Laboratory Manual: Signal Temporal Logic for Digital Twin Monitoring

BIP MBSE4DT

1 Introduction to Digital Twin Monitoring

Digital twins are virtual representations of physical systems that enable real-time monitoring, analysis, and optimization. A critical aspect of digital twins is their ability to monitor system behaviour and verify that the physical system operates within specified boundaries and meets performance requirements.

We will investigate some of the crane's properties and how to implement them in STL. We will work with an optimal path calculated by an optimal control algorithm and the path done by the gantry system. The crane's motion must be controlled to minimize swing while ensuring accurate positioning. Our digital twin will monitor two key aspects:

- Position (x) along the rail
- Angular position (θ) of the suspended load

2 Signal Temporal Logic (STL)

Signal Temporal Logic is a formal specification language that allows us to express and verify properties of continuous-time signals. Key operators in STL include:

- **G** (Globally): Property must hold for all time points
- **F** (Eventually): Property must hold at some future time point
- **U** (Until): The first property must hold until second property becomes true

Temporal operators can include timing bounds. For example:

- $\mathbf{G}_{[0,5]}(x \leq 0.4)$: Position must stay below 0.4m for the first 5 seconds
- $\mathbf{F}_{[0,2]}(|\theta| \leq 0.003)$: Swing angle must become less than 0.003 rad within 2 seconds

2.1 Python Implementation

We use the `py-metric-temporal-logic` library, which requires a two-step approach, as the MTL parser can only handle statements combined with the logic operators defined in the manual of py-metric-temporal-logic, and not the

- To allow for checks of complex statements that involve data, we must convert the numerical signals into boolean signals using lambda functions. Lambda functions are small anonymous functions.

```
# Example: Convert position measurements to boolean signal to
check if the position is within bounds (< 0.4)
x_within_bounds = lambda lst: [
    (t, x <= 0.4) for (t, x) in lst
]
```

- Then, we can write STL formulas that operate on these boolean signals:

```
# The MTL formula refers to the boolean signal names
formula = mtl.parse('G(x_bound)')  # Always within bounds

# To evaluate, combine signals in a dictionary
data = {
    'x_bound': x_within_bounds(measurements)
}
result = formula(data, quantitative=False)
```

This separation is crucial because:

- The MTL parser cannot handle numerical comparisons directly
- Lambda functions pre-process the numerical signals into boolean signals
- The MTL formula then operates on these pre-processed boolean signals

A complete example is found in the exercise.

3 System Architecture

3.1 Database Structure

The system stores two types of time series data:

- **Trajectory**: The ideal, planned motion generated by the controller
- **Measurements**: Actual sensor readings from the physical system

The database connection provides methods to retrieve both:

```
dbc = DatabaseConnection()
trajectory = dbc.get_trajectory()  # Planned motion
measurements = dbc.get_measurement()  # Actual motion
```

Data is returned in a dictionary. The dictionary contains the following keys: 'position', 'angular position'. The value of each key in the dictionary is a list of time-value pairs: `[(t1, val1), (t2, val2), ...]`

4 Laboratory Exercises

The exercises progress from basic safety monitoring to complex dynamic behaviors. Each part builds upon the previous ones, gradually introducing more sophisticated STL properties.

4.1 Part 1: Basic Safety Bounds

Implement basic position bounds checking. This introduces the fundamental concept of converting numerical signals to boolean signals and writing simple STL formulas.

4.2 Part 2: Position-Based Properties

Monitor if the crane reaches its target positions within specified tolerances. This exercise introduces the concept of parameterized properties based on trajectory planning.

4.3 Part 3: Complex Dynamic Behavior

Implement swing detection and settling time requirements. This combines multiple signals and introduces nested temporal operators to specify complex timing requirements.

4.4 Part 4: Custom Property

Design and implement an additional monitoring property. This will test your understanding of both STL and the system's physical constraints.