Design and Implementation of a Vision Integrated Autonomous Research Rover

Visruth K
Department of Electronics and Communication Engineering
Excel Engineering College
Namakkal, Tamil Nadu, India
https://orcid.org/0009-0006-0952-8436

Saran M S
Department of Electronics and Communication Engineering
Excel Engineering College
Namakkal, Tamil Nadu, India
saransureshkmr@gmail.com

Tamil Selvan M
Department of Electronics and Communication Engineering
Excel Engineering College
Namakkal, Tamil Nadu, India
tamilselvanece22@gmail.com

Sevakumarr P R
Department of Electronics and Communication Engineering
Excel Engineering College
Namakkal, Tamil Nadu, India
sevakumarrpr@gmail.com

V Sakthivel
Department of Electronics and Communication Engineering
Excel Engineering College
Namakkal, Tamil Nadu, India
sakthivelv.eec@excelcolleges.com

***Abstract*—This paper presents the design and implementation of a low-cost, vision-integrated autonomous rover for automated soil data collection. Traditional soil sampling methods are labor-intensive, and existing robotic solutions frequently depend on costly hardware and Global Positioning System (GPS) navigation, which can easily degrade in obstructed environments. To address these issues, a GPS-denied rover was developed using a Raspberry Pi 4B for high-level computer vision and a Robot Operating System 2 (ROS2) framework, paired with an Arduino Nano for real-time hardware control. The platform utilizes a linear actuator to autonomously deploy sensing probes for localized soil analysis. For stable navigation, an Extended Kalman Filter (EKF) is implemented to fuse wheel encoder odometry with inertial data. An OpenCV-based vision pipeline directs autonomous path following and marker detection. To mitigate tracking errors caused by processing and mechanical latency, a custom delay-compensation filter is introduced to actively reverse the rover and correct overshoot. This is paired with a history-dependent sweep prioritization algorithm that intelligently re-acquires lost paths based on prior trajectory data. The resulting system provides a reliable, accessible framework for independent environmental monitoring.**

***Keywords*—Autonomous rover, ROS2, computer vision, Extended Kalman Filter, delay compensation, precision agriculture.**

1. INTRODUCTION

Collecting accurate soil data is a fundamental requirement for environmental research and localized field analysis. Manual soil sampling is notoriously slow and requires significant human effort to cover large areas. While autonomous robots can automate this process, commercial solutions are typically expensive and rely heavily on Global Positioning Systems (GPS) to navigate. This reliance on satellite data is a major limitation in obstructed environments or smaller research plots where signals easily degrade. There is a clear need for cost-effective data collection systems that can operate independently of external positioning networks.

This project addresses that gap by presenting the design and implementation of a vision-integrated autonomous rover built entirely from accessible, low-cost hardware. The system completely bypasses external GPS, relying instead on onboard edge processing and visual markers to navigate and identify sampling locations. The architecture splits processing between two primary units. A Raspberry Pi 4B runs the ROS2 Jazzy middleware to handle computer vision, node orchestration, and local Wi-Fi communication. Simultaneously, an Arduino Nano executes real-time hardware operations, acquiring sensor data and controlling the wheel locomotion via a high-power BTS7960 motor driver.

The technical focus of this paper centers on achieving reliable state estimation and navigation despite the inherent physical limitations of budget hardware. First, we outline the implementation of an Extended Kalman Filter (EKF) that fuses basic wheel encoder odometry with MPU6050 inertial data to maintain stable localization. Second, we detail a lightweight OpenCV vision pipeline designed for path following and marker detection. Because low-cost mechanical and processing components introduce latency that causes the rover to overshoot its path, we developed a custom delay-compensation filter. This filter actively reverses rover movement to self-correct tracking errors. Additionally, we implemented a history-dependent sweep prioritization algorithm that uses past trajectory data to intelligently re-acquire lost tracking lines.

The paper is structured as follows. Section II details the hardware layout and system architecture. Section III covers the kinematics and EKF mathematical formulation. Section IV explains the vision-based navigation logic, including the delay-compensation filter and sweep prioritization. Section V presents field results from the soil data acquisition, and Section VI provides the conclusion.

2. SYSTEM ARCHITECTURE

The rover is engineered as a distributed, edge-computing platform designed to isolate high-level decision-making from low-level real-time actuation. The system relies on a local Wi-Fi communication layer established by the primary computing node, eliminating the need for internet connectivity or external networking infrastructure.

1. High-Level Processing, Vision, and Soil Sensing

The primary computational node is a Raspberry Pi 4B, which serves as the central brain of the rover. It executes the Robot Operating System 2 (ROS2 Jazzy) middleware, pro-

cessing high-bandwidth data from the visual sensors and computing the navigational logic. A Raspberry Pi Camera Module V2 is connected via the Camera Serial Interface (CSI) port for real-time image capture.

Unlike traditional setups where all sensors are routed through a microcontroller, the Raspberry Pi directly handles the environmental data acquisition and actuator deployment loop. It interfaces directly with a waterproof DS18B20 temperature sensor probe and an SP Electron soil moisture sensor. To physically collect soil data, a linear actuator is integrated into the chassis. Power to this actuator is managed by a high-power BTS7960 H-bridge motor driver, which receives its deployment commands directly from the Raspberry Pi's GPIO pins.

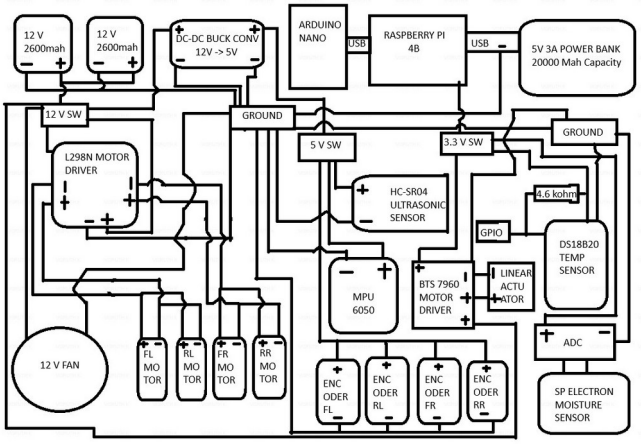

Fig. 1. Hardware power distribution architecture. The schematic illustrates the physically isolated 12V supply for the high-draw motor drivers, step-down DC-DC conversion for the sensor array, and the dedicated 5V power bank ensuring stable voltage for the primary Raspberry Pi compute node.

2. *Low-Level Hardware Interface and Sensing*

An Arduino Nano functions as the dedicated locomotion and navigation hardware controller, communicating with the Raspberry Pi via a Universal Asynchronous Receiver-Transmitter (UART) serial bridge. The Nano is responsible for real-time motor control and polling high-frequency navigational sensors without being interrupted by the Pi's heavier processing tasks. It directly interfaces with the following components:

- An MPU-6050 Inertial Measurement Unit (IMU) communicating via I2C to provide 6-axis motion tracking for spatial orientation.
- An HC-SR04 ultrasonic sensor utilized strictly for forward obstacle avoidance to prevent collisions.
- Four Direct Current (DC) motors for locomotion, powered via an L298N motor driver controlled by the Nano.
- To capture localized odometry, each wheel is equipped with a rotary encoder wired to the Arduino; OE-28 encoders are used on the front wheels, and OE-775 encoders are mounted on the rear wheels.

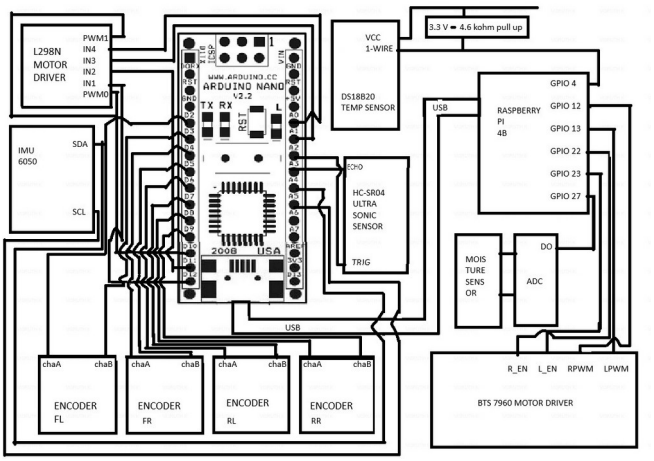

Fig. 2. Low-level logic and communication wiring schematic. The Arduino Nano functions as a dedicated edge controller for high-frequency encoder polling and real-time motor actuation, bridging sensory data to the Raspberry Pi via a continuous serial interface.

3. KINEMATICS AND STATE ESTIMATION

To achieve autonomous navigation in a GPS-denied environment, the rover must accurately track its position and orientation over time. This is accomplished through a combination of raw wheel odometry and inertial sensor fusion.

1. *Skid-Steer Kinematics*

The rover utilizes a four-wheel skid-steer drive mechanism. A primary challenge in budget hardware is manufacturing asymmetry, which results in irregular pulse generation across different wheel encoders. To prevent the navigational algorithm from interpreting these irregular pulses as phantom rotations, the linear displacement of each wheel (d_i) is calculated independently using bespoke Ticks-Per-Meter (TPM_i) calibration constants:

$$d_i = \frac{\Delta tick\, s_i}{TP\, M_i}$$

The physical track width of the rover is asymmetric, measuring 19.6 cm at the front and 18.5 cm at the rear. An effective track width (L) of 0.191 m is used for kinematic calculations. The average linear displacement for the left ($d_{\square}$) and right ($d_{\square}$) sides of the chassis are derived from the individual wheel distances. From these, the center displacement (d_{center}) and the change in heading ($\Delta\theta$) are calculated as:

$$d_{center} = \frac{d_{+d_{\square}}}{2}$$

$$\Delta = d_{-\frac{d_{\square}}{L}}$$

The global pose of the rover (x,y,θ) is then updated continuously using the following integration:

$$x_t = x_{t-1} + d_{center} \cos\left(\theta_{t-1} + \frac{\Delta}{2}\right)$$

$$y_t = y_{t-1} + d_{center} \sin\left(\theta_{t-1} + \frac{\Delta}{2}\right)$$

$$\theta_t = \theta_{t-1} + \Delta$$

2. Mechanical Drift Compensation

While the mathematical formulation is sound for ideal differential drive systems, the physical rover exhibits an outward alignment (toe-out) on the front wheels. In a skid-steer configuration, this introduces a permanent mechanical conflict, causing the wheels to scrub laterally across the ground during forward motion. This friction guarantees rapid rotational drift when relying exclusively on dead-reckoning odometry.

3. Extended Kalman Filter (EKF) Formulation

To eliminate the accumulated error from wheel slip and mechanical drift, an Extended Kalman Filter (EKF) is implemented using the ROS2 `robot_localization` package. The EKF fuses the raw wheel odometry with data from the onboard MPU-6050 IMU to estimate a highly accurate state.

Because the wheel encoders are reliable for longitudinal distance but highly inaccurate for rotation, the EKF measurement matrix is configured to trust *only* the linear X velocity () from the odometry. Conversely, because the IMU's accelerometer is susceptible to high-frequency motor vibration, the filter ignores the linear acceleration data and strictly trusts the IMU's gyroscope for angular Z velocity ().

To satisfy the covariance requirements of the EKF and prevent matrix calculation errors, specific variance values (e.g., for angular velocity) are artificially injected into the raw IMU messages at the serial bridge layer. Furthermore, a software deadband filter is applied to the IMU data stream; angular velocities below rad/s are forced to zero, successfully eliminating phantom "ghost" drift when the rover is physically stationary.

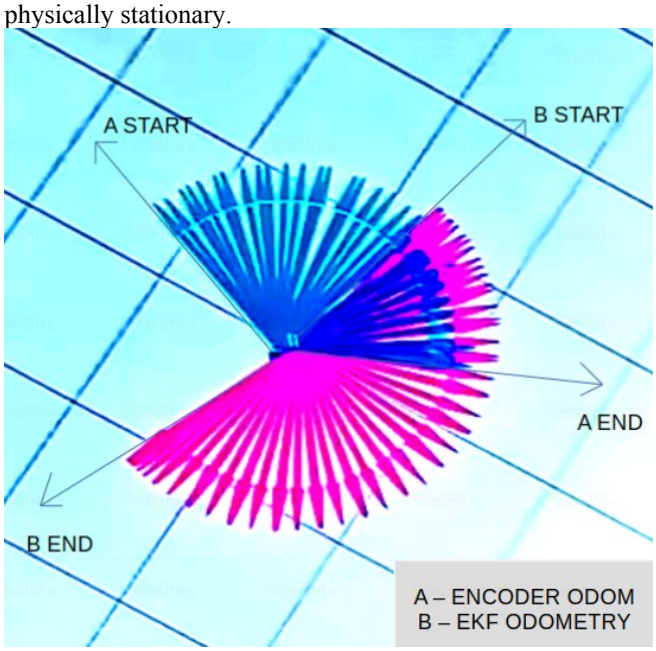

Fig. 3. Comparative trajectory analysis during a commanded 180-degree reversal maneuver on a high-slip dirt surface. Direct annotations highlight the robust spatial alignment of the EKF-fused path compared to the severe rotational drift exhibited by the encoder-only dead reckoning.

IV. VISION-BASED NAVIGATION AND EVENT HANDLING

Autonomous navigation in the GPS-denied environment is driven by a monocular vision pipeline utilizing OpenCV. To manage computational load, the system operates on a finite state machine that dynamically switches between reactive visual servoing and Extended Kalman Filter (EKF) dead-reckoning.

A. Image Processing and Marker Detection

To optimize the limited processing capacity of the Raspberry Pi, incoming camera frames are subjected to a Region of Interest (ROI) crop, discarding the top 50% of the image to isolate the floor plane. The vision pipeline continuously processes two simultaneous tracks:

1. Path Tracking: The cropped frame is converted to grayscale, smoothed via a Gaussian blur, and processed through an inverted binary threshold. The algorithm isolates the largest continuous contour and utilizes image moments to calculate the horizontal centroid (cx). A proportional controller generates the angular steering command (ωz) based on the pixel error between cx and the frame center.
2. Destination Trigger: The algorithm searches for green sampling markers by converting the ROI to the HSV color space. To prevent the rover from classifying background objects (e.g., green walls) as targets, a geometric aspect-ratio filter is applied. The system only registers a valid destination if the bounding box width is greater than half its height, and the bottom edge of the box touches the lower pixel threshold of the frame.

B. Latency Compensation and Active Braking

A critical challenge in budget robotic systems is the inherent latency between frame capture (operating at 10 FPS), image processing, and mechanical actuation. In a skid-steer chassis, this delay causes the rover to physically overshoot the path during sharp corrections, leading to a phenomenon known as "ping-ponging" or oscillatory tracking error. To mitigate this, a custom delay-compensation filter termed Active Braking was developed. Rather than commanding a zero-velocity stop when the camera re-acquires a lost line, the control node dynamically reverses the rover's rotational momentum. If the rover was sweeping right, the algorithm applies a localized, high-torque reverse pulse (e.g., ωz = 0.8 rad/s to the left) for 150 milliseconds. This actively dampens the mechanical inertia, stopping the rover dead-center on the newly acquired path. Furthermore, the forward tracking utilizes a "stutter-step" routine, briefly pausing motor actuation at calculated intervals to allow the camera to process unblurred frames.

C. *History-Dependent Sweep Prioritization*

When the rover encounters a sharp 90-degree corner, the path rapidly exits the camera's field of view, triggering a blind state. To re-acquire the path efficiently without executing a full 360-degree rotation, the system utilizes a history-dependent sweep prioritization algorithm. In the frames immediately preceding the loss of visual tracking, the vision pipeline records the path's exiting trajectory. If the calculated centroid cx was located in the right hemisphere of the frame array (cx > width / 2), the directional memory stores a "RIGHT" bias. Upon entering the blind state, the rover references this memory and initiates a localized, EKF-guided micro-search strictly in the prioritized direction. By scanning the mathematically probable location first before widening

its search arc, the rover drastically reduces recovery time, minimizes cumulative wheel slip, and prevents unnecessary straying from the operational grid.

V. EXPERIMENTAL RESULTS AND CONCLUSION

A. Experimental Setup

The autonomous agricultural rover was evaluated across two distinct environments to assess the robustness of the vision pipeline and kinematic control. The standardized test course consisted of a 1m x 4m rectangular loop featuring sharp 90-degree corners. A successful trial was defined as completing one full traversal of the loop (clockwise or counter-clockwise) and executing a commanded halt at a designated stopping zone without critical failure.

B. Navigation and Kinematic Performance

In the controlled indoor environment (flat tile surface with high-contrast black tracking lines and stable lighting), the rover achieved a 75% success rate. Failures in this environment were minimal, consisting of occasional line-loss events requiring a localized rescan (20%) and rare software anomalies triggering the motor deadband (5%). The semi-structured outdoor environment (flat dirt surface) presented significantly higher kinematic and visual challenges, yielding a 56% success rate. The primary failure mode was attributed to the skid-steer chassis; severe wheel slip on the loose particulate surface altered the rover's turning radius during 90-degree maneuvers. This slip frequently pushed the tracking line outside the limited field of view (FOV) of the camera. Secondary failures were caused by environmental lighting variations disrupting the fixed thresholding of the vision pipeline and recurrent deadband triggers.

C. Latency Compensation and Sensor Deployment

During the destination triggering phase, systemic latency between frame processing and mechanical actuation caused a consistent, measurable braking delay. Rather than attempting to overdrive the hardware limits of the Raspberry Pi, this was successfully mitigated through physical offset calibration, placing the green destination marker at a calculated distance preceding the true target zone. Upon successful halting, the linear actuator system performed with high reliability. The mechanism accurately deployed the ground-penetrating sensor into the soil. Initial data relays demonstrated clear sensitivity to environmental factors, successfully outputting distinct signal variances between high-moisture and low-moisture soil profiles.

D. Conclusion

This architecture validates the viability of utilizing constrained, budget-friendly hardware for autonomous navigation and soil sampling. While the vision-based tracking performs reliably in controlled, high-contrast settings, the rigid thresholding and limited camera FOV prove restrictive on high-slip, variable-lighting agricultural surfaces. Future iterations will focus on integrating a wider-angle lens, implementing dynamic auto-exposure adjustments within the OpenCV pipeline, and utilizing the onboard EKF to better compensate for the unpredictable odometry drift caused by dirt-surface wheel slip.

REFERENCES

[1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," Science Robotics, vol. 7, no. 66, May 2022.

[2] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000.

[3] T. Moore and D. Stouch, "A Generalized Extended Kalman Filter Implementation for the Robot Operating System," in Intelligent Autonomous Systems 13, Springer, 2016, pp. 335-348.

[4] Raspberry Pi 4 Computer Model B Datasheet, Raspberry Pi (Trading) Ltd., Cambridge, UK, 2019.

[5] ATmega328P Datasheet, Microchip Technology Inc., Chandler, AZ, USA, 2015.

[6] MPU-6000 and MPU-6050 Product Specification, InvenSense Inc., San Jose, CA, USA, 2013.