Autonomous Mobile Industrial Robot (Final Report)

In association with

International Society of Automation

Submitted by students of

Manipal Institute of Technology, Manipal

By

Team Lunokhod

Neehal Sharrma	Abhishek Deshpande	Karthik Gogisetty	Arvind Kaushik	Kevin Peter	Mufaddal Abuwala

Guide

Mr. Sivayazi Kappagantula

Faculty Advisor

Mr. DAP Prabhakar

1 Introduction

1.1 Introduction to Project FIEPER

FIEPER, short for *Field Operator Robot*, is an Operations Assistive Robot built to carry out various tasks whilst deployed in process industries in hazardous environments, where human intervention may prove harmful to life and safety. The aim of Project FIEPER is to take this abstract idea to its logical conclusion, thereby setting the goal of working towards the end-to-end design, prototyping and final implementation of a fully autonomous bipedal humanoid robot that can replace human workers in such dangerous environments.

1.2 Project Lunokhod

Project Lunokhod is a step towards this ideal portrayed by FIEPER. The name “Lunokhod” was inspired by the Soviet *Lunokhod* robotic lunar rovers. Like the rovers, Project Lunokhod is one of our first forays into uncharted territory with Project FIEPER.

1.2.1 Problem Statement

One of the first problems to be addressed by any robot that needs to work independently is the ability to traverse an environment autonomously, without any prior external assistance from a human operator. The problem statement addressed for this project is:

> *To design a Mobile Robot platform capable of fully autonomous navigation for use in industrial warehouses and other inventory / stock processing applications.*

Expanding on this abstract, the following specific features and functionalities for Lunokhod were proposed:

Navigation

1. Navigate a structured environment (like a warehouse) autonomously
2. Assist the human operator when in manual mode

Decision-Making

1. Decision Tree to make decisions based on the allotted tasks
2. Determine the order of tasks and actions based on priority

Collaboration

1. Operator-controlled GUI to send goals and tasks to the robot
2. Tele-op system with assisted control when in manual mode

1.2.2 Team Organisation

The 6-member student project team, consisting of two junior and five sophomore students, all pursuing a B.Tech. in Mechatronics, was divided into three overarching subsystems for effective delegation:

Artificial Intelligence

The AI Subsystem is responsible for programming the path planning, obstacle avoidance and other autonomous navigation capabilities, as well as performing proof-of-concept simulations in ROS.

Electronics and Control Systems

The ECS Subsystem is responsible for setting up communication systems and designing robot control algorithms for effective traversal capability in real-life environments and communication with the base control room / other robots and machines.

Mechanics and Robotic Systems

The MRS Subsystem is responsible for generating mathematics models for quantitative mechanical analysis and creating Mechanical CADs for visualisation and to set the stage for future real-life implementation.

1.2.3 Projected Timeline

The projected timeline is elaborated in Figure 1. The team started work in October 2020, with the expected completion of the simulation stage in May 2021.

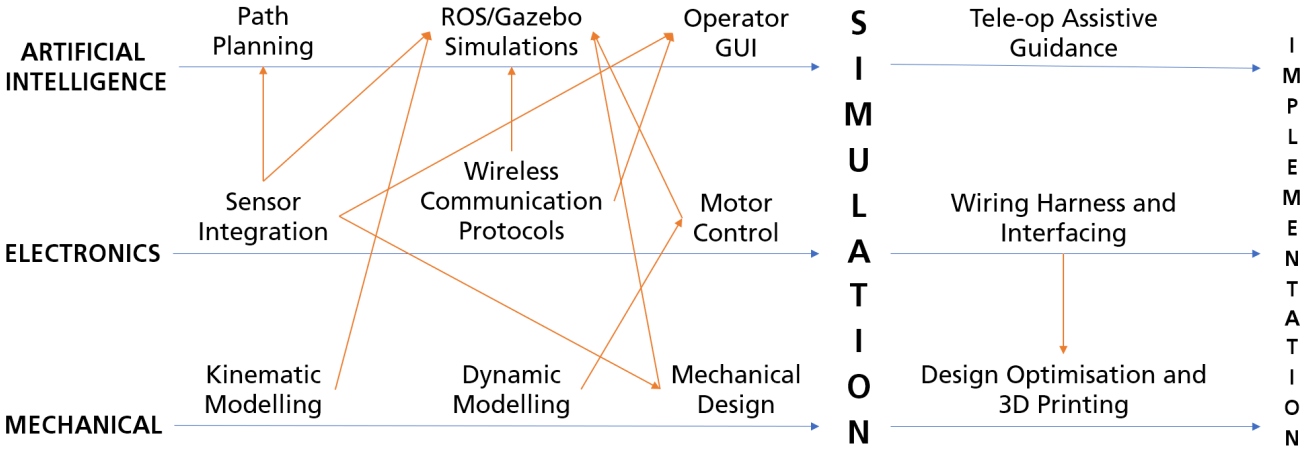

Figure 1: Projected Task Timeline for the Project.

2 About Mobile Industrial Robots

2.1 Introduction to MIRs

The advent of the Industrial Revolution 4.0 is something that we are witnessing today. With the emergence of smarter and more efficient technologies every day, we are getting more connected and productive. With unprecedented advancements in fields of Autonomous Robotics, Artificial Intelligence, and Internet of Things, we are employing robots to do tasks which come under the 3 Ds- Dirty, Dull and Dangerous. This reduces human intervention in tasks which do not genuinely require them to actively apply their intelligence to solve problems.

Now, countless robotic industry players view Mobile Industrial Robots (MIRs) as the next big movement in industry to address labour shortages, the growing demand for customized order fulfilment, and increasingly dynamic production environments that are pushing manufacturers to employ ever-leaner, more agile technologies.

Some salient features of MIRs that have played a part in making them popular amongst industrialists include:

1. The robot can be programmed with the path to be followed defined. The programming, usually, is simple and does not require much programming expertise.
2. Autonomous navigation using light detection and ranging (LiDAR) technology, and on-board intelligence and collision-detection safety systems that allow for the real-time selection of the most appropriate route to any given destination at a particular instance.
3. The time of operation can also be specified, and the robots will follow the designated paths to carry the required material.

2.2 Goal Parameters for Project Lunokhod

Based on our study of currently operational MIRs and the feasibility of completion, we selected the following goal parameters for Project Lunokhod for us to work towards:

Artificial Intelligence	Electronics and Control Systems	Mechanics and Robotic Systems
Autonomous Navigation Communications Portal Operator GUI	Communication range > 15 m Response time lesser than 2 s	Max. footprint 1x1x1 metre Max. velocity 1m/s Net weight 15kg Max. payload 5kg

3 Autonomous Navigation

Autonomous navigation means that a vehicle can plan its path and execute its plan without human intervention. In some cases, remote navigation aids are used in the planning process, while at other times the only information available to compute a path is based on input from sensors aboard the vehicle itself. An autonomous robot is one which not only can maintain its own stability as it moves but also can plan its movements.

3.1 Need for Autonomous Navigation

As our requirements become more and more complicated, it becomes less feasible for humans to control the robot. It might even become impossible at a particular stage if the control requires co-ordination with other robots and their movement patterns. To solve this issue, we use autonomous navigation which will free up manpower, prevent accidents due to human error and find more efficient paths that a human controller might not use.

Autonomous robots use navigation aids when possible but can also rely on visual, auditory, and olfactory cues. Once basic position information is gathered in the form of triangulated signals or environmental perception, machine intelligence must be applied to translate some basic motivation (reason for leaving the present position) into a route and motion plan. This plan may have to accommodate the estimated or communicated intentions of other autonomous robots to prevent collisions, while considering the dynamics of the robot's own movement envelope.

At a more complex stage in the project, autonomous navigation would also allow us to communicate with other bots in the warehouse. This can speed up transport as the bots would avoid paths that are being taken by other bots or calculate the exact speed and direction it would need to take an intersecting path without collision.

3.2 Path Planning Algorithms

The problem to find a “shortest” path from one vertex to another through a connected graph is of interest in multiple domains. One of the earliest and simplest algorithms is Dijkstra’s algorithm. Starting from the initial vertex where the path should start, the algorithm marks all direct neighbours of the initial vertex with the cost to get there. It then proceeds from the vertex with the lowest cost to all its adjacent vertices and marks them with the cost to get to them via itself if this cost is lower. Once all neighbours of a vertex have been checked, the algorithm proceeds to the vertex with the next lowest cost. Once the algorithm reaches the goal vertex, it terminates, and the robot can follow the edges pointing towards the lowest edge cost.

Another algorithm that can be used is the A* algorithm. To solve the problem that we encounter in Dijkstra’s algorithm where it starts exploring areas that do not actually take it closer to the goal, we could give priority to nodes that have a lower estimated distance to the goal than others. For this, we would mark every node not only with the actual distance that it took us to get there (as in Dijkstra’s algorithm), but also with the estimated cost “as the crows flies”, by calculating the Euclidean distance or the Manhattan distance between the vertex we are looking at and the goal vertex.

When the search space is very large, the two algorithms mentioned above become very computationally expensive and slow. To solve this, we can use the RRT algorithm. the algorithm selects a random point in the environment and connects it to the initial vertex. Subsequent random points are then connected to the closest vertex in the emerging graph. The graph is then connected to the goal node, whenever a point in the tree comes close enough given some threshold. Although RRT is generally a coverage algorithm, it can be used for path-planning by maintaining the cost-to-start on each added point and biasing the selection of points to occasionally falling close to the goal. RRT can also be used to consider the non-holonomic constraints of a specific platform when generating the next random waypoint. RRT quickly finds some solution for the path planning problem, but smooth paths usually require additional search algorithms that start from an initial estimate provided by RRT. Figure 2 shows a demonstration of the RRT algorithm in action.

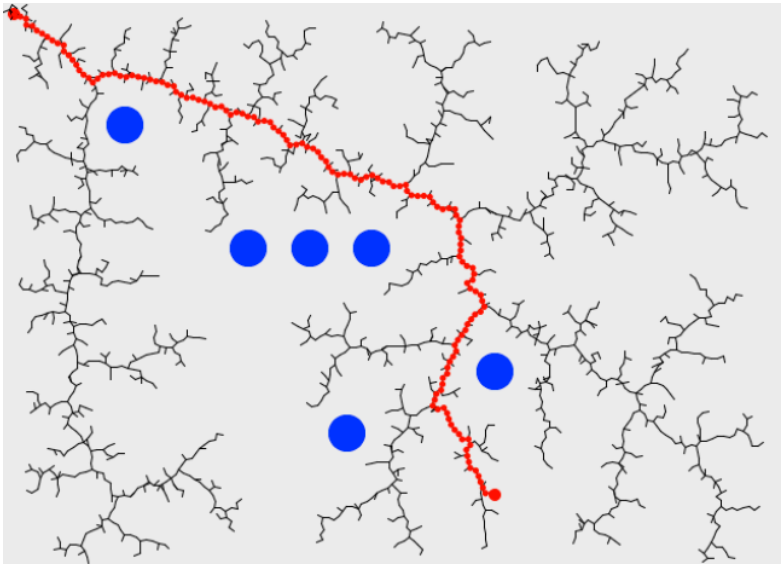

Figure 2: A visualisation demonstrating the RRT Search Algorithm in action.

Therefore, a combination of RRT must be used with the A* algorithm to allow for smooth and fast path planning compared to using only one of the algorithms.

3.3 Simultaneous Localisation and Mapping (SLAM)

Bots cannot always rely on GPS, especially when they operate indoors. GPS is not sufficiently accurate enough outdoors because precision within a few inches is required to move about safely. Using SLAM, robots build their own maps as they go. It lets the robot know its position by aligning the sensor data it collects with whatever sensor data it has already collected to build out a map for navigation. We can then use this map and the RRT algorithm in conjunction with the A* algorithm to get a path to the goal. Once a path is calculated, we can use simple kinematics to calculate how to navigate the path with the available motion.

SLAM is a complicated multi-stage process involving sensor data alignment, motion estimation, sensor data registration, visual odometry and map building for localization. Robots continuously do split-second gathering of sensor data on their surroundings. Camera images are taken about 90 times a second for depth-image measurements along with LiDAR images, used for precise range measurements, which are taken 20 times a second.

Wheel odometry considers the rotation of a robot's wheels to help measure how far it is travelled. Inertial measurement units are also used to gauge speed and acceleration to track a robot's position. All these sensor streams are taken into consideration using sensor fusion (explained in the next section) to get a better estimate of how a robot is moving. Kalman filter algorithms and particle filter algorithms are used to fuse these sensor inputs. Bayesian filters are applied to mathematically solve where the robot is located, using the continuous stream of sensor data and motion estimates.

3.4 Simulation in ROS

To simulate the behaviour for Lunokhod, we used Robot Operating System (ROS). ROS is a flexible framework for writing robot software, comprising of a collection of programming tools, libraries, and software that aim to simplify the task of creating complex and robust robot behaviour.

3.4.1 Tools and Software

The following open-source tools and software were used alongside ROS to model the simulation:

URDF

Unified Robot Description Format (URDF) is an XML format for representing a robot model. URDFs are used to generate rigid body trees that represent the robot through links and joints be describing the properties of each link and their connections to each other. URDFs are essential to describing a robot in any environment and making sensible connections between robot controllers and real-life sensors on a robot.

To simulate the bot in ROS, we need to create a model of the robot with the URDF format. This allows us to use all the various plugins to simulate and use a LiDAR, GPS, IMU amongst others. These sensors are by default ideal, and do not show the noisy output we get from real-world sensors. To account for this, we can add a Gaussian noise parameter, which will make sensor output noisy, to better emulate real-world sensor output.

RViz

RViz or Robot Visualiser is a 3D visualization tool for ROS applications. It provides a view of the robot model, captures sensor information from robot sensors, and replays captured data. It can display data from cameras, lasers, and from 3D and 2D devices including pictures and point clouds, thus allowing us to view the world through the robot's eyes and better understand the way it observes and can interact with its environment.

Gazebo

Gazebo is one of the primary simulation environments for ROS, featuring a robust physics engine, high-quality graphics, and programmatic and graphical interfaces to help simulate ROS implementations. By creating environments for specific use-cases (for example, city roads for a self-driving car or a factory line for an industrial robot), we can observe how the robot autonomously interacts with its environment.

3.4.2 Simulation Setup

Figure 3 showcases the robot in action, clearing a room from one end to the other. The LiDAR mounted on the front of the robot feeds data through a ROS Topic to the path planner, which then generates a path trajectory for the robot to follow. This path is iteratively updated by new data coming in from the LiDAR.

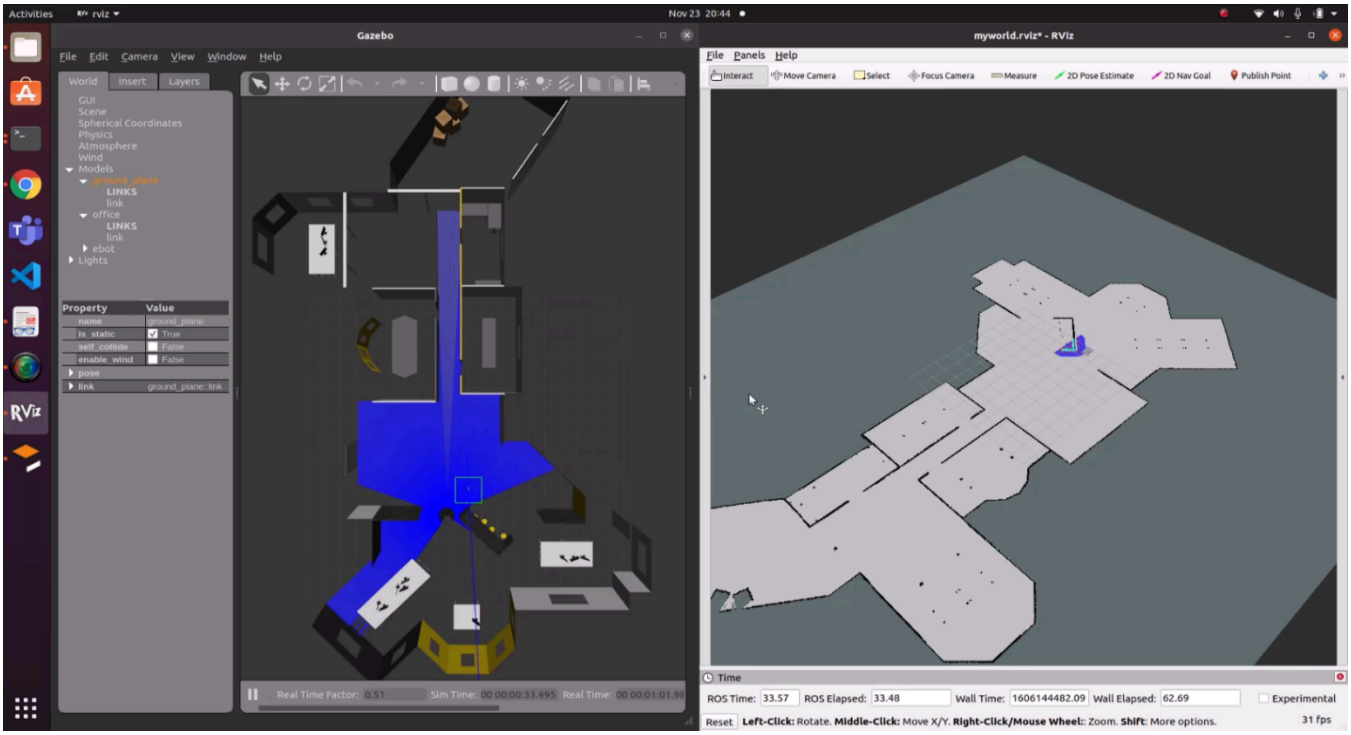

Figure 3: A screenshot of the simulation in action. The left window shows the Gazebo environment while the right shows the RViz visualisation.

The LiDAR publishes a message of type `LaserScan` to the topic `ebot/laser/scan`. Data can be retrieved from this topic in the form of a list containing detected distances for each LiDAR step. The GPS publishes data to the topic `/fix`, with a message of type `NavSatFix`, providing the latitude and longitude, which must be converted to regular cartesian coordinates before being used by the path planner. This conversion is done using forward equirectangular projection and is very accurate. The IMU publishes to the topic `/imu`, with a message of type `Imu`. The orientation data is stored in the form of quaternions and must be converted to the Euler angle notation before being used for calculations.

Once all the data is retrieved and the required conversions are performed, the data from the sensors is used by the path planner algorithm as explained above. When a path is calculated, the required base wheel speeds are published to the topic `/cmd_vel`, which controls the motion of the

base.

4 Electronics and Controls

Perception and control are one of the most important aspects of the robot makeup. Tying together the various cognitive autonomous and mechanical aspects of the robot makes for a successful control scheme that can enable the robot to function effectively in the environment.

4.1 Sensor Fusion

Any robot needs an effective way to receive external stimuli and react to it; this “nervous system” of the robot is the domain of Sensors. Each sensor can provide data about the surrounding to feed its perception system. However, it is imperative to create an efficient system that fully utilises the powers of each sensor to complement gaps in observational information so the robot may make the most impactful decision based on the information available.

This is where Sensor Fusion steps in. Most such sensors include LiDAR's, which provide point cloud of returns from the obstacles in the environment. Sensors like IMUs, GPS are used to localise the bot designed and get an estimate of the current position. All the data provided by sensors usually have disturbances (noise) due to internal or external factors for a given sensor and workspace. Hence, the data is iteratively collected and filtered to have proper estimate and act accordingly as per the goal required using the acquired data.

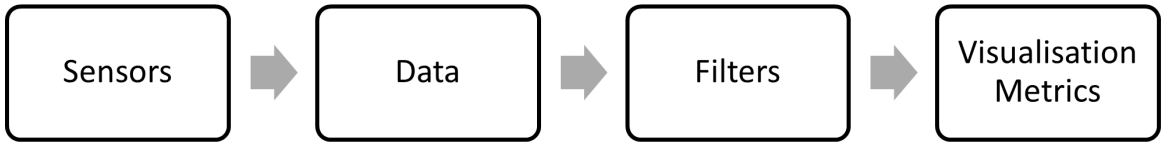

Figure 4: Schematic Block Diagram explaining the overall fusion process.

While sensing is usually done by quality sensors that are readily available, execution is done by well-known and mature control setup. The stage of perception poses the core of developing Lunokhod for a higher degree of autonomous operation.

4.1.1 Equipment Setup

JPDA tracker

JPDA is a tracker system available in MATLAB. It uses joint probabilistic data association to assign detection to each track. It estimates the state vector and state estimate covariance matrices for each track. Each detection is assigned to at least one track. If the detection cannot be assigned to any existing track the tracker creates a new track.

RANSAC Algorithm

The principle mainly covers searching the best plane among a cloud of 3D points to reduce the number of iterations for the collected data by sensors even if the number of points is very large, like with LiDAR's. It randomly selects any three points and calculates the required parameters for the best selected plane.

LiDAR

The LiDAR returns necessary measurements per object. It returns a cloud of point data from the obstacles in the workspace. In the example the data is processed using conventional joint probabilistic data association (JPDA), configured with an interacting multiple model filter. The pre-processing of LiDAR point cloud data to remove noise is performed by using a RANSAC-based plane-fitting algorithm and bounding boxes are formed by performing a Euclidian-based distance clustering algorithm.

4.1.2 Tracking Extended Objects

When more than one object is detected some form of isolation technique to distinguish each object must be implemented. Two common issues that arise are:

1. Loss of valuable on size and orientation of object under track. It results in inconsistency of the turn position, which will vary based on the angle of sensor with respect to the object being tracked
2. Imperfect clustering can lead to false tracks.

Hence, we can approach the problem by moving to an extended object tracker like radar. As mentioned, radars have higher resolution than the object and return multiple detection per object.

In general, it offers better estimation of objects as they handle clustering and data association simultaneously using temporal history of tracks. For an optimal result we fuse the information provided by two sensors (LiDAR and radar trackers.) to have proper estimate and perception of the scenario. Figure 4 is an example demonstration of autonomous bot moving in an environment using sensors to collect the data and act accordingly similar principles as explained are implemented in Lunokhod using on board computers like Raspberry Pi and sensors as listed above.

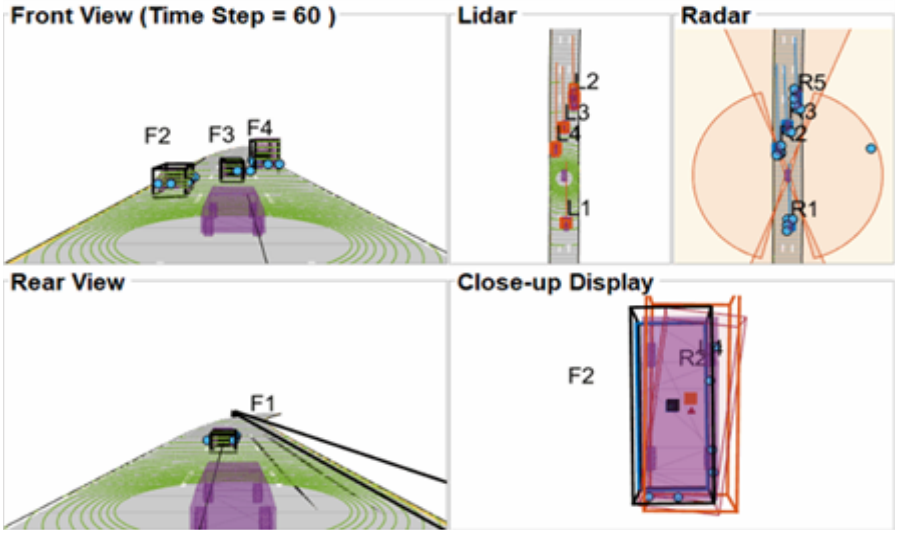

Figure 5: A visualisation of the isolation technique at work.

4.2 PID Control

Using sensor data, we can get our desired state or the goal to be achieved; to reach that desired state / position is where control engineering plays an equal role. In Lunokhod, a Proportional-Integral-Derivative (PID) control loop is implemented using the Arduino development platform that will actuate motors for the system to attain the desired state with minimum possible error.

Proportional Controller

It produces output proportional to the error signal sent in the feedback loop. Here, u(t) is the signal that is sent to actuators and e(t) is the error signal. The proportional controller is used to change the transient response as per the set goal parameter obtained from sensors.

$$u(t) \propto e(t)$$

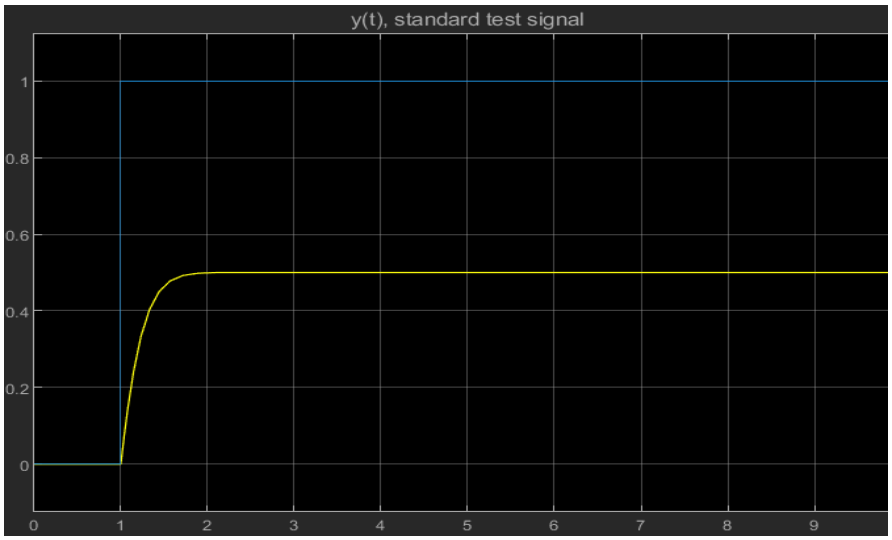

Figure 6: A graph plotted in MATLAB showing the test signal and resultant signal.

Integral Controller

It produces an output which is integral of the error signal to decrease the steady state error. It sends signal i.e., summation of all the error value occurred up to time 't'

$$u(t) \, \alpha \int e(t) \mathrm{d}t$$

Derivative Controller

It produces an output which is derivative of the error signal. It is used to make the unstable control system into a stable control system by checking the slope of the error signal after every iteration.

$$u(t) \propto \frac{d\, e(t)}{\mathrm{d}t}$$

Now, we use these concepts to write a PID loop for the actuators to be in safe operating range which is safe for the workers in the workspace. The PID loop takes the error = desired state – current state as input and gives the required rpm signal for motors to move the lunokhod to a desired state. Below flow diagram shows the functioning of PID in any system. This is important in any autonomous bot so that stability is obtained with a smaller number of oscillations to reach a set point. Figure 6 shows an overview of the PID loop.

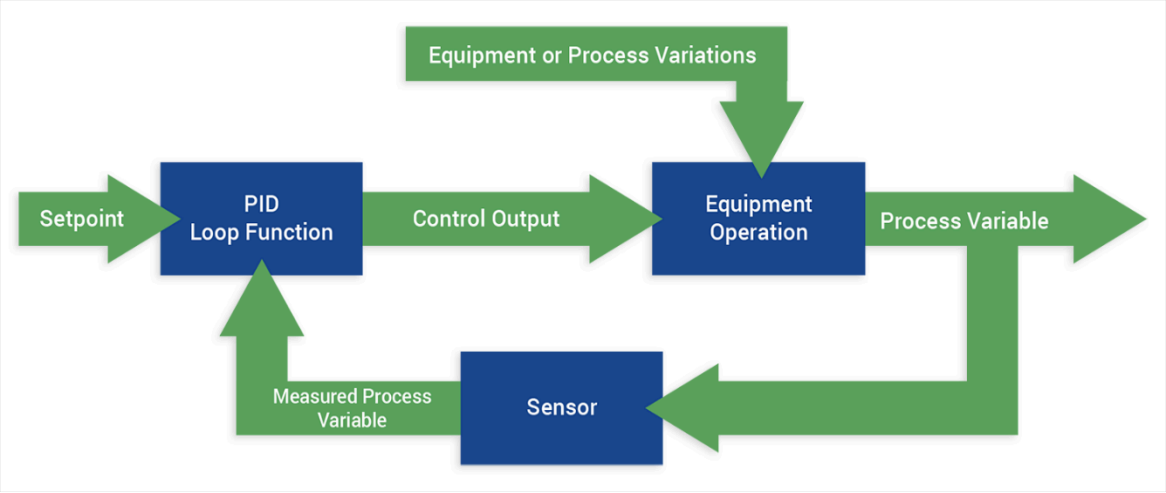

Figure 7: Schematic Block Diagram explaining the PID control loop.

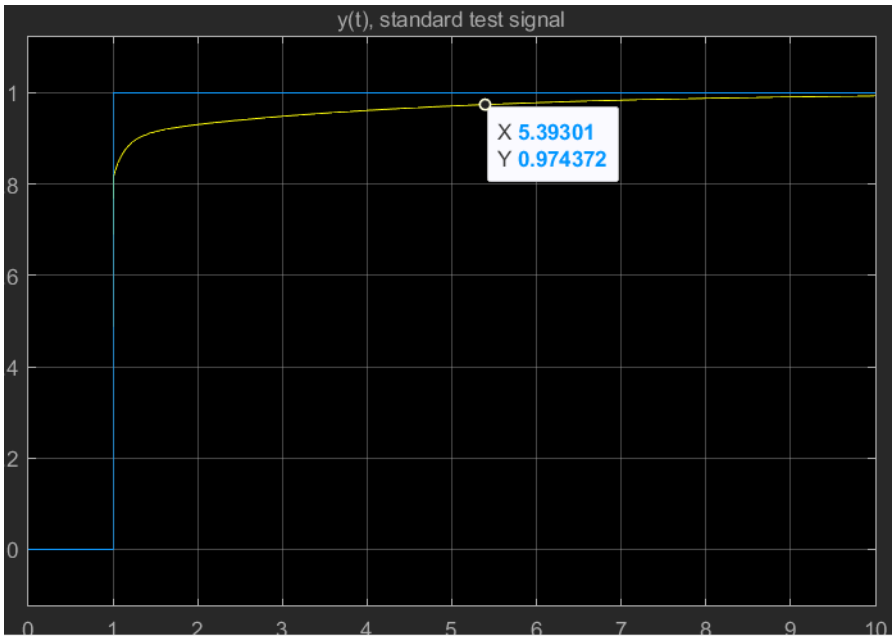

Figure 8: Graph displaying actual performance of the model with the steady state error.

5 Mechanical Design and Simulation

To implement solutions in the real world, an effective mechanical design backed up by a robust mathematical model is required. The CAD was created with Autodesk Fusion 360, while the general mathematical model was created by hand and a basic version coded using MATLAB.

5.1 Mathematical Model

A kinematic model of a mobile robot governs how wheel speeds map to robot velocities, while a dynamic model governs how wheel torques map to robot accelerations. The mobile robot is assumed to

have a single rigid-body chassis, with a configuration $T_{ob} \in SE(2)$ representing the coordinate frame of the chassis $\{b\}$ relative to a fixed original space frame $\{o\}$ in the horizontal plane.

5.1.1 Basic Model

We represent T_{ob} by the three coordinates $q = (\phi, x, y)$. We also usually represent the velocity of the chassis as the time derivative of the coordinates, $\dot{q} = (\dot{\phi}, \dot{x}, \dot{y})$. An omnidirectional mobile robot must have at least three wheels to achieve an arbitrary three-dimensional chassis velocity since each wheel has only one motor controlling its forward / backward velocity. In a coordinate frame placed at the centre of the wheel, the linear velocity of the centre of the wheel is represented by $v = (v_x, v_y)$, which must satisfy the following relation:

Equation 1

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = v_{\text{drive}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + v_{\text{slide}} \begin{bmatrix} -\sin\gamma \\ \cos\gamma \end{bmatrix}$$

Where γ is the angle at which "sliding" of the wheel occurs; for an omni wheel, this is zero, whereas a mecanum wheel has the value of $\pm$ 45°. We thus receive the following:

Equation 2

$$v_{\text{drive}} = v_x + v_y \tan\gamma,$$
$$v_{\text{slide}} = v_y / \cos\gamma.$$

Figure 9 shows the resultant model of a 4-wheeled robot using mecanum wheels:

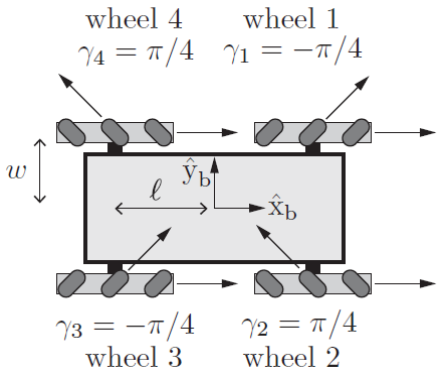

Figure 9: Basic Kinematic Model of a 4-wheeled mobile robot using mecanum wheels.

Thus, the final kinematic model for the robot is:

Equation 3

$$u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = H(0)\mathcal{V}_b = \frac{1}{r} \begin{bmatrix} -\ell - w & 1 & -1 \\ \ell + w & 1 & 1 \\ \ell + w & 1 & -1 \\ -\ell - w & 1 & 1 \end{bmatrix} \begin{bmatrix} \omega_{bz} \\ v_{bx} \\ v_{by} \end{bmatrix}$$

For the mecanum robot, to move forward, all wheels drive forward at the same speed; to move backward, wheels 1 and 3 drive backward and wheels 2 and 4 drive forward at the same speed; and to rotate in the counterclockwise direction (performing a so-called "zero radius turn"), wheels 1 and 4 drive backward and wheels 2 and 3 drive forward at the same speed.

5.1.2 MATLAB Programming

The basic model for the robot was coded in MATLAB to model a simple rigid body tree for the robot. This was primarily an explorative exercise to attempt to integrate the robot model with Simscape Multibody, the primary dynamics modelling environment for robots in MATLAB.

BASIC MATH MODEL - LUNOKHOD

GIVEN PARAMETERS

```
no_wheels = 4;
%chassis - fully covered
speed = 1; % in m/s
acc = 0.5; % in m/s^2
net_wt = 15; % in kg
payload = 1:5; % in kg
```

SOLVING

```
body1 = rigidBody('lk');
jnt1 = rigidBodyJoint('jnt1','fixed');
body1.Joint = jnt1;

lunokhod = rigidBodyTree;
addBody(lunokhod,body1,lunokhod.BaseName);
```

Figure 10: A screenshot of the basic MATLAB code for the mathematical model.

5.2 Mechanical Design

The mechanical design for the chassis, as shown in Figure 11, was inspired by the KUKA Youbot Mobile Robot, as its makeup was closest to the vision we had for the chassis, and the use of mecanum wheels as used in the Youbot are a reliable solution providing quick omnidirectional traversal on flat surfaces like warehouse floors. The robot also features a side access panel to access the electronics housed inside.

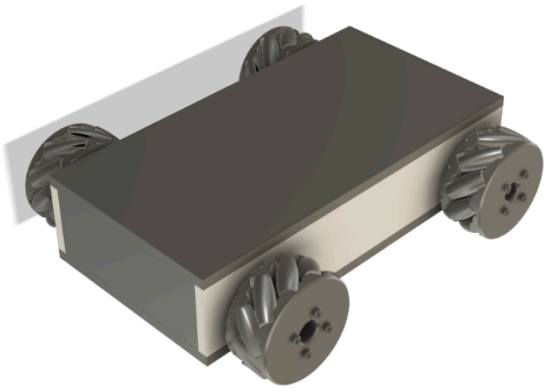

Figure 11: The CAD of Lunokhod's chassis with the wheels attached.

5.2.1 Wheels

Mecanum wheels have been selected for use as the main wheels for Lunokhod. These wheels were selected because of their easy manoeuvrability. In mecanum, wheels by varying the rotational speed and direction of each wheel, the summation of the force vectors from each of the wheels as elaborated in the mathematical model will create both linear motions and/or rotations of the vehicle, allowing it to manoeuvre around with minimal need for space. We choose these over omni wheels because:

1. Omni wheels are less efficient than mecanum in directions other than forward and backward.
2. The smaller wheels in Mecanum are bigger than the corresponding omni wheel. Thus, when the wheels are being dragged sideways if the robot is pushed sideways, the larger diameter of the large wheel permits it to roll over larger obstacles.

Figure 12: A mecanum wheel. The tilted cylindrical attachments divert the force vectors on the wheel as it rotates, thus giving it its manoeuvrability.

The mecanum wheels selected have a diameter of 152 mm; this was selected because each wheel can take a load of 15-20 kg, making the whole setup capable of resisting a load of 60 – 80kg. This is more than what we require for the target, as serves as good futureproofing for scaling up the design to a larger scale.

5.2.2 Chassis

The chassis design, inspired by the KUKA Youbot, is a fully covered body-on-frame type of chassis, wherein cover plates are installed onto a skeletal structure. This improves ease of access to the inner electronics, as any side of the chassis can be opened. It is also cheaper to manufacture owing to its non-complex design and usage of readily available prefabricated materials; our aim was to make the chassis as easy to make as possible, to cut down on unnecessary costs that can be better diverted to better electronics and sensors.

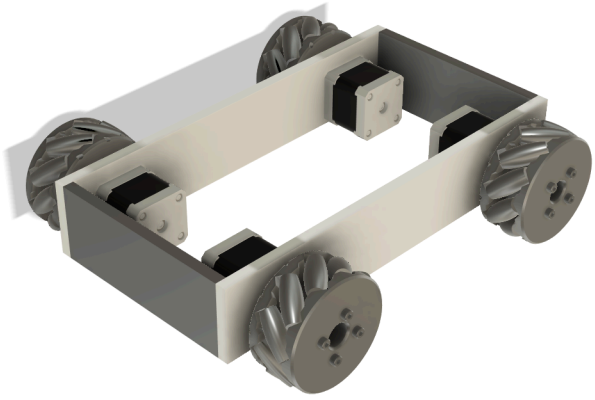

Figure 13: Opened-up chassis, showing the motors attached to the wheels.

---

6 Conclusion and Future Scope

Conclusion

The project was able to reach a satisfactory conclusion with completion of the Simulation Stage. While not all the proposed features and functionalities were achieved by the projected deadlines, we were able to finalise a working implementation that may be improved upon in the future. Unfortunately, at the time of writing in May 2021, prevailing pandemic conditions made it unable for us to proceed to the Prototyping stage for the project due to unavailability of logistics. We estimate that once situation becomes more manageable, we will be able to proceed with sourcing parts and materials for proceeding with the Prototype phase.

Future Scope

As this project was a parallel effort with our other endeavour, Project Goertz, a 6-DoF Industrial Arm, we envision a combined implementation of both projects as a single integrated product that will bring a lot of added industrial automation value. Features and functionalities that were left behind in the first attempt may be retackled with renewed knowledge and vigour. An improved transfer function of the real-time model for Lunokhod, with an analysis of its step response and root locus to understand and analyse the stability of the autonomous bot can be made. Advanced sensor fusion techniques and communication strategies can be explored, as can the possibility of more than one robot operating at one time, thereby creating a swarm of MIRs that can then use a Fleet Management System (FMS) to further increase productivity.
