# An Algorithmic Approach to Adapting Edge-based Devices for Autonomous Robotic Navigation

Mbadiwe S. Benyeogor[1,3,*], Kosisochukwu P. Nnoli[2], Oladayo O. Olakanmi[1,3], Olusegun I. Lawal[4], Eric J. Gratton[1], Sushant Kumar[1], Kenneth A. Akpado[5], and Piyal Saha[1]

[1]Automata Research Group (ARG), OEMA Tools and Automation Ltd., Ibadan, Nigeria
[2]Department of Computer Science and Electrical Engineering, Jacobs University, Bremen, Germany
[3]Department of Electrical and Electronic Engineering, University of Ibadan, Ibadan, Nigeria
[4]Advanced Aerospace Engine Laboratory, NASRDA, Abuja, Nigeria
[5]Department of Electronic and Computer Engineering, Nnamdi Azikiwe University, Awka, Nigeria

## Abstract

A recent significant progress has been made in development of intelligent mobile robots that is capable of autonomous navigation using an edge-computing system. This could sense changes in its environment to control its mechanical behavior towards accomplishing preprogrammed motions. Several algorithms were used in developing the robot's control software. These include the moving average filter, the extended Kalman filter, and the covariance algorithm. Using these algorithms, the robot could learn from its sensors to estimate and control its position, velocity, and the proximity of obstacles along its path, while autonomously navigating to a predetermined location on the earth's surface. Results show that our algorithmic approach to developing software systems for autonomous robots using edge-computing devices is viable, cost-efficient, and robust. Hence, our work is a proof of concept for the further development of edge-based intelligence and autonomous robots.

Received on 04 June 2021; accepted on 26 June 2021; published on 05 August 2021

**Keywords:** autonomous navigation, edge computing, intelligence schema, localization, obstacle avoidance

doi:10.4108/eai.5-8-2021.170559

## Nomenclature

| | |
|---|---|
| $\delta\phi$ | Error tolerance in longitude measurement |
| $\delta\theta$ | Error tolerance in latitude measurement |
| $\ell$ | Displacement of point $P_2$ from point $P_1$ |
| $\gamma$ | Halve the distance between the wheels on either sides |
| $\mathbb{H}(\Theta)$ | Haversine function |
| $\mathbf{x}[k]$ | State vector |
| $\mathbf{z}[k]$ | GPS sensor's model |
| $\omega$ | Yaw-rate of the robot |
| $\phi_1, \phi_2$ | Longitude of points $P_1$ and $P_2$ respectively |
| | Bearing of $P_2$ from $P_1$ with reference to the magnetic north pole |
| $\tau_L$ | Torque generated by wheels on the left |
| $\tau_R$ | Torque generated by wheels on the right |
| $\Theta$ | Central angle between $P_2$ and $P_1$ on earth surface |
| $\theta_1, \theta_2$ | Latitude of points $P_1$ and $P_2$ respectively |
| $\varepsilon$ | Course angle with reference to the desired path |
| $\varphi$ | Bearing of robot at $P_1$ with reference to the magnetic north pole |
| $\vartheta$ | Steering angle with reference to the desired path |
| $dist_F$ | Estimated proximity of an obstacle from the robot |

*Corresponding author. Email: samrexbenzil@gmail.com

| | |
|---|---|
| $I_T$ | Total inertial moment of the robot |
| $I_w$ | Inertial moment of a particular robot wheel |
| $J_F$ | Jacobian state transition matrix |
| $J_H$ | Jacobian observation transition matrix |
| $K[k]$ | Kalman gain |
| $m_T$ | Total mass of the robot |
| $P[k]$ | Covariance matrix of the state |
| $Q$ | Covariance matrix of the noise |
| $R$ | Covariance matrix of the observation noise |
| $r, \beta$ | Radius of the of the earth and wheels respectively |
| $V$ | Resultant linear velocity of the robot |

## 1. Introduction

A popular approach to developing an autonomous robot is to adopt Central Processing Unit (CPU) based computers for running the robot's control software [1]. The NVidia hybrid GPU-CPU computer is a popular platform for developing such autonomous systems, which comes with great development cost [2]. Evidently, the size of computing power that is required to make a robot become intelligent depends on the type and complexity of the intelligent algorithm that needs to be computed [3]. Considering the recent advances in microprocessor technology, today's roboticists now have more robust and versatile system-on-chip (SoC) computers and microcontroller platforms at their disposal. Unlike the types of microcontrollers that were available few decades ago, today's microcontrollers and single-board computers now compete with the orthodox central processing unit (CPU) based platforms in terms of speed, processing power, memory size, input/output interfaces, and programmability for computing simple artificial intelligence (AI) algorithms and autonomous functions. These have made it possible to develop various AI-based embedded software that run on SoC-computers and microcontrollers, by an approach that is popularly and more recently dubbed "edge-computing".

In this paper, we propose an algorithmic approach to implementing edge-based intelligent motion control scheme for our quadrupedal-wheeled robot in [4]. The hardware components of the robot's control system include a system-on-chip computer and a microcontroller. The robot's control software comprises both parallel and object-oriented algorithms, which constitute its intelligence schema. We also introduced a method for creating robotic systems that are intelligent and capable of autonomous behaviors using edge-computing devices. This enabled us to explore the possibilities in achieving microprocessor-grade computations with edge-based devices. Our aim is to show how effectively an edge-computing device could be adapted for point-to-point autonomous navigation of a mobile robot. This involves physical improvements on the robotic system developed by us in [4], development of the controller, and formulation of relevant parallel computation and control algorithms using object-oriented (OOP) programming techniques.

We would start with a review of relevant literature in Section 2. Our experimental platform would be detailed in Section 3 with the explanation of the newly proposed intelligent schema in Section 4. We would discuss the states estimation and motion control of the robot in Section 5. We conclude with the summary of the result of our field experiments in Section 6 while discussing its possible applications.

## 2. Literature Review

The microcontroller is a complete single-chip computer system that is optimized for the primary function of control. Basically, the microcontroller comprises a microprocessor, a Read-Only Memory (ROM), a Random-Access Memory (RAM), several Input/output (I/O) interfaces, and one or more serial ports. Modern microcontrollers are enhanced with higher processor's speed, Radio/Wi-Fi capability, sufficient memory, integrated Analog-Digital Converter (ADC), and a boot-loader (i.e., an embedded operating system) to facilitate OOP. These novel features have opened the possibility of implementing AI and Internet of Things (IoT) functions with the microcontroller [5].

Besides high development cost, the adoption of CPU-based computers for autonomous robot could lead to over computerization, which in turn could result to a huge and clumsy robot that takes so much time and energy to perform simple intelligent tasks. This is evident in a study by Stewart *et al.*, which discusses the need to implement AI right inside the microchip [6]. According to them, "it makes no sense to use the CPU to put just a bit of intelligence into a thermostat". Thus, the future of AI will see a major paradigm shift, from the traditional method of cloud- and CPU-based AI computation to localized computations in the microcontroller, which is referred to as edge intelligence or edge-computing [6]. To avoid the computational redundancy and the economic constraints associated with CPU-based AI-computation, some roboticists now adopt the concept of edge-computing. For instance, Mamdoohi *et al.* adopted the PIC32MX Microcontroller to demonstrate how a microcontroller could be used to implement a genetic algorithm for polarization control [7].

Their experiment showed real time computation of the complex algorithm with an average latency of 17 microseconds, which according to them, is low enough for their application. Also, Hussain *et al.* developed autonomous robot using the ATMEL AT89C52 Microcontroller for logistical navigation, based on their hypothesis that high-level algorithms can be encoded into microcontroller for simple AI-based tasks [8].

The concept of edge-computing is also applicable to domestic service robotics as a home automation system. This has inspired the development of microcontroller based robots with sufficient intelligence to perform simple household chores. For example, Mir-Nasiri *et al.* developed a pneumatically actuated wall-climbing robot using the PIC16F877A Microcontroller [9]. This could perform glass cleaning tasks, while it is navigating autonomously along the exterior walls of high-rise buildings, under the guidance of four proximity sensors and an optical odometer. Similarly, Apoorva *et al.* used an Atmega328 Microcontroller board to develop an autonomous robot with a low-level intelligence for tracking, picking, and disposing garbage. Thus, citing how this simple AI could relieve humans from the monotonous and hazardous job of waste collection [10].

For autonomous navigation, the SoC-computer and the microcontroller have proven to be effective edge computing devices. An exciting demonstration of this, is the work of Efaz, which involves the design of a speed-controlled path-finding obstacle avoidance robot, using OOP techniques [11]. This shows the practicality of edge-based physical computing using object-oriented algorithms. The Kalman filter is a simple AI algorithm that takes input data from multiple sensors and estimates unknown variables, amidst potentially high level of signal noise; making it a very significant tool for autonomous navigation. Because of its simplicity, many engineers use it to develop edge-based guidance systems for autonomous robots. For example, Vukelic *et al.* implemented the Extended Kalman Filter (EKF) algorithm for fusing data from an inertial sensor and a Global Positioning System (GPS) Sensor using the mbed-LPC1768 Microcontroller, for the autonomous robot navigation [12]. Their results showed no difference between the practical implementation of the EKF on the microcontroller and that of the system simulation.

The second class of edge-computing devices that competes well with many desktop machines in terms of computing power, graphics processing and versatility are the aforementioned SoC-computers. This is attributed to the continual shift from single-core to multi-core processors in embedded systems and edge computing, coupled with the availability of efficient parallel programming technologies [13]. A typical example is the Linux-based Raspberry Pi-2, which becomes very crucial when two or more algorithms are to be executed in parallel. With SoC platforms, robot software developer can now create complex edge intelligent systems that were only possible with conventional desktop computers [14].

## 3. Experimental Platform

To test our computational hypothesis, we adopted the robotic system that we developed in [4], as our experimental platform. The Fig. 1 shows that the robotic system features an active suspension system, skid steering mechanism, several sensors, and an edge-based control system.

The 3D model of our robot's mechanical system is shown Fig. 2. The details of the underlying mechanics are given in [4]. Hence, the rest of this paper will focus on the formulation of parallel/objected-oriented mathematical algorithms that can be executed by mean of edge-computing to estimate and control the navigational status of our robot as an intelligent system.

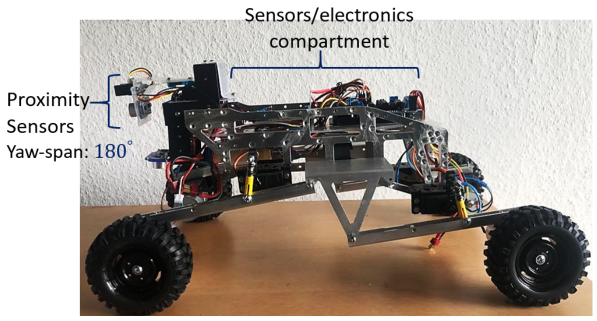
Sensors/electronics compartment
Proximity Sensors Yaw-span: 180°

**Figure 1.** Experimental platform

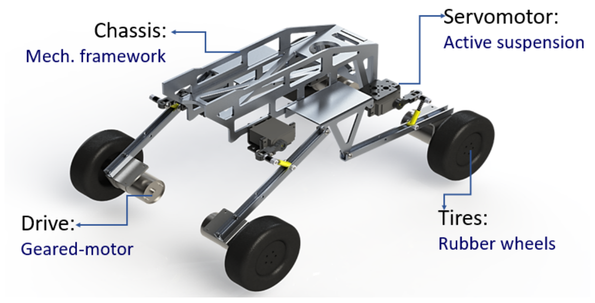

**Figure 2.** 3D model of robot mechanical system

### 3.1. Edge-computing architecture and hardware

The computing and control architecture of our robotic system incorporates the Raspberry PI-2 SoC-computer and an Atmega2560 Microcontroller (MCU) board as shown in Fig. 3. The Raspberry PI-2 SoC-computer serves as the "companion computer" that executes complex calculations in parallel to support the control functions of the MCU board (i.e., the main controller), which performs all the low-level computing and control functions. The algorithms that are implemented on the companion computer include the proximity data fusion algorithm (i.e., Algorithm 2), covariance algorithm (i.e., Algorithm 3), and the localization algorithm (i.e., Algorithm 5). A companion computer is necessary because these algorithms require high computing power and also the scheduling function of an operating system to run in parallel. For this purpose, the open message passing (Open-MP) application program interface (API) is used based on the concepts in [13]; and takes advantage of the multi-core ARM processor in the Raspberry Pi-2 (i.e., the companion computer of our robot). This performs the functions of

- collecting measurement data from the navigational sensors on-board the robot
- estimating the states of the robot
- transmitting the resulting information to the main controller.

The types and functions of these sensors are cataloged in Table 1. Among these sensors are the proximity sensors of our robotic system, which comprises an ultrasonic sensor and an Infrared (IR) distance sensor that are mounted on the frontal projection of the robot's chassis through a servo-controlled revolver (with a yaw rotation span of 0° to 180°). At the lower-level, the algorithm that are implemented on the main controller include the intelligence scheming algorithm (i.e., Algorithm 1), obstacle avoidance algorithm (i.e., Algorithm 4), path-tracking algorithm (i.e., Algorithm 6), and the maneuvering algorithm (as described in Subsubsection 5.2). The main controller is enhanced with an L293D-IC based motor driver, which enables it to regulate the flow of electrical power to the driving motors, during motion control.

Again, using the Universal Asynchronous Reception and Transmission (UART) protocol, two serial communication channels are established between the companion computer and the main controller, to enable real-time transfer of information, control signal, and computational request between the two devices. Both the companion computer and the main controller features additional I/O ports for the integration of more sensors and actuators as external peripherals, when necessary or during field tests. The entire hardware system is powered through a DC-DC bulk converter, which is used to convert the 12 volts DC supply from the robot's battery to the current/voltage requirement of the companion computer, main controller, and their peripherals.

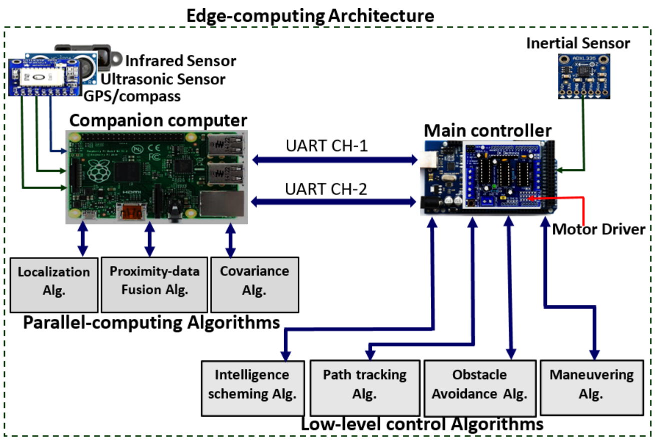

**Figure 3.** Edge-computing architecture, hardware, and algorithmic structure

## 4. Intelligence Schema

The intelligence schema (i.e., INTEL_SCHEMA function in Algorithm 1) of our robot's control flow involves three basic functions for the direct control of the robot's perceptual responses and motion. These objective functions are enumerated as follows:

1. CHANGE_PATH (in Algorithm 4).
2. AUTO_NAVIGATE (in Algorithm 6).
3. MOVE_FWD (in Listing 2).

These above listed functions call upon one another and other subordinating functions (that are discussed in Section 5) to make the robot act as an intelligent agent. Algorithm 1 starts up the robotic system once its power switch is turned on. It coordinates Algorithm 4 and 6, which are the actual autonomous control functions of the robot, of which precision dependent on the accuracy of two other subordinate functions, which are

1. PROX_ESTIMATE (in Algorithm 2), and
2. POSITION_ESTIMATE (in Algorithm 5).

The Algorithm 2 is a data fusion algorithm. The input to this algorithm are proximity measurements from the ultrasonic sensor and IR distance sensor that are embedded on the robot. This algorithm fuses the duo proximity data into a single distance estimate, $dist_F$, to minimize measurement error and noise. The estimated value of $dist_F$ is used in Algorithm 1 to decide what action the robot should take (i.e., obstacle avoidance or auto-navigation), while it is moving to the target location. The value of $dist_F$ is also used to regulate the driving speed ($V$) of the robot. Here, $V$ is a scalar

**Table 1.** List of navigational sensors on-board the robot and their functions (Note: 'wrt' is short for "with respect to")

| S/N | Sensor type | Function |
|---|---|---|
| 1 | Ultrasonic sensor | Obstacle proximity measurement |
| 2 | Infrared sensor | Obstacle proximity measurement |
| 3 | GPS sensor | Geo-spatial position measurement |
| 4 | Compass sensor | Measures bearing wrt magnetic North |
| 5 | Inertial sensor | Measures roll/pitch and acceleration |

function of $dist_F$. This enables the robotic system to adapt to the variation in the proximity of obstacles along its path, while it is maneuvering to the target location. In essence, Algorithm 1 acts as a central caller based on the value of $dist_F$ as computed by Algorithm 2. The complete description of how Algorithm 1 functions in coordination with the other subordinate Algorithms is discussed in the remaining part of this paper.

**Algorithm 1** Intelligence schema (Note: $P_1$ and $P_2$ are the location of the robot and the target respectively)

```
Require: $dist_F$                                   ▹ proximity estimate
 1: function INTEL_SCHEMA
 2:     repeat
 3:         $dist_F \leftarrow$ PROX_ESTIMATE              ▹ in Algorithm 2
 4:         ▹ ———Proximity controlled motion———
 5:         set $Left\_motor\_speed$ as ($dist_F$ cm/s)
 6:         set $Right\_motor\_speed$ as ($dist_F$ cm/s)
 7:         continue                                 ▹ To keep moving
 8:         ▹ ——————Decision——————
 9:         if d $\leq$ 40cm then
10:             call CHANGE_PATH                     ▹ Avoid obstacles
11:         else if 40cm < d $\leq$ 120cm then
12:             call MOVE_FWD                        ▹ Move forward
13:         else if d $\geq$ 120cm then
14:             call AUTO_NAVIGATE                   ▹ Move to target
15:         end if
16:     until $(P_1[\theta_1, \phi_1] \approx P_2[\theta_2 \pm \delta\theta, \phi_2 \pm \delta\phi])$
17: end function
```

## 5. States Estimation and Motion Control

Autonomous navigation involves the solution to the problem of finding a collision-free motion between an initial and a target location in space and time [15]. Therefore, mathematical algorithms are formulated in this section for accurate estimation of an obstacle's proximity from the robot and the position of the robot in the geographical coordinate system. These are used to perform obstacle avoidance and path tracking motion-control functions respectively. The former is discussed in Subsection 5.1 while the latter is discussed in Subsection 5.2. We ensured that the adopted mathematics and models are as simple as possible so that the resulting algorithms can be implemented using the companion computer and the main controller. In this regard, only a non-holonomically constrained 2-D model of our robot is used.

### 5.1. Proximity sensing and obstacle avoidance

To enhance the control of our robot's motion during obstacle avoidance, a technique was developed for detecting the proximity of an obstacle from it as shown in Fig. 4. This technique involves the use of the ultrasonic sensor and the infrared distance (IR) sensor to simultaneously measure how distant the obstacle is from the robot. This is to minimize the error associated with each of these two sensors, while also harnessing their peculiar advantages. For instance, unlike the infrared sensor, the ultrasonic sensor can scan a wider volume of space and detect transparent barrier, but has some limitation when it comes to detecting hot materials. In contrast, the infrared sensor is more accurate since its beam is less conical than the ultrasound wave. To economize computing resources, we formulated a data fusion algorithm that combines the Moving Average Filter (MAF) and the covariance formula to fuse the incoming data from the two sensors and to also filter-off the noise in the signals, thereby minimizing error in proximity measurement. Based on [16], the mathematical derivation of the MAF is shown as follows,

$$\bar{d}_k = \frac{d_{k-n+1} + d_{k-n+2} + d_k}{n}, \tag{1}$$

where $\bar{d}_k$ in Eq. (1) is the average from $(k-n+1)^{\text{th}}$ to $k^{\text{th}}$ measurement values, while $n$ is the total number of values. Hence, the moving average of the previous measurement is given in Eq. (2) as,

$$\bar{d}_{k-1} = \frac{d_{k-n} + d_{k-n+1} + d_{k-1}}{n}, \tag{2}$$

$$\therefore \quad \bar{d}_k = \bar{d}_{k-1} + \frac{d_k - d_{k-n}}{n}. \tag{3}$$

Eq. (3) is the MAF formula in the form of a recursive function. The application of Eq. (3) is described in Algorithm 2, which contains the MOVE_AVE ($dist$)

function, where *'dist'* is the parameter for fetching raw proximity input-data from either sensors. This function could be called upon at real-time to consecutively calculate the moving averages of the streams of measurement data from each of these sensors. Hence, two moving average proximity values are computed at every instant – one for the ultrasonic sensor's measurements and the other for that of the infrared sensor. Prior to fusing these two moving averages, we derive a covariance ($Cov$) formula which is applied to ensure that the raw measurements from the two sensors are consistent with each other – both sensors are ranging the same obstacle. The $Cov$ is given by

$$Cov = \sum_{index=1}^{n} \frac{(L_1[index] - Ave_1) \cdot (L_2[index] - Ave_2)}{n-1}, \tag{4}$$

where $L_1 \Leftarrow dist_1$ denote proximity measurements from the ultrasonic sensor, and $L_2 \Leftarrow dist_2$ denote proximity measurements from the infrared sensor. The $index$ is the sampling integer (where $index = 1, 2, ..., n$), and $n$ is the total number of measurement samples.

At any instant, if the value of $Cov$ in Eq. (4) is positive, the mean value of the two moving averages is calculated as $dist_F$ and returned as the measurement estimate of an object's proximity from the robot. But if the value of $Cov$ is negative, Algorithm 2 is recalled, to repeat the data fusion process.The algorithm for the application of Eq. (4) is given in Algorithm 3. With this scheme, we significantly reduced error in measurements by the robot's proximity sensors to a level that is acceptable and applicable for the obstacle avoidance motion control of our robot. The Algorithm for obstacle avoidance is given in Algorithm 4. This involves several calls to various maneuvering functions (as described in Subsubsection 5.2), in an effort to find the most obstacle-free direction, before returning control to the INTEL_SCHEMA in Algorithm 1. Therefore, our robot could reliably and intelligently avoid both static and moving obstacles along its navigational pathway to a given target location.

## 5.2. Mechanics, Localization and Path-tracking

Here, a model is developed to describe how the robot would navigate, from an initial point, $P_1(\theta_1, \phi_1)$, to the target point, $P_2(\theta_2, \phi_2)$, via the shortest path possible between the points as shown in Fig. 5. This involves the knowledge of the the robots dynamics, position, and the application of control. Fig. 5 describes our robot as a skid-steering robot where $\varphi$ is the instantaneous bearing of the robot at position, $P_1(\theta_1, \phi_1)$, and $\psi$ is its bearing from the targeted location, $P_2(\theta_2, \phi_2)$, with respect to the magnetic North. The angle $\varphi$ is the

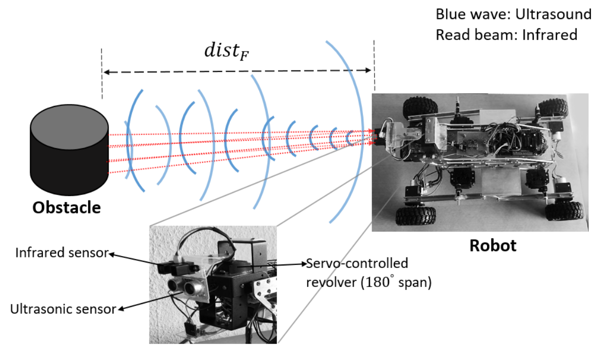

**Figure 4.** Depiction of obstacle's proximity measurement using both the ultrasonic and infrared sensor (Note: $dist_F$ is the proximity of obstacle from the robot)

variable to be controlled, while $\psi$ is the reference angle.

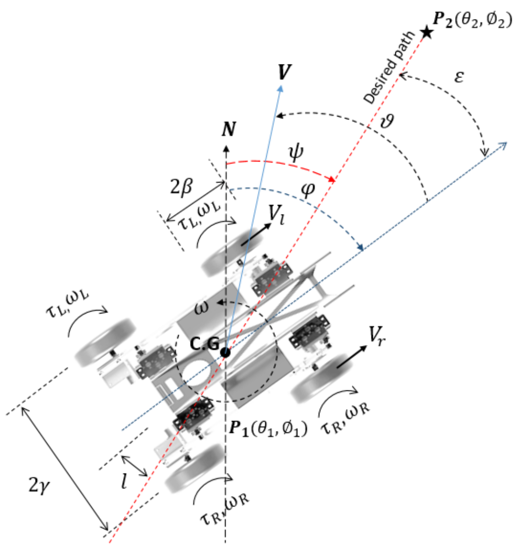

**Figure 5.** Robot navigation control model (Note: $\theta_1$ and $\phi_1$ are the latitude and longitude of $P_1$, while $\theta_2$ and $\phi_2$ are those of $P_2$, respectively)

For unregulated mobility, the yaw rate (i.e., sideways angular velocity) of the robot, $\omega$, and linear velocity, $V$ are calculated by the difference between the torques of the left and right wheels (i.e., $\tau_L$ and $\tau_R$), which directly influence the speeds ($V_L$ and $V_R$) of the wheels. The term $2\gamma$ is the kinematic width of the robot, while $\beta$ is the radius of each of the wheels. Based on rotational mechanics, $V$ and $\omega$ are expressed in Eqs. (5) and (6) as,

$$V = (\omega_L + \omega_R)\frac{\beta}{2} \tag{5}$$

$$\omega = (V_R - V_L)\frac{\beta}{2\gamma} \tag{6}$$

Following [17], the dynamics of motion of the robot is expressed in Eq. (7) as,

**Algorithm 2** Proximity-data fusion algorithm (Note: $dist_1$ and $dist_2$ are incoming data from the ultrasonic and infrared sensors respectively)

```
Require: $dist_1$ and $dist_2$                         ▷ Sensors' data
Require: $dist_F$                                ▷ Fused sensors' data
 1: $n \leftarrow 10$                                ▷ Number of samples
 2: ▷ ————————Main_Function————————
 3: function PROX_ESTIMATE
 4:     $Ave_1 \leftarrow$ MOV_AVE($dist_1$)
 5:     $Ave_2 \leftarrow$ MOV_AVE($dist_2$)
 6:     $Cov_{VALUE} \leftarrow$ COVARIANCE($dist_1, dist_2$)        ▷ In
    Algorithm 3
 7:     if $Cov_{VALUE} > 0$ then
 8:         $dist_F \leftarrow (Ave_1 + Ave_2) \div 2$
 9:     else
10:         call PROX_ESTIMATE                   ▷ Repeat process
11:     end if
12: return $dist_F$
13: end function
14: ▷ ————————Mov_Average_Filter————————
15: function MOV_AVE($dist$)
16:     $index \leftarrow 0$
17:     $sum \leftarrow 0$
18:     $average \leftarrow 0$
19:     repeat
20:         $k \leftarrow 0$
21:         $\mathbf{d}[k] \leftarrow 0$                      ▷ Initialize array cells
22:         $k \leftarrow k + 1$
23:     until $k \geq n$
24:     while $loop < 0$ do
25:         $loop \leftarrow 0$                               ▷ Start loop
26:         $sum \leftarrow sum - \mathbf{d}[index]$
27:         $\mathbf{d}[index] \leftarrow dist$
28:         $sum \leftarrow sum + \mathbf{d}[index]$
29:         $index \leftarrow index + 1$
30:         if $index \geq n$ then
31:             $index \leftarrow 0$
32:         end if
33:         $loop \leftarrow loop + 1$                     ▷ Loop forever
34:     end while
35: $average \leftarrow sum \div n$
36: return $average$
37: end function
```

$$(m_T\gamma + 2\frac{I_w}{\beta})\dot{V} = (I_T + 2\frac{\gamma^2}{\beta^2}I_w)\dot{\omega} = \tau_R - \tau_L. \qquad (7)$$

where $\dot{\omega}$ is the angular acceleration in $rad.s^{-2}$ and $I_w$ is the moment of inertial in $Kg.m^2$

For regulated navigation, $\mathbf{u} = \begin{bmatrix} \tau_R & \tau_L \end{bmatrix}^T$ is the input vector to the robot drive system, whose function is to implement motion control, by differentially

**Algorithm 3** Covariance algorithm

```
Require: $Ave_1, Ave_2, dist_1$ and $dist_2$
Ensure: $cov$
 1: $n \leftarrow 10$                              ▷ Number of samples
 2: ▷ ————————Covariance_computation————————-
 3: function COVARIANCE($l_1, l_2$)
 4:     $index \leftarrow 0$
 5:     $sum \leftarrow 0$
 6:     repeat
 7:         $k \leftarrow 0$
 8:         $\mathbf{L_1}[k] \leftarrow 0$
 9:         $\mathbf{L_2}[k] \leftarrow 0$
10:         $k \leftarrow k + 1$
11:     until $k \geq n$
12:     while index < n do
13:         $\mathbf{L_1}[index] \leftarrow l_1$
14:         $\mathbf{L_2}[index] \leftarrow l_2$
15:         $sum \leftarrow sum + (\mathbf{L_1}[index] - Ave_1) *$
    $(\mathbf{L_2}[index] - Ave_2)$
16:         $index \leftarrow index + 1$
17:         $cov \leftarrow sum \div (n - 1)$
18:     end while
19:     if $index = n$ then
20: return $cov$
21:     end if
22: end function
```

driving either pair of the robot's wheels according to Eq. 7. This input vector is electronically generated by the main controller, during which instantaneous value of $\tau_R$ and $\tau_L$ are determined, based on the robot's states and perception of its environment. Therefore, the value of $\tau_R$ and $\tau_L$ influence the mobility and direction of motion of the robot. This is measured by the embedded inertial sensor as $\mathbf{u}^* = \begin{bmatrix} V & \omega \end{bmatrix}^T$. The application and direct control of these $\tau_R$ and $\tau_L$, at the electromechanical level, are extensively discussed in Subsubsection 5.2.

According to the work of [18], the lateral, $l$, of the robot with respect to the desired path is related to $\omega$ by Eqs.(8) and (9), where $\vartheta$ and $\varepsilon$ are the steering angle and the course angle, with respect to the desired path.

$$\dot{l} = V \sin(\varepsilon) \qquad (8)$$

and,

$$\dot{\varepsilon} = \omega + \dot{\vartheta}. \qquad (9)$$

Again, the time-discrete state space model of our robot is given in

$$\mathbf{x}[k+1] = F \cdot \mathbf{x}[k] + B \cdot \mathbf{u}^*[k] \qquad (10)$$

and,

**Algorithm 4** Obstacle avoidance algorithm

```
Require: $dist_F$                                    ▷ from Algorithm 2
 1: function CHANGE_PATH
 2:     call BRAKE
 3:     call REVERSE
 4:     continue for 1.5 s
 5:     call BRAKE
 6:   ▷ —-Measure proximity of obstacle on the right—-
 7:     point $proximity\_sensor$ right
 8:     continue for 0.5 s
 9:     $dist_{Right} \leftarrow$ PROX_ESTIMATE
10:     continue for 0.5 s
11:   ▷ —-Measure proximity of obstacle on the left——
12:     point $proximity\_sensor$ left
13:     continue for 0.5 s
14:     $dist_{Left} \leftarrow$ PROX_ESTIMATE
15:     continue for 0.5 s
16:
17:     point $proximity\_sensor$ forward                ▷ Default
18:     continue for 0.5 s
19:     ▷ ——————-Decision———————
20:     if $dist_{Right} > dist_{Left}$ then
21:         while $\varphi \neq 90°$ do
22:             call ROTATE_CW
23:             if $\varphi \approx 90°$ then
24:                 call INTEL_SCHEMA
25:             end if
26:         end while
27:     else if $dist_{Left} > dist_{Right}$ then
28:         while $\varphi \neq 270°$ do
29:             call ROTATE_CCW
30:             if $\varphi \approx 270°$ then
31:                 call INTEL_SCHEMA
32:             end if
33:         end while
34:     else
35:         call INTEL_SCHEMA
36:     end if
37: end function
```

$$l[k] = M \cdot \mathbf{x}[k] + 0, \tag{11}$$

where;

$$F = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ e^{-\frac{T_s}{\tau}} & -(1 + 2e^{-\frac{T_s}{\tau}}) & (2 + e^{-\frac{T_s}{\tau}}) \end{bmatrix}, \tag{12}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \tag{13}$$

and

$$M = \begin{bmatrix} (1 - e^{-\frac{T_s}{\tau}})\frac{VT_s^2}{4\gamma} \\ (1 - e^{-\frac{T_s}{\tau}})\frac{VT_s^2}{4\gamma} \\ 0 \end{bmatrix}. \tag{14}$$

While Eq. (12) is the state matrix, Eqs. (13) and Eq. (14) are the input and output matrices respectively. Also, the variables $\tau$ and $T_s$ are the time delay and sampling time respectively. Either $l$ or $\varphi$ can serve as the controlled variable , with respect to $\varepsilon$ and $\psi$, respectively. Unlike, [18], we selected $\varphi$ as the controlled variable in order to minimize computational load. The model in Eq. (10) only serve to provide real-time estimates of the robot's location on the earth surface, which is a prerequisite for path tracking. Thus, our robot could be controlled towards the target location as a quasi-closed feedback system.

In order to ensure accurate positional mappings of the robot at all instants, we derive a position vector ($\mathbf{x}$) equation given by

$$\mathbf{x} = \begin{bmatrix} x(k) & y(k) & \vartheta \end{bmatrix}^T \Rightarrow \mathbf{map}\begin{bmatrix} \theta_1 & \phi_1 & \varphi \end{bmatrix}^T. \tag{15}$$

The position vector, $\mathbf{x}$, is mapped to the instantaneous location of the robot as expressed in Eq. (15). By applying the methods in [19] and [20], we performed the transformation between the Cartesian and geographical coordinate system as this is crucial for the real-time visualization of our robot's navigational routes.

Again, the input variable, $V$ in vector $\mathbf{u}^*[k]$, is approximately equal to the integration of the instantaneous acceleration, $a_{inst}$, of the robot in the forward direction (i.e., $v \approx \int(a_{inst})dt$ ), according to the inertial sensor's measurements. The computational syntax for this is expressed in Listing 1. Alternatively, we could directly measured the linear speed, $V$, of the robot, by attaching an optical speed sensor to one of the robot's wheels.

Listing 1: Function for computing speed from acceleration

```
/*delta_t = time between inertial
sensor readings.*/
float speed_estimator ()
{
  V_forward += accel_forward * delta_t;
  return V_forward;
   }
```

The instantaneous yaw-rate, $\omega$, of the robot equals the first derivative of the yaw angle, $\vartheta$ (i.e., $\omega = \dot{\vartheta}$), as measured by the inertial sensor and calculated by the controller. Hence, the vectors $\mathbf{x}[k] \Rightarrow \mathbf{x}[k-1]$ and $\mathbf{u}^*[k]$ serve as the inputs for estimating the real-time position of the robot.

Using the GPS sensor, the robot could directly measure its real-time position as $\mathbf{z}[k] = [x(k), y(k)]$. Therefore, the GPS sensor's model is expressed in Eq. (16) as,

$$\mathbf{z}[k] = H\mathbf{x}[k]. \tag{16}$$

In the present paper, two algorithms are adopted for the control of point-to-point navigation, based on the work of [21]. These include the localization and the path tracking algorithms in Subsubsections (5.2) and (5.2) respectively.

**Localization: position estimation function.** For real time estimation of the robots motion on the surface of the earth, the EKF is applied. This is used to formulate a localization algorithm that starts by predicting and then, estimating (i.e., updating) the position of the robot. This function is outlined in Algorithm 5.

**Algorithm 5** Localization algorithm

```
Require: $\mathbf{x}[k-1], \mathbf{P}[k-1], \mathbf{u}^*[k], \mathbf{z}[k]$
Ensure: $position[lat_1, lon_1, \varphi]$
 1: function POSITION_ESTIMATE
 2:     const $j = 0$
 3:     repeat                                          ▷ Start loop
 4:         ▷ ——————Predict——————
 5:         $\mathbf{x}[k] \leftarrow F * \mathbf{x}[k-1] + B * \mathbf{u}^*[k]$
 6:         $\mathbf{P}[K] \leftarrow J_F * \mathbf{P}[k-1] * J_F^T + Q$
 7:         ▷ ————Compute Kalman gain————
 8:         $K[k] \leftarrow \mathbf{P}[k] * J_H^T * (J_H * \mathbf{P}[k] * J_H^T + R)^{-1}$
 9:         ▷ ——————Update——————
10:         $\mathbf{x}[k+1] \leftarrow \mathbf{x}[k] + K[k] * (\mathbf{z}[k] - H * \mathbf{x}[k])$
11:         $\mathbf{P}[k+1] \leftarrow (1 - K[k] * J_H) * \mathbf{P}[k]$
12:         float $position[lat_1, lon_1, \varphi] \leftarrow \mathbf{x}[k+1]$
13:         return $position[lat_1, lon_1, \varphi]$
14:     until $j > 0$                                   ▷ Loop forever
15: end function
```

Here, the state transition and observation matrices are defined by the Jacobians in Eq. (17) and Eq. (18) respectively as,

$$J_F = \frac{\partial F}{\partial x}\bigg|_{\mathbf{x}_t, \mathbf{u}_t}, \tag{17}$$

and

$$J_H = \frac{\partial H}{\partial x}\bigg|_{\mathbf{x}_t^*}. \tag{18}$$

Algorithm 5 describes a recursive function that would run forever to continuously update the information about the position of the robot once its controller is electrically powered.

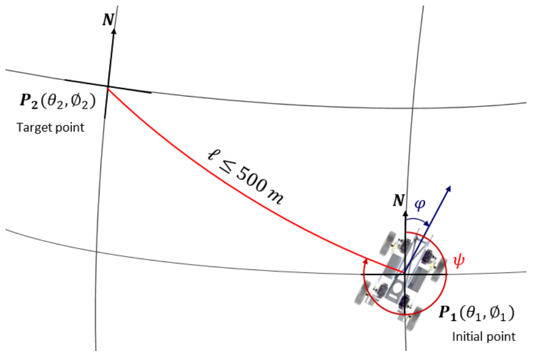

**Figure 6.** Navigational geometry of the robot in the geographic coordinate system

**Path tracking: auto-navigation function.** Having developed a means (i.e., Algorithm 5) to reliably estimate the position of the robot on the surface of the earth, we can now formulate an algorithm that computes the displacement of the target location, $P_2$, from the measured initial location, $P_1$, of the robot in the geographic coordinate system; and also, to maneuver the robot to the target location, as shown in Fig. 6. To do this, we implemented the "Haversine formula" according to the review by [22], for computing the great-circle distance (i.e., the shortest path), $\ell$, on the earth's surface between $P_2$ and $P_1$, from their longitudes and latitudes, while ignoring the presence of opportunistic obstacles along the path. This formula is described in Eq.(19) and Eq.(20) as,

$$\mathbb{H}(\Theta) = \sin^2\left(\frac{\Delta\theta}{2}\right) + \cos(\theta_1) \cdot \cos(\theta_2) \cdot \sin^2\left(\frac{\Delta\phi}{2}\right), \tag{19}$$

and

$$\ell = 2r \arcsin\left(\sqrt{\mathbb{H}(\Theta)}\right), \tag{20}$$

where $\Theta = \frac{\ell}{r}$, $\Delta\theta = \theta_2 - \theta_1$, and $\Delta\phi = \phi_2 - \phi_1$.

The bearing of $P_2$ from $P_1$ (i.e., $\psi$) is to be computed as well. Therefore, the adopted geometrical formula is given in Eq. (21) as,

$$= \arctan(X, Y), \tag{21}$$

where

$$X = \cos\theta_2 \cdot \sin\Delta\phi \tag{22}$$

and

$$Y = \cos\theta_1 \cdot \sin\theta_2 - \sin\theta_1 \cdot \cos\theta_2 \cdot \cos\Delta\phi. \tag{23}$$

In our path tracking and auto-navigation algorithm, the required destination coordinates, $P_2 = \begin{bmatrix} \theta_2 & \phi_2 \end{bmatrix}^T$, is requested from the user as an input value during the start-up sequence of the robotic system; while the updated value of initial position, $P_1 = \begin{bmatrix} \theta_1 & \phi_1 & \varphi \end{bmatrix}^T$, is recursively fetched from Algorithm 5, such that $P_1 \Leftarrow \mathbf{x}[k+1]$. The goals of the path tracking algorithm are to:

1. plot the shortest path between $P_1$ to $P_2$,
2. calculate the bearing, $\psi$, of $P_2$ from $P_1$,
3. orient the robot's motion along the bearing, $\psi$ and
4. cause the robot to move in this direction until $P_1 \approx P_2$.

In pseudocodes, this task is more sufficiently described as Algorithm 6. The boundary conditions for the application of Algorithm 6 are enumerated in Definition 5.2.

Algorithm 6 is valid under the following conditions:

1. The earth is assumed to be a perfect sphere.
2. The range of navigation (i.e., $\ell$) is limited to 500 m.
3. The robot can not reverse its motion (i.e., it can only yaw and drive forward), except Algorithm 4 is called to enable the robot avoid an obstacle.
4. Proximity of obstacles ahead must be greater than 120 cm.
5. Test navigation is performed in a controlled environment.

**Maneuvering: motor control functions.** For efficient maneuverability during obstacle avoidance or auto-navigation of our robot, the main controller should be able to control the flow of the required electric power to the robot's drive motors. To achieve this, we introduced the L293D-IC based motor driver, as an electro-software and mechanical interface, between the main controller and the four drive motors as shown in Figs. 7 and 8. This configuration involves four geared Direct Current (DC) motor (i.e., $M_1, \ldots, M_4$) that produces mechanical torques (i.e., $\tau_L$ and $\tau_R$) in order to propel the robot over a given terrain. This process is activated by a maneuvering algorithm (i.e., Algorithm 7), which is also encoded into the main controller.

The maneuvering algorithm involves several low-level functions, some of which are given in Table 2. Any of these functions could be called to control the supply of electrical voltage (ranging from 0 to 12 Volts) to the robot's four drive motors, so that they would rotate according to the set speed for the robot to move in the specified direction. This is done so that the speed and direction of rotation of each drive motor is directly proportional to the magnitude and sign of the voltage applied across its terminals. For instance, the function/function call for MOVE_FWD and ROTATE_CW are expressed in Algorithms 8 and 9 respectively. The control parameters (i.e., v1, v2, v3, v4, dir1, dir2, dir3, dir4) of Algorithm 7 only have to be modified according to Table 3 to create the remaining low-level functions for motor control, which in turn, translate to a possible

**Algorithm 6** Path tracking algorithm

```
Require: $position[lat_1, lon_1, \varphi]$            ▷ from Algorithm 5
Require: $lat_2, lon_2, \delta\theta, \delta\phi$ and $\delta\psi$    ▷ Inputs from user
Ensure: $\ell, \theta_1, \phi_1$ and $\psi$          ▷ Outputs for visualization
 1: function AUTO_NAVIGATE
 2:     while $dist_F \geq 120$ cm do
 3:         $r \leftarrow 6.399\text{x}10^7$ cm          ▷ Approx. earth radius
 4:         ▷ ——-Fetches compass/GPS sensor data——-
 5:         float $P_1[] \leftarrow$ POSITION_ESTIMATE
 6:         $\theta_1 \leftarrow P_1[lat_1]$
 7:         $\phi_1 \leftarrow P_1[on_1]$
 8:         $\varphi \leftarrow P_1[\varphi]$                    ▷ Robot yaw angle
 9:         ▷ ———Requests input data from user———
10:         float $P_2[] \leftarrow$ (Enter destination, $[lat_2, lon_2]$)
11:         $\theta_2 \leftarrow P_2[lat_2]$
12:         $\phi_2 \leftarrow P_2[lon_2]$
13:         ▷ ————Haversine Computations————
14:         $\Delta\phi \leftarrow (\phi_2 - \phi_1)$
15:         $\Delta\theta \leftarrow (\theta_2 - \theta_1)$
16:         $\mathbf{H} \leftarrow [\sin(\Delta\theta/2)]^2 + \cos\theta_1 * \cos\theta_2 * [\sin(\Delta\phi/2)]^2$
17:         $\ell \leftarrow 2 * r * \sin^{-1}(\sqrt{\mathbf{H}})$
18:         if $\ell \leq 5.0\text{x}10^4$ cm then
19:             $X \leftarrow \cos\theta_2 * \sin(\Delta\phi)$
20:             $Y \leftarrow \cos\theta_1 * \sin\theta_2 - \sin\theta_1 * \cos\theta_2 * \cos(\Delta\phi)$
21:                 $\leftarrow \tan^{-1}(X, Y)$
22:             ▷ ————Yaw control————
23:             while $\varphi \neq \psi \pm \delta\psi$ do
24:                 if $\psi \leq 180°$ then
25:                     call ROTATE_CW
26:                 else
27:                     call ROTATE_CCW
28:                 end if
29:             end while
                ▷ ————Position control————
30:             while $\varphi = \psi \pm \delta\psi$ do
31:                 if $P_1[\theta_1, \phi_1] \neq P_2[\theta_2 \pm \delta\theta, \phi_2 \pm \delta\phi]$ then
32:                     call MOVE_FWD               ▷ Move to target
33:                 else
34:                     call BRAKE                  ▷ Stop at target
35:                 end if
36:             end while
37:         end if
38:     end while
39: return $\ell, \theta_1, \phi_1, \psi$
40: end function
```

movement of the robot towards a desired direction, as required by Algorithm 1, 4 or 6. Also included in Table 3, are the redundant low-level functions, which could also be activated to modify the robot's motion. The integers 1, 2, and 4 in Table 3 are internally assigned to the parameters FORWARD, BACKWARD and RELEASE (for brake), respectively, in the motor driver library.

**Algorithm 7** Maneuvering algorithm: Motor Control Function

```
Require: Parameters from function calls
1: function MOTOR_CONTROL(v1, v2, v3, v4, dir1, dir2, dir3, dir4)
2:     M1.setSpeed (v1)          ▷ Sets motor1 rpm
3:     M2.setSpeed (v2)          ▷ Sets motor2 rpm
4:     M3.setSpeed (v3)          ▷ Sets motor3 rpm
5:     M4.setSpeed (v4)          ▷ Sets motor4 rpm
6:     M1.run (dir1)             ▷ Sets direction for Motor1
7:     M2.run (dir2)             ▷ Sets direction for Motor2
8:     M3.run (dir3)             ▷ Sets direction for Motor3
9:     M4.run (dir4)             ▷ Sets direction for Motor4
10: end function
```

**Algorithm 8** MOVE_FWD function
(FORWARD ← 1 in the motor driver's library)

```
1: function MOVE_FWD                          ▷ The function
2:     float $v \leftarrow dist_F$            ▷ Proximity value as speed parameter
3:     call MOTOR_CONTROL (v, v, v, v, 1, 1, 1, 1)
4: end function
5: call MOVE_FWD                              ▷ The function call
```

**Algorithm 9** ROTATE_CW function
(FORWARD ← 1 and BACKWARD ← 2 in the motor driver's library)

```
1: function ROTATE_CW                         ▷ The function
2:     float $v \leftarrow 250.00$            ▷ Sets motors speed to 250.00 rpm
3:     call MOTOR_CONTROL (v, v, v, v, 1, 2, 2, 1)
4: end function
5: call ROTATE_CW                             ▷ The function call
```

## 6. Results and Discussion

The navigational schemes and algorithms in Sections 4 and 5 are implemented using the experimental platform in Section 3. The Algorithms 2, 3, and 5 are encoded in Python programming language for execution on the companion computer as objected-oriented parallel programs. For low-level control, the

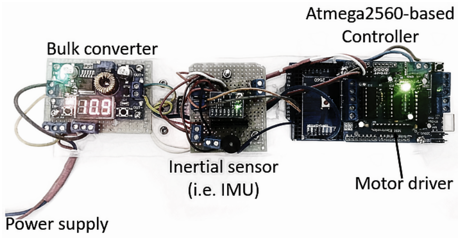

**Figure 7.** Configuration of the low-level controller (featuring the power supply, inertial sensor, main controller, motor driver, and their electrical connections)

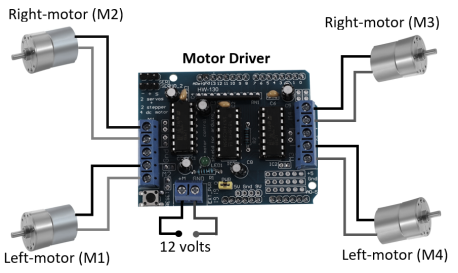

**Figure 8.** Connection of motors to the motor driver (the motor driver enables the main controller to control the speed and direction of the motors)

maneuvering functions (as described in Subsubsection (5.2)) and Algorithms 1, 4, and 6 are written in embedded C++ language and then uploaded onto the main controller to serve as functions for localized computation and real-time control of the robot's motion.

We conducted experiments in an open field as shown in Fig. 9 to test the validity of the mathematical models upon which the algorithms are based, and to evaluate the performance of the robot in the physical world. These involve the telemetry of navigational data to the remote data acquisition computer for real-time analysis. Our experimental procedure, analytical techniques, and the results are discussed below in Subsections 6.1 and 6.2.

### 6.1. Experimental procedure and results

As the navigational precision of our robotic system depends on the accuracy of its sensors, our field tests aim at evaluating the accuracy with which our robot could perform both obstacle avoidance and autonomous navigation, while maneuvering towards a target location.

**Evaluation of proximity measurement technique.** To ensure precision in obstacle avoidance, we evaluate the accuracy of each of the two adopted proximity sensors and the Algorithm 2. To do this, we plot the real-time values from the the ultrasonic sensor (i.e., $dist_{Ultrasonic}$), infrared sensor (i.e., $dist_{Infrared}$), and their fusion (i.e.,

**Table 2.** List of relevant motion control functions

| S/N | Function | Actuational effect |
|---|---|---|
| 1 | Move_fwd | Robot moves forward in a straight-line |
| 2 | Reverse | Robot moves backward in a straight-line |
| 3 | Rotate_cw | Robot yaws clockwise direction |
| 4 | Rotate_ccw | Robot yaws counter-clockwise direction |
| 5 | Brake | Robot stops moving |

**Table 3.** Full list of motion control functions and their parameters corresponding to Algorithm 7 (Note: v = $dist_F$, the redundant functions are in gray-colored texts)

| S/N | Function | v1 | v2 | v3 | v4 | dir1 | dir2 | dir3 | dir4 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Move_fwd | v | v | v | v | 1 | 1 | 1 | 1 |
| 2 | Reverse | 200 | 200 | 200 | 200 | 2 | 2 | 2 | 2 |
| 3 | Rotate_cw | 250 | 250 | 250 | 250 | 1 | 2 | 2 | 1 |
| 4 | Rotate_ccw | 250 | 250 | 250 | 250 | 2 | 1 | 1 | 2 |
| 5 | Brake | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 |
| 6 | Move_right | 250 | 50 | 50 | 250 | 1 | 1 | 1 | 1 |
| 7 | Move_left | 50 | 250 | 250 | 50 | 1 | 1 | 1 | 1 |
| 8 | Reverse_right | 250 | 50 | 50 | 250 | 2 | 2 | 2 | 2 |
| 9 | Reverse_left | 50 | 250 | 250 | 50 | 2 | 2 | 2 | 2 |
| 10 | Fast_fwd | 250 | 250 | 250 | 250 | 1 | 1 | 1 | 1 |
| 11 | Slow_fwd | 100 | 100 | 100 | 100 | 1 | 1 | 1 | 1 |

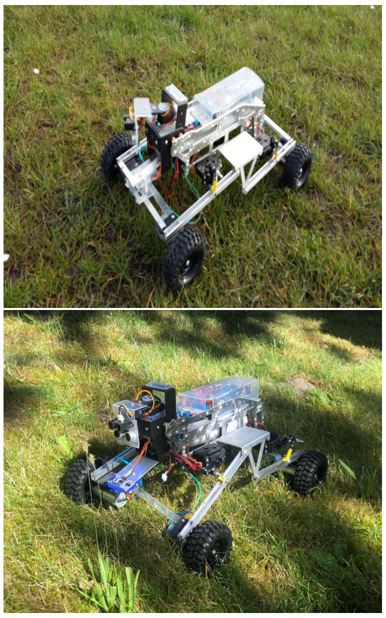
**Figure 9.** Performance evaluations in field tests

$dist_F$ from Algorithm 2) against time (in seconds). Also, we plot the estimate '$dist_F$' and the true proximity of the obstacle from the robot (i.e., $dist_{Actual}$, as measured with a meter-rule), against time (in seconds). The result of this experiment is presented in Fig. (10 and 11). This visualizes the error present in the robot's sensitivity to the proximity of obstacles along its path.

**Graphical visualization of the robot's routes.** To evaluate how accurately our robot could autonomously move to a target location (based on Definition 5.2), we visualized the actual navigational routes of our robot in comparison to the desired path, using real-time position estimates from Algorithm 5. For experimentation, variations in the navigational environment included the number of pre-stationed obstacles along the robot's path of travel and also, the length of the path. The results of this experiment are presented in Figs. 12 to 14.

## 6.2. Discussion

The plots in Fig. 10 shows how effectively Algorithm 2 fuses the measurement data from the ultrasonic and infrared sensor as $dist_F$. Based on Fig. 11, we observe that the fusion product ($dist_F$) is consist with the proximity values ($dist_{Actual}$), thus increasing the accuracy in the measurement of an obstacle's proximity from the robot, unlike the direct measurements from the individual sensors (i.e., $dist_{Ultrasonic}$ and $dist_{Infrared}$), which individually contains higher level of noise.

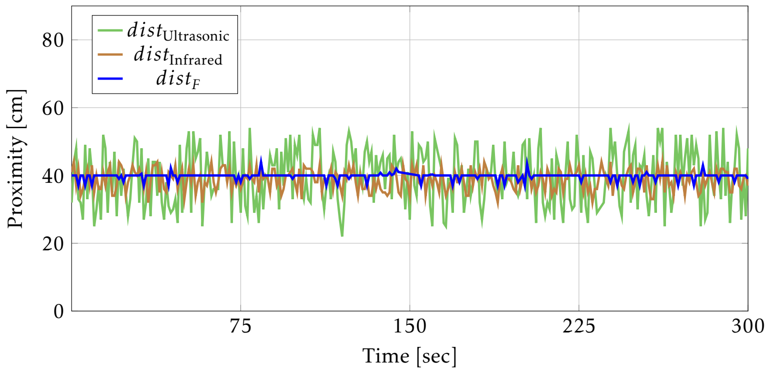

**Figure 10.** Proximity values (i.e., $dist_{\text{Ultrasonic}}$, $dist_{\text{Infrared}}$, and $dist_F$) vs time

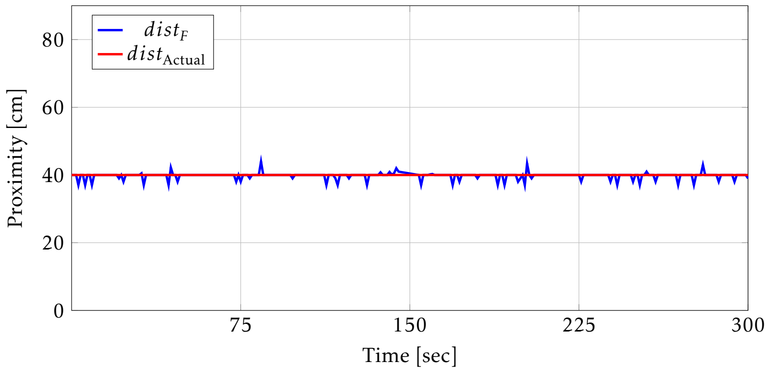

**Figure 11.** Proximity values ($dist_F$ and $dist_{\text{Actual}}$) vs time

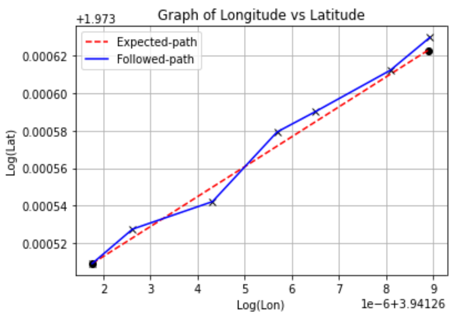

**Figure 12.** Path-plot 1: Graph visualizing the robot's routes (navigated distance ≈ 69.72 m, number of obstacles = 0)

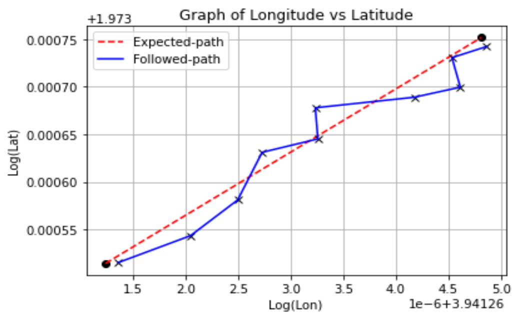

**Figure 13.** Path-plot 2: Graph visualizing the robot's routes (navigated distance ≈ 120.78 m, number of obstacles = 5)

According to our observations in Figs. 12 to 14, Algorithm 6 is effective, within the bounds of Definition 5.2, at homing the robot to a position near the target location (Note: the blue dots in the plot for the followed-path indicates the turning points, that is, the points at which the robot changes its direction of motion). In particular, the plot in Fig. 12 shows that in the absence of obstacles, the robot will move along a near-straight line from its starting point to the target location, which is evident in the relatively less number of turning points. Comparatively, the plots in Figs. 13 and 14 show that the motion of the robot to a target point will exhibit more turning points, deviations from the expected-path, and course-correction with increase in the range of the navigation path and the number of obstacles along the path.

**Limitations.** Based on our field tests observations, the technical possibility of flaws in the sensors measurements limit the performance of our robot. Other limitations arises from the topography of the terrains and the mechanical constraints of the robot's drive system. We observe that unlike on paved paths, the robot experiences transitional difficulties and

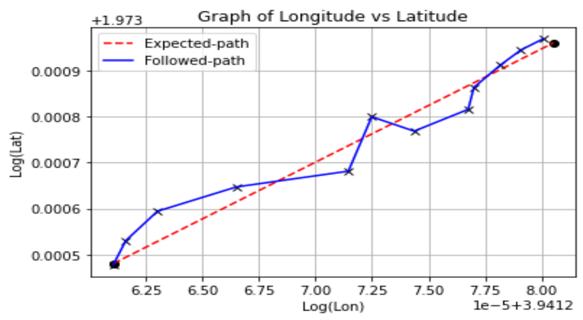

**Figure 14.** Path-plot 3: Graph visualizing the robot's routes (navigated distance ≈ 216.40 m, number of obstacles = 10)

inadequate steering power when maneuvering over rough terrains and grasslands; which is as a result of increased friction between the wheels and the ground. Apart from these constraints, the overall performance of our robot is satisfactory and meets the design objectives.

**Applications.** Potential applications of our autonomous robotic system include:

1. **Autonomous seed-planting:** Similar to the work of [18], the construct of our robotic system as well as the underlying auto-navigation algorithm could be applied in the development of an autonomous seed-planter for enhanced precision and efficiency in crop farming.

2. **Environmental monitoring:** Similar to the work of Olakanmi *et al.* in [23], our robotic system could be re-purposed as a multi-sensor surveillance system for environmental monitoring, especially in hazardous and industrial environments.

3. **Office-file movement:** Our robotic system could be developed into a semi-autonomous door-to-door file mover along passageways in a large office building.

4. **Home delivery:** Our robotic system could find application in logistics as a logistical robot that uses geospatial data and local beacon signals to autonomously navigate along streets and deliver merchandises to homes.

## 7. Conclusion

In this work, an edge-based autonomous robotic system is developed for point-to-point navigation using geospatial data. This system is able to avoid obstacles on its path to a target location using the fusion of proximity measurements of obstacles from itself, as detected by both the ultrasonic and infrared proximity sensor. In operation, several algorithms are used. These included the proximity sensor fusion algorithm for obstacle avoidance motion control and the localization algorithm for autonomous navigation of the robot to a target location. The embedded hardware of the robot comprises two edge-devices – the main controller and the companion computer, which work together as complimentary systems to implement the robot's control algorithms.

Several field tests were also conducted to evaluate the performance of the robot. These included the evaluation of how accurately the robot could detect obstacles along its travel path and maneuver them and also, how precisely it could get to its target location. Results show that our robot performs as expected, regardless of its operational constraints. In essence, our work proves that edge-devices like the microcontroller and SoC-computers are applicable to the development of intelligent and autonomous systems. Future developments in this area of research may explore the potential applications of our system; such as autonomous seed-planting, environmental monitoring, logistical automation etc., as mentioned in Section 6.2. Therefore, we hope that our work stimulates interest and enthusiasm, especially with regard to the practical applications of our robot and its corresponding algorithms.

## References

[1] Fodor, J. and Kacprzyk, J. (2009) Aspects of soft computing. *Intelligent Robotics and Control, Berlin, Heidelberg: Springer* .

[2] Campmany, V., Silva, S., E.A., Moure, J.C. and Vazquez, D., L.A. (2016) GPU-based pedestrian detection for autonomous driving. *Procedia Computer Science* **80**: 2377–2381.

[3] Sakr, F., Bellotti, F., Berta, R. and De Gloria, A. (2020) Machine learning on mainstream microcontrollers. *Sensors* **20**: 2638.

[4] Gratton, E., Benyeogor, M., Nnoli, K. and et al. (2020) Multi-terrain quadrupedal-wheeled robot mechanism: Design, modeling, and analysis. *European Journal of Engineering Research and Sciences* **5**: 24–33.

[5] Bolanakis, D.E. (2019) A survey of research in microcontroller education. In *in IEEE, Revista Iberoamericana de Tecnologias del Aprendizaje*.

[6] Stewart, D., Loucks, J., Casey, M. and Wigginton, C. (2019), Bringing AI to the device: Edge ai chips come into their own. tmt predictions 2020," deloitte insights, `https://www2.deloitte.com/us/en/insights/industry/technology/technology-media-and-telecom-predictions/2020/ai-chips.html`.

[7] Mamdoohi, G., Abas, A.F., Samsudin, K., Hidayat, A. and Mahdi, M.A. (2012) Implementation of genetic algorithm in an embedded microcontroller-based polarization control system. *Engineering Applications of Artificial Intelligence* **25**: 869–873.

[8] Hussain, I., Chakma, R., Ali, S. and Ali, I. (2017) A microcontroller based autonomous robot for navigating

and delivering purpose. In *in The Fifth International Academic Conference for Graduates, NUAA, Nanjing, China*.

[9] Mir-Nasiri, N., S., H.J. and Ali, M.H. (2018) Portable autonomous window cleaning robot. *Procedia Computer Science* **133**: 197–204.

[10] Apoorva, S., Chaithanya, R.S., Prabhu, S.B. and Shetty, D.D. (2017) Autonomous garbage collector robot. *International Journal of Internet of Things* **6**: 40–42.

[11] Efaz, E.T. (2018), Pathfinder: Design of an indicative featured and speed controlled obstacle avoiding robot, https://create.arduino.cc/projecthub/maverick/pathfinder-229d5d.

[12] Vukelic, B.M., Stancic, R. and Graova, S.G. (2013) Microcontroller based implementation of an integrated navigation system for ground vehicles. *IFAC Proceedings Volumes* **46**: 139–144.

[13] Parviainen, O. (2014) Software coven: OpenMP parallel computing in raspberry pi https://www.softwarecoven.com/parallel-computing-in-embedded-\mobile-devices/ Accessed on 14th April, 2021
.

[14] Chang, Y., Chung, P. and Lin, H. (2018) Deep learning for object identification in ros-based mobile robots. In *IEEE International Conference on Applied System Invention, Chiba*.

[15] Gasparetto, A., Boscariol, P., Lanzutti, A. and Vidoni, R. (2015) *Motion and Operation of Robotic System Chapter 1: Path Planning and Trajectory Planning Algorithms: A General Overview* (Springer International Publishing).

[16] Mohsin, O.Q. (2000), Mobile robot localization based on kalman filter: Dissertations and theses. paper 1529, Portland State University Library https://doi.org/10.15760/etd.1528.

[17] Bertoncelli, F., Ruggiero, F. and Sabattini, L. (2019) Wheel slip avoidance through a nonlinear model predictive control for object pushing with a mobile robot. *IFAC-PapersOnLine* **52**(8): 25 – 30. doi:https://doi.org/10.1016/j.ifacol.2019.08.043. 10th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2019.

[18] Fernandez, B., Herrera, P.J. and Cerrada, J.A. (2019) A simplified optimal path following controller for an agricultural skid-steering robot. *IEEE Access* **7**: 95932–95940.

[19] Gerdan, G.P. and Deakin, R.E. (1999) Transforming cartesian coordinates x,y,z to geographical coordinates , , h. *The Australian Surveyor* **44**(1): 55–63.

[20] Claessens, S. (2019) Efficient transformation from cartesian to geodetic coordinates. *Computers Geosciences* **133**: 104–307. doi:https://doi.org/10.1016/j.cageo.2019.104307.

[21] Atsushi Sakai, e.a. (2018), Pythonrobotics: A python code collection of robotics algorithms. https://arxiv.org/pdf/1808.10703v1.pdf.

[22] Faulkner, P. (2004) Notes for applied mathematics in trigonometry and earth geometry/navigation. *Australian Senior Mathematics Journal* **18**(1): 55 – 58.

[23] Olakanmi, O.O. and Benyeogor, M.S. (2019) Internet based tele-autonomous vehicle system with beyond line-of-sight capability for remote sensing and monitoring. *Internet of Things* **5**: 97–115. https://www.doi.org/10.1016/j.iot.2018.12.003.