Technical Design Paper for the Robotics Dojo Competition 2024

First A. Amos Oniare, Second B. Dismas Karimi, Third C. Gareth Kipkoech, and Fourth D. Fundi Brian (KNIGHTS)
First A. Daniel Karume, Second B. Joseph Kirika, Third C. Peaches Njenga, Fourth D. Geoffrey Chege Kimani, and Fifth E. Irke Konzolo (Pentagon).

I. Introduction

Robotics Dojo is a project-based robotics training program under the AFRICA-ai-JAPAN Project at the Jomo Kenyatta University of Agriculture and Technology. Each year, the program holds a competition that challenges student teams to come up with a robot that can perform a set of requirements accurately and quickly. This year's topic is Autonomous Mobile Robots. A robot is to be made that can autonomously navigate the competition's environment and pass through given checkpoints to a destination. The robot's performance is to be evaluated based on its speed of doing the tasks and how accurately it passes through the checkpoints.

A. *Design Strategy*

Some guidelines and requirements are provided by Robotics Dojo:

1. It is required to use a lidar
2. It is required to use ROS2
3. Due to requirement (2) above, it is required to use a raspberry pi
4. The build is broken down to two parts: mobile platform and navigation

Upon brainstorming, the following systems are required for an autonomous mobile robot:

1. Drivetrain and propulsion
2. Robot frame
3. Wheel mounts and Wheels
4. Control scheme
5. Environment detection device
6. Power supply system
7. Mapping, Localization, and Navigation

With the above two, some design decisions are made as discussed below:

Drivetrain

For simplicity, cost minimization and rapidity of production, it was decided that the robot drivetrain should employ two-wheeled differential drive. Further, it was decided that the powered wheels should be placed equidistant from the front and back ends of the chassis, for compactness and to minimize turning radius. As such, the powered wheels are minimized to just two with requirement for two unpowered support caster wheels.

To power the driven wheels, it was decided to have two motors, as opposed to a single motor with a differential. This was to minimize mechanical complexity. The motors chosen were brushed DC motors. It was decided that the motors chosen should have built-in rotary encoders, to minimize work during assembly. The calculation of the required motor specifications is detailed in the Mobile Platform subtitle in the Vehicle Design section.

Robot Frame

The frame design chosen is laser-cut acrylic. The primary motivation was that acrylic and a laser-cutting machine were provided. This minimized cost incurred in building the frame. Other pros to this construction include lightweight construction and clean finish.

Inspiration was drawn from Turtlebot3 Burger. It was decided that the robot shall have three stacked compartments and each compartment being of a small area. This was to reduce the breadth of the robot for maneuverability. The heavy battery and motors were to be placed in the lowest compartment to lower the center of mass for stability. [1]

Wheel mounts and Wheels

Having chosen laser-cut acrylic, it was also decided that the wheel mounts should also be laser-cut.

To avoid traversal difficulty caused by uneven floors, it was decided that the wheels used should be rubber with small spikes for traction. This choice sacrifices smooth surface traction for rough surface traction but this was considered a worthwhile trade-off since the wooden floor of the play area is not tiled but rather rough wood. The wheels that meet these requirements were found to be 85mm diameter 36mm breadth.

Control Scheme

It was decided that the mobile platform was to be controlled by a microcontroller. An arduino mega was chosen, primarily because of availability. Another pro that reinforced the choice is its large number of Input/Output pins, including pins with hardware interrupts.

For the navigation, since ROS2 is required, Raspberry Pi 4 was provided by Robotics Dojo. This choice was justified by the Pi 4's speed and comparatively low power requirements.

Environment Awareness Device

It was required to employ the use of a Lidar. For this, Robotics Dojo provided the RPLidar A1M8, a cheap entry-level lidar with a working range of 0.3-12m.

Power Supply System

It was decided to have two independent power sources, supplying the mobile platform and navigation respectively. To supply the navigation system (Raspberry Pi 4) an 18W USB power bank was chosen as it was already owned by a team member, and to supply the mobile platform (Arduino Mega, Lidar, motors, and encoders) a 2-cell Lithium-Polymer battery is chosen. Lithium-ion cells were considered for the mobile platform but were decided against due to reliability concerns of the locally available lithium-ion cells, raised by former Robotics Dojo participants. The calculation of the required battery capacity is detailed in the Mobile Platform subtitle in the Vehicle Design section.

Mapping, Localization, and Navigation

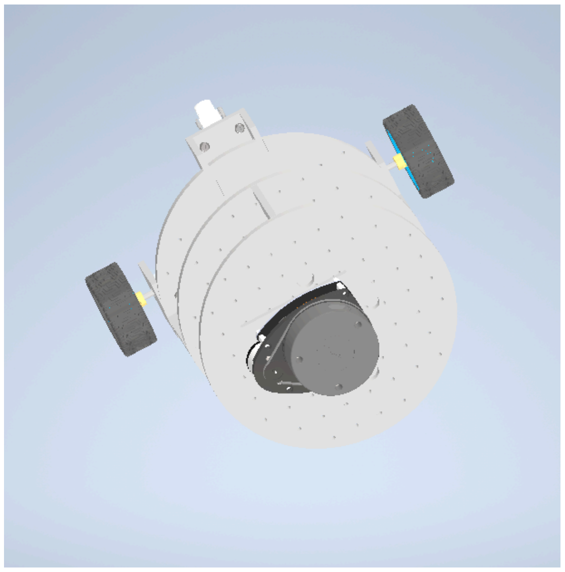
ROS2 provides packages for mapping and localization (SLAM toolbox) as well as localization and navigation (Navigation2). It was decided to make use of these packages rather than writing new custom algorithms. This was to save time and engineering effort. These systems are discussed in more detail in the Navigation subtitle in the Vehicle Design section.

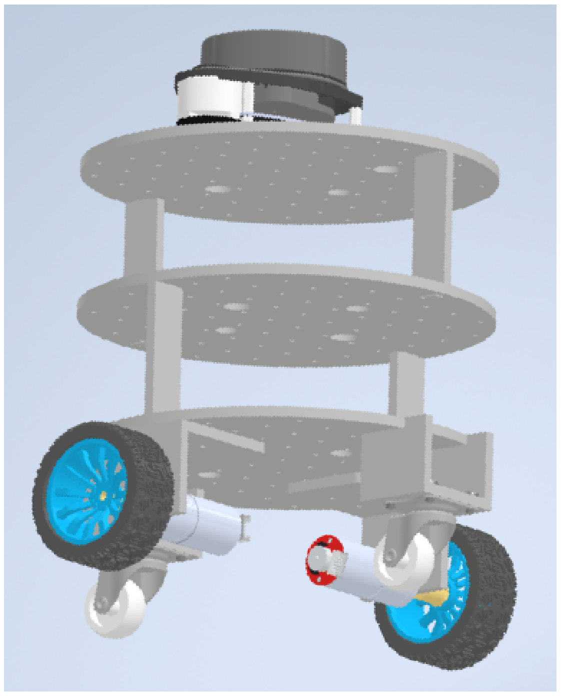
The following steps were followed to create the robot:

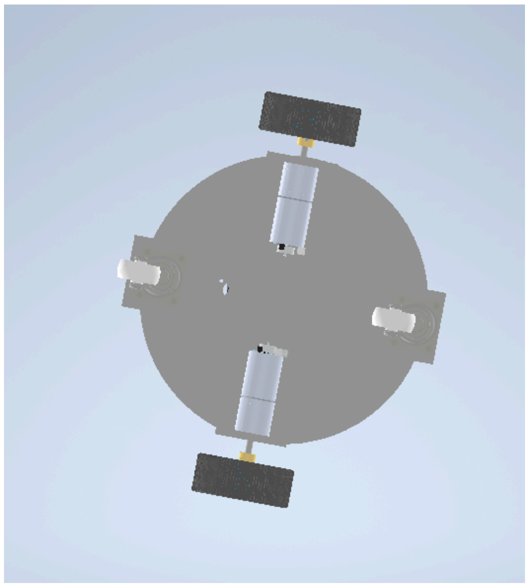
- A 3-dimensional model of the robot was made in Computer Aided Design (CAD) software. The dimensions were made so as to comply with the height limit of 300mm.
- 2-dimensional drawings of each part were generated and placed on a single pdf file for laser-cutting.
- A Bill of Materials (BOM) was generated.
- The total power consumption of the robot is approximated using power ratings of each device to determine the ideal battery and wire gauge to use safely. The items are procured.
- The robot frame was assembled.
- The electrical and electromechanical devices were tested, one at a time, to ensure correct working before assembling them in the robot. Firmware for the Arduino Mega was written to allow control of the motors and encoders via the Serial interface.
- The electronics were assembled into the frame of the robot.
- The robot was controlled manually using the Raspberry Pi via the Serial interface.

B. Vehicle Design

As stated earlier, the robot design was split into two sections, the Mobile Platform and Navigation. Autodesk Inventor, a CAD software, was used to first develop a model of the Robot body which was then laser cut and assembled before adding in the other components of the mobile and navigation section. A 3-Dimensional model of the robot is as shown below in three views:

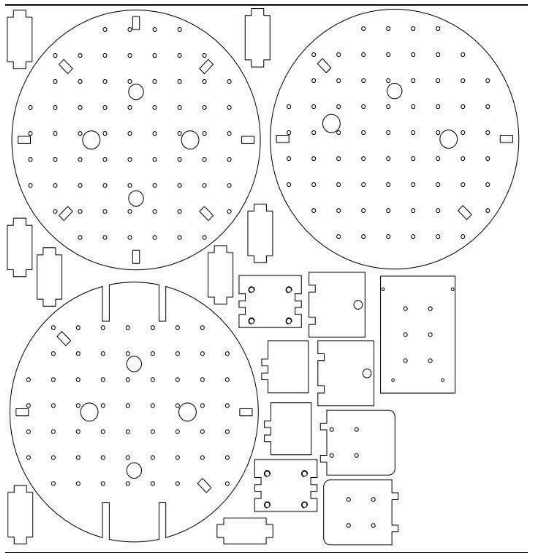






As discussed earlier, the frame of the robot is made up of 3 layers supported by 4 supports for each layer. The layers allowed enough space for the hardware and other electrical components to be placed without congestion and reduce the risk of short-circuits. The supports ensure the body is rigid and that each layer is parallel to the one above or below it. Stability was maximized by lowering the center of mass by placing the heavy components, such as the battery, and motors, on the lowest layer. It was also designed with 4 wheels, 2 being castor wheels and the other 2 are part of the differential drive system. This increases stability of the robot and reduces the weight supported by each wheel hence reducing the structural stresses on the wheel supports. The differential drive wheels were placed along the axis passing through the center of the base to reduce the radius of turning on the spot and allow faster turning which makes the robot more maneuverable.

A 2-Dimensional drawing of each part was created and put in a pdf file as shown below:



The above file was then used by the laser cutting machine to cut each part from a sheet of acrylic. The body of the robot was made from acrylic due the following:

- High impact resistance which is 10 times higher than glass.
- It is Lightweight.
- It has excellent dimensional stability.
- Innate weatherability and UV resistance.
- High optical clarity. (Makes it easier to identify problems in electrical connections within the body)
- Requires less energy when laser cutting as compared to other materials like aluminum.
- Its lower price compared to other durable alternatives like metals.

Laser cutting uses a highly focused laser beam to rapidly heat up a small section of the material, melting and vaporizing it across its thickness. The laser head is moved by stepper motors in the x, y and z axes to allow intricate parts to be cut out from a solid sheet of material.

The chassis of the robot can be made in a number of ways such as 3D printing, or milling but laser cutting was chosen due to the following advantages:

- Almost unlimited 2D complexity.
- High precision as the diameter of the laser is that of a hair and laser head is controlled by high resolution stepper motors.
- No material contamination as no coolant is used.
- High speed. It took only about 30 minutes to produce all the parts of the robot described earlier.

Precautions that should be taken against the harmful gasses produced by the laser cutter machine by ensuring it is equipped with an air filter.

1. *Mobile Platform*

This section consists of the hardware, electromechanical devices and the various electrical components that are sources of power and transmit the required power to the components. They work together to physically move the vehicle based on commands received from the navigation section.

The final robot was estimated to weigh approximately 3kg.

A target maximum acceleration of $0.5m/s^2$ is selected. Given the above two considerations and assuming a worst case 50% efficiency, the required motor torque is given by: $T = \frac{a \times m \times r}{N \times \eta} = \frac{0.5 \times 3 \times 0.0425}{2 \times 0.5} = 0.06375\ Nm$, where a is the target acceleration, m mass of the robot, r radius of the drive wheels and N the count of driven wheels

A target speed of $0.8m/s$ is selected. Given the wheel radius, the appropriate motor rotation count is given by: $RPM = \frac{60v}{2 \times \pi \times r} = \frac{60 \times 0.8}{2 \times \pi \times 0.0425} = 179.751rpm$, where v is the target speed, and r is the driven wheel radius.

Typical current draw is calculated with the relation:
$I = \frac{T\times\omega}{V} = \frac{0.06775\times2\pi\times179.751\div60}{12} = 0.1A$, where T is the required torque, ω is the required angular velocity and V is the supply voltage.

The motors best matching the requirements above are found to be 12V, 200rpm with encoders.

The components making up the Mobile Platform include:

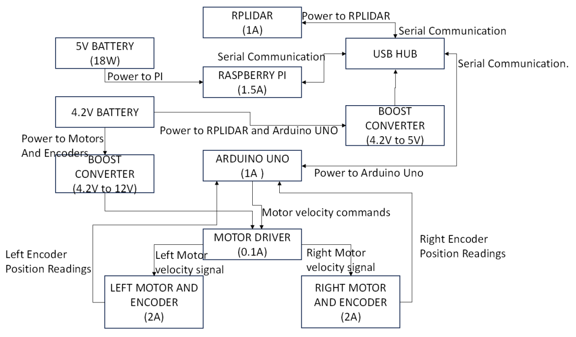
- Two 200rpm, 12 Volt DC motors with encoders.
- L298N motor driver.
- Arduino Mega.
- Two DC to DC Boost converters
- Powered USB hub.
- Castor Wheels.
- Rubber Wheels.
- Slide Switches.
- Two Shaft couplers.
- Two Battery Power Sources.
- Jumper wires.
- Boost converter.

The first step before assembling the components was to determine the amount of power required by each electrical and electromechanical component.

$P = IV\ (Watts)$ equation 'I'

Where; 'P' is Power in Watts
'I' is Current in Amperes
'V' is Voltage in Volts

This information was used in purchasing the right battery with sufficient power for all the devices. The first step in determining total power required was to find the operating voltage and current for each device. Components with the same operating voltages were connected to the same voltage line with the current on that wire being the sum of the currents drawn to each device. The following chart shows the electrical and electromechanical components, the amount of current drawn by each and their connections with each other.


RPLIDAR (1A)
Power to RPLIDAR
Serial Communication
5V BATTERY (18W)
Serial Communication
USB HUB
Serial Communication.
Power to PI
RASPBERRY PI (1.5A)
4.2V BATTERY
Power to RPLIDAR and Arduino UNO
BOOST CONVERTER (4.2V to 5V)
Power to Motors And Encoders
BOOST CONVERTER (4.2V to 12V)
ARDUINO UNO (1A)
Power to Arduino Uno
Motor velocity commands
MOTOR DRIVER (0.1A)
Left Encoder Position Readings
Left Motor velocity signal
Right Motor velocity signal
Right Encoder Position Readings
LEFT MOTOR AND ENCODER (2A)
RIGHT MOTOR AND ENCODER (2A)

It is observed that there are two power sources in the system. One is used to power the Raspberry Pi and the other powers the remaining components. The purpose for this is the Raspberry Pi is powered through a Type-C USB and not a positive and ground line like most of the other devices like motors and found it convenient to give it its own smaller power source.

The total sum of the currents of all the devices was found to be 7.6A but since there are two power sources, they had power ratings based on the currents drawn by the devices they powered. The battery powering the Raspberry Pi must have a minimum of 7.5W rating since the Pi requires:

$P = 1.5 * 5 = 7.5W$.

The Battery used for this project to power the Pi was rated at 18W which was well above the required amount hence safe to use.

The other battery power source was needed to supply 4A to both motors and encoders at 12V, 1A at 5V to the RPLIDAR and Arduino Mega each and 0.1A at 12V to the motor driver. The total power is then calculated as follows:

$$P = (4 * 12) + 2(1 * 5) + (0.1 * 12) = 59.2W$$

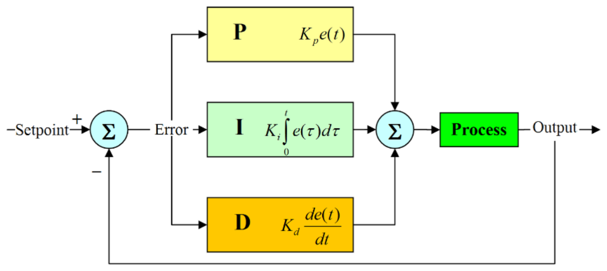
This means that the minimum power rating of the battery to be used should be 59.2W in order to satisfy the motors' requirements at stall. If a single battery were to be used to power everything including the Raspberry Pi, it should have a minimum power rating of 66.7W. A point to note is that when determining the amount of current drawn by a motor for calculations of the required power supply, one should use the stall current as it is the highest motor current in all of its operating modes hence a power supply that is sufficient during motor stall is to be chosen.

When using batteries, it is sometimes observed that it may have the required Power but at a lower voltage than that required by the device. This is where Boost Converters are used to raise the supply voltage of the battery to the voltage required by the device. In this project, two boost converters were used to raise the voltage from 4.2V to 12V for the motors and 5V for the RPLIDAR and Arduino Mega. Portable power sources have a supply voltage value, supply power rating and a capacity value. The capacity of a battery is the measure of how much current it can supply in one hour before depleting its charge, e.g. a 3000mAh fully charged battery can supply 3A for an hour before fully discharging. The higher the battery capacity, the longer it will last on a single charge.

The USB hub was used in order to power multiple devices and allow them to send and receive data from one port of the Raspberry Pi. The boost converter inputs 5V to the hub which powers the Arduino Mega and RPLIDAR which both send and receive serial data from the Raspberry Pi through the Hub. The RPLIDAR sends data representing the distance of points detected and their angle from a reference axis while the Arduino Mega sends serial data on the position of the encoders and receives serial commands from the Raspberry Pi on the velocity of the motors. The Arduino Mega, after receiving velocity commands from the Raspberry Pi, calculates the appropriate PWM (Pulse Width Modulation) signals to send to the motor driver. The code in the Arduino incorporates the PID (Proportional Integral

Differential) controller to minimize the error between the required motor velocity and its actual velocity. This is a form of closed loop control.
The components of the PID control include:

- Proportional Control (P) multiplies the current error value to the proportional gain (Kp). This means that the proportional reaction is large when the error is large and is small when the error is small. P, however, is unable to get rid of steady-state error because when the error becomes infinitely small, the proportional reaction is also infinitely small.
- Integral Control (I) multiplies the integral of the error with a gain (Ki). This means that it is able to eliminate steady state error because as time goes by, the sum of the error accumulates leading to a larger integral reaction to correct the error.
- Derivative Control (D) multiplies the derivative of the error to the derivative gain (Kd). This means that when the error is predicted to be large at a future time, then the derivative response is increased. This produces a damping effect by smoothing down oscillations and reducing overshoot.



2. Navigation

This section consists of the software that enables the robot to map the game field using the RPLIDAR and navigate through it autonomously using the map generated and goal points which are given as input. The software controlling the vehicle can be written from scratch which is cumbersome but there exists an alternative to this method. To better understand it, first know how smart phones and computers work and what led to them operating as they do now. When phones and computers were first produced, each manufacturer made their own hardware and created very specific firmware to allow applications to run. This made it very hard for developers to make applications as there were so many different firmware to learn. The solution for this was brought about by the appearance of operating systems which unified hardware and software which led to the modularity of hardware. Mass production reduced cost, specialized development brought high performance, and ultimately made it possible for computers and mobile phones to be personalized [2].

The robotics community followed this development in order to prevent each company from having their own firmware by creating a common robotics operating system for all robots. This allows immense collaboration in creating new packages to run new or existing hardware and eliminating the need for companies or individuals to come up with software from the firmware level to allow applications to run.

Robotics Operating System (ROS) is what was used for this project because of its popularity and it includes tools that are used to develop robot application programs such as hardware abstraction, low-level device control, sensing, recognition, SLAM (Simultaneous Localization And Mapping), navigation, manipulation and package management, libraries, debugging and development tools. Therefore, is ROS an operating system for robots? The answer is no, it is a middleware that needs to run on an existing operating system. ROS is not a conventional operating system such as Windows, Linux and Android, but a meta-operating system that runs on the existing operating system. In order to operate ROS, an operating system such as Ubuntu, which is one of Linux's distributions, must be installed first. After completing ROS installation on top Linux, features provided by the conventional operating system such as process management system, file system, user interface and program utility (compiler, thread model) can be used. In addition to the basic features provided by Linux, ROS provides essential functions required for robot application programs as libraries such as data transmission/reception among heterogeneous hardware, scheduling, and error handling. This type of software is also called middleware or software framework.[3]

In this project ROS2 Humble was chosen for its maturity and stability. The accompanying operating system chosen is Ubuntu 22.04. Before proceeding to programming, there are some terms used in ROS that would be useful to understand: [4]

- The Node is the smallest unit of processor running in ROS. For example, in the case of mobile robots, the program to operate the robot is broken down into specialized functions. Specialized node is used for each function such as sensor drive, sensor data conversion, obstacle recognition, motor drive, encoder input, and navigation.
- A Package is the basic unit of ROS. The ROS application is developed on a package basis, and the package contains either a configuration file to launch other packages or nodes.
- A Metapackage is a set of packages that have a common purpose.
- A Message is made up of variables such as integers, floating point and Boolean.
- The Topic is where all messages pass through. Each topic is used for a unique type of message e.g. a motor velocity topic.
- Publish is the act of sending a message via a topic

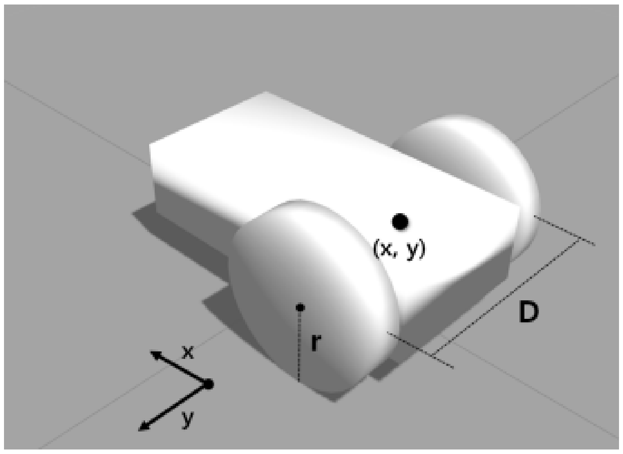
and a publisher is the node that sends the message.

- Subscribe is the action of receiving a message through a topic while Subscriber is a node that receives the message.
- A Service is synchronous bidirectional communication between the service client that requests a service regarding a particular task and the service server that is responsible for responding to requests.
- A Service Server is a server in the service message communication that receives a request as an input and transmits a response as an output.
- A Service Client is a client in the service message communication that requests service to the server and receives a response as an input.
- An Action is another message communication method used for asynchronous bidirectional communication. Action is used where it takes longer time to respond after receiving a request and intermediate responses are required until the result is returned, before which, feedback sends status messages.
- An Action Server is in charge of receiving goals from the client and responding with feedback and results.
- An Action Client is in charge of transmitting the goal to the server and receives result or feedback data as inputs from the action server.
- A Parameter refers to variables used in the node. For example, you can specify settings such as USB port number, camera calibration parameters, maximum and minimum values of the motor speed.
- A Parameter Server is where parameters that are called in the package are registered before itself being loaded to the master.
- URI (Uniform Resource Identifier) is a unique address that represents a resource on the Internet.
- RPC (Remote Procedure Call) stands for the function that calls a sub procedure on a remote computer from another computer in the network.
- XML (Extensible Markup Language) utilizes tags in order to describe the structure of data. In ROS, it is used in various components such as *.launch, *.urdf, and package.xml.
- XMLRPC (XML-Remote Procedure Call) is a type of RPC protocol that uses XML as the encoding format and uses the request and response method of the HTTP protocol which does not maintain nor check the connection.
- The Package.xml contains package information that describes the package name, author, license, and dependent packages.

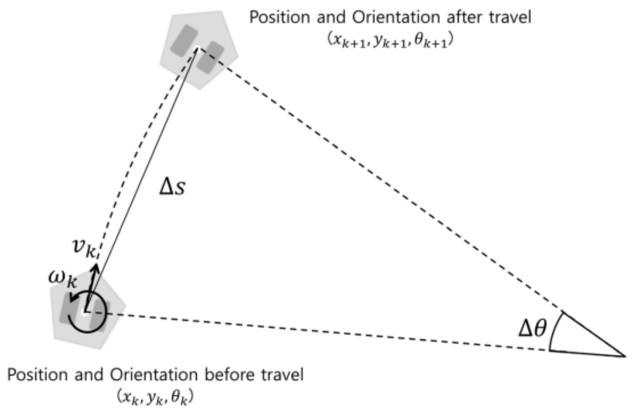
After understanding the introductory concepts of ROS, next is SLAM and Navigation, which were used during mapping, localization and autonomous navigation in the game field.

SLAM allows a robot to generate a map of an environment with very little human assistance. This is a method of creating a map while the robot explores the unknown space and detects its surroundings and estimates its current location as well as creating a map. The sensing device used here is the RPLIDAR which emits modulated infrared laser signals and the laser signal is then reflected by the object to be detected. The returning signal is sampled by the vision acquisition system in RPLIDAR and the DSP embedded in RPLIDAR starts processing the sample data and output distance value and angle value between object and RPLIDAR through a communication interface. The high-speed ranging scanner system is mounted on a spinning rotator with a built-in angular encoding system. During rotation, a 360-degree scan of the current environment will be performed.[7].

Pose of a robot refers to its position and orientation and it must be calculated based on lidar information and encoder readings. Dead reckoning is used to estimate pose of a robot by comparing the sensor readings of pose to the calculated readings of pose from the encoders which use the velocities of the wheels.[5]



The above represents the center (x, y), wheel-to-wheel distance D and wheel radius r.


Position and Orientation after travel
$(x_{k+1}, y_{k+1}, \theta_{k+1})$
Δs
v_k
ω_k
$\Delta\theta$
Position and Orientation before travel
(x_k, y_k, θ_k)

Assuming the robot traveled a very short distance during time T_e , the rotational speed(v_l , v_r) of the left and right wheels are calculated as shown in Eqs. (1) and (2) with the amount of left and right motor rotation (current encoder value E_l c, E_r c and the previous encoder value El p, E_r p).

$$v_l = \left(\frac{El\,c - El\,p}{Te}\right) * \frac{pi}{180} \; \left(\frac{radian}{second}\right) \qquad \text{equation 1.}$$

$$v_r = \left(\frac{Erc - Er\,p}{Te}\right) * \frac{pi}{180} \; \left(\frac{radian}{second}\right) \qquad \text{equation 2}$$

The Equations 3 and 4 calculate the velocity of the left and right wheel (V_l , V_r). From the left and right wheel velocity, linear velocity (v_k) and the angular velocity (w_k) of the robot can be obtained as shown in Equations 5 and 6.

$$Vl = vl * r \left(\frac{meter}{sec}\right) \qquad \text{equation 3}$$

$$Vr = vr * r \left(\frac{meter}{sec}\right) \qquad \text{equation 4}$$

$$V_k = \frac{Vr + vl}{2} \left(\frac{meter}{sec}\right) \qquad \text{equation 5}$$

$$W_k = \frac{Vr - vl}{D} \left(\frac{radian}{sec}\right) \qquad \text{equation 6}$$

Finally, using these values, we can obtain the position (x(k+1), y(k+1)) and the orientation (theta(k+1)) of the robot from Equation 7 to 10.

$$\Delta s = vk * Te \qquad \Delta\Theta = wk * Te \qquad \text{equation 7}$$

$$X(k+1) = Xk + \Delta s * cos(\Theta k + \frac{\Delta\Theta}{2}) \qquad \text{equation 8}$$

$$Y(k+1) = Yk + \Delta s * sin(\Theta k + \frac{\Delta\Theta}{2}) \qquad \text{equation 9}$$

$$\Theta(k+1) = \Theta k * \Delta\Theta \qquad \text{equation 10}$$

There are two main categories of algorithms used in SLAM, but one is most common, that is Smoothing using Pose Graph Estimation. The general idea behind Pose Graph Estimation is as follows. For successful SLAM, at least one absolute measurement device (like a Lidar) and one relative measurement device (like rotary encoders or Inertial Measurement Unit IMU) are required. The lidar is used to detect features in the environment whereas the encoders are used to approximate the robot pose. Say the robot moves in a rectangular box. The lidar captures a feature, say the first wall. Then the robot is moved so that the lidar detects the second wall. The new pose of the robot is estimated using the encoder data by dead reckoning. The robot is then moved so as to capture the third and fourth wall in a similar fashion. Upon reaching the first wall, the algorithm detects a feature that it had recorded earlier, along with the approximate new pose. This completes a loop. Now since the first and last poses have to be identical, the recorded pose is adjusted so that the poses match. However, since each of the poses are relative to their respective preceding poses, they have to be adjusted as well. Doing so produces an approximation of the robot poses *and* the features in the environment. Now the environment is divided into a grid of cells called Probabilistic Occupancy Grid. Each cell is assigned a probability of occupancy, that is, all cells have a probability between 0 and 1 that they are occupied. They can be visualized as occupied cells being black, unoccupied cells being white, and the rest being a shade of gray, showing that the model is not sure whether or not the cell is occupied. This description is superficial and does not showcase the calculations done. Now the slam-toolbox package makes it easy to conduct SLAM. The technicalities are abstracted away and only a handful of parameters are available for tweaking.[8]

Navigation refers to the process of a robot moving in an environment autonomously. Of importance in navigation is motion planning and path planning. A path is a sequence of poses that smoothly connect the start and the goal, and path planning is the process of determining said sequence. Motion planning is the process of not only determining the sequence of poses but also their derivatives, namely, velocities, and accelerations. A basic overview of the path planning process using the Rapidly Exploring Random Trees (RRT*) algorithm is as follows. A map of the environment is a prerequisite. A tree (a network of interconnected nodes) is required. From the starting node, a random position in the state space is selected and a node is placed there. The random selection is biased so that there is greater probability of a new node being placed in the unexplored areas of the space than in the explored areas. Assuming there is no obstacle between the new node and the nearest existing node, the two nodes are connected by an edge, adding the node to the tree. Otherwise, the node is not added to the tree. Now another node is selected randomly as already described. But instead of connecting the new node to the nearest existing node in the tree, the nodes within a specified search radius are analyzed with the objective being finding out a way to connect the nodes that minimizes total path length. As the process is repeated, the nodes approach the goal. Once they reach the goal, more iterations refine the path so that it approaches the optimal path to the goal as the number of samples approaches infinity. This process not only generates a near optimal path to the goal, but also a near optimal path to any location in the environment. Just like in SLAM, the Navigation2 and Nav2_bringup packages in ROS2 abstract the technicalities and present only a handful of parameters for refinement. [6], [9].

The next step after understanding how ROS is used to implement mapping and navigation is a general overview of the steps and terminal commands followed in implementing this on the robot:

1. Setup Ubuntu and ROS2 on your computer.
2. Create your workspace folder/directory, in it add folder named 'src' and in it clone the following Github repository: https://github.com/ru3ll/dojo.git..This repository contains the ROS package and instructions to install additional packages needed for this project.
3. Run the command 'colcon build –symlink-install' in the workspace directory to build all your

packages in the 'src' directory.

4. Run the command 'source install/setup.bash' to allow the package commands to be accessible.
5. Run 'ros2 launch 'nameOfYourPackage' launch_sim.launch.py worlds:= 'path to your world file use_sim_time:=true'' This starts gazebo which simulates the robot with its environment in a virtual world.
6. Run 'ros2 run teleop_twist_keyboard teleop_twist_keyboard' to activate the controls needed to move the robot in gazebo.
7. Run 'rviz2' to launch rviz which is a software that displays the calculated position of the robot in gazebo or the real world based on the wheel velocities. It simulates where it thinks the robot is located based on odometery.
8. Run 'ros2 launch slam_toolbox online_async_launch.py params_file:=./src/dojo/config/mapper_params_online_async.yaml' to launch SLAM and begin mapping.
9. Save and serialize the map created using the slam toolbox plugin in rviz in order for it to be used during navigation.
10. Run 'ros2 launch nav2_bringup localization_launch.py map:=path_to_your_map' that acts as the map server of the map saves during mapping.
11. Run 'ros2 launch nav2_bringup navigation_launch.py use_sim_time:=true map_subscribe_transient_local:=true' to launch navigation package that allows one to place goals on a map and have the robot move autonomously to the goal.
12. To control the real robot, run 'ros2 launch 'nameOfYourPackage' launch_robot.launch.py use_sim_time:=false' then rerun the previous commands and set use_sim_time:=false. Launch the lidar package using 'ros2 launch 'nameOfYourPackage' rplidar.launch.py'

C. *Experimental Results*

Once assembly of the robot and development of its software was done, it was ready for testing.

The robot's maneuverability was tested by remotely controlling its motion and observing how it behaves round tight turns at higher speeds and if it stops safely after applying a stop signal to the motors when at full speed. It was able to accomplish the above two without falling on its side. Additionally, because of the differential drive system, it was able to turn on the spot by giving the two motors the same magnitude but different direction of velocity because the left and right wheels were the same distance from the center of the chassis base.

Traction is how well the robot's wheels are able to push it without slipping. This property is increased by the increase of the friction between the wheels and the surface it is on. Therefore, to ensure high traction, wheels with rough surfaces were used to increase friction. Since the vehicle has four wheels, two being castor wheels, it is recommended to ensure that the two differential wheels are always receiving maximum weight to reduce slip. This was achieved by raising the castor wheels a small amount above the differential wheels from the ground, about one millimeter, in order to increase traction on the differential wheels without causing the vehicle to noticeably pivot about the two differential wheels.

Before testing the mapping and navigating ability of the vehicle, first simulate it using gazebo, RVIZ, slam package and nav2 packages for localization and navigation. The reason for this is to improve on the operator's ability to control the vehicle quickly and perform numerous iterations of mapping and navigation over short periods of time. During mapping, it was observed that RVIZ misrepresented the actual position of the virtual robot in Gazebo when the robot had entered a region in the game field that resembled a previous section that had already been mapped. The result of this problem was the destruction of the map generated, rendering it unusable requiring a new map to be generated which leads to loss of time. To solve this problem, ensure the speed of the robot is adjusted to lower velocities to allow for enough processing time for the map being generated and for subtle corrections of the lidar data to take place efficiently. Once mapping has been done successfully, navigation is then tested using the map generated. Nav2 packages serve the map in RVIZ and create regions on the map known as regions of high risk, regions of medium risk and regions of low risk. The wall represents a region of high risk where the vehicle should never pass through, about 0.3 meters and below from the wall represents the region of medium risk where the vehicle only chooses to pass through if it is the only option and the region of low risk is about 0.3 meters and above away from the wall which is where the vehicle picks as the first choice to pass through as it is farthest from the wall. It was observed that during navigation the regions of medium risk would overlap in areas of a constriction, making it harder for the vehicle to pass through autonomously as it would spend more time processing the best path as there would be no region of low risk. To solve this problem, the navigation parameter known as inflation radius must be decreased in order to prevent overlapping of the region of medium risk. A precaution is to ensure the inflation radius is more than or equal half the distance of differential wheels separation to prevent collision during navigation.

Once the simulation and changing of parameter values is done, begin performing mapping and navigation on the real robot. It is here that the durability of the robot is observed in how long its structure can hold up after numerous trials of mapping and navigation. It is observed that using glue on joints will wear out faster and make them loose but the better alternative is to use screws on most or all the joints as they are more rigid and unaffected by higher temperatures.

D. References

1. "TurtleBot3." n.d. TurtleBot. Accessed September 21, 2024. https://www.turtlebot.com/turtlebot3/
2. ROS Robot Programming, 1st ed., ROBOTICS Co., Seoul, Republic of Korea, 2017, pp. 2-6.
3. ROS Robot Programming, 1st ed., ROBOTICS Co., Seoul, Republic of Korea, 2017, pp. 10-13.
4. ROS Robot Programming, 1st ed., ROBOTICS Co., Seoul, Republic of Korea, 2017, pp. 41-49.
5. ROS Robot Programming, 1st ed., ROBOTICS Co., Seoul, Republic of Korea, 2017, pp. 313-316.
6. ROS Robot Programming, 1st ed., ROBOTICS Co., Seoul, Republic of Korea, 2017, pp. 332-360.
7. SLAMTEC. (2020, October).RPLIDAR A1.Low Cost 360 Degree Laser Range Scanner [Online].
8. H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," in IEEE Robotics & Automation Magazine, vol. 13, no. 2, pp. 99-110, June 2006, doi: 10.1109/MRA.2006.1638022.keywords: {Simultaneous localization and mapping;Mobile robots;Robotics and automation;History;Artificial intelligence;Navigation;Vehicles;Buildings;Bayesian methods;Particle filters},Link: https://ieeexplore.ieee.org/document/1638022
9. "1105.1186v1 [cs.RO] 5 May 2011." 2011. arXiv.Link: https://arxiv.org/pdf/1105.1186.pdf