

# Data Parsing Implementation in Node.js for Arduino Sensor Data

## Overview

For this project, the goal was to capture sensor data from an Arduino device, transmit it via a serial connection, and store it in a MongoDB database using a Node.js application. To enhance the utility and accessibility of the stored data for future querying and analysis, it was essential to implement data parsing before data storage.

## Rationale

Raw data transmitted from Arduino typically includes various sensor readings in a non-structured format, which might be interspersed with metadata or other information. Storing such data directly into a database can lead to issues with data retrieval, increase the complexity of query operations, and potentially affect performance due to the unstructured nature of the data. By parsing the data before storage:

- **Structure is imposed**, making the database schema more consistent and predictable.
- **Data integrity is enhanced**, as only valid and correctly formatted data is stored.
- **Query performance is improved**, allowing for more efficient data analysis and retrieval operations.

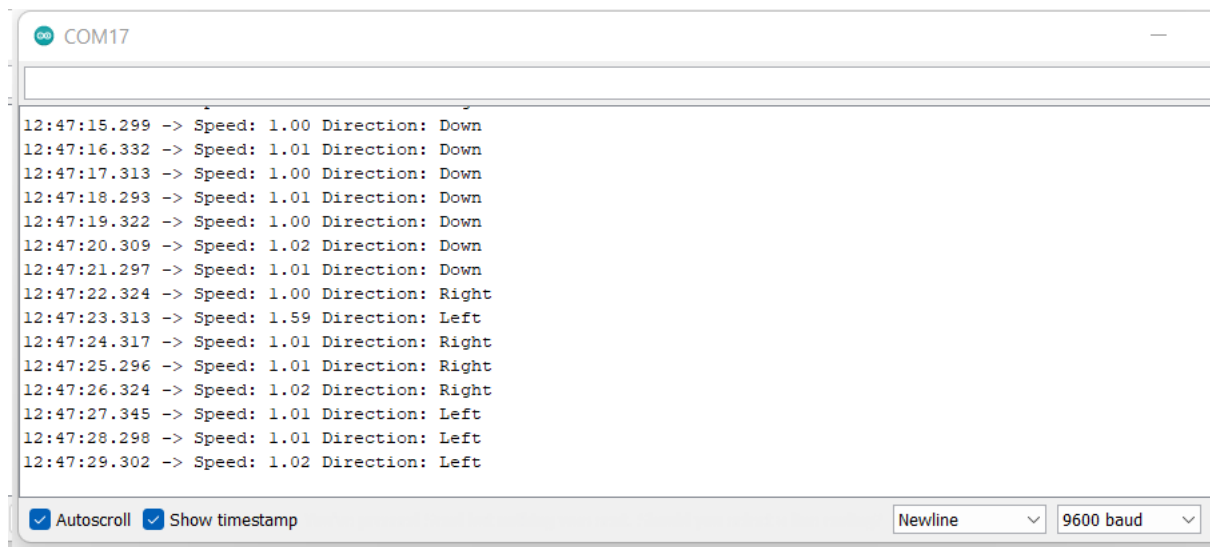
## Data Parsing Process

### Step 1: Data Reception

The Node.js application listens to the serial port for data transmitted by the Arduino. Each piece of data received is a string that potentially contains multiple sensor readings formatted in a predefined pattern.

Example of raw data received:

```
18:45:19.234 -> Speed: 1.01 Direction: Down
```



```
COM17

12:47:15.299 -> Speed: 1.00 Direction: Down
12:47:16.332 -> Speed: 1.01 Direction: Down
12:47:17.313 -> Speed: 1.00 Direction: Down
12:47:18.293 -> Speed: 1.01 Direction: Down
12:47:19.322 -> Speed: 1.00 Direction: Down
12:47:20.309 -> Speed: 1.02 Direction: Down
12:47:21.297 -> Speed: 1.01 Direction: Down
12:47:22.324 -> Speed: 1.00 Direction: Right
12:47:23.313 -> Speed: 1.59 Direction: Left
12:47:24.317 -> Speed: 1.01 Direction: Right
12:47:25.296 -> Speed: 1.01 Direction: Right
12:47:26.324 -> Speed: 1.02 Direction: Right
12:47:27.345 -> Speed: 1.01 Direction: Left
12:47:28.298 -> Speed: 1.01 Direction: Left
12:47:29.302 -> Speed: 1.02 Direction: Left

☒ Autoscroll ☒ Show timestamp Newline 9600 baud
```

## Step 2: Parsing Logic

The data parsing function extracts relevant pieces of information from the raw data strings using regular expressions. This method ensures that only the necessary data (e.g., speed and direction) is extracted, ignoring any irrelevant metadata or formatting characters.

### Parsing Function Example:

```
function parseData(data) {
  const speedMatch = data.match(/Speed: ([\d.]+)/);
  const directionMatch = data.match(/Direction: (\w+)/);
  if (speedMatch && directionMatch) {
    return {
      speed: parseFloat(speedMatch[1]),
      direction: directionMatch[1],
      timestamp: new Date() // Adding a timestamp for each data entry
    };
  }
  return null; // Return null if data does not match expected pattern
}
```

## Benefits

Implementing data parsing within this Node.js application provided several key benefits:

- **Improved Data Management:** Structured data storage makes it easier to manage, update, and delete data entries.
- **Enhanced Analytical Capabilities:** Well-structured data supports more complex queries and analytical operations, such as aggregations and statistical analysis.
- **Scalability:** As the system scales and handles more data sources or varied data formats, maintaining structured data input becomes crucial for system performance and reliability.

```
_id: new ObjectId('662b55cb4af3b2fcd6d00e1e'),
timestamp: 2024-04-26T07:20:43.577Z,
__v: 0
}
Received data: Speed: 1.04 Direction: Stationary
Data saved to MongoDB: {
  speed: 1.04,
  direction: 'Stationary',
  _id: new ObjectId('662b55cc4af3b2fcd6d00e20'),
  timestamp: 2024-04-26T07:20:44.581Z,
  __v: 0
}
Received data: Speed: 0.95 Direction: Left
Data saved to MongoDB: {
  speed: 0.95,
  direction: 'Left',
  _id: new ObjectId('662b55cd4af3b2fcd6d00e22'),
  timestamp: 2024-04-26T07:20:45.585Z,
  __v: 0
}
Received data: Speed: 0.97 Direction: Left
Data saved to MongoDB: {
  speed: 0.97,
  direction: 'Left',
  _id: new ObjectId('662b55ce4af3b2fcd6d00e24'),
  timestamp: 2024-04-26T07:20:46.589Z,
  __v: 0
}
```

Filter

Type a query: { field: 'value' }

QUERY RESULTS: 1-20 OF MANY

```
_id: ObjectId('662a5da699d8ce1f6d88020c')
speed : 1.01
direction : "Down"
timestamp : 2024-04-25T13:41:58.952+00:00
__v : 0
```

## Conclusion

Data parsing plays a critical role in the efficient and effective storage and management of sensor data in IoT applications. By structuring data before it enters the database, the system ensures that the data is ready for immediate use in applications and analyses, thus enhancing the overall utility of the data collection system.