TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 69, Issue Special III, July, 2026

LOW-COST AND RAPID PROTOTYPING IN AUTONOMOUS NAVIGATION BY USING MATLAB MOBILE AND SMARTPHONE SENSORS FOR A DRIVING ROBOT

Cristin BIGAN, Valentin PANDURU

Abstract: *This paper explores the integration of MATLAB Mobile and smartphone sensor systems to develop and test autonomous driving robot control functions. With the ubiquity of mobile computing and embedded sensing, smartphones provide accessible inertial, vision, and localization data that can be leveraged for robotics. The research investigates methods of interfacing smartphone sensors with robot control via MATLAB, establishing real-time data acquisition, defining processing workflows, and implementing autonomous navigation strategies. Experimental results demonstrate that smartphone sensors, when fused via an Extended Kalman Filter (EKF) and processed on MATLAB Mobile, can support reliable perception and control loops for lane keeping and waypoint navigation, offering a cost-effective and rapid prototyping alternative to traditional LiDAR/DGPS setups.*
Keywords: *MATLAB Mobile, Smartphone, Sensors, Driving, Prototyping.*

1. INTRODUCTION

Autonomous driving systems rely heavily on multi-modal sensor fusion to perceive the environment and make safety-critical decisions.

Traditional research platforms utilize high-fidelity sensors like Velodyne LiDAR, millimeter-wave radar, and RTK-GPS. While accurate, these systems are prohibitively expensive for educational and prototyping purposes [1]. Conversely, modern smartphones possess a suite of sensors: accelerometers, gyroscopes, magnetometers, GPS, high-resolution cameras and significant processing power and connectivity (4G/5G/Wi-Fi).

MATLAB has long been a standard in robotics research for algorithm prototyping. The introduction of **MATLAB Mobile** bridges the gap between theoretical simulation and physical hardware, allowing the smartphone to act as an edge sensing node [2]. The research reported here shows a rapid prototyping methodology by using MATLAB Mobile for multiple sensors real time data streaming to a processing software.

The proposed framework is experimentally validated through direction-change (position correction) maneuvers aimed at improving trajectory tracking performance during autonomous navigation.

The presented implementation focuses on evaluating the effectiveness of smartphone-based sensor fusion for low-cost mobile robot localization and control. More advanced environmental perception tasks, such as traffic signs or object recognition, are considered as extensions of the current platform and remain subject to future investigation.

2. CURRENT FRAMEWORK ON MATLAB MOBILE AND ROBOTICS

2.1 Smartphone Sensors in Robotics

Modern smartphones function as compact sensor pods. Key components include (but are not limited to):

IMU (Inertial Measurement Unit): MEMS-based accelerometers and gyroscopes. While subject to integration drift, they provide high-

frequency state estimation. GPS: Provides absolute global positioning, though typical accuracy is limited to 2–5 meters without augmentation.
Camera: increasingly powerful ISPs (Image Signal Processors) allow for object detection.
Connectivity: Wi-Fi and Bluetooth Low Energy (BLE) facilitate telemetry.

2.2 MATLAB in Robotics

MATLAB provides specific toolboxes that streamline the development pipeline:
Computer Vision Toolbox: For feature extraction and optical flow.

Sensor Fusion and Tracking Toolbox: For implementing AHRS (Attitude and Heading Reference Systems) and Kalman Filters.

ROS Toolbox: Enabling integration with the Robot Operating System if scale-up is required.

2.3 MATLAB Mobile Architecture

MATLAB Mobile functions as a remote sensor node. It streams data to the MathWorks Cloud or a desktop MATLAB session (Connector). This allows the heavy computational lifting (image processing, matrix operations) to occur on a powerful desktop while the phone handles sensing.

3. PROPOSED SYSTEM ARCHITECTURE

The system is divided into three distinct layers: the Sensing Layer (Smartphone), the Processing Layer (MATLAB Desktop/Cloud), and the Actuation Layer (Robot Chassis).

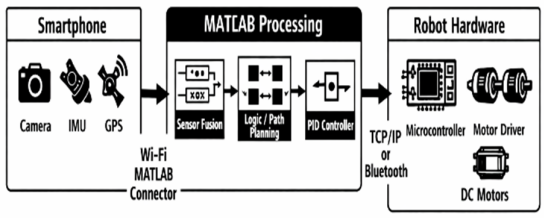


Fig. 1. High-level system architecture illustrates the data stream from the smartphone to the MATLAB processing core and the command signal to the robot actuators.

3.1 Hardware Components

To implement the specified architecture in Figure 1 the following main parts are needed:

-Mobile Platform: A differential-drive chassis driven by DC motors without wheel encoders.
-Microcontroller: An ESP32 (or Arduino-based) board acting as the low-level motor controller.
-Smartphone: Samsung Galaxy S/iPhone mounted on a vibration-dampening gimbal on the robot chassis.
-Communication Link: A local Wi-Fi router establishes a specific subnet for low-latency communication between the MATLAB host and the robot.

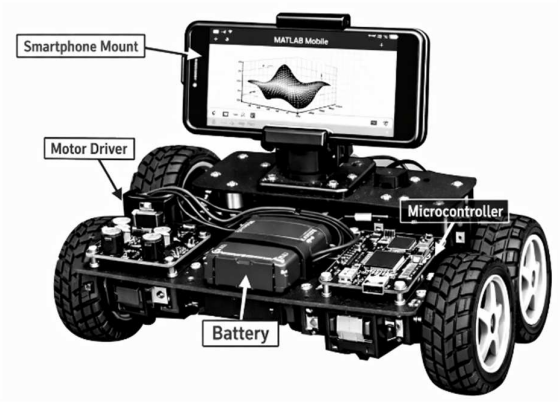


Fig. 2. Experimental setup shows the differential drive robot with the smartphone rigidly mounted to align the camera and IMU axes with the vehicle frame.

3.2 Robot organology and motion parameters

The experimental platform is a differential-drive mobile robot equipped with two independently actuated driving wheels and passive supporting wheels. The robot motion is constrained to planar movement and is described in the body-fixed coordinate system $\{B\}$ centred between the two driving wheels.

The main geometric parameters are:

- wheel radius r [m]
- wheelbase L (distance between left and right wheels) [m]
- total robot mass m [kg]
- camera offset (x_c, y_c, z_c)relative to the robot center

The robot can use:

- smartphone IMU (accelerometer + gyroscope)
- GPS receiver integrated in the smartphone
- RGB camera for visual lane detection.

Since wheel encoders are not available, linear velocity is estimated through sensor fusion using inertial measurements and GPS position updates.

3.3 Differential-drive kinematic model and simplified motion dynamics

The kinematic model of the differential-drive robot follows the classical formulation described in [5,6]:

$$v=\frac{r}{2}(\omega_R+\omega_L) \quad (1)$$

$$\omega=\dot{\theta}=\frac{r}{L}(\omega_R-\omega_L) \quad (2)$$

where:
v – linear velocity of the robot center,
ω – angular velocity around the vertical axis,
r – wheel radius,
L – distance between the left and right driving wheels,
ωL, ωR – angular velocities of the left and right wheels.
The planar kinematic equations become:

$$\dot{x}=v\cos\theta \quad (3)$$

$$\dot{y}=v\sin\theta \quad (4)$$

$$\dot{\theta}=\omega \quad (5)$$

In the proposed implementation, the angular velocity ω used in the kinematic model is obtained directly from the gyroscope measurements provided by the smartphone IMU.

The discrete-time implementation used in MATLAB is:

$$x_{k+1}=x_k+v_k\cos\theta_k\Delta t \quad (6)$$

$$y_{k+1}=y_k+v_k\sin\theta_k\Delta t \quad (7)$$

$$\theta_{k+1}=\theta_k+\omega_k\Delta t \quad (8)$$

Angular velocity ω_k is obtained from the gyroscope signal after coordinate frame alignment, $\omega_k=\omega_{IMU.}$

Due to the absence of wheel encoders and motor current sensing, a simplified first-order dynamic model is used:

$$\dot{v}=-\frac{1}{\tau}v+\frac{K}{\tau}u \quad (9)$$

where u represents the normalized motor command, K is the steady-state gain, and τ is the experimentally identified time constant.

The estimated linear velocity v obtained from the dynamic model is subsequently used in the kinematic equations (3) – (5) for pose prediction.

This model provides a practical approximation sufficient for sensor-fusion-based state estimation.

4. METHODOLOGY

4.1 Sensor Data Acquisition and Transformation

Sensors are streamed via MATLAB Mobile at a chosen value up to maximal sampling rate of 100Hz (IMU) and 1Hz (GPS). A critical step is the Coordinate Frame Transformation. The smartphone's local axes (X_p, Y_p, Z_p) often differ from the robot's body frame (X_r, Y_r, Z_r). A rotation matrix R_p^r is applied to align sensor data with the robot's kinematics.

4.2 Sensor Fusion

For this step, an Extended Kalman Filter (EKF), was used to fuse the inputs as the GPS data has noise, and IMU data drifts [3]. The probabilistic state estimation approach using EKF is based on standard mobile robotics localization techniques [5].
State vector:

$$x=[x\ y\ \theta\ v]^T \quad (10)$$

Prediction model (11):

$$x_{k-1}=f(x_{k-1},u_k)+w_k \quad (11)$$

$$f=\begin{bmatrix} x+v\cos\theta\Delta t \\ y+v\sin\theta\Delta t \\ \theta+\omega_{imu}\Delta t \\ v+a_{long}\Delta t \end{bmatrix} \quad (12)$$

where:

- ω_{imu}— gyroscope yaw rate
- a_{long}— longitudinal acceleration from IMU

The Jacobian matrix used for linearization is:

F_k =[1 0 -v sinθΔt cosθΔt
0 1 v cosθΔt sinθΔt
0 0 1 0
0 0 0 1]

Measurement model (13)

GPS provides position updates:

$$z_k = \begin{bmatrix} x_{gps} \\ y_{gps} \end{bmatrix} \quad (13)$$

$$h(\mathbf{x}) = \begin{bmatrix} x \\ y \end{bmatrix} \quad (14)$$

The EKF computes the correction term using:

$$K_k = P_k H_k^T (H_k P_k H_k^T + R)^{-1} \quad (15)$$

As a matter of how the fusion is made, MATLAB's AHRS filter object is used to estimate orientation (Quaternion) by fusing Accelerometer, Gyroscope, and Magnetometer data, which is fed in the localization loop [4].

4.3 Vision Processing

Image frames are captured at 640x480 resolution. The processing pipeline includes:

1. **Region of Interest (ROI):** Cropping the top 40% of the image to remove the horizon.
2. **Color Thresholding:** Converting to HSV space to filter specific lane colors (e.g., yellow/white).
3. **Edge Detection:** Applying Canny Edge detection.
4. **Hough Transform:** Identifying straight lines to calculate the steering angle error.

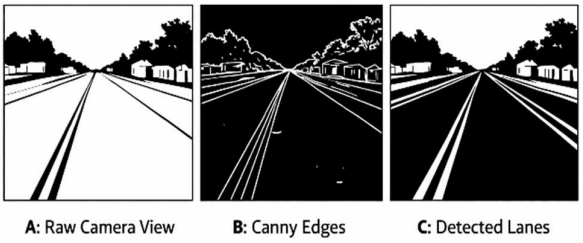


Fig. 3. Vision processing pipeline: (a) Raw input, (b) Edge detection map, (c) Hough Transform line detection identifying the drivable path.

4.4 Control Strategy

A PID controller regulates the steering angle (δ) based on the deviation from the lane center or GPS waypoint heading.

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{d}{dt} e(t) \quad (16)$$

Where $e(t)$is the lateral error calculated from vision or GPS cross-track error.

5. IMPLEMENTATION

5.1 MATLAB Mobile Setup

The MATLAB Mobile app is configured to stream to the desktop via the MATLAB Connector (connector on). This ensures the mobiledev object is accessible in the Command Window.



Fig. 4. MATLAB Mobile Interface showing real-time sensor logging configuration.

5.2 From EKF estimation to experimental visualization

The real-time interface shown in Fig. 4 receives sensor data from MATLAB Mobile. At each iteration:

1. IMU data are acquired and transformed into the robot frame.
2. EKF prediction step is applied.
3. GPS measurements trigger the correction step.
4. Estimated states $(x, y, \theta)$are logged and plotted.

Figure 6 presents the comparison between raw GPS trajectory and EKF-estimated trajectory.

5.3 MATLAB Code Structure

The main control loop runs on the desktop, as in Fig. 5.

6. RESULTS AND DISCUSSION

6.1 Trajectory Tracking

The experiments were repeated five times on the same trajectory on an outdoor track to evaluate repeatability. The EKF-based estimation reduced trajectory dispersion

compared to raw GPS measurements, improving stability during autonomous motion.

The raw GPS path was compared against the IMU-fused path. As seen in Figure 5, the raw GPS (dotted line) exhibits "jumps" of up to 3 meters. The fused path (solid line) smooths these outliers, resulting in a trajectory that more closely matches the ground truth.

6.2 Latency Analysis

One limitation observed was network latency then processing duration especially for image leading to the following values:

-Sensor Stream Latency: ~20-50ms (Wi-Fi dependent).

-Image Processing Time: ~150ms per frame (Desktop CPU).

-Total Loop Time: ~250ms. While sufficient for low-speed navigation (< 1 m/s), this latency induces oscillation at higher speeds, requiring predictive control algorithms in future iterations.

6.3 Discussion

The visual module currently performs lane detection rather than full visual odometry. The challenges described in [3], such as scale ambiguity, illumination changes, and motion blur, were observed during experiments and motivate future integration of visual odometry as an additional EKF measurement source.

7. CONCLUSION

This work demonstrates that MATLAB Mobile transforms a standard smartphone into a capable sensing unit for autonomous robotics, making it useful for rapid prototyping without other sensors and circuitry.

By leveraging the mobiledev interface and MATLAB's robust toolboxes, we successfully implemented a sensor fusion and control loop capable of lane following and waypoint navigation.

While limitations regarding network latency and GPS accuracy persist, this platform offers a high-fidelity, low-cost solution for robotics education and rapid prototyping.

Future Work includes proposals for on-Device Deployment such as using MATLAB Coder to generate C++ code to run directly on the smartphone or an embedded Raspberry Pi to eliminate network latency and integrating Visual Localization and Mapping Simultaneous for working in places without access to GPS.

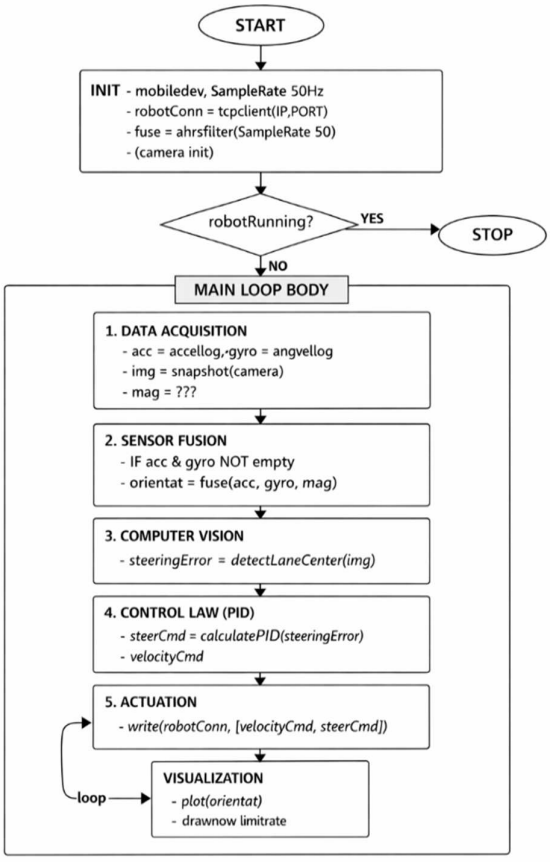


Fig. 5. Main diagram of MATLAB code processing

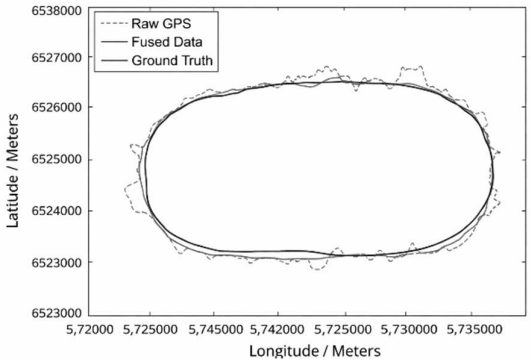


Fig. 6. Comparison of Raw GPS data versus Kalman Filtered state estimation during a circular track test. The fused estimate significantly reduces high-frequency noise.

8. REFERENCES

[1] S. Thrun, W. Burgard, D. Fox, *Probabilistic Robotics*, MIT Press, 2005.
[2] MathWorks, "MATLAB Mobile Support Package for Android/iOS Sensors," *MathWorks Documentation*, 2024.
[3] D. Scaramuzza & F. Fraundorfer, "Visual Odometry for Mobile Robots," *IEEE Robotics & Automation Magazine*, 2011.
[4] J. Fayyad et al., "Deep Learning Sensor Fusion for Autonomous Vehicle Perception and Localization: A Review", *Sensors* (Basel), 2020
[5] Siegwart, I.R. Nourbakhsh, and D. Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd Edition, MIT Press, Cambridge, MA, 2011.
[6] B. Siciliano and O. Khatib (Eds.), Springer Handbook of Robotics, 2nd Edition, Springer, 2016.

Prototipare rapidă și cu cost redus în domeniul navigației autonome prin utilizarea senzorilor din smartphone și a aplicației MATLAB Mobile pentru conducerea robotică a unui șasiu motorizat

Rezumat: Acest articol explorează integrarea sistemelor de senzori din telefoanele tip smartphone folosind MATLAB Mobile pentru a dezvolta și testa funcții ale unui robot autonom de conducere. Având în vedere omniprezența calculului mobil și a senzorilor încorporați, smartphone-urile oferă date inerțiale, vizuale și de localizare accesibile, care pot fi utilizate în robotică. Cercetarea investighează metode de interfațare a senzorilor smartphone-urilor cu controlul robotului prin intermediul MATLAB, stabilind achiziția de date în timp real, definind fluxurile de lucru pentru procesare și implementând strategii de navigație autonomă. Rezultatele experimentale demonstrează că senzorii smartphone-urilor, atunci când sunt fuzionați printr-un filtru Kalman extins (EKF) și procesați pe MATLAB Mobile, pot suporta bucle de percepție și control fiabile pentru menținerea benzii de rulare și navigarea prin puncte de referință, oferind o alternativă rentabilă și rapidă de prototipare la configurațiile tradiționale LiDAR/DGPS.

Cristin BIGAN, PhD, Associated Professor, National University of Science and Technology Politehnica Bucharest, Faculty of Industrial Engineering and Robotics, iulian.bigan@upb.ro, +40 723 920 009, 15 Unirii Blvd., Bucharest, Romania.
Valentin PANDURU, PhD, Lecturer, National University of Science and Technology Politehnica Bucharest, Faculty of Industrial Engineering and Robotics, valentin.panduru67@upb.ro, +40 721 078 737, 94 Grigore Alexandrescu Street, Bucharest, Romania.