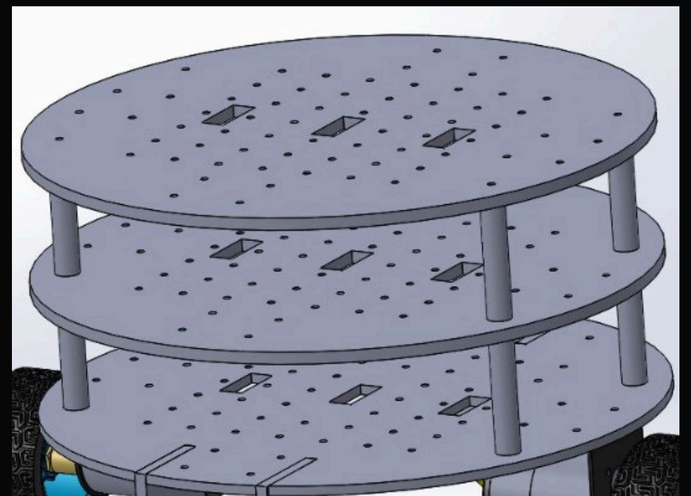
1.DRIVE-TRAIN AND PROPULSION

- This consists of the differential drive system and two 12V 200rpm DC motors with quadrature encoders.
- The differential drive system employs the use of two motors connected to wheels that are aligned to the axis passing through the center of the robot base which allows turning on the spot which increases maneuverability of the robot.
- By use of a motor driver, the speed and direction of the motors are controlled.



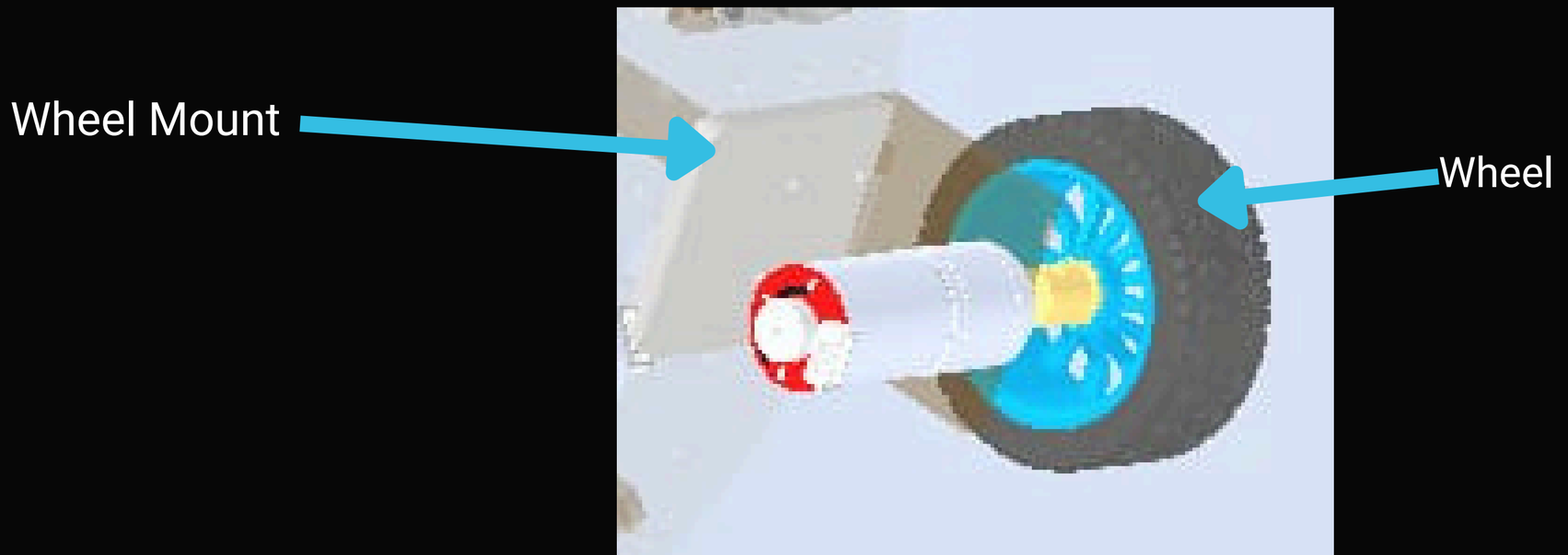
2.ROBOT FRAME

- It is the support structure for all the components of the robot.
- Made of Acrylic due to its high impact resistance,light weight, ease of use with a laser cutter machine and its lower price compared to other materials such as metals.
- Designed into three circular layers to support accommodate all components while reducing the robot’s diameter.
- Heavier components placed at the bottom layer to lower Center of Gravity.



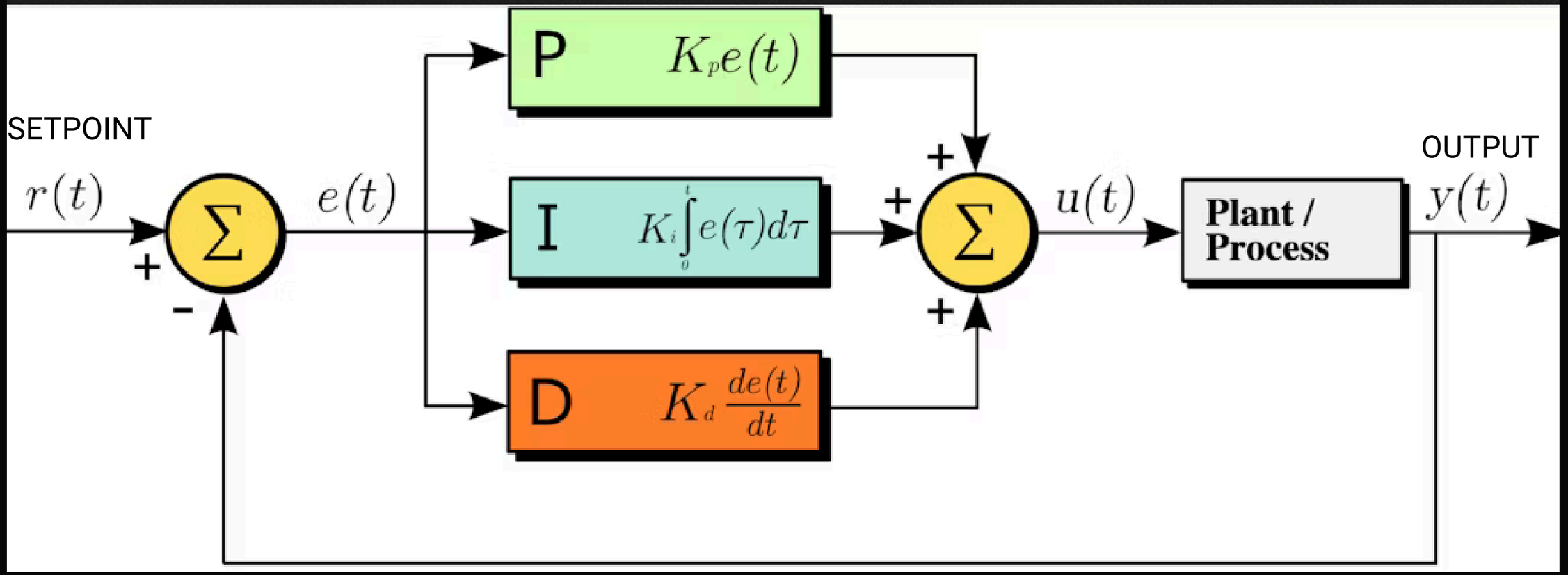
3. WHEEL MOUNTS AND WHEELS

- The wheel mounts were made to extend outwards to increase the wheel separation which increases stability due to wider base support.
- The wheels are 85mm in diameter with 36mm breadth. They have studs on their rubber surface to increase traction and prevent slipping.



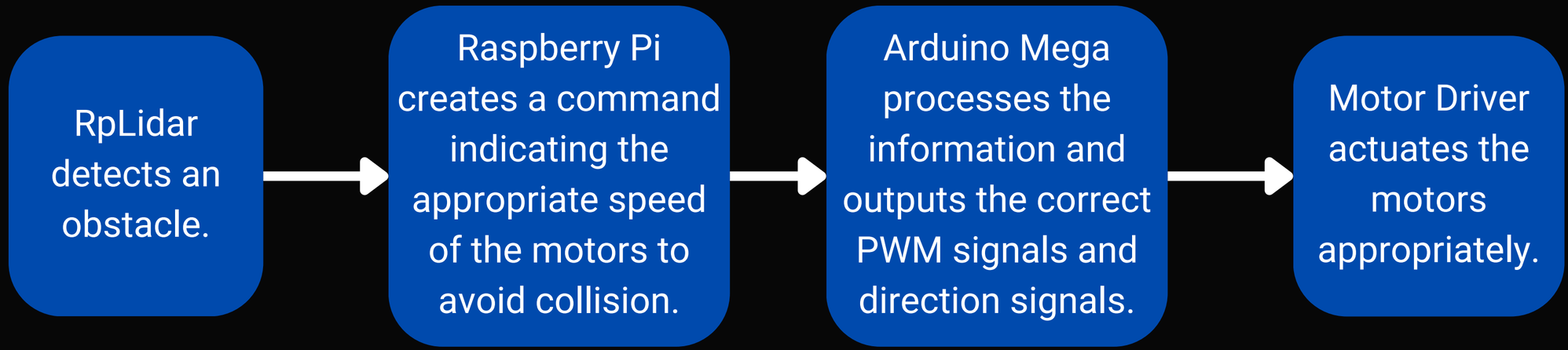
4. CONTROL SCHEME

- The Arduino Mega microcontroller was used due to its large number of pins.
- The microcontroller contains programmes that receive commands from the navigation section, process them and send output signals to the motor driver to control the motor speeds and direction thereby moving the robot to the desired location in the correct orientation without collision.
- Within the program, PID (Proportional-Integral-Derivative) is implemented for closed loop control of the motor speeds and direction. It minimizes error by comparing the **required** encoder counts per time interval and the **obtained** encoder counts per time interval and adjusting the PWM output to ensure the two values are as close as possible.


SETPOINT
$r(t)$
Σ
+
-
$e(t)$
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau)d\tau$
D
$K_d \frac{de(t)}{dt}$
+
+
+
Σ
$u(t)$
Plant / Process
OUTPUT
$y(t)$

5. ENVIRONMENT DETECTION DEVICE

- Since the robot’s purpose is to move safely to a predetermined location at a specific orientation, it requires input commands that control this motion.
- They could be human commands such as from a joystick or from sensor data that is processed to produce control commands and used on the motors.
- The sensor used was the rplidar which scans a 2D environment and creates an image of the obstacles.The Navigation section converts this data to serial commands indicating each motor’s speed and direction and are the sent to the Arduino mega which converts it to appropriate signals for the motor driver.



6. POWER SUPPLY SYSTEM

- Summing up the power requrements for each device is done beforehand to determine the appropriate battery to use. P = IV (Watts).First determine the operating voltage of each device, place devices of the same operating voltage on the same voltage line. Next determine the maximum currents drawn by each device and sum them up. Multiply the total currents on each wire with its voltage to obtain the power required on each line.
- In situations where the voltage required by a device is higher or lower than that of the battery, use a Boost Converter to raise it or a Buck Converter to lower it to a fixed stable value.
- The robot’s total power requirement including the navigation section was found to be at a maximum of 66.7W when both motors were stalled and a value of about 24.7W during rated load conditions which can be safely powered by a 3 Cell LiPo battery as it has 11.1V and maximum draw current of 60A.

- The S number, is the number of cells in the battery.
- Each of the cells has a typical voltage of 3.7V, so for example a 3 cell battery has a total of 11.1V. When they're fully charged, the total will be closer to 12.6V (that's 4.2V per cell), and the voltage will drop as we use them.
- The capacity, usually measured in milliamp-hours (mAh). That's how much current you could continuously draw for the battery to go from full to empty in an hour.
- The discharge rating, or "C number" tells you the maximum current that can be drawn from the battery.Multiply the capacity by the C number.This battery is 20C and 3000mAh, so that's 60000mA, or 60A (more than 7 times the maximum current (7.6A)required for the robot).

AN INSIGHT INTO:

NAVIGATION

What is Navigation? Navigation is the science and technology of accurately determining the position and velocity of an airborne, land, or marine vehicle relative to a known reference, wherein the planning and execution of the maneuvers necessary to move between desired locations are analyzed.

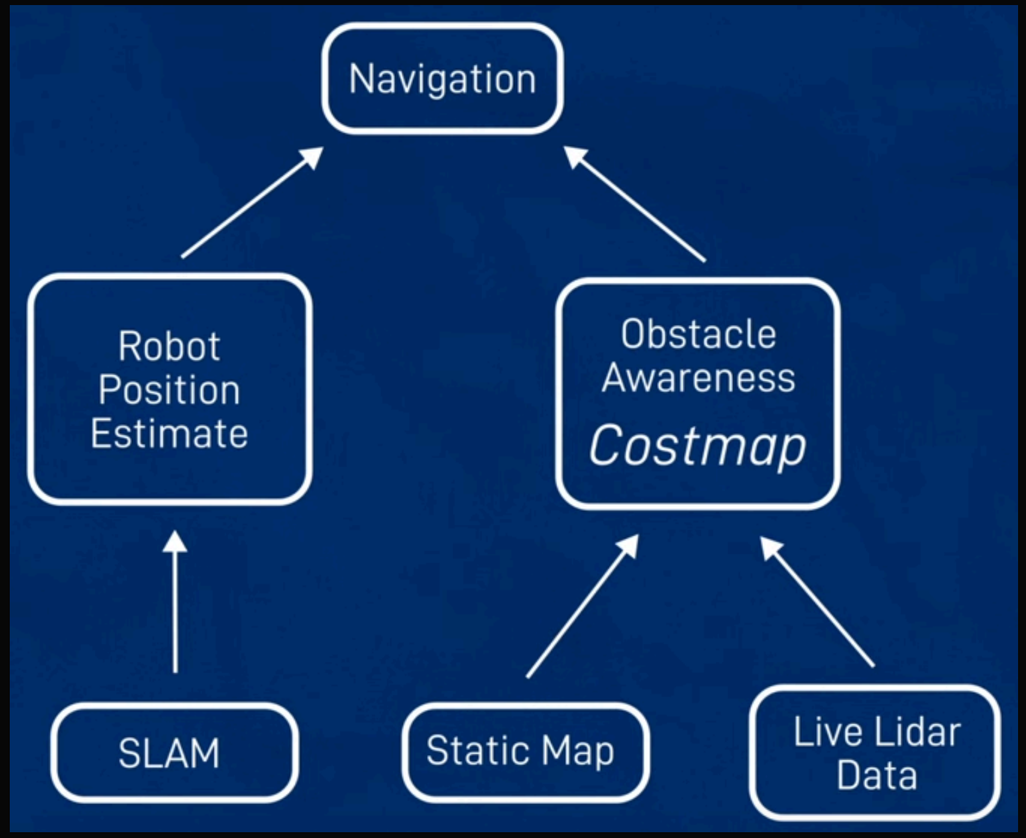
THIS SECTION WAS SPLIT INTO TWO:

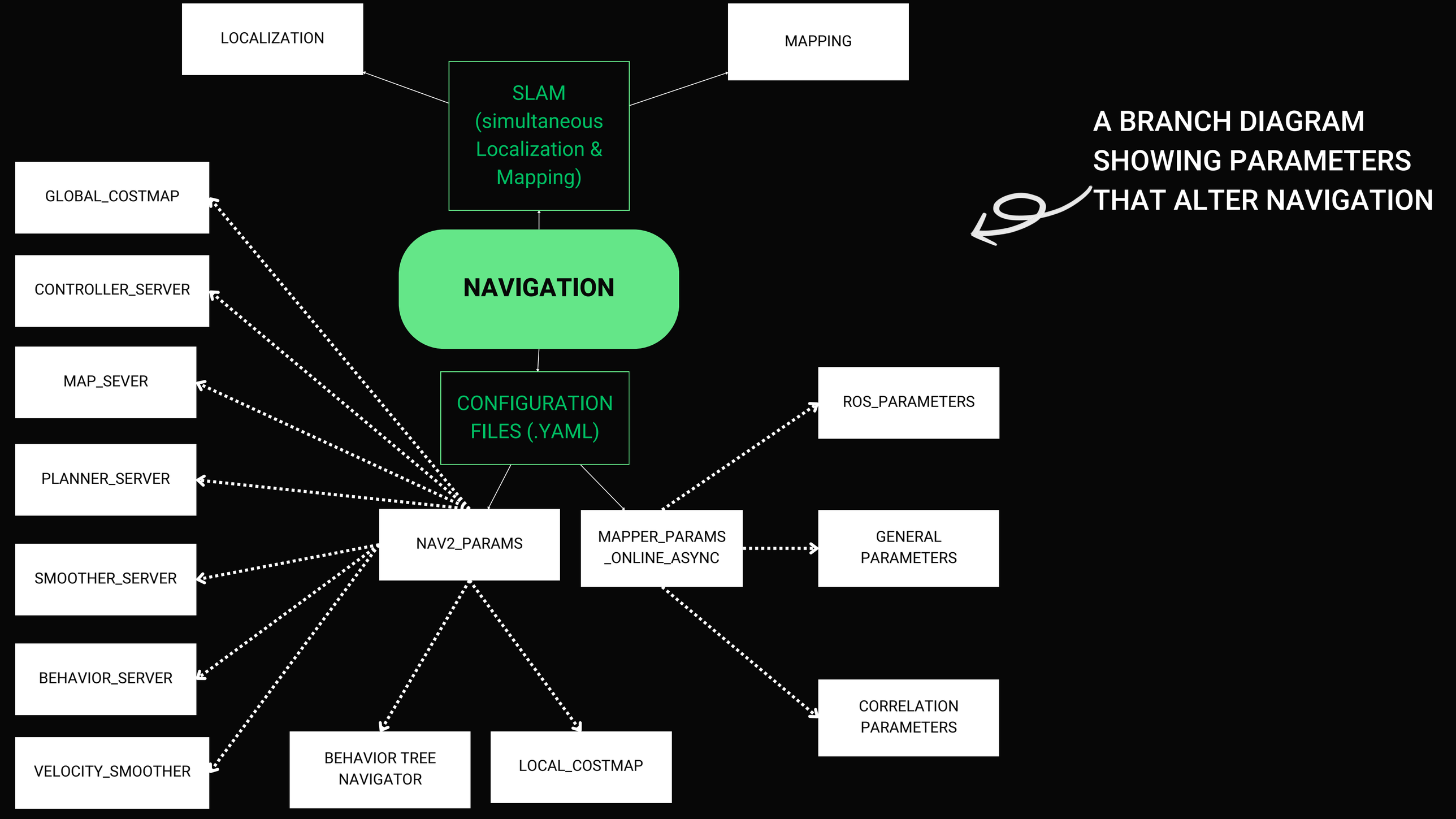
1.SIMULATION IN A VIRTUAL GAME FIELD ON GAZEBO

2.NAVIGATION OF THE REAL ROBOT ON THE GAME FIELD

KNIGHTS x PENTAGON

VISUAL REPRESENTATION OF THE DEFINITION OF NAVIGATION

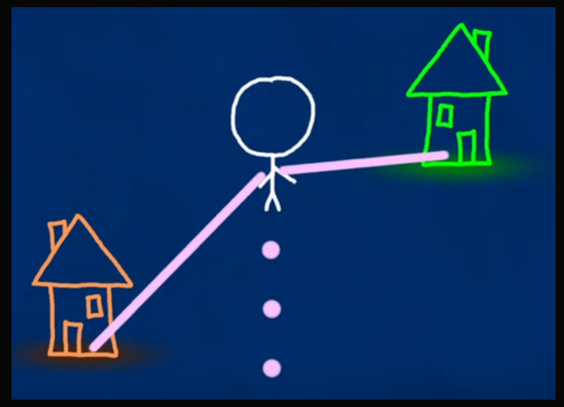



LOCALIZATION
MAPPING
SLAM (simultaneous Localization & Mapping)
A BRANCH DIAGRAM SHOWING PARAMETERS THAT ALTER NAVIGATION
GLOBAL_COSTMAP
CONTROLLER_SERVER
MAP_SEVER
PLANNER_SERVER
SMOOTHER_SERVER
BEHAVIOR_SERVER
VELOCITY_SMOOTHER
NAVIGATION
CONFIGURATION FILES (.YAML)
NAV2_PARAMS
MAPPER_PARAMS _ONLINE_ASYNC
ROS_PARAMETERS
GENERAL PARAMETERS
CORRELATION PARAMETERS
BEHAVIOR TREE NAVIGATOR
LOCAL_COSTMAP

overview of SLAM

SLAM stands for Simultaneous Localization and Mapping. It is a technique used in robotics to solve the problem of building a map of an unknown environment while simultaneously localizing the robot within that map.

The main goal of SLAM is for a robot to navigate and explore an unknown environment, create a map of that environment, and at the same time, determine its own position within that map. This is done in real-time, without any prior knowledge of the environment.



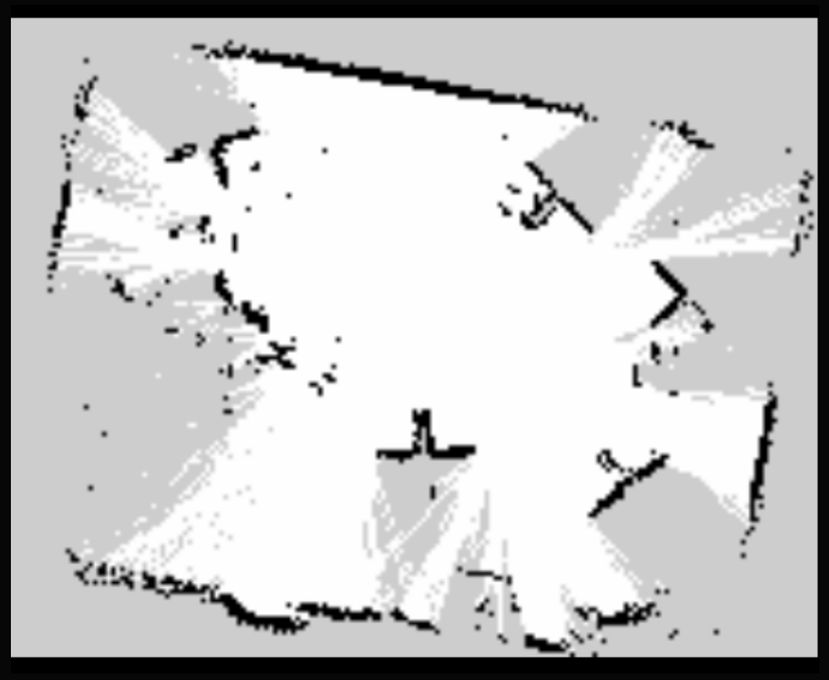
UNKNOWN ENVIRONMENT

SLAM

A NOW FAMILIAR ENVIRONMENT

FEATURE BASED SLAM

overview of SLAM



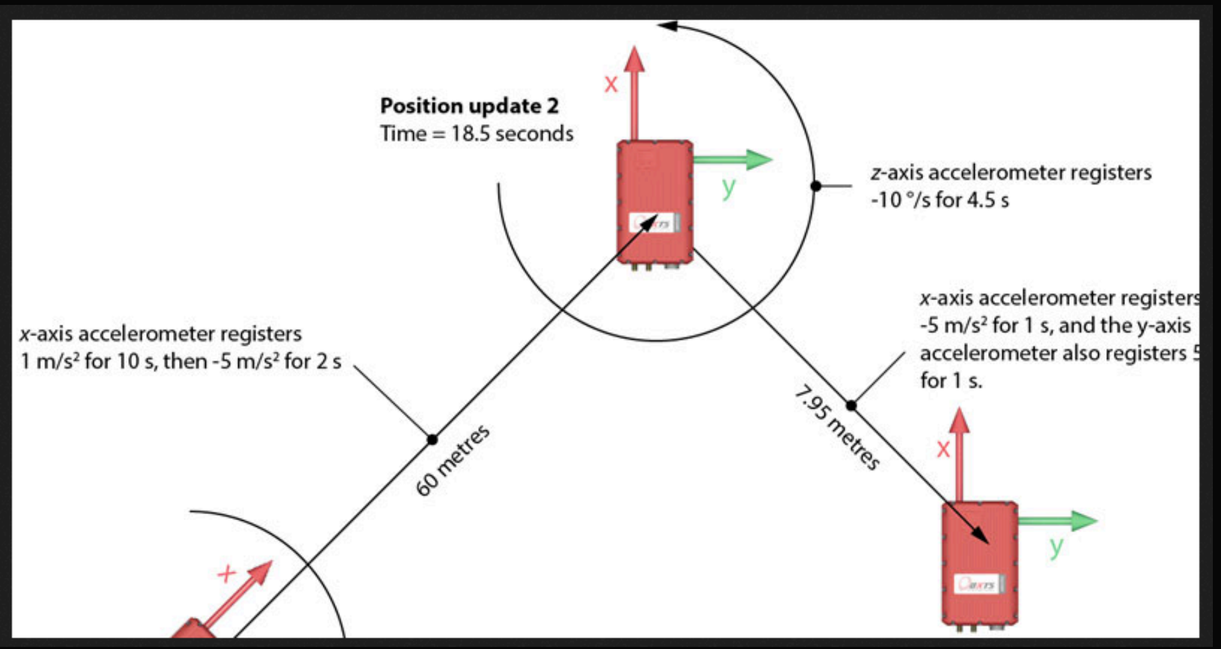
A MAP OF THE BRAINSTORMING ROOM AT IPIC

- Apart from feature based slam there is grid based slam where each section the environment is subdivided into smaller sections called grids.

- Can we solve the SLAM problem if no pre-defined landmarks are available?
  Can we use the ideas of FastSLAM to build grid maps?
  As with landmarks, the map depends on the poses of the robot during data acquisition
  If the poses are known, grid-based mapping is easy (“mapping with known poses”)

GRID BASED SLAM

overview of SLAM



How are poses set during localization?
They are set with the assistance of an technique called the **dead reckoning**

Dead Reckoning is a navigation technique used in robotics and other applications to estimate the current position of a vehicle (like a robot) based on its last known position and the measured speeds and directions over time. In the context of Nav2 (Navigation 2) in ROS 2, dead reckoning can help maintain the robot's estimated position when GPS or other external positioning systems are not available or unreliable.

overview of SLAM

By using SLAM, robots can navigate, explore, and perform tasks in environments where pre-built maps are not available or may be outdated.

To achieve SLAM, robots typically use a combination of sensor data, such as cameras, lidars, or range finders, to perceive the environment. They also utilize algorithms that process this sensor data, estimate the robot’s position, and update the map as the robot moves.

USED THIS FOR THIS PROJECT

DEPTH CAMERA

RANGE FINDER

Here is a brief outline of the roles of each component in the ROS 2 Nav2 stack:

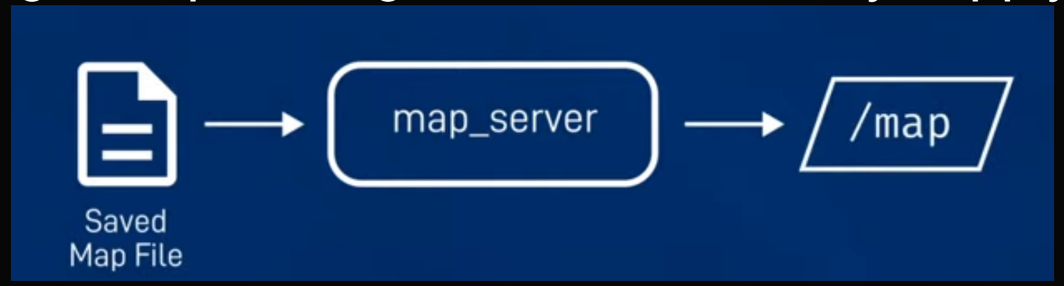
1. Controller Server

- Role: Executes a local plan by controlling the robot's movements based on velocity commands. It uses the local costmap to avoid obstacles and follow the global path.
- Purpose: Converts high-level navigation goals into detailed, short-term control commands (e.g., velocities).

Examples of available controllers: DWB, RPP, MPPI and Graceful

2. Map Server

- Role: Provides the global map of the environment, typically a 2D probabilistic occupancy grid that represents static obstacles and free space.
- Purpose: Used for global planning and localization by supplying the map data to other nodes.



3. Planner Server
 - Role: Generates a global path from the robot’s current position to the target location, avoiding obstacles.
 - Purpose: It computes the global plan using algorithms like Dijkstra or A based on the global costmap and the provided map.
Examples of available planners: Navfn, Smac and Global

 4. Smoother Server
 - Role: Adjusts or optimizes the global path to make it smoother and more feasible for the robot to follow.
 - Purpose: Reduces sharp turns and improves the quality of the path for better navigation performance.
Examples of available smoothers: Simple, Constrained, Savitzky-Golay

5. Behavior Server
- Role: Executes recovery behaviors, like clearing obstacles from the costmap or replanning the path when the robot gets stuck.
- Purpose: Helps the robot recover from failure scenarios during navigation, such as blocked paths. Examples of behaviors: backup, spin & wait.

6. Velocity Smoother
- Role: Ensures that the robot's velocity commands are smoothed out, preventing abrupt changes in speed or direction.
- Purpose: Provides smoother and safer motion by controlling how quickly the robot can accelerate or decelerate.

7. Global Costmap
- Role: Maintains a costmap of the entire environment (based on the map) used by the global planner to compute a safe path.
- Purpose: Represents obstacles and free space globally, so the planner can generate obstacle-free paths.

8. Local Costmap
 - Role: Maintains a dynamic, real-time costmap around the robot based on sensor data. It's used by the local controller to avoid obstacles during movement.
 - Purpose: Provides real-time obstacle information to ensure safe local navigation.

 9. Behavior Tree Navigator
 - Role: Manages the high-level logic for the robot’s navigation tasks using behavior trees, which define the sequence of actions like planning, controlling, and recovering.
 - Purpose: Organizes the flow of navigation, ensuring that actions happen in the right order, and handling failures or unexpected situations effectively.

Each of these components plays a vital role in ensuring the robot can navigate effectively, safely, and recover from issues while executing tasks.

Here is a brief outline of the roles of each component in the ROS 2 Mapper_Params:

1. ROS Parameters
   - Role: Define overall behavior and configuration within the ROS environment.
   - Purpose:
     - Specify coordinate frames (odom, map, base).
     - Define subscription/publishing topics (e.g., laser scan).
     - Indicate operational modes (mapping or localization).
     - Control map management (saving, initialization).
2. General Parameters
   - Role: Determine core functionality and behavior of the SLAM algorithm.

- Purpose:
  - Control scan matching processes (alignment and accuracy).
  - Set thresholds for travel distance and heading changes.
  - Regulate loop closure mechanisms to correct drift over time.

3. Correlation Parameters
   - Role: Focus on the search and matching processes in scan correlation.
   - Purpose:
     - Define dimensions and resolution of the search space.
     - Manage tolerance for deviations in scan alignment.
     - Optimize loop closure processes to recognize previously mapped areas.

Simulation in Gazebo

KEY STEPS TO ACHIEVE THIS:

Prerequisites:

- Installation of Ubuntu 22.04.4 .
- Installation of Ros2 Humble binary version.
- Installation of all necessary tools to work with Ros2 i.e Gazebo, rviz2, rqt_graph, nav2_bringup and Slam_toolbox.

Created a workspace and then cloned this repository “https://github.com/joshnewans/articubot_one.git”.

Built the workspace and sourced it and then modified the urdf file to suit our robot design.

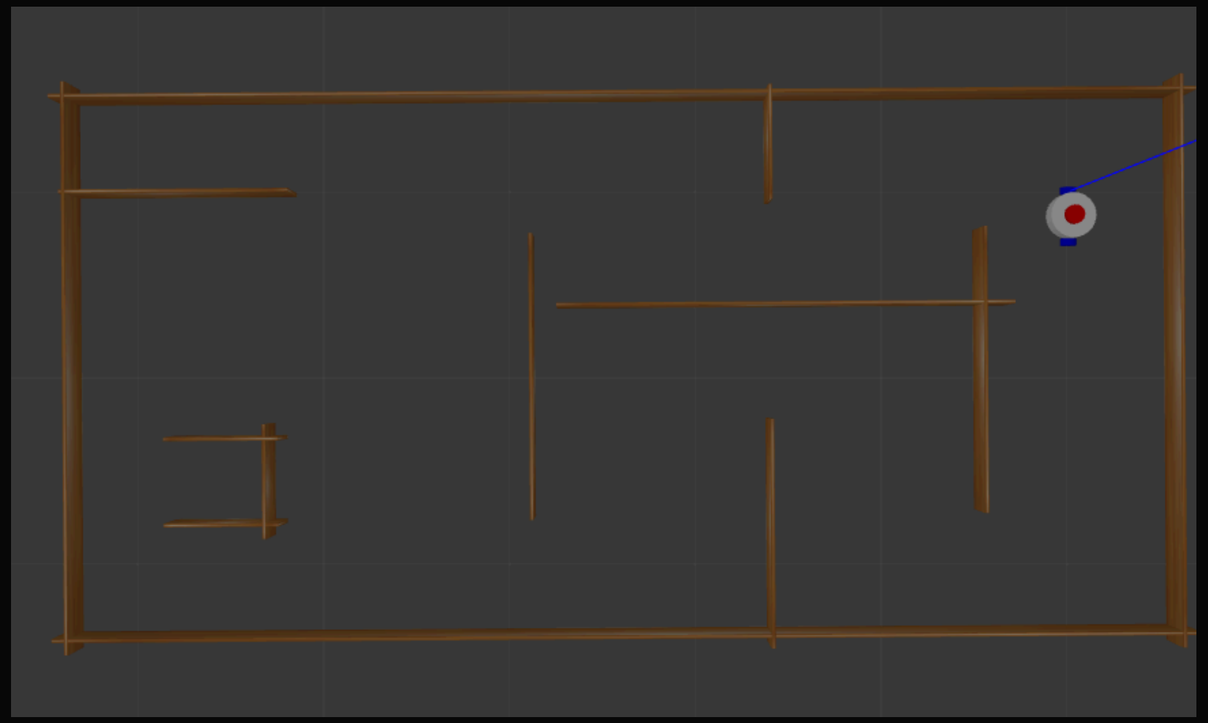
Altered a few parameters on the launch files to update the “spawner.py” to “spawner” entity to suit the new version of ros2 distro that we were using which is Humble Hawksbill.

Added the game field’s virtual world created into the worlds directory for use during simulation tests.

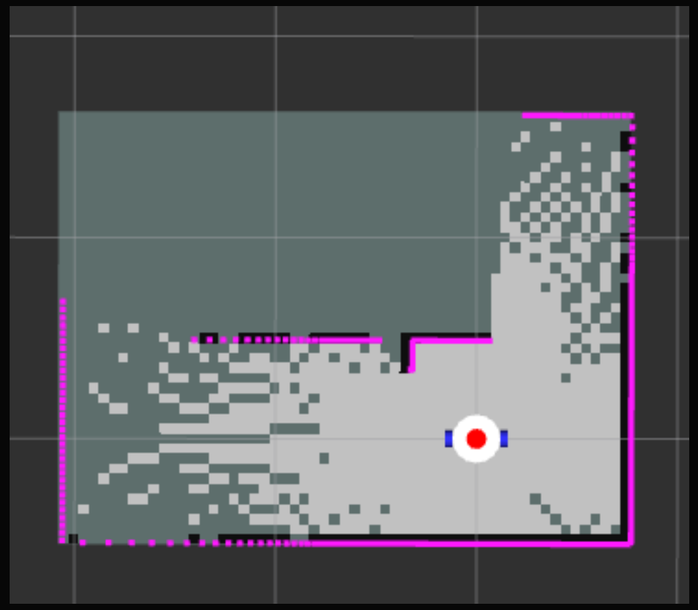
Changed all instances of package name to “jkl” from articubot_one to avoid resource conflicts.

Key Steps To Launch the robot model for Simulation

To launch the robot model while spawned in the gazebo virtual game field, we run the following command: **ros2 launch jkl launch_sim.launch.py world:=./src/jkl/worlds/dojo2024 use_sim_time:=true**



Key Steps To Launch the robot model for Simulation

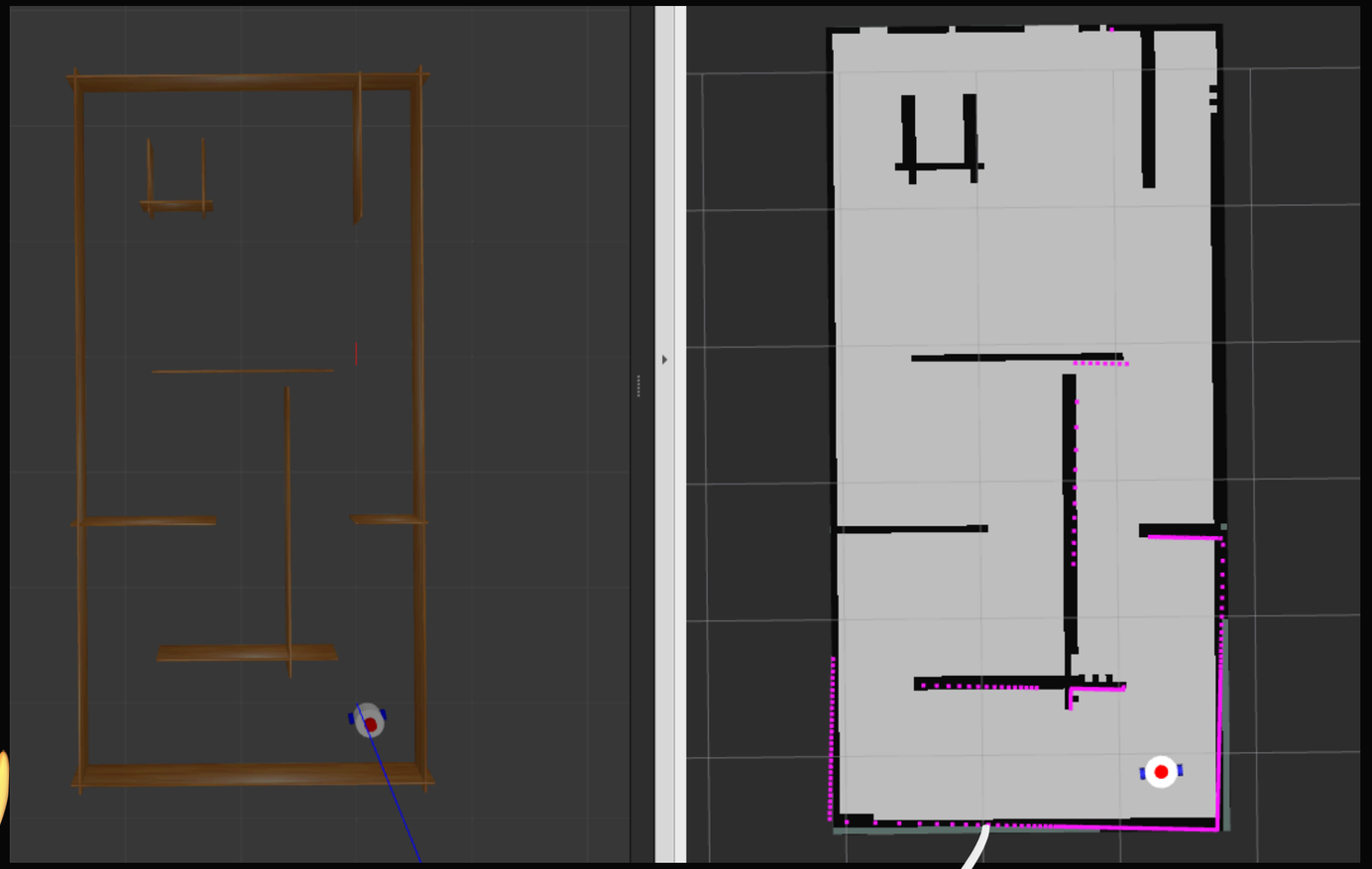


To see the robot behavior in the game field, run Rviz2 using the command: **rviz2**

Mapping the game field was the next thing. This was possible because of the RPLidar A1 M8 laser scanning capabilities in 2D. To achieve this the following command was run: **ros2 launch jkl online_async_launch.py slam_params_file:=./src/jkl/config/mapper_params_online_async.yaml use_sim_time:=true**
For this to work for mapping only without localization at first, the mode in the yaml file path given in the above command needed to be changed to mapping.

NOTE: To drive the robot around during mapping one can use teleop_twist_keyboard or the joy stick node packages

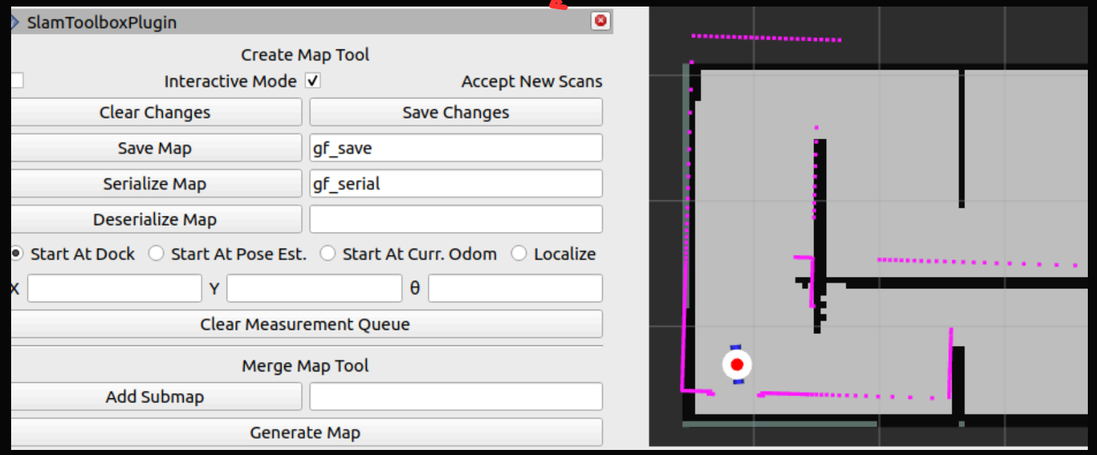
Key Steps To Launch the robot model for Simulation



robot in gazebo during mapping

robot in rviz2 during mapping

Key Steps To Launch the robot model for Simulation

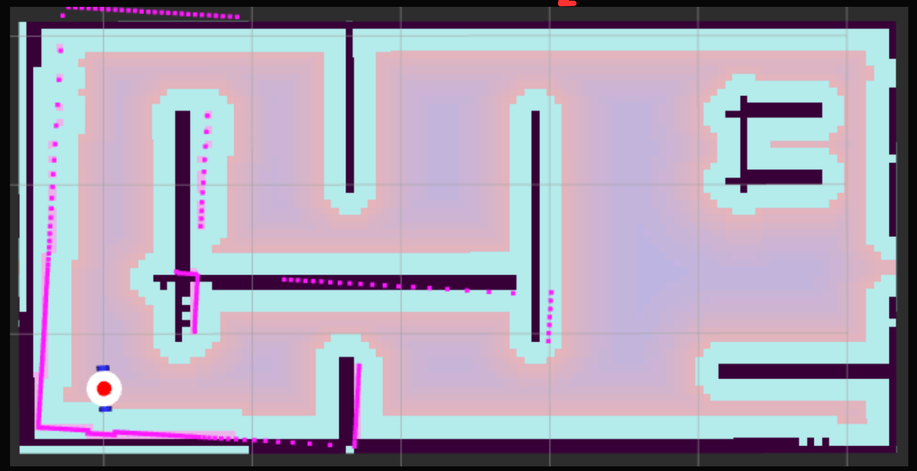


After mapping the next step was to save the map using the slam_toolbox plugin provided in the rviz2 visualization tool. The map was saved in two formats. One as a serial map and the other as a save map. The former was for use within the slam_toolbox while the latter was for use by external tools like amcl (Adaptive Monte Carlo Localization).

After having saved the two maps, the next step was to allow the robot to do localization with assistance from SLAM (simultaneous Localization & Mapping). To do this, run the following command: **ros2 launch jkl localization_launch.py map:=/home/d/robotics-dojo-2024/pi-code/gf_save.yaml use_sim_time:=true**

After the map whose path is given in above command appears, the next step was to set an **initial pose** and let amcl decide the position of the robot in the game field.

Key Steps To Launch the robot model for Simulation



This is the global costmap that helps identify passable and impassable regions via cost.

The next step was to now let the robot navigate through the game field autonomously. To do this, needed to run the following command: **ros2 launch jkl navigation_launch.py use_sim_time:=true map_subscribe_transient_local:=true params_file:=./src/jkl/config/nav2_params.yaml**

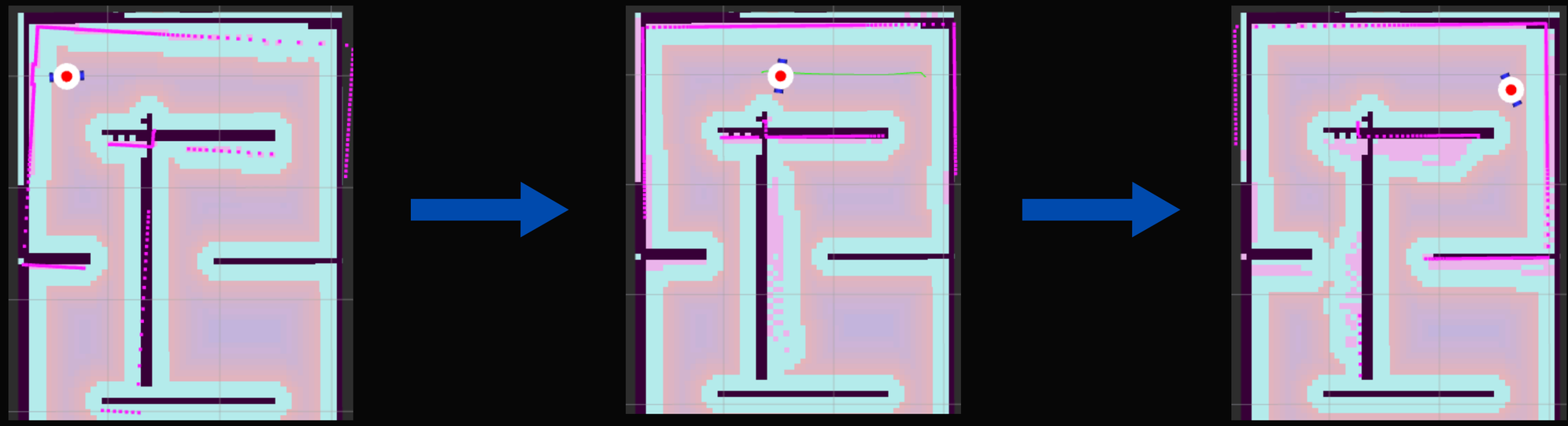
After launching the command above, now at rviz2 a few buttons needed to be added which are:

1. Robot model - topic is /robot_description
2. map - topic is /map
3. LaserScan - topic is /scan
4. Camera - topic is /camera/image_raw
5. map - topic is /global_costmap/costmap
6. Path - topic is /plan

Those are just a few of the basic ones, more could be added for better display while visualizing the robot behavior.

Key Steps To Launch the robot model for Simulation

Rviz2 and Nav2 provides us with two ways to carry out navigation.
One is using the 2D goal pose. Here, one can only set one destination at a time as well as the direction that the robot should face upon arrival at the destination.



start

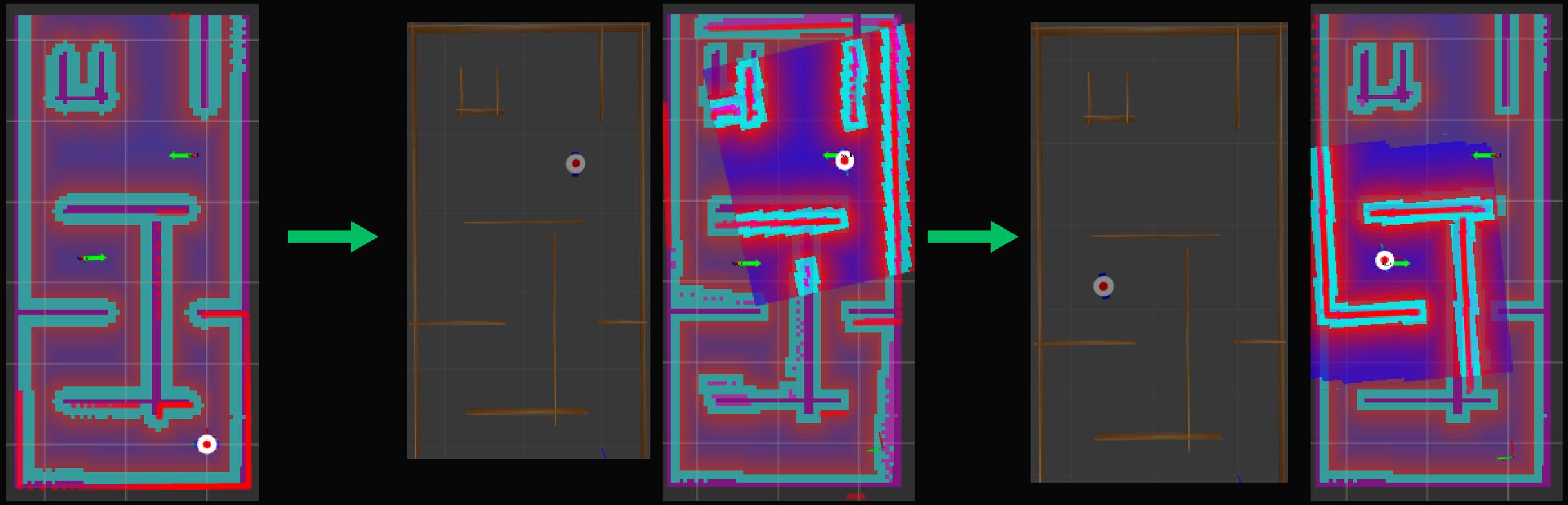
navigation in progress

robot at the set destination
&
Facing the set direction

Second is using the Nav2 goal tool along with the Navigation panel to set waypoints. Here, one can set several successive destinations at once as well as the direction that the robot should face upon arrival at each of the destinations.

GAZEBO

RVIZ2



ROBOT AT THE START POSITION

ROBOT AT THE FIRST WAYPOINT

ROBOT AT THE SECOND WAYPOINT

HOW DIFFERENT IS THE SIMULATION FROM THE ACTUAL ROBOT’S NAVIGATION?	
1	INSTEAD OF SIMULATION TIME BEING TRUE, IT IS FALSE.
2	INSTEAD OF SPAWNING THE ROBOT IN GAZEBO, THE ROBOT IS LAUNCHED ON THE RASPBERRY PI ALONGSIDE WITH THE PACKAGE TO DRIVE THE MOTORS.
3	THE RPLIDAR IS LAUNCHED BY ITSELF USING ITS OWN LAUNCH FILE.
4	TOPICS BEING STARTED AT RASPBERRY PI NEED TO BE LISTED ON THE LAPTOP AND VICE VERSA.