Pure Gazebo Simulation for Multirotor Slung Load System

Mohssen Elshaar

November 2024

Contents

1. Background

The first step involves comprehending the MATLAB-based simulation of the quasi-static feedback (QSF) linearization controller. This controller simplifies the complex nonlinear dynamics of the slung load system (SLS) by linearizing the outer-loop dynamics, yielding linear time-invariant (LTI) error dynamics that facilitate stability proofs and gain tuning. The MATLAB simulation, available at the GitHub repository[1], models the SLS as a rigid pendulum suspended from a quadrotor UAV's center of mass. The configuration dynamics are expressed as:

$$p_L = p_Q + Lq,$$

[1] `https://github.com/ANCL/QuasiSLSExp/tree/main/MatlabMapleCode/Matlab`

where p_L is the pendulum's position, p_Q is the UAV's center of mass position, L is the pendulum length, and q is the unit vector representing pendulum orientation in the inertial frame. The translational and rotational dynamics, including compensations for parasitic rotor drag forces, are detailed in equations provided in [1], making the simulation an essential reference for step-wise validation.

The second step delves into the PX4-based Software-In-The-Loop (SITL) simulation, which integrates the QSF controller with the PX4 autopilot and the Gazebo simulator. Unlike MATLAB simulations, this framework allows for hardware-in-the-loop testing and model validation under realistic operating conditions. The customized simulation package, accessible at the GitHub repository[2], features rotor drag modeling and controller auto-generation from MATLAB code. The SLS system is represented using a physics-based multi-body dynamics approach in Gazebo, while the QSF controller outputs thrust and torque commands for attitude stabilization. Figure 1 shows the PX4-based SITL work flow.

The third step, which is the core part of this work, is the implementation of a pure-Gazebo simulation of the SLS, eliminating dependencies on PX4 or Rotors plugins. This approach models the UAV as a point mass while the QSF controller generates 3D force vectors to replicate the system dynamics. By leveraging Gazebo's capability to apply forces and torques, the simulation will align closely with the MATLAB-based ODE framework, allowing validation of nonlinear control strategies in a modular, accessible, and scalable simulation environment. Figure 2 shows the pure Gazebo-based work flow.

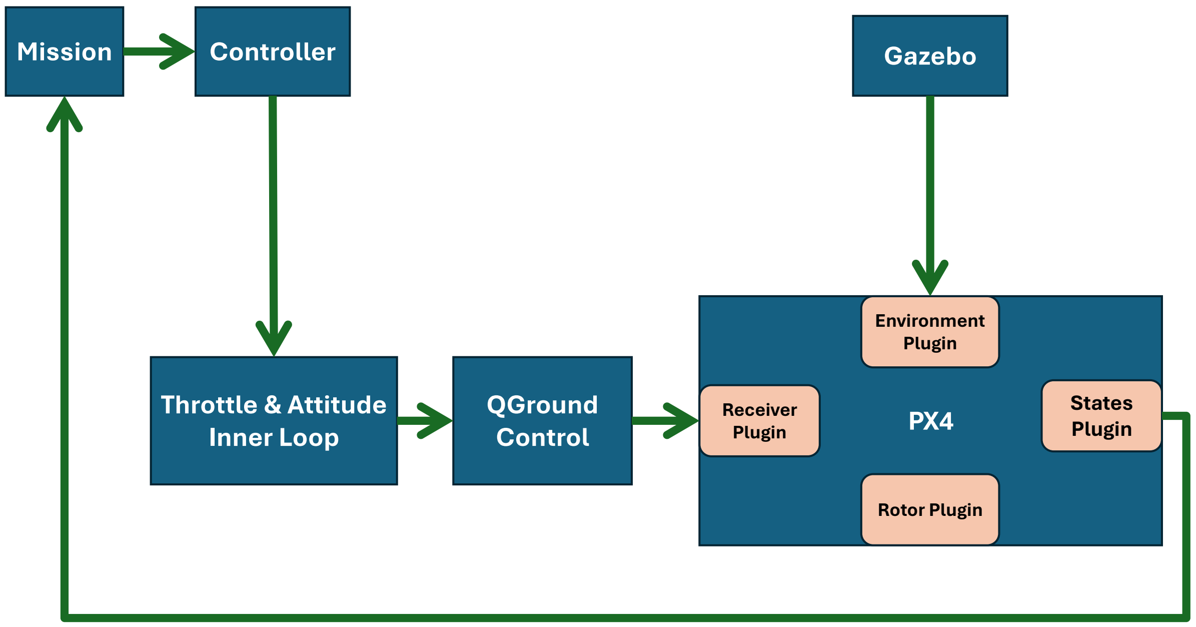


Figure 1: PX4-based SITL Work Flow

2. Pure Gazebo Implementation

This section describes the design and implementation of the Slung Load System (SLS) model in Gazebo, along with the stabilization and tracking controller developed as a Gazebo plugin. The system models

[2]https://github.com/ANCL/ACC-Repository/tree/master/Tools/sitl_gazebo/ancl_sls

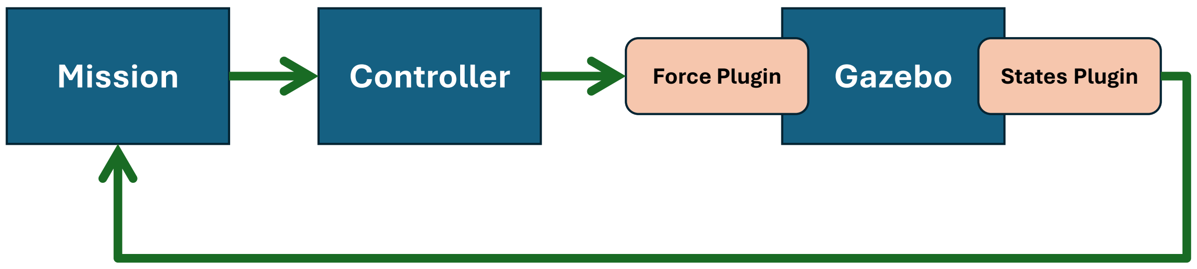


Figure 2: Pure Gazebo Work Flow

dynamic behavior and applies control forces to achieve desired stability and trajectory tracking. Key functionalities include:

- Logging simulation data,
- Computing pendulum dynamics,
- Implementing control algorithms.

2.1. Slung Load System Model

The SLS model is constructed using files from the directory `iris_pendulum/`. The main quadcopter geometry is defined in `iris.sdf`, alongside associated mesh files. The quadcopter dynamics' inner control loop is not included, as it is beyond the scope of this study. The pendulum model is sourced from the repository file: iris_pendulum.sdf

2.1.1. Model Components and Configuration

The `iris_pendulum` model simulates a quadcopter with a suspended pendulum and load, comprising three components:

1. **Base Link**: Represents the quadcopter body with the following properties:
   - Initial Pose: (0, 0, 0, 0, 0, 0).
   - Inertial Properties:
     - Mass: 1.5 kg,
     - Inertia Tensor: $I_{xx} = I_{yy} = 0.029125$, $I_{zz} = 0.055225$ (kg·m^2).
   - Collision Geometry: Box with dimensions `0.47 x 0.47 x 0.11 m`.
   - Visual Geometry: References `iris.stl` and uses material `Gazebo/DarkGrey`.
2. **Pendulum**: Extends downward with the following properties:
   - Initial Pose: (0, 0, -0.5) m.
   - Inertial Properties:
     - Mass: 0.001 kg,

    - Inertia Tensor: $I_{xx} = I_{yy} = 0.0008333$, $I_{zz} = 0.00005$ (kg·m^2).
  - Collision Geometry: Cylinder with radius 0.01 m and length 1.0 m.
  - Visual Geometry: Matches the collision shape and uses material `Gazebo/White`.

3. **Load**: Positioned at (0, 0, -1.1) m with the following properties:
   - Inertial Properties:
     - Mass: 0.2 kg,
     - Inertia Tensor: $I_{xx} = I_{yy} = I_{zz} = 0.0008$ (kg·m^2).
   - Collision Geometry: Sphere with radius 0.1 m.
   - Visual Geometry: Matches the collision shape and uses material `Gazebo/Red`.

2.1.2. Connections and Joints

The components are connected using universal joints:

- `pendulum_joint`: Connects the base link and pendulum.
  - Parent: Base Link,
  - Child: Pendulum,
  - Axes: Primary along x-axis, Secondary along y-axis.
- `load_joint`: Connects the pendulum and load.
  - Parent: Pendulum,
  - Child: Load.

The model incorporates a plugin named `MissionPlugin` (`libMission_plugin.so`), which handles simulation-specific tasks, discussed in the next section.

2.2. Mission Plugin Implementation

The `MissionPlugin` is implemented in C++ and extends `ModelPlugin`. It leverages Gazebo's API for real-time simulation and logging. Key components include:

1. **Initialization:** Loading model links, joints, and parameters.
2. **Logging System:** Dual-format logging for textual and CSV data.
3. **Control Logic:** Real-time computation of control forces.
4. **State Updates:** Continuous monitoring of system dynamics.

2.2.1. Control Strategies

The plugin utilizes two control C++ libraries found at: `StabController` & `TracController`.

- **Stabilization Controller:** Maintains system stability using corrective forces based on angular and positional deviations.
- **Tracking Controller:** Guides the quadrotor along predefined trajectories.

2.2.2. Main Functions

`Load()` **Function** This function is responsible for initializing the plugin. It performs the following tasks:

- Loads the quadrotor and pendulum components.
- Sets up logging systems for recording simulation data.
- Connects to Gazebo's update event for periodic execution of the simulation steps.

`OnUpdate()` **Function** This function is executed at each simulation step and performs the following operations:

- Retrieves and processes the state of the system.
- Computes the pendulum angles β and α, as well as their angular derivatives, γ_β and γ_α, using the following formulas:
$$\beta = \arcsin\left(\frac{L_x}{L}\right), \quad \alpha = \arcsin\left(-\frac{L_y}{L\cos\beta}\right),$$
$$\gamma_\beta = \frac{\dot{p}_L^x - \dot{p}_Q^x}{\cos\beta}, \quad \gamma_\alpha = \frac{-\dot{p}_L^y + \dot{p}_Q^y - \sin\alpha\sin\beta\gamma_\beta}{-\cos\alpha\cos\beta},$$
where:
  - L_x, L_y: Components of the pendulum's position relative to the quadrotor.
  - L: Length of the pendulum.
  - $\dot{p}_L^x, \dot{p}_L^y$: Components of the load's velocity.
  - $\dot{p}_Q^x, \dot{p}_Q^y$: Components of the quadrotor's velocity.
- Applies control algorithms to compute the necessary forces for stabilization.
- Directly applies the computed forces to the quadrotor body, treated as a point mass, using the `AddForce(_)` function.
- Logs the system state and control forces for analysis.

`IntegralStabController()` **Function** This function implements an integral stabilization controller with bounded error integration. It enhances the functionality of the `StabController()` by adding an integral component to improve steady-state error correction. However, it is important to note that this controller was not utilized in generating the reported results.

2.2.3. Logging Mechanism

Simulation data is logged in two formats:

- **Textual Log:** Summarizes state information for human readability.
- **CSV Log:** Provides detailed machine-readable data for analysis.

Logged data includes time, positions, orientations, velocities, pendulum angles, and control forces.

2.3. Code Repository

The full implementation can be accessed at:

`https://github.com/Hosnooo/STIL-Gazebo-Iris-Pendulum`.

3. Simulations & Results

The performance of the implemented pure-Gazebo simulation was evaluated using both set-point stabilization and trajectory tracking scenarios. The results were compared against the PX4 and MATLAB (ODE)-based simulations to validate the proposed approach.

3.1. Data Collection

To the best of my knowledge, the PX4 implementation lacked a mechanism for logging mission data. To overcome this, a simple CSV logging block was added to the `off_board_holy_node.cpp` file. This modification ensures that essential mission parameters, such as system states and control inputs, are systematically recorded, enabling comprehensive post-mission analysis and validation of the control strategies.

Important Notes

- The Gazebo simulation time is independent of the machine's clock, whereas ROS time, used in the PX4 implementation, operates based on the machine's clock by default. To synchronize mission trajectories and ensure consistent tests across both simulations, ROS time must be set to the Gazebo simulation time. This synchronization can be achieved by adding the following line to the PX4 launch file:

  ```
  <param name="use_sim_time" value="true"/>
  ```

- The ROS messaging frequency differs from the simple Gazebo logging frequency. It is recommended to control the logging samples in the Gazebo simulation or manage separate time vectors during data analysis to ensure alignment and accurate comparison.

- In the Gazebo simulation, the Z and Y axes are defined in the negative direction relative to the model used for controller design. Therefore, it is crucial to adjust the signs of command points and states obtained from Gazebo accordingly when passing them to the controller.

3.2. Set-Point Stabilization

The set points $(1, -1, -2)$, $(-1, -1, -4)$, and $(1, 1, -3)$ were tested independently, allowing sufficient time for the steady state to prevail in each case.

Figures 3a and 3b demonstrate the system's response for stabilization at multiple set-points. The MATLAB ODE-based simulation showed a perfect fit over the desired states, with no overshoot. The Gazebo-based simulation successfully converged to the desired set-points, as shown by the positional errors in Figure 3b. The QSF controller in the Gazebo simulation outperformed the PX4 simulation, achieving faster convergence and reduced oscillations in all axes. This is intuitive, as the complexities of the attitude dynamics no longer influence the control forces applied; the forces are applied exactly as desired.

3.3. Trajectory Tracking

For trajectory tracking, the system was tested on a demanding figure-eight trajectory (Figure 5). The reference trajectory is parameterized as follows:

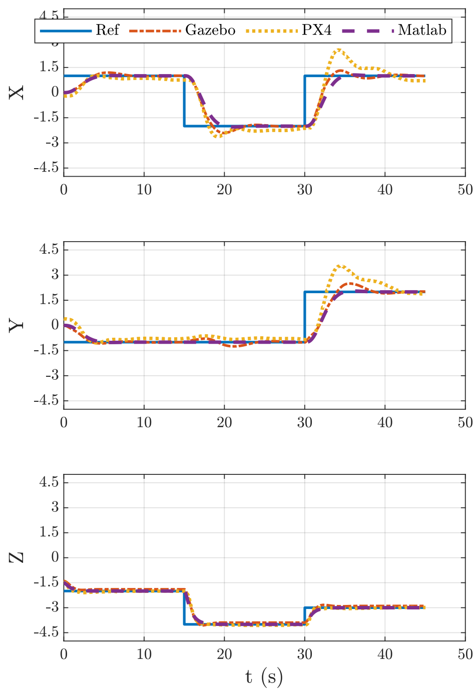
$$\mathbf{p}_r(t) = \begin{bmatrix} p_{1r}(t) \\ p_{2r}(t) \\ p_{3r}(t) \end{bmatrix} = \begin{bmatrix} 1.5\sin\left(\frac{2\pi t}{T}\right) \\ 0.75\sin\left(\frac{4\pi t}{T}\right) \\ -0.5 + 0.5\sin\left(\frac{2\pi t}{T}\right) \end{bmatrix},$$

where $T = 16$ is the period, and $t \in [0, 2T]$ represents the time vector.

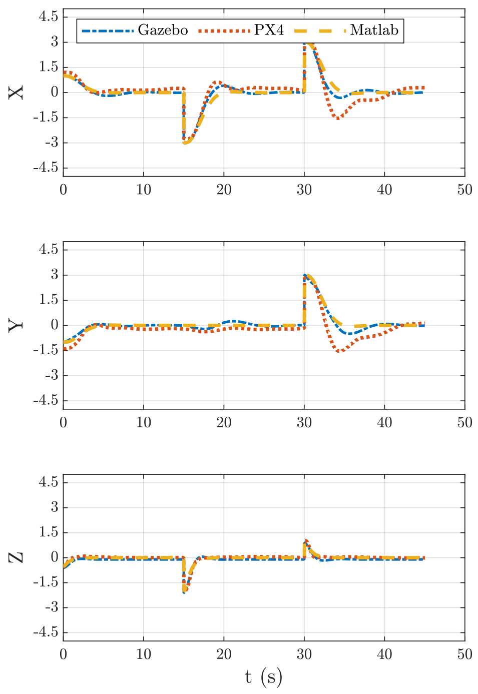
Figures 4a and 4b highlight the system's performance. The MATLAB ODE-based simulation precisely tracks the desired states without any overshoot. The Gazebo simulation closely follows the reference trajectory, as evidenced by the consistently low positional error across all axes in Figure 4b. In comparison, the Gazebo simulation outperforms the PX4-based approach, demonstrating better control performance.

3.4. Key Observations

The MATLAB ODE-based simulation consistently demonstrated superior performance compared to both the Gazebo-based and PX4 simulations. Gazebo showed better results than PX4, primarily due to its modular design and the direct force application method, which closely aligns with the theoretical MATLAB ODE framework. Furthermore, Gazebo's reduced reliance on specific hardware or plugins enhances its accessibility and scalability, making it a more versatile simulation platform.

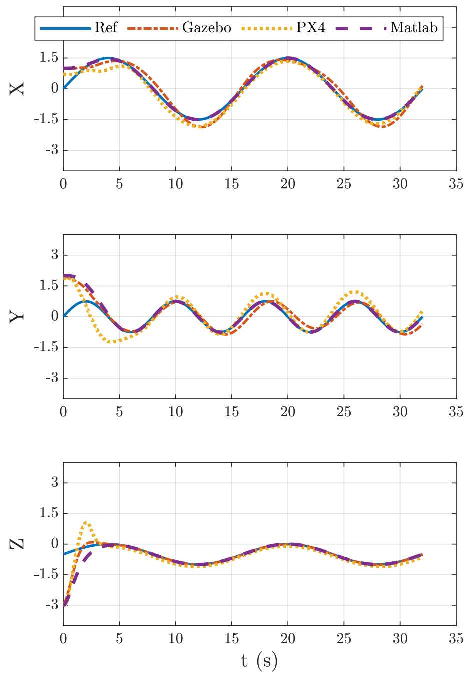


(a) 3 set-points trajectory

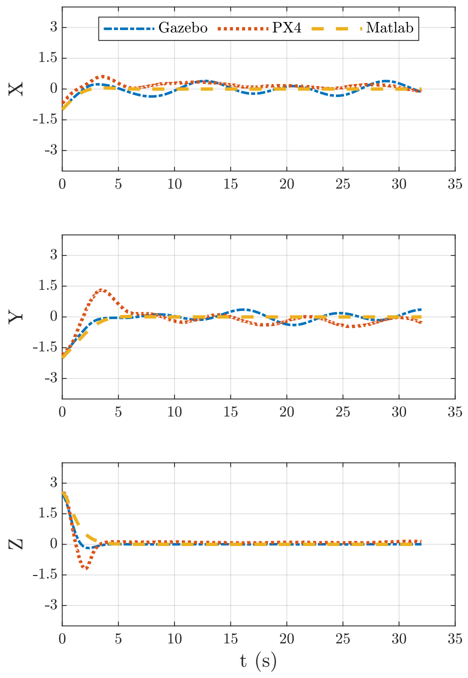


(b) Positional errors

Figure 3: 3 set-point stabilization results



(a) Figure-eight trajectory



(b) Positional errors

Figure 4: Figure-eight trajectory tracking results

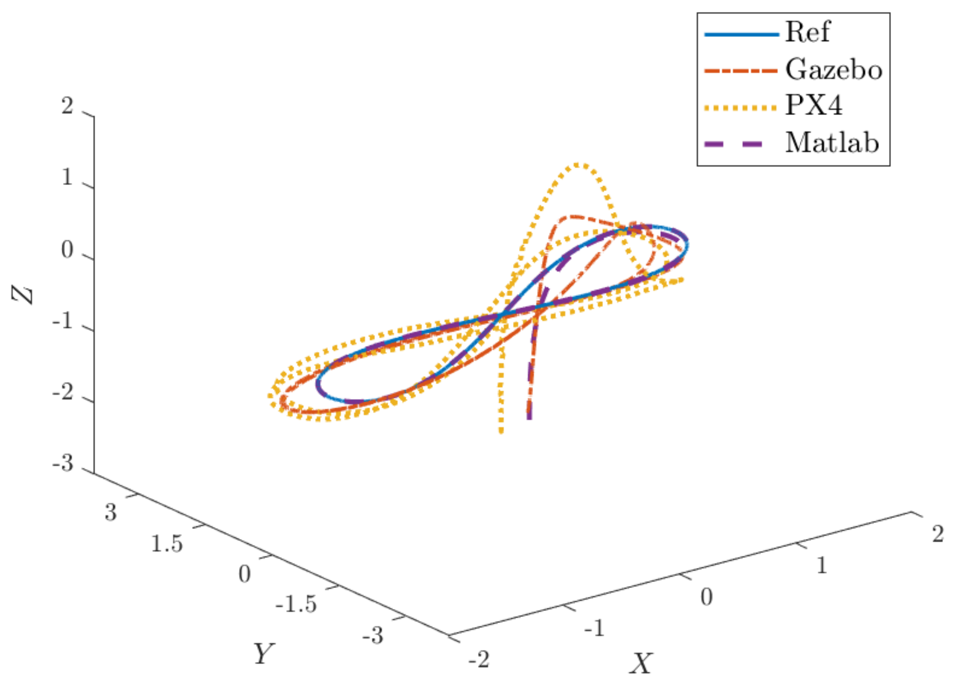


Figure 5: Figure-eight 3-D View

References

[1] Z. Jiang, Y. Yu, and A. Lynch, “Nonlinear motion control of a multirotor slung load system: Experimental results,” in *2024 American Control Conference (ACC)*, pp. 2851–2857, 2024.