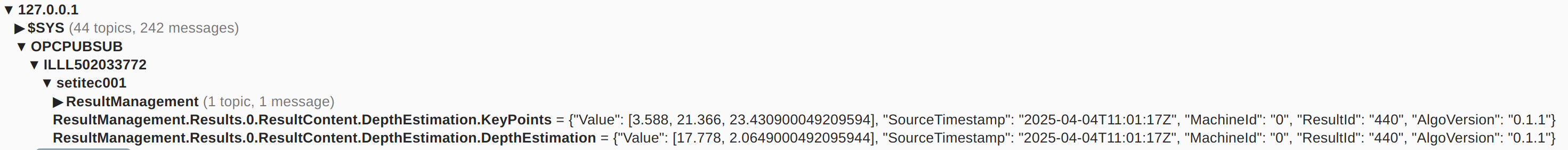
Deep Learning Drilling Depth Estimation System Documentation

This documentation provides a guide to the Deep Learning Drilling Depth Estimation System, which processes drilling data from MQTT messages to perform depth estimation.

Changes in 0.2.2

1. Addressed a bug that led to the subscriber always dropping the first MQTT message received. This is done by ensuring that the main function calls the listening thread and the analysis thread independently.

Changes in 0.2.0

1. Replaced the two-point algorithm with the three-point one, providing entry, transition, and exit location estimates. 

Changes in 0.1.1

1. Switch to wheels instead of tarballs to further reduce installation size and time
2. Ability to run through an executable in addition to the Python script
3. Addressed primary and secondary bugs due to repurposing of topic names
4. Added more contextual information: the published message now contains the depth estimation results along with the source timestamp, the ResultID, the Tool serial number and the algorithm version
5. Updated the Dockerfile; now it works (tested on Sheffield machine)

# Overview

Currently, the system consists of: 1. A Python script (listen-continuous.py) and an executable (uos\_dephest\_listener) that listens for MQTT messages containing drilling data 6. A configuration file (mqtt\_conf.yaml) for customizing connection settings and data mappings

The system listens for two types of MQTT messages:

* Result messages: Contains metadata and operation results
* Trace messages: Contains detailed trace data (position, torque, thrust) for drilling operations

When matching result and trace messages are received, the system processes them to estimate drilling depth and publishes the results back to the MQTT broker. At the moment the mapping is done by reusing the relevant sections of the MQTT topic, defined in the configuration file.

# Installation

## Prerequisites

* Python 3.9+
* The abyss package
* Key dependencies, namely transformers, tsfm\_public, and accelerate
* Optional: for local testing, an MQTT broker such as mosquitto should already be installed
* Optional: Docker for virtual machine installation

## Steps for virtual machine installation (using Docker)

1. Clone the entire repository
2. Build the Docker images for publisher and listener
3. Run the Docker images

Details on step #1

git clone https://github.com/whutabarat-sheffield/uos-drilling-wh.git .

Details on step #2 and #3

# step #2 build the listener  
docker build -t listener .

# step #3 run the listener  
docker run -t listener

The listener is callable from the command line as

uos\_depthest\_listener --config (path to the configuration file)

The docker build will already setup a workspace under /app directory, which contains the configuration file mqtt\_conf.yaml.

## Steps for direct installation using sources

Clone the repository

Change into abyss/

Type ‘pip install .’

Go into abyss/src/tests and run it

## Steps for direct installation using wheels

1. Download all of the files in the release section apart from the source code files.
2. Place them into a single directory, such as dist. The exceptions are listen-continuous.py and mqtt\_conf.yaml, which should be placed in a project directory such as run. For example:

[C:\Users\windo\Downloads\image.png](file:///C:\Users\windo\Downloads\image.png)

When running with the new uos\_dephest\_listener, only mqtt\_conf.yaml needs to be copied into the directory.

1. Install the required dependencies as well as the algorithm from the provided wheels. This is easily done by a single command:

cd dist  
pip install -r requirements.txt

This process can take multiple minutes as the installation will download additional dependencies that are not yet available within the Python environment.

On an Airbus machine:

1. To test with the python script, go to the run directory that contains both listen-continuous.py and mqtt\_conf.yaml, and run it:

python listen-continuous.py

To test with the executable, go to the run directory that contains mqtt\_conf.yaml, and run the executable:

uos\_dephest\_listener &

# Configuration

The mqtt\_conf.yaml file contains all necessary configuration for the MQTT connection and data processing. It is fully commented. It is recommended to create copies and name them according to the specific configuration. More details are provided below under [Usage](#usage).

## Key Configuration Sections

1. **Broker Settings**: Connection details for the MQTT broker; the example below shows the host definition that is needed for testing on Docker images

* broker:  
   host: host.docker.internal  
   port: 1883  
   username: "" # Optional  
   password: "" # Optional

1. **Topic Structure**: Defines the MQTT topics to listen for messages. This was based on the design shared in December 2024.

* listener:  
   root: "OPCPUBSUB"  
   toolboxid: "+" # "+" is a wildcard  
   toolid: "+"  
   result: "ResultManagement"  
   trace: "ResultManagement/Trace"

1. **Data Field Mappings**: Paths to extract specific data from messages. This was based on the design shared in December 2024.

* data\_ids:  
   machine\_id: 'ResultManagement.Results.0.ResultMetaData.SerialNumber'  
   position: 'ResultManagement.Results.0.ResultContent.Trace.StepTraces.PositionTrace.StepTraceContent[0].Values'  
   # additional mappings...

1. **Estimation Output**: Topics for publishing depth estimation results. This can be changed to suit.

* estimation:  
   keypoints: 'ResultManagement.Results.0.ResultContent.DepthEstimation.KeyPoints'  
   depth\_estimation: 'ResultManagement.Results.0.ResultContent.DepthEstimation.DepthEstimation'

# Usage

## Basic Usage

Run the script or the executable with default configuration:

python listen-continuous.py

uos\_dephest\_listener &

## Command-line Arguments

The script supports the following command-line arguments:

* --config: Path to YAML configuration file (default: mqtt\_conf.yaml)
* --log-level: Logging level (choices: DEBUG, INFO, WARNING, ERROR, CRITICAL; default: INFO)

Example:

python listen-continuous.py --config=custom\_config.yaml --log-level=DEBUG

## Logging

The system uses Python’s built-in logging module with configurable levels: - DEBUG: Detailed information, typically for debugging. Reconstructed JSON files are written to disk. - INFO: Confirmation that things are working as expected. Many tracking messages are printed on screen. - WARNING: Something unexpected happened, but the system is still working - ERROR: Serious problems that prevent some functionality - CRITICAL: Critical errors that prevent the system from working

## How It Works

1. **Message Listening**: The system creates two MQTT clients to listen for result (e.g. machine id, cycle number) and trace (e.g. position, torque and thrust) messages.
2. **Message Buffering**: Messages are buffered and matched based on:
   * Same toolbox ID and tool ID
   * Timestamps within the configured time window (default: 1 second)
3. **Processing**: When matching messages are found:
   * Data is extracted according to the configured mappings
   * Matched pairs are saved to file for reference (in DEBUG mode only)
   * The depth inference model processes the data and hands it over to the publisher
4. **Results Publishing**: Depth estimation results are published back to the MQTT broker
5. **Cleanup**: Old messages are periodically removed from the buffer

## Example Use Cases

### Industrial Drilling Monitoring

Monitor drilling operations in real-time by analyzing depth data:

python listen-continuous.py --log-level=INFO

The system will: 1. Connect to the configured MQTT broker 2. Listen for drilling data messages 3. Process and analyse depth information 6. Publish depth estimation results 7. Save matched data pairs for further analysis

### Debugging Data Issues

To troubleshoot data processing problems:

python listen-continuous.py --log-level=DEBUG

This will provide detailed logs about: - Message receiving and parsing - Buffer management - Matching process - Data processing steps

### Custom Configuration

For different environments or data structures:

1. Create a custom configuration file (e.g., production\_conf.yaml)
2. Update broker settings, topic structure, and data mappings
3. Run with the custom configuration:

* python listen-continuous.py --config=production\_conf.yaml

# Key Components

## DrillingDataAnalyser Class

The main class that: - Connects to the MQTT broker - Listens for messages - Buffers and matches messages - Processes matched pairs - Publishes depth estimation results

## TimestampedData Class

A data structure that stores: - Unix timestamp of the message - Message data - Source topic

## Main Functions

* find\_and\_process\_matches(): Matches result and trace messages
* process\_matching\_messages(): Processes matched messages to estimate depth
* depth\_est\_ml\_mqtt(): Performs depth estimation using the ML model

## Customization

To adapt the system to your specific needs:

1. Update the mqtt\_conf.yaml file with your specific:
   * MQTT broker details
   * Topic structure
   * Data field mappings
2. Modify the depth inference processing in the process\_matching\_messages() method
3. Adjust the time window and cleanup intervals in the DrillingDataAnalyser class

## Troubleshooting

Common issues and solutions:

1. **Connection Problems**:
   * Verify broker host and port in config file
   * Check network connectivity
   * Ensure credentials are correct if authentication is required
   * For testing in linux localhost, update the broker host to ‘localhost’
   * Ensure that a broker such as mosquitto is installed. Testing in Windows without a broker present creates this issue:

image

1. **No Messages Being Processed**:
   * Verify topic structure in configuration matches actual messages
   * Check log for subscription confirmations
   * Use --log-level=DEBUG for detailed connection information
2. **Data Extraction Issues**:
   * Ensure data\_ids paths in the config file match actual message structure. If the names of the topics have changed, then the appropriate keys needs to be updated

* # Data Field Mappings  
   # -----------------  
   # These paths define where to find specific data fields in the message payload  
   # TODO: Update these paths based on the actual data structure  
   data\_ids:  
   # Machine identification  
   machine\_id: 'ResultManagement.Results.0.ResultMetaData.SerialNumber'  
    
   # Trace data paths  
   position: 'ResultManagement.Results.0.ResultContent.Trace.StepTraces.PositionTrace.StepTraceContent[0].Values'  
   thrust: 'ResultManagement.Results.0.ResultContent.Trace.StepTraces.IntensityThrustTrace.StepTraceContent[0].Values'  
   torque: 'ResultManagement.Results.0.ResultContent.Trace.StepTraces.IntensityTorqueTrace.StepTraceContent[0].Values'  
   step: 'ResultManagement.Results.0.ResultContent.Trace.StepTraces.StepNumberTrace.StepTraceContent[0].Values'  
    
   # Empty value references  
   torque\_empty\_vals: 'ResultManagement.Results.0.ResultContent.StepResults.0.StepResultValues.IntensityTorqueEmpty'  
   thrust\_empty\_vals: 'ResultManagement.Results.0.ResultContent.StepResults.0.StepResultValues.IntensityThrustEmpty'  
   step\_vals: 'ResultManagement.Results.0.ResultContent.StepResults.0.StepResultValues.StepNumber'
  + Save and examine raw messages using DEBUG mode
  + Verify the format of timestamp fields, which should follow this format: "%Y-%m-%dT%H:%M:%SZ"

1. **Depth Estimation Failures**:
   * Ensure that the PSTEP in the xls file looks exactly as the screengrab below: image
   * As shown in the screengrab, the current model is trained on a specific drilling configuration that has two different types of program steps.
     + If the data only have one program step type, the algorithm fails.
     + If the data have three or more program step types, the depth prediction will be meaningless.
   * Check the specific error messages in the logs
   * Verify the depth inference model is properly initialised, with messages such as below:

* INFO:p-3643:t-140230986614592:config.py:<module>:PyTorch version 2.3.1+cpu available.  
  INFO:p-3643:t-140230986614592:listen-continuous.py:\_\_init\_\_:Loaded configuration from mqtt\_conf.yaml  
  /home/windo/mambaforge/lib/python3.10/site-packages/abyss/trained\_model/has\_tool\_age\_predrilling/cv4

## What’s Changed

* Deploy by @whutabarat-sheffield in https://github.com/whutabarat-sheffield/uos-drilling-wh/pull/2
* Vscode docker by @whutabarat-sheffield in https://github.com/whutabarat-sheffield/uos-drilling-wh/pull/3

**Full Changelog**: https://github.com/whutabarat-sheffield/uos-drilling-wh/compare/v0.1.1-beta…v0.2.0-beta