Screw-Propelled Omni-Directional Robot: Automation of Waypoint Navigation and Control System Integration

by

Mohamed Rizwan Jelani Batcha

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

University of Alberta

Abstract

Helix-25 is a screw-propelled mobile robot designed by Copperstone Technologies® to survey hazardous ponds and fields. Currently this robot is manually operated, which may not be feasible during various circumstances. The control strategies for screw propelled robots have been limited, with a small element of control and they are all focussed on modelling. This project is a first pass effort to automate the movement of Helix -25 through waypoint navigation. A preliminary attempt to implement PID controller for automation of Helix is made as a first pass method, as dwelling into other sophisticated controllers may depend on dynamics of the terrain. In this project, GPS and IMU sensors are integrated into the Helix rover and the sensor data is fused using EK3 filter provided in-built by Pixhawk unit to get the pose estimate, and the ROS drivers provided by Copperstone Technologies® are used to send feedback-controlled commands to the motors based on PID errors. PID algorithms for roll, scroll and omni-directional modes were simulated in Gazebo using ZM robot and Helix model before being applied on field tests. Robots were made to follow preprogrammed Square, Hexagon and Circle trajectories to validate the robustness of control algorithm in various modes.

A trial field test was conducted on beach sand in a volleyball court, and it was found that Helix needs a cohesive medium to exhibit scroll and omnidirectional movement. Based on the inference, field testing is moved to hard surface medium to automate way point navigation in roll mode using the PID control on field test and the Helix is made to follow a 10 m x 10 m square trajectory successfully to prove the appropriate working of the automation controller interface. Later, a refinement in the robotic system and electronic design is suggested based on the field test learnings. Further, this proof of concept and mechatronic system integration builds a pathway for expanding research as well as facilitating tests of more sophisticated autonomous controller development on Helix -25.

Acknowledgments

'Being grateful to the opportunities is the humblest way of cognizing their values...

Being thankful is the simplest way to recognize one's effort.'

With warm regards, I express my heartfelt gratitude to my faculty advisor, Dr. Martin Barczyk for his tireless support, valuable suggestions, and precious time all through the research work. I owe him a lot more thanks for the faith and patience he endowed upon me giving me this great opportunity. I am also indebted to him for his trust in my capability with invaluable cooperation, encouragement, and challenges set forth for effectual learning. His unwavering support and understanding at various hard times have been a source of inspiration and motivation for me and I'm blessed to have worked under such a wonderful human.

My special thanks are due to Dr. Michael Lipsett, all the research scholars associated with his lab and Mechatronics lab, UofA for providing the research facilities to prepare the robot for testing. Special mention to Aravind, Shayan, Jorge, Sharayat and all others who encouraged me throughout this journey with their precious affability and valuable expertise.

Thanks to Copperstone Technologies®. for providing the support to debug and to fix the rover whenever required and to make it ready for field testing. I would like to express my appreciation to all my friends and colleagues who have helped me during my field tests in all possible ways. Many thanks to all the best of staff and my friends for the warmth and moral support they have given me here, making this institute my next home. I am obliged to the University of Alberta for providing this conducible environment and facilities for quality research. Above all, I owe almost everything I have to nature for the gift of this life and my wonderful family. Respectful thanks are due to my beloved wife for her steady support and confidence in me.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AGV – Automated Guided Vehicle	
AIV – Autonomous Intelligent Vehicle	
ATV – All Terrain Vehicle	
CCW – Counterclockwise	
CW – Clockwise	
DOF – Degree of Freedom	
ECEF - Earth-Centered, Earth-Fixed	
EK3 – Extended Kalman 3 Filter	
EKF – Extended Kalman Filter	
FMU – Flight Management Unit	
GLONASS – Global Orbit Navigation Satellite System	
GNSS – Global Navigation Satellite System	
GPS –Global Positioning System	
IMU – Inertial Measurement Unit	
INS – Inertial Navigation System	
LIDAR – Light Detection and Ranging	
LQR – Linear Quadratic Regulator	
MPC – Model Predictive Control	
NAVSTAR – Navigation and Satellite Timing and Ranging	
OMR – Omnidirectional Mobile Robot	
PID – Proportional Integral Derivative control	
ROS – Robot Operating System	
RTK – Real Time Kinematic Positioning	

SLAM – Simultaneous Localization and Mapping

SSH - Secure Shell

UAV – Uncrewed Aerial Vehicle

UGV – Uncrewed Ground Vehicle

URDF – Unified Robot Description Format

VSLAM – Visual Simultaneous Localization and Mapping

List of Symbols

δ_m : Number of Mobility Degrees

λ : Longitude in ECEF Frame

φ : Latitude in ECEF Frame

h : Altitude in ECEF Frame

ϕ : Rotation angle in X -axis; Roll

θ : Rotation angle in Y -axis; Pitch

ψ : Rotation angle in Z -axis; Yaw

v_x : Linear Velocity Component of the Rover in X-Axis

v_y : Linear Velocity Component of the Rover in Y-Axis

v_θ : Angular Velocity Component of the Rover

l_x : Distance between Front and Back Scrolls

l_y : Distance between Left and Right Scrolls

r : Radius of each Scroll

ω_1 : Angular Velocity of each Scroll

F_G: Force on the Scroll due to Gravity

F_f : Force Acting on the Scroll due to Friction

F_N : Normal Force Acting on the Scroll

F_{res}: Resultant Force Acting on the Scroll

F_{f-D} : Frictional Force Acting on the Drum of the Scroll

F_{f-Fl} : Frictional Force Action on the Flight of each Scroll

F_{Th} : Thrust Force Acting on the Flights of each Scroll during Scrolling Mode

θ_{ref} : Heading angle of the goal point

x_{ref} : X coordinate of the goal point

y_{ref} : Y coordinate of the goal point

E_{Yaw}: Error in heading angle with respect to current yaw

$E_{Position}$: Error in position with respect to the current position

Chapter 1 : Introduction

1.1. Background and Motivation

In recent years, there has been a growing interest in the development of advanced robotic systems capable of versatile locomotion in various environments [1] [2]. Mobile robotics face the challenge of finding efficient methods of locomotion that can navigate a wide range of terrains [3]. Traditional wheeled or legged robots, although suitable for certain environments, struggle to generate traction in many challenging terrains such as uneven surfaces, sandy or muddy areas, and bodies of water. Moreover, alternative bio-inspired locomotion methods like legged and crawl locomotion are not suitable for cohesive muddy terrains [4] [5].

To overcome these limitations, researchers and engineers have explored alternative locomotion mechanisms that can adapt effectively to different terrains [1]. One such mechanism is the screw movement where rotational motion transforms to linear motion. The screw-wheel or screw-propelled wheel, derived from the concept of the Archimedes screw conveyor, features a helical surface surrounding a central cylindrical shaft inside a hollow pipe [6] [7]. Screw-wheeled movement becomes omnidirectional in extreme terrains, including mud, snow, and tailings. By rotating the screw wheels with helical edges on their surface, these robots generate a resisting reaction force that provides buoyancy and enables to traverse extreme environments [8] [9]. This unique propulsion mechanism opens new possibilities for exploring previously inaccessible environments [10].

Helix – 25 is one such screw-wheeled robot developed by Copperstone Technologies®, an Edmonton based startup focused on developing a robot which can maneuver through challenging terrains. Thereby, the Helix Robot finds applications in various fields, such as environmental monitoring, and surveying tasks in remote, uneven, or hazardous areas [1].

1.2. Relevance of Thesis to Industrial partner

Despite of the promising capabilities of screw-propelled robots, challenges remain in terms of control and autonomy. The current control method for the Helix Robot relies on manual operation using an RC controller, limiting its autonomy and precision. To fully exploit the potential of these robots, feedback control systems are needed, enabling autonomous navigation, waypoint following, and adaptive responses to changing environmental conditions. Therefore, the main objective of this thesis work is to enable autonomous operation of the screw-propelled robot through waypoint navigation. The objective is to reduce the tedium of having a human operator control the rover for many hours while performing repetitive tasks such as performing bathymetric surveys on a body of water, thus also eliminates the presence of human in extreme environmental conditions.

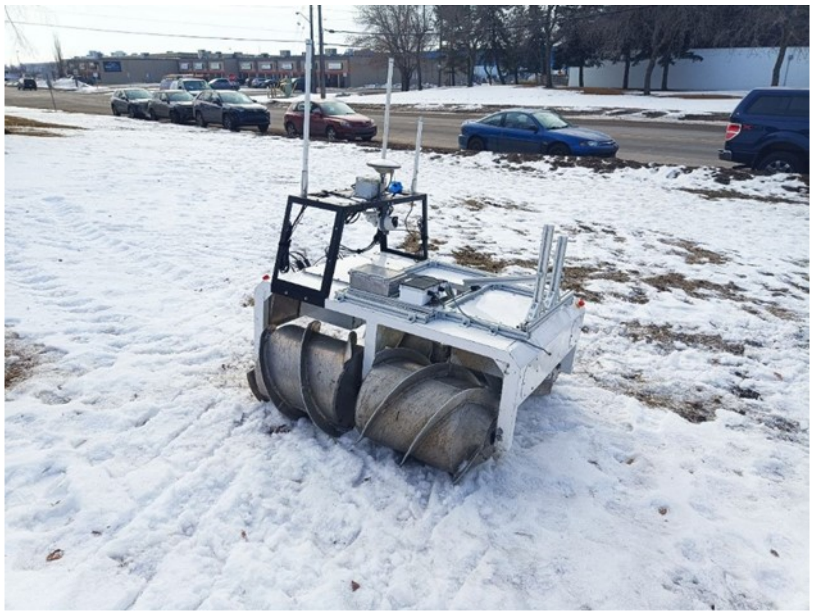


Figure 1. Screw-propelled rover traversing in snow, built by Copperstone® Technologies.

A major obstacle to achieving autonomy in the screw-propelled robot lies in the lack of reliable odometry data; these robots often experience significant slippage, making dead reckoning-based odometry unreliable. Considering these challenges, this thesis focuses to enable the robot's autonomy and navigation capabilities by implementing a GPS and IMU based pose estimation system by using commercially available sensor-fusion solution, establishing reliable communication with the robot's onboard computer [7], and applying a basic feedback control algorithm for motion control.

Another intention behind this thesis is to prepare the Helix robot for development and implementation of control strategies, starting with the basic control loop mechanism - PID. The implementation of a PID controller serves as a stepping-stone for future developments, enabling the robot to navigate autonomously along predefined paths within desired tolerance of accuracy and precision. By fine-tuning the PID controller's parameters, the robot can effectively follow waypoints, adjusting its movement based on feedback from the sensor fusion-based pose estimation system. This first pass control strategy provides a solid foundation and understanding of robot's physical limitations to be considered for future implementation of more advanced control strategies, such as model predictive control, to further enhance the robot's autonomy and navigation capabilities, considering its unique mobility characteristics and operation in diverse terrains.

1.3. Thesis Contribution

The principal contribution by thesis involves bringing up the Helix–25 and preparing it for experimental testing of autonomous control, executing a proof-of-concept with a basic control scheme. In this work, a set of PID-based autonomous control algorithms are developed and implemented for the Roll, Scroll, and

Omni-directional modes of the Helix–25. It also includes installing and establishing an interconnection between an off-the-shelf flight management unit (FMU)(PIXHAWK), a single-board computer running Linux (Jetson TX2), and the Helix's motor controller using the widely used communication protocol Mavlink® through the MavROS module running within ROS Melodic Morena. The challenging aspect during implementation is to fine tune and configure EK3 sensor fusion parameters in PIXHAWK in order to fuse data from both GPS and IMU to acquire accurate pose estimate. Prior to testing the entire system in an appropriate field, there are a set of simulation tests run that require development of a URDF (Unified Robot Description Format) describing the geometry of Helix-25. Through simulations, the control algorithms for each motion mode were validated by using an open-source 3D robotics simulator – Gazebo, linked with corresponding ROS distribution release version. The successful validation of the control algorithm in simulation as well as in a preliminary field testing occurred at a outdoor court near Lister Hall, situated near the University of Alberta campus, marking an initial assessment of the Helix – 25 robot's autonomous navigation capabilities.

1.4. Structure of Thesis

This thesis is divided into six chapters. The first chapter provides an overview of the project, highlighting the thesis objectives. The second chapter gives a comprehensive insight into the earlier works on mobile robots and explores the growth of autonomy in navigation for varied kinematic systems over time. Chapter 3 provides a comprehensive overview of the hardware and software systems integral to the project. Chapter 4 delves into the simulation of rover motion algorithms within the ROS framework, offering a detailed exploration of its key components and functionalities. It also focuses on establishing the conditioning and testing methodology on the autonomous waypoint navigation algorithms developed specifically for three modes of kinematic operation of the rover. Chapter 5 provides an account of field-testing procedures and their corresponding results, shedding light on the real-world performance of the rover and control algorithms. Finally, Chapter 6 summarizes the research's findings, concluding with a vision for suggested future work for this project.

Chapter 2 : Literature Review

In this chapter, we conduct a review of the existing literature pertaining to mobile robots, with a specific focus on three key areas: Screw-Propelled Vehicles as Mobile Robots, Aided navigation systems employed in outdoor environments, and flight management units (FMUs) used for fusing sensor data and control of autonomous vehicles. By examining these topics, we aim to gain a deeper understanding of the current state-of-the-art in autonomy of screw propelled mobile robotics and identify the gaps and opportunities for further development.

2.1. Screw Propelled Vehicles as Mobile Robots

Screw-propelled vehicles capable of navigating various terrains such as soil, marshes, snow, and ice emerged by the mid-20th century [11]. This locomotion transitioned from larger vehicles to smaller mobile robots, finding applications in diverse fields [12]. The multi-domain feasibility of screw propulsion provides numerous benefits for exploratory mobile robotics in situations where conventional locomotion methods are ineffective. Recently, Villacres et al [13] surveyed the history of screw-propelled vehicles and its usage in various commercial applications. These studies shed light on the design, performance, and possible applications of screw-propelled vehicles. However, despite these explorations, further research is needed, particularly in the areas of vehicle-specific dynamics modeling, control systems, and optimization strategies [13] [14] [15].

Mobile robot research has primarily concentrated on mechanical aspects, dynamic behaviors, and autonomous navigation methods. Various methods and techniques have been proposed and utilized on different mobile robot platforms with different wheel mechanisms and topologies. For instance, Panah et al. [16] introduced a method to reduce slippage errors by using sensors for odometry reading detached to the wheels. Stonier et al. [17] addressed the nonlinear dynamic effects of slip. Loh et al. [18] presented the design and kinematics analysis of a three-wheeled OMR with singularity solutions. Muir and Neuman [19] proposed a method to enhance dead reckoning by utilizing kinematic and feedback control for wheel slippage detection. Wong et al. [20] presented a tracking control method based on GA fuzzy control approach for a three-wheeled OMR. Liu et al. [21] developed a linearization method for controlling a three-wheeled OMR. Huang and Tsai [22] employed Backstepping control method. Velasco-Villa et al. [23] formulated passivity control. Model Predictive Control (MPC) and Nonlinear MPC (NMPC) methods have been applied to OMR navigation, incorporating potential field-based algorithms, and considering state and control variable constraints [24] [25]. Several studies have utilized MPC for OMR applications [26] [27] [28]. Araujo et al. [29] implemented a state feedback MPC-based method, while Dinh et al. [30] applied a controller based on Differential Sliding Mode Tracking (DSMT). Barreto et al. [31] enhanced an MPC approach by using friction compensation techniques. Timothy et al. [32] designed an NMPC structure based on omnidirectional motion. Cuevas et al. [33] proposed an MPC method based on a potential field approach for independently controlling three individual Mecanum wheels.

This review of mobile robot control methods & navigation techniques research to mitigate navigation errors due to environmental and dynamic influences, has significantly deepened my understanding of advanced control strategies, navigation algorithms, and effective approaches to enhance autonomous navigation and control in robotic systems. Despite the valuable contribution made by Thoesen et al [34], the references specifically addressing control methods for screw-propelled robots remain limited. Thoesen et al. introduced a helical granular scaling law for predicting output velocity and power in a screw-propelled vehicle [34]. While their work provides fundamental insights, additional research is required to develop and validate more comprehensive and specific control strategies tailored to the unique dynamics and control challenges of screw-propelled robots.

The identified research gap primarily arises from limited exploration into practical implementations and validation of these control strategies in real-world scenarios for screw propelled vehicles [35].The main limitation is that the handling characteristics and control strategies for screw propelled vehicles [36] are terrain dependent and need a model based controller to appropriately navigate different terrains like water, mud and on intermediate grounds [37]. Our project seeks to bridge a gap by focusing on the practical implementation of pose estimate and experimental validation of pilot control methodologies for screw-propelled robots which can setup the basic setup for development of model-based controller in future.

2.2. Aided Navigation Systems

Creating a screw-propelled vehicle capable of autonomously navigating various terrains begins with localizing the UGV in every environment. After localization, the UGV has the ability to retrieve, manipulate, and return a load. Commonly, a Global Navigation Satellite System (GNSS) [38] is used to determine a location in an outdoor environment. GNSS is an umbrella term for various satellite navigation system kinds. The United States government owns the NAVSTAR Global Positioning System, commonly known as GPS, which is one of the most popular GNSS. GPS services are a radio navigation system based in space that offers positioning, navigation, and timing services [39]. For GPS localization, at least four satellites must be electronically visible to the receiver. A GPS satellite transmits a radio signal with the current time and location; the distance across the transmitter and receiver can be calculated using the known speed of radio waves and the latency time between signal transmission and retrieval. Using numerous satellites and localization algorithms, it is possible to pinpoint the precise location of the receiver [40].

The accuracy of GPS is limited by the line of sight across satellites and the receiver. 95% of the time, a well-designed GPS receiver has a horizontal accuracy of three meters or greater [39]. System accuracies of up to 30 centimetres (about 11.81 inches) have been reported for GPS receivers with enhanced performance [41]. Such accuracies are adequate for tasks that do not require a high degree of precision (such as vehicle navigation), but they are inadequate for a variety of robotic applications [42]. The second limit of a GPS is noise or feeble signals in areas with obstructions. For instance, according to a study, GPS accuracy decreases by 29% in juvenile forest conditions (and by 50% under closed forest covers) compared to clear open heavens [43].

In recent years, there has been growing interest in aided inertial navigation systems, driven by the emergence of fast and cost-effective microprocessors [44]. Inertial navigation systems determine the state estimates of a vehicle or missile through dead reckoning calculations based on data from inertial sensors, measuring the vehicle's acceleration and angular velocity at high rates. However, a key limitation of inertial navigation is the accumulation of errors over time, leading to unbounded error growth, which cannot be self-corrected without external aiding measurements [44].

State estimates derived from the integration of inertial sensors can be corrected to rectify accumulated errors using an aided navigation system [45]. Aided navigation systems, where low-rate sensors like GPS correct the state estimates from high-rate inertial sensors, have gained prominence. Since the early 1960s, modern navigation has embraced hybrid (integrated) navigation systems, combining various electronic sensors to collect data necessary for continuous vehicle position determination and error reduction in inertial sensors [46]. Sindlinger [47] explored the optimization of integrated navigation systems, which amalgamated independent navigation sensors like inertial measurement units (IMUs), and radar [44].

Numerous GPS-IMU integration techniques have been explored in the literature. For instance, Grewal et al. [48] extensively discuss INS, GPS, and Kalman filtering, presenting a 54-states Kalman filter model. Magnusson delves into sensor fusion models based on the extended Kalman filter, incorporating inputs from low-grade GPS receivers, IMU sensors, and odometers to enhance absolute position estimation. Vishisht Gupta [49] focused on vehicle localization using IMUs, GPS, and a monocular camera in conjunction with an environmental map to bolster localization accuracy.

Maklouf [50] describes the integration of GPS with INS using a Kalman filter in loosely coupled mode, estimating INS error states and navigation states using GPS measurements. Martin [51] explores differential GPS methods for automated vehicle convoy positioning, detailing the Dynamic Real-Time Kinematic (DRTK) algorithm to estimate carrier phase ambiguity and relative position vectors between GPS receivers. Iozan [52] presents a Hybrid Navigation System (HNS) combining the strengths of inertial sensors and Global Navigation Satellite Systems (GNSS) for 2D navigation, even during GNSS signal unavailability or intermittency. Zhao [53] emphasizes the significance of the bridging ability of standalone IMUs during GPS signal outages, greatly influencing the performance of GPS/IMU integrated navigation systems.

2.3. Sensor Integration Units

There are two main components required in controlling a robot autonomously: a microcontroller, also termed as the vehicle controller, and sensors connected to the microcontroller. The microcontroller runs a control algorithm that uses data from the sensors and commands sent from a radio transmitter or ground station to control the robot by varying the speed of the motors [54]. The emergence of small, inexpensive and powerful flight management units has made the fusion of sensor data simpler. Some of the popular open-source firmware packages include Arducopter, Openpilot, Paparazzi, and Multiwii, flashed into FMU such as Pixhawk, Mikrokopter, Kkmulticopter, and Aerocopter [55]. These FMUs also have the capability to acquire sensor signals and fuse them to estimate the pose of the vehicle. On this list, the Pixhawk is one of the most versatile and advanced FMUs when running the PX4 open-source firmware [56]. It uses a 32-bit ARM CortexM4 processor [57]. The system includes a programming environment, allowing the implementation of complex autopilot functions and sophisticated scripting of missions and flight behavior [58]. Being open source, it has a large community of developers constantly improving its functionality and adding new features [57].The Pixhawk supports a wide range of application mobile robots, from racing and cargo drones to large fixed-wing models, and supports autonomous functionality, a key feature for many drone applications. These features make the Pixhawk the final choice for this project implementation.

Chapter 3 : Hardware System Overview

The mechanical vehicle used as the mobile base platform for this project is a screw-propelled omni-directional robot named HELIX 25. This chapter will focus on the design and build of the robot and the kinematics of its movement. It also includes the details of the essential electronics and controllers used for upgrading the robot for automation. Further, the software architecture built over the hardware is also discussed in this chapter.

3.1. HELIX 25 Rover and its Build

The HELIX 25 rover is a multifunctional and amphibious vehicle designed and manufactured by Copperstone® Technologies Ltd., a Canadian-based engineering firm in Edmonton, Alberta. The primary materials used in its making are aluminum, stainless steel, and plastics. It has four screw-shaped wheels, referred as scrolls, that can be independently driven. The construction of these scrolls facilitates amphibious capabilities for varied types of terrains and the helical shape enables omni-directional motion capabilities. Each 19-inch-long scroll is made of a 13-inch-long cylindrical drum that keeps the rover buoyant, with 17.3-inch-wide helical flight around the drum, that makes the Archimedean screw design. The movement of the rover along the longitudinal axis of the scrolls is called scrolling, while the movement of the rover perpendicular to this axis is called rolling (Refer to Figure 2 for fixed-body coordinate frame).

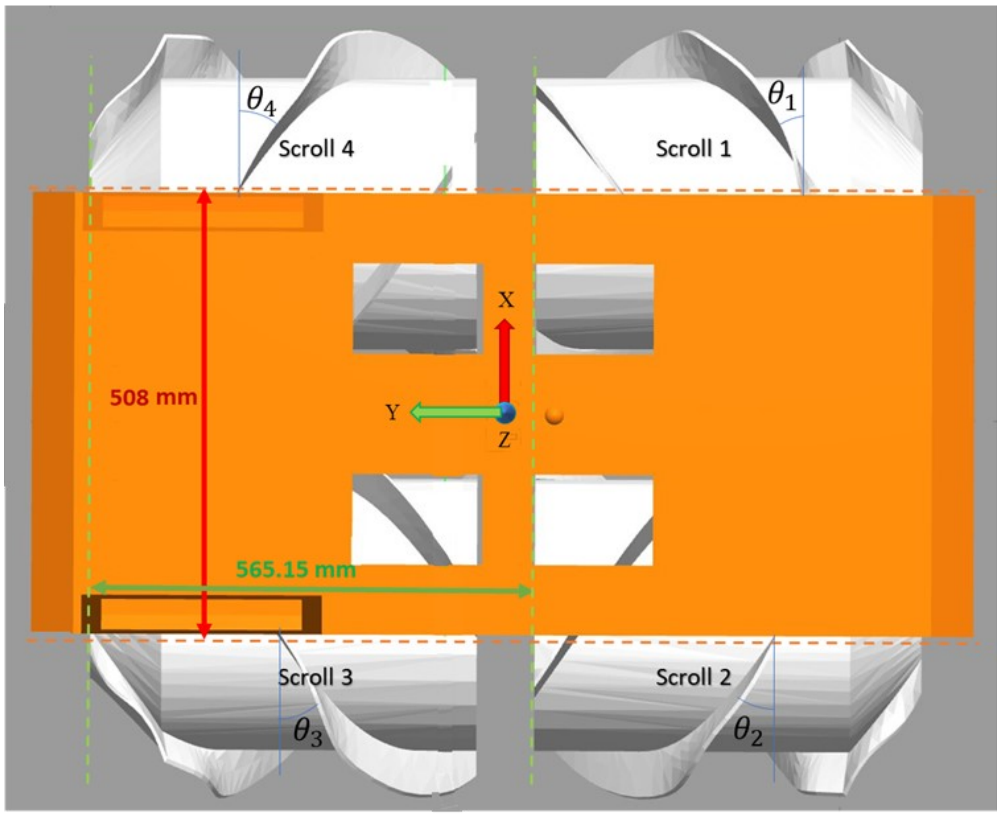


Figure 2. HELIX 25 Rover – Top View: The coordinate axis of body-fixed frame and the scroll IDs

As per the axes shown in Figure 2, the linear forward and backward motion along Y-axis happens in Scrolling mode, while the linear forward and backward motion along X-axis happens in Rolling mode. The scrolling movement is used for propulsion in water, snow or soft sand, and the rolling movement is used for hard surfaces. The rotational movement of rover, which determines the heading angle of the rover, is called steering. The steering movement is achieved by maintaining differential speed proportions of individual scrolls relative to each other. The coordinate axes considered for our equations and the numbering scheme followed for each of the scrolls is also mentioned in Figure 2, which will be used in the further sections of this chapter where we explore the kinematic equations of the rover's movement. However, it must be noted in the design of Helix rover that all the four scrolls are mounted such that the relative helical winding angle of each scroll complements the adjacent one (Refer to $\theta_{1,2,3,4}$ in Figure 2). That makes it more stable and flexible for more control in motion. Scroll 2 is complementary to Scroll 1 for motion guidance, stability, and more control in motion. That way, Omni-directional movements and rotation/turns are made possible. Scroll 4 is similar to Scroll 2, flipped in the opposite direction for reverse motion capability. As the HELIX rover is a professionally developed product used for monitoring in hazardous environments, it is equipped to be controlled remotely with a manual hand-held transmitter, a similar teleoperation unit called Ground Control Station (GCS). All these operations, however, do include emergency stop switches for freezing the entire system immediately in cases of any emergency situations.

3.2. Kinematics of Rolling

In hard terrains, the scrolls act like wheels where the contact of rover and the terrain surface is only at the outer tips of the flights. Irrespective of the helical winding of flights, the resulting velocity of each scroll at the point of contact owing to reaction force from the terrain surface is perpendicular to the longitudinal axis of each scroll (Refer Figure 4**)**. Therefore, the resultant motion of the center of gravity of the rover is also perpendicular to the longitudinal axis of the scrolls, that is in its X-direction, positive for clockwise rotation of all scrolls and negative for counterclockwise rotation of all scrolls as shown in Figure 3.

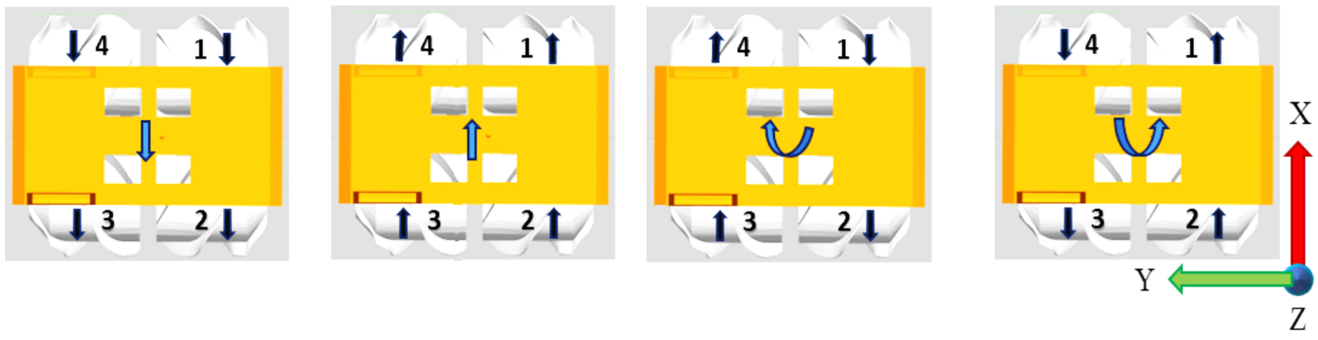


Figure 3. HELIX 25 Rover – ROLL mode: The dark blue arrows on each scroll represent the direction of motion of each scroll and the light blue arrow in the center of the rover represents the direction of motion of the rover, effectively.

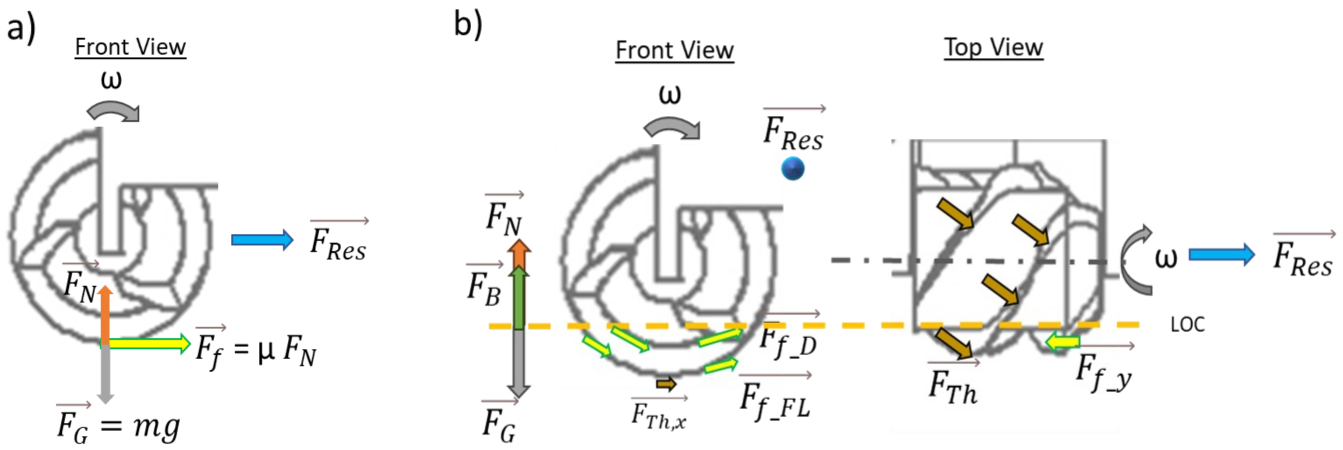

a)
Front View
ω
$\overrightarrow{F_{Res}}$
$\overrightarrow{F_N}$
$\overrightarrow{F_f} = \mu F_N$
$\overrightarrow{F_G} = mg$
b)
Front View
Top View
ω
$\overrightarrow{F_{Res}}$
$\overrightarrow{F_N}$
$\overrightarrow{F_B}$
$\overrightarrow{F_G}$
$\overrightarrow{F_{f\_D}}$
$\overrightarrow{F_{f\_FL}}$
$\overrightarrow{F_{Th,x}}$
$\overrightarrow{F_{Th}}$
$\overrightarrow{F_{f\_y}}$
ω
LOC
$\overrightarrow{F_{Res}}$

Figure 4. Free-body Diagram of Scrolls during the motion in (a) Roll Mode and (b) Scroll Mode: The gray curved arrow represents the rotational direction (ω); The gray arrow represents the direction of gravitational force (F_G); The orange arrow represents the direction of normal reaction force from ground (F_N); The brown arrows represent the direction of thrust force (F_{Th}); The blue arrow represents the direction of resultant force (F_{Res}) and the effective motion of the rover.

The rover is steered by skid movement to turn left or right with rotation of one pair of scrolls opposite to the rotation direction of other pair. The pairing sets of the scrolls for this action are: {Scroll 1, Scroll 2} and {Scroll 3, Scroll 4}. This essentially signifies that the net velocity of rover in Y-direction is always zero in ROLL mode of motion. In Figure 4 (a), the free-body diagram of a scroll during the ROLL mode shows that the vertical force due to gravity F_G acting on the mass of the scroll as well as the loading force on it faces a normal reaction force F_N at the point of contact on the hard ground. For the given direction of rotation of the wheel ω, the frictional force F_f proportional to the normal force acts laterally, leading to a resultant force F_{Res} as shown.

3.3. Kinematics of Scrolling

In fluidic terrains, the scrolls act like Archimedean screws where the scrolls have contact with the medium through the width of flights as well as the drum. In this way, the scrolls are partially or fully engulfed in the medium depending on its type, where all the forces cancel each other except for the propulsion forces ideally (Refer Figure 4).

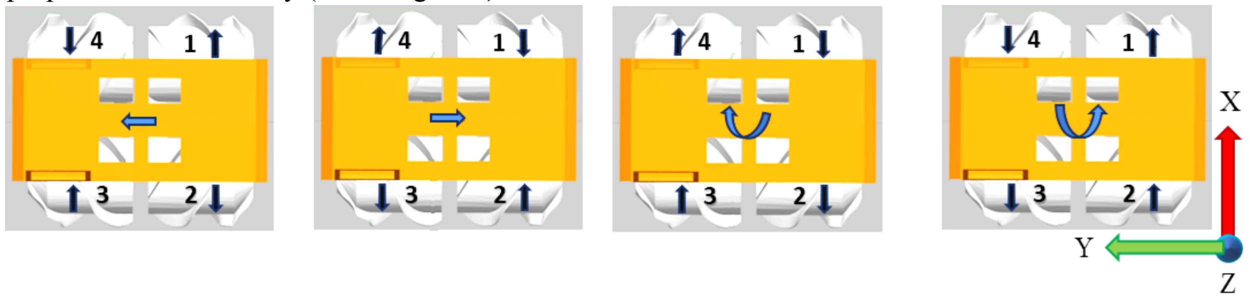


Figure 5. HELIX 25 Rover – SCROLL mode: The dark blue arrows on each scroll represent the direction of rotation of each scroll and the light blue arrow in the center of the rover represents the direction of motion of the rover, effectively.

These propulsion forces need not be purely aligned with the longitudinal axis of scrolls as the frictional effect caused by drum movement relative to medium causes slippage and the scrolls are not guided. Therefore, more than two scrolls are used for this rover and are also paired for scroll mode such that the rotation of scrolls effectively results in net velocity along the longitudinal axis of each scroll, like that of the motion of a screw. Consequently, the resultant motion of center of gravity of the rover is also along the longitudinal axis of the scrolls, that is in its Y-direction as shown in Figure 5.

The rover is steered by skid movement to turn left or right with rotation of scroll in each pair opposite to the rotation of other one. The pairing sets of the scrolls for this mode are: {Scroll 1, Scroll 3} and {Scroll 2, Scroll 4}. This essentially signifies that the net velocity of rover in X-direction is always zero in SCROLL mode of motion.

In Figure 4 (b), the free-body diagram of a scroll during the SCROLL mode shows that the vertical force due to gravity F_G acting on the mass of the scroll as well as the loading force on it faces a normal reaction force F_N and the buoyancy force F_B as per the medium of terrain and extent of engagement of the scrolls. For the given direction of rotation of the wheel ω, the frictional force F_f proportional to the normal force acts parallel to the surface of contact at both the drum surface F_{f_D} as well as the flight surfaces F_{f_FL}. Further, as per the direction of ω and the helical design of the scroll, the thrust force F_{Th} acting on the flights has components along the longitudinal axis of the scroll as well as perpendicular to it. This thrust force pushes the scroll in the longitudinal direction, thereby facing a frictional force F_{f_l} against the thrust direction, leading to a resultant force F_{Res} as shown.

This Helix rover is facilitated with a manual switch for the two modes, ROLL and SCROLL, to be supported by the same joystick controls of the remote. The built-in on-board motor-controller on Helix rover translates the manual joystick control inputs to respective scroll rotations with the rotational speed of each scroll proportional to the magnitude of input.

3.4. Kinematics of Holonomic Motion: Scrolling and Rolling

In viscous terrains, the scrolls can be controlled with independent rotational movements to combine the mechanics of both scrolling and rolling modes, such that it results in the holonomic movement. The resultant motion of the center of gravity of the rover in this mode would be in any quadrant of the XY coordinate frame, with a heading direction dependent on the resulting velocity vector of component velocities in X and Y direction respectively. Considering the ideal conditions of operation for a controlled holonomic motion of the robot, the required movement of the scrolls relative to each other is illustrated in Figure 6.

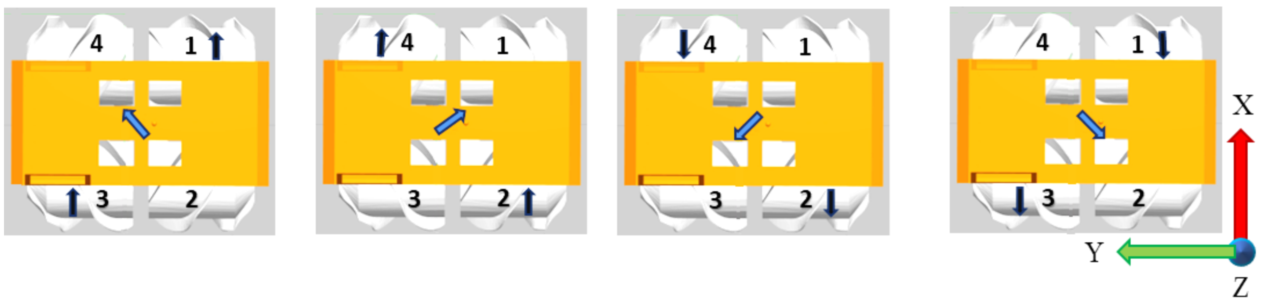


Figure 6. HELIX 25 Rover – HOLONOMIC mode: The dark blue arrows on each scroll represent the direction of rotation of each scroll and the light blue arrow in the center of the rover represents the direction of motion of the rover, effectively.

Holonomic motion implies that a system can move freely without constraints limiting its motion. While commonly discussed in planar motion, the concept can be extended to three-dimensional space and generalized to manifolds. Depending on the viscosity and density properties of the media, the scrolls may be partially or fully engulfed. Based on the viscosity of the liquid, the thrust force required to move the rover varies. All these analyses are applicable only for Newtonian fluids where the viscosity remains constant under shear stress. For certain types of media where the scroll is partially immersed, it is possible that the propulsion forces along the longitudinal axis of scrolls are accompanied by reaction force of the terrain along the length of the flight. That implies forces in two different directions acting on the rover, one propelling it longitudinally and the other pushing it sideways laterally. The lateral force direction need not be exactly perpendicular to the longitudinal axis; Its magnitude and direction will be highly dependent on the specific properties of the medium and a deeper understanding with more experiments is needed to model the same.

Furthermore, with precise control of independent scroll rotational direction and magnitude, variable combinations of velocities can be achieved that can lead to steering movements of the rover on the spot with zero turning radius, thus making it more flexible and adaptable for hazardous and narrow terrains as well, subject to the interaction mechanics of scrolls with the medium. The desired steering speed, the overall yaw magnitude along global Z axis, is achieved through appropriate control of proportional speeds of the four scrolls as explained for all the practical cases.

With this basic understanding of direction of movements, the next step is to derive the necessary magnitude of movement. The equations governing the kinematics of the HELIX through independent movement of the scrolls. Using the X axis, Y axis and Z axis (for rotation in θ) given in Figure 3, the kinematic equations for rover movements are given below, considering that the scrolls are paired, and their movement is synchronized respectively for each of the modes as additional constraints i.e. $v_x = 0$ for scroll, $v_y = 0$ for roll and $v_\theta = 0$ for omni directional mode, to these equations under no slip condition:

$$v_x = \frac{r}{4}\,(\omega_1 + \omega_2 - \omega_3 - \omega_4)$$

$$v_y = \frac{r}{4}(\omega_1 - \omega_2 - \omega_3 + \omega_4) \qquad \ldots (1)$$

$$v_\theta = \frac{r}{4\left(l_x + l_y\right)}(\omega_1 + \omega_2 + \omega_3 + \omega_4)$$

Given than all scrolls are identical and of equal radius, $r_1 = r_2 = r_3 = r_4 = r$ for solving the equations to give appropriate command values of ω_i for each scroll, that gives the following final set of expressions for each scroll's rotational speed in equation 2, provided l_x is the distance between front and back scrolls, and l_y is the distance between left and right scrolls as shown in Figure 2. Although, it must be noted that these physical parameters do not comprehensively reflect the dynamic parameters that are affected by the media properties, such as the effective radius of scroll, effective change in center of gravity of the rover, the slippage considerations and so on. That study requires an extensive set of experiments in a set of controlled environments to properly identify the interactions between the scrolls and the media.

$$\omega_1 = \frac{1}{r}\left(v_x + v_y + \left(l_x + l_y\right)v_\theta\right)$$

$$\omega_2 = \frac{1}{r}\left(v_x - v_y + \left(l_x + l_y\right)v_\theta\right)$$

$$\omega_3 = \frac{1}{r}\left(-v_x - v_y + \left(l_x + l_y\right)v_\theta\right) \qquad \ldots (2)$$

$$\omega_4 = \frac{1}{r}\,\left(-v_x + v_y + \left(l_x + l_y\right)v_\theta\right)$$

As per the hardware dimensions and measurements, the values of known physical parameters are: r = 175 mm, l_x = 565.15 mm, l_y = 508 mm. Thus, the individual rotational velocity commands to the scrolls are given in RPM to the on-board low-level controller when set in GCS mode, for a desired rover movement towards a target position, which is computed by the high-level controller as per the current state of the rover. The RPM commands to each of the scrolls are sent relative to the maximum capacity of the scroll in percentage. It is also essential to note that, in GCS mode of operation, in order to enable Scroll mode of motion, the velocity along X-axis is preset to zero, and to enable Roll mode of motion, the velocity along Y-axis is preset to zero throughout the operation. In the holonomic mode, the rotational velocity about the Z-axis, i.e., for the turns, is set to zero, which implies that there is no heading adjustment enabled. Further, the pairing of the scrolls corresponding to each mode and their synchronization is done as mentioned above.

The rover also has an additional payload capacity to house essential electronics, sensors and edge-computational devices required for advanced control. The next section elaborates on the control interface hardware developed and integrated for autopilot mode.

3.5. Electrical Hardware Architecture

The rover is incorporated with computational units to facilitate real-time pose estimation and control. It features onboard GPS and IMU sensors for accurate localization and navigation. At the core of the electronic system is the Pixhawk Cube Black flight management unit, which can operate with either PX4 or Ardupilot® firmware. This unit acts as a hub for collecting sensor data, executing sensor fusion algorithms, and managing the overall pose estimation operations of the rover.

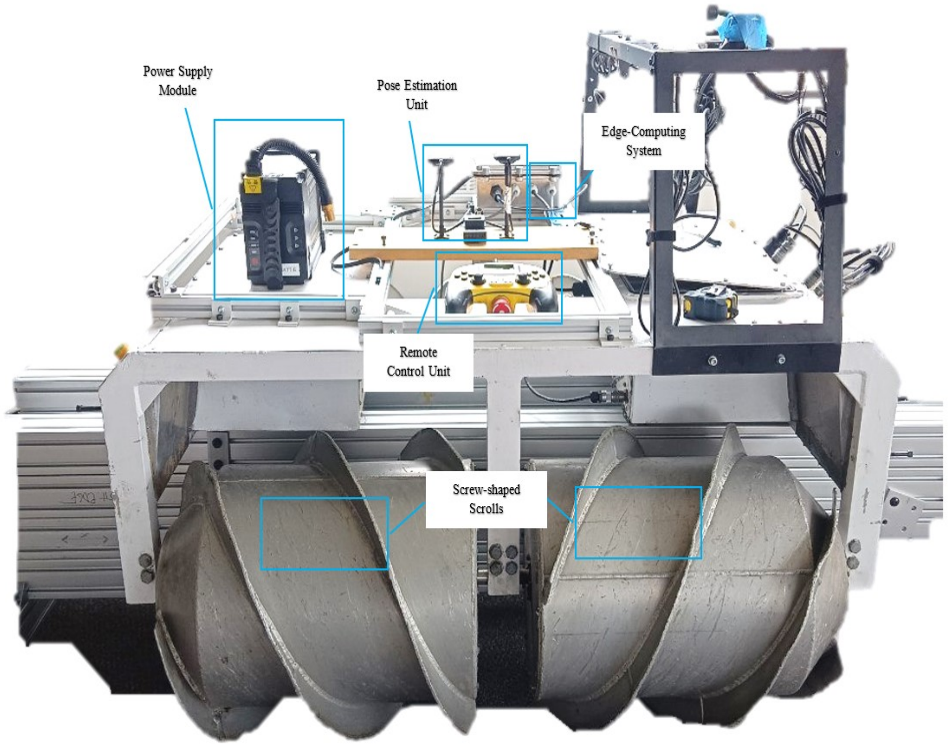


Figure 7. Overview of hardware system with sensors and electronic modules mounted on Helix.

In addition to the flight management unit, the rover is equipped with an NVIDIA Jetson® TX2 onboard computer (Refer Figure 7). This computing unit runs ROS Melodic version of ROS framework. The Jetson® TX2 is used to read in the pose estimates from the FMU, execute waypoint navigation control algorithms, and communicate with the Helix's onboard motor controller board through MAVROS driver provided by Copperstone Technologies®. The software stack run on this computer includes the implementation of a waypoint algorithm specifically tailored for the HELIX rover to navigate to the waypoint autonomously.

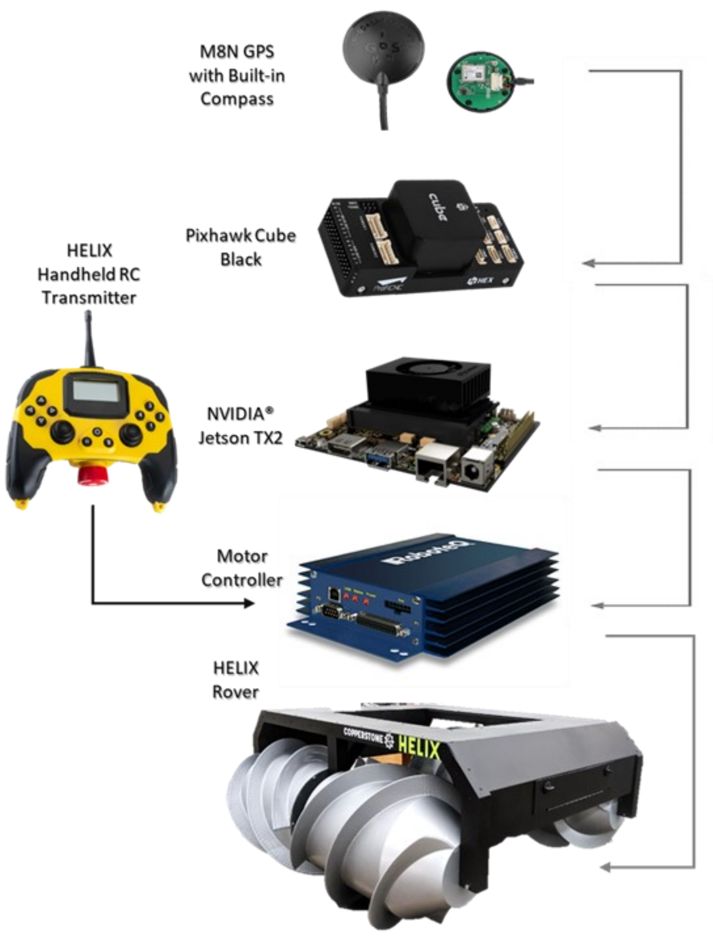


Figure 8. Major electronic components interfaced with Helix for remote and autonomous control.

The electronic system, as depicted in the provided Figure 8, forms the hardware framework for control interface of the Helix rover which has direct access to on-board motor controller. This is being controlled either by the program running on Jetson for autonomous navigation or by the human operator using RC to takeover functionalities as it allows us to regain control in case of emergency or when the edge computer turns off.

3.5.1. On-board Computing Module

The onboard computer possesses the necessary computational capacity to execute the software stack, which includes a node to receive pose estimates from FMU and control strategies for navigation.

The Jetson® TX2 module features a dual-core NVIDIA Denver 2 CPU and a quad-core ARM Cortex-A57 CPU, offering a balance between high-performance and power efficiency. It also includes a 256-core NVIDIA Pascal GPU for accelerated computing tasks. With 8 GB of LPDDR4 RAM, the module provides sufficient memory and threads for complex computations and data processing. Operating under a Nvidia's customized Linux-based system, specifically NVIDIA L4T 32.2.3 [59].

The Jetson® TX2 [59] module is equipped with multiple interfaces, including USB 3.0 ports, Ethernet connectivity, and support for display and storage devices. The Jetson® TX2 module is ideal for running and processing various tasks, including computationally intensive algorithms. The NVIDIA JetPack software, which includes the Jetson® Linux Driver Package and CUDA libraries, enabling efficient utilization of the module's GPU for tasks such as optimization, and navigation.

Figure 9. The edge-computational onboard fast processing unit: NVIDIA® Jetson® TX2 [59].

3.5.2. Flight Management Unit (FMU)

The flight management unit utilized in our case is the Pixhawk Cube Black. This flight controller is based on the Pixhawk FMUv3 open-source project and it is used only as a source of pose estimation. Pixhawk Cube Black equipped with 32-bit Arm Cortex-M4 running at 168MHz with 2MB of memory and 512KB of RAM, the processor is capable of handling complex sensor fusion algorithms and calculations. [60]

In addition to the processor, the Pixhawk Cube Black includes three Inertial Measurement Units (IMUs). The Pixhawk Cube Black also features a barometer for altitude measurement and a vibration isolation system to minimize disturbances from external sources. One of the key advantages of the Pixhawk Cube Black is its compatibility with two popular open-source autopilot firmware projects: PX4 and ArduPilot [61]. Both firmware options provide a wide range of tools and capabilities for our design solutions and applications. Both PX4 and ArduPilot were extensively tested in our rover application, and based on the performance evaluation, the ArduPilot is chosen as it allows various customized tuning EKF parameters.

The Pixhawk Cube Black flight controller offers sensor fusion capabilities through its EK3 algorithm, which combines data from the Inertial Measurement Unit (IMU) and GPS sensors. By integrating these sensors, the EK3 algorithm estimates the pose of the rover. The IMU contributes high-frequency measurements, while the GPS provides absolute position information at a lower rate.

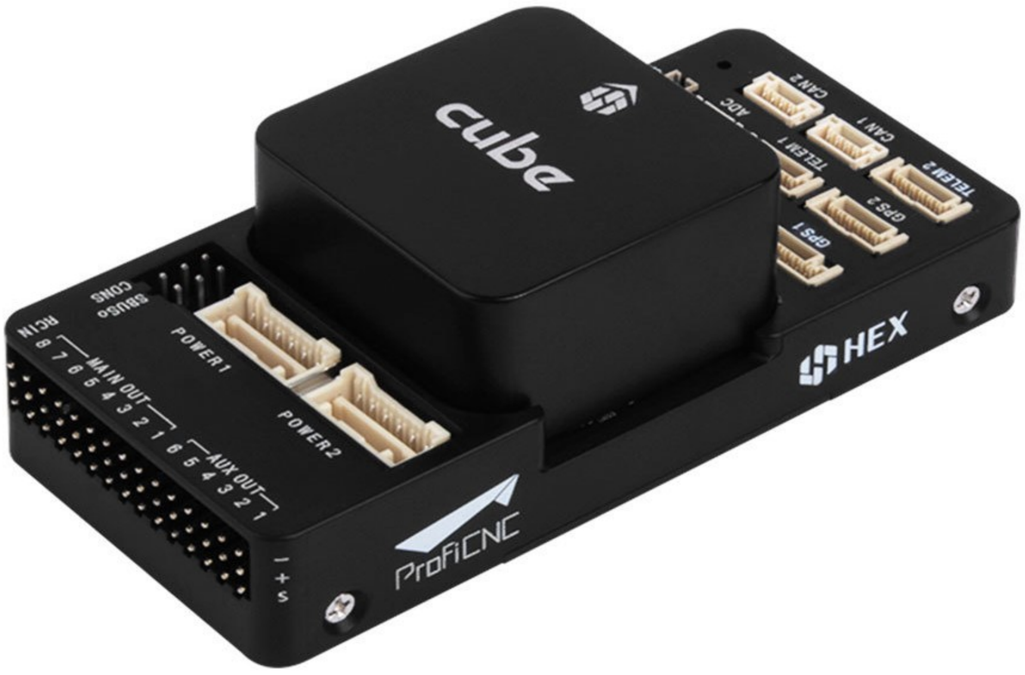


Figure 10. Low-level flight controller repurposed for ground rover application for sensor data fusion: Pixhawk Cube Black. [61]

Here are the detailed specifications and descriptions of the sensors integrated into the Pixhawk Cube Black: [62]

IMU. The Inertial Measurement Unit (IMU) integrated into the Pixhawk Cube Black flight controller consists of three-axis accelerometers and three-axis gyroscopes. The accelerometers have a resolution of up to 16 bits and can measure accelerations with a range of +/- 16g (g = acceleration due to gravity on earth). The gyroscopes have a resolution of up to 16 bits as well and can measure angular rates with a range of +/- 2,000 degrees per second. These sensors are used by EK3 as one of the sources for motion sensing to estimate poses. [60]

Barometer. The Pixhawk Cube Black flight controller includes a barometer that provides altitude measurements and atmospheric pressure readings. The barometer sensor has a typical range from 10 to 100 Pa, allowing to detect a change in air pressure. It operates based on the principle of barometric pressure sensing, utilizing a pressure-sensitive diaphragm to measure the ambient pressure. The barometer's resolution is typically in the range of 0.01 to 0.1 Pa, enabling altitude estimation with a resolution of a few centimeters. [60]

Magnetometer and Compass. The Pixhawk Cube Black flight controller is also equipped with a magnetometer and compass sensor, specifically the Honeywell HMC5883L. The HMC5883L is a three-axis magneto-resistive sensor designed to measure magnetic fields. It has a measurement range of ±8 Gauss and a resolution of 2 milligauss. The sensor operates on the principle of Anisotropic Magnetoresistance.

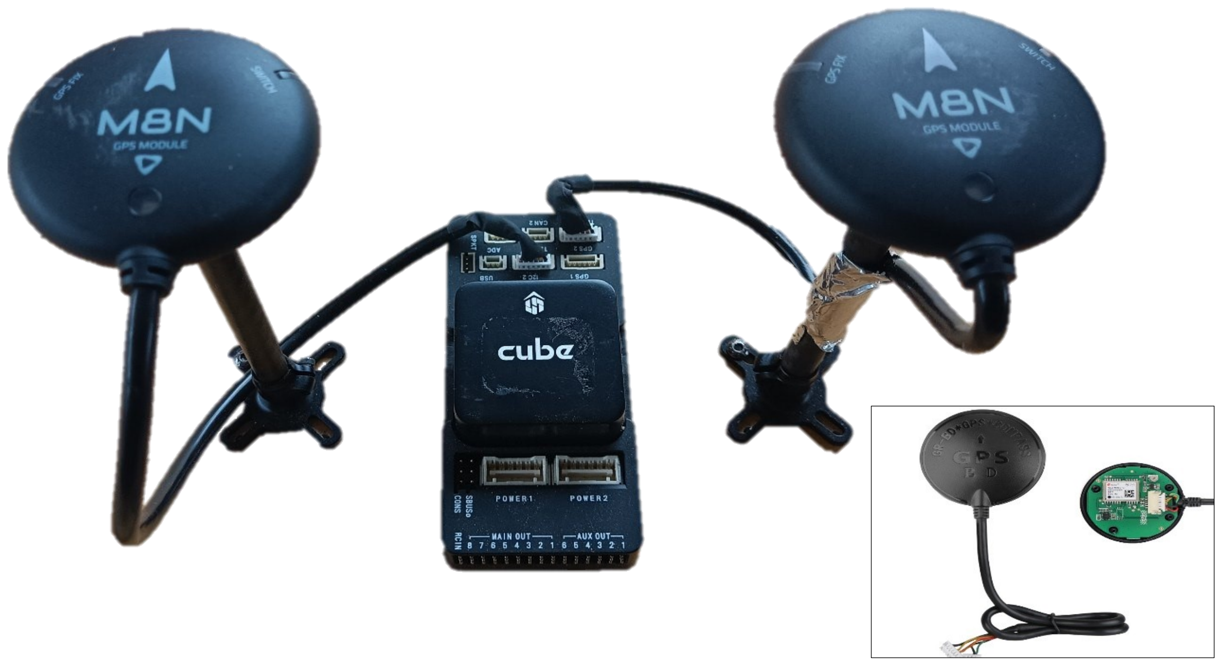


Figure 11. Interfaced module of two M8N receivers with Pixhawk Cube. Inset: GPS signal receiver M8N with a built-in BDS compass module and antenna.

The HMC5883L also includes built-in automatic calibration capabilities, which helps in heading measurements. This serves as one of the sources of heading angle in EKF to measure yaw. [60]

GPS Receiver. The M8N GPS receiver module is utilized in the thesis project for precise positioning and navigation. The module is based on the u-blox M8N chipset, which provides high-performance positioning with a wide range of GNSS, including GPS, GLONASS, Galileo, and BeiDou [63].

The Pixhawk Cube Black features dedicated GPS ports, making it easy to connect the M8N module to the flight controller. The module communicates with the Pixhawk Cube Black using the Universal Asynchronous Receiver-Transmitter (UART) interface, transmitting the GPS data to the flight controller for processing. Dual GPS usage is also supported by the Pixhawk Cube Black, it is taken advantage in this project to use dual GPS receiver to estimate yaw. It is done because the magnetometer data reading is not reliable as it is deep inside the Pixhawk and can miss subtle changes in magnetic field. [60]

3.6. Software Architecture

In the software architecture of the project, the data from the two GPS receivers, IMU, compass, and barometer sensors are fused using the EK3 architecture within the Pixhawk Cube Black flight controller. This fusion process provides odometry information about the rover's position and orientation in the physical environment. The pose estimate is then transmitted to the high-level Jetson® onboard computer using the MAVROS communication interface, which is used to communicate between the flight management unit (FMU) and the Jetson® TX2.

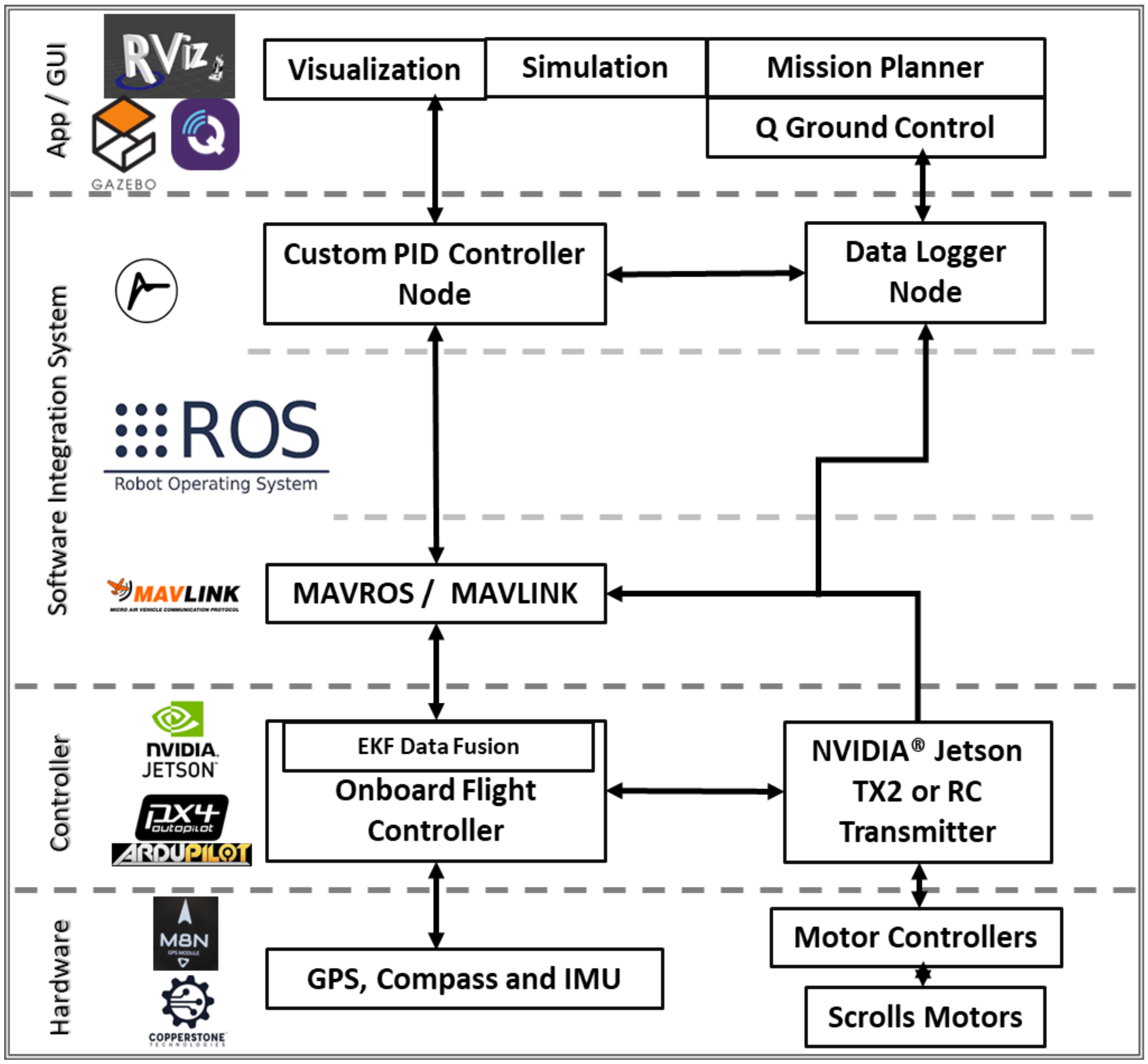


Figure 12. Overview of overall software architecture and the developed nodes of communication.

On the Jetson®, a custom programmed and compiled Python ROS node is implemented to utilize the global pose estimate and errors for the waypoint algorithm. This algorithm receives the pose updates and calculates the motor commands required for the rover's movement towards goal. The motor commands are sent from the Jetson® to the scroll motors of the Helix rover, passing via a PID-based motion controller. Concurrently, two MAVROS nodes run on the Jetson®: one node receives the pose estimate from the Pixhawk, while the other node sends motor commands to the rover. The motion planner or RVIZ visualization tool can be used to visualize the rover's trajectory and monitor its motion on real time to know if the rover is following given trajectory or not.

To facilitate communication between the high-level Jetson® and an external laptop or base station, VNC protocol or SSH can be employed. This allows for the sending of commands to Jetson® or receiving data from it remotely. Additionally, the waypoint algorithm and the rover's behavior can be simulated in Gazebo, a robot simulation environment, providing a virtual representation of the robot's model, its kinematics, and dynamics, enabling rigorous existential testing and validation before deployment. The overall architecture, the tools used for software development, the communication paths connected and how all the systems interact seamlessly in relation to individual modules are demonstrated and also shows how the units interact with the graphical interface is shown Figure 12.

3.6.1. Pose Estimation in Pixhawk using EK3

The Estimation and Control Library (ECL) in Pixhawk uses an Extended Kalman Filter (EKF) algorithm to process sensor measurements and provide an estimate of Quaternion defining the rotation from North, East, down local earth frame to X, Y, Z body frame [64]. The EKF runs on a delayed 'fusion time horizon' to allow for different time delays on each measurement relative to the IMU. The default behavior is to run a single instance of the EKF. Depending on the number of sensors it is decided to run 6 instances of the EKF to predict the pose estimate as shown in Figure 13.

For optimal pose estimation performance, parameter tuning of Extended Kalman Filter (EKF) algorithm in mission planner is conducted based on iterative practical experience in the ambient conditions. Below Table 1 gives the list of main parameters changes manually based on field experience after loading the standard parameters into the Pixhawk. These parameters are the parameters directly depending upon the sensor bias, sensor noise and working range. Parameters values are tweaked based on the selected M8N GPS sensor module and it is noise range in the ambient environmental conditions. Ranges for the tweaking are found from the literature or in the sensor manuals and based on the initial experience in the field when working with the sensor system.

Table 1. Pixhawk EKF tuned parameters.

Parameter Name	Value	Units	Range	Description
EK3_ABIAS_P_NSE	0.003	m/s/s/s	0.00001 - 0.02	This noise controls the growth of the vertical accelerometer delta velocity bias state error estimate.
EK3_ACC_BIAS_LIM	1	m/s/s	0.5 - 2.5	Accelerometer bias state is constrained within the specified value. Increasing it makes the filter trust the accelerometer measurements less and other measurements more.
EK3_ACC_P_NSE	0.35	m/s/s	0.01 - 1.0	This control disturbance noise controls the growth of estimated error due to accelerometer measurement errors excluding bias.
EK3_ALT_M_NSE	2	m	0.1 - 100.0	This is the RMS value of noise in the altitude measurement. Increasing it reduces the weighting of airspeed measurements
EK3_EAS_I_GATE	400		100 - 1000	This sets the percentage number of standard deviations applied to the airspeed measurement innovation consistency check.
EK3_EAS_M_NSE	1.4	m/s	0.5 - 5.0	This is the RMS value of noise in equivalent airspeed measurements used by planes. Increasing it reduces the weighting of airspeed measurements.
EK3_ERR_THRESH	0.2		0.05 - 1	Lane consistency threshold determines the necessary error level difference to reduce overall relative core error during lane switching.

EK3_GYRO_P_NSE	0.015	rad/s	0.0001 - 0.1	This control disturbance noise controls the growth of estimated error due to gyro measurement errors excluding bias.
EK3_POS_I_GATE	500		100 - 1000	This sets the percentage number of standard deviations applied to the GPS position measurement innovation consistency check.
EK3_POSNE_M_NSE	0.5	m	0.1 - 10.0	This sets the GPS horizontal position observation noise.
EK3_VEL_I_GATE	500		100 - 1000	This sets the percentage number of standard deviations applied to the GPS velocity measurement innovation consistency check.
EK3_VELD_M_NSE	0.7	m/s	0.05 - 5.0	This sets a lower limit on the speed accuracy reported by the GPS receiver that is used to set vertical velocity observation noise.
EK3_VELNE_M_NSE	0.7	m/s	0.05 - 5.0	This sets a lower limit on the speed accuracy reported by the GPS receiver that is used to set horizontal velocity observation noise.
EK3_YAW_I_GATE	300		100 - 1000	This sets the percentage number of standard deviations applied to the magnetometer yaw measurement innovation consistency check.
EK3_YAW_M_NSE	0.5	rad	0.05 - 1.0	This is the RMS value of noise in yaw measurements from the magnetometer. Increasing it reduces the weighting on these measurements.

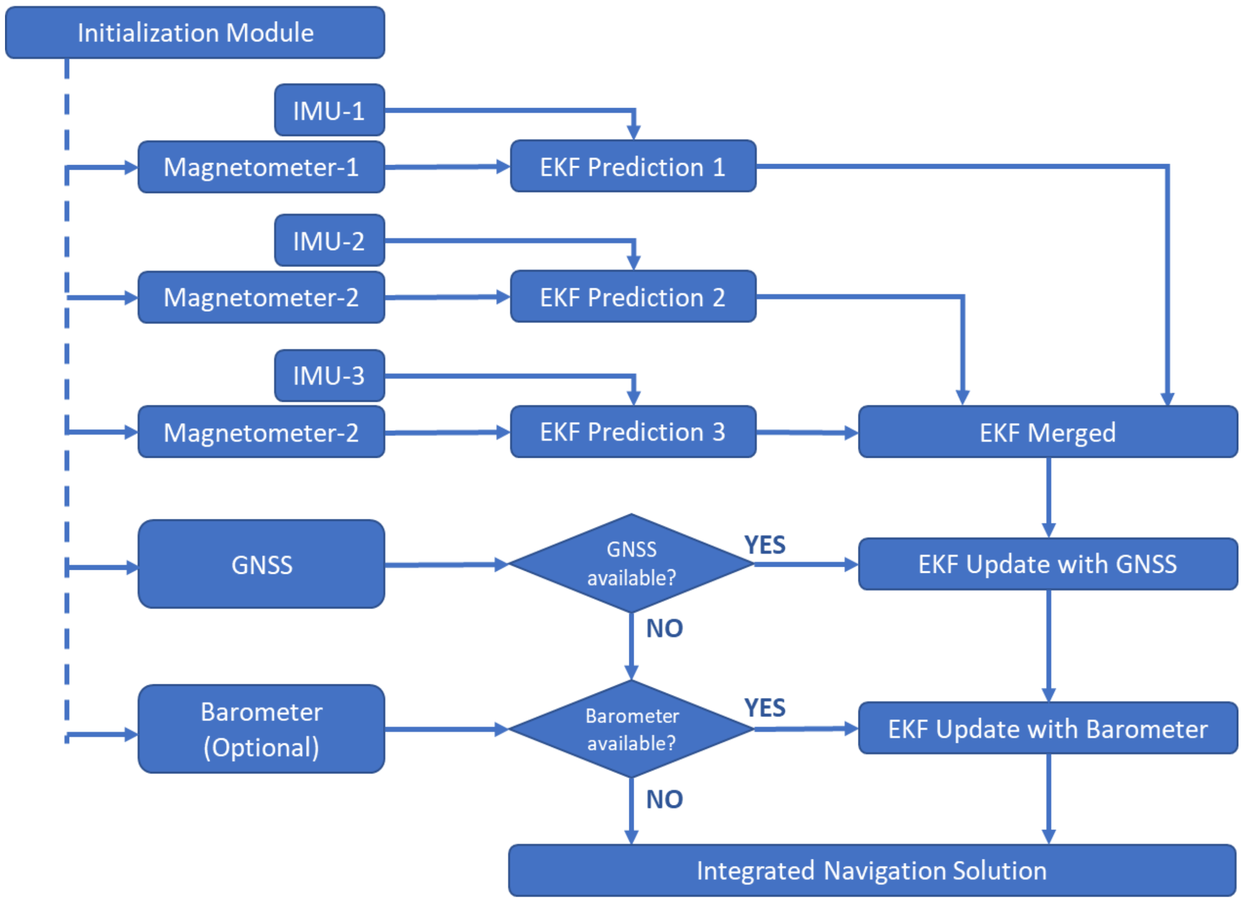


Figure 13. Flowchart of the Integrated Navigation Solution with updated EKF output of data fusion from multiple sensors.

The Estimation and Control Library (ECL) in Pixhawk uses an Extended Kalman Filter (EKF) algorithm to process sensor measurements to give pose estimates. The first three instances are for getting prediction for each of the available Inertial Measurement Unit (IMU) in Pixhawk and the fourth one to merge the results of all the IMU and fifth one to merge GPS position and last Extended Kalman Filter is to merge barometer. It is designed in this way because EKF can still give the prediction even if any one of the input sensors is not available for a period due to any reason. For optimal pose estimation performance, parameter tuning of EK3 is done using the above 15 parameters mentioned in Table 1. These parameters are tuned to affect accuracy of the EK3 and the band for modifying each of them is finalized based on operating range and they are tuned on trial-and-error method to come up with best set of parameters to get acceptable accuracy for this system.

3.6.2. Robot Operating System - ROS

The software architecture used in this project is ROS Melodic. ROS Melodic, released in May 2018, is designed to work under Ubuntu 18.04. The Robot Operating System (ROS) serves as the underlying software architecture for interfacing and communication within the HELIX robot's system. In this thesis, ROS is utilized to communicate between the Pixhawk Cube Black flight controller, the high-level Jetson® computer, sensors and motor control unit.

ROS is used to receive pose estimates from the Pixhawk FMU, run a control loop for waypoint following, and issuing commands to the Helix's onboard motor control board.

3.6.3. MAVLINK / MAVROS

MAVLink®, also known as Micro Air Vehicle Communication Protocol, is a lightweight communication protocol originally designed for unmanned systems. MAVLink® enables the exchange of data between the ground control station (GCS) and the unmanned vehicle (UV). It provides a standardized format for sending and receiving messages related to vehicle state, sensor data, control commands, and more. [65]

For our purposes, we employ MAVROS, a ROS package that allows sending and receiving MAVLink messages. (Refer Figure 14).

By utilizing MAVROS [65], the high-level computer Jetson® is able to receive pose estimates and other relevant data from the Pixhawk Cube Black FMU. This data is used by the control algorithm running onboard the Jetson. Moreover, MAVROS allows sending commands from the Jetson® to the rover's motor control board. Two MAVROS nodes are used, and one is designed by us for sending pose estimates to the Jetson and MAVROS node is provided by Copperstone technologies® for communicating with the motor drivers.

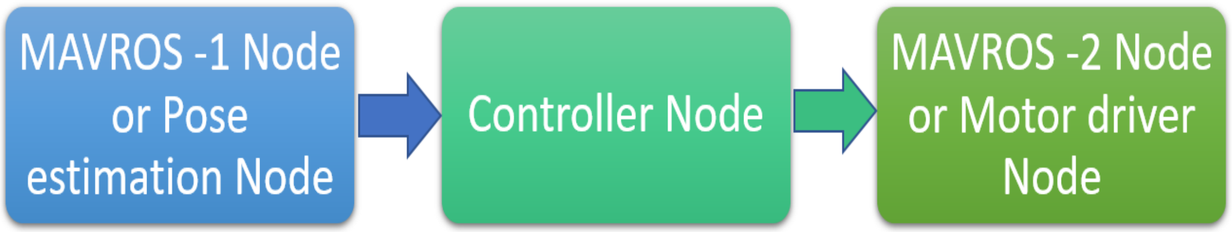

MAVROS -1 Node or Pose estimation Node
Controller Node
MAVROS -2 Node or Motor driver Node

Figure 14. Communication flow between the MAVROS nodes and the Controller.

3.7. High-Level Logic and PID Control for Waypoint Navigation

The goal of the algorithm is to navigate the rover to a series of predefined waypoints by using the current position and heading of the rover and creating a command for onboard motor controller to move the rover towards the target waypoint. The algorithm utilizes a PID (Proportional-Integral-Derivative) controller as the base controller for the rover's motion control.

The main loop of the algorithm loops around constantly checking the distance error between the rover's current position and the goal waypoint. If the distance error is below a certain threshold, the algorithm considers the goal waypoint has reached and proceeds to the next one; Else, the core command-velocity algorithm comes into operation. This core velocity-command algorithm is developed for different modes of rover movement: ROLL mode and SCROLL mode that give twist command based on calculated heading and distance error, OMNI directional mode that also gives twist command with no heading adjustment but only for the x and y axis.

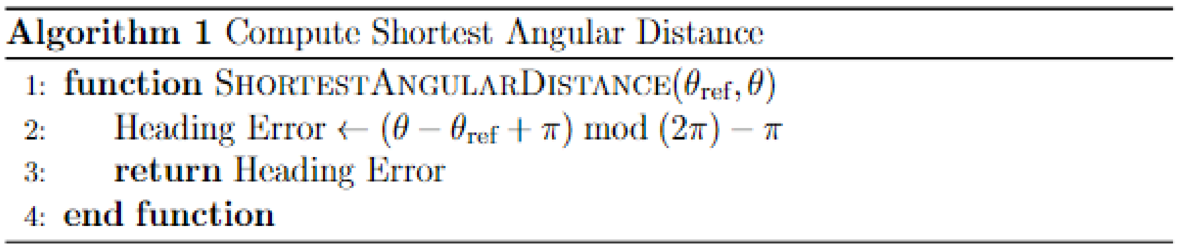
For the ROLL and SCROLL modes, the command-velocity algorithm calculates the heading error by comparing the rover's current orientation with the desired heading towards the goal waypoint as shown in below equations. $(x_{\text{ref}}, y_{\text{ref}})$ and (x, y) are the coordinates of goal position point and current position point respectively. Whereas $\theta_{\text{ref}}\ and\ \theta$ are heading goal angle and current yaw angle of the rover.

$$\theta_{\text{ref}} = \text{atan2}(y_{\text{ref}} - y, x_{\text{ref}} - x) \qquad \ldots (3)$$

$$E_{Yaw} = (\theta_{\text{ref}} \ominus \theta) \qquad \ldots (4)$$

$$\theta_{\text{ref}} \ominus \theta = (\theta - \theta_{\text{ref}} + \pi) mod(2\pi) - \pi \qquad \ldots (5)$$

Heading error is calculated based on shortest angular distance algorithm given below. This calculates the shortest angular distance between the current yaw and the heading goal angle in a unit circle.



Algorithm 1 Compute Shortest Angular Distance

```
1: function ShortestAngularDistance($\theta_{\text{ref}}, \theta$)
2:     Heading Error ← $(\theta - \theta_{\text{ref}} + \pi) \bmod (2\pi) - \pi$
3:     return Heading Error
4: end function
```

Figure 15. Shortest Angular Distance Logic

$$E_{distance} = \sqrt{(x_{\text{ref}} - x)^2 + (y_{\text{ref}} - y)^2} \qquad \ldots (6)$$

Distance error is calculated based on equation. Depending on the magnitude of the heading error, this algorithm determines the appropriate action for each of the modes. If the heading error is more than 0.2 rad, the rover is instructed to turn in place until the error becomes smaller than 0.2 rad. The objective of turn is what differentiates the ROLL mode from SCROLL mode, where the target axis of alignment in each case is rover's X-axis or Y-axis respectively as shown in Figure 17. Once the heading error is within a prescribed threshold of 0.2 rad, the rover then moves forward while correcting the heading error using the PID controller law as given by below equations).

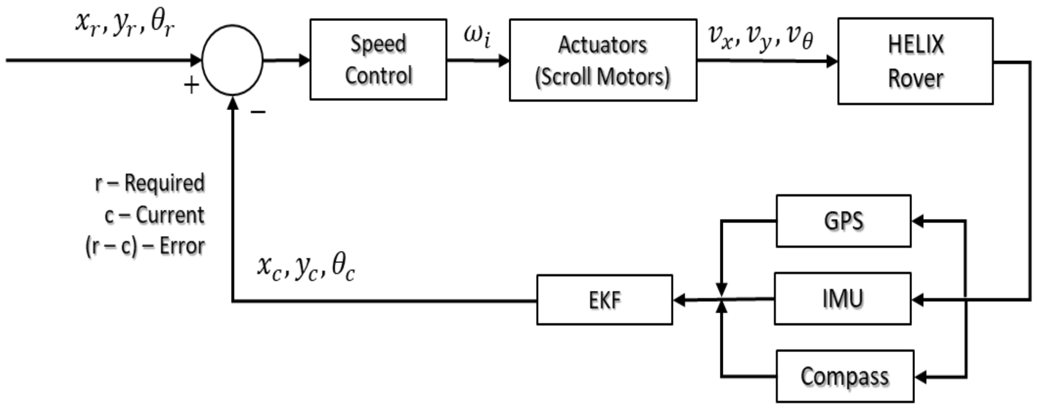


Figure 16. Control Strategy of rover to generate velocity command.

The PID controller at the base motion-control stage considers the proportional, integral, and derivative terms of the error in distance and heading of the rover from its real-time feedback sensors to generate appropriate linear velocity and angular velocities to minimize the real-time error.

$$v = k_p^v * E_{distance} + k_d^v * \frac{[E_{distance} - Prev\ E_{distance}]}{\Delta T} + k_i^v * \int E_{distance}\, \Delta T \qquad \ldots (7)$$

$$\gamma = k_p^\gamma * E_{Yaw} + k_d^\gamma * \frac{[E_{Yaw} - Prev\ E_{Yaw}]}{\Delta T} + k_i^\gamma * \int E_{Yaw}\, \Delta T \qquad \ldots (8)$$

As the control algorithm is linear and based on Corke's algorithm it can go into the limit cycle based on the goal points it's following. Goal points should be selected in such a way that it won't go into cyclic error and end up executing limit cycle. Pre-alignments were taken care to decrease the chances of encountering a limit cycle in the system when control logic is performed. This forward motion along positive X-Axis is referred to as forward movement in ROLL mode, while the similar forward motion along positive Y-Axis is referred to as forward movement in SCROLL mode when the rover approaches the goal waypoint from the current position.

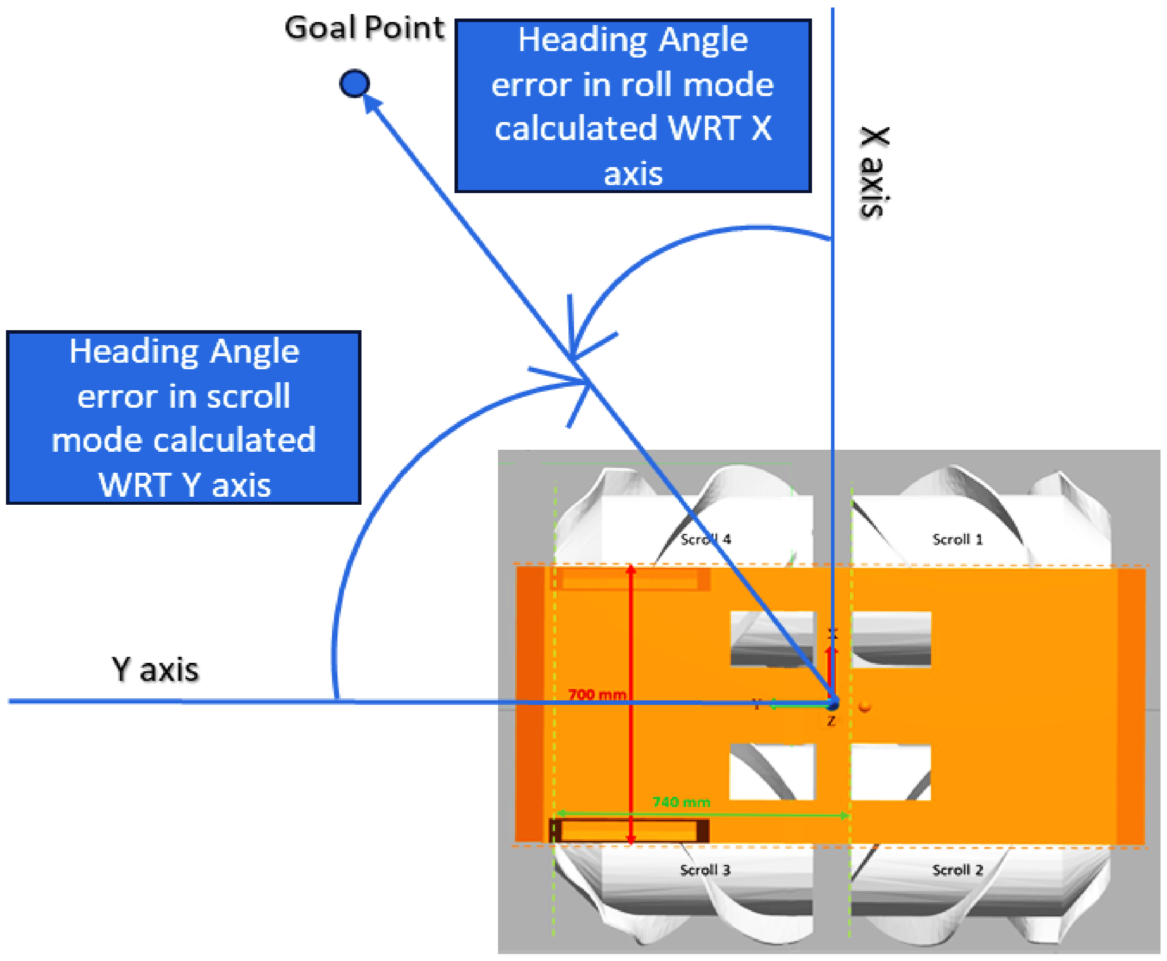

Goal Point
Heading Angle error in roll mode calculated WRT X axis
X axis
Heading Angle error in scroll mode calculated WRT Y axis
Y axis
Scroll 4
Scroll 1
700 mm
740 mm
Scroll 3
Scroll 2

Figure 17. Heading angle error for roll mode and scroll mode, aligning X and Y axes, respectively.

Throughout the execution of the overall algorithm, the rover's individual scroll rotational velocities are calculated based on the global velocity commands to the rover, and the individual motor commands are published to the appropriate topic for controlling the scroll motors as a percentage of maximum RPM; maximum RPM of motors in helix -25 can go up to 195 RPM at no load condition. The algorithm continues to execute until all waypoints have been reached. Once the goal has been reached, the algorithm stops the rover, publishes the status that goal is reached, and saves the trajectory data for history and overview. Finally, the rover is disarmed, and the program ends. Figure 18 given below shows the algorithm used for PID way point navigation in this thesis. This forms the core of control strategy for way point navigation in reaching goals autonomously.

```
Algorithm 2: Helix Navigation Control with PID
Input  : ROS Topics, Constants, Parameters
Output: Controlled Movement of the Robot
Initialize ROS Node;
Initialize Publishers and Subscribers;
Initialize PID Parameters, Waypoints, and Robot Information;
while Goal is not reached do
    Receive Odometry from Pixhawk;
    Calculate distance error and heading goal angle;
    $dx \leftarrow$ goal.x $-$ x;
    $dy \leftarrow$ goal.y $-$ y;
    $distance\_error \leftarrow \sqrt{dx^2 + dy^2}$;
    $heading\_goal \leftarrow atan2(dy, dx)$;
    if Distance error < 1.0 and Goal not reached then
        Stop the robot;
        Publish goal-reached message;
    end
    if Heading error > 0.2 then
        $error_{Yaw} \leftarrow$ desired heading $-$ Yaw;
        $integral \leftarrow integral + error_{Yaw} \times \Delta t$;
        $derivative \leftarrow (error - prev\_error_{Yaw})/\Delta t$;
        $angular\_velocity \leftarrow$
          $K_{pz} \times error_{Yaw} + K_{iz} \times integral + K_{dz} \times derivative$;
        Turn the robot using calculated angular velocity;
        Publish Angular Velocity;
    end
    else
        $integral_{linear} \leftarrow integral_{linear} + distance\_error \times \Delta t$;
        $derivative_{linear} \leftarrow (distance\_error - prev\_error_{linear})/\Delta t$;
        $linear\_velocity \leftarrow K_{px} \times distance\_error + K_{ix} \times integral_{linear} +$
          $K_{dx} \times derivative_{linear}$;
        Move the robot forward while correcting heading;
        Publish Linear & Angular Velocities;
    end
    if All Points Covered then
        Exit loop and stop the robot;
    end
end
```

Figure 18. Helix PID Control Algorithm

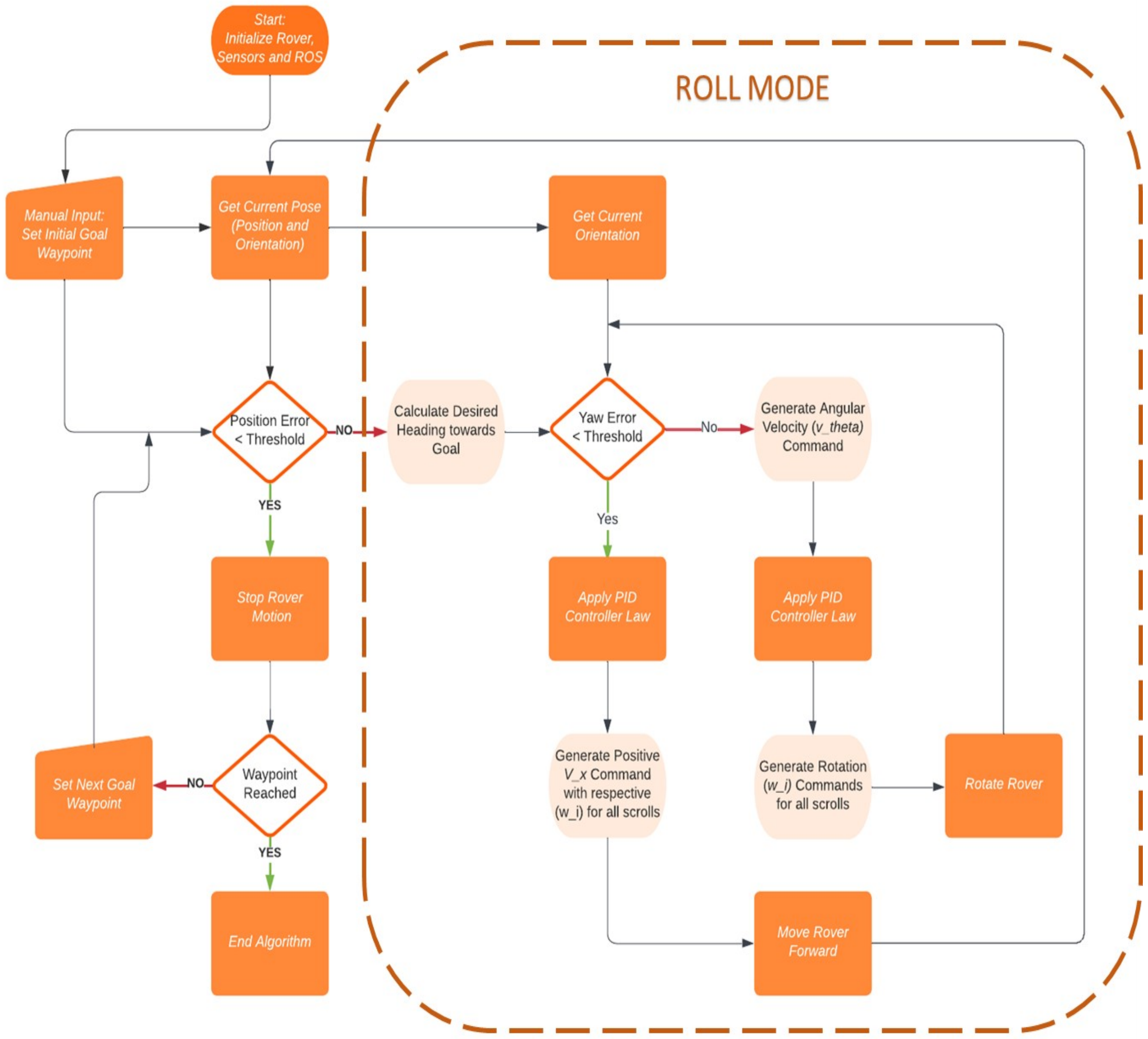

Start: Initialize Rover, Sensors and ROS
ROLL MODE
Manual Input: Set Initial Goal Waypoint
Get Current Pose (Position and Orientation)
Get Current Orientation
Position Error < Threshold
NO
YES
Calculate Desired Heading towards Goal
Yaw Error < Threshold
No
Yes
Generate Angular Velocity (v_theta) Command
Stop Rover Motion
Apply PID Controller Law
Apply PID Controller Law
Set Next Goal Waypoint
NO
Waypoint Reached
YES
Generate Positive V_x Command with respective (w_i) for all scrolls
Generate Rotation (w_i) Commands for all scrolls
Rotate Rover
End Algorithm
Move Rover Forward

Figure 19. ROLL-Mode Algorithm Flow: Align the front heading towards goal and move forward (X).

This waypoint algorithm provides a fundamental framework for autonomous navigation of the rover using PID controller for motion commands based on pose estimates. It enables the rover to follow a predefined path efficiently and accurately with tolerable deviations.

3.8. Experimental Set-up

3.8.1. Lab Set-up – Helix on Block

The storage location for the Helix rover is typically on the 4th floor of the Mechanical Building at the University of Alberta. Given the dimensions of the rover, approximately 4 feet by 4 feet, and its weight of approximately 250 pounds, a minimum team of five individuals is necessary to facilitate its transportation from the 4th floor to the ground floor using the building's elevator. These logistical constraints have led to the majority of the rover's functional testing being conducted within the confines of the laboratory facility, where the rover is securely mounted on a support block. All testing needs to be done in outdoor settings to obtain a GPS signal for the pose estimation system. This is particularly relevant due to the laboratory's location, surrounded by tall buildings adjacent to the Mechanical Engineering Block within the University of Alberta campus.

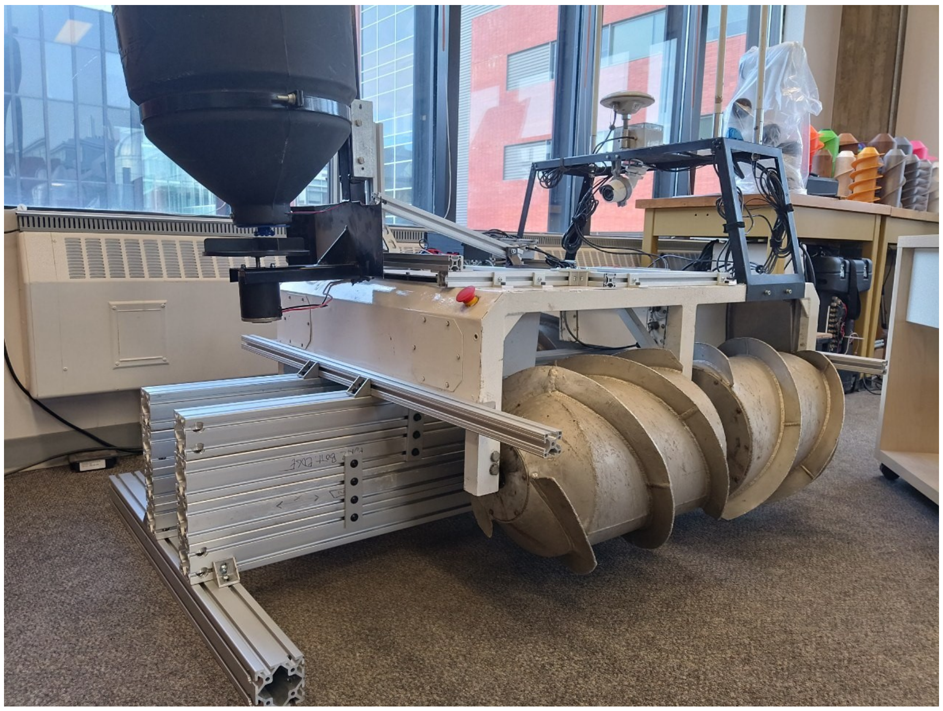


Figure 20. Fully Assembled Helix Rover on Block: The Helix rover, equipped with a comprehensive array of electronic components and hardware interfaces, stands ready for testing and experimentation in the controlled laboratory environment.

The proper working of all the individual scroll modules and electronics modules were tested mounting the helix on block here before moving on to the field. Figure 21 shows the setup of the system for field testing.

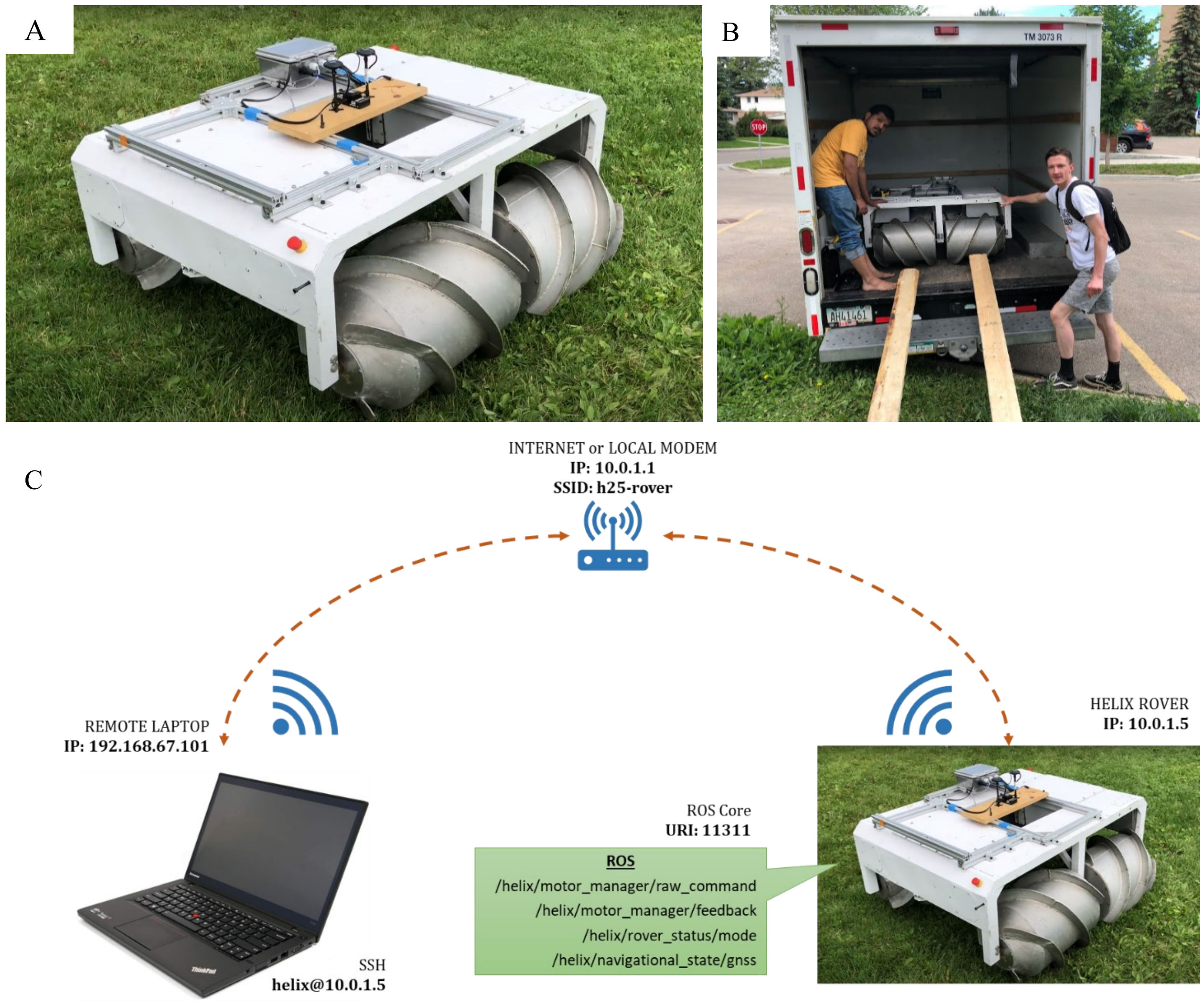


Figure 21. The setup of the system for field test includes – A) Fully assembled and control interface-integrated Helix rover, B) Safe loading and unloading of rover from a truck at the desired location by at least two persons, and C) The internal communication architecture for interacting with the edge-computer on rover from a remote laptop carried by the user via internet or intranet modem.

Chapter 4 : Simulation of Rover Motion Algorithms in ROS

The algorithms programmed for the developed control logic need to be examined closely to ensure correctness and comprehensive flow of instructions. Simulations in robotics are, therefore, deemed as essential mode of such evaluation and verification of the algorithms, wherein the system kinematics are also modelled for direct implementation tests as required. This chapter will focus on the kinematics of robot movement in the simulation environment, the implementation of algorithms and the testing for various trajectories. Further, the software architecture used for simulation is replicated to be same as the one built over the hardware as discussed in the previous chapter, except for the sensors and actuators that are transformed into imitation modules. Since the software architecture is built on ROS framework, the simulation tool used is a built-in simulator called Gazebo™.

4.1. ZM Mecanum Robot and its Kinematics

To begin with the development of algorithms for a holonomic vehicle, we employ an open-source model of ZM Mecanum Robot. This mobile robot is made of four independently driven mecanum wheels, which enable the robot to practice omni-directional movement [66] (Refer to Figure 22). As the locomotion behavior of the HELIX rover is conditionally holonomic (as explained in Chapter 3 :), paring wheels structure and the kinematics are similar to that of ZM's omni-directional movement and thereby, the simulation is first run on ZM model for initial verification.

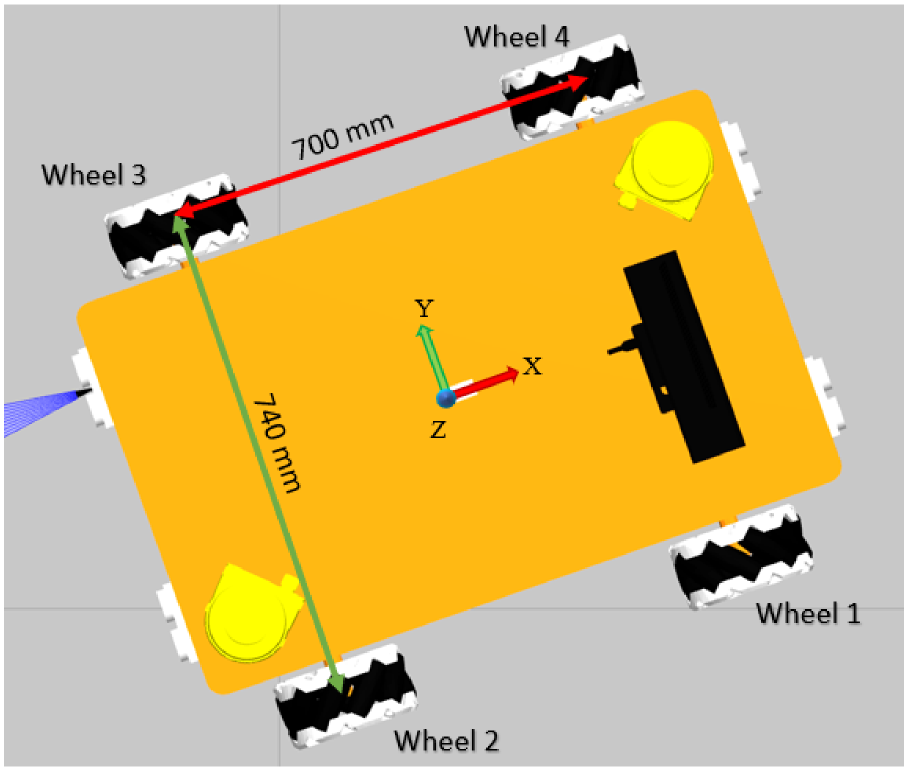


Figure 22. 3D Simulation model of four-Mecanum-wheeled robot platform

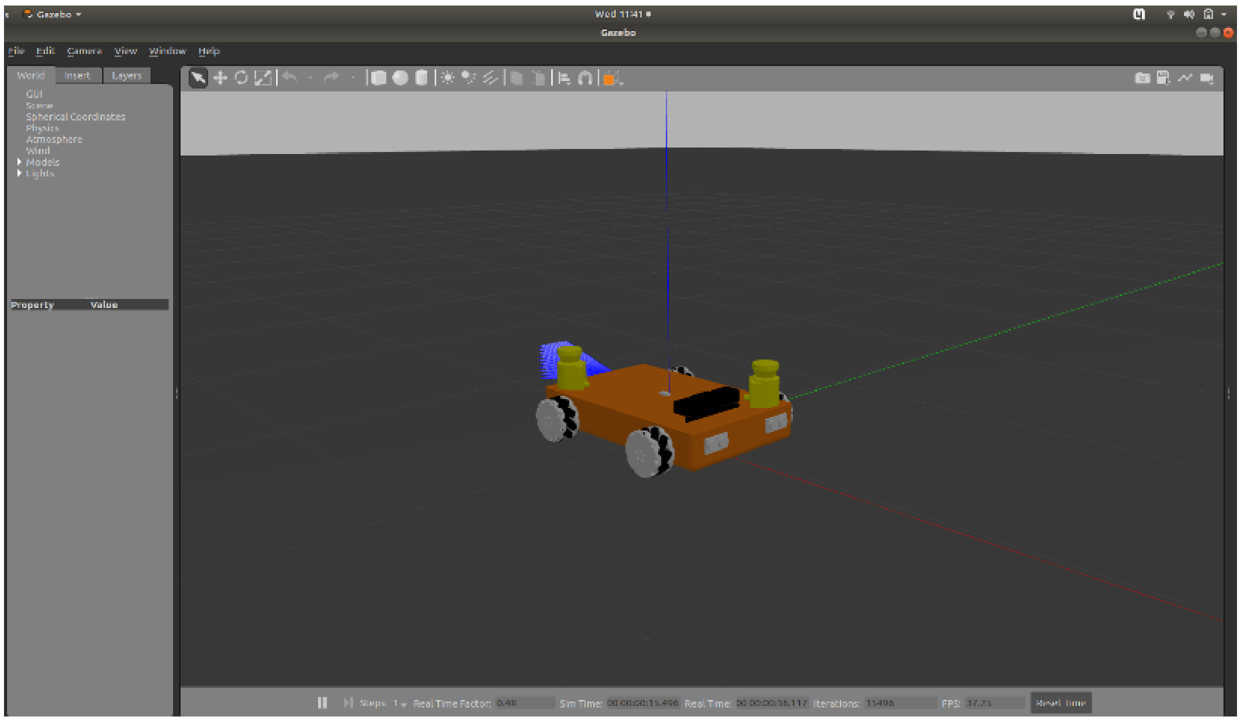


Figure 23. The 3D model of ZM Mecanum robot, spawned at origin in Gazebo™ simulation environment launched in ROS.

The decision to employ the ZM robot with mecanum wheels for the autonomous waypoint PID control algorithm pilot test within the Gazebo simulation framework was driven by the need to verify the implementation of the control algorithm before hardware testing. While the ZM robot and the Helix rover differ in their physical attributes, both share similar mechanics for holonomic movement, although it is only conditional for Helix rover to exhibit holonomic movement. This shared characteristic enables the ZM robot model to provide a deeper understanding of simulation framework to repurpose the software package for Helix rover as well for its own inertial parameters and dimensions which would affect its performance.

Although not a direct replica of the Helix, the ZM robot's mecanum wheel design and omnidirectional functionality effectively facilitate the algorithm's validation, ensuring its correctness and functionality for varied trajectories prior to application on the 3D model of Helix rover. It also expedites the algorithm's development, allowing for parameter tuning, scenario exploration, and behavior analysis without the logistical complexities associated with real-world testing.

4.1.1. Simulation Environment Set-up

The Gazebo simulator uses the URDF (Unified Robot Description Format) file to load the 3D model of the ZM robot, capturing its visual and inertial properties. It also launches required ROS controllers, the so-called fake controllers to enable interaction with the simulated robot. These controllers translate algorithmgenerated commands into actions within Gazebo simulation environment for thorough control algorithm assessment and enhancement.

The Forward Command Topic ('/cmd_vel') serves as the mode for transmitting motion commands from the control algorithm to the simulated robot. The real-time feedback is carried by Odometry Feedback ('/odom'), that gives detailed position and orientation data, that is used for calculating distance and heading errors with respect to desired values for dynamic trajectory tracking. We developed a controller module in Python that subscribes to odometry (/odom) and publishes commands to the robot (/cmd_vel). It must be noted that the odometry information obtained from simulation is always perfect as Gazebo™ simulator version used here is limited to kinematics only with no force dynamics included. That makes the actual process pipeline oblivious to errors in hardware trials, where the data quality depends on the sensors used. Therefore, in the field tests, the accuracy of the pose estimates by Pixhawk will have significant influence on controller's performance and thereby determines the tolerance in execution.

4.1.2. Experimental Methodology

The primary objective of this study is to comprehensively assess the efficacy of the three control algorithms: ROLL, SCROLL, and OMNI directional modes. To thoroughly examine the versatility and performance of these methods, a deliberate choice of diverse trajectory patterns was made. These trajectories encompass a 10m x 10m SQUARE, a HEXAGON with each side measuring 2m, and a 2m-radius CIRCLE centered at (2, 2). This trajectory selection aims to evaluate the algorithms across a range of transition movements and shapes, providing insights into their adaptability and robustness to also determine the feasible waypoint distance for ZM as well as Helix rover. To ensure accurate comparisons, the maximum velocity was constrained to 0.6 m/s. During simulation, we log all the signals, including odometry and control inputs for offline plotting, that aids visualizing and troubleshooting.

4.2. Motion Planning Algorithms Testing – Plots and Results

In this section, we delve into the key observations derived from the data and corresponding visual representations, as depicted in the accompanying plots and figures. We study the robot's trajectory tracking, looking for any deviations and variations against the expected paths. Thus, we examine the error in the simulation, that could be attributed to issues with the logic and/or its program to be corrected accordingly. We also delve into detailed analysis of the algorithm's performance, to interpret its strengths and limitations.

4.2.1. Trajectory 1: Square

We use a 10mx10m square as the reference trajectory. In Figure 24, we outline the critical dimensions and parameters of our square trajectory, a fundamental testing scenario for our autonomous navigation algorithms.

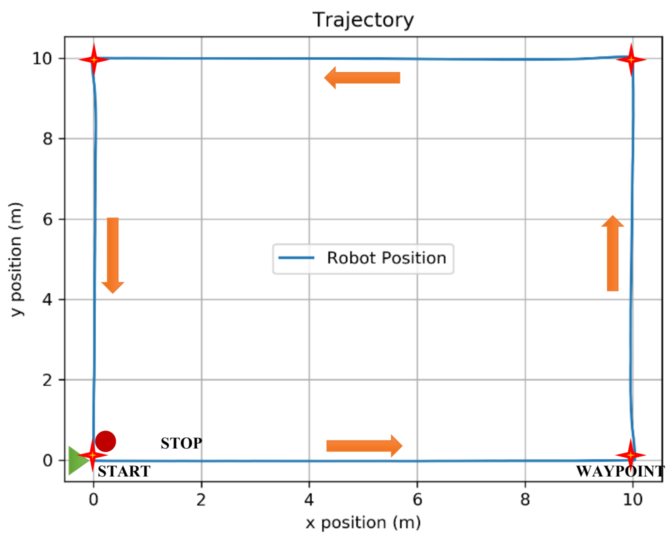
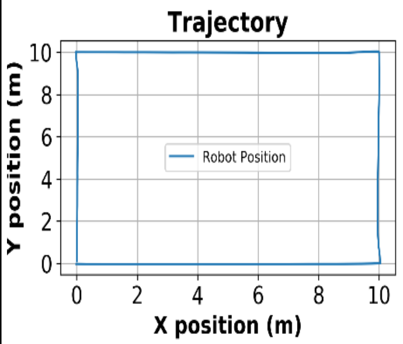
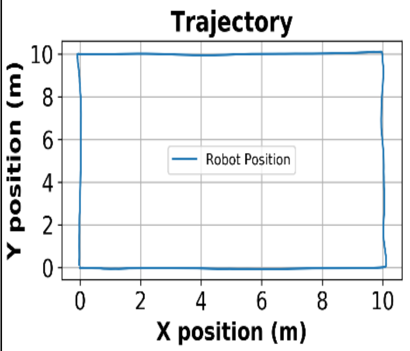
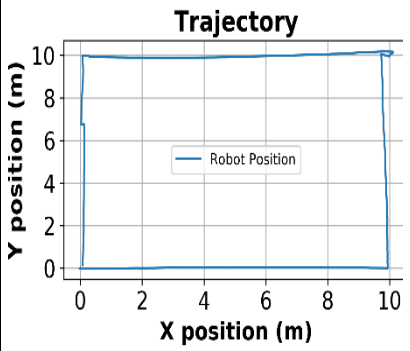
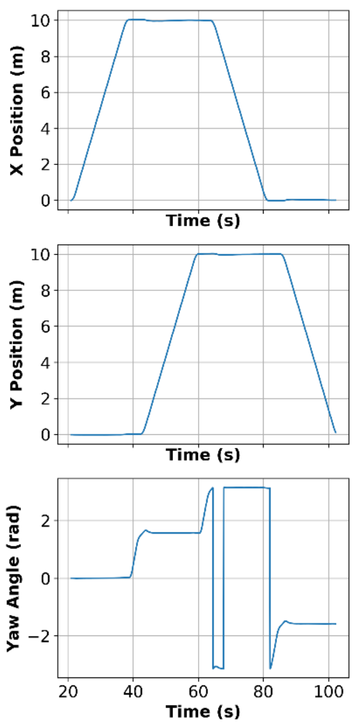


Figure 24. The definitive details of square trajectory movement: the start point, the stop point, the position of four waypoints, and the direction of motion.

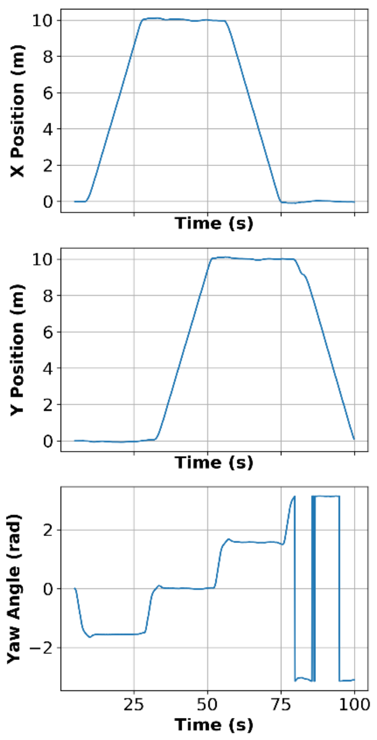
This geometry simplifies the analysis of the robot's performance and error accumulation during traversal, with four key waypoints where the robot changes its heading. The robot moves in a counterclockwise direction. Table 2 presents a comprehensive overview of our simulation results for the execution of three distinct algorithms - Roll, Scroll, and Omni-directional modes - when applied to a square trajectory movement scenario.

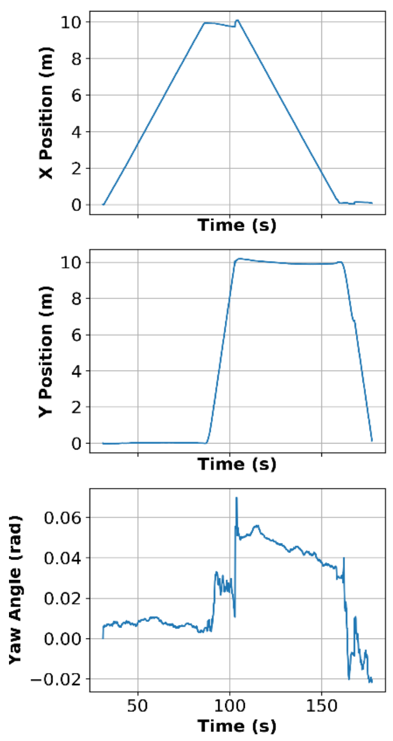
Table 2. Simulation results for execution of three algorithms for (i) roll, (ii) scroll and (iii) omni-directional modes, respectively for square trajectory movement, which includes: (a) resultant trajectory followed, (b) position variation with time, (c) velocity variation with time, (d) error in position and angle with time, (e) wheel power (proportional to wheel rpm).

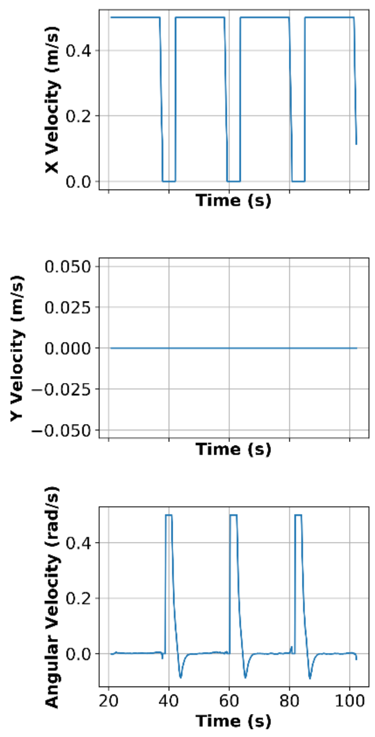
Algorithm-1 ROLL Mode	**Algorithm-2** SCROLL Mode	**Algorithm-3** OMNI-DIRECTIONAL
 TRAJECTORY		



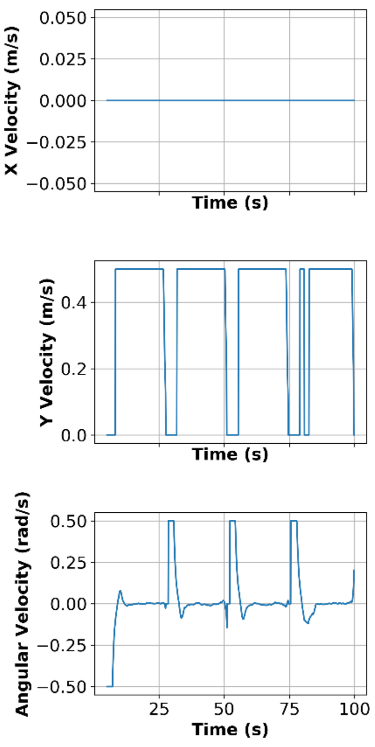
POSITION (X, Y, Theta)

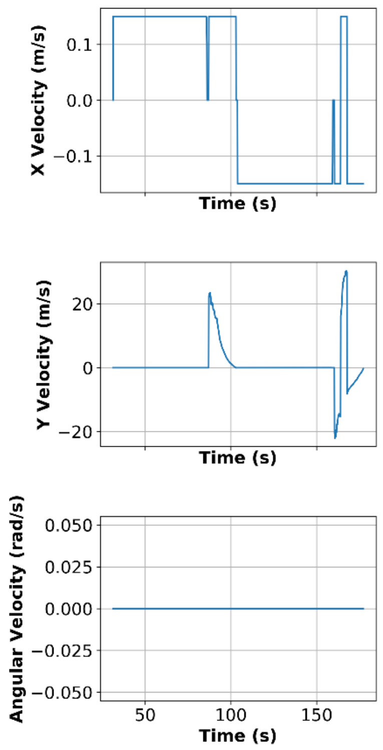


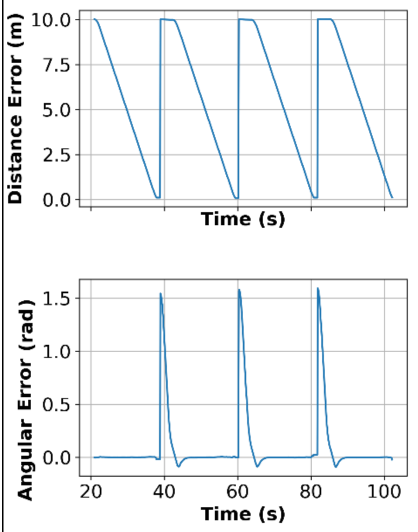
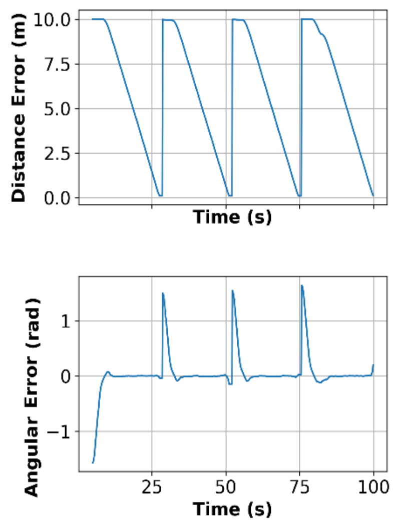
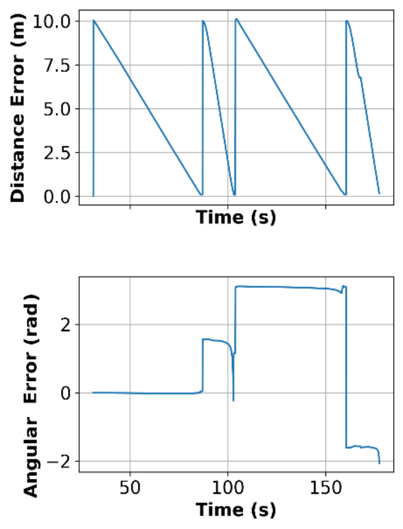
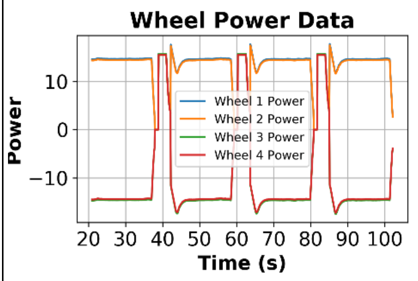
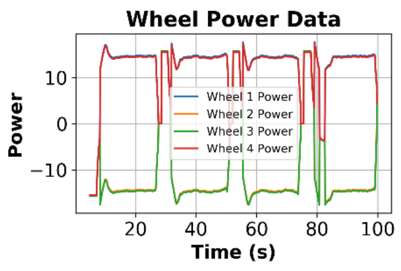
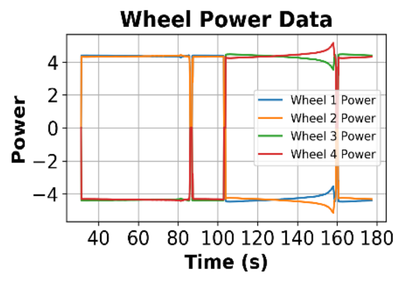




VELOCITY (X, Y, Theta)





 ERROR (X, Y, Theta)		
 WHEEL POWER / RPM		

(a-i, a-ii, a-iii): Resultant Trajectory Followed –

This set of figures visually represents the paths followed by the robot when executing the Roll, Scroll, and Omni-directional algorithms. In the case of the Roll algorithm (i) and the Scroll algorithm(ii), the robot's trajectory produces a near-perfect square trajectory, indicating superior precision at the corners that results in smoother turns with fewer deviations. Indeed, the roll and scroll modes are identical to each other except for the 90-degree rotation difference. Conversely, the Omni-directional mode (iii) exhibits relatively sharp corners but with significant adjustments and noticeable deviations from the ideal square shape.

(b-i, b-ii, b-iii): Position Variation with Time –

This set of figures illustrates how the robot's position evolves over time during each algorithm's execution. For the Roll algorithm (i) and the Scroll algorithm (ii), the rover shows stable position maintenance with respect to X and Y axes that display minimal position deviations, indicating precise control, and a deterministic show of yaw angle adjustments at each waypoint where the heading is adjusted.

The Omni-directional mode (iii) stands out with significant oscillations in yaw angle that indicates continuous heading adjustments throughout the trajectory, and it also shows deviations in X and Y axes that attribute to the noticeable deviations observed in resultant trajectory is visible from the position plot variation with time.

(c-i, c-ii, c-iii): Velocity Variation with Time –

This set of figures provides insights into how the robot's velocity changes throughout the simulation. In the Roll algorithm (i), we observe synchronous pulses when the linear X motion and rotational motion are complementarily active, and zero velocity along Y-axis as expected. The Scroll algorithm (ii) exhibits similar velocity transitions as for the Roll algorithm, leading to graceful motion with Y-axis heading for the entire trajectory and zero velocity along X-axis. Omni-directional mode (iii) shows frequent and abrupt changes in velocity, resulting in jerky movements along both the X and Y axes, ensuring no adjustment in its heading with zero rotational velocity throughout.

(d-i, d-ii, d-iii): Error in Position and Angle with Time –

This set of figures tracks the errors in both position and orientation (angle) of the robot over time. In the Roll algorithm (i) and the Scroll algorithm (ii), position errors are identical, and angular errors are present only during sharp turns. The Omni-directional mode (iii) shows completely alike behavior for variations in both X and Y axes, with the peak variation only at the waypoints.

(e-i, e-ii, e-iii): Wheel Power (Proportional to Wheel RPM) –

This set of figures displays the power consumed by each wheel, which is proportional to its rotational speed, theoretically for a given torque consumption. All the modes clearly demonstrate the pairing of the scrolls for each movement and their synchronized power distribution at equal magnitude is very visible from the wheel power plots.

4.2.2. Trajectory 2: Hexagon

We now use a 2-meter sided hexagon as the second reference trajectory. This trajectory brings in more waypoints at closer distance to each other and a shorter change in heading angle at each waypoint, compared to previous trajectory. This helps us understand how the control algorithm behaves if the goal points are closer to each other when navigating from one to another goal. In Figure 25, we outline the critical dimensions and parameters of our hexagon trajectory, along with the positions of six waypoints at its vertices. The rover again moves in a counterclockwise direction to navigate through all the six points of the hexagon.

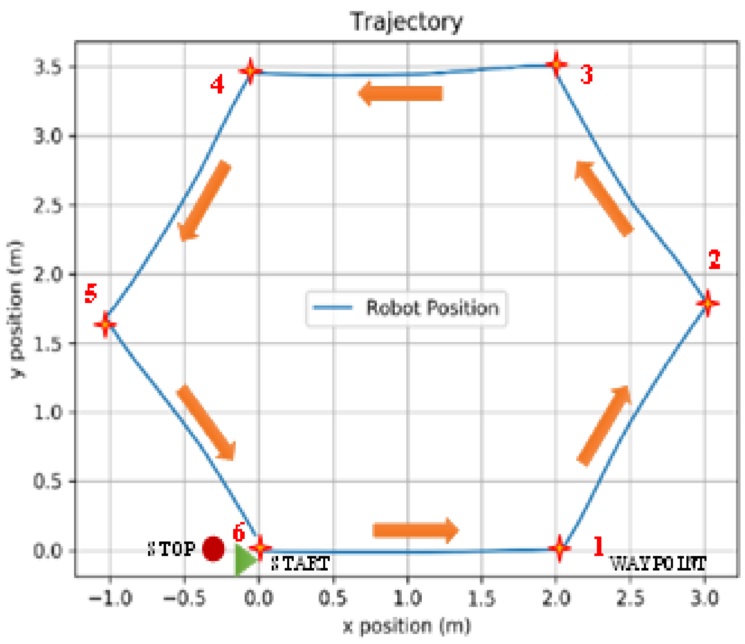
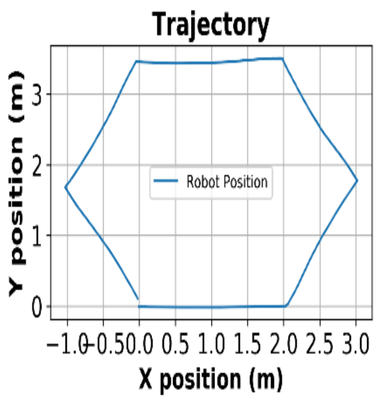
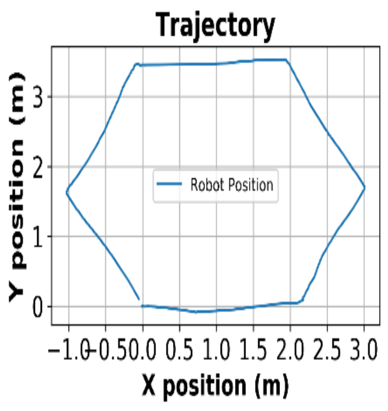
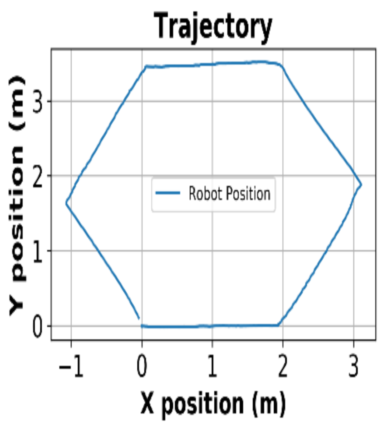
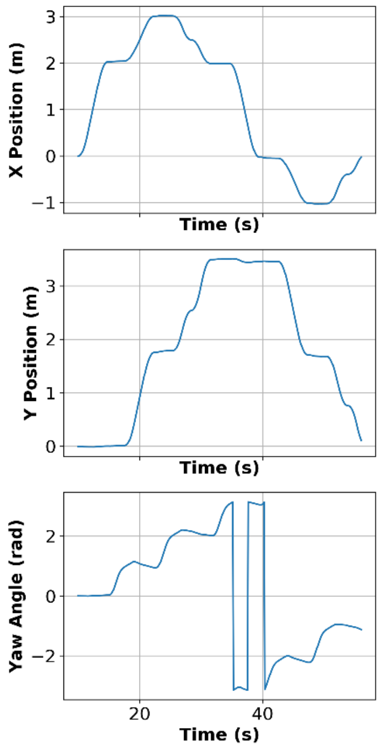


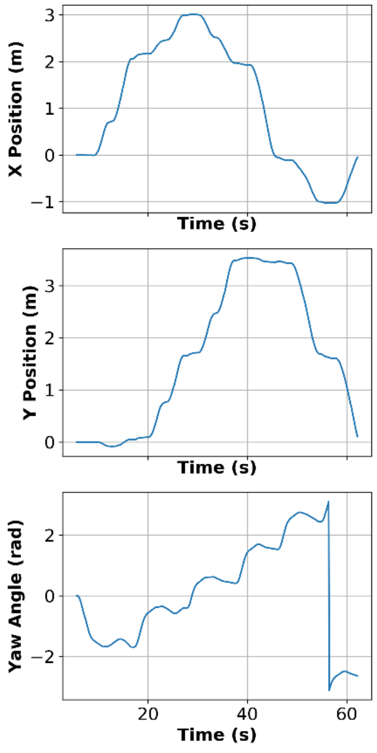
Figure 25. The definitive details of hexagon trajectory movement: the start point, the stop point, the position of four waypoints, and the direction of motion.

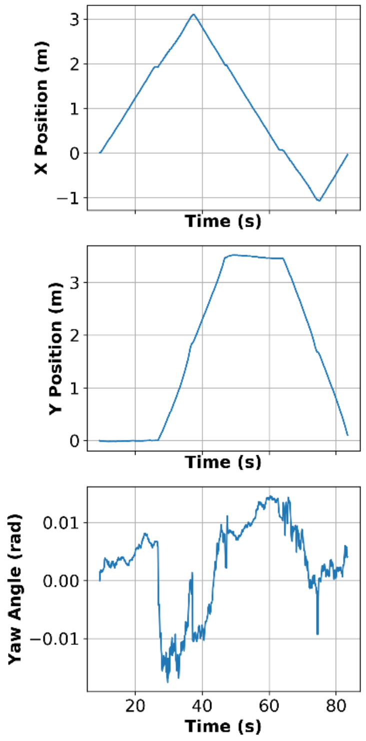
Table 3. Simulation results for execution of three algorithms for (i) roll, (ii) scroll and (iii) omni-directional modes, respectively for hexagon trajectory movement, which includes: (a) resultant trajectory followed, (b) position variation with time, (c) velocity variation with time, (d) error in position and angle with time, (e) wheel power (proportional to wheel rpm).

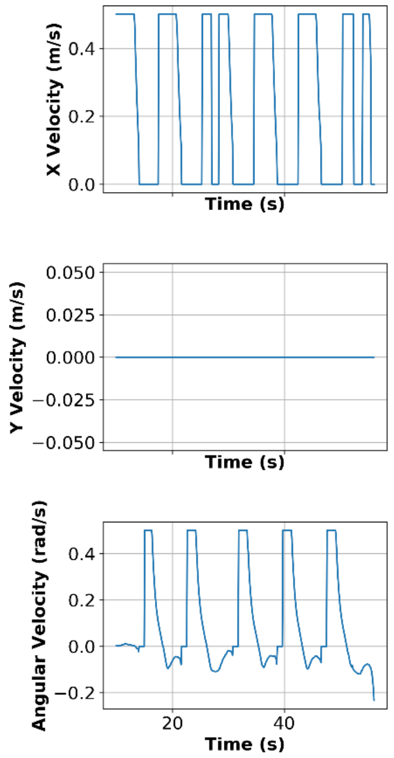
Algorithm-1 ROLL Mode	**Algorithm-2** SCROLL Mode	**Algorithm-3** OMNI-DIRECTIONAL
 TRAJECTORY		



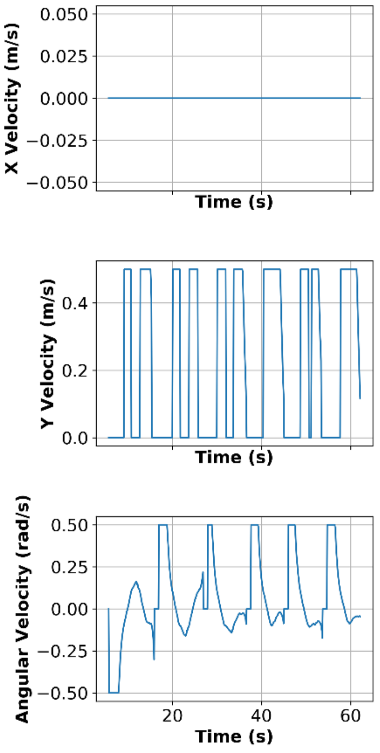
POSITION (X, Y, Theta)

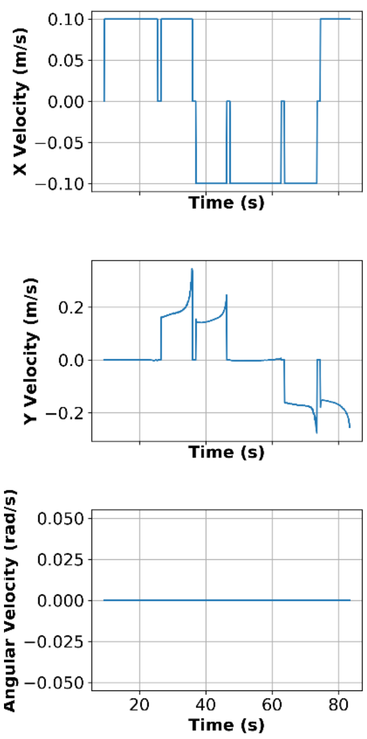


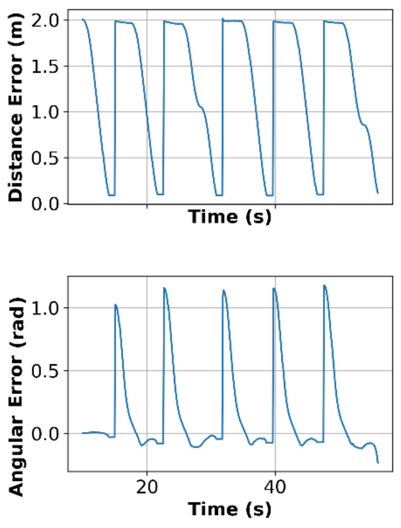
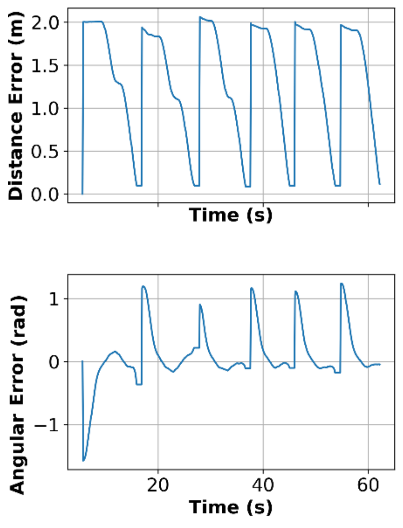
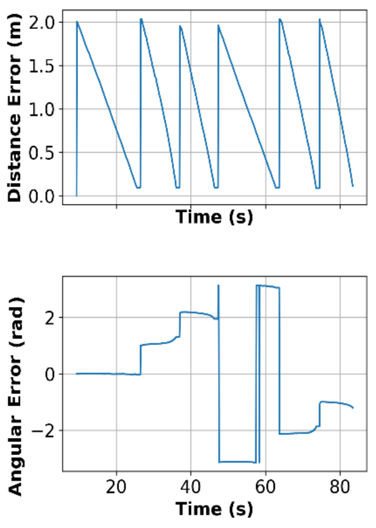
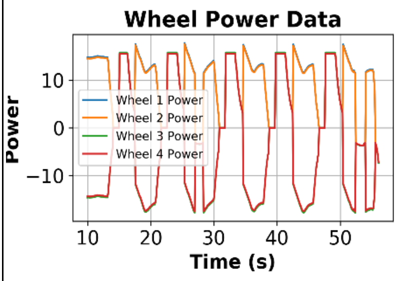
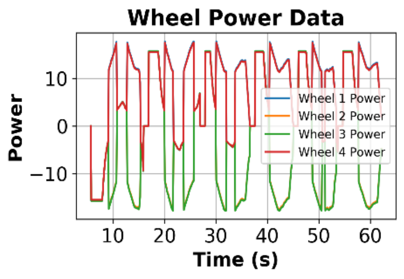
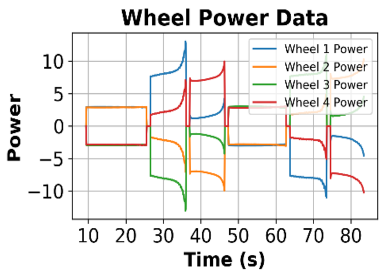




VELOCITY (X, Y, Theta)





 ERROR (X, Y, Theta)		
 WHEEL POWER / RPM		

The figures in Table 3 above provide a comprehensive analysis of the three control algorithms (Roll, Scroll, Omni-Directional) applied to the Hexagon trajectory.

The Roll and Scroll modes present precise trajectory tracking with minor heading adjustments along the edges, while the Omni-Directional mode showcases a sharper trajectory with minimal deviations at waypoint 2 and waypoint 5, where the offset is distinguishable.

In comparison to square trajectory omni-directional mode (iii) shows more frequent changes in velocity as the waypoints are closer, resulting in jerky movements along both the X and Y axes, while ensuring no adjustment in its heading with zero rotational velocity throughout. All the three modes show similar position errors in X and Y directions at each waypoint, except that the Roll/Scroll modes demonstrate smoother transitions through sharp turns than the other. However, in the angular deviations, the Omni-Directional mode shows incremental behavior unlike the Roll mode or Scroll mode, as expected from its kinematics.

4.2.3. Trajectory 3: Circle

The Circle trajectory shown in Figure 26, consists of a 2-meter radius circle traversed in counterclockwise direction. The starting point of the Circle trajectory is at coordinates (4m, 2m), while its center is at coordinates (2m, 2m) in the workspace. This trajectory consists of 25 waypoints to make it smooth and continuous, which expects rover to adjust its position and heading more frequently, while navigating in a confined environment. Table 4 presents simulation results for the execution of three algorithms across Roll, Scroll, and Omni-Directional modes, focusing on the Circle trajectory.

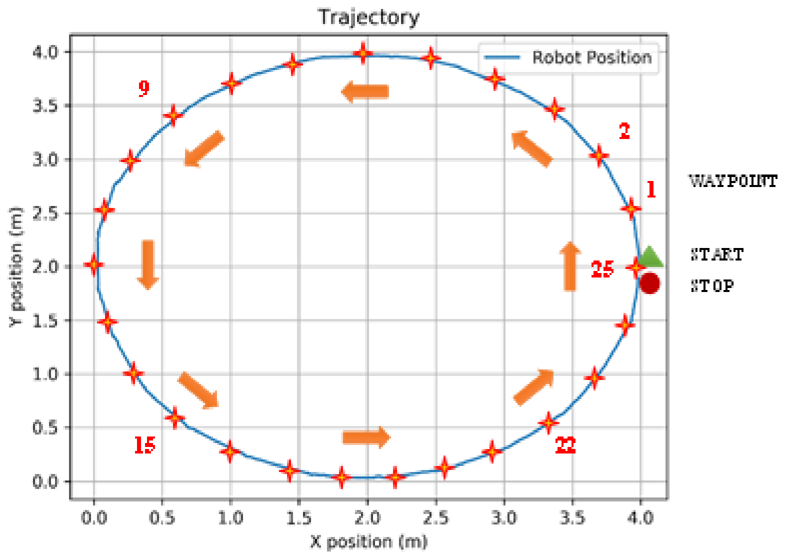
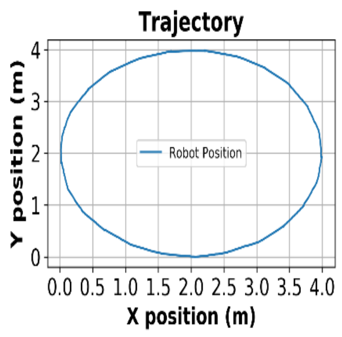
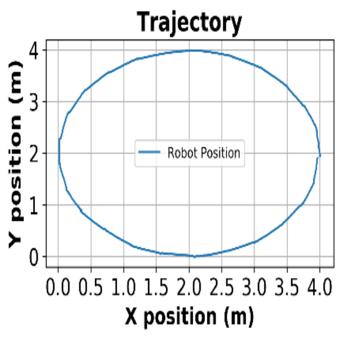
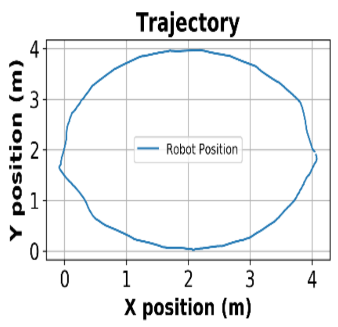
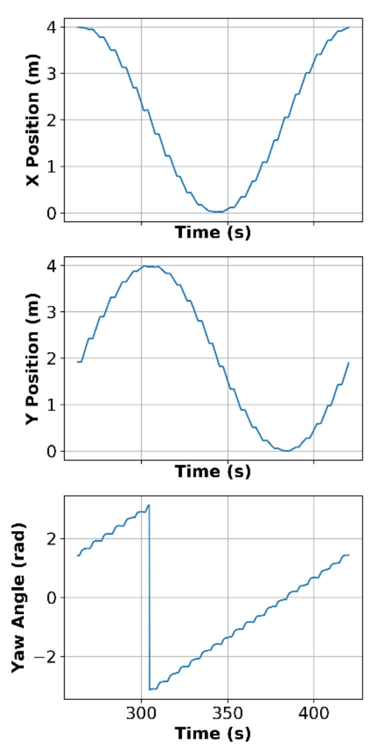
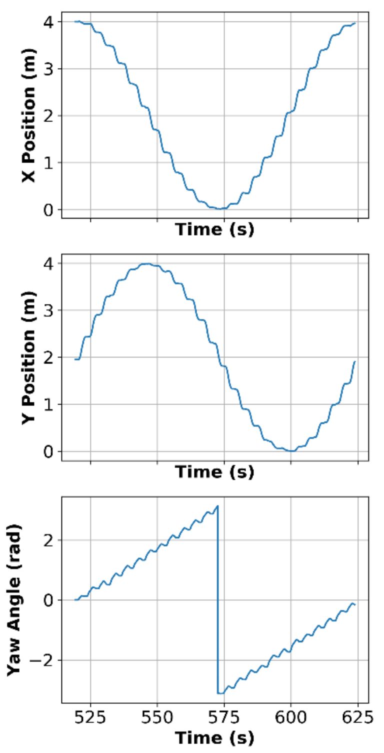


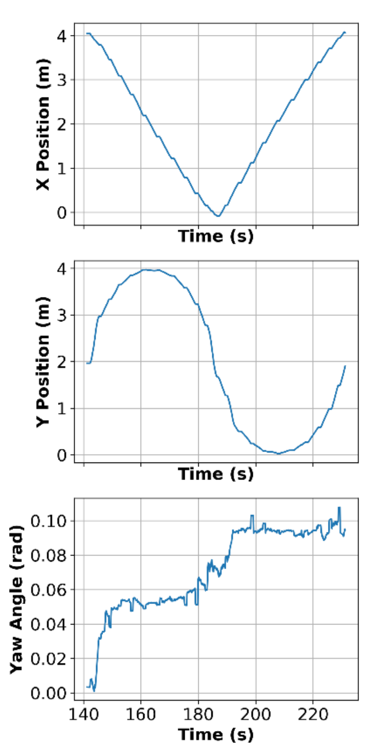
Figure 26. The definitive details of circle trajectory movement: the start point, the stop point, the position of four waypoints, and the direction of motion.

Table 4. Simulation results for execution of three algorithms for (i) roll, (ii) scroll and (iii) omni-directional modes, respectively for circle trajectory movement, which includes: (a) resultant trajectory followed, (b) position variation with time, (c) velocity variation with time, (d) error in position and angle with time, (e) wheel power (proportional to wheel rpm).

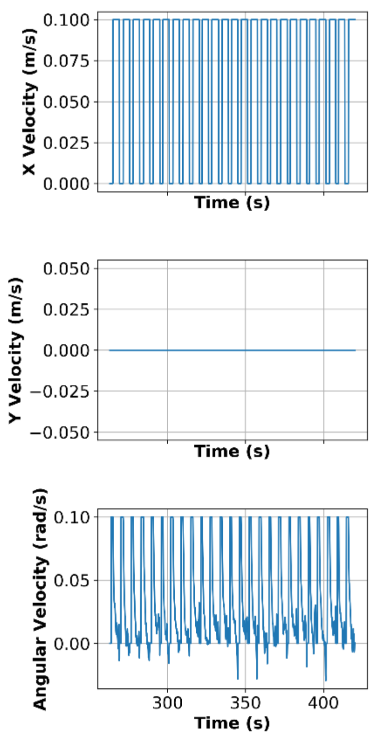
Algorithm-1 ROLL Mode	**Algorithm-2** SCROLL Mode	**Algorithm-3** OMNI-DIRECTIONAL
 TRAJECTORY		

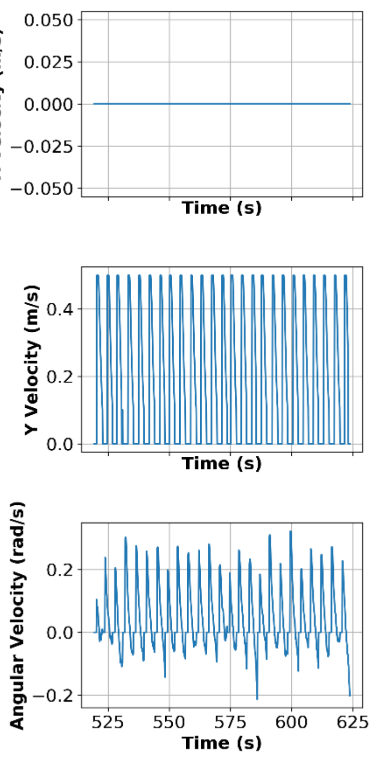


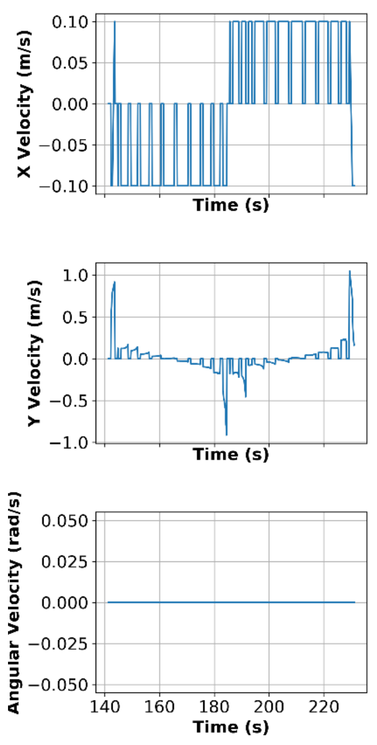




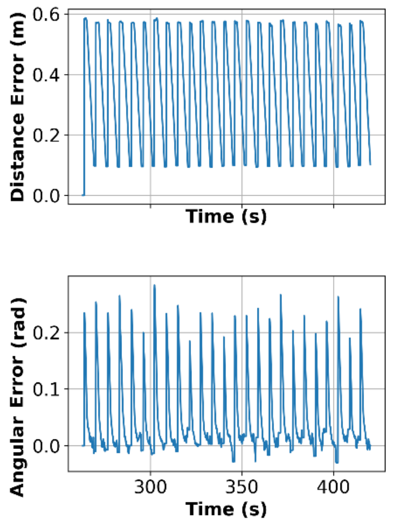
POSITION (X, Y, Theta)



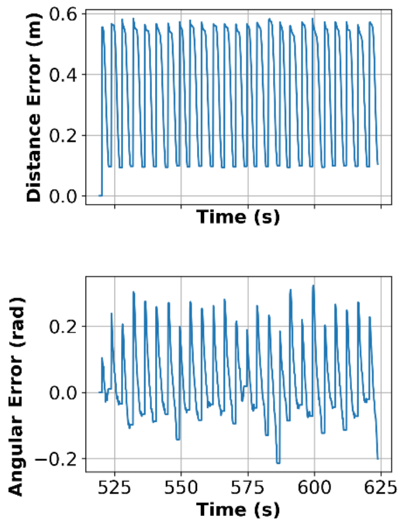


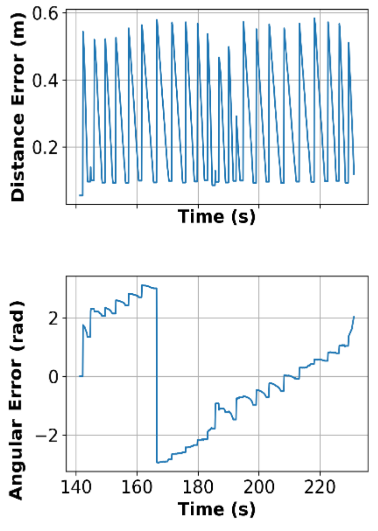


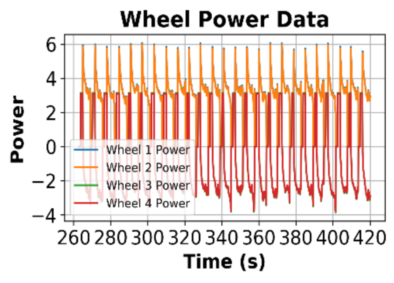
VELOCITY (X, Y, Theta)



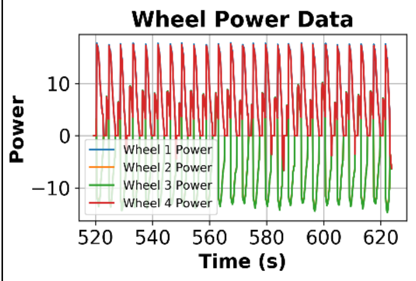
ERROR (X, Y, Theta)

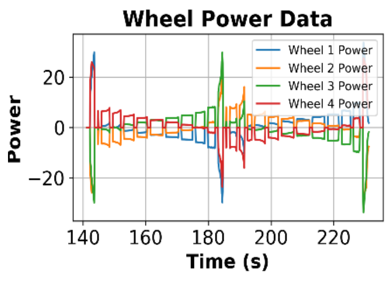






WHEEL POWER / RPM





The Omni-Directional mode exhibits occasional path adjustments after finishing each semi-circle. Both the modes display negligible velocity variations about the heading angle, signifying the reorienting behavior at every waypoint. Omni-Directional mode showcases smooth and precise velocity control. It also shows gradually incremental and significant yaw error. The synchronization of wheel movements for each movement is more prominent in circular trajectory plot and demonstrates the adaptability of the algorithm for close waypoints as well. For all the three trajectories, the only source of error in tracking trajectory is from the PID tuning in such simulation environment where every other sensor/feedback source is ideal.

4.3. HELIX Screw-Propelled Rover – Model and its Kinematics

4.3.1. Digital 3D Model and Robot Description

In Gazebo simulations, the URDF and model's inertial parameters are crucial for an accurate kinematics simulation. The inertial parameters include the mass, center of mass location, and the moment of inertia matrix of all links. An accurate simulation requires physically plausible inertial parameters, which directed us to develop the standard robot description for Helix rover as well. The moments of inertia in URDF are expressed as the components of a symmetric positive-definite 3x3 matrix, with 3 diagonal elements, and 3 unique off-diagonal elements. For custom structures like Helix rover, the best approach to obtain these parameters is from its 3D digital model, created as shown in Figure 27.

The design and modeling of the Helix rover are facilitated through an application called SolidWorks®, a computer-aided modeling software. The rover's dimensions are based on physical measurements, enabling its seamless integration into simulations with direct export of parameters in URDF format to Gazebo™. The Helix rover's design is represented in the. sldasm format. This combination of SolidWorks® and Gazebo™ enables acceptable kinematic simulations to understand the Helix rover’s performance requirements, owing to its size, structure, and mechanics. This conversion process is accomplished using a specialized URDF converter designed to transform the. sldasm file into a .urdf file. During this conversion, particular attention is given to specifying the control and limiting variables that are integral to the robot's behavior and movement, as shown in Table 5.

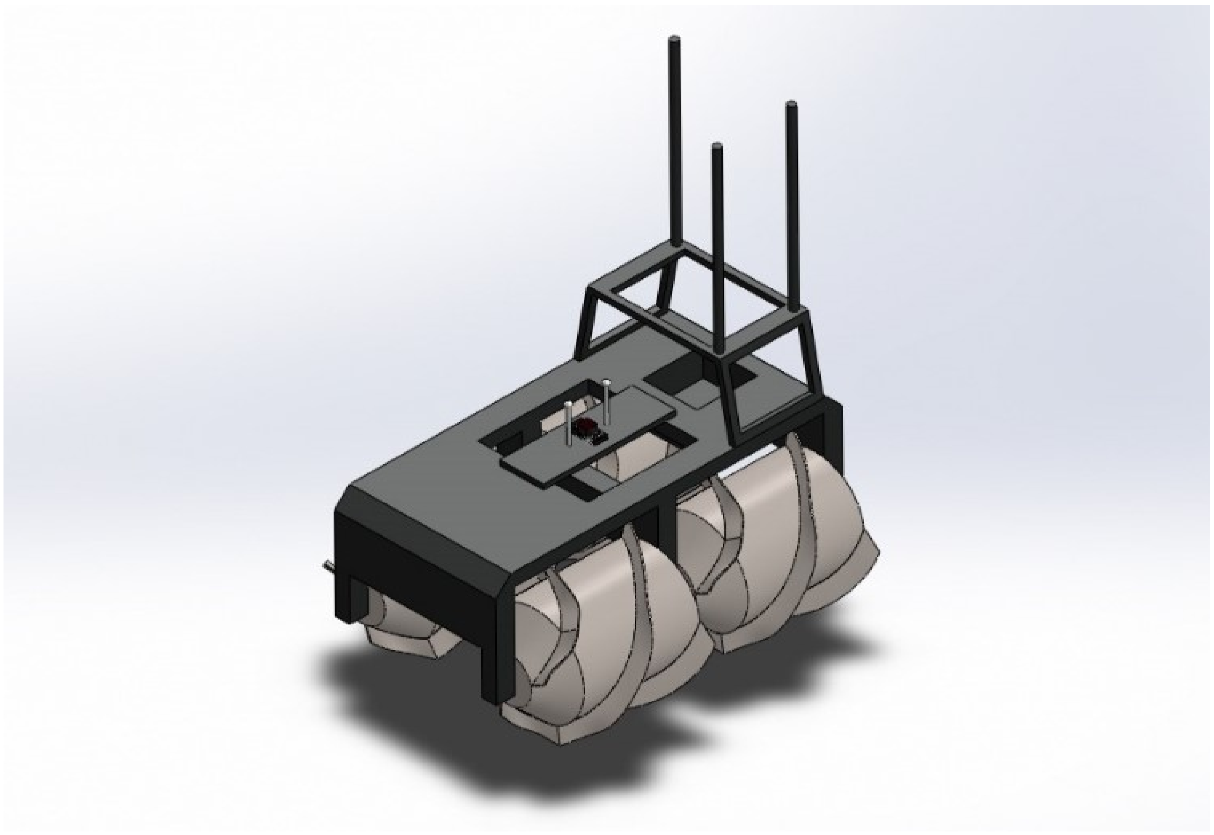


Figure 27. Digital 3D Model of Helix Screw-Propelled Rover.

Table 5. Input limits added to Helix Rover URDF, varying from ZM robot.

Properties	*Base Link*	*Scroll 1*	*Scroll 2*	*Scroll 3*	*Scroll 4*
Center of Mass X Coordinate (m)	0.0014244	0.24964	-0.26396	-0.26388	0.24915
Center of Mass Y Coordinate (m)	0.026935	-5.1047E-07	-5.1047E-07	-5.2268E-07	-5.657E-07
Center of Mass Z Coordinate (m)	-0.079565	3.7589E-07	3.7589E-07	3.6946E-07	4.6229E-07
Mass (kg)	48.194	17.19	17.19	17.19	17.19
Moment of Inertia I_{xx} (kg.m^2)	1.4866	0.62344	0.62344	0.62344	0.62344
Moment of Inertia I_{xy} (kg.m^2)	0	0	0	0	0
Moment of Inertia I_{xz} (kg.m^2)	0	0	0	0	0
Moment of Inertia I_{yy} (kg.m^2)	2.587	0.62344	0.6234	0.6234	0.6234
Moment of Inertia I_{yz} (kg.m^2)	0	0	0	0	0
Moment of Inertia I_{zz} (kg.m^2)	5.053	1.0424	1.0424	1.0424	1.0424
Joint Name	base_joint	jointLF	jointLB	jointRB	jointRF
Joint Type	Fixed	Continuous	Continuous	Continuous	Continuous
Joint Origin X (m)	0	0.25396	-0.25409	-0.25391	0.25404
Joint Origin Y (m)	0	0.01	-0.0165	0.01	-0.019
Joint Origin Z (m)	0.5	-0.3429	-0.3429	-0.3429	-0.3429
Joint Origin Roll (rad)	0	0.22071	0.22071	1.864	0.67383
Joint Origin Pitch (rad)	0	0	0	0	0
Joint Origin Yaw (rad)	0	1.5734	-1.5734	-1.5734	1.5734
Limit Velocity (rad/s)		1.5	1.5	1.5	1.5

Limit Lower (rad)		-3.1412	-3.1412	-3.1412	-3.1412
Limit Upper (rad)		3.1412	3.1412	3.1412	3.1412

This model can be further enhanced with actual 3D design models used for fabrication of Helix rover, and thereby could be made more comprehensive of its physical properties to run more advanced simulation tests that may include evaluation of force dynamics and assist for simulated testing of more sophisticated motion controllers.

4.3.2. Motion Planning Algorithms Testing

Extending the simulation tests for the Helix rover is important to ensure that the test requirements and control algorithms align with the physical and inertial properties of Helix rover. Since the algorithms have undergone correctness verification in ZM simulations, covered in the previous section, this set of tests also allows us to validate the developed digital model and its robot description.

The behavior of the Helix rover, although identical to ZM robot, may differ in control requirements and geometric capacities due to varied physical parameters and input limits for the same set of algorithms and trajectories. With that limited purpose, the simulations are only conducted on ROLL mode for the same set of three trajectories.

4.3.3. Helix Simulation – Plots and Results

The simulation tests for the Helix rover encompassed three distinct trajectory shapes: Square, Hexagon, and Circle, as shown in Figure 28. Each trajectory brought forth unique challenges and insights into the rover's expected performance. The square trajectory is 10m x 10m path, starting from the origin (0,0) with 4 waypoints. The hexagon trajectory is a 2m sided path, starting from the origin (0,0) with 6 waypoints. The circle trajectory is of radius 2m, centered at (2,2) with 25 waypoints. All the movements are in a counterclockwise direction.

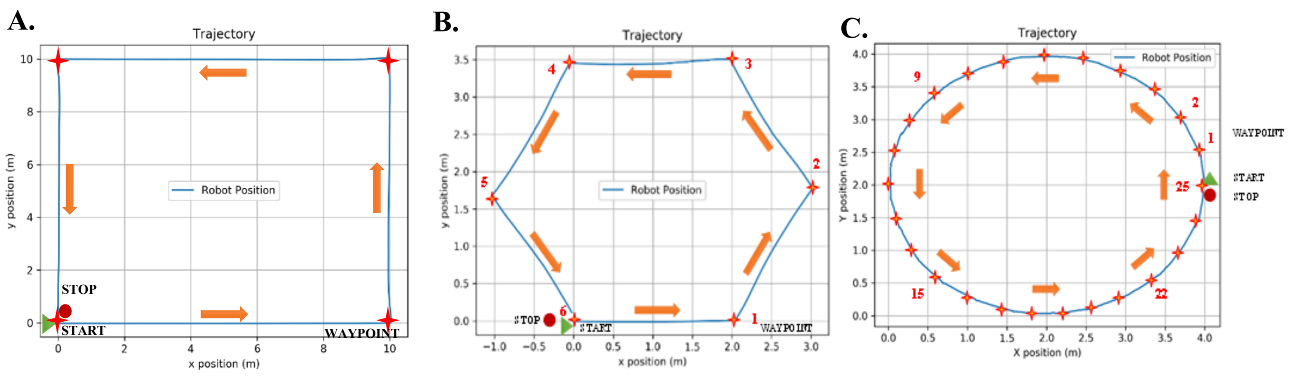
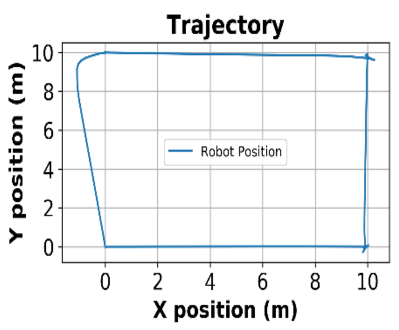
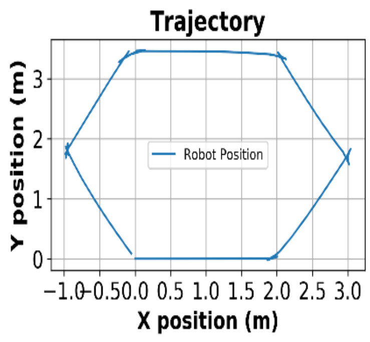
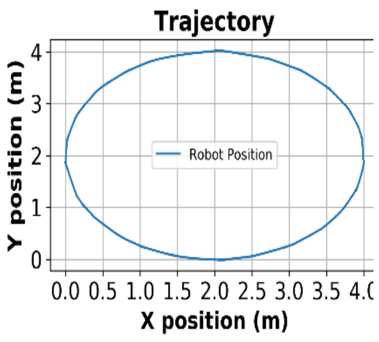
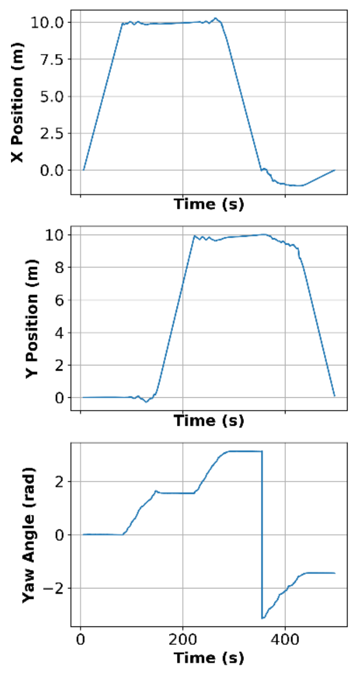
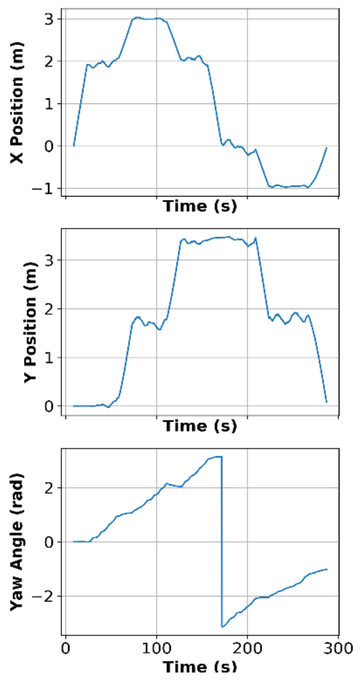
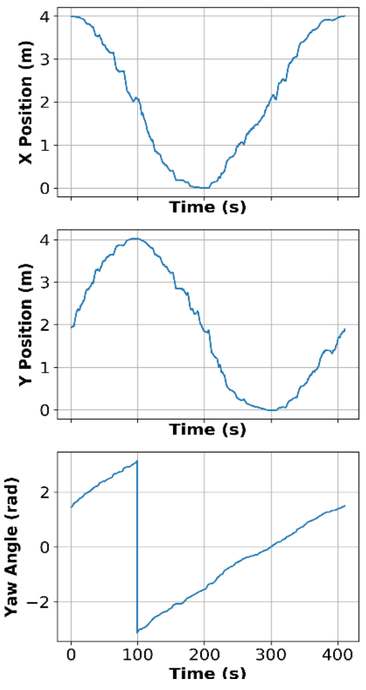
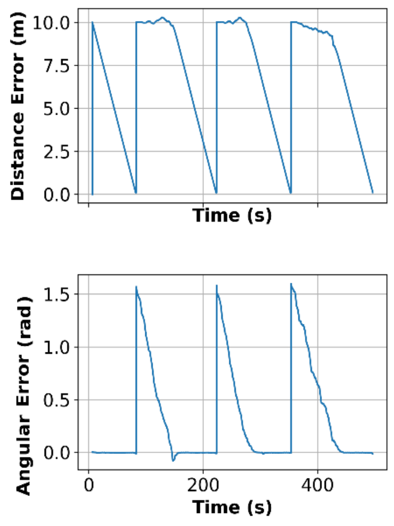
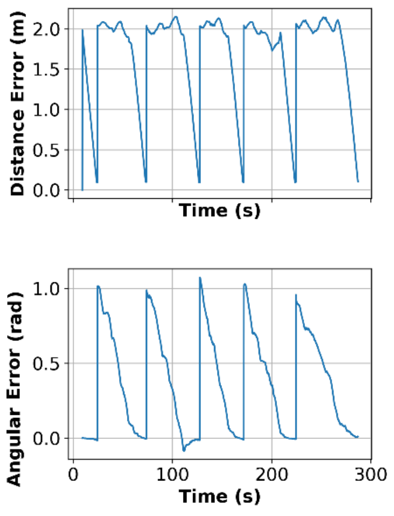
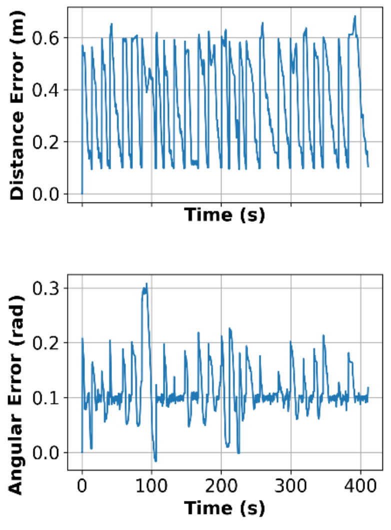
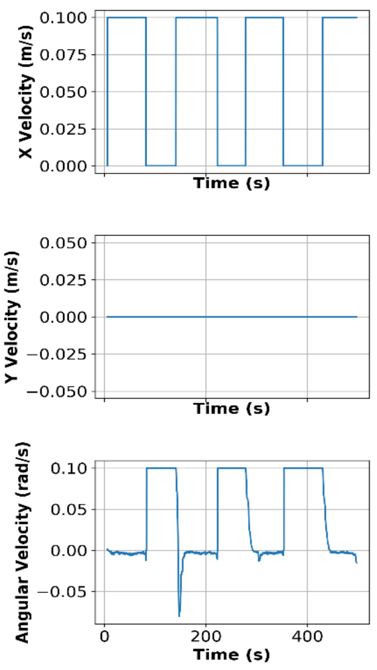
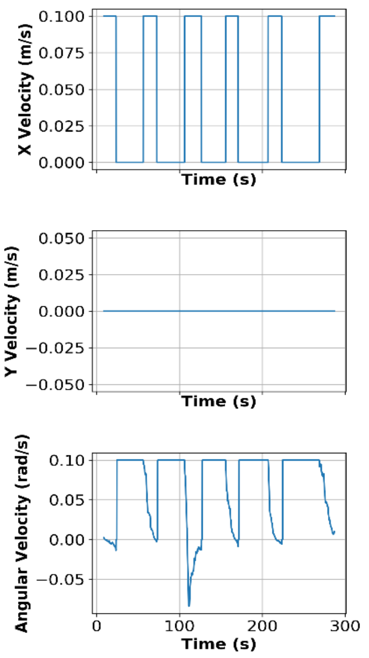
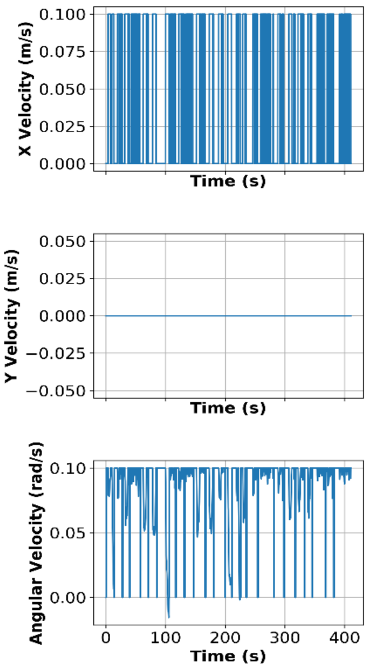
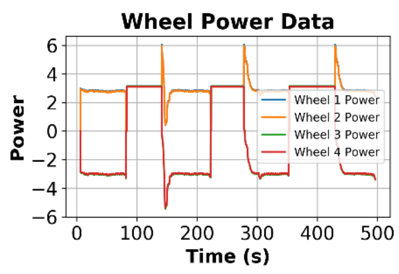
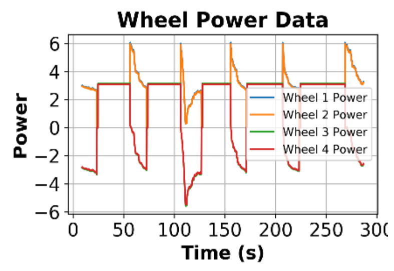
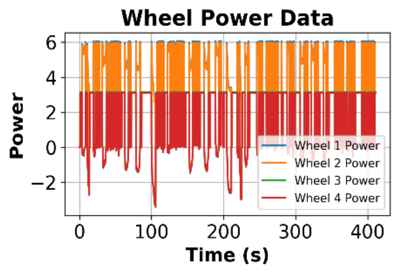


Figure 28. The definitive details of A. Square, B. Hexagon, and C. Circle trajectory movements, respectively: the start point, the stop point, the position of four waypoints, and the motion direction.

Comparing the results across the different trajectory shapes allowed for an evaluation of the ROLL mode control algorithm and Helix's expectations. The square trajectory provided insights into the rover's ability to navigate sharp turns, while the hexagon tested its performance on a path with numerous angles. The circle trajectory, being continuous, assessed the rover's precision in maintaining a curved trajectory with constant heading adjustments required. Table 6 presents a detailed analysis of the simulation results.

Table 6. Simulation results of Helix rover for execution of Roll mode algorithm for three different trajectories (i) Square, (ii) Hexagon, and (iii) Circle, respectively for square trajectory movement, which includes: (a) resultant trajectory followed, (b) position variation with time, (c) velocity variation with time, (d) error in position and angle with time, (e) wheel power (proportional to wheel rpm).

Trajectory-1 SQUARE	**Trajectory-2** HEXAGON	**Trajectory-3** CIRCLE
 TRAJECTORY		
 POSITION (X, Y, Theta)		

 ERROR (X, Y, Theta)		
 VELOCITY (X, Y, Theta)		
 WHEEL POWER / RPM		

Position Variation with Time (Table 6-a): In the Square trajectory, the rover exhibits relatively steady position tracking with minimal deviations except for the overshoot the end of third goal point, which can be mitigated by tuning k_i. The Hexagon trajectory introduces more complex curves compared to the Square. As a result, there is a slight increase in position variation, but it still remains within acceptable limits. The Circle trajectory introduces continuous curvature, resulting in slightly increased position variation. However, the rover maintains a generally smooth path.

Velocity Variation with Time (Table 6-b): The velocity profile for square shows that the rover maintains consistent speeds along the straight sections of the path and executes smooth deceleration and acceleration at the corners. The rover adeptly adjusts its speed during the sharper turns of the Hexagon, maintaining controlled movement. It also shows controlled speed adjustments, particularly during the transition from straight sections to curved segments in circle trajectory.

Error in Position and Angle with Time (Table 6-c): The error in position and angle remains low throughout the trajectory, indicating the rover's high precision in navigating the Square path. While there is a modest increase in error compared to the Square trajectory, the rover effectively manages to follow the Hexagon path. There is a further slight increase in error in both position and angle, which is expected given the continuous curvature of the Circle.

Wheel Power (Figure Table 6-d): This curve mainly confirms the synchronization and pairing of the scrolls for respective motions corresponding to each movement.

The results for all three trajectories confirm that the Roll mode algorithm successfully enables the Helix rover to navigate different path shapes effectively, confirming that the results are similar to those observed for ZM Mecanum robot.

Chapter 5 : Field Testing and Results

In order to evaluate the control logic developed for autonomous waypoint navigation, the closed-loop control needs to be tested for its adaptability to real-world terrains and practical scenarios. Although Gazebo® simulations offered verification of correctness for the algorithm, they are constrained in simulating environmental complexities and system dynamics of real scenarios. Consequently, the outcomes of control and behavior of Helix rover are sought to be validated through real-world implementation, elaborated upon in this chapter. The initial objective encompasses testing of the pose estimation module to ensure it attains the requisite accuracy before embarking on control algorithm assessments. Subsequently, the focus shifts to field trials on varied terrains.

5.1. Location and Ground Truth

The selected evaluation site is the volleyball field situated at Lister Hall Field (Street Address: 8709 117 St NW, Edmonton, AB, Canada). This well-defined area serves as a model outdoor environment for assessing and showcasing the accuracy of pose estimation, while also enabling the testing of the all-terrain capabilities of the Helix rover. The rectangular shape of the field serves as a ground truth for our testing. Geolocation is measured using Google maps measure feature.

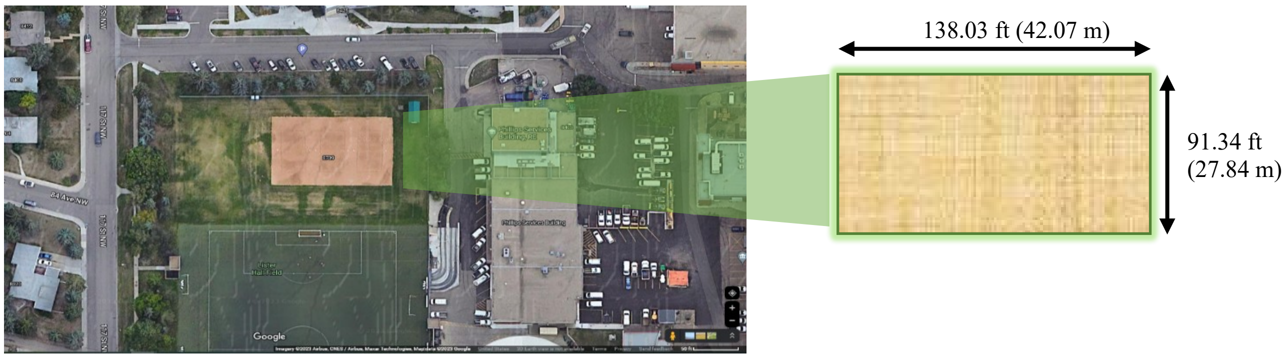


Figure 29. Aerial View and Field Illustration: The figure provides an aerial view of the field test location, revealing the court's distinct shape. Source: Google Maps

5.2. Pose Estimation

Firstly, it is important to ensure that the location chosen for the field-testing has good GPS signal reception. For that reason, a place with an open ground area with less obstruction from buildings or any other tall construction entities or trees is chosen. To verify this, the pose estimation module, made of two GPS receivers, connected to the Pixhawk system, is manually carried along the periphery of a volley-ball field. This also enables us to determine the accuracy and reliability of the pose estimation module.

The pose estimate values are relayed from the Pixhawk Flight Management Unit (FMU) to a local PC via MAVROS messages. This communication is established through a USB connection between the local PC and the FMU.

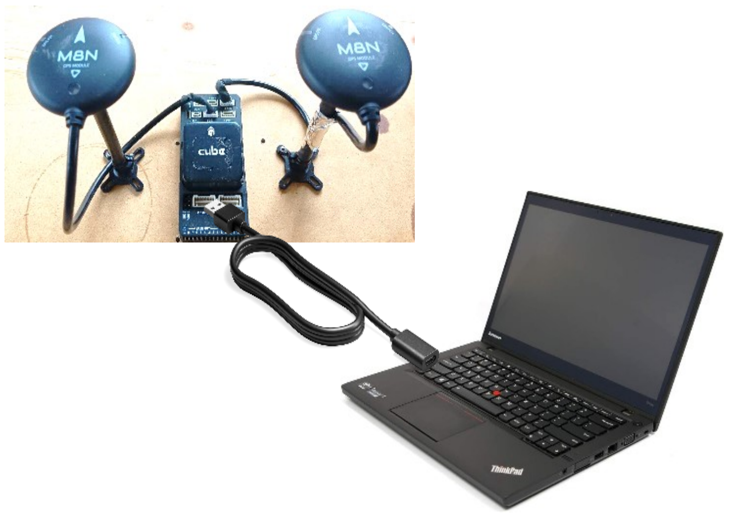
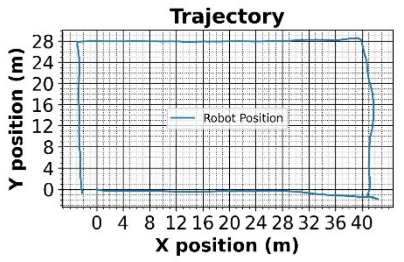
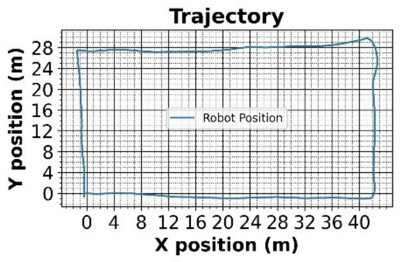
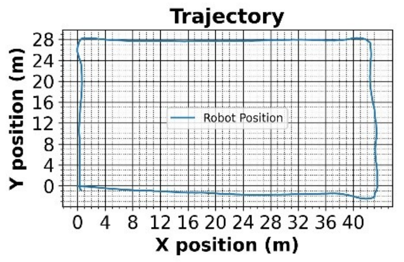
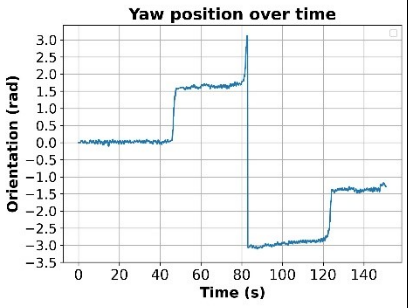
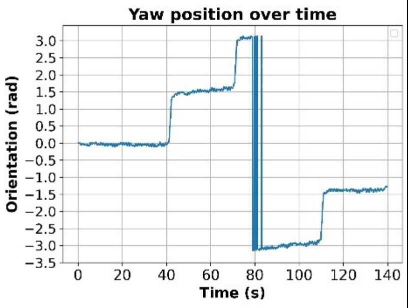
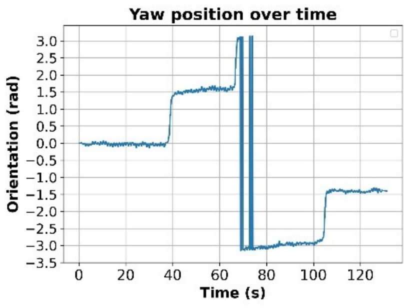


Figure 30. The stand-alone pose estimation module connected to local PC for initial evaluation.

The evaluation of the pose estimation module involved a manual traversal in a counterclockwise direction. This method was repeated multiple times for verifying its repeatability and reliability. The resulting estimates are shown in Table 7.

Table 7. Pose Estimation Verification Test results, which includes: (a) trajectory followed, (b) yaw angle error, and (c) accuracy summary per test.

Test 1	Test 2	Test 3
		
		
Position Accuracy: -2.07 m to +1.70 m Yaw Accuracy: 6 degrees Human Error: ± 1 m	Position Accuracy: -1.36 m to +2.16 m Yaw Accuracy: 3 degrees Human Error: ± 1 m	Position Accuracy: -2.30 m to +0.83 m Yaw Accuracy: 5 degrees Human Error: ± 1 m

The accuracy of the pose estimation module is approximately 1 meter with an error margin on absolute values at ± 2 meter; This error margin accounts for the human error that is expected to be within 1 meter, considering an average human stride length. The verification results also include a yaw plot that shows the 90-degree turns at each corner of the rectangular trajectory and the measured yaw error of the system that is within 10-degrees.

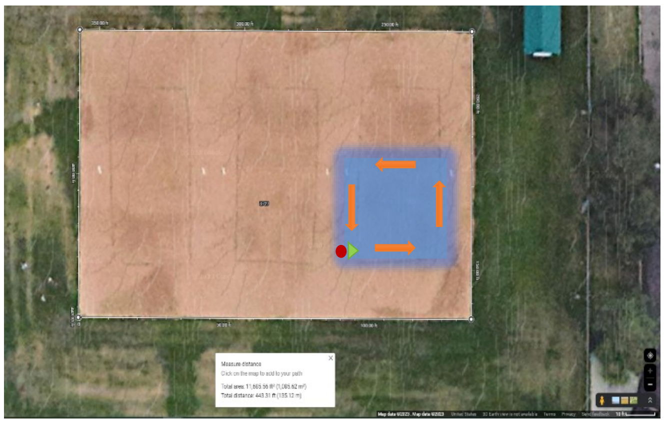
Although it is not ideal, it is reasonable to expect such results from a setup that does not use RTK or DGPS technology. However, they are still repeatable and fall within the acceptable accuracy range. It is important to note that the accuracy of the pose estimation setup may limit the accuracy of the closed-loop algorithm results.

5.3. Field Tests

The field test attempts to implement the control algorithms developed on the hardware and examine the system integration of the sensors/control interface by observing the system behavior and its response. Ideally, the plan is to test the control logic for all the modes of rover movement. The primary objective of the initial test is to assess the rover's capability to autonomously follow a designated trajectory encompassing the perimeter of the field. The trial takes place in dry sand, a cohesionless medium, chosen to evaluate the feasibility of motion control.

5.3.1. Field Test 1 – Sand Medium

The trial attempted for the first test is to run the rover inside the volleyball court to check the feasibility for SCROLL mode. The rover was placed inside the sand court and the autonomous algorithm was instantiated for a square trajectory within the internal periphery of the sand court as shown in Figure 31. The recorded observations set includes the same set of parameters traced across time and trajectory.



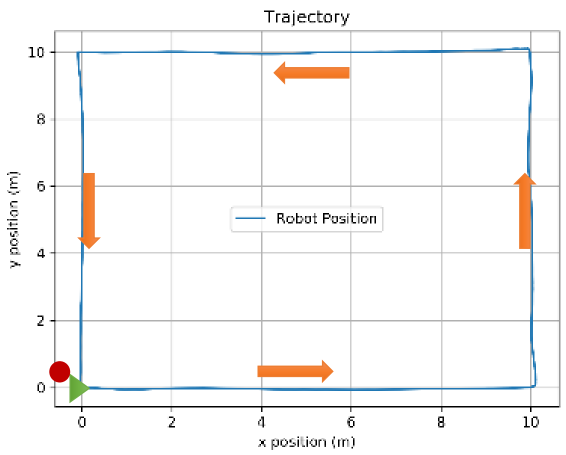


Figure 31. The illustrative details of square trajectory movement of dimensions 10 m x 10 m: the start point, the stop point, the position of four waypoints, and the direction of motion, defined within the dry sand court.

In the attempt to perform SCROLL movement in the sandy terrain, the rover encountered challenges as depicted in the figures of Table 8. It managed to progress in a straight line up to a certain point. However, as it approached the designated waypoint where a 90-degree turn was expected, issues arose. The rover's rotation within the SCROLL mode failed to execute effectively, preventing the desired change in direction at this critical juncture as shown in Figure 32.

Despite our best efforts, the rover encountered persistent difficulties during its SCROLL movement on the sandy terrain. Even manual intervention to adjust the rover's orientation proved ineffective in facilitating forward progress. A closer examination of the situation unveiled a crucial issue; The rover's scrolls encountered difficulties due to their interaction with the sandy surface. It became evident that the scrolls were not able to gain traction and were instead digging themselves into the sand. It is due to the lack of cohesion in the medium, which caused it to displace sand. This behaviour created a 'pit' behind the scrolls, as illustrated in Figure 32 (b). Consequently, the rover could not effectively turn at the spot, effectively restricting experimentation in SCROLL mode impeding the rover's ability to execute precise rotational movements. This behavior highlighted the limitations of the current version of SCROLL mode in similar terrains with less cohesion. It also emphasizes the need for more feedback and more features to be added to the control algorithm for dynamic adaptation such as: to control individual scroll movements, to optimize the turn radius, and/or to automatically retune the PID values based on the behavior.

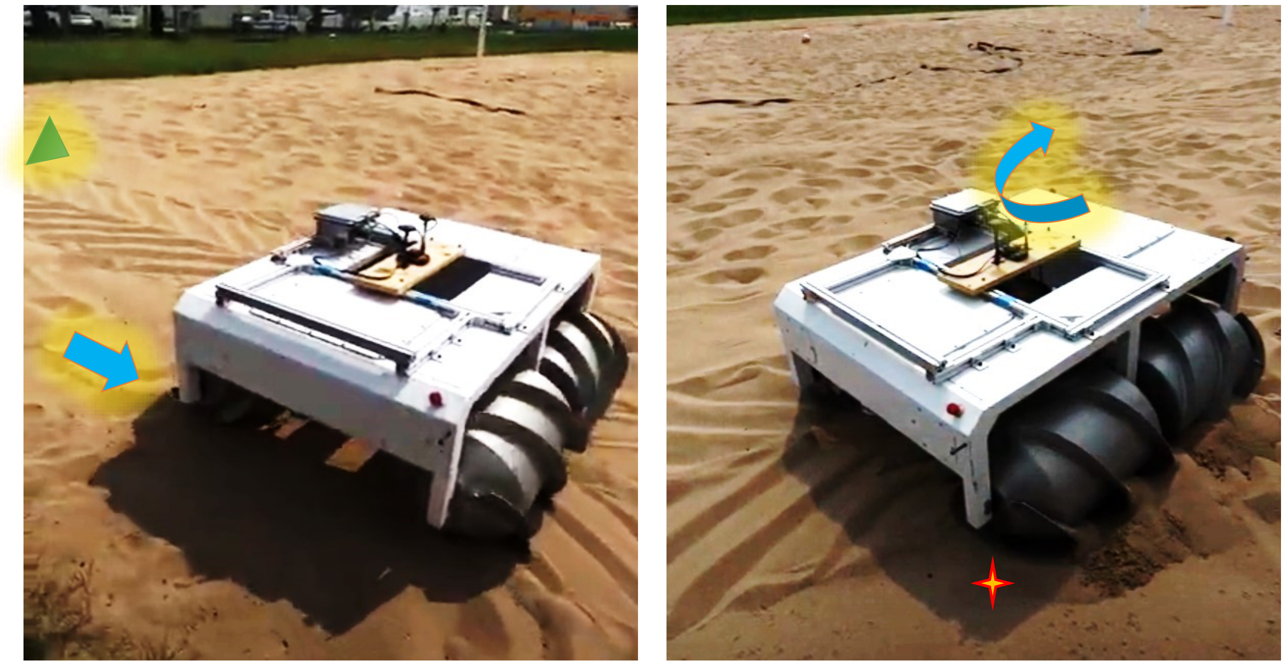


Figure 32. The results of scroll movement implementation in sand medium for a square trajectory: (a) Left image shows the start point and the direction of motion of rover along one edge of the square trajectory, (b) Right image shows the first waypoint out of the four waypoints at which the rover is expected to make a 90 degree turn on-spot to traverse along the adjacent edge of the trajectory.

Table 8. Field Test results for execution of SCROLL mode for square trajectory movement in sand medium, which includes: (a) resultant trajectory followed, (b) position variation with time, (c) velocity variation with time, (d) error in position and angle with time, (e) wheel power (proportional to wheel rpm).

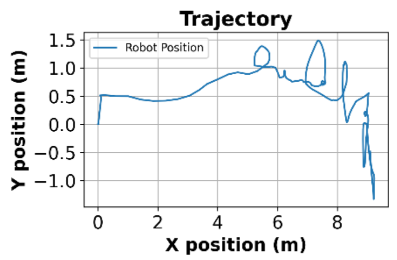
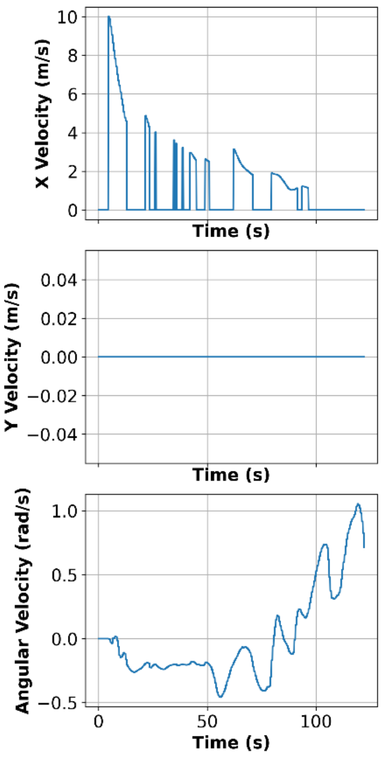
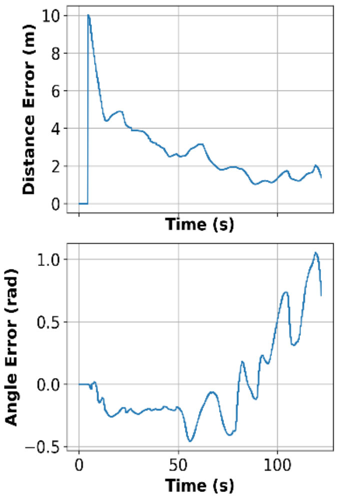
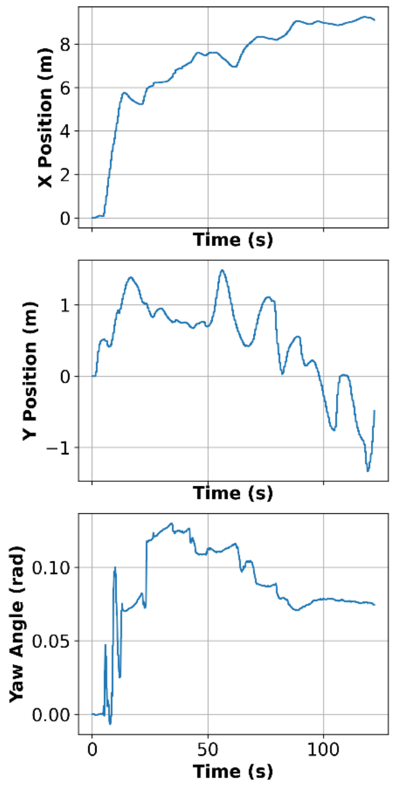
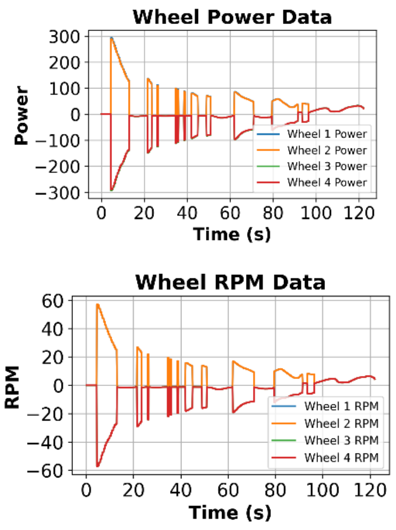
Experiment-1 SCROLL Mode		
TRAJECTORY 	VELOCITY (X, Y, Theta) 	ERROR (Distance, Heading) 
POSITION (X, Y, Theta) 		WHEEL POWER / RPM 

Table 8 elucidates the rover's limitations and performance challenges in kinematic plots when operating in a sandy environment as discussed previously. In trajectory plot, it can be clearly seen that the rover initially exhibited commendable straight-line movement, adhering to the desired trajectory within the sand medium. However, when attempting to execute a 90-degree turn at the waypoint, it encountered substantial difficulties. Notably, it indicates how the rover initially struggled to make the turn in the trajectory plot, leading to substantial positional and heading errors. The rover's scrolls struggled to gain traction for rotation, effectively stalling the rover's motion forward. Due to the limited approach of control algorithm, the rover is severely hampered to complete the desired trajectory. As discussed in Figure 4 for scroll mechanics, the effective result of frictional forces versus the thrust force components reflects on the movement of the scroll and its slipping conditions. Those dynamics are completely determined by the interaction mechanics of the medium, which are caused by its specific physical properties during execution. For instance, in this case, the extent of dryness of the sand inherently affects its cohesiveness, which could be dependent on specific time of day, weather, and/or external interventions.

This initial field test conducted within a sand medium provided valuable insights into the rover's requirements from motion control algorithms, particularly in the SCROLL mode. However, the results of this test highlighted significant challenges and limitations that warrant a shift in our testing approach. To ensure more controlled and reliable evaluations of our autonomous motion control algorithms, we have chosen to conduct tests solely within hard, solid mediums.

5.3.2. Field Test 2

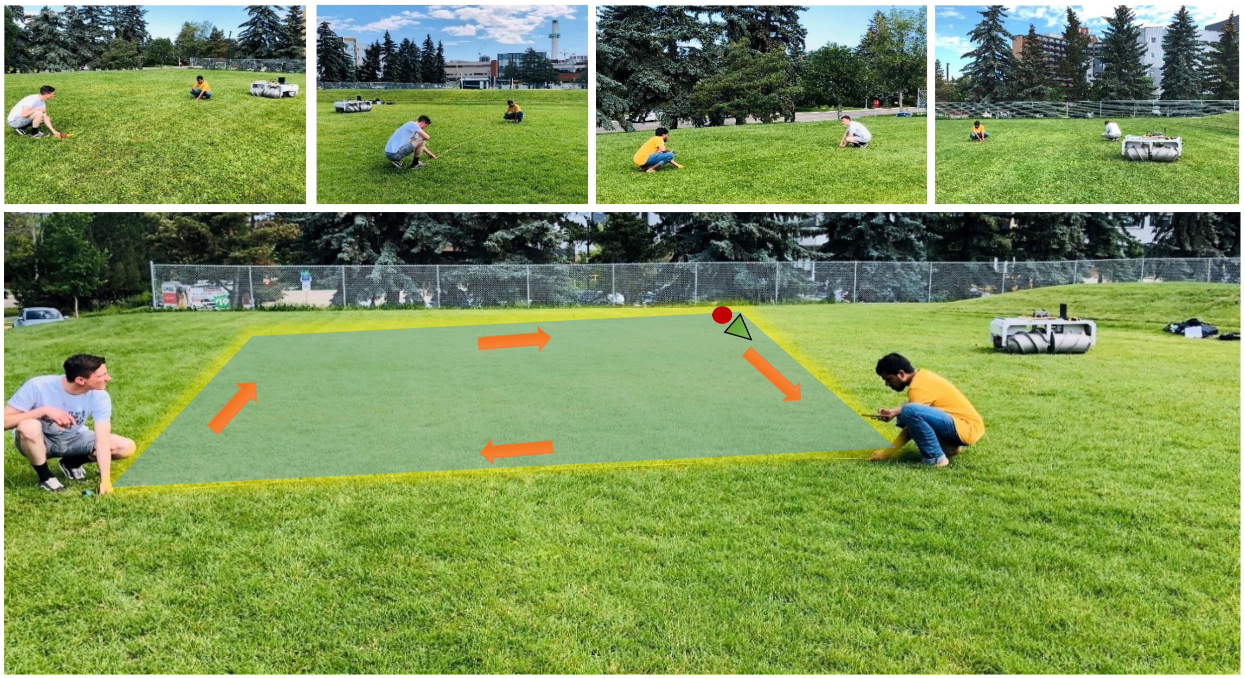


Figure 33. The start point, the stop point, and the direction of movement are displayed in this image of a square trajectory of dimensions 10 m x 10 m, where the ground truth dimensions are measured and marked for validation. The inset pictures show the dimensions taken for other edges of the square trajectory for reference.

Given the objective of conducting a series of three comprehensive tests, it becomes apparent that the sand medium chosen for the initial trial does not provide the necessary cohesion to effectively support the

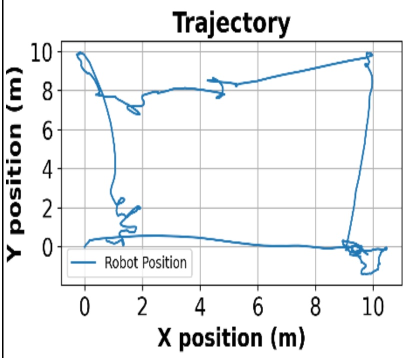
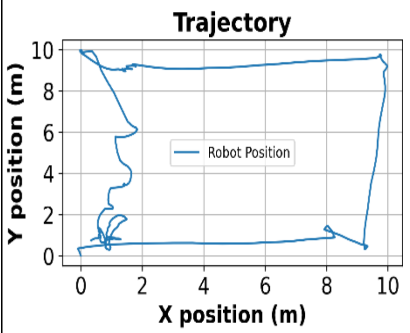
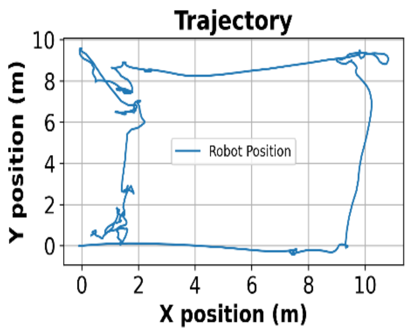
SCROLL locomotion of the rover within the limitations of the algorithm that outputs only explicit control actions for straight-line movements or turns. Therefore, we move ahead to testing the ROLL mode.

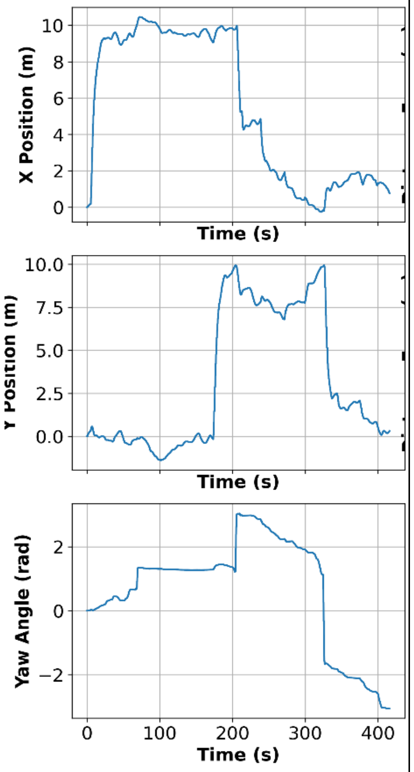
In Figure 33, we provide a visual representation of the experimental setup for ROLL mode testing on a grassy, hard surface. The Figure 33 includes the starting point, the stopping point, and the direction of movement, all of which define a square trajectory with dimensions of 10 m x 10 m. To ensure the accuracy of our testing, the ground truth dimensions are measured and marked within the trajectory.

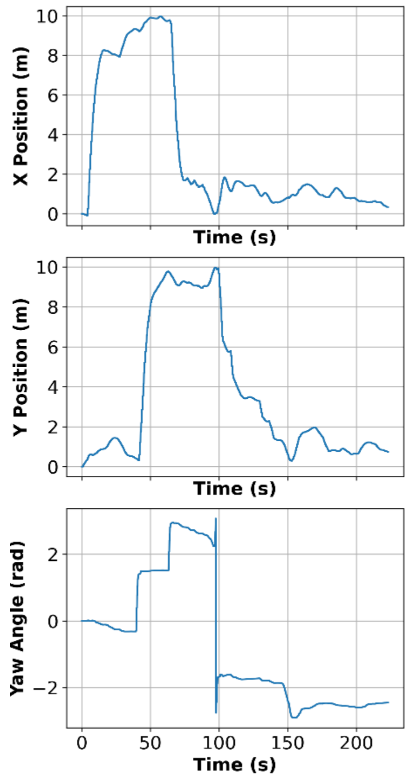
The rover is run for an initial set of tests on a 5 m x 5 m square trajectory on grass with default values for proportional (K_P), integral (K_I) and derivative (K_D) gains. For reducing the deviations of the rover from desired path, K_P is set to 1.2 and 8 for X and Z axis respectively. With the CG of Helix rover being at a low position and close to ground, the rover is quite stable within the operational limits. So, the effect of K_I is focused to reduce steady-state accumulation errors only and hence set to its minimum value of 0.0002. For further dampening the oscillating error rates for positional and heading adjustments, K_D is set to 0.05. The goal was to strike a balance between maintaining proximity to the desired trajectory while avoiding erratic behaviour.

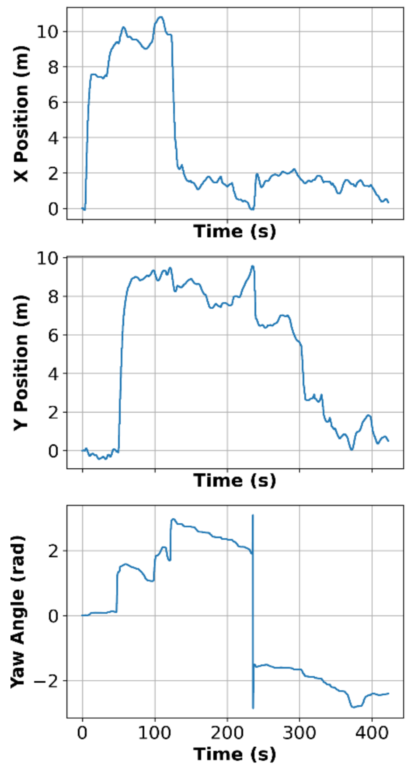
The acceptable tolerance values were determined through empirical testing. Having systematically tuned the PID parameters to fine-tune the rover's control, we tested the rover on a 10 m x 10 m square trajectory on a grassy, hard surface.

Table 9. Field Test results for execution of three experiments for ROLL for 10m x 10m square trajectory movement on grass, which includes: (a) resultant trajectory followed, (b) position variation with time, (c) velocity variation with time, (d) error in position and angle with time, (e) wheel power (proportional to wheel rpm).

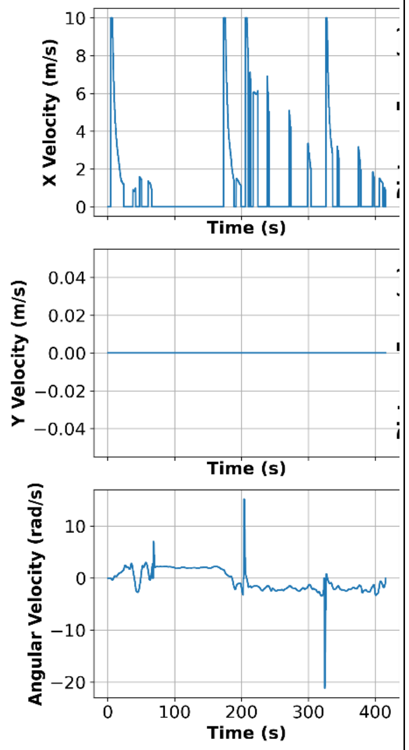
Experiment-6 ROLL Mode	**Experiment-7 ROLL Mode**	**Experiment-8 ROLL Mode**
 **`TRAJECTORY**		

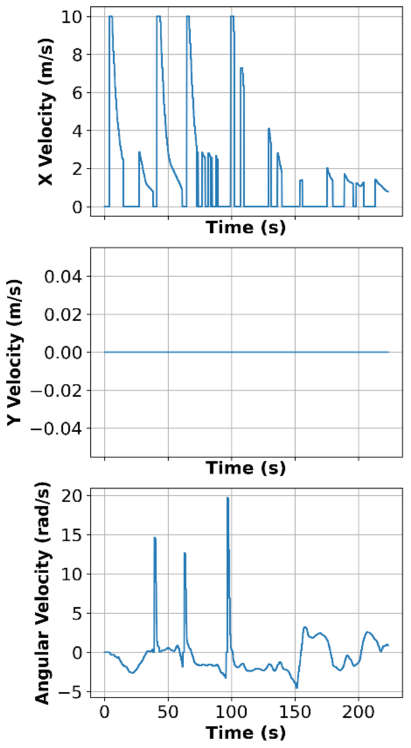


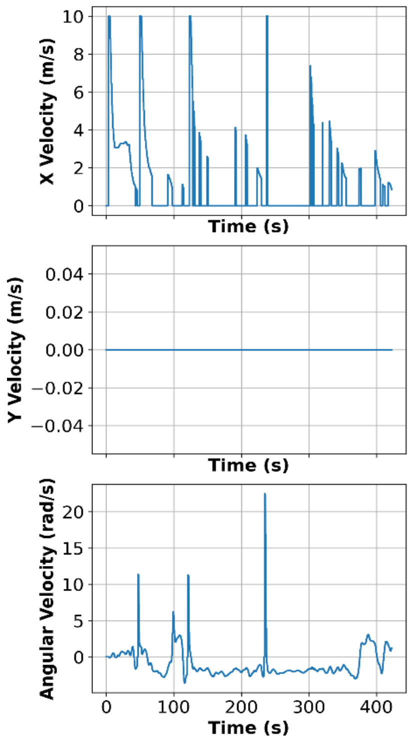




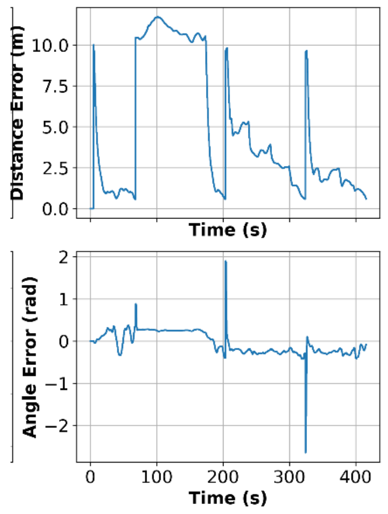
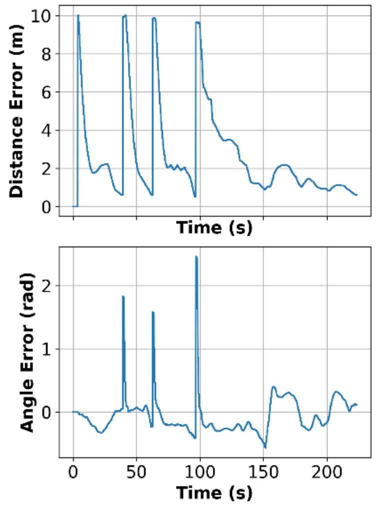
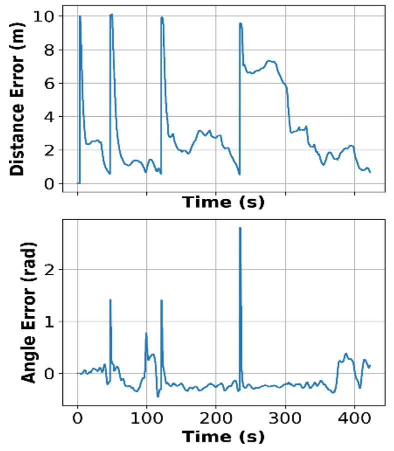
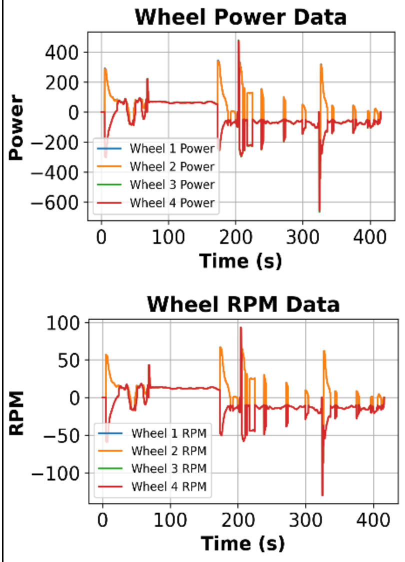
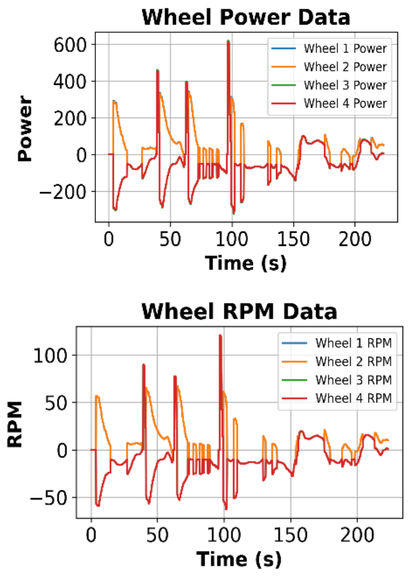
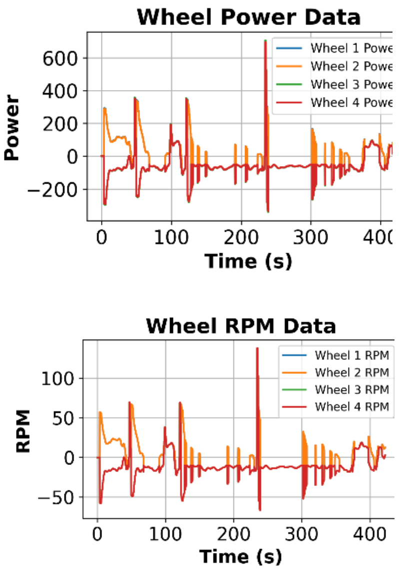
POSITION (X, Y, Theta)







VELOCITY (X, Y, Theta)

ERROR (Distance, Theta) 		
 **WHEEL POWER / RPM**		

In this final field test on the 10 m x 10 m square trajectory, as depicted in Table 9, we observed an improvement in the rover's performance after fine-tuning the PID control parameters. The resultant trajectory showcases a square path, closely adhering to the intended route. As we delve into the specifics, the positional variations with time and the velocity variations exhibit consistent behaviour. The errors in position and angle remain well within the acceptable tolerance limits, signifying the control exerted by the PID system.

The wheel power, which proportionally imitates the scroll revolutions per minute, offers a glimpse into the synchronization of scrolls. This representative curve serves as a reference to desired power consumption to track the actual log of the current consumption or torque exertion by motor, which can serve as additionally valuable feedback to further identify the scroll's dynamic behaviour during slippage or pitting in any terrain such as the one observed for sand medium in Field Test 1. This deviation data between expected and actual power consumption could be most useful feedback for advanced versions of control logics developed in this project, and for more sophisticated Model-based Predictive Control methods as well, without needing to add extra sensors for the same. It must be noted that this still serves only in the synchronized curve behaviour tracking but not the magnitude of power consumption as it varies with media properties.

Overall, these outcomes assess the practicality of the developed motion control algorithms and highlight the limitations of their preliminary versions. It is the first step towards facilitating Helix rover for further research and practical applications in the realm of autonomous navigation and terrain-adaptable control.

In conclusion, this on-field testing of our motion control algorithms in a real-world setting, as detailed in this chapter, marks the first milestone towards practical implementation of basic control logic. While the initial challenges in the sand medium revealed limitations in the SCROLL mode and hindered the tests for omni-directional mode as well, the ROLL mode could still be tested for proof-of-concept on hard grass terrain. The rover successfully navigated the 10 m x 10 m square trajectory, closely mirroring our simulation expectations. This outcome underscores the adaptability and accuracy of our control algorithms and highlights their seamless transition from simulated scenarios to the real world.

Chapter 6 : Conclusions and Future Work

This thesis studied the development, simulation, and real-world validation of control algorithms for the Helix rover. The foundation of this work rests upon the controller integration in hardware and the control algorithms developed for different modes of motion, initially tested on a simulation environment with a 3D model designed using SolidWorks® for verification, and thereafter implemented on hardware for on-field testing.

The integration of FMU, PIXHAWK, rover, and local PC demonstrated effective communication and collaboration, laying the groundwork for on-field applications. The reasonable success of the Helix rover tests was fortified by the acceptable precision of the pose estimation module, a fusion of GPS and IMU technologies. This module exhibited proof-of-concept in guiding the rover's autonomous movements for outdoor environment with a few conditional requirements.

This thesis validated that by interfacing the Helix rover system with a basic controller setup, it could serve as a promising starting point for Copperstone Technologies®, an Edmonton-based startup, to venture into autonomous on-field operations within adverse environments. While the scroll and omni directional modes remain untested, the results achieved in the Roll mode lay a proven framework for more investigations into various terrains and control modes. These outcomes serve as a testament to the system's viability, demonstrating that with a suitable testbed and refined PID tuning, other algorithms can confidently follow suit and also be more successful with adaptive mechanisms, provided it has access to more feedback. In conclusion, this study successfully demonstrates the fundamental functioning of the autonomous waypoint navigation of helix rover for a predefined trajectory in roll mode.

6.1. Scope for Future Work

1. **Advancements in Localization:**

    - **LiDAR and Motion Cam Integration:** To enhance pose estimation accuracy and reduce reliance on GPS, a shift to high-precision sensors like Velodyne LiDAR or vision camera such as Intel RealSense T265 could significantly improve localization performance. This transition would mitigate errors associated with GPS, such as atmospheric disturbances and multipath reflections, enabling the Helix rover to navigate with higher precision and reliability.

2. **Refinement of Robotic Systems:**

    - **Resolution of RC Transmitter Signal Loss:** Rigorously examine and address the issue of the rover entering hold mode due to signal loss from the Remote Control (RC) transmitter.

Potential causes, such as antenna performance, should be thoroughly investigated to ensure uninterrupted rover operation.

- **Weather-Resistant Pose Estimate Systems:** Implement comprehensive weatherproofing measures for exposed add-on electronic components such as GPS and FMU units of the Helix rover. This protection will mitigate risks associated with sudden rain and dust exposure during field testing, enhancing the rover's robustness.
- **On-line parameter estimation of interaction forces:** Having a method to estimate interaction force can help in improving control strategy. This can serve as one of the feedback items for the control strategy to decide the output of controller.
- **Sensor Fusion with Visual Cue:** It can help in classifying the terrains while driving when the robot navigates through various terrain and can help in selecting control strategy based on the terrain in which the robot is moving now.

3. **Thorough On-Field Testing:**
    - **Testing in Cohesive Soil Environments:** Testing of control algorithms across diverse terrains, including cohesive soil like mud or snow, will provide insights into the Helix rover's capabilities in challenging environments.
    - **Fine Tuning PID for Scroll and Omni Directional mode:** Further fine-tuning and testing of PID controller parameters for each algorithm mode will optimize performance and stability during autonomous operations.
4. **Advancements in Control Strategies:**
    - **Multi-Axis Control Testing:** Test control algorithms which enable the Helix rover to perform multi-axis movements.
    - **Intermediate Path Planning Integration**: Enhance control algorithms by incorporating intermediate path planning between waypoints. This strategic approach reduces cumulative errors and elevates the accuracy of trajectory tracking and overall motion control.
    - **Dynamic Formulations**: Modelling of the dynamics of the system and implementing control strategies based on them.
    - **Exploration of Model-Based Predictive Control (MPC):** Delve into the implementation of Model-Based Predictive Control techniques. Also gain scheduling or MPC for adaptive control in different terrains with variable soil interaction forces and torques.

- **Additional Feedback Interface:** Despite of modelling the scroll interactions in sand, we may not be able to confidently predict the rover behaviour without the aid of additional sensors to be aware of slip, topple or non-contact situations of individual scrolls.

By venturing into these areas, the research will advance the Helix rover's capabilities. This will lead to a more capable robotic platform, capable of relieving the tedium of performing missions such as sampling and surveying currently carried out entirely by a human operator.

Bibliography

[1] "Unusual Locomotion: Screw Propelled Vehicles," Unusual Locomotion, [Online]. Available: https://www.unusuallocomotion.com/pages/locomotion/screw-propelled-vehicles.html. [Accessed 31 08 2023].

[2] Z. Y. Wu, J. Qi and S. Zhang, "Amphibious robots: a review," *Applied Mechanics and Materials,* vol. 494, pp. 1036 -- 1041, 2014.

[3] J. Y. Wong, Theory of ground vehicles, New York: John Wiley & Sons, 2001.

[4] A. Roennau, G. Heppner, M. Nowicki, J. M. Zollner and R. Dillmann, "Reactive posture behaviors for stable legged locomotion over steep inclines and large obstacles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 4888--4894*, 2014.

[5] E. Guizzo, "By leaps and bounds: An exclusive look at how boston dynamics is redefining robot agility," *IEEE Spectrum,* vol. 56, no. 12, pp. 34 -- 39, 2019.

[6] J. Morath, "Agricultural Machines". United States of America Patent 635501, 18 05 1899.

[7] K. Nagaoka, M. Otsuki, T. Kubota and S. Tanaka, "Terramechanics-based propulsive characteristics of mobile robot driven by Archimedean screw mechanism on soft soil," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 4946 -- 4951*, 2010.

[8] D. Osinski and K. Szykiedans, "Small remotely operated screw-propelled vehicle," in *Springer: Progress in Automation, Robotics and Measuring Techniques: Volume 2 Robotics, 191 -- 200*, 2015.

[9] J. Liedke, L. Winkler and H. Worn, "An alternative locomotion unit for mobile modular self-reconfigurable robots based on archimedes screws," in *IEEE 9th International Symposium on Mechatronics and its Applications (ISMA)*, 2013.

[10] M. Safar, Y. Chandradekaran, S. Basah, K. Basaruddin and M. Hashim, "Kinematic analysis of a screw wheeled omnidirectional mobile robot," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC),* vol. 10, no. 1 - 15, pp. 111 -- 115, 2018.

[11] J. T. Freeberg, A study of omnidirectional quad-screw-drive configurations for all-terrain locomotion, United states of America: University of South Florida, 2010.

[12] K. Nagaoka, M. Otsuki, T. Kubota and S. Tanaka, "Development of lunar exploration rover using screw propulsion units: Note on dynamic behavior and moving direction control," *19th Work. JAXA Astrodyn. Flight Mech., Kanagawa, Japan,* pp. 143 - 148, 2009.

[13] J. Villacrés, M. Barczyk and M. Lipsett, "Literature review on Archimedean screw propulsion for off-road vehicles," *Journal of Terramechanics,* vol. 108, pp. 47 - 57, 2023.

[14] V. Leonov and O. Kenlip, "Screw drive vehicle". United States of America Patent 6966807, 2005.

[15] J. Yuen, O. Nicolas, D. Stephen and L. Michael, "All-terrain vehicle". United States of America Patent 10076938B2, 08 03 2016.

[16] M. Danesh Panah, A. Abdollahi, H. Ostadi and H. A. Samani, "Comprehensive omni-directional soccer player robots," *IntechOpen - Robotic Soccer,* 2007.

[17] D. Stonier, S.-H. Cho, S.-L. Choi, N. S. Kuppuswamy and J.-H. Kim, "Nonlinear slip dynamics for an omniwheel mobile robot platform," *Proceedings 2007 IEEE International Conference on Robotics and Automation,* pp. 2367 - 2372, 2007.

[18] W. Loh, K. H. Low and Y. Leow, "Mechatronics design and kinematic modelling of a singularityless omni-directional wheeled mobile robot," *IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422),* vol. 3, pp. 3237 - 3242, 2003.

[19] P. F. Muir and C. P. Neuman, "Kinematic modeling for feedback control of an omnidirectional wheeled mobile robot," *Autonomous robot vehicles,* pp. 25 - 31, 1990.

[20] C.-C. Wong, Y.-H. Lin, S.-A. Lee and C.-H. Tsai, "GA-based fuzzy system design in FPGA for an omni-directional mobile robot," *Springer: Journal of Intelligent and Robotic Systems,* vol. 44, pp. 327 - 347, 2005.

[21] Y. Liu, J. J. Zhu, R. L. Williams II and J. Wu, "Omni-directional mobile robot controller based on trajectory linearization," *Robotics and autonomous systems,* vol. 56, no. 5, pp. 461 - 479, 2008.

[22] H.-C. Huang and C.-C. Tsai, "FPGA implementation of an embedded robust adaptive controller for autonomous omnidirectional mobile platform," *IEEE Transactions on Industrial Electronics,* vol. 56, no. 5, pp. 1604 - 1616, 2008.

[23] M. Velasco-Villa, H. Rodriguez-Cortes, H. Sira-Ramirez, I. Estrada-Sanchez and J. Vazquez, Dynamic trajectory-tracking control of an omnidirectional mobile robot based on a passive approach, Advances in Robot Manipulation, 2010.

[24] G. Klancar and I. skrjanc, "Tracking-error model-based predictive control for mobile robots in real time," *Robotics and autonomous systems,* vol. 55, no. 6, pp. 460 - 469, 2007.

[25] T. P. Nascimento, C. E. T. Dorea and L. M. G. Gonccalves, "Nonlinear model predictive control for trajectory tracking of nonholonomic mobile robots: A modified approach," *International Journal of Advanced Robotic Systems,* vol. 15, no. 1, p. 1729881418760461, 2018.

[26] A. G. Conceicao, C. E. Dorea and J. C. B. Sb, "Predictive control of an omnidirectional mobile robot with friction compensation," *Latin American Robotics Symposium and Intelligent Robotics Meeting,* pp. 30 - 35, 2010.

[27] K. J. AAstrom and T. Hagglund, "Advanced PID control," *ISA-The Instrumentation, Systems and Automation Society,* 2006.

[28] Z. Zeng, H. Lu and Z. Zheng, "High-speed trajectory tracking based on model predictive control for omni-directional mobile robots," *IEEE 25th Chinese Control and Decision Conference (CCDC),* pp. 3179 - 3184, 2013.

[29] H. X. Araujo, A. G. Conceiccao, G. H. Oliveira and J. Pitanga, "Model predictive control based on LMIs applied to an omni-directional mobile robot," *IFAC Proceedings Volumes,* vol. 44, no. 1, pp. 8171 - 8176, 2011.

[30] T. D. Viet, P. T. Doan, N. Hung, H. K. Kim and S. B. Kim, "Tracking control of a three-wheeled omnidirectional mobile manipulator system with disturbance and friction," *Journal of mechanical science and technology,* vol. 26, pp. 2197 - 2211, 2012.

[31] A. G. S. Conceiccao, C. E. Dorea, L. Martinez, E. R. de Pieri and others, "Design and implementation of model-predictive control with friction compensation on an omnidirectional mobile robot," *IEEE/ASME Transactions On Mechatronics,* vol. 19, no. 2, pp. 467 - 476, 2013.

[32] T. A. Teatro, J. M. Eklund and R. Milman, "Nonlinear model predictive control for omnidirectional robot motion planning and tracking with avoidance of moving obstacles," *Canadian Journal of Electrical and Computer Engineering,* vol. 37, no. 3, pp. 151 - 156, 2014.

[33] F. Cuevas, O. Castillo and P. Cortes-Antonio, "Towards an Adaptive Control Strategy Based on Type-2 Fuzzy Logic for Autonomous Mobile Robots," *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE),* 2019.

[34] A. Thoesen, T. McBryan, D. Mick, M. Green, J. Martia and H. Marvi, "Granular scaling laws for helically driven dynamics," *Physical Review E,* vol. 102, no. 3, p. 032902, 2020.

[35] D. He and L. Long, "Design and Analysis of a Novel Multifunctional Screw-propelled vehicle," *IEEE International Conference on Unmanned Systems (ICUS),* 2017.

[36] Y. Liberman, N. Shonokhova and O. Lukashuk, "Development of Control Systems for Screw Propellers," *Proceedings of the 6th International Conference on Industrial Engineering ,* 2021.

[37] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan and N. B. Ismail, "Review of visual odometry: types, approaches, challenges, and applications," *SpringerPlus,* vol. 5, p. 1897, 2016.

[38] J. P. Queralta, C. M. Almansa, F. Schiano, D. Floreano and T. Westerlund, "Uwb-based system for uav localization in gnss-denied environments: Characterization and dataset," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 4521 - 4528*, 2020.

[39] D. o. Defense, *GLOBAL POSITIONING SYSTEM STANDARD POSITIONING SERVICE PERFORMANCE STANDARD (5E),* Washington, DC: Department of Defense, 2020.

[40] B. Hofmann-Wellenhof, H. Lichtenegger and J. Collins, Global Positioning System: Theory and Practice., Springer-Verlag, 2012.

[41] S. K. Moore, "Super-accurate GPS coming to smartphones in 2018 [News]," *{IEEE Spectrum,* vol. 54, no. 11, pp. 10 - 11, 2017.

[42] R. Ross and R. Hoque, "Augmenting GPS with geolocated fiducials to improve accuracy for mobile robot applications," *Applied Sciences,* vol. 10, no. 1, p. 146, 2019.

[43] M. G. Wing, A. Eklund and L. D. Kellogg, "Consumer-grade global positioning system (GPS) accuracy and reliability," *Journal of forestry,* vol. 103, no. 4, pp. 169 - 173, 2005.

[44] J. Farrell, Aided Navigation: GPS with High Rate Sensors, McGrawHill professional engineering: Electronic engineering. McGraw-Hill Education, 2008.

[45] S. Liu, K. Wang, D. Abel and R. Zweigel, "Robust State Estimation and Integrity Monitoring within Multi-Sensor Navigation System," *Electrical Engineering and Systems Science; Systems and Control,* 2021.

[46] S. Ismaeel, "Design of Kalman Filter of Augmenting GPS to INS Systems, Ph.D," Computer Engineering Dept., College of Engineering, Al-Nahrain University, 2003.

[47] R. Sndlinger, "Investigations on the Optimization of Aided Inertial Navigation System," *Advisory Group for Aerospace Research and Development No.95, Strapdown Systems,* pp. 124 - 139, 1978.

[48] R. Lawrence, W. Mohinder, S. Grewal and A. Andrews, Global Positioning Systems, Inertial Navigation, and Integration,, John Wiley & Sons, Inc., 2001.

[49] V. Gupta., "Vehicle localization using low-accuracy gps, imu and map - aid vision," The Pennsylvania State University., 2009.

[50] O. Maklouf, A. Abdulla, A. Yousef and A. Ghila, "Low Cost IMU-GPS Integration using Kalman Filtering for Land Vehicle Navigation Application," International Journal of Electrical, Computer Energetic, Electronic and Communication Engineering., 2013.

[51] S. M. Martin, "Closely coupled gps/ins relative positioning for automated vehicle convoys,," Auburn University, 2011.

[52] I. I. Lucian, J. Collin and J. Takala, "Integrating MEMS Sensors with GPS Technology for Obtaining a Continuous Navigation Solution in Urban Areas," Signal Processing and Applied Mathematics for Electronics and Communication (SPAMEC), Cluj-Napoca, Romania, 2011.

[53] Y. Zhao, "GPS/IMU Integrated System for Land Vehicle Navigation based on MEMS,," Royal Institute of Technology. Division of Geodesy and Geoinformatics, Sweden, 2011.

[54] B. Leong and S. M. Low, "Low-cost microcontroller-based hover control design of a quadcopter," *Procedia Engineering, vol. 41, pp. 458–464.,* vol. 41, no. 2, pp. 458 - 464, 2012.

[55] H. Lim, J. Park, D. Lee and H. Kim, "Build your own quadrotor: Open source projects on unmanned aerial vehicles," *IEEE Robotics & Automation Magazine,* vol. 19, no. 3, pp. 33 - 45, 2012.

[56] J. Garcia, J. Molina and J. Trincado, "Analysis of real data with sensors and estimation outputs in configurable UAV platforms," *Sens. Data Fusion,* 2017, 10–12 Oct..

[57] C. Bergquist, "Selecting a Drone Flight Controller," Drone Dojo, [Online]. Available: https://dojofordrones.com/drone-flight-controller/. [Accessed 1 12 2023].

[58] RoboCraze, "Pixhawk 2.4.8 Drone Flight Controller PX4 32 Bit Autopilot," RoboCraze, [Online]. Available: https://robocraze.com/products/pixhawk-2-4-8-drone-flight-controller-px4-32-bit-autopilot. [Accessed 1 12 2023].

[59] NVIDIA, "Jetson TX2 Module," 1 10 2023. [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx2. [Accessed 1 10 2023].

[60] CubePilot, "The Cube Module Overview," [Online]. Available: https://docs.cubepilot.org/user-guides/autopilot/the-cube-module-overview. [Accessed 02 12 2023].

[61] PX4 Auto Pilot, "PX4 Auto Pilot," PX4, 1 10 2023. [Online]. Available: https://docs.px4.io/main/en/flight_controller/pixhawk-2.html. [Accessed 1 10 2023].

[62] ArduPilot, "The Cube Black," ArduPilot, [Online]. Available: https://ardupilot.org/copter/docs/common-thecube-overview.html. [Accessed 02 12 2023].

[63] GPS, PX4 Autopilot, "PX4 Autopilot GPS," PX4, 1 10 2023. [Online]. Available: https://docs.px4.io/main/en/gps_compass/gps_holybro_m8n_m9n.html. [Accessed 1 10 2023].

[64] Ardupilot, "EKF-3," Ardupilot, 1 10 2023. [Online]. Available: https://ardupilot.org/copter/docs/common-apm-navigation-extended-kalman-filter-overview.html. [Accessed 1 10 2023].

[65] Ardupilot, "Mavros Documentation," Ardupilot, 1 10 2023. [Online]. Available: https://ardupilot.org/dev/docs/ros.html#:~:text=MAVROS%20is%20a%20ROS%20package,vehicles%20to%20communicate%20with%20ROS.. [Accessed 1 10 2023].

[66] qaz9517532846_ZM_robot, "ZM Robot," 1 10 2023. [Online]. Available: https://github.com/qaz9517532846/zm_robot. [Accessed 1 10 2023].

Appendix A: Launch File for Helix Simulation

```
<launch>

  <!-- these are the arguments you can pass this launch file, for example
paused:=true -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

    <!-- Set the robot_description parameter with the URDF file -->
  <param name="robot_description" textfile="$(find
helixfinal)/urdf/helixfinal.urdf"/>

  <!-- We resume the logic in empty_world.launch, changing only the name of
the world to be launched -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">

    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
    </include>

  <node
    name="tf_footprint_base"
    pkg="tf"
    type="static_transform_publisher"
    args="0 0 0 0 0 0 base_link base_footprint 40" />
  <node
    name="spawn_model"
    pkg="gazebo_ros"
    type="spawn_model"
    args="-file $(find helixfinal)/urdf/helixfinal.urdf -urdf -model
helixfinal"
    output="screen" />
  <node
    name="fake_joint_calibration"
    pkg="rostopic"
    type="rostopic"
    args="pub /calibrated std_msgs/Bool true" />
  <!-- send fake joint values -->
  <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher"/>
</launch>
```

Appendix B: URDF File for Helix Simulation

```
<?xml version="1.0" encoding="utf-8"?>

<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="helixfinal">

  <!-- Used for fixing robot to Gazebo 'base_link' -->
  <link name="base_footprint"/>

  <joint name="base_joint" type="fixed">
    <parent link="base_footprint"/>
    <child link="base_link"/>
    <origin xyz="0 0 0.5" rpy="0 0 0"/>
  </joint>

  <link
    name="base_link">
    <inertial>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <mass
        value="48.194" />
      <inertia
        ixx="1.4866"
        ixy="0"
        ixz="0"
        iyy="2.587"
        iyz="0"
        izz="5.053" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://helixfinal/meshes/base_link.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.79216 0.81961 0.93333 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://helixfinal/meshes/base_link.STL" />
      </geometry>
    </collision>
  </link>
  <link
    name="frontleft">
    <inertial>
      <origin
        xyz="0.001 0 0"
        rpy="0 0 0" />
      <mass
        value="17.19" />
      <inertia
        ixx="0.62344"
        ixy="0"
        ixz="0"
        iyy="0.62344"
        iyz="0"
        izz="1.0424" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://helixfinal/meshes/backleft.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.79216 0.81961 0.93333 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://helixfinal/meshes/backleft.STL" />
      </geometry>
    </collision>
  </link>
  <joint
    name="jointLF"
    type="continuous">
    <origin
      xyz="0.25396 0.01 -0.3429"
      rpy="0.22071 0 1.5734" />
    <parent
      link="base_link" />
    <child
      link="frontleft" />
    <axis
      xyz="1 0 0" />

    <dynamics damping="25.0" friction="30.0"/>
  </joint>
  <link
    name="backleft">
    <inertial>
      <origin
```

```
        xyz="0.001 -0 0"
        rpy="0 0 0" />
      <mass
        value="17.19" />
      <inertia
        ixx="0.6234"
        ixy="0"
        ixz="0"
        iyy="0.6234"
        iyz="0"
        izz="1.0424" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://helixfinal/meshes/backleft.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.79216 0.81961 0.93333 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://helixfinal/meshes/backleft.STL" />
      </geometry>
    </collision>
  </link>
  <joint
    name="jointLB"
    type="continuous">
    <origin
      xyz="-0.25409 -0.0165 -0.3429"
      rpy="0.22071 0.0 -1.5734" />
    <parent
      link="base_link" />
    <child
      link="backleft" />
    <axis
      xyz="-1 0 0" />

    <dynamics damping="25.0" friction="30.0"/>
  </joint>
  <link
    name="backright">
    <inertial>
      <origin
        xyz="0.001 0 3.6946E-07"
        rpy="0 0 0" />
      <mass
        value="17.19" />
      <inertia
        ixx="0.62344"
        ixy="0"
        ixz="0"
        iyy="0.6234"
        iyz="0"
        izz="1.0424" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://helixfinal/meshes/backright.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.79216 0.81961 0.93333 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://helixfinal/meshes/backright.STL" />
      </geometry>
    </collision>
  </link>
  <joint
    name="jointRB"
    type="continuous">
    <origin
      xyz="-0.25391 0.01 -0.3429"
      rpy="1.864 0 -1.5734" />
    <parent
      link="base_link" />
    <child
      link="backright" />
    <axis
      xyz="-1 0 0" />

    <dynamics damping="25.0" friction="30.0"/>
  </joint>
  <link
    name="frontright">
    <inertial>
      <origin
        xyz="0 0 4.6229E-07"
        rpy="0 0 0" />
      <mass
        value="17.19" />
      <inertia
        ixx="0.62344"
        ixy="0"
        ixz="0"
        iyy="0.6234"
        iyz="0"
        izz="1.0424" />
    </inertial>
    <visual>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://helixfinal/meshes/backright.STL" />
      </geometry>
      <material
        name="">
        <color
          rgba="0.79216 0.81961 0.93333 1" />
      </material>
    </visual>
    <collision>
      <origin
        xyz="0 0 0"
        rpy="0 0 0" />
      <geometry>
        <mesh
          filename="package://helixfinal/meshes/backright.STL" />
      </geometry>
    </collision>
  </link>
  <joint
    name="jointRF"
    type="continuous">
    <origin
      xyz="0.25404 -0.019 -0.3429"
      rpy="0.67383 0 1.5734" />
    <parent
      link="base_link" />
    <child
      link="frontright" />
    <axis
      xyz="1 0 0" />

    <dynamics damping="25.0" friction="30.0"/>
  </joint>
    <!-- Add the mecanum drive plugin -->

    <!-- Import Plugin -->


  <gazebo>
    <plugin name="joint_state_publisher"
filename="libgazebo_ros_joint_state_publisher.so">
      <robotNamespace>helixfinal</robotNamespace>
    </plugin>
    <plugin name="mecanum_drive_controller"
filename="libgazebo_ros_mecanum_drive.so">
      <legacyMode>false</legacyMode>
      <alwaysOn>true</alwaysOn>
      <odometryRate>20.0</odometryRate>
      <updateRate>1000.0</updateRate>


      <LeftRear>jointRB</LeftRear>
      <LeftFront>jointLF</LeftFront>
      <RightFront>jointRF</RightFront>
      <RightRear>jointLB</RightRear>
      <WheelSeparationL>0.565</WheelSeparationL>
      <WheelSeparationW>0.565</WheelSeparationW>
      <wheelRadius>0.175</wheelRadius>
      <wheelDiameter>0.35</wheelDiameter>
      <wheelAccel>5</wheelAccel>
      <WheelTorque>30.0</WheelTorque>
      <commandTopic>cmd_vel</commandTopic>
      <publishOdom>1</publishOdom>
      <publishWheelJointState>1</publishWheelJointState>
      <publishWheelTF>1</publishWheelTF>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <robotBaseFrame>base_link</robotBaseFrame>
      <isRollerModel>true</isRollerModel>
    </plugin>
  </gazebo>
  <!-- base_link -->
  <gazebo reference="base_link">
    <selfCollide>false</selfCollide>
    <gravity>true</gravity>
    <mu1>10</mu1>
    <mu2>10</mu2>
    <kp>1e30</kp>
    <kd>1e20</kd>
    <material>Gazebo/Orange</material>
  </gazebo>
</robot>
```

Appendix C: Launch File for Helix Controller

```
<launch>

  <!-- Launch data_collection_csv node -->
  <node name="data_collection_csv" pkg="pose_estimation"
type="logging_1_pose.py" output="screen"/>

  <!-- Launch data_collection_bag node -->
  <node name="data_collection_bag" pkg="pose_estimation"
type="logging_2_pose.py" output="screen"/>

  <!-- Launch main_node -->
  <node name="robot_move" pkg="pose_estimation"
type="Helix_way_point_1.py" output="screen"/>


</launch>
```

Appendix D: Python Node – logging_1_pose.py

```
#!/usr/bin/env python

import rospy
import pandas as pd
import matplotlib.pyplot as plt
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
from sensor_msgs.msg import Imu, NavSatFix
from nav_msgs.msg import Odometry
from geometry_msgs.msg import PoseStamped, TwistStamped
from tf.transformations import euler_from_quaternion
import numpy as np
from std_msgs.msg import Float32, Bool
import os
import signal
import sys
from datetime import datetime
import math


# Get the current date and time
current_time = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

# Create the folder path
folder_path = '/home/helix/ros_ws/src/pose_estimation/data/Plots_data_' +
current_time

# Create the folder
os.makedirs(folder_path)


# Initialize data arrays
time_data = []
local_odom_x_data = []
local_odom_y_data = []
local_odom_yaw_data = []
body_odom_x_data = []
body_odom_y_data = []
body_odom_yaw_data = []
yaw_imu_data = []
global_odom_x_data = []
global_odom_y_data = []
global_odom_yaw_data = []
twist_linear_x_data = []
twist_linear_y_data = []
twist_angular_z_data = []
wheel1_rpm_data = []
wheel2_rpm_data = []
wheel3_rpm_data = []
wheel4_rpm_data = []
wheel1_power_data = []
wheel2_power_data = []
wheel3_power_data = []
wheel4_power_data = []
distance_error = []
```

```
yaw_error = []
lat = []
lon = []
alt = []

global
goal_reached,vx,vy,vw,wheel_rpm,x_l,y_l,x_g,y_g,x_b,y_b,yaw_b,yaw_g,yaw_l
,yaw_imu,d_error,wheel_power,y_error,latitude,longitude,altitude

goal_reached = False
x_g = 0
y_g = 0
yaw_g = 0
x_l = 0
y_l = 0
yaw_l = 0
yaw_imu = 0
x_b = 0
y_b = 0
yaw_b = 0
vx = 0
vy = 0
vw = 0
wheel_rpm = np.zeros(4)
wheel_power = np.zeros(4)
d_error = 0
y_error = 0
latitude = 0.0
longitude = 0.0
altitude = 0.0


# Define callback functions
def odom_callback(data):
    global x,y,yaw
    x = data.pose.pose.position.x
    y = data.pose.pose.position.y
    orientation = data.pose.pose.orientation
    (_, _, yaw) = euler_from_quaternion([orientation.x, orientation.y,
orientation.z, orientation.w])



def twist_to_wheel_rpm(twist):
    global goal_reached,vx,vy,vw,wheel_rpm,wheel_power
    # define the mecanum wheel properties
    wheel_radius = 0.175  # in meters 0.35 m diameter
    wheel_sep_x = 0.508   # in meters axle to axle
    wheel_sep_y = 0.565   # in meters left to right

    # calculate the wheel speeds from twist message
    vx = twist.linear.x
    vy = twist.linear.y
    vw = twist.angular.z


    wheel_rpm = np.zeros(4)
    wheel_rpm[0] = (1/wheel_radius) * (vx + vy + (wheel_sep_x +
wheel_sep_y)*vw)
    wheel_rpm[1] = (1/wheel_radius) * (vx - vy + (wheel_sep_x +
wheel_sep_y)*vw)
    wheel_rpm[2] = (1/wheel_radius) * (-vx - vy + (wheel_sep_x +
wheel_sep_y)*vw)
    wheel_rpm[3] = (1/wheel_radius) * (-vx + vy + (wheel_sep_x +
wheel_sep_y)*vw)

    wheel_power = np.zeros(4)
    wheel_power[0] = (1000.0 /195.0) * wheel_rpm[0]
    wheel_power[1] = (1000.0 /198.0) * wheel_rpm[1]
    wheel_power[2] = (1000.0 /195.0) * wheel_rpm[2]
    wheel_power[3] = (1000.0 /198.0) * wheel_rpm[3]

    if abs(wheel_power[0]) >= 900:
        wheel_power[0] = wheel_power[0]/abs(wheel_power[0])*900
    if abs(wheel_power[1]) >= 900:
        wheel_power[1] = wheel_power[1]/abs(wheel_power[1])*900
    if abs(wheel_power[2]) >= 900:
        wheel_power[2] = wheel_power[2]/abs(wheel_power[2])*900
    if abs(wheel_power[3]) >= 900:
        wheel_power[3] = wheel_power[3]/abs(wheel_power[3])*900

    return wheel_rpm,wheel_power

def distance_error_callback(msg):
    global d_error

    d_error = msg.data

def yaw_error_callback(msg):
    global y_error

    y_error = msg.data

def get_gps(msg):

    global latitude,longitude,altitude
    latitude = msg.latitude
    longitude = msg.longitude
    altitude = msg.altitude



def goal_callback(msg):
    global goal_reached
    goal_reached  = msg.data

def get_rotation(msg):
    global yaw_imu
    orientation_q = msg.orientation
    orientation_list = [orientation_q.x, orientation_q.y,
orientation_q.z, orientation_q.w]
    (_, _, yaw_imu) = euler_from_quaternion(orientation_list)
    # print('yaw_imu = ', yaw_imu * 180.0 / 3.142)
```

```python
def get_localpose(msg):

    global x_l,y_l,yaw_l,x_b,y_b

    position = msg.pose.position
    x_l = position.x
    y_l = position.y

    orientation_q = msg.pose.orientation
    orientation_list = [orientation_q.x, orientation_q.y,
orientation_q.z, orientation_q.w]
    (roll, pitch, yaw_l) = euler_from_quaternion(orientation_list)
    # print('local_pose =', x,y, 'yaw =', yaw_l * 180.0 / 3.142)

def get_localodom(msg):

    global x_l,y_l,yaw_l

    position = msg.pose.pose.position
    x_l = position.x
    y_l = position.y


    orientation_q = msg.pose.pose.orientation
    orientation_list = [orientation_q.x, orientation_q.y,
orientation_q.z, orientation_q.w]
    (roll, pitch, yaw_l) = euler_from_quaternion(orientation_list)

    # print('l_odom =', x_l,y_l, 'yaw =', yaw_l * 180.0 / 3.142)
    # print('l_odom =', x,y,z)

def get_globalodom(msg):

    global x_g,y_g,yaw_g

    position = msg.pose.pose.position
    x_g = position.x
    y_g = position.y
    z_g = position.z


    orientation_q = msg.pose.pose.orientation
    orientation_list = [orientation_q.x, orientation_q.y,
orientation_q.z, orientation_q.w]
    (roll, pitch, yaw_g) = euler_from_quaternion(orientation_list)

    # print('g_odom =', x,y,z, 'yaw =', yaw * 180.0 / 3.142)
    # print('g_odom =', x,y,z)


# Main loop
def listener():

    # Initialize ROS node
    rospy.init_node('data_collection_csv')

    # Initialize time variables
    start_time = rospy.Time.now()
    prev_time = start_time

    # Define loop rate
    loop_rate = rospy.Rate(10)  # 10 Hz

    # Initialize publishers and subscribers
    cmd_vel_sub = rospy.Subscriber('/cmd_vel', Twist,
twist_to_wheel_rpm,queue_size=10)
    odom_sub = rospy.Subscriber('/odom', Odometry,
odom_callback,queue_size=10)
    error_sub = rospy.Subscriber('/distance_error', Float32,
distance_error_callback,queue_size=10)
    yaw_error_sub = rospy.Subscriber('/yaw_error', Float32,
yaw_error_callback,queue_size=10)
    goal_sub = rospy.Subscriber('/goal_reached', Bool, goal_callback,
queue_size = 10)
    imu_sub = rospy.Subscriber('/mavros2/imu/data', Imu, get_rotation,
queue_size = 10)
    local_pose_sub = rospy.Subscriber('/mavros2/local_position/pose',
PoseStamped, get_localpose, queue_size = 10)
    local_odom_sub = rospy.Subscriber('/mavros2/local_position/odom',
Odometry, get_localodom, queue_size = 10)
    global_odom_sub = rospy.Subscriber('/mavros2/global_position/local',
Odometry, get_globalodom, queue_size = 10)
    gps_sub = rospy.Subscriber('/mavros2/global_position/raw/fix',
NavSatFix,get_gps)

    i = 0
    while not rospy.is_shutdown() and not goal_reached:
        # Get current time
        current_time = rospy.Time.now()

        # Compute time elapsed since last loop iteration
        dt = (current_time - prev_time).to_sec()


        # Record Position message data
        local_odom_x_data.append(x_l)
        local_odom_y_data.append(y_l)
        local_odom_yaw_data.append(yaw_l)

        global_odom_x_data.append(x_g)
        global_odom_y_data.append(y_g)
        global_odom_yaw_data.append(yaw_g)

        yaw_imu_data.append(yaw_imu)

        if i <= 10:
            x_b = -(local_odom_x_data[i] - x_l)
            y_b = -(local_odom_y_data[i] - y_l)
            yaw_b = yaw_imu - yaw_imu_data[i]
            # print (yaw_imu_data[i],yaw_imu, yaw_b)
            if(math.fabs(yaw_b) > math.pi):
                yaw_b = yaw_b - (2 * math.pi * yaw_b) /
(math.fabs(yaw_b))
        else:
            x_b = -(local_odom_x_data[6] - x_l)
            y_b = -(local_odom_y_data[6] - y_l)
            yaw_b = yaw_imu - yaw_imu_data[6]
            if(math.fabs(yaw_b) > math.pi):
                yaw_b = yaw_b - (2 * math.pi * yaw_b) /
(math.fabs(yaw_b))
            # print (yaw_imu_data[6],yaw_imu, yaw_b)



        body_odom_x_data.append(x_b)
        body_odom_y_data.append(y_b)
        body_odom_yaw_data.append(yaw_b)

        # Record twist message data
        twist_linear_x_data.append(vx)
        twist_linear_y_data.append(vy)
        twist_angular_z_data.append(vw)

        wheel1_rpm_data.append(wheel_rpm[0])
        wheel2_rpm_data.append(wheel_rpm[1])
        wheel3_rpm_data.append(wheel_rpm[2])
        wheel4_rpm_data.append(wheel_rpm[3])

        wheel1_power_data.append(wheel_power[0])
        wheel2_power_data.append(wheel_power[1])
        wheel3_power_data.append(wheel_power[2])
        wheel4_power_data.append(wheel_power[3])



        # Record Error message data
        distance_error.append(d_error)

        yaw_error.append(y_error)

        lat.append(latitude)
        lon.append(longitude)
        alt.append(altitude)

        # Wait for next loop iteration
        loop_rate.sleep()

        # Record current time
        time_data.append(current_time.to_sec() - start_time.to_sec())

        i=i+1



        # Update previous time
        prev_time = current_time

        # Register the signal handler for the interrupt signal (Ctrl+C)
        signal.signal(signal.SIGINT, signal_handler)
```

```
# Define the signal handler function
def signal_handler(signal, frame):
    global goal_reached
    print("Ctrl+C pressed. Stopping...")
    goal_reached = True


if __name__ == '__main__':
    try:
        listener()
    except rospy.ROSInterruptException:
        pass

# Convert data to pandas DataFrame

# print (distance_error)

time_data[0] = time_data[1]
local_odom_x_data[0] = local_odom_x_data[1]
local_odom_y_data[0] = local_odom_y_data[1]
local_odom_yaw_data[0] = local_odom_yaw_data[1]
global_odom_x_data[0] = global_odom_x_data[1]
global_odom_y_data[0] = global_odom_y_data[1]
global_odom_yaw_data[0] = global_odom_yaw_data[1]
body_odom_x_data[0] = body_odom_x_data[1]
body_odom_y_data[0] = body_odom_y_data[1]
body_odom_yaw_data[0] = body_odom_yaw_data[1]
yaw_imu_data[0] = yaw_imu_data[1]
twist_linear_x_data[0] = twist_linear_x_data[1]
twist_linear_y_data[0] = twist_linear_y_data[1]
twist_angular_z_data[0] = twist_angular_z_data[1]
wheel1_rpm_data[0] = wheel1_rpm_data[1]
wheel2_rpm_data[0] = wheel2_rpm_data[1]
wheel3_rpm_data[0] = wheel3_rpm_data[1]
wheel4_rpm_data[0] = wheel4_rpm_data[1]
wheel1_power_data[0] = wheel1_power_data[1]
wheel2_power_data[0] = wheel2_power_data[1]
wheel3_power_data[0] = wheel3_power_data[1]
wheel4_power_data[0] = wheel4_power_data[1]
distance_error[0] = distance_error[1]
yaw_error[0] = yaw_error[1]
lat[0] = lat[1]
lon[0] = lon[1]
alt[0] = alt[1]

# print
(np.size(time_data),np.size(odom_x_data),np.size(odom_y_data),np.size(odo
m_yaw_data),np.size(twist_linear_x_data),np.size(twist_linear_y_data),np.
size(twist_angular_z_data),np.size(wheel1_rpm_data),np.size(wheel2_rpm_da
ta),np.size(wheel3_rpm_data),np.size(wheel4_rpm_data))
data = pd.DataFrame({
    'time': time_data,
    'local_odom_x': local_odom_x_data,
    'local_odom_y': local_odom_y_data,
    'local_odom_yaw': local_odom_yaw_data,
    'global_odom_x': global_odom_x_data,
    'global_odom_y': global_odom_y_data,
    'global_odom_yaw': global_odom_yaw_data,
    'body_odom_x': body_odom_x_data,
    'body_odom_y': body_odom_y_data,
    'body_odom_yaw': body_odom_yaw_data,
    'yaw_imu' : yaw_imu_data,
    'twist_linear_x': twist_linear_x_data,
    'twist_linear_y': twist_linear_y_data,
    'twist_angular_z': twist_angular_z_data,
    'Wheel_1_rpm': wheel1_rpm_data,
    'Wheel_2_rpm': wheel2_rpm_data,
    'Wheel_3_rpm': wheel3_rpm_data,
    'Wheel_4_rpm': wheel4_rpm_data,
    'Wheel_1_power': wheel1_power_data,
    'Wheel_2_power': wheel2_power_data,
    'Wheel_3_power': wheel3_power_data,
    'Wheel_4_power': wheel4_power_data,
    'distance_error': distance_error,
    'yaw_error': yaw_error,
    'latitude': lat,
    'longitude' : lon,
    'altitude' : alt,
})

# Save data to CSV file
csv_file = os.path.join(folder_path, 'data.csv')
data.to_csv(csv_file, index=False)

# Plot charts
# Plot x position
plt.figure()
plt.plot(data['time'], data['local_odom_x'], label = 'local_x')
# plt.plot(data['time'], data['global_odom_x'], label = 'global_x')
plt.plot(data['time'], data['body_odom_x'], label = 'body_x')
plt.xlabel('Time (s)')
plt.ylabel('X position (m)')
plt.title('X position over time')
plt.grid()
plt.legend()
plt.savefig(os.path.join(folder_path, 'x_position.png'), dpi=300)

# Plot y position
fig = plt.figure()
plt.plot(data['time'], data['local_odom_y'], label = 'local_y')
# plt.plot(data['time'], data['global_odom_y'], label = 'global_y')
plt.plot(data['time'], data['body_odom_y'], label = 'body_y')
plt.xlabel('Time (s)')
plt.ylabel('Y position (m)')
plt.title('Y position over time')
plt.grid()
plt.legend()
plt.savefig(os.path.join(folder_path, 'y_position.png'), dpi=300)

# Plot orientation
fig = plt.figure()
plt.plot(data['time'], data['local_odom_yaw'],label = 'local_yaw')
# plt.plot(data['time'], data['global_odom_yaw'], label = 'global_yaw')
plt.plot(data['time'], data['body_odom_yaw'], label = 'body_yaw')
plt.plot(data['time'], data['yaw_imu'], label = 'imu_yaw')
plt.xlabel('Time (s)')
plt.ylabel('Orientation (rad)')
plt.title('Yaw position over time')
plt.grid()
plt.legend()
plt.savefig(os.path.join(folder_path, 'yaw_position.png'), dpi=300)

# Plot x velocity
fig = plt.figure()
plt.plot(data['time'], data['twist_linear_x'])
plt.xlabel("Time (s)")
plt.ylabel("linear Velocity (m/s)")
plt.title('X velocity')
plt.grid()
plt.legend()
plt.savefig(os.path.join(folder_path, 'x_velocity.png'), dpi=300)

# Plot y velocity
fig = plt.figure()
plt.plot(data['time'], data['twist_linear_y'])
plt.xlabel("Time (s)")
plt.ylabel("linear Velocity (m/s)")
plt.title('Y velocity')
plt.grid()
plt.legend()
plt.savefig(os.path.join(folder_path, 'y_velocity.png'), dpi=300)

# Plot angular velocity
fig = plt.figure()
plt.plot(data['time'], data['twist_angular_z'])
plt.xlabel("Time (s)")
plt.ylabel("Angular Velocity (rad/s)")
plt.title('Angular velocity')
plt.grid()
plt.legend()
plt.savefig(os.path.join(folder_path, 'z_velocity.png'), dpi=300)

# Plot trajectory
fig = plt.figure()
plt.plot(data['body_odom_x'], data['body_odom_y'], label = 'Robot
Position')
plt.xlabel("x position (m)")
plt.ylabel("y position (m)")
plt.title("Trajectory")
plt.grid()
plt.legend()
plt.savefig(os.path.join(folder_path, 'trajectory.png'), dpi=300)

# Plot Distance error
fig = plt.figure()
plt.plot(data['time'], data['distance_error'], label = 'Distance Error')
plt.xlabel("Time (s)")
plt.ylabel("Error (m)")
plt.title("Distance Error")
plt.grid()
plt.legend()
plt.savefig(os.path.join(folder_path, 'distance_error.png'), dpi=300)

# Plot yaw error
fig = plt.figure()
plt.plot(data['time'], data['yaw_error'], label = 'Heading Error')
plt.xlabel("Time (s)")
plt.ylabel("Error (m)")
plt.title("Heading Error")
plt.grid()
plt.legend()
plt.savefig(os.path.join(folder_path, 'heading_error.png'), dpi=300)

# plot wheel rpm data
fig = plt.figure()
plt.plot(data['time'], data['Wheel_1_rpm'])
plt.plot(data['time'], data['Wheel_2_rpm'])
plt.plot(data['time'], data['Wheel_3_rpm'])
plt.plot(data['time'], data['Wheel_4_rpm'])
plt.legend()
plt.title('Wheel RPM Data')
plt.xlabel('Time (s)')
plt.ylabel('RPM')
plt.grid()

# Save the figure
plt.savefig(os.path.join(folder_path, 'wheel_rpm_data.png'), dpi=300)

# plot wheel power data
fig = plt.figure()
plt.plot(data['time'], data['Wheel_1_power'])
plt.plot(data['time'], data['Wheel_2_power'])
plt.plot(data['time'], data['Wheel_3_power'])
plt.plot(data['time'], data['Wheel_4_power'])
plt.legend()
plt.title('Wheel Power Data')
plt.xlabel('Time (s)')
plt.ylabel('Power')
plt.grid()

# Save the figure
plt.savefig(os.path.join(folder_path, 'wheel_power_data.png'), dpi=300)


# Create subplots
fig, axes = plt.subplots(2, 2, figsize=(10, 8), sharex=True)

# Plot wheel RPM data in subplots
axes[0, 0].plot(data['time'], data['Wheel_1_rpm'])
axes[0, 0].grid(True)
axes[0, 1].plot(data['time'], data['Wheel_2_rpm'])
axes[0, 1].grid(True)
axes[1, 0].plot(data['time'], data['Wheel_3_rpm'])
axes[1, 0].grid(True)
axes[1, 1].plot(data['time'], data['Wheel_4_rpm'])
axes[1, 1].grid(True)

# Set labels and titles for each subplot
```

```
axes[0, 0].set_ylabel('Wheel 1 RPM')
axes[0, 1].set_ylabel('Wheel 2 RPM')
axes[1, 0].set_ylabel('Wheel 3 RPM')
axes[1, 1].set_ylabel('Wheel 4 RPM')
axes[1, 0].set_xlabel('Time (s)')
axes[1, 1].set_xlabel('Time (s)')
axes[0, 0].set_xlabel('Time (s)')
axes[0, 1].set_xlabel('Time (s)')

# Set overall title and grid
fig.suptitle('Wheel RPM Data')

# Adjust subplot spacing
plt.subplots_adjust(hspace=0.4, wspace=0.3)

# Save the figure
plt.savefig(os.path.join(folder_path, 'wheel_rpm_sub_plot_data.png'),
dpi=300)


# Create subplots
fig, axes = plt.subplots(2, 2, figsize=(10, 8), sharex=True)

# Plot wheel RPM data in subplots
axes[0, 0].plot(data['time'], data['Wheel_1_power'])
axes[0, 0].grid(True)
axes[0, 1].plot(data['time'], data['Wheel_2_power'])
axes[0, 1].grid(True)
axes[1, 0].plot(data['time'], data['Wheel_3_power'])
axes[1, 0].grid(True)
axes[1, 1].plot(data['time'], data['Wheel_4_power'])
axes[1, 1].grid(True)

# Set labels and titles for each subplot
axes[0, 0].set_ylabel('Wheel 1 Power')
axes[0, 1].set_ylabel('Wheel 2 Power')
axes[1, 0].set_ylabel('Wheel 3 Power')
axes[1, 1].set_ylabel('Wheel 4 Power')
axes[1, 0].set_xlabel('Time (s)')
axes[1, 1].set_xlabel('Time (s)')
axes[0, 0].set_xlabel('Time (s)')
axes[0, 1].set_xlabel('Time (s)')

# Set overall title and grid
fig.suptitle('Wheel Power Data')

# Adjust subplot spacing
plt.subplots_adjust(hspace=0.4, wspace=0.3)


# Save the figure
plt.savefig(os.path.join(folder_path, 'wheel_power_sub_plot_data.png'),
dpi=300)
```

Appendix E: Python Node – logging_2_pose.py

```
#!/usr/bin/env python
import rospy
import rosbag
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
from std_msgs.msg import Float32, Bool
from sensor_msgs.msg import Imu, NavSatFix
import signal
import sys
import os
from datetime import datetime
from cst_msgs.msg import RawMotorCommand, MotorFeedback

# Initialize the bag variable
bag = None

# Get the current date and time
current_time = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

# Create the folder path
folder_path = '/home/helix/ros_ws/src/pose_estimation/data/Bag_' +
current_time

# Create the folder
os.makedirs(folder_path)

# Define callback functions
def cmd_vel_callback(data):
    bag.write('/cmd_vel', data)

def odom_callback(data):
    bag.write('/odom', data)

def distance_error_callback(data):
    bag.write('/distance_error', data)

def yaw_error_callback(data):
    bag.write('/yaw_error', data)

def imu_callback(data):
    bag.write('/mavros2/imu/data', data)

def gps_callback(data):
    bag.write('/mavros2/global_position/raw/fix', data)

def localodom_callback(data):
    bag.write('/mavros2/odometry/in', data)

def globalodom_callback(data):
    bag.write('/mavros2/global_position/local', data)

def goal_callback(data):
    bag.write('/goal_reached', data)

def motor_feedback_callback(data):
    bag.write('/helix/motor_manager/feedback', data)

def motor_command_callback(data):
    bag.write('/helix/motor_manager/raw_command', data)

# Main loop
def listener():
    global bag
    # Initialize ROS node
    rospy.init_node('data_collection_bag')

    # Create a ROS bag file
    bag_file = os.path.join(folder_path, 'data.bag')
    bag = rosbag.Bag(bag_file, 'w')

    # Initialize subscribers
    cmd_vel_sub = rospy.Subscriber('/cmd_vel', Twist, cmd_vel_callback)
    odom_sub = rospy.Subscriber('/odom', Odometry, odom_callback)
    distance_error_sub = rospy.Subscriber('/distance_error', Float32,
distance_error_callback)
    yaw_error_sub = rospy.Subscriber('/yaw_error', Float32,
yaw_error_callback)
    goal_sub = rospy.Subscriber('/goal_reached', Bool, goal_callback)
    imu_sub = rospy.Subscriber('/mavros2/imu/data', Imu, imu_callback)
    #state_sub = rospy.Subscriber('/mavros/state', State, state_callback)
    local_odom_sub = rospy.Subscriber('/mavros2/local_position/odom',
Odometry, localodom_callback)
    global_odom_sub = rospy.Subscriber('/mavros2/global_position/local',
Odometry, globalodom_callback)
    gps_sub = rospy.Subscriber('/mavros2/global_position/raw/fix',
NavSatFix,gps_callback)
    motor_feedback_sub =
rospy.Subscriber('/helix/motor_manager/feedback',
MotorFeedback,motor_feedback_callback)
    motor_command_sub =
rospy.Subscriber('/helix/motor_manager/raw_command',
RawMotorCommand,motor_command_callback)


    # Spin ROS node
    rospy.spin()
    # Close the bag file
    bag.close()

# Define the signal handler function
def signal_handler(sig, frame):
    rospy.sleep(2)
    # Close the bag file
    bag.close()
    rospy.loginfo("Bag file closed.")
    sys.exit(0)

if __name__ == '__main__':
    try:
        listener()
    except rospy.ROSInterruptException:
        pass
```

Appendix F: Python Node – Helix_way_point_1.py

```
import rospy
from geometry_msgs.msg import Twist, Point, PoseStamped
from nav_msgs.msg import Odometry
from tf.transformations import euler_from_quaternion
#from gazebo_mecanum_plugins.msg import gazebo_mecanum_plugins_vel
import math
from std_msgs.msg import Float32, Bool
import numpy as np
import matplotlib.pyplot as plt
from mavros_msgs.srv import CommandLong
from sensor_msgs.msg import Imu, NavSatFix
from cst_msgs.msg import RawMotorCommand

#WAYPOINTS = [[0,5],[5,5],[5,0],[0,0]]

WAYPOINTS = [[4,0.0],[4,-4],[0,-4],[0,0]]

class ZmRobotController:
    def __init__(self):
        rospy.init_node('zm_robot_controller')

        # Initialize the publisher for the Twist commands
        self.twist_pub = rospy.Publisher('/cmd_vel', Twist,
queue_size=10)
        self.motor_pub =
rospy.Publisher('/helix/motor_manager/raw_command', RawMotorCommand,
queue_size=10)

        self.distance_error_pub = rospy.Publisher('/distance_error',
Float32, queue_size=10)
        self.yaw_error_pub = rospy.Publisher('/yaw_error', Float32,
queue_size=10)

        # Initialize the subscriber for the odometry messages
        # rospy.Subscriber('/odom', Odometry, self.odom_callback)
        # rospy.Subscriber('/mavros/local_position/odom', Odometry,
self.odom_callback, queue_size = 10)
        rospy.Subscriber('/mavros2/local_position/odom', Odometry,
self.odom_callback)
        imu_sub = rospy.Subscriber('/mavros2/imu/data', Imu,
self.get_rotation, queue_size = 10)

        # Subscribe to the mecanum wheel velocities topic to get the
current wheel velocities
        rospy.Subscriber('/cmd_vel', Twist, self.vel_callback)

        self.goal_reached_pub = rospy.Publisher('/goal_reached', Bool
,queue_size=10)

        # Initialize the goal point
        self.goal = Point()
        self.goal.x = WAYPOINTS[0][0]
        self.goal.y = WAYPOINTS[0][1]
        self.goal.z = 0
        self.max_vel = 10

        self.trajectory = list()

        self.counter = 0
        self. i = 0
        self.x_temp = 0
        self.y_temp = 0
        self.yaw_temp = 0

        # Initialize the PID parameters
        self.kp = 1.5
        self.ki = 0.0002
        self.kd = 0.05
        self.kp_z = 8
        self.ki_z = 0.0
        self.kd_z = 0.05
        self.integral = 0
        self.prev_error = 0

        self.wheel_radius = 0.175  # in meters 0.35 m diameter
        self.wheel_sep_x = 0.508   # in meters axle to axle
        self.wheel_sep_y = 0.565   # in meters left to right

        self.yaw_imu = 0

        # Set the control rate
        self.rate = rospy.Rate(100) # 50 Hz

        # Initialize the flag for indicating whether the goal has been
reached
        self.goal_reached = False

    def set_rover_mode(self,mode):
        rospy.wait_for_service('/mavros/cmd/command')
        try:
            command_long = rospy.ServiceProxy('/mavros/cmd/command',
CommandLong)
            command_long(False, 176, 0, 0.0, mode, 0.0, 0.0, 0.0, 0.0,
0.0)
        except rospy.ServiceException as e:
            print("Command call failed: %s"%e)

    def get_rotation(self,msg):

        orientation_q = msg.orientation
        orientation_list = [orientation_q.x, orientation_q.y,
orientation_q.z, orientation_q.w]
        (_, _, self.yaw_imu) = euler_from_quaternion(orientation_list)
        # print('yaw_imu = ', yaw * 180.0 / 3.142)



    def vel_callback(self, data):
        # Get the wheel velocities from the mecanum wheel velocities
message
        #fr_1, rr_2, rl_3, fl_4
        vx = data.linear.x
        vy = data.linear.y
        vz = data.angular.z

        fr = (1 / self.wheel_radius) * (vx + vy + (self.wheel_sep_x +
self.wheel_sep_y) * vz)
        rr = (1 / self.wheel_radius) * (vx - vy + (self.wheel_sep_x +
self.wheel_sep_y) * vz)
        rl = (1 / self.wheel_radius) * (-vx - vy + (self.wheel_sep_x +
self.wheel_sep_y) * vz)
        fl = (1 / self.wheel_radius) * (-vx + vy + (self.wheel_sep_x +
self.wheel_sep_y) * vz)

        fr_p = (1000.0 /195.0) * fr
        rr_p = (1000.0 /198.0) * rr
        rl_p = (1000.0 /195.0) * rl
        fl_p = (1000.0 /198.0) * fl

        if abs(fr_p) >= 900:
            fr_p = fr_p/abs(fr_p)*900
        if abs(rr_p) >= 900:
            rr_p = rr_p/abs(rr_p)*900
        if abs(rl_p) >= 900:
            rl_p = rl_p/abs(rl_p)*900
        if abs(fl_p) >= 900:
            fl_p = fl_p/abs(fl_p)*900

        # Publish the velocity commands to the robot

        # print (fr,rr,rl,fl,vx,vy,vz,fr_p,rr_p,rl_p,fl_p)

        return fr,rr,rl,fl,fr_p,rr_p,rl_p,fl_p

    def normalize_angle(angle):
        if(math.fabs(angle) > math.pi):
            angle = angle - (2 * math.pi * angle) / (math.fabs(angle))

        return angle

    def odom_callback(self, msg):
        # Extract the robot's current position and orientation from the
odometry message
        position = msg.pose.pose.position
        x_l = position.x
        y_l = position.y

        orientation_q = msg.pose.pose.orientation
        orientation_list = [orientation_q.x, orientation_q.y,
orientation_q.z, orientation_q.w]
        (roll, pitch, yaw_l) = euler_from_quaternion(orientation_list)

        self.i = self.i+1
        # print(self.i)

        if self.i <= 10:
            self.x_temp = x_l
            self.y_temp = y_l
            self.yaw_temp = self.yaw_imu
            return
        else:
            x = -(self.x_temp - x_l)
            y = -(self.y_temp - y_l)
            yaw = self.yaw_imu - self.yaw_temp
            if(math.fabs(yaw) > math.pi):
                yaw = yaw - (2 * math.pi * yaw) / (math.fabs(yaw))

            # print(self.x_temp,self.y_temp,x_l,y_l)
            # print (yaw_imu_data[6],yaw_imu, yaw_b)
        # x = msg.pose.pose.position.x
        # y = msg.pose.pose.position.y
        # orientation = msg.pose.pose.orientation
        # (_, _, yaw) = euler_from_quaternion([orientation.x,
orientation.y, orientation.z, orientation.w])

        # Calculate the error between the robot's current position and
the goal position
        dx = self.goal.x - x
        dy = self.goal.y - y
        distance_error = math.sqrt(dx**2 + dy**2)
        self.distance_error_pub.publish(distance_error)


        # Check if the robot has reached the goal point
        if distance_error < 1.0 and not self.goal_reached:
            # Stop the robot
            twist = Twist()
            twist.linear.x = 0
            twist.linear.y = 0
            twist.angular.z = 0
            self.twist_pub.publish(twist)

            fr,rr,rl,fl,fr_p,rr_p,rl_p,fl_p= self.vel_callback(twist)

            self.motor_pub.publish([fr_p,rr_p,rl_p,fl_p])

            # Set the flag to indicate that the goal has been reached
            # track a sequence of waypoints

            self.counter = self.counter+1

            print("Action done.",self.counter, "th Way point reached")

            if self.counter == len(WAYPOINTS):
                self.goal_reached = True
                # Print a success message
                rospy.loginfo('Goal reached!')
                self.goal_reached_pub.publish(self.goal_reached)
            else:
                self.goal_reached = False
                self.set_goal(WAYPOINTS[self.counter][0],
WAYPOINTS[self.counter][1])
                rospy.sleep(1)
        else:
            # Calculate the heading angle to the goal
            heading_goal = math.atan2(dy, dx)


            # Calculate the heading error
            #heading_error = heading_goal - yaw


            # # Normalize the heading error to the range [-pi, pi]
            # if heading_error > math.pi:
            #     heading_error -= 2*math.pi
            # elif heading_error < -math.pi:
            #     heading_error += 2*math.pi

            heading_error = heading_goal - yaw

            if(math.fabs(heading_error) > math.pi):
                heading_error = heading_error - (2 * math.pi *
heading_error) / (math.fabs(heading_error))


            # print("a",heading_goal,heading_error,distance_error)

            self.yaw_error_pub.publish(heading_error)

            if math.fabs(heading_error) > 0.2:
                # If heading error is large, turn in place until it's
small
                twist = Twist()
                twist.linear.x = 0
                twist.linear.y = 0
                twist.angular.z = self.kp_z*heading_error
                if abs(heading_error) > self.max_vel:
                    twist.angular.z =
self.kp_z*(heading_error/abs(heading_error)*self.max_vel)
                self.twist_pub.publish(twist)
                fr,rr,rl,fl,fr_p,rr_p,rl_p,fl_p= self.vel_callback(twist)

                self.motor_pub.publish([fr_p,rr_p,rl_p,fl_p])
                self.trajectory.append([x,y])
                rospy.loginfo("odom: x=" + str(x) + ";  y=" + str(y) + ";
theta=" + str(yaw))
            else:
                # If heading error is small, move forward while
correcting heading error
                proportional = distance_error
                self.integral +=
distance_error/self.rate.sleep_dur.to_sec()
                derivative = (distance_error -
self.prev_error)/self.rate.sleep_dur.to_sec()
                pid = self.kp*proportional + self.ki*self.integral +
self.kd*derivative

                # Save the current error for use in the next iteration
                self.prev_error = distance_error
                # Create the Twist message
                twist = Twist()
                twist.linear.x = pid
                twist.linear.y = 0
                twist.angular.z = self.kp_z*heading_error

                if abs(pid) > self.max_vel:
                    twist.linear.x = pid/abs(pid)*self.max_vel
                if abs(heading_error) > self.max_vel:
                    twist.angular.z =
self.kp_z*(heading_error/abs(heading_error)*self.max_vel)

                # Publish the Twist message
                self.twist_pub.publish(twist)
                fr,rr,rl,fl,fr_p,rr_p,rl_p,fl_p= self.vel_callback(twist)

                self.motor_pub.publish([fr_p,rr_p,rl_p,fl_p])
                self.trajectory.append([x,y])
                rospy.loginfo("odom: x=" + str(x) + ";  y=" + str(y) + ";
theta=" + str(yaw))

    def set_goal(self, x, y):
        # Set a new goal point for the robot
        self.goal.x = x
        self.goal.y = y
        self.goal.z = 0

        # Reset the PID variables
        self.integral = 0
        self.prev_error = 0

        # Reset the goal reached flag
        self.goal_reached = False

    def run(self):
        # Run the main loop of the controller

        self.set_rover_mode(3)
        print("Rover armed")
        while not rospy.is_shutdown() and not self.goal_reached:
            self.rate.sleep()

        # Stop the robot
        twist = Twist()
        twist.linear.x = 0
        twist.linear.y = 0
        twist.angular.z = 0
        self.twist_pub.publish(twist)
        fr,rr,rl,fl,fr_p,rr_p,rl_p,fl_p = self.vel_callback(twist)

        self.motor_pub.publish([fr_p,rr_p,rl_p,fl_p])
        rospy.sleep(1)

        # Print a message indicating that the program is ending
        rospy.loginfo('Program ended.')

        self.goal_reached = True
        self.goal_reached_pub.publish(self.goal_reached)

        # plot trajectory
        data = np.array(self.trajectory)
        np.savetxt('trajectory.csv', data, fmt='%f', delimiter=',')
        plt.plot(data[:,0],data[:,1])
        plt.show()
        twist.linear.x = 0
        twist.linear.y = 0
        twist.angular.z = 0
        self.twist_pub.publish(twist)
        fr,rr,rl,fl,fr_p,rr_p,rl_p,fl_p= self.vel_callback(twist)

        self.motor_pub.publish([fr_p,rr_p,rl_p,fl_p])
        self.set_rover_mode(0)
        print("Rover disarmed")


if __name__ == '__main__':
    controller = ZmRobotController()
    controller.run()
```