Tesla Optimus Data Collection Documentation

Tesla Data Operations

June 2025

1 Overview

This script simulates real-time data collection for Tesla's Optimus robot, storing sensor data in MySQL, validating it for anomalies, and logging issues to Jira. It demonstrates skills in Python, MySQL, Linux, and Jira API integration, based on experience as a Data Collection Operator at Tesla.

2 Setup Instructions

2.1 MySQL Setup

- Create a MySQL database named `optimus_robot_data`.
- Execute the following SQL to create the required table:

```sql
CREATE TABLE sensor_data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    timestamp DATETIME,
    joint_angle FLOAT,
    torque FLOAT,
    temperature FLOAT,
    battery_level FLOAT
);
```

3 Python Script

The following Python script handles data collection, storage, validation, and issue reporting:

```python
import mysql.connector
import time
import random
import requests
import json
import logging
```

```
import os
from datetime import datetime

# Configure logging for debugging and tracking
logging.basicConfig(
    filename='optimus_data_collection.log',
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

# MySQL database connection configuration
db_config = {
    'host': 'localhost',
    'user': 'tesla_user',
    'password': 'secure_password',
    'database': 'optimus_robot_data'
}

# Jira API configuration
JIRA_URL = 'https://your-jira-instance.atlassian.net'
JIRA_API_TOKEN = 'your_api_token'
JIRA_USER = 'your_email@example.com'
JIRA_PROJECT_KEY = 'OPTIMUS'

# Simulate real-time sensor data collection from Optimus robot
def collect_sensor_data():
    """Simulate collecting real-time sensor data from Optimus robot."""
    return {
        'timestamp': datetime.now(),
        'joint_angle': round(random.uniform(0, 360), 2),  # Joint angle in degrees
        'torque': round(random.uniform(10, 100), 2),      # Torque in Nm
        'temperature': round(random.uniform(20, 80), 2),  # Temperature in Celsius
        'battery_level': round(random.uniform(0, 100), 2) # Battery percentage
    }

# Store sensor data in MySQL database
def store_data_in_mysql(data):
    """Store sensor data in MySQL database."""
    try:
        conn = mysql.connector.connect(**db_config)
        cursor = conn.cursor()

        insert_query = """
```

```
        INSERT INTO sensor_data (timestamp, joint_angle, torque, temperature, battery_level)
        VALUES (%s, %s, %s, %s, %s)
        """
        values = (
            data['timestamp'],
            data['joint_angle'],
            data['torque'],
            data['temperature'],
            data['battery_level']
        )

        cursor.execute(insert_query, values)
        conn.commit()
        logging.info("Data stored successfully: %s", data)

    except mysql.connector.Error as err:
        logging.error("MySQL Error: %s", err)
        raise
    finally:
        cursor.close()
        conn.close()

# Validate data and detect potential bugs
def validate_data(data):
    """Validate sensor data and flag potential issues."""
    issues = []

    if data['joint_angle'] > 350 or data['joint_angle'] < 10:
        issues.append(f"Joint angle out of range: {data['joint_angle']} degrees")
    if data['torque'] > 90:
        issues.append(f"Excessive torque detected: {data['torque']} Nm")
    if data['temperature'] > 70:
        issues.append(f"High temperature detected: {data['temperature']} C")
    if data['battery_level'] < 20:
        issues.append(f"Low battery level: {data['battery_level']}%")

    return issues

# Log issues to Jira
def log_issue_to_jira(issue_description):
    """Log detected issues to Jira as bug tickets."""
    headers = {
        'Authorization': f'Basic {JIRA_USER}:{JIRA_API_TOKEN}',
```

```
        'Content-Type': 'application/json'
    }

    issue_data = {
        'fields': {
            'project': {'key': JIRA_PROJECT_KEY},
            'summary': f'Optimus Robot Sensor Issue: {issue_description[:50]}',
            'description': issue_description,
            'issuetype': {'name': 'Bug'}
        }
    }

    try:
        response = requests.post(
            f'{JIRA_URL}/rest/api/3/issue',
            headers=headers,
            data=json.dumps(issue_data)
        )
        if response.status_code == 201:
            logging.info("Jira issue created: %s", response.json()['key'])
        else:
            logging.error("Failed to create Jira issue: %s", response.text)
    except requests.RequestException as e:
        logging.error("Jira API error: %s", e)

# Main data collection and processing loop
def main():
    """Main function to collect, store, validate, and report sensor data."""
    try:
        # Check for Linux environment
        if os.name != 'posix':
            logging.warning("This script is optimized for Linux/Unix environments.")

        while True:
            # Collect real-time sensor data
            sensor_data = collect_sensor_data()
            logging.info("Collected sensor data: %s", sensor_data)

            # Store data in MySQL
            store_data_in_mysql(sensor_data)

            # Validate data for potential issues
```

```
            issues = validate_data(sensor_data)
            if issues:
                for issue in issues:
                    logging.warning("Issue detected: %s", issue)
                    log_issue_to_jira(issue)

            # Simulate real-time data collection (every 5 seconds)
            time.sleep(5)

    except KeyboardInterrupt:
        logging.info("Data collection stopped by user.")
    except Exception as e:
        logging.error("Unexpected error: %s", e)

if __name__ == "__main__":
    main()
```