Run-time Modelling of Energy Consumption in Mobile Robots

Technical Report - 2024

Abdul Malik
Hashem Haghbayan
Juha Plosila
Department of Computing
University of Turku
September 11, 2024

Abstract

In recent years, the deployment of mobile robots in various industries, from manufacturing to healthcare, has seen a significant rise. This necessitates an in-depth understanding of their energy consumption patterns to optimize efficiency and sustainability. This report presents a comprehensive platform developed to measure and analyze the electric current consumed by both the computational and mechanical components of mobile robots. At first, the project focuses on developing a mobile rover by integrating NVIDIA Jetson Nano, Pixhawk 4 Flight Controller along PM07 Power management Board on a Reely rover. Secondly, a runtime current measuring system has been developed using HSTS016L current sensor, ADS1115 ADC along with NVIDIA Jetson Nano. The primary aim of this study is to provide a detailed and accurate model for run-time energy consumption measurement. Additionally, this report covers the methodology for measuring system dynamics i.e. distance, velocity, acceleration and jerk using GPS data, processed through both the Haversine method and Python's Pyproj library. This dual-method approach enhances the accuracy of distance measurement, crucial for correlating energy consumption with the rover's operational parameters. The study can be further extended to study real-time energy consumption of autonomous robots in different autonomous applications.

Keywords: Energy Consumption, Mobile Robots, Jetson Nano, System Dynamics

---

Permanent URL address: `https://urn.fi/URN:ISBN:978-951-29-9877-7`

Note: The source code used in this project is open-source and available on GitHub. You can find it at the following link:
https://gitlab.utu.fi/asl/run-time-modelling-of-energy-consumption-in-mobile-robots

Contents

Chapter 1

Introduction

The rapid advancement and deployment of mobile robots in various applications—from manufacturing [9] and logistics [11] to healthcare [3] service industries [1] and defense [2] has underscored the need for efficient energy management. Mobile robots, which combine mechanical components and computational units, are integral to the automation of numerous processes. However, the energy consumption of these robots significantly impacts their operational efficiency, cost-effectiveness, and environmental footprint [12]. Thus, a detailed understanding and optimization of their energy use are crucial.
This report presents in detail the development of a mobile rover by integrating NVIDIA Jetson Nano for onboard processing and autonomous applications. Furthermore, it describes a platform designed to measure and analyze the runtime energy consumption of mobile robots. The primary focus is on accurately capturing the electric current consumed by both the computational and mechanical components during operation. By extending this platform to mobile rover, we aim to provide a comprehensive model that facilitates real-time monitoring and optimization of energy consumption.

1.1 Objectives

1. Development of Mobile Rover: Integrate key components such as NVIDIA Jetson Nano for computational and Pixhawk4 flight controller, PM07 power management board for navigation and control to the Reely Rover consisted of KV3000 motor, WB-10BL50-RTR electronic speed controller.
2. Development of Current Measuring System: Integrate the HSTS016L current sensors for current measurement, ADS1115 ADC for data conversion and establish communication with Jetson Nano to obtain data for further processing.
3. Measurement and Analysis: Develop a system to measure the electric current consumed by the computational and mechanical parts of mobile robots accurately and calculate the power consumption.
4. Calibration and Validation: Calibrate the current sensors to reduce noise and validate their measurements using a reliable multi meter.
5. System Dynamics Measurement: Utilize GPS data to measure the distance travelled, speed and acceleration/jerk of the mobile rover, employing both the Haversine method and Pyproj library for increased accuracy.
6. Real-time Data Logging: Develop a Python script to handle real-time data logging, calibration, and analysis of energy consumption.

1.2 Significance

The significance of this research lies in its potential to improve the operational efficiency of mobile robots. By providing a detailed analysis of energy consumption, the developed platform allows for the identification of high-energy-consuming components and operations. This insight is crucial for optimizing the design and operation of mobile robots, leading to enhanced performance, reduced operational costs, and a lower environmental impact.

1.3 Scope

The scope of this project encompasses the following:
1. Component Integration: Detailed integration of various sensors, computational units, and navigation systems into a mobile rover.
2. Current Measurement: Accurate measurement of the electric current consumed by the rover's components.
3. System Dynamics Measurement: Implementation of GPS-based methods for precise distance and speed calculations.
4. Data Analysis: Real-time logging and analysis of data to provide actionable insights into energy consumption patterns.

Chapter 2

Mobile Rover Development

2.1 Components

2.1.1 Reely Stagger Rover

The Reely Stagger Rover is a robust and high-performance RC vehicle designed for enthusiasts seeking power and precision. It features a Power Management Board, ensuring efficient energy distribution and management. The Reely 538527C KV3000 motor delivers high-speed performance with exceptional torque. The WB-10BL50-RTR electronic speed controller provides smooth acceleration and reliable control. Its power source is a LiPo 3500 mAh 7.4V max battery, offering long-lasting run times. Control is handled by the FS-I6 remote controller and FS-IB6 receiver, offering precise and responsive handling for an unparalleled driving experience.

Figure 2.1: Reely Stagger Rover (left) and Flysky Remote Controller(right)

2.1.2 NVIDIA Jetson Nano

The NVIDIA Jetson Nano is a compact, powerful computer designed for AI development, capable of running multiple neural networks in parallel for applications like image classification, object detection, and speech processing. [5]

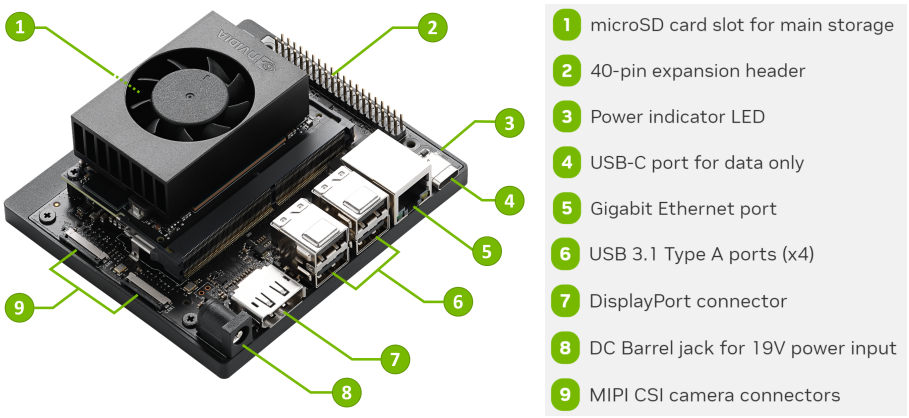


Figure 2.2: NVIDIA Jetson Nano

2.1.3 Pixhawk 4 and PX4 GPS Module

The Pixhawk 4 is an advanced autopilot system designed for drones and other unmanned vehicles, featuring a powerful processor, multiple sensor integration, and versatile connectivity options for precise control and navigation. Paired with the PX4 GPS Module, it provides high-accuracy positioning and reliable GPS data, enhancing the overall performance and stability of autonomous systems. [7]

Figure 2.3: Pixhawk 4 Flight Controller (left) and PX4 GPS Module (right)

2.1.4 PX4 Power Module PM07

The PM07 Power Management Board is a specialized component used primarily in drones and other unmanned vehicles to ensure efficient power distribution and monitoring. It is designed to handle high current loads, providing a stable power supply to the flight controller, electronic speed controllers (ESCs), and other onboard systems.[8]

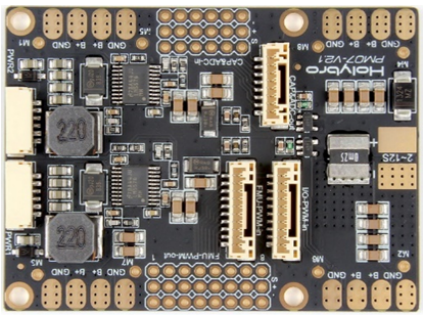


Figure 2.4: Power Module PM07

2.1.5 FS-IA6B Receiver

The FS-IA6B receiver is a 6-channel, 2.4GHz radio receiver commonly used in RC (radio-controlled) models such as drones, airplanes, and cars. It supports PPM and iBUS protocols, allowing for easy integration with various flight controllers.

Figure 2.5: FS-IA6B Receiver

2.2 PX4 and J-Nano Connection Test using Python Script

DroneKit is a Python library basically made to communicate with and control drones, particularly those using the ArduPilot flight control system. With DroneKit, we can write simple Python scripts to automate drone tasks like takeoff, navigation, and landing. It allows to access PX4's telemetry, control its movement, and even set up autonomous missions without needing to dig into low-level programming. We can use Dronekit library to control rover, as we can connect and communicate the PM07 power management board via PX4's "I/O PWM Out" pin. At first we do verify connection and arm the rover. Secondly, we perform a test to check connection and control of rover's motor and servo.

2.2.1 Rover Connection Test and Arming

The Python script connects to a drone using the DroneKit library and prepares it for flight. It establishes a connection to the drone through MAVLink, checks that the drone is ready, sets it to "GUIDED" mode, and then arms it. The script is designed to ensure that the drone is in a proper state before allowing it to take off or perform other guided maneuvers.
(See Appendix .1)

2.2.2 Rover Motor and Servo Test

The Python script connects to a drone using the DroneKit library and performs various operations by sending PWM signals to control motors and servos. After establishing a connection, the script prints the drone's basic information and then sequentially sets motor and servo PWM values to control their movement. It tests motor's forward and backward movement as well as servo's left and right turning. It also clears all channel overrides before closing the connection to ensure the vehicle is safely stopped.
(See Appendix .2)

2.3 Schematics

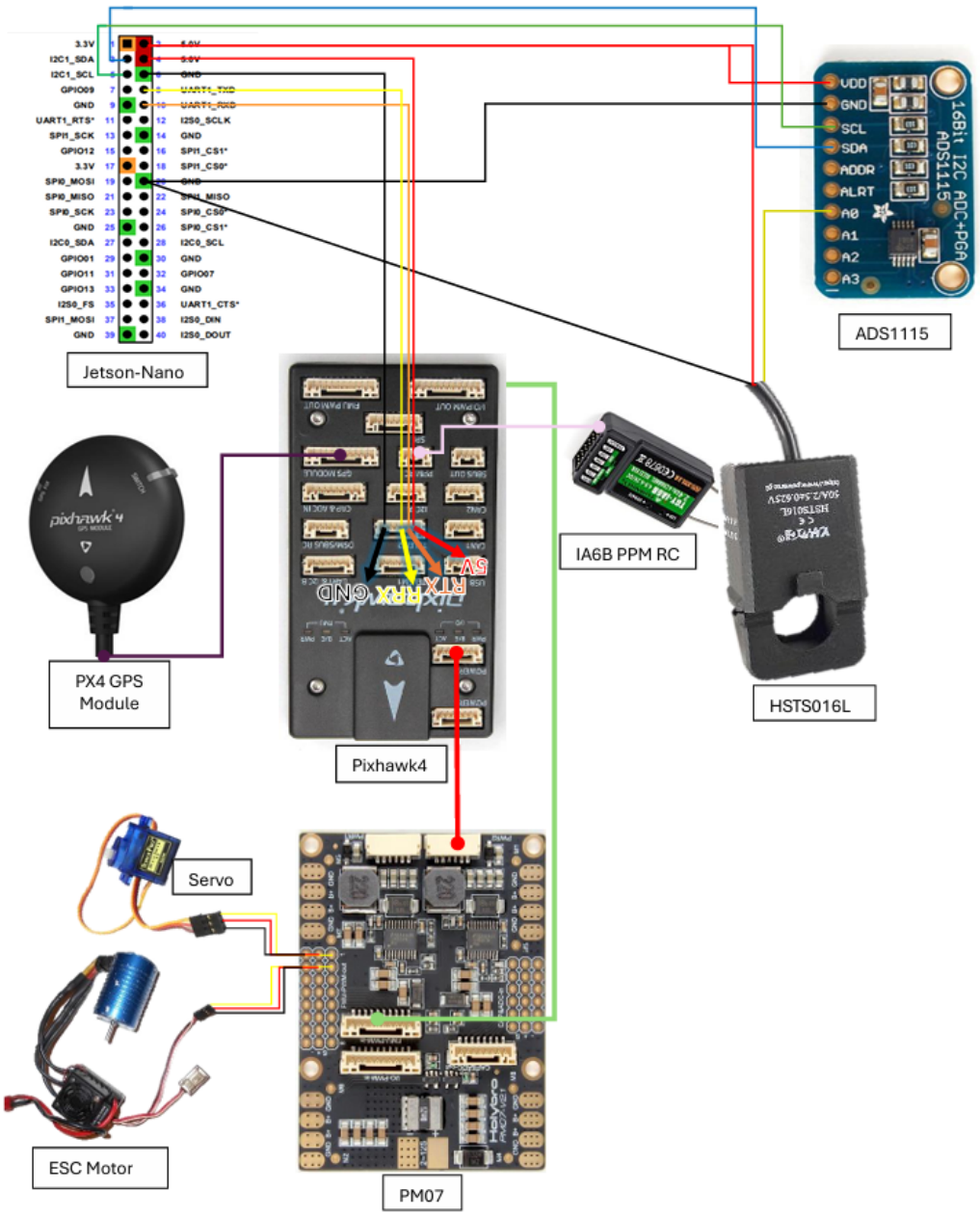


Figure 2.6: Mobile Rover Schematics

Chapter 3

Current Measurement System

3.1 Components

3.1.1 HSTS016L 30A/2.5V±0.625V

The HSTS016L 30A/2.5V±0.625V is a highly accurate Hall effect current sensor designed to measure DC and AC currents. The one we are using measures current up to 30A with excellent linearity.[6]

Figure 3.1: HSTS016L 30A/2.5V±0.625V Current Sensor

3.1.2 ADS1115 ADC 16-Bit

The ADS1115 is a precision 16-bit analog-to-digital converter (ADC) with an integrated programmable gain amplifier (PGA), suitable for high-resolution measurement applications. It offers four single-ended or two differential inputs, operates over a wide power supply range, and provides an easy-to-use I2C interface for seamless integration with various microcontrollers.

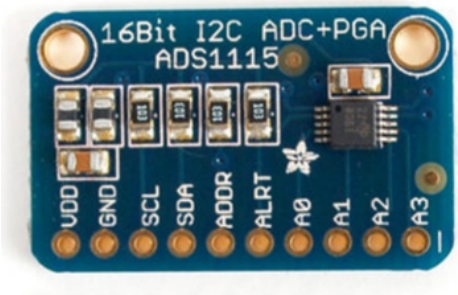


Figure 3.2: ADS1115 ADC 16-Bit

3.1.3 Multimeter UT203R

We are using Multimeter UT203 for validation of the current measured by HSTS016L. The UT203R is a versatile digital clamp Multimeter designed for measuring AC/DC voltage, AC current, resistance, and continuity with high accuracy.[10]

Figure 3.3: Multimeter UT203R

3.2 Schematics Current Measurement

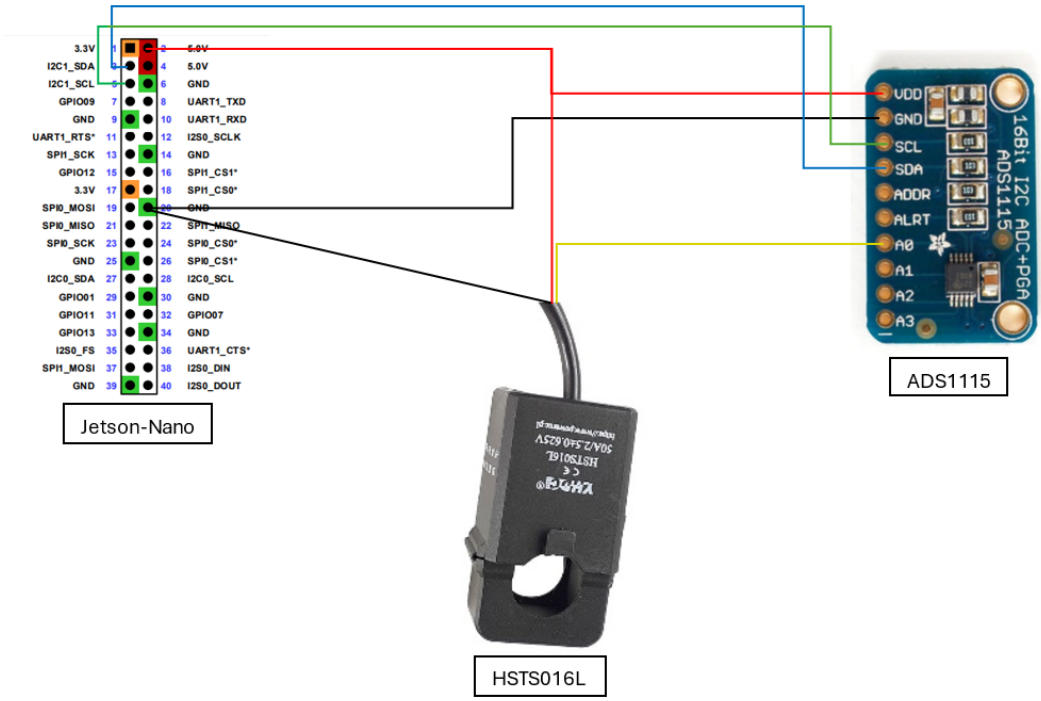


Figure 3.4: Current Measurement Schematics

3.3 Calculations

We are using HSTS016L 30A/2.5V±0.625V sensor that uses Hall effect to sense and measure current. As the range of the sensor is 30A/2.5±0.625V which means the max output voltage will be 3.125(2.5+0.625) against 30A forward current and min output voltage will be 1.875V(2.5-0.625) against 30A current in reverse direction.

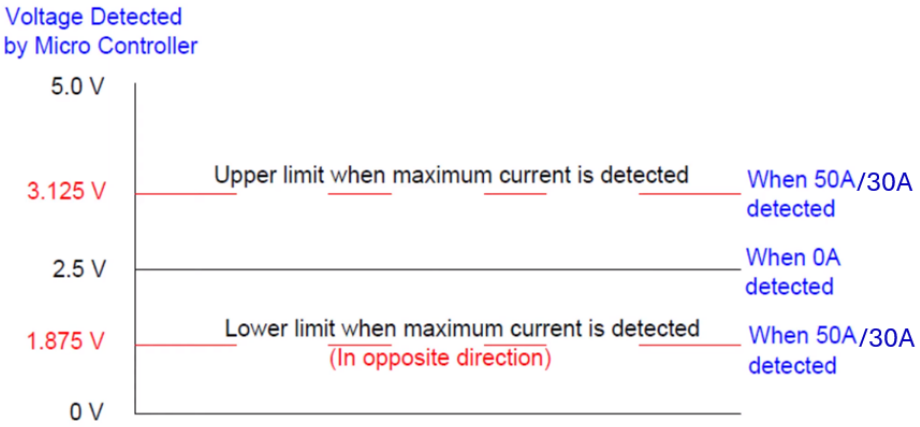


Figure 3.5: HSTS016L Sensor Output Diagram

$$\text{Voltage} = 2.5V \pm 0.625V \tag{3.1}$$

$$V_{\text{ZERO}} = 2.5 \tag{3.2}$$

$$\text{Rated Input} = \pm 30A \tag{3.3}$$

$$\text{SENSITIVITY} = \frac{\text{Voltage}}{\text{Rated Input}} \tag{3.4}$$

$$\text{SENSITIVITY of 30A Sensor} = 0.02083\,\text{V/A} \quad \left(\frac{0.625V}{30A}\right) \tag{3.5}$$

$$\text{Current} = \frac{\text{Voltage} - V_{\text{ZERO}}}{\text{SENSITIVITY}} \tag{3.6}$$

$$\tag{3.7}$$

We are using ADS1115 analogue to digital converter to convert analogue output of the current sensor HSTS016L. With a python code running on Jetson-Nano we are processing the Voltage Input from the ADS1115 to get the run-time current value drawn from the battery or being consumed by the motor of the robot. We can print these values run-time in the csv file with the help of Python code with which we can further process the current output for the calculation of the energy consumption of the mobile robot.

3.4 Python code to measure voltage and current

The Python script reads voltage and current data from HSTS016L sensor using an ADS1115 analog-to-digital converter (ADC). It initializes the ADS1115 on I2C bus 1 with address 0x48 and sets the gain to measure up to ±4.096V. The script continuously reads raw ADC values, converts them to voltage, and calculates the current based on a

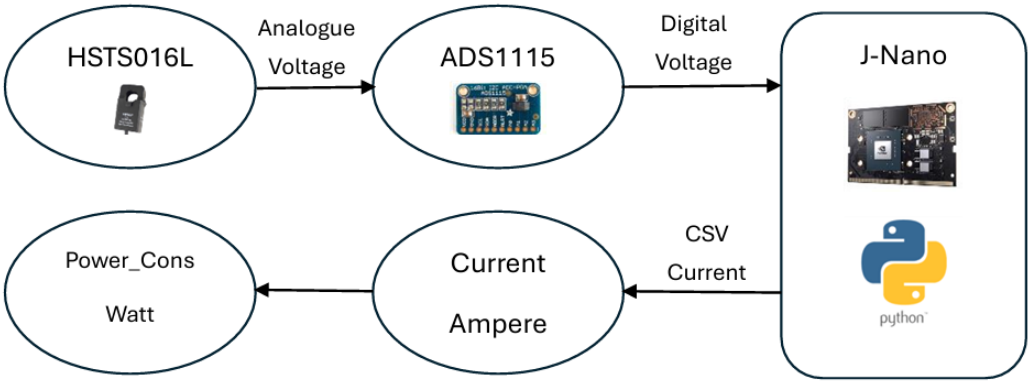


Figure 3.6: Current Measurement Diagram

predefined sensitivity and zero-current voltage. The results are printed every 0.1 seconds to monitor real-time voltage and current readings.
(See Appendix .3)

3.5 Calibration of the Current Measurement

As HSTS016L sensor uses Hall effect to sense and measure current. We observe some noise even when there is no wire between the clamps. The same behaviour is observed with the Multimeter which shows -0.20 to -0.40 A values as noise. There are multiple reasons for this behaviour from sensors based on Hall effect.
We have used 3 different HSTS016L sensors to observe the behaviour and calibrate the sensors. We recorded 50 measurements and below is given the statistical outcome of the observations.

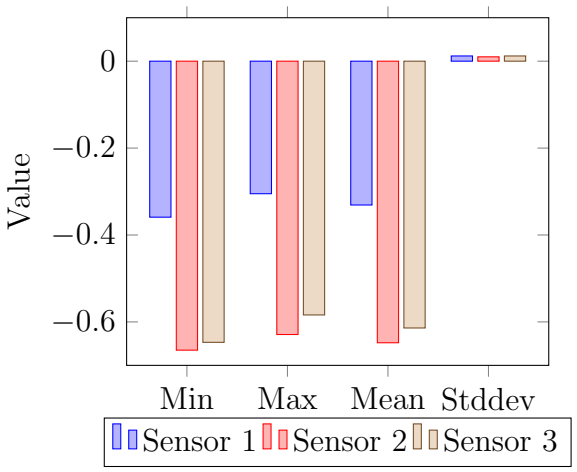
Table 3.1: Noise Data for Three Sensors

Noise	**Sensor 1**	**Sensor 2**	**Sensor 3**
Min	-0.359	-0.665	-0.647
Max	-0.305	-0.629	-0.584
Mean	-0.331	-0.648	-0.614
Standard dev	0.012	0.010	0.012

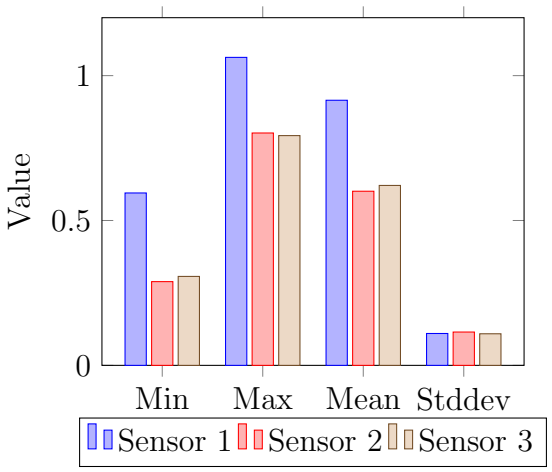
Again, we used all 3 sensors on a wire simultaneously and recorded 50 observations. Below are the statistical outcomes of the observations.

Table 3.2: Observation Data for Three Sensors

Observations	**Sensor 1**	**Sensor 2**	**Sensor 3**
Min	0.595	0.289	0.307
Max	1.063	0.802	0.793
Mean	0.915	0.601	0.621
Standard dev	0.110	0.115	0.109



(a) Bar Chart of Noise Data



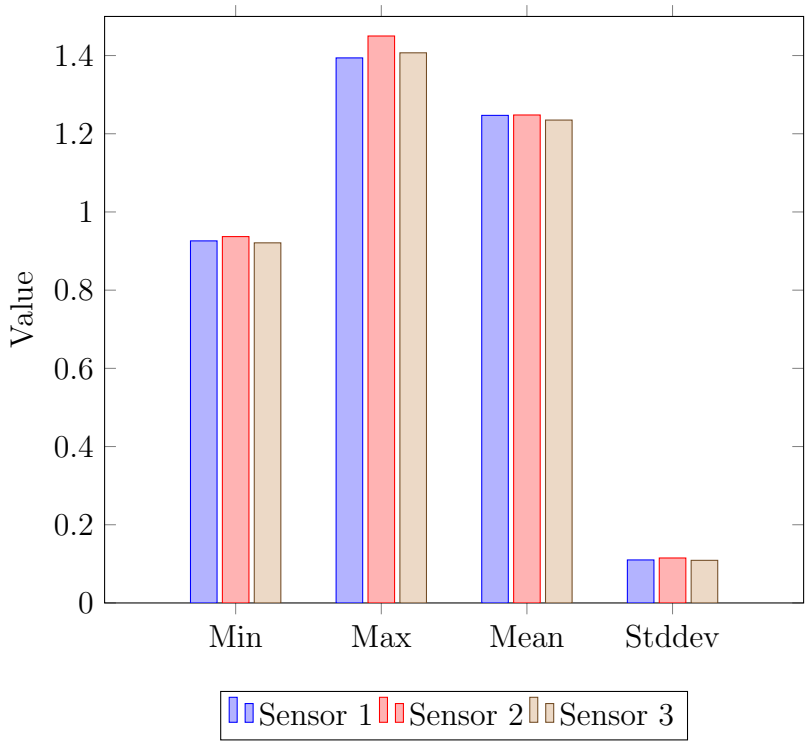
(b) Bar Chart of Observation Data

Figure 3.7: Comparison of Noise and Observation Data for Three Sensors

For the calibration purpose we subtracted the average of noise (measurements without wire) from each instance and below are the statistical outcomes.After the Calibration values are quite reliable and shows almost equivalent values.

Table 3.3: Calibrated Observations Data for Sensors

Calibrated Observations	**Sensor 1**	**Sensor 2**	**Sensor 3**
Min	0.926	0.937	0.921
Max	1.394	1.450	1.407
Mean	1.247	1.248	1.235
Standard dev	0.110	0.115	0.109



Chapter 4

System Dynamics (Distance, Velocity and Acceleration) Measurement

System dynamics are important aspects to to be considered in mobile robots and their power consumption. As we aim to measure the trends of energy consumption against velocity and acceleration of the rover so have used two methods to calculate distance covered by rover using GPS data; one is Haversine method and other is Python's Pyproj library. We will briefly explain both below.

4.1 Haversine Method

The Haversine method is used to calculate the great-circle distance between two points on the Earth's surface given their latitude and longitude coordinates. This method accounts for the spherical shape of the Earth, providing accurate distances over large areas.[4]

Haversine Formula: The formula is given by:

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{4.1}$$

$$c = 2 \cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{1-a}\right) \tag{4.2}$$

$$d = R \cdot c \tag{4.3}$$

Where:

- ϕ_1, ϕ_2 are the latitudes of the two points in radians.
- λ_1, λ_2 are the longitudes of the two points in radians.
- $\Delta\phi = \phi_2 - \phi_1$ is the difference in latitude.
- $\Delta\lambda = \lambda_2 - \lambda_1$ is the difference in longitude.
- R is the Earth's radius (mean radius = 6,371 km).
- d is the distance between the two points.

4.2 Pyproj Library

The Pyproj library in Python provides tools for performing coordinate transformations and geodetic calculations, making it essential for converting GPS data into various coordinate systems. We have used "transform" function from Pyproj library to convert WGS84(GPS coordinate system) coordinates to ECEF(Earth-Centered, Earth-Fixed) coordinates resulting in Cartesian coordinates x, y, and z. Once we have the cartesian coordinates we can use Euclidean distance formula to calculate the distance between with points.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{4.4}$$

After getting the distance between two points against time steps we can calculate the speed and acceleration of the rover. Further, we can sum up the distance covered in each instance to get total distance covered by the rover over the time.

4.3 Python code to measure current and system dynamics

We are using Python program to record and store the current, GPS coordinates and distance covered against the time steps into a "csv" file format using both "haversine" and "Pyproj" methods to have a comparison of both methods.
(See Appendix .4)

4.4 Observations

As we have used 2 methods to calculate the distance. We have noticed minor fluctuations in the GPS coordinates even when the rover is stationary. Which have been resulting in some distance being covered every instant we take recording. As the GPS module we have been using had an accuracy of 2.5m so it does not suit for very short distance measurement.
Below are given the observations measured while rover being stationary for 90 seconds and observations were taken every half second:

	Haversine Distance	**Pyproj Distance**
Min (cm)	0	0
Max (cm)	6.460692122	8.206047218
Mean (cm)	1.651890789	2.6901064
Stdev	0.01190920657	0.01496427356
Sum (cm)	267.6063078	435.7972368

Table 4.1: Comparison of Haversine and Pyproj Distance Measurements

We run the rover on a track of 30 meters and took observation each half second for runtime monitoring and below are the results based on both methods used for distance calculations:

As we see the total distance is much deviated from the actual distance. The reason is that we have been taking observations in very short interval and considering the GPS

	hav_Dist (m)	**hav_Speed (m/s)**	**hav_Acc (m/s^2)**	**pyproj_Dist (m)**	**pyproj_Speed (m/s)**	**pyproj_Acc (m/s^2)**
Min	0.00000	0.00000	-0.78650	0.00000	0.00000	-0.94822
Max	0.60730	1.08200	1.10072	0.67039	1.19441	1.12178
Mean	0.14403	0.25641	0.02203	0.15963	0.28417	0.02242
Stdev	0.13167	0.23422	0.31760	0.14451	0.25704	0.34426
Total	7.48963			8.30064		

Table 4.2: Comparison of Haversine and Pyproj Distance, Speed, and Acceleration Measurements

module's accuracy which is 2.5m, velocity accuracy 0.05 m/s, latency in signals and noise factors GPS do not seems to be right option for this purpose on such a short distance.

Chapter 5

Conclusion

We successfully developed a platform that accurately measures the real-time current consumption of both the computational and mechanical components of a mobile robot. This platform allows for precise calculation of power and energy consumption, providing valuable insights into the robot's overall efficiency. Additionally, we implemented two methods— the Haversine formula and Python's Pyproj library— to measure the distance covered by the rover using GPS data over time, enabling the calculation of key system dynamics such as distance, speed, acceleration, and jerk.

However, these methods showed limitations in short-distance and indoor environments due to the inherent inaccuracy of GPS, particularly for precise measurements. To address this limitation, further development is recommended, including the integration of an Inertial Measurement Unit (IMU) and Visual Inertial Odometry (VIO). These enhancements would significantly improve the accuracy of system dynamics measurements, especially in GPS-denied environments.

This platform serves as a foundation for further research into mobile robot energy consumption trends. Future work could focus on analyzing the energy usage of mechanical, computational, and communication systems in relation to the robot's system dynamics. This would provide a deeper understanding of how different components and operations contribute to overall energy consumption, allowing for better optimization of mobile robotic systems and task distribution.

Appendix

.1 Rover Connection Test and Arming

```
##########DEPENDENCIES#############

from dronekit import connect, VehicleMode, LocationGlobalRelative, APIException
import time
import socket
import math
import argparse

#########FUNCTIONS#################

def connectMyCopter():
    parser = argparse.ArgumentParser(description='commands')
    parser.add_argument('--connect')
    args = parser.parse_args()

    connection_string = args.connect

    vehicle = connect(connection_string, baud=57600, wait_ready=True)

    return vehicle

def arm(vehicle):
    while not vehicle.is_armable:
        print("Waiting for vehicle to become armable.")
        time.sleep(1)
    print("Vehicle is now armable")

    vehicle.mode = VehicleMode("GUIDED")

    while vehicle.mode != 'GUIDED':
        print("Waiting for drone to enter GUIDED flight mode")
        time.sleep(1)
    print("Vehicle now in GUIDED MODE. Have fun!!")

    vehicle.armed = True
    while not vehicle.armed:
        print("Waiting for vehicle to become armed.")
        time.sleep(1)
    print("Vehicle is now armed.")

    return None

##########MAIN EXECUTABLE###########

vehicle = connectMyCopter()

vehicle.wait_ready('autopilot_version')
print('Autopilot version: %s' % vehicle.version)

arm(vehicle)
```

Listing 1: Python code to assure rover's controller connection to J-Nano

.2 Rover Motor and Servo Test

```
##########DEPENDENCIES#############
from dronekit import connect
import time
```

```python
import argparse
from pymavlink import mavutil

#########FUNCTIONS#################

def connectMyCopter():
    parser = argparse.ArgumentParser(description='commands')
    parser.add_argument('--connect')
    args = parser.parse_args()

    connection_string = args.connect
    vehicle = connect(connection_string, baud=57600, wait_ready=True)

    return vehicle

def print_vehicle_info(vehicle):
    print("Vehicle connected")
    print(f"Vehicle mode: {vehicle.mode.name}")
    print(f"Armed status: {vehicle.armed}")
    print(f"Vehicle type: {vehicle._vehicle_type}")

def set_motor(vehicle, pwm_value):
    print(f"Setting motor to PWM value: {pwm_value}")
    vehicle.channels.overrides[2] = pwm_value
    time.sleep(5)

def set_servo(vehicle, pwm_value):
    print(f"Setting servo to PWM value: {pwm_value}")
    vehicle.channels.overrides[1] = pwm_value
    time.sleep(5)

def clear_overrides(vehicle):
    print("Clearing all channel overrides.")
    vehicle.channels.overrides = {}

##########MAIN EXECUTABLE###########

vehicle = connectMyCopter()

# Print vehicle info
print_vehicle_info(vehicle)

# Run motor forward for 5 seconds
set_motor(vehicle, 1550)  # Adjust PWM value as needed

time.sleep(5)

# Run motor backward for 5 seconds
set_motor(vehicle, 1400)  # Adjust PWM value as needed

time.sleep(5)

# Center the motor (stop)
set_motor(vehicle, 1500)  # Center PWM value

time.sleep(5)


# Move servo to one side for 5 seconds
set_servo(vehicle, 2000)  # Adjust PWM value as needed

# Move servo to the other side for 5 seconds
set_servo(vehicle, 1000)  # Adjust PWM value as needed

# Center the servo
set_servo(vehicle, 1500)  # Center PWM value

# Clear all overrides
clear_overrides(vehicle)

# Close vehicle object before exiting script
vehicle.close()
```

Listing 2: Python code to test motor and servo

.3 Python code to measure voltage and current

```
import os
import time
import ADS1x15

ADS = ADS1x15.ADS1115(1, 0x48)

print(os.path.basename(__file__))

# Set gain to +/- 4.096V (1 bit = 0.125mV)
ADS.setGain(0)

print("Voltage and Current")

# Sensor characteristics
V_ZERO = 2.5  # Voltage at 0A
#SENSITIVITY = 0.0125  # V/A (0.625V / 50A)
SENSITIVITY = 0.02083  # V/A (0.625V / 30A)

while True:
    raw = ADS.readADC(0)
    voltage = ADS.toVoltage(raw)
    current = (voltage - V_ZERO) / SENSITIVITY
    print("Voltage: {0:.5f} V, Current: {1:.3f} A".format(voltage, current))
    time.sleep(0.1)
```

Listing 3: Python code to measure voltage and current using ADS1115

.4 Python code to measure current and system dynamics

```
import os
import time
from ADS1x15 import ADS1115
from pymavlink import mavutil
from datetime import datetime
import math
import csv
from pyproj import Proj, transform
import numpy as np

# Initialize the ADS1115 ADC
ADS = ADS1115(1, 0x48)
ADS.setGain(0)  # Set gain to +/- 4.096V (1 bit = 0.125mV)

# Sensor characteristics
V_ZERO = 2.5  # Voltage at 0A
SENSITIVITY = 0.02083  # V/A (0.625V / 30A)

# UART Port, Baud Rate and connection
telemetry_port = '/dev/ttyTHS1'  # Replace with the correct port
baud_rate = 57600
connection = mavutil.mavlink_connection(telemetry_port, baud=baud_rate)

# Print the header
print("Time (HH:MM:SS), Current (A), Latitude, Longitude, Altitude (m), Hav_Distance (m)
    , Hav_Speed (m/s), Hav_Total_Distance (m), Hav_Acceleration (m/s^2), Pyproj_Distance
     (m), Pyproj_Speed (m/s), Pyproj_Total_Distance (m), Pyproj_Acceleration (m/s^2)")

# CSV file setup
csv_file = 'haversine_vs_pyproj.csv'
with open(csv_file, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Time", "Current (A)", "Latitude", "Longitude", "Altitude (m)",
                     "Hav_Distance (m)", "Hav_Speed (m/s)", "Hav_Total_Distance (m)", "
                         Hav_Acceleration (m/s^2)",
                     "Pyproj_Distance (m)", "Pyproj_Speed (m/s)", "Pyproj_Total_Distance
                         (m)", "Pyproj_Acceleration (m/s^2)"])
# Initialize previous GPS coordinates for both methods
previous_lat = None
previous_lon = None
previous_time = None
hav_previous_speed = 0.0
hav_total_distance = 0.0

pyproj_previous_lat = None
```

```
pyproj_previous_lon = None
pyproj_previous_alt = None
pyproj_previous_time = None
pyproj_previous_speed = None
pyproj_total_distance = 0.0

# Define the WGS84 coordinate system (latitude, longitude, altitude)
wgs84 = Proj(init='epsg:4326')

# Define the ECEF (Earth-Centered, Earth-Fixed) coordinate system
ecef = Proj(proj='geocent', ellps='WGS84', datum='WGS84')

def read_current():
    raw = ADS.readADC(0)
    voltage = ADS.toVoltage(raw)
    current = (voltage - V_ZERO) / SENSITIVITY
    return current

def read_gps():
    msg = connection.recv_match(type='GPS_RAW_INT', blocking=True, timeout=1)
    if msg:
        lat = msg.lat / 1e7  # Latitude in degrees
        lon = msg.lon / 1e7  # Longitude in degrees
        alt = msg.alt / 1e3  # Altitude in meters
        return lat, lon, alt
    else:
        return None, None, None

def haversine(lat1, lon1, lat2, lon2):
    R = 6371000  # Radius of the Earth in meters
    phi1 = math.radians(lat1)
    phi2 = math.radians(lat2)
    delta_phi = math.radians(lat2 - lat1)
    delta_lambda = math.radians(lon2 - lon1)
    a = math.sin(delta_phi / 2.0) ** 2 + math.cos(phi1) * math.cos(phi2) * math.sin(
        delta_lambda / 2.0) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    distance = R * c  # Distance in meters
    return distance

def gps_to_ecef(lat, lon, alt):
    x, y, z = transform(wgs84, ecef, lon, lat, alt)
    return np.array([x, y, z])

while True:
    current = read_current()
    lat, lon, alt = read_gps()
    now = datetime.now()
    timestamp = now.strftime("%H:%M:%S")

    if lat is not None and lon is not None and alt is not None:
        if previous_lat is not None and previous_lon is not None:
            # Haversine method calculations
            hav_distance = haversine(previous_lat, previous_lon, lat, lon)
            hav_total_distance += hav_distance  # Accumulate the distance
            time_diff = (now - previous_time).total_seconds()
            hav_speed = hav_distance / time_diff if time_diff > 0 else 0.0
            hav_acceleration = (hav_speed - hav_previous_speed) / time_diff if time_diff
                > 0 else 0.0

            # Pyproj method calculations
            current_xyz = gps_to_ecef(lat, lon, alt)
            previous_xyz = gps_to_ecef(pyproj_previous_lat, pyproj_previous_lon,
                pyproj_previous_alt)
            pyproj_distance = np.linalg.norm(current_xyz - previous_xyz)
            pyproj_total_distance += pyproj_distance  # Accumulate the distance
            pyproj_speed = pyproj_distance / time_diff if time_diff > 0 else 0.0
            if pyproj_previous_speed is not None:
                pyproj_acceleration = (pyproj_speed - pyproj_previous_speed) / time_diff
                    if time_diff > 0 else 0.0
            else:
                pyproj_acceleration = 0.0
        else:
            hav_distance = 0.0
            hav_speed = 0.0
            hav_acceleration = 0.0

            pyproj_distance = 0.0
            pyproj_speed = 0.0
            pyproj_acceleration = 0.0

        print(f"{timestamp}, {current:.2f}, {lat:.7f}, {lon:.7f}, {alt}, "
```

```
            f"{hav_distance:.2f}, {hav_speed:.2f}, {hav_total_distance:.2f}, {
                hav_acceleration:.2f}, "
            f"{pyproj_distance:.2f}, {pyproj_speed:.2f}, {pyproj_total_distance:.2f},
                {pyproj_acceleration:.2f}")

        # Write to CSV
        with open(csv_file, mode='a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow([timestamp, current, lat, lon, alt,
                             hav_distance, hav_speed, hav_total_distance,
                                 hav_acceleration,
                             pyproj_distance, pyproj_speed, pyproj_total_distance,
                                 pyproj_acceleration])

        # Update previous coordinates, speed, and time
        previous_lat = lat
        previous_lon = lon
        previous_time = now
        hav_previous_speed = hav_speed

        pyproj_previous_lat = lat
        pyproj_previous_lon = lon
        pyproj_previous_alt = alt
        pyproj_previous_time = now
        pyproj_previous_speed = pyproj_speed
    else:
        print(f"{timestamp}, {current:.3f}, No data, No data, No data, "
              f"0.00, 0.00, {hav_total_distance:.2f}, 0.00, "
              f"0.00, 0.00, {pyproj_total_distance:.2f}, 0.00")

        # Write to CSV
        with open(csv_file, mode='a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow([timestamp, current, "No data", "No data", "No data",
                             0.00, 0.00, hav_total_distance, 0.00,
                             0.00, 0.00, pyproj_total_distance, 0.00])

    time.sleep(0.5)
```

Listing 4: Python code to measure current and system dynamics

Bibliography

[1] Juan Angel Gonzalez-Aguirre et al. "Service robots: Trends and technology". In: *Applied Sciences* 11.22 (2021), p. 10702.

[2] Eric Krotkov and John Blitch. "The defense advanced research projects agency (DARPA) tactical mobile robotics program". In: *The International Journal of Robotics Research* 18.7 (1999), pp. 769–776.

[3] Maria Kyrarini et al. "A survey of robots in healthcare". In: *Technologies* 9.1 (2021), p. 8.

[4] E Maria, E Budiman, M Taruk, et al. "Measure distance locating nearest public facilities using Haversine and Euclidean Methods". In: *Journal of Physics: Conference Series*. Vol. 1450. 1. IOP Publishing. 2020, p. 012080.

[5] NVIDIA. *Jetson Nano Developer Kit*. Accessed: 2024-09-05. 2024. URL: https://developer.nvidia.com/embedded/jetson-nano.

[6] Power UC. *PX4 Power Module PM07*. Accessed: 2024-09-05. 2024. URL: https://www.poweruc.pl/products/hall-split-core-current-sensor-hsts016l-rated-input-10-200a-rated-output-2-5-0-625v?srsltid=AfmBOoq-1oY1WujgxbOhWgWhLHfLqzVB82B32hCXNfUf_dBtcke0M6jB.

[7] PX4 Development Team. *Pixhawk 4 Flight Controller*. Accessed: 2024-09-05. 2024. URL: https://docs.px4.io/main/en/flight_controller/pixhawk4.html.

[8] PX4 Development Team. *PX4 Power Module PM07*. Accessed: 2024-09-05. 2024. URL: https://docs.px4.io/main/en/power_module/holybro_pm07_pixhawk4_power_module.html.

[9] M Shneier and R Bostelman. "Literature Review of Mobile Robots for Manufacturing, National Institute of Standards and Technology (US)". In: *Engineering Laboratory. Intelligent Systems Division* (2015).

[10] UNI-Trend. *UT203R Multimeter*. Accessed: 2024-09-05. 2024. URL: https://meters.uni-trend.com/product/ut200plus-series/#Docs.

[11] Chunjie Wang and Dandan Du. "Research on logistics autonomous mobile robot system". In: *2016 IEEE International Conference on Mechatronics and Automation*. IEEE. 2016, pp. 275–280.

[12] Mingyu Wu et al. "A review on energy efficiency in autonomous mobile robots". In: *Robotic Intelligence and Automation* 43.6 (2023), pp. 648–668.